Fakultät Informatik/Mathematik
Lehrstuhl für Softwaretechnologie

HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Vorlesungsmitschrift

SOFTWARE ENGINEERING 1

Mitschrift von
Falk-Jonatan Strube
Vorlesung von
Prof. Dr. Anna Sabine Hauptmann

27. März 2018

INHALTSVERZEICHNIS

VORBEMERKUNG

2 Teile (jeweils 50 Punkte)

- Praktikum (kleine Anforderungsanalyse, vgl. Belegarbeit)
- Vorlesung (Fragen, wie sie auch auf den Folien formuliert sind)

Extra-Punkte für Klausur:
Analyse, wie man das gelernte aus der Veranstaltung SE 1 auf andere Fächer, Bereiche usw. anwenden konnte (spezifisch). Ca. eine Seite.
Per Email an Frau Hauptmann. Betreff: Reflexion SE 1 IA

1 EINFÜHRUNG

VO1.pdf
Folie 3

1.1 GLIEDERUNG

VO1.pdf
Folie 5

VO1.pdf
Folie 6

1.2 DIE ERSTEN PHASEN

VO1.pdf
Folie 14

- Wie werden Softwaresysteme entwickelt?
- Was ist Software-Engineering?
- Warum haben Analyse und Definition von Anforderungen an das SW-System große Bedeutung im Entwicklungsprozess?

1.3 VERGLEICH BAUINGENIEUR UND SOFTWAREENTWICKLUNG

VO1.pdf
Folie 16

Begleitend:

- Projektleiter
- technischer Autor
- Qualitäts-Beauftragter

Zertifizierung für Software: IREB

VO1.pdf
Folie 17

1.4 ERFOLGSQUOTE VON SOFTWARE-PROJEKTEN GESTERN UND HEUTE

VO1.pdf
Folie 19

Also: Analyse ist Grundvoraussetzung für erfolgreiche Softwareentwicklung.
Veranschaulichung:

VO1.pdf
Folie 20

Warum haben Analyse und Definition von Anforderungen an das SW-System große Bedeutung im Entwicklungsprozess?

- Auch heute noch werden nur die Hälfte aller SW-Projekte wie geplant realisiert.
- 60% der Fehler im SW-System resultieren aus Fehlern in der Analysephase, d.h. aus Fehlern bei der Definition der Anforderungen.
- Die Behebung von Fehlern aus der Analysephase sind vergleichsweise sehr teuer.

1.5 DEFINITIONEN SOFTWARE ENGINEERING

VO1.pdf
Folie 22

VO1.pdf
Folie 23

VO1.pdf
Folie 24

Also:

Software-Engineering ist

- die effektive und effiziente Entwicklung und Weiterentwicklung komplexer SW-Systeme
- sowie begleitender Dokumente
- in einem bewusst arbeitsteilig gestalteten Prozess
- unter Anwendung bewährter Prinzipien, Methoden und Modellen.

oder Vergleich:

VO1.pdf
Folie 25

- sauberes Arbeiten (vgl. Chirurg: kann auch so operieren, wenn er aber ohne Hygiene operiert, ist der Ausgang ungewiss)
- „Wenn's nicht funktioniert, ist man immer wieder damit beschäftigt die gleichen Fehler zu bearbeiten."

VO1.pdf
Folie 26

2 SYSTEM

Wie werden Software-Systeme entwickelt?

- Bevor mit der Implementierung begonnen wird, werden die Anforderungen ermittelt und beschrieben.
- Danach wird ausgehend von der Beschreibung der Anforderungen eine Struktur festgelegt, nach der SW-System gebaut wird.
- Erst dann beginnt die Implementierung.
- Dabei begleiten ständig Tests und Dokumentation die Arbeit.

Was ist „Software-Engineering“? (siehe weitere Definitionen)
Software-Engineering ist:

- die effektive und effiziente Entwicklung und Weiterentwicklung KOMPLEXER SW-Systeme
- sowie BEGLEITENDER DOKUMENTE
- in einem bewusst arbeitsteilig gestalteten PROZESS
- unter Anwendung bewährter PRINZIPIEN, METHODEN UND MODELLEN.

Warum haben Analyse und Definition von Anforderungen an das SW-System große Bedeutung im Entwicklungsprozess?

- Auch heute noch werden nur die Hälfte aller SW-Projekte wie geplant realisiert.
- 60% der Fehler im SW-system resultieren aus Fehlern in der Analysephase, d.h. aus Fehlern in der Definition der Anforderungen.
- Die Behebung von Fehlern aus der Analysephase sind vergleichsweise sehr teuer.

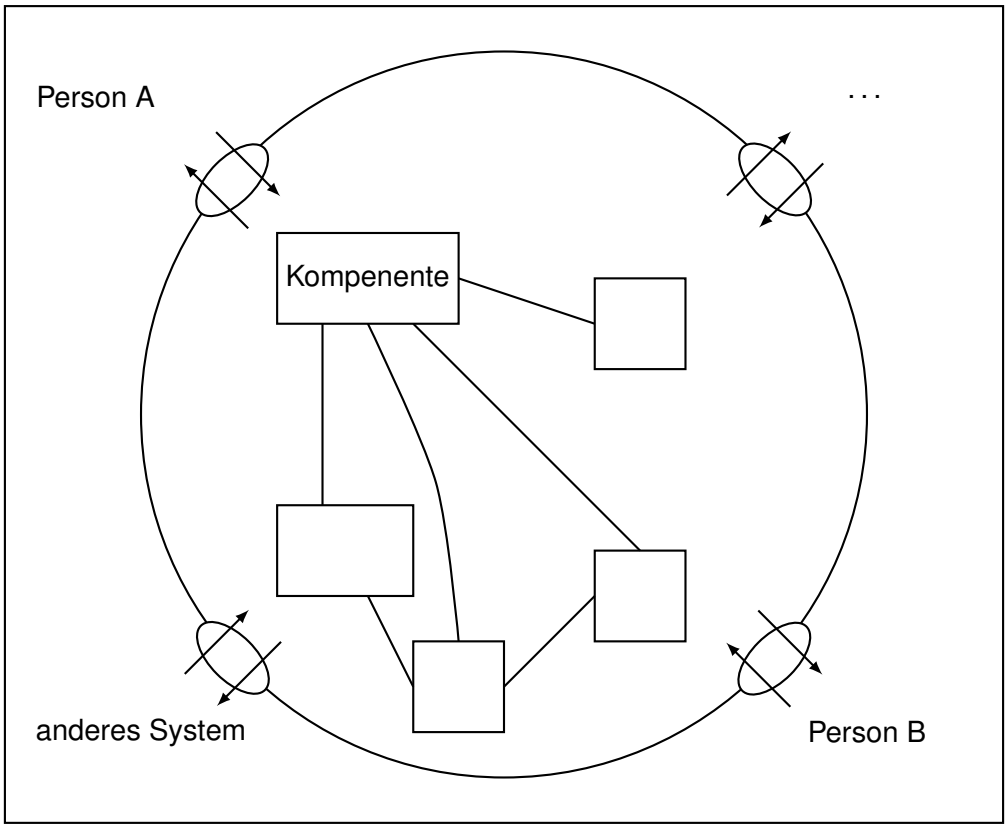
2.1 ENTWICKLUNG VON SW-ENTWICKLUNGSSTRATEGIEN

Problem der SW-Entwicklung am Ende der 60-ziger Jahre:
veränderte Rahmenbedingungen beim Einsatz und der Entwicklung von Software-Systemen
Also: Strategien bei der SW-Entwicklung ändern

2.2 DEFINITION SOFTWARESYSTEM

2.2.1 WAS KENNZEICHNET EIN SYSTEM?

- Gesetzmäßigkeiten (Regeln, . . .) im Inneren, damit es funktioniert [Wenn es ein Innen gibt, muss es auch ein Außen geben]
- $\Rightarrow$ Ein System besteht aus Komponenten, die Beziehung zueinander haben.



VDI 3633 Blatt 1: System (VDI = Verein Deutscher Ingenieure)

VO2.pdf
Folie 6

VO2.pdf
Folie 7

Also:

- Ein System besteht aus Komponenten, die miteinander in Beziehung stehen.
- Eine Grenze trennt das System von seiner Umgebung (auch Kontext genannt).
- Schnittstellen (Verbindungsstellen) verbinden das System mit seiner Umwelt.
- Ein System kann Subsysteme enthalten. Ein System befindet sich zum Zeitpunkt t in einem def. Zustand.

2.2.2 WARUM SIND SOFTWARE-SYSTEME GANZ SPEZIELLE SYSTEME?

- SW-Systeme sind „immateriell“ und komplex; sie haben keine natürliche Lokalität.
- SW-Systeme sind aus einem Werkstoff hergestellt, der von sich aus keine Strukturierung im „Großen“ erfordert (Sprich: niemand wird „gezwungen“ die Software zu strukturieren, es ist aber wohl hilfreich).

2.2.3 WELCHEN UNTERSCHIED GIBT ES ZWISCHEN MODELLEN UND PROTOTYPEN BEI DER SOFTWARE-ENTWICKLUNG?

VDI 3633 Blatt 1: Modell

VO2.pdf
Folie 10

VO2.pdf
Folie 11

Softwareprojekte können aufgrund ihrer Komplexität niemals nur durch ein Modell abgebildet werden, sondern brauchen mehrere!

- Modelle bilden ab.
- Prototypen sind lauffähig.

2.2.4 WELCHE GRUNDPRINZIPIEN FINDEN IN DER SW-ENTWICKLUNG ANWENDUNG?

- Abstraktion

  VO2.pdf
  Folie 15

- Strukturierung
- Zerlegung

  VO2.pdf
  Folie 17

  Entwicklerdokumentation braucht auch ein grobe Übersicht (nicht nur Doxygen generiertes)

- Kapselung

  VO2.pdf
  Folie 18

- Hierarchisierung

  VO2.pdf
  Folie 16

- Standardisierung
- (integrierte) Dokumentation
  $\Rightarrow$ Weitergabe von Wissen

  VO2.pdf
  Folie 22

weitere:

- Typisierung

VO2.pdf
Folie 19

- Nebenläufigkeit

VO2.pdf
Folie 20

- Persistenz

VO2.pdf
Folie 21

Also:

- Abstraktion
- Strukturierung
- Zerlegung
- Kapselung
- Hierarchisierung
- Standardisierung
- integrierte Dokumentation

$\Rightarrow$ Mittel um Komplexität zu beherrschen

3 QUALITÄT VON SOFTWARE-SYSTEMEN

3.1 WAS KENNZEICHNET QUALITATIV HOCHWERTIGE SOFTWARE-SYSTEME?

Gedanken:

- stabil laufen
- wenig Fehler
- erfüllt seine Funktion
- intuitiv und einfach benutzbar
- kompatibel
- wenig Ressourcenverbrauch (effizient)
- schöne Oberfläche (Achtung, hier wird klar: Qualität kann auch subjektiv sein)
- transparenter Systemzustand (z.B. aussagefähige Statusmeldungen)
- ...

Lösung:

- Funktionserfüllung
- Benutzerfreundlichkeit
- Wirtschaftlichkeit
- Je nach gesetzten Prioritäten:
  - Zuverlässigkeit
  - Sicherheit
  - Flexibilität (Änderbarkeit, Erweiterbarkeit, Portabilität, Kompatibilität) Wiederverwendbarkeit

VO3.pdf
Folie 2

Qualität hat mehrere Sichten:

- Benutzersicht (Außen)
- Entwicklersicht (Innen)

VO3.pdf
Folie 3

VO3.pdf
Folie 4

Wann weiß man, ob das SW-System seine Funktion gut erfüllt? Übereinstimmung von GEPLANTER (spezifiziert z.B. im Pflichtenheft) und realisierter Funktionalität.

3.2 QUALITÄTSMERKMALE

3.2.1 BENUTZUNGSSCHNITTSTELLE

(DIN EN ISO 9241-110)
„Alle Bestandteile eines interaktiven Systems (Software oder Hardware), die Informationen und Steuerelemente zur Verfügung stellen, die für den Benutzer notwendig sind, um eine bestimmte Arbeitsaufgabe mit dem interaktiven System zu erledigen."

3.2.2 USABILITY

(DIN EN ISO 9241-110)
„Ausmaß, in dem ein System, ein Produkt oder eine Dienstleistung durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um festgelegte Ziele effektiv, effizient und zufriedenstellend zu erreichen."

3.2.3 USER CENTERED DESIGN

(DIN EN ISO 9241-210)

VO3.pdf
Folie 6

- Erreichbar gut über Prototypen, schon vor Beta-Tests

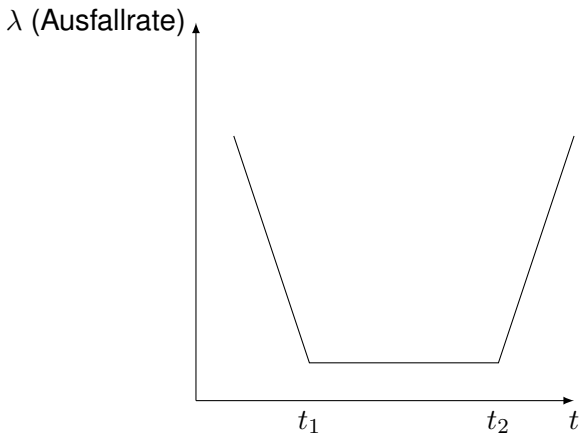
Benutzbarkeit
Achtung: Subjektiv (es gibt auch objektive Regeln); gestützt durch Oberflächenprototyp (Check-Dummy)

3.2.4 EFFIZIENZ

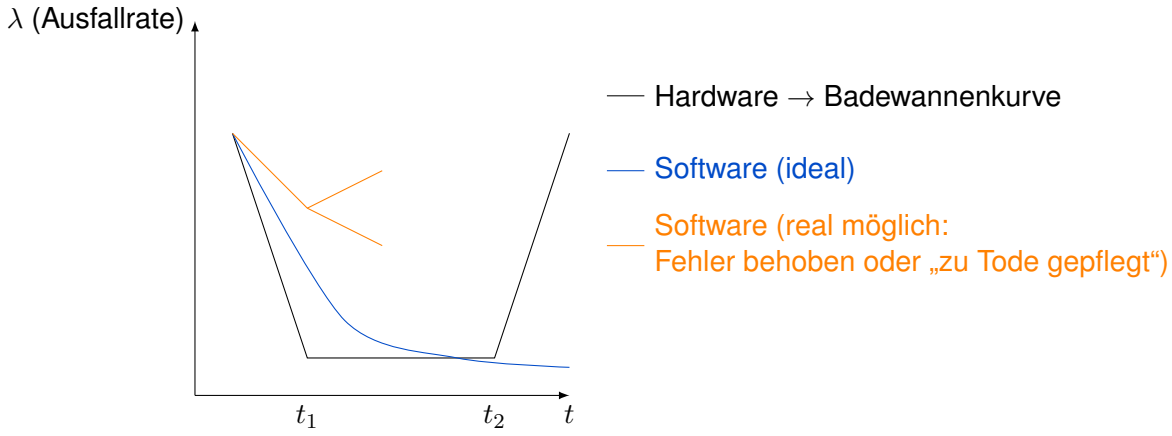
- Speicherausnutzung
- Transport
- Antwortzeit (durch bessere Algorithmen (bspw. Sortierung))

3.2.5 ZUVERLÄSSIGKEIT

- Fehlerfrei $\rightarrow$ Vermeidung von Ausfällen



- Frühausfälle durch Produktionsfehler t_1
- Spätausfälle durch physischen Verschleiß t_2



3.2.6 SICHERHEIT

- Vermeidung von gefährlichen Zuständen

Mögliche Lösung/Herangehensweise für Zuverlässigkeit und Sicherheit:
Redundanz!

3.2.7 ERWEITERBARKEIT

- neue Funktionalität kommt hinzu → welcher Aufwand resultiert daraus?

3.2.8 ANWENDBARKEIT

- gleiche Funktionalität mit anderen Eigenschaften z.B. kürzere Antwortzeit

3.2.9 KOMPATIBILITÄT

Kompatibel → anschließbar

- Das SW-System A muss zusammen mit dem SW-System B arbeiten.

3.2.10 PORTABILITÄT

Portabel → „fortschleppbar“

- Das SW-System A läuft in der Umgebung von 1 bzw. 2 (Plattform) [Linux bzw. Windows]

3.3 WIE KANN HOHE SOFTWARE-QUALITÄT ERREICHT WERDEN?

VO3.pdf
Folie 7

Analyse: man hat schon etwas und schaut nach (Test: analytische Fehlersuche).
Konstruktion: es wird erst gebaut (im Vornherein gut Programmieren).

VO3.pdf
Folie 9

SW-Qualität – erfordert Kosten und Zeit

VO3.pdf
Folie 10

Lösung: Durch gezielte Qualitätssicherung im

- organisatorischen Bereich
  (Vorgehen bei der Software-Entwicklung → Vorgehensmodelle)
- im Rahmen der Anforderungsmodellierung
- im konstruktiven Bereich
  (Analyse, Entwurf, Implementierung → Muster, Schnittstellen, Werkzeuge. . .)
- im analytischen Bereich (Test)

4 ANFORDERUNGEN

4.1 DIE ROLLE DER ANFORDERUNGEN DES KUNDEN IM RAHMEN DER SOFTWAREENTWICKLUNG

- Anforderungen sind Ausgangspunkt der Entwicklung, daraus resultiert ihre große Bedeutung (sowohl inhaltlich als auch zeitlich).
- Die Anforderungen verbinden Kunden und Entwickler.
- Die Anforderungen sind Maß der Dinge bei der Übergabe des Produktes.

4.2 WAS SIND ANFORDERUNGEN?

Beispiel für Anforderungen Das SW-System „xy“ soll...

- auf verschiedenen Plattformen laufen (Achtung: was sind die konkreten Platformen?).
- im Web-Browser laufen.
- in verschiedenen Sprachen den Benutzer führen.
- immer verfügbar sein (7/24).
- Die Zeit zur Fehlerbehebung soll minimal sein.
- eine grafische Benutzeroberfläche haben.

(sprich: soll etwas HABEN)

MS-Word soll...

- es erlauben, die Zeichen zu formatieren (fett, kursiv, ...).
- es erlauben Text zu drucken.

(sprich: soll etwas TUN)

VO4.pdf
Folie 4

4.2.1 FUNKTIONELLE UND NICHT-FUNKTIONELLE ANFORDERUNGEN

VO4.pdf
Folie 5

VO4.pdf
Folie 6

- funktionale Anforderungen:
  Funktionserfüllung
- nicht-funktionale Anforderungen
  - Qualitätseigenschaften:
    Benutzerfreundlichkeit
    Effizienz
    Zuverlässigkeit/Sicherheit
    Portabilität
    Kompatibilität
    Erweiterbarkeit
    Änderbarkeit
  - einschränkende Rahmenbedingung
    - organisatorische Rahmenbedingungen
    - technisch/technologische Rahmenbedingungen
    - rechtliche Rahmenbedingungen

Also vielmehr:

VO4.pdf
Folie 7

Unterschied technologisch vs technisch (Bsp Datenbank)

- technologisch:
  relationale, hierarchisch, ...
- technisch:
  MySQL, Oracle, ...

4.2.1.1 UNTERSCHEIDUNG DER ANFORDERNUGSTYPEN AM BEISPIEL BIBLIOTHEK

VO4.pdf
Folie 9

funktionale Anforderungen brauchen in jedem Fall ein VERB.
Auslöser-Reaktions-Tabelle (ART) für (unabhängige) funktionale Anforderungen (Bsp. Bibliothek).

Ereignis	funktionale Anforderungen	Eingabe-Daten	Ausgabe-Daten	Bemerkung (Q,R)
Person will sich in der Bibo anmelden	Benutzer anmelden	Anmelde-wunsch	Benutzer-ausweis, alternativ: Absage	darf noch nicht angemeldet sein, Alter $\geq$ 18, Wohnort $\rightarrow$ Sachsen, ...
Person will im Katalog suchen	Buch suchen	Suchanfra-ge	Suchant-wort	Antwortzeit $\leq$ 3 Sek, ...
...				

4.2.2 DEFINITION ANFORDERUNG

Eine Anforderung ist eine Beschaffenheit oder Fähigkeit, die von einem Benutzer zur Lösung eines Problems oder zur Erreichung eines Zieles benötigt wird.
Es gibt funktionale und nicht-funktionale Anforderungen.
Diese nicht-funktionalen Anforderungen sind genau genommen

- Qualitätsanforderungen oder
- einschränkende Rahmenbedingungen
  (technisch/technologische, organisatorische, rechtliche Rahmenbedingungen).

Problematische Synonyme sind: Leistungsmerkmale, feature

4.2.3 BEGRIFF FEATURE

VO5.pdf
Folie 1

5 RISIKEN DER ANFORDERUNGSANALYSE

Am Beispiel:

VO5.pdf
Folie 4

Risiko	Vermeidung	Qualitätsmerkmal
Missverständlichkeit (Uneindeutigkeit)	Regeln einhalten Schaubild: DKB AG $\xrightarrow{\text{Geld}}$ Baustelle Rückfragen beim Kunden $\rightarrow$ Kommunikation (Validierung) Prototyp	eindeutig
Anforderung ist nicht realisierbar	experimenteller Prototyp (bspw. Überprüfung der geforderten Antwortzeit) Abschätzung	realisierbar
Anforderungen sind unvollständig	Validierung ($\rightarrow$ Kommunikation mit Kunden) grafische Darstellung Prototyp (Click Dummy)	vollständig
falsch	Rückfragen $\rightarrow$ Validierung $\rightarrow$ Kommunikation mit Kunden (grafische Darstellung) [möchte Kunde mit Grafiken umgehen?]	korrekt
falsch priorisiert / zu viel Aufwand	Kommunikation mit Kunden	angemessen
widersprüchlich	offenlegen von Abhängigkeiten $\rightarrow$ (direkte) Kommunikation mit dem Kunden	widerspruchsfrei
zu umfangreich beschrieben	grafische Darstellungen	minimal

Unterschied Verifizierung/Validierung (am Beispiel):
Verifizierung: „Baue ich das Haus richtig?“
Validierung: „Baue ich das richtige Haus?“
Also:

5.1 DIE RISIKEN

Verletzung der Qualitätsmerkmale:

- Eindeutigkeit
- Realisierbarkeit
- Vollständigkeit
- Korrektheit
- Angemessenheit
- Widerspruchsfreiheit
- Minimal

5.2 DIE VERMEIDUNG

- bewusste Diskussion mit dem Kunden ($\rightarrow$ soziale Kompetenz)
- klare, eindeutige Formulierung,
- Verwendung eines Glossars
- Ergänzung der Dokumente in natürlicher Sprache durch geeignete Modelle ($\rightarrow$ Dokumente mit grafischen Darstellungen)
- Verwendung von Werkzeugen, die die Konsistenz unterstützen
- Einbeziehung von Prototypen (experimentelle Prototypen, funktionale Prototypen)

Wichtig: Die Kommunikation

VO5.pdf
Folie 6

5.3 DIE ANALYSE

VO5.pdf
Folie 8

- Ziele:
  Klare Formulierung des Problems, das durch Einsatz des SW-Systems gelöst werden soll.
  „Auf welche Frage ist das zu entwickelnde SW-System die Antwort?“
- Stakeholder:
  Anwender, Wissensträger, Interessenvertreter, Lobby, ...
- Kontext:
  Wer/was aus dem Kontext interagiert über die Schnittstellen (Verbindungsstellen) mit dem System?
  Anwender, Administrator, andere Systeme, ...

VO5.pdf
Folie 13

VO5.pdf
Folie 14

VO5.pdf
Folie 15

5.3.1 DER ANFANG DER ANFORDERUNGSANALYSE

Systemziele und Systemkontext sind die Basis für alle Anforderungen und damit auch für das gesamte Projekt. Wichtig ist ein wertungsfreier Umgang mit Systemzielen und Systemkontext. Ebenso wichtig ist es, zu Beginn alle am Projekt Beteiligten Personen mit ihren Rollen und Interessen zu kennen.

5.3.2 ERMITTLUNG DER ANFORDERUNG

- Kreavitätstechniken (Brainstorming, Wechsel der Perspektive, . . .)
- Befragungstechniken (Fragebogen, Interview, . . .)
- artefaktbasierte Techniken (Eingabe-, Ausgabedokumente, $\rightarrow$ Wiederverwendung)
- Audio-, Videoaufzeichnungen
- Anwendungsfallmodellierung, Essenzbildung (das wichtige heraus filtern)

5.3.3 ZU BERÜCKSICHTIGENDE INFORMATIONSQUELLEN

Alle Stakeholder, also alle Beteiligten.

Wichtig: Zu Beginn definieren, wer Stakeholder ist, damit klar ist, wer was zu sagen hat und wer nicht.

6 ANFORDERUNGENSANALYSE

BESCHREIBUNG ERMITTELTER ANFORDERUNGEN

- textuelle → natürlich-sprachlich
  - \+ Sprache allgemein bekannt
  - − Missverständlichkeit / Interpretierbarkeit
  - − Redundanz
- graphisch → Modellierungs-Sprache
  - \+
  - − „Sprache“ muss bekannt sein (UML, . . .)

Neutral für beide Seiten:

- ◦ kann übersichtlicher/unübersichtlicher sein (je nach Strukturierung und was der Person mehr liegt: subjektiv)
- Satzschablonen (Chris Rupp)

6.1 GRAPHISCHE BESCHREIBUNG VON FUNKTIONALEN ANFORDERUNGEN

- strukturierte (prozedurale) Programmierung (SP)
  → strukturierter Entwurf (SD)
  → strukturierte Analyse (SA)
  ⇒ SA→SD→SP
  - – Funktionsdiagramm
  - – Datenstrukturdiagramm
  - – Datenflussdiagramm
  - – ERM
  - – Zustandsdiagramm
- objektorientierte Programmierung (OOP)
  → objektorientierter Entwurf (OOD)
  → objektorientierte Analyse (OOA)
  ⇒ UML (OOA→OOD→OOP)
  - – Klassendiagramm
  - – Aktivitätsdiagramm
  - – Zustandsdiagramm
  - – Anwendungsfalldiagramm
  - ⋮

UML → offener Standard

→ in Entwicklung

→ Stereotypen können vom Entwickler definiert und benutzt werden

6.2 STRUKTUR-DIAGRAMM / UML

Ein Strukturdiagramm hat:

- Knoten
- Kanten
- Topologie (Netz, Baum, Hierarchie, . . .)
- Semantik (je nach Diagrammtyp)

UML kennt:

- Strukturdiagramme (Aufbaustruktur → statische Eigenschaften)
  - Klassendiagramm
  - Paketdiagramm
  - Verteilungsdiagramm
  - Objektdiagramm
  - Komponentendiagramm
  - Kompositionsstrukturdiagramm
- Verhaltensdiagramme (Ablaufstruktur → dynamische Eigenschaften)
  - Anwendungsfalldiagramm
  - Aktivitätsdiagramm
  - Zustandsdiagramm
  - Sequenzdiagramm
  - Kommunkationsdiagramm
  - Interaktionsübersichtsdiagramm
  - Zeitverlaufsdiagramm

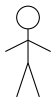
6.2.1 DAS ANWENDUNGSFALLDIAGRAMM

6.2.1.1 SEMANTIK

Das AWF-Diagramm stellt die funktionalen Anforderungen aus der Sicht der Anwender dar.
Die funktionalen Anforderungen werden zu den Beteiligten aus dem Kontext in Beziehung gesetzt.
Die Beziehung zwischen dem (externen) Anwender und dem AWF ist eine Kommunikationsbeziehung.

6.2.1.2 KNOTEN

-  Anwendungsfälle



- Akteur

6.2.1.3 KANTEN

- $\longleftrightarrow$ Kommunikationsbeziehung zwischen AWF und Akteur
-  Generalisierung / Spezialisierung zwischen Akteuren
- $\longrightarrow$ «enthält» (include), «erweitert» (extend) $\rightarrow$ zwischen AWF
  Tipp Pfeilrichtung: Durch aktiven Satz verdeutlichen (ausleihen enthält prüfen wird zu: prüfen $\overset{\text{«enthält»}}{\longleftarrow}$ ausleihen)

6.2.1.4 TOPOLOGIE

Netzförmig

6.2.1.5 BEGRIFFE

AWF $\widehat{=}$ Prozess, Szenario, d.h.: Ein AWF kapselt eine Menge von Aktionen, die sequentiell nacheinander ablaufen oder zyklisch oder bedingungsabhängig gesteuert werden.

- Der AWF wird ausgelöst durch Ereignis:
  $\rightarrow$ datengetriebenes Ereignis (bspw. Anmeldewunsch $\rightarrow$ Benutzer anmelden)
  $\rightarrow$ zeitgesteuertes Ereignis
- Der AWF führt in der Regel zu einem von außen sichtbaren Ereignis für den Anwender (bspw. Benutzerausweis als Ergebnis von „Benutzer anmelden“).

Konzept: $\xrightarrow{\text{E}} \boxed{V} \xrightarrow{\text{A}}$ (Eingabe, Verarbeitung, Ausgabe)

Akteur $\widehat{=}$ Ein AUSSERHALB DES SOFTWARESYSTEMS existierendes Objekt, das mit dem SW-System INTERAGIERT.

- ist rollenorientiert.
- ist aktiv oder passiv.
  Aktive Rollen:
  - Anwender
  - zeitliches Ereignis
  - externes System

  Passive Rolle:
  - tut nichts, wird aber zur Abarbeitung des AWF'es benötigt (bspw.: Kurstabelle für eine Wechselstube / Aktien, Fahrplan, ...) $\Rightarrow$ vorhandene Daten

6.2.2 AKTIVITÄTSDIAGRAMM

Für Anwendungsfälle, die Abhängigkeiten erschaffen (voneinander):

- Bibliothek nutzen und verwalten (Gesamtfunktion)
  - Literatur suchen
  - Benutzerdaten verwalten
    - Benutzer anmelden
    - Benutzer abmelden
    - Benutzerdaten ändern
    - Benutzerausweis verlängern
  - Ausleihdaten verwalten (intern)
  - Mahnungen verwalten

    ⋮

$\Rightarrow$ diese Anwendungsfälle sind alle unabhängig voneinander!
Abhängigkeiten entstehen, wenn man Anwendungsfälle weiter verfeinert. Beispielsweise beim Verfeinerung von „Benutzer anmelden“:
Der Fall „Anmeldewunsch prüfen“ wäre abhängig von „Anmeldewunsch annehmen“ usw.
Ein Anwendungsfall ist dann nicht mehr abstrakt, wenn es sich nicht mehr verfeinern lässt, ohne Abhängigkeiten zu erzeugen!

6.2.2.1 SEMANTIK

Ein Aktivitätsdiagramm beschreibt einen Algorithmus (einen Algoritmus, den der Anwendungsfall kapselt). D.h. das Aktivitätsdiagramm stellt dar, welche Aktionen aufeinander folgen, zyklisch oder bedingungsabhängig gesteuert sind Sequenz, Iteration, Alternative).
Besonderheit:
Aktionen KÖNNEN definierten Verantwortungsbereichen zugeordnet werden ($\rightarrow$ Swimlane)
entwickelt aus:
PAP, Petrinetze, Datenflussdiagramme

6.2.2.2 KNOTEN

- 1 Anfangs-, eventuell mehrere Endknoten
- Aktionen 
- Entscheidungsknoten 
- Synchronisationsknoten   (AND, OR, XOR)

6.2.2.3 KANTEN

- Transition
- Objektfluss 

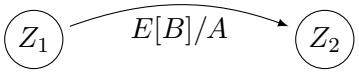
6.2.2.4 TOPOLOGIE

netzförmig

6.2.3 DAS ZUSTANDS-(ÜBERGANGS-)DIAGRAMM

6.2.3.1 SEMANTIK

Das Zustandsdiagramm stellt die einzelnen Zustände dar, die eine Betrachtungseinheit haben kann, inklusive der Zustandsübergange.
Das heißt das Zustandsdiagramm zeigt, wann ein ZUSTAND in einen anderen WECHSELT (nach welchem EREIGNIS?) und welche nach außen sichtbare Reaktion dabei erfolgt (welche AKTION wird dabei ausgeführt?)
Der Zustandswechsel kann BEDINGUNGSABHÄNGIG sein.
Zu einem definierten Zeitpunkt befindet sich die Betrachtungseinheit in einem definierten Zustand
$\rightarrow$ Modellierung von Automaten ($\rightarrow$ Steuersysteme)



6.2.3.2 KNOTEN

- 1 Startknoten, einen oder mehrere Endknoten, Zwischen-Knoten
- Entscheidungsknoten
- Terminator
- für paralleles Verhalten: Regionen mit Gabelung/Zusammenführung

⋮

6.2.3.3 KANTEN

Transition mit Ereignis, eventuell Bedingung und Aktion.

6.2.3.4 TOPOLOGIE

netzförmig

6.2.4 DAS KLASSENDIAGRAMM

6.2.4.1 SEMANTIK

Das KD stellt die KLASSEN ($\widehat{=}$ Bauanleitung für Objekte) und ihre STATISCHEN Beziehungen dar.

6.2.4.2 KNOTEN

- konkrete Klassen (instanziierbar)
- abstrakte Klassen (nicht instanziierbar)
- generische Klassen
- Interface

6.2.4.3 KANTEN

- Assoziation ($\rightarrow$ Operationsaufruf)
- Generalsierung/Spezialisierung („Vererbung“)
  Einordnung von Typen
- Aggregation (Teile-Ganze-Beziehung)
  hohles Viereck am anderen Ende des Pfeils
  Das Teil kann weiter existieren, wenn es das Ganze nicht mehr gibt.
  Bsp: Belegarbeitsgruppe $\diamond\!\!\rightarrow$ Gruppenmitglied (Gruppenmitglieder existieren noch[als Person], wenn sich Gruppe auflöst)
- Komposition (strenge Teile-Ganze-Beziehung)
  gefülltes Viereck am anderen Ende des Pfeils
  Das Teil kann nicht mehr existieren, wenn es das Ganze nicht mehr gibt.

6.2.4.4 TOPOLOGIE

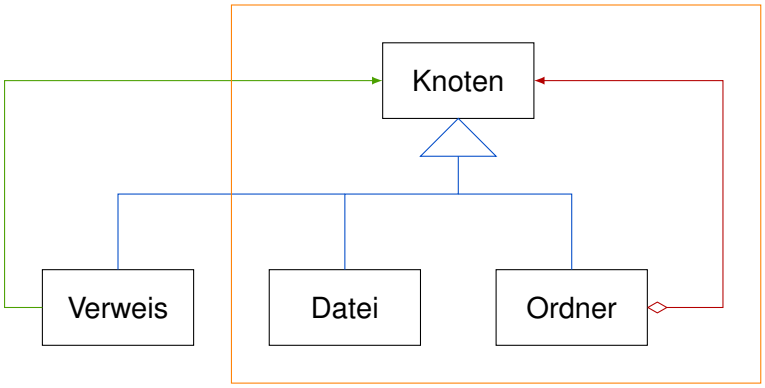
netzförmig mit eingelagerten Hierarchien

6.2.4.5 BEISPIEL

Datei-System

Knoten $\rightarrow$ Klassenkandidaten:

- Datei
- Ordner



Entwurfsmuster: Kompositum (Composite)

6.2.4.6 PERSPEKTIVEN

Drei Perspektiven eines Klassendiagramms:

1. ANALYSE-Klassendiagramm
   Perspektive des Kunden:
   Sicht auf KlasssenKANDIDATEN aus dem Anwendungskontext

   „was muss gemacht werden?“
   OOA

2. ENTWURFS-Klassendiagramm
   Perspektive des Entwerfers:
   Sicht auf die SCHNITTSTELLEN der Klassen und ihre Beziehungen → alle ÖFFENTLICHEN Operationen
   Grobentwurf (Systemarchitektur) → Feinentwurf (Komponenten-Entwurf)

   „wie muss das aus der Analyse gemacht werden ⇒ wer macht was?“
   OOD

3. IMPLEMENTATIONS-Klassendiagramm
   Perspektive des Entwicklers (Programmierers):
   Sicht auf ALLE Details einer Klasse
   (Attribute, Operationen [Signaturen: `... funktname (...)`; auch: Prototyp, Funktionskopf])

   „wie wird der Entwurf tatsächlich umgesetzt?“
   OOP

6.3 VERWENDUNGEN VON UML-DIAGRAMMEN

Klassendiagramm: statische Struktur des Softwaresystems (Perspektiven: erste Analyse / Blick auf Schnittstellen / Implementation)
Anwendungsfalldiagramm: Beschreibung der funktionalen Anforderungen (Überblick)
Aktivitätsdiagramm: Beschreibung eines Anwendungsfalls (Details)
Zustandsdiagramm: Bei welchem Ereignis ändert das Objekt seinen Zustand (und was bekommt die Außenwelt davon mit). Diese Zustände beschreiben die Werte eines Status-Variable einer Klasse.

6.4 ESSENTIELLE FUNKTIONEN

Was ist eine essentielle Funktion?

Ereignis	funktionale Anforderungen	Eingabe-Daten	Ausgabe-Daten	Bemerkung (Q,R)
beliebige Person will im Katalog suchen	im Katalog suchen	Suchanfrage	Suchantwort	–
Person will sich anmelden	Benutzer anmelden	Anmeldewunsch	Benutzerausweis / Absage	beim wiederholten Anmelden → Absage
Benutzer will sich abmelden	Benutzer abmelden	...	...	...
Benutzer will Daten ändern	Benutzerdaten ändern	...	...	...
Benutzer will Gültigkeit verlängern	Benutzerausweis verlängern	...	...	...

Benutzerdaten verwalten besteht aus Benutzer anmelden/abmelden, Benutzerdaten ändern und Benutzerausweis verlängern.
Benutzer anmelden → essentielle Funktion

Benutzerdaten verwalten → essentielle GRUPPE

Eine essentielle Funktion ist:

- kleinste, von anderen essentielle Funktion UNABHÄNGIGE Funktion
- hat genau einen AUSLÖSER
  - datengetrieben (→ Eingabedaten)
  - zeitlich
- hat kein, eine oder mehrere Reaktionen (→ Ausgabedaten)
- wiederholbar, ohne dass dazwischen eine essentielle Funktion ausgeführt werden muss

6.5 SATZSCHABLONEN

6.5.1 BESCHREIBUNG VON ANFORDERUNGEN AN EINEM SW-SYSTEM

- textlich
- Satzschablonen (nach Chris Rupp)
- graphisch (z.B. UML)
  - AWF-Diagramm
  - Aktivitäts-Diagramm
  - Zustandsdiagramm

  ⇒ Klassen-Diagramm (Analyse-Klassen-Diagramm)

6.5.2 REGEL DER SATZSCHABLONEN

VO11.pdf
Folie 1

VO11.pdf
Folie 2

6.5.3 BILDEN VON SATZSCHABLONEN

6.5.3.1 CHARAKTER DER AKTIVITÄT

VO11.pdf
Folie 3

6.5.3.2 RECHTLICHE VERBINDLICHKEITEN

VO11.pdf
Folie 4

VO11.pdf
Folie 5

VO11.pdf
Folie 6

6.5.3.3 OBJEKT EINBEZIEHEN

VO11.pdf
Folie 7

6.5.3.4 BEDINGUNGSABHÄNGIKEITEN EINBEZIEHEN

VO11.pdf
Folie 8

7 LASTEN UND PFLICHTENHEFT

VO11a.pdf
Folie 1

Unterschied Lasten $\leftrightarrow$ Pflichtenheft:

- Das Lastenheft (siehe DIN 69905 1997)
  „Das Lastenheft enthält eine Definition der Systemvision eine Beschreibung der wesentlichen Systemziele (Funktionen und Qualitäten) und benennt wichtige Kontextaspekte (z.B. Rahmenbedingungen) der vier Kontextfacetten sowie ihre Beziehung zur Vision und zu den definierten Systemzielen.“
- Das Pflichtenheft (siehe DIN 69905 1997)
  „Das Pflichtenheft detailliert die im Lastenheft beschriebene Vision und die Systemziele (abstrakte Funktionen und Qualitäten) sowie ggf. im Lastenheft definierte Rahmenbedingungen IM HINBLICK auf die angestrebte technische Umsetzung (Realisierung) des Systems.“

bzw. kurz:

- Das Lastenheft beschreibt die System-VISION des Auftraggebers.
- Das Pflichtenheft detailliert diese Vision aus der Sicht des Auftragnehmers.
  Es ist eine mögliche Realisierungsvariante.

7.1 PFLICHTENHEFT

- Adressiert an:
  1. AUFTRAGNEHMER:
     Auftraggeber übergibt Auftrag (beschrieben durch Pflichtenheft) an Auftragnehmer.
     Auftragnehmer übergibt SW-System an Auftraggeber.
  2. DRITTE / AUFTRAGGEBER / POTENZIELLE AUFTRAGNEHMER:
     Auftraggeber übergibt Auftrag zum Erstellen des Pflichtenheftes an Dritten.
     Dritter übergibt Pflichtenheft an Auftraggeber.
     Auftraggeber verteilt Pflichtenheft an potentielle Auftragnehmer, die sich für diesen Auftrag bewerben. Von denen wird einer ausgewählt.
- Charakter: Vertragscharakter

7.1.1 PFLICHTENHEFT MUSS ENTHALTEN

Alle Anforderungen an das zu entwickelnde SW-(Teil-)System, die (zu dieser Zeit) erkennbar sind.
D.h. funktionale und Qualitätsanforderungen ebenso wie einschränkende Rahmenbedingungen (organisatorische, rechtliche und technische/technologische).
Da es in der Regel zwei Adressaten (AG/AN) und quasi Vertragscharakter hat, muss es analog zur Anforderungsanalyse folgende Eigenschaften haben:

- verständlich sein für den Auftraggeber und hinreichend präzise für den Auftragnehmer,
- korrekt, vollständig, eindeutig, widerspruchsfrei, minimal, erweiterbar sein,
- realisierbar, nachvollziehbar und hinsichtlich der Erfüllung überprüfbar sein.

7.1.2 AUFBAU EINES PFLICHTENHEFTS

1. Ausgangssituation und Zielsetzung
2. Systemeinsatz und Umgebung
3. Benutzerschnittstellen
4. Funktionale Anforderungen
5. Qualitätsanforderungen
6. Rahmenbedingungen (techn./technologisch, org., rechtlich)
7. Fehlertoleranzmaßnahmen
8. Anforderung an die Dokumentation
9. Abnahmekriterien

   Glossar (Begriffslexikon)

   Index

   Anhang

8 VORGEHENSMODELLE DER SW-ENTWICKLUNG

8.1 SW-LEBENSZYKLUS

VO12.pdf
Folie 8

(auch „Wasserfallmodell“ genannt)

8.1.1 SW-LEBENSZYKLUS MIT RÜCKKOPPLUNG

VO12.pdf
Folie 9

8.2 V-MODELL

VO12.pdf
Folie 11

VO12.pdf
Folie 12

8.2.1 INTERAKTION DER SUBMODELLE

VO12.pdf
Folie 13

8.2.2 DAS SUBMODELL SWE

VO12.pdf
Folie 14

8.2.3 ROLLENZUORDNUNG

VO12.pdf
Folie 15

VO12.pdf
Folie 16

8.3 SPIRALMODELL NACH BOEHM

VO12.pdf
Folie 18

Jeder Quadrant wird pro Phase nur einmalig besucht!

8.4 ITERATIV-INKREMENTELLES VORGEHEN

VO12.pdf
Folie 21

8.4.1 RATIONAL UNIFIED PROCESS (RUP)

Unified Process → Rational Unified Process (RUP) (Booch, Jakobsen, Rumbaugh)

VO12.pdf
Folie 22

VO12.pdf
Folie 23

8.4.2 ITERATIV-INKREMENTELLES VORGEHEN

VO12.pdf
Folie 24

8.5 VERGLEICH DER MODELLE

8.5.1 SW-LEBENSZYKLUS

Wesen

- SW-Entwicklungsprozess ist zerlegt in sachliche Phasen, die aufeinander folgen (→ sachliche + zeitliche Gliederung).
- Das Ergebnis einer Phase ist Ausgangspunkt der folgenden Phase.

Vorteil

- Einfache Projektverwaltung

Nachteil

- KEINE Rückkopplung
- bzw. mit Rückkopplung: Im Fehlerfall wird nur um eine Stufe zurück gegangen.

Rahmenbedingung

- Alle Anforderungen stehen zu Beginn fest.

Rahmenbedingung siehe SW-Lebenszyklus

8.5.2 V-MODELL

Wesen

- Standard im öffentlichen Bereich
- 4 Submodelle (QS, SWE, KM, PM)
- Submodell SWE betrachtet den gesamten Geschäftsprozess (System-/DV-/Software-Analyse zur Implementation bis zur Integration und wieder zurück mit jeweils Rückkopplung zur Analyse)

Vorteil

- Einbeziehung der QS, KM, PM

Nachteil

- Großer Verwaltungsaufwand.

Rahmenbedingung

- Alle Anforderungen stehen zu Beginn fest.

8.5.3 SPIRALMODELL

Wesen

- klare Zielbestimmung
- Suche nach LösungsALTERNATIVEN
- Evaluierung der Lösungsalternativen auf der Basis der Risikoabschätzung $\rightarrow$ Risikominimierung

Vorteil

- Einbeziehung der alternativen
- Risikobetrachtung

Nachteil

- fehlende Rückkopplungen

Rahmenbedingung

- Alle Anforderungen stehen zu Beginn fest.

8.5.4 ITERATIV-INKREMENTELLES VORGEHEN

Wesen

- Gesamtfunktionalität wird in Teilfunktionalitäten aufgeteilt (Inkremente), die iterative entwickelt werden

  $\rightarrow$ evolutionärer Prototyp

Vorteil

- Komplexitätsbeherrschung durch „kleine Schritte“

Nachteil

- hoher Verwaltungsaufwand

Rahmenbedingung

- Anforderungen können sich in geringem Maß ändern
- Es müssen nicht alle Anforderungen zu Beginn bekannt sein

9 AGILE SOFTWAREENTWICKLUNG

Sich ändernde Anforderungen des Kunden gehören zum Projekt. D.h. Wandel ist integraler Bestandteil des Projektes.

9.1 AGILITÄT

„Kleine Schritte führen zum Ziel.“

- Stück für Stück implementieren
- immer wider überprüfen
- immer miteinander reden
- einfache Lösungen suchen und implementieren
- wenn nötig korrigieren

9.1.1 KORREKTUR

Korrigiert wird: der bisherige Aufbau des Systems
Also:
Wechsel von ARCHITEKTUR-ZENTRIERT $\rightarrow$ CODE-ZENTRIERT
Achtung: es ist nicht gemeint, dass nur noch gecodet wird.

9.1.2 BEDEUTUNG

- Akzeptanz von Wandel
- Forderung von Wandel, wenn es sinnvoll erscheint

$\Rightarrow$ SW-Entwicklungsteam muss gewandt sein
nötig ist:

- umfangreiches theoretisches Wisen
- praktische Erfahrung
- Bereitschaft zur Überarbeitung und Änderung

9.1.3 MANIFEST

Es wird unterscheiden zwischen wichtigem und wichtigerem:

wichtigER	wichtig
Individuen und Interaktion	Prozesse und Werkzeuge
funktionierende Software	umfangreiche Dokumente
Zusammenarbeit mit Kunden	Vertragsverhandlung
auf Änderungen reagieren	einem Plan folgen

Achtung: wichtiges ist nicht unwichtig, nur weniger wichtig im Vergleich zum jeweiligem wichtigerem!

9.1.4 BEISPIELE

XP: extremes Programmieren
SCRUM

9.2 EXTREMES PROGRAMMIEREN

VO13.pdf
Folie 9

9.2.1 4 GRUNDWERTE

VO13.pdf
Folie 10

9.2.2 PRAKTIKEN

VO13.pdf
Folie 11

9.2.3 ZUSAMMENFASSUNG

VO13.pdf
Folie 12

9.2.4 CHANCEN

VO13.pdf
Folie 13

9.3 SCRUM

VO13.pdf
Folie 14

VO13.pdf
Folie 15

VO13.pdf
Folie 16

10 ANALYSE UND TEST

10.1 TESTS

Es kann bspw. getestet werden auf:

- Geschwindigkeit (Antwortzeit)
- Was mit einem Input als Output raus kommt. Bspw. bei:
  - Funktion (bei der Implementierung → Whiteboxtest)
  - Klassen (KOMPONENTENTEST)
  - Zusammenarbeit zwischen Klassen/Komponenten (INTEGRATIONSTEST, man testet Schnittstellen)
  - ganzes System (SYSTEMTEST → Blackboxtest)
    Perspektive: des Kunden

10.2 SYSTEMTEST

VO15.pdf
Folie 6

⇒ Systematischer Test aus der Perspektive des Kunden
Daten die zu testen sind:

- kritische Daten (bspw. Division durch 0)
- Grenzdaten (bspw. 18 bei Zulassung ab 18)

10.2.1 TESTDATEN, TESTFALL UND RAHMENBEDINGUNGEN

VO15.pdf
Folie 7

10.3 ZUSAMMENHANG ZWISCHEN ANFORDERUNGSANALYSE UND TEST EINES SW-SYSTEMS

- Die funktionalen Anforderungen (konkrete Anwendungsfälle, insbesondere die essenziellen Funktionen) sind wichtige TESTFÄLLE im Kontext des Systemtestes.
- Die Auslöser/Reaktionen der essenziellen Funktionen (Ein- und Ausgabedaten zu den konkreten Anwendungsfällen) sind wichtige TESTDATEN im Kontext des Systemtestes.
- Die Beschreibung der Anwendungsfälle (textlich, Aktivitätsdiagramm, Zustandsdiagramm, Satzschablonen) können wichtige RAHMENBEDINGUNGEN für die jeweiligen Testfälle enthalten.

11 KONFIGURATIONSVERWALTUNG

11.1 DER SINN VON KONFIGURATIONSVERWALTUNG

Die Konfigurationsverwaltung ist eine Rolle oder Organisationseinheit im Kontext des SW-Entwicklungsprozesses, die die SW-Einheiten

- identifiziert,
- bei Bedarf bereitstellt
- ihre Änderungen überwacht und dokumentiert.

Dazu gehört auch die Rekonstruktion älterer SW-Einheiten und Konfigurationen.
D.h. ein Konfigurationsverwaltungssystem ermöglicht das effiziente Bereitstellen definierter Konfigurationen eines SW-Systems abhängig von Varianten und Versionen.
Das Konfigurationsmanagement ist demzufolge eine Methode für die Organisation der täglichen Arbeit. ($\rightarrow$ organisatorische Qualitätssicherung)

11.2 BEZIEHUNG ZWISCHEN KONFIGURATIONS-, VARIANTEN- UND VERSIONSVERWALTUNG

Versions- und Variantenverwaltung sind Bestandteile der Konfigurationsverwaltung.

11.3 BEGRIFFE

- SOFTWARE-EINHEIT:
  atomar im Sinne ihrer Verwaltung, d.h. sie kann unabhängig von anderen SW-Einheiten bearbeitet/gespeichert/ausgetauscht werden.
  Änderung einer SW-Einheit führt zu einer neuen VERSION.
  $\rightarrow$ d.h. eine Version hat (in der Regel) einen Vorgänger und einen Nachfolger.
  $\rightarrow$ d.h. Versionen stehen in zeitlicher Ordnung (Chronik).
  Beachte Unterschied zwischen REVISION: SW-Einheit wird KOMPLETT ersetzt (nicht nur zu Teilen
- VARIANTE:
  existiert nicht zeitlich hintereinander, sondern nebeneinander (d.h. zeitgleich)
- KONFIGURATION:
  eine Menge von SW-Einheiten, die für einen definierten Zweck zusammengestellt ist und diese SW-Einheiten passen auch zueinander (= Auswahlverfahren).
- BASELINE:
  Konfiguration, die stabil ist (zusammengestellt und geprüft).

11.4 AUFGABEN DER KONFIGURATIONSVERWALTUNG

- SW-Einheiten identifizieren
- SW-Einheiten bereit stellen:
  - zum Verändern
  - zum Integrieren
- Änderungen von SW-Einheiten überwachen und verwalten
- Rekonstruktion älterer SW-Einheiten

$\Rightarrow$ d.h. Versions- und Variantenverwaltung ist Bestandteil der Konfigurationsverwaltung.
Wie werden dies Aufgaben realisiert?

- Welche Arbeitsorganisation?
- Wie wird diese Arbeitsorganisation technisch unterstützt (Subversion, Git, ...)? $\rightarrow$ Repository mit geeigneter Speichertechnik

11.4.1 ARBEITSORGANISATION

$\rightarrow$ Regeln

1. Die Entwickler TEILEN sich den Quellcode.
2. Alle Entwickler arbeiten in der gleichen ENTWICKLERUMGEBUNG.
3. Die Konnfigurationsverwaltung hat das Monopol bei der Bereitstellung von SW-Systemen. Damit wird vermieden, dass
   - unbemerkt Varianten entstehen
   - unbemerkt Änderungen vorgenommen werden
   - jemand direkten Zugriff auf die Dateien hat
4. SW-Einheit wird (als Arbeits-zwischen-Ergebnis) zyklisch verwaltet:
   - abholen
   - ändern
   - abliefern und bekanntmachen $\rightarrow$ Commit
5. Veraltete Versionen werden archiviert
6. Verzweigungen werden als Varianten verwaltet
7. SW-Systeme, die an Kunden ausgeliefert werden, werden mit dem Konfigurationsmanagementsystem zusammengestellt und präzise dokumentiert. Verwendet werden dabei nur geprüfte und freigegebene SW-Einheiten.

Werkzeug mit Repository

- zentrale Versionsverwaltung
- dezentrale Versionsverwaltung

11.4.1.1 ZENTRALE VERSIONSVERWALTUNG

- Entwickler und Reposiory-Umgebung (in der Regel auf einem Server) sind getrennt.
- Jeder Entwickler arbeitet auf einer Kopie.

$\rightarrow$ Subversion

11.4.1.2 DEZENTRALE VERSIONSVERWALTUNG

- Entwickler- und Repository-Umgebung sind nicht getrennt.
- Jeder Entwickler verfügt über ein eigenes Repository (Klon). Aus diesem Repository arbeitet er.

$\rightarrow$ Git