Fachhochschule Wedel

University of Applied Sciences

Software Engineering

Zusammenfassung

---

erstellt im:	Dezember 2000
von:	Alexander Markowski
Basis:	Vorlesung im WS 00/01

Inhaltsverzeichnis

1 Qualität

Qualität ist die Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder einer Tätigkeit, die sich auf die Eignung zur Erfüllung gegebener Erfordernisse beziehen.

1.1 Qualitätsmerkmale für die Anwendung

1.1.1 Korrektheit

- Übereinstimmung zwischen funktioneller Spezifikation und Programmfunktionalität
- Korrektheit schwer nachzuweisen. Wenn nur in Teilalgorithmen. $P = p^n$

1.1.2 Effizienz

- $\neq$Effektivität
- Bestimmt Bedarf an Betriebsmitteln.
- Effizientes Programm: kein unnötiger Verbrauch an Betriebsmitteln
- Konflikt zu: Änderbarkeit, Testbarkeit, Portierbarkeit
- Unterschied zwischen Laufzeit-/Speichereffizienz (Kurve)

1.1.3 Zuverlässigkeit

- Zusammenspiel von Korrektheit, Robustheit und Verfügbarkeit
- Wahrscheinlichkeit, dass ein System seine Funktion während eines Zeitintervalls korr. erfüllt

1.1.4 Verfügbarkeit

- Wahrscheinlichkeit, dass ein System zu einem gegebenen Zeitpunkt funktionsfähig ist.
- $V = \frac{MTBF}{MTBF+MTTR}$

1.1.5 Robustheit

- Definierte und sinnvolle Reaktion des Programms bei beliebiger ext. Kommunikation
- Verhindern von undefinierten Systemzuständen.
- Wichtig: Abfangen fehlerhafter Benutzereingaben
- Beseitigung der Fehlersymptome, nicht der Ursachen

1.1.6 Benutzerfreundlichkeit

- Aufgabenangemessenheit
- Selbstbeschreibungsfähigkeit
- Steuerbarkeit
- Erwartungskonformität
- Fehlerrobustheit

1.1.7 Datensicherheit

- Schutz gegen unerwünschte bzw. unerlaubte Verfälschung/Zerstörung von Daten
- Behandlung von Ausnahmesituationen (Stromausfall, Systemabsturz)
- Kombination von Softwaretechnischen und Organisatorischen Maßnahmen

1.2 Qualitätsmerkmale für die Entwicklung

1.2.1 Verständlichkeit

- (Voraussetzung für Änderbarkeit und Reparierbarkeit)
- Maß für den Aufwand eine fremde Software zu verstehen.
- Maßnahmen zur Erhöhung der V. „sprechende Bezeichner“ , Inlinedokumentation

1.2.2 Änderbarkeit

- Möglichkeit zur Anpassung von SW an veränderte Einsatzbedingungen und Anforderungen
- Abhängig von einer geeigneten Modularisierung

1.2.3 Prüfbarkeit (Testbarkeit)

- Möglichkeit zum Testen einer SW hinsichtlich Korrektheit, Robustheit und Zuverlässigkeit
- Wesentlich abhängig von Modularität und Strukturierung

1.2.4 Wiederverwendbarkeit

- Verlängerung der Lebensdauer. Anlegen von Bibliotheken.
- Ziel: Senkung der Entwicklungskosten

2 Ziele

Es gibt 3 Ziele die man verfolgt: hohe Qualität, niedrige Kosten und geringen Zeiteinsatz

3 Prinzipien des Software-Engineering

- Abstraktion
  - Verallgemeinerung durch Vernachlässigen von Eigenschaften
  - Wesentliches und Unwesentliches trennen
  - bei Softwareentwicklung Konzentration auf Daten- und Informationsflüsse
- Zerlegung
  - Bestandteile vom Ganzen abtrennen
  - Bestandteile getrennt betrachten und bearbeiten
- Perspektivenbildung
  - Separate Betrachtung/Analyse eines Sachverhaltes unter verschiedenen Blickwinkeln
  - Einnehmen unterschiedlicher Perspektiven stellt große Anforderungen an Systementwickler
- Einige weitere Prinzipien
  - Integrierte Dokumentation
  - Mehrfachverwendbarkeit
  - Standardisierung

4 Methoden des Software-Engineereing

4.1 Top-Down (Outside-In)

- Ausgangspunkt: abstrakte, umfassende Zielvorgabe für das System als vollständige Beschreibung
- Mit jedem Entwicklungsschritt Zerlegung in Teilsysteme und Konkretisierung

4.2 Bottom-Up (Inside-Out)

- Ausgangspunkt: Konkrete Basismaschine mit bekannter Funktionalität
- Schrittweise Abstraktion und Kombination bekannter Einzelelemente zu größeren Systemeinheiten
- Letzter Schritt: Realisierung der Benutzermaschine

5 Phasen der Software-Entwicklung

- Problemanalyse und Planung
- Systemspezifikation
- Entwurf
- Implementation
- Integration und Test
- Wartung

6 Lastenheft

- Das Lastenheft enthält eine Zusammenfassung der fachlichen Basisanforderungen an das Software-Produkt aus Sicht des Auftraggebers
- Konzentration auf fundamentale Eigenschaften der Software. Beschreibung des „Was“, nicht des „Wie“
- Relativ abstrakte, verbale Darstellung der Software-Eigenschaften in schriftlicher Form (wenige (einige)DIN A4-Seiten)
- Adressaten sind Auftraggeber und Auftragnehmer mit Leitungs-/Planungsaufgaben
- Das Lastenheft ist das erste Dokument, daß die Anforderungen an ein neues Produkt grob beschreibt

7 Aufwandschätzung

Abschätzen der Kosten aus Hersteller Sicht. Entwicklungskosten setzen sich zusammen aus:

- Personalkosten (größter Teil)
- Kosten der Entwicklungsumgebung
- Anteile an Gemeinkosten

Die meisten Schätzmethoden beruhen auf Umfangsschätzung nach LOC.
Gruppen von Einflussfaktoren auf die Schätzung:

- Qualität
- Quantität
- Entwicklungskosten
- Entwicklungsdauer

Verbindung zu Teufelsquadrat.

Qualität: Festlegung von Qualitätsstufen und Zuordnung von Kennzahlen.

Quantität: Kriterium der Größe anhand der Anzahl von LOC. Probleme: frühzeitige Schätzung und abhängigkeit von der eingesetzten PS.

Entwicklungsdauer: Absolute Zeitbedarf zur Entwicklung eines SW Produktes. E. kann durch mehr Mitarbeiter verkürzt werden. Der Verkürzungseffekt durch zusätzliche Mitarbeiter sinkt mit der Gesamtzahl an Mitarbeitern im Team. $E_{opt.} = 2.5 \cdot (\text{Aufwand in MM})^s$

7.1 Analogiemethode

- Vergleich der zu entwickelnden SW mit bereits abgeschlossenen SW-Entwicklungen.
- Ähnlichkeitskriterien sind: Gleiches oder ähnliches - Anwendungsgebiet, Produktumfang, Komplexitätsgrad, PS.
- Aus Erfahrungen des Schätzenden wird Ähnlichkeit abgeschätzt (1/4 der Zeit für Neuentw.)
- Nachteile: intuitive Schätzung, nicht nachvollziehbar, keine allg. Vorgehensweise

7.2 Relationsmethode

- Vergleich mit ähnlichen Entwicklungen. Formalisierte Aufwandsanpassung mit Faktorlisten.
- Dadurch wird Mehr oder Minderaufwand ermittelt.

7.3 Multiplikatormethode

Zerlegung des SW Produktes in Teilsysteme bis jedem Teilsystem ein Aufwand zugeordnet werden kann. Multiplikation der Anzahl der LOC jeder Kategorie mit dem Gewichtungsfaktor.

7.4 Methode der parametrischen Gleichungen

Durchführung von Regressionsanalysen zur Ermittlung der Einflussstärke von Faktoren auf den Gesamtaufwand. Basis große Anzahl von Entwicklungsprojekten und Vielzahl von Faktoren.

7.5 Prozentsatzmethode

Ermittlung von typischen prozentualen Verteilungen des Entwicklungsaufwandes auf die verschiedenen Entwicklungsphasen. Zwei mögliche Vorgehensweisen:

- Eine Phase abschließen, aus dem Aufwand den Gesamtaufwand berechen
- Eine Phase detailliert schätzen, daraus Gesamtaufwand schätzen

	Bertelsmann (nach Bender 83)	Hewlett-Packard (nach Grady 92)
Definition	30%	18 %
Entwurf	30%	19 %
Codierung	15-20%	34 %
Test	20-25%	29 %

Tabelle 7.1: Typische Verteilungen

7.6 Gewichtungsmethode

Bestimmung von Merkmalen, die für die Schätzung relevant sind:

- Subjektive Merkmale (wie gut ist Mitarbeiter)
- Objektive Merkmale (wie geeignet ist PS)

Jedem Merkmal sind Gewichtungswerte zugeordnet. Die Werte werden mathematisch verknüpft und der Gesamtaufwand berechnet.

7.7 Function-Point-Methode

Ausgangspunkt:

- Aufwand hängt vom Umfang und vom Schwierigkeitsgrad ab
- Umfang wird nicht in LOC ausgedrückt, sondern aus Produktnforderung ermittelt

1. Vorgehen: Zuordnung jeder Anforderung in eine Kategorie:
   - Eingabedaten
   - Ausgabedaten
   - Abfragen
   - Datenbestände
   - Referenzdaten
2. Einordnen der Anforderungen in eine Klasse:
   - einfach
   - mittel
   - komplex
3. Eintrag in Berechnungsformular:
   - Die Häufigkeiten der ermittelten Kategorien werden entsprechend ihrer Klassifizierung in das Formular eingetragen.
   - Die Zeilensumme sind Function Point (FP) für die jeweilige Klasse.
   - Alle Zeilensummen werden aufsummiert und ergeben die unbewerteten FP.
4. Berechnung der bewerteten FP:
   - Bewertungszahlen werden addiert
   - Es ergibt sich eine Summe zwischen 0 und 60
   - Dieser Wert wird durch 100 dividiert
   - Zum Ergebnis werden 0,7 addiert (!?)
   - Wert wird mit den unbewerteten FP multipliziert
5. bewertete Gesamtpunkte in Tabelle nachsehen $\Rightarrow$ Aufwand in MM

nicht berücksichtigt: Wiederverwendung

8 Systemspezifikation

Es gibt 4 Hauptsichten auf ein System

Daten: Datenkataloge, ER-Diagramme

Funktionen: Datenflußdiagramme, Funktionsbäume

Dynamik: Petri-Netze

Benutzeroberfläche:

8.1 Datenflussdiagramme

Enthalten alle Funktionen/Prozesse des Systems, die Eingabedaten in Ausgabedaten wandeln und Verknüpfungen der Funktionen auf Ein - Ausgabeebene, außerdem Speicher für Daten Aufbewahrung. Symbolik:

8.2 Datenkataloge

Strukturierte Darstellung aller Datenelemente, die im betrachteten System auftreten. Verwendung von formalen, textuellen Notationen zum Aufbau der Definitionen. Bsp. Fluganfrage = *Frage nach Verbindung* Termin + Route

8.3 Funktionsbäume

- Grafische Darstellung d. Hierarchien v. Funktionen/Prozessen/Aufgaben
- Kriterien für Hierarchiebildung:
  - Besteht-aus-Beziehung
  - Ruft-auf-Beziehung

8.4 Prozessspezifikation

Beschreibung der Abläufe im System z.B. durch Strukturierte Sprache. (Wenn-Dann-Sonst)

8.5 ER Diagramme

Entity Relationship Diagramme als grafisches Werkzeug zur Modellierung der Datenaspekte eines Systems. Darstellung logischer Beziehungen zwischen den Datenbeständen.
Objekttypen: Entities
Beziehungstypen: Relationships

8.6 Jackson Diagramme

Einheitliche, grafische Darstellung für Datenstrukturen und Kontrollstrukturen. Die Darstellung ist mit einer Zeitachse(Reihenfolge) verbunden. J-D bestehen aus folgenden Komponenten:

- Sequenz
- Auswahl
- Wiederholung

8.7 Petri Netze

Modellierungsmethodik für dynamische Systeme. Besonders geeignet für nebenläufige und nicht deterministische Vorgänge. Dynamik beruht auf

1. Belegung von Stellen mit Marken
2. Schaltregeln der Transitionen

Allgemeine Schaltregel

- Eine Transition kann schalten, wenn jede Eingabestelle mind. eine Marke enthält
- Schaltet eine Transition, dann wird aus jeder Eingabestelle eine Marke entfernt und zu jeder Ausgabestelle eine hinzugefügt

8.7.1 Bedingungs/Ereignis Netze

- Jede Stelle kann genau eine oder keine Marke enthalten
- Schaltregel: Eine T. kann schalten, wenn jede ihrer Eingabestellen eine Marke enthält und jede ihrer Ausgabestellen frei ist

8.7.2 Stellen/Transitions Netze

- Stellen können mehrere Marken enthalten
- Angabe von Kantengewichtungen
- Schaltregel: Eine T. kann schalten, wenn Eingabestellenmarken größer als Kantengewichtung
- Nach dem Schalten Kapazität der Ausgangsstellen nicht überschritten ist

8.7.3 Prädikat/Transitions Netze

- Individuelle Marken
- Explizit formulierte Schaltbedingungen und Schaltresultate
- Schaltregel: Eine T. kann nur schalten, wenn die in den Schaltbedingungen genannten Variablen mit Marken versorgt werden können
- Marken vorhanden sind , für die die Schaltbedingung gilt

9 Modularisierung

Modul: Zusammenfassung von Operationen und Daten einer in sich geschlossenen Aufgabe. Die Komunikation eines Moduls mit der Außenwelt erfolgt über definierte Schnittstellen.

9.1 Modulgeschlossenheit

- Aufgabenstellung muß abgegrenzt sein
- Möglichst eigenständige Bearbeitung eines abgegrenzten Aufgabenbereichs
- Bsp.: Sysutils, nicht geschlossen! Wirrwarr div. Funktionen

9.2 Modulbindung

- Summe der Beziehungen, die zwischen Bestandteilen eines Moduls bestehen
- Daten und Operationen sollen in einem engen Zusammenhang stehen

9.3 Modulkopplung

- Anzahl der Beziehungen der Module untereinander
- Hohe Modulkopplung = große Anzahl von Schnittstellen
- Niedrige M. =¿ hohe Unabhängigkeit der Module
- Gründe für hohe Modulkopplung:
  - Zusammengehörige Operationen in mehreren Module
  - Zahlreich globale Variablen
- Direkt abhängig v. d. Zahl der Module

9.4 Minimalität der Schnittstelle

- Merkmale minimaler Schnittstelle:
  - Wenige globale Variablen

  - Wenige Prozeduren
  - Wenige Parameter

- Wenige Schnittstellen tragen zu geringer Modulkopplung bei

9.5 Modulgröße

- Kleine Module sind leichter zu überschauen, zu verwenden und zu testen

9.6 Testbarkeit

- Gute Testbarkeit , wenn ein Modul ohne Kenntnis seiner Verwendung im Gesamtsystem auf seine Korrektheit geprüft werden kann
- Minimale Schnittstellen fördern Testbarkeit
- Hohe Modulkopplung behindert Testbarkeit

9.7 Interferenzfreiheit

- Ausschluß von Nebenwirkungen auf andere Module
- Wichtig um Änderbarkeit zu gewährleisten
- I. wird durch Aufgabenverteilung auf Module behindert

9.8 Importzahl

- Anzahl der Module, die zu Implementierung eines Moduls importiert werden
- Hohe Importzahl = hohe Modulkopplung
- Niedrige Importzahl Hinweis auf zu umfangreiche Module

9.9 Verwendungszahl

- Anzahl der Module, in denen ein Modul benutzt wird
- Hohe Verwendungszahlen: Einerseits Indiz für gute Wiederverwendbarkeit aber anderseits Hinweis auf geringe Modulbindung $\Rightarrow$ übertriebene Zerlegung

9.10 Modulhierarchie

Es gibt 4 Ebenen

- Steuermodul für Koordinationsarbeiten
- Problemorientierte Module (Algorithmen)
- Hilfsmodule (häufig benötigte Operationen)
- Kommunikationsmodule (mit Betriessystem und Hardware)

9.11 Zusammenfassung

- Ausgewogenes Verhältnis zwischen Modulbindung und Modulkopplung
- Minimale Schnittstellen
- Mittlere Import und Verwendungszahlen
- Klare Hierarchie

9.12 Module zur Datenkapselung

- Zweck: Direkten Zugriff auf Daten von anderen Programmteilen aus durch Kapselung der Daten im Modul verhindern.
- Datenkapsel besteht aus abstrakten Datenstrukturen/-typen und Zugriffsroutinen
- Konkreter Aufbau Datenstrukturen/-typen außerhalb des Moduls unbekannt
- Diese Module exportieren nur Zugriffsroutinen
- Vorteile: Wartbarkeit steigt, ungeeignete Zugriffe werden vermieden

9.13 Datenstruktur/Datentyp

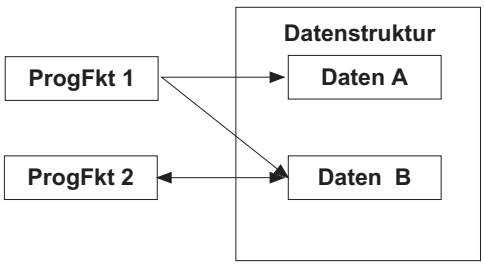

Abbildung 9.1: Ungekapselte Zugriffe

Abstrakte Datenstruktur

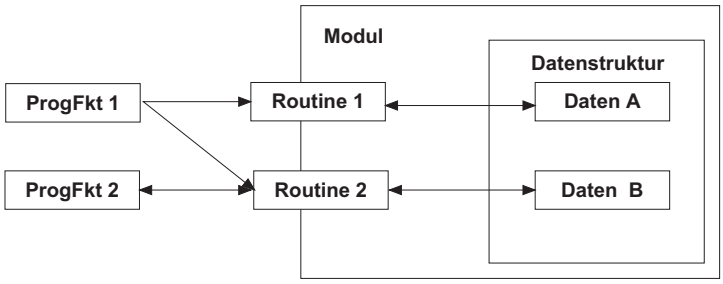

Abbildung 9.2: Gekapselte Zugriffe (abstrakte Datenstruktur)

- Einzelner Datenbestand nur durch Routine erreichbar
- Daten selbst im Modul gekapselt
- Außerhalb nur Zugriffsroutinen bekannt

Abstrakter Datentyp

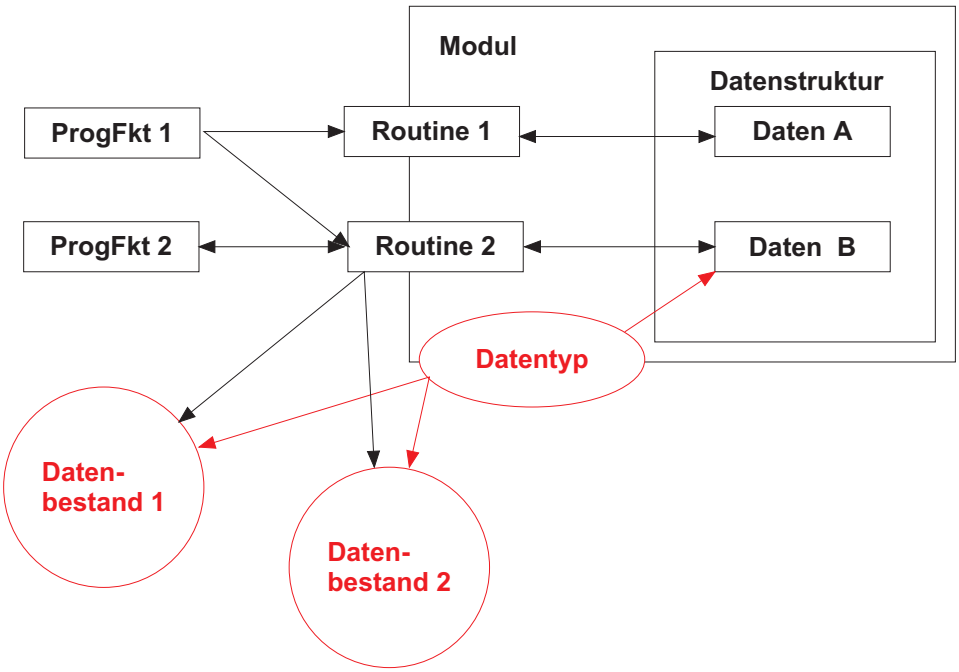

Abbildung 9.3: Gekapselte Zugriffe (abstrakter Datentyp)

- Bezeichnung des Datentyps und Zugriffsroutinen außerhalb bekannt
- Innerer Aufbau des D. ist außerhalb unbekannt
- Beliebig viele Datenbestände des Datentyps können gebildet werden, Daten selbst außerhalb des Moduls

10 Testverfahren

Allgemeines:

- Explizite Festlegung der Werte / Resultate ist notwendig (vor dem eigentlichen Test)
- Programme sollen nicht durch Ersteller getestet werden
- Reproduzierbarkeit von Tests ist wichtig
- In Programmteilen mit vielen Fehlern ist die Wahrscheinlichkeit weitere zu finden sehr groß
- Robustheit: Sinnlose Eingabewerte sind keine Fehlerquellen
- Qualitätssicherung:
  - Testen
  - Debuggen
  - Verifikation
  - Validierung

Grundsätzlich: Blackbox und Whitebox Verfahren

10.1 Statische Testverfahren

- SW wird nicht auf dem Computer ausgeführt
- Primärer Gegenstand ist der Programmtext
- Maßnahmen
  - Review/Code-Inspektionen/Walkthrough
  - Komplexitätsanalysen
  - Strukturanalysen
  - Datenflußanalysen

10.2 Dynamische Testverfahren

- SW wird mit Testdaten ausgeführt
- Ausführung in realer Umgebung
- Stichprobenverfahren

10.3 Unterscheidung Testfälle - Testdaten

Testfall: Menge von Eingabedaten und zu erwartenden Resultaten Testdaten: Teilmenge der Eingabedaten der Testfälle

10.4 Kontrollflussbezogene Verfahren

- Darstellung des Programms Als Kontrollflussgraphen
  - Anweisungen als Knoten
  - Kontrollfluß als gerichtete Kanten zwischen Knoten
  - Zweig: Einheit aus Kante und den dadurch verbunden Knoten
  - Pfad: Sequenz von Knoten und Kante (Start bis Endknoten)
- Anweisungsüberdeckung:
  Alle Anweisungen werden mindestens einmal ausgeführt
- Zweigüberdeckung:
  Alle Verzweigungen im Kontrollfluß werden mindestens einmal verfolgt
- Bedingungsüberdeckung:
  Alle booleschen Wertekonstellationen von (Teil-) Bedingungen werden einmal berücksichtigt
  - Einfache Bedingungsüberdeckung
  - Mehrfache Bedingungsüberdeckung
  - Minimale Mehrfachüberdeckung
- Pfadüberdeckung:
  Durchlaufen aller Pfade von Start- zum Endknoten
  - Boundary-interiour-Pfadtest
  - Boundary: Alle Testfälle, die den Schleifenrumpf genau einmal ausführen
  - Interior: In jedem Schleifendurchlauf ein anderer Pfad durch den Schleifenrumpf

➠Überschaubare Anzahl der Testfälle

10.5 Datenflussbezogene Verfahren

- Erweiterung der Kontrollflussgraphen um Datenflüsse
- Unterscheidung verschiedener Situationen hinsichtlich des Zugriffs auf Variablen im Programm:
  - Definition von Variablenwerten (Wertzuweisung)
  - Berechnende Benutzung (zur Ermittlung eines Wertes in Ausdrücken)
  - Prädikative Benutzung (Auswertung der Bedingungen in bedingten Anweisungen oder Schleifen)
- Prinzip des Verfahrens:
  - Für jeden Definitionsknoten werden definitionsfreie Pfade zu allen Benutzungsknoten ermittelt (all-uses-Kriterium)
  - Die Auswahl der Testdaten muß das Durchlaufen jedes identifizierten definitionsfreien Pfades sicherstellen
- Drei Funktionen als Basis des Verfahrens:
  - def: bildet jeden Knoten auf die Menge der darin global def. Var. ab
  - c-use: ordnet jedem Knoten die in ihm berechnend benutzten Var. zu
  - p-use: weist jeder Kante die Menge von Variablen zu, deren prädikative Auswertung zum Durchlaufen der Kante notwendig ist.

10.6 Funktionale Verfahren

- Überprüfung der in der Spezifikation festgelegten Funktionalität
- Programmstruktur irrelevant
- Beste Grundlage: formale Spezifikation
- Bestimmung der Testdaten:
  - Bildung funktionaler Äquivalenzklassen: Eingabe- und Ausgabemengen, die jeweils zu einer (Teil-) Funktionalität gehören
  - Alle Werte einer Äquivalenzklasse verursachen ein identisches funktionales Verhalten eines Programms
  - Auswahl von Testdaten aus den Äquivalenzklassen:
    - Zufällig
    - Test spezieller Werte (z.B. 0, nil)
    - Grenzwertanalyse (primär Grenzbereiche der Eingabemengen als Testdaten)

11 Vorgehensweise / Phasenmodelle

- Legen logische Ordnung und zeitliche Reihenfolge der Aktivitäten fest
- Geben Software-Projekten eine Struktur
- Umsetzung Vorgehensmodelle ist Aufgabe Projektmanagement

11.1 Klassisches Phasenmodell

Zerlegung der Software-Entwicklung in zeitlich aufeinanderfolgende Phasen:

- Problemanalyse u. Planung
- Systemspezifikation
- Entwurf
- Implemwntierung
- Integration u. Test
- Einsatz u. Wartung

Vorteile:

- Klare Strukturierung der Vorgehensweise
- Erleichtert die Planung und das Management der Software-Entwicklung

Nachteile:

- Strenge Trennung der Phasen ist eine unzutreffende Idealisierung, Rücksprünge sind meist notwendig
- Ergebnisse im Sinne des Zielsystems liegen erst sehr spät vor, Validierung kann erst sehr spät erfolgen

11.2 V-Modell

- Unterscheidungen von Ebenen der Software-Entwicklung hinsichtlich der Abstraktion bzw. Konkretisierung sowie Detaillierung/Aggregation
- Querbezüge der Phasen auf einer Ebene

11.3 Wasserfall-Modell

- Explizite Einführung von Rücksprungmöglichkeiten zur unmittelbar vorherigen Phase
- Explizite Berücksichtigung der Validierung der Ergebnisse jeder Phase
- Beginn der nächsten Phase erst nach erfolgreicher Validierung im Hinblick auf Vorphase
- Rücksprünge zu vorherigen Phasen bei Mängeln in den Ergebnissen der Vorphase (z.B. lückenhafte Spezifikationen)

Vorzüge

- Streng sequentielle Vorgehensweise wird aufgelockert
- Inkrementelle Entwicklungsstrategie mit schrittweiser Absicherung wird gefördert
- Bessere Kontrollmöglichkeiten im Entwicklungsprozeß
- Bessere Qualität des Software-Produktes
- Reduzierung der Auswirkungen von Fehlentscheidungen in vorheriger Phase
- Begrenzung des Aufwandes durch eingeschränkte Rücksprungmöglichkeiten

11.4 Iterative Vorgehensweise

- Beliebige Wiederholbarkeit von Phasen
- Rücksprünge über mehrere Phasen hinweg
- Problem: Bei „weiten“ Rücksprüngen großer Aufwand

11.5 Prototyping-orientierte Vorgehensweise

- Prinzip: Software so früh wie möglich
- Ausgangspunkt: Prototyp auf der Basis der Spezifikation zu deren Prüfung
- Ausweitung der prototyping-orientierten Vorgehensweise auf andere Phasen des Entwicklungsprozesses.
- Unterscheidung von Prototypen
  - Wegwerf Prototypen
  - Wiederverwenbare
- Abhängig von der Art des Prototypen hohes Maß an Werkzeugunterstützung notwendig

- Keine Einschränkung hinsichtlich des inhaltlichen Anwendungsbereiches
- Besonders geeignet bei Unklarheiten bezüglich der Funktionalität und ihrer Realisierbarkeit
- Verbesserte Kommunikation zwischen Entwickler und (zukünftigem) Anwender,
- Einbeziehung der Benutzer in unterschiedlichem Ausmaß (Benutzer:Entwickler):
  - bei Spezifikationsprototyping ca. 60:40
  - bei Architekturprototyping ca. 20:80

Vorteile:

- Bessere Software-Qualität
- Verbesserte Kommunikation zwischen Entwicklern und (zukünftigen) Anwendern
- Höhere Motivation und verbesserte Einbindung der Anwender
- Verbesserte Akzeptanz durch die Anwender
- Verringerung der Gesamtkosten im Life-Cycle

Nachteile:

- Höherer Entwicklungsaufwand (gewisser Ausgleich durch geringere Wartung)
- Unstrukturierter Entwicklungsverlauf („Prototyp als System“)
- Prototyp anstelle einer Anforderungsdefinition (Spezifikation)
- Schwierigkeiten der organisatorischen Einbettung
- Hohe Anforderungen an technische Unterstützung (Entwicklungswerkzeuge, Sprachen für das Rapid Prototyping)