Testen von Programmen

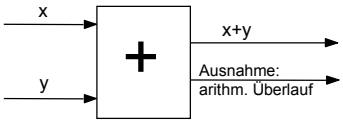
- Definitionen
- Black-box Testen
- White-box Testen
- Test-Prinzipien

Testen - Definition

- **Testen ist der Prozeß, ein Programm mit der Absicht auszuführen, Fehler zu finden. (Myers 1979)**
  - Wurde ein Programm sorgfältig getestet (und sind alle gefundenen Fehler korrigiert), so ***steigt die Wahrscheinlichkeit***, daß das Programm sich auch in den nicht getesteten Fällen wunschgemäß verhält.
- **Ziel des Tests ist es,**
  - Fehler zu entdecken!
- **Ein Test ist erfolgreich,**
  - wenn er einen Fehler gefunden hat.

Testen - Ziele

- **Ein erfolgloser Test ist niemals ein Beweis für ein korrektes Programm – es wurden lediglich keine Fehler gefunden!**
  - Die Korrektheit eines Programms kann durch Testen (außer in trivialen Fällen) nicht bewiesen werden.
- **Grund: alle Kombinationen aller möglichen Werte der Eingabedaten müßten getestet werden**
  - Anzahl möglicher Eingaben: $2^{16} \cdot 2^{16} = 2^{32}$
  - Ein vollständiger Test erfordert mehr als 4'000'000'000 Testfälle
  - Ein Programm kann niemals “ausgetestet” werden!

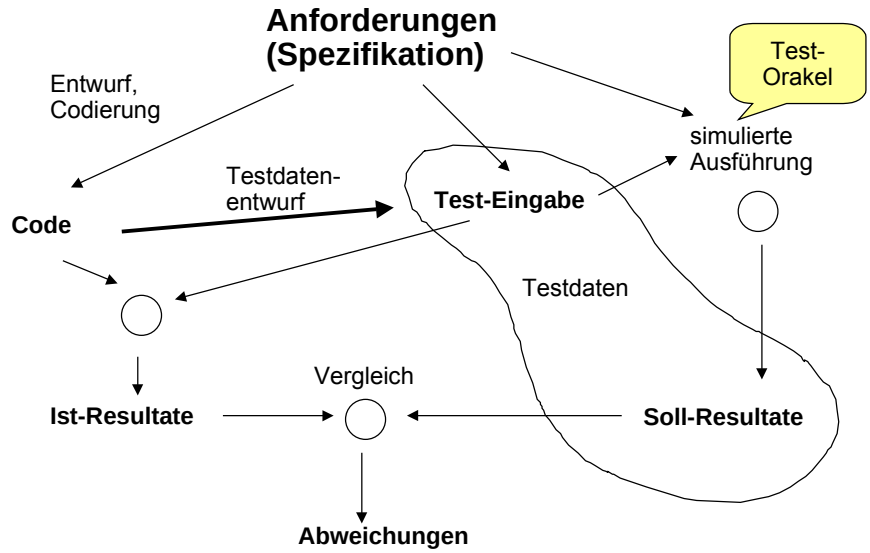


x, y ganze Zahlen, mit 16 Bit repräsentiert

Testfälle

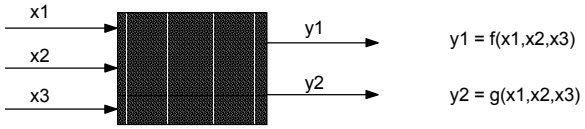
- **Auswahl der Testfälle ist die zentrale Aufgabe des Testens!**
  - Anforderungen an Testfälle
    - repräsentativ
    - fehlersensitiv
    - redundanzarm
    - ökonomisch
- **Ziel:**
  - Mit einer möglichst kleinen Auswahl der Testfälle möglichst vielen Fehlern auf die Spur kommen.
- **Auswahl entsprechend der (angestrebten Merkmale des Programms (Basis: Spezifikation)**
  - Black-Box-Test (Funktionstest)
- **Auswahl unter Einfluß der inneren Struktur des Prüflings**
  - Glass-Box-Test (Strukturtest)

Vorgehensweise beim Test

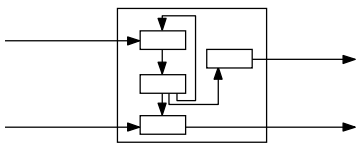


Testverfahren

- **funktionsorientiert (Black-Box-Test)**
  - Testfall-Auswahl aufgrund der Spezifikation
  - Programmstruktur kann unbekannt sein



- **strukturorientiert (White-Box-Test, Glass-Box-Test)**
  - Testfall-Auswahl aufgrund der Programmstruktur
  - Spezifikation muß ebenfalls bekannt sein (wegen der erwarteten Resultate)



Black-box Test (funktionaler Test)

- **Ausgangspunkt für Testfälle ist die Spezifikation**
  - Fehlt eine Spezifikation, dann hat das erhebliche Konsequenzen auch für das Testen
  - Wogegen soll getestet werden? Was sind die Sollergebnisse?
- **Motivation**
  - Es ist nicht zulässig, ein Programm lediglich gegen sich selbst zu testen.
- **Ziel**
  - Möglichst umfassende Prüfung der ***spezifizierten Funktionalität***.
- **Nachteile**
  - Die konkrete Implementierung wird ***nicht geeignet*** berücksichtigt
  - Häufig wird mit einem reinen Black-box Test nicht die Minimalanforderung von White-box-Tests erzielt
    - 100 % Zweigüberdeckung

Black-box - Testfallauswahlkriterien

- **Funktionsüberdeckung**
  - Jede spezifizierte Funktion wird mindestens einmal ausgeführt.
- **Eingabeüberdeckung**
  - Jedes Eingabedatum wird in mindestens einem Testfall verwendet.
  - Macht in der Regel Probleme!
- **Ausgabeüberdeckung**
  - Jede “Ausgabesituation” wird mindestens einmal erzeugt.
  - Beispiele: Bildschirmmasken, Fehlermeldungen, etc.
- **Aufwand dafür ist z.T. erheblich**
  - Bspl: Textverarbeitungsprogramm
  - Häufig müssen Funktionen in ihrem Zusammenspiel geprüft werden.
  - Zusätzlich müssen Leistungs- und Robustheitsprüfungen gemacht werden

Techniken der Testfall-Auswahl

- **Äquivalenzklassenbildung**
  - Um eine repräsentative Menge von Eingabedaten zu testen, werden die Eingaben in Äquivalenzklassen eingeteilt. Aus jeder Klasse wird ein Repräsentant getestet.
- **Grenzwertüberprüfung**
  - Da an den Grenzen zulässiger Datenbereiche erfahrungsgemäß häufig Fehler auftreten, werden außerdem auch Testfälle für solche Grenzfälle definiert.

Äquivalenzklassenbildung

- **Prinzip**
  - Zerlegung aller möglicher Eingabedaten in
    - gültige und
    - ungültige Äquivalenzklassen
  - Jede Äquivalenzklasse wird durch einen (oder mehrere) Repräsentanten getestet.
  - Werte aus einer Äquivalenzklasse verursachen ein identisches funktionales Verhalten.
  - Somit können alle Programmfunktionen getestet werden.
  - Die Anzahl der Testfälle wird reduziert.
  - Äquivalenz ist hypothetisch
    - Klassen werden z.T. nach Erfahrung und Intuition gebildet
- **Zerlegung mit Hilfe von**
  - spezifizierten Gültigkeitsbereichen
  - spezifizierten (oder vermuteten) Sonderbehandlungen

Vorgehensweise

- **Schritt 1:**
  - Aufstellung von Eingabebedingungen
- **Schritt 2:**
  - Bildung von Äquivalenzklassen für jede Eingabebedingung.
- **Schritt 3:**
  - Äquivalenzklassen nummerieren.
- **Schritt 4:**
  - Testfälle definieren (gültige Ä-Klassen)
- **Schritt 5:**
  - Testfälle definieren (ungültige Ä-Klassen)

Beispiel: Ä-Klassenbildung - 1

- **Spezifikation eines PRINT-Befehls**
  - Mit dem Ausgabebefehl PRINT wird die Datei auf den Bildschirm ausgegeben.
  - Der Befehl hat zwei Parameter: Dateiname und Zeilenanzahl.
  - Beide Parameter müssen angegeben werden.
  - PRINT hat folgende Syntax: PRINT <Dateiname> <Zeilenanzahl>
  - Der Dateiname besteht aus mindestens einem und bis zu sechs Zeichen, die Buchstaben oder Ziffern sein können.
  - Das erste Zeichen des Dateinamens muß ein Buchstabe sein.
  - Die Zeilenanzahl besteht aus mindestens einer und bis zu 3 Ziffern. Sie muß größer 0 und kleiner 1000 sein.

Beispiel: Ä-Klassenbildung - 2

- **Schritt 1: Eingabebedingungen definieren**
- **Schritte 2 und 3: Äquivalenzklassen finden**

Eingabebedingungen	gültige Ä-Klasse	ungültige Ä-Klasse
Anzahl der Parameter	zwei (1)	keine(1a), einen(1b) mehr als 2 (1c)
Dateiname (Länge)	1 bis 6 (2)	0 (2a), >6(2b)
Dateiname (Zeichen)	Buchst. oder Ziffern (3)	sonstige Zeichen(3a)
Dateiname (erstes Zeichen)	Buchstabe(4)	kein Buchstabe(4a)
Zeilenanzahl(Zeichen)	Ziffern (5)	ein Zeichen ist keine Ziffer(5a)
Zeilenanzahl (Ziffern)	1 bis 3 (6)	>3 (6a), 0 (6b = 1b)
Zeilenanzahl (Größe)	1 <= w <= 999 (7)	w <=0 (7a), w >=1000(7b)

Diskussion Ä-Klassenbildung

- **Feststellung**
  - Bei der Äquivalenzklassenmethode ist die ***Güte*** der Testfälle abhängig von der ***Aussagekraft*** der Spezifikation
  - Bspl:
    - Führende Nullen beim zweiten Parameter des PRINT Befehls
    - Die Spezifikation macht keine Aussagen dazu:
      - Es entstehen zwei ungültige Ä-Klassen (mehr als 3 Zeichen, Wert >= 1000)
- **Die Definition der Testfälle hängt nicht nur von den Eingabebedingungen ab**
  - Welche ***Werte*** aus einer Ä-Klasse sollen gewählt werden?
  - Welche Kombinationen von ***Eingabebedingungen*** sollen getestet werden?

Grenzwertanalyse -1

- **Wahl der Repräsentanten einer Ä-Klasse**
  - Besteht eine Ä-Klasse aus einer ***geordneten Menge von Werten***, dann kann zur Auswahl von Repräsentanten die Grenzwertanalyse durchgeführt werden.
- **Schritt 1 der Grenzwertanalyse**
  - Testdaten identifizieren, die direkt auf oder neben den Grenzen des Ä-Klasse liegen (plus einen mittleren Wert).
    - Richtlinien
      - Wertebereich :
        - gültige Testwerte: kleinster und größter Wert
        - ungültige Werte: Werte, die direkt daneben und außerhalb liegen
      - Beispiel
        - Natürliche Zahlen 1 .. 255
        - 4 Repräsentanten: 0, 1, 255, 256
        - Liste maximal 100 Elemente
        - 4 Repräsentanten der Längen: 0, 1, 100, 101

Grenzwertanalyse -2

- **Schritt 2 der Grenzwertanalyse**
  - Zusätzlich zu den Eingabe-Ä-Klassen werden ***Ausgabe-Ä-Klassen*** für die erwarteten Resultate gebildet.
  - Dies geschieht ***analog*** zur Vorgehensweise bei der Definition der Eingabe-Ä-Klassen.
  - Ausgabe-Ä-Klassen stellen ***Soll-Werte*** dar, für die die Eingabedaten bestimmt werden müssen.
  - Hinweis:
    - Es ist manchmal nicht möglich, diese Eingabedaten zu finden, da eine zu produzierende ungültige Ausgabe nicht vom Programm zugelassen wird.

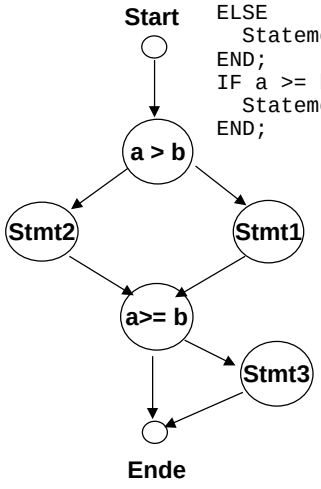
White-box Tests

- **Auswahl der Testfälle so, daß der Programmablauf oder der Datenfluß im Programm überprüft wird.**
  - Meistens wird der Programmablauf getestet: Testfälle werden so gewählt, daß das Programm systematisch durchlaufen wird.
- **Gebräuchlich sind drei sogenannte Testüberdeckungen:**
  - Anweisungsüberdeckung
  - Zweigüberdeckung
  - Pfadüberdeckung
- **Nachteil**
  - Fehlende, in der Spezifikation beschriebene Funktionalitäten, werden nicht erkannt.
- **Vorteil**
  - Es lassen sich formale Testauswahlkriterien für wb-Tests definieren.

Ablaufgraph

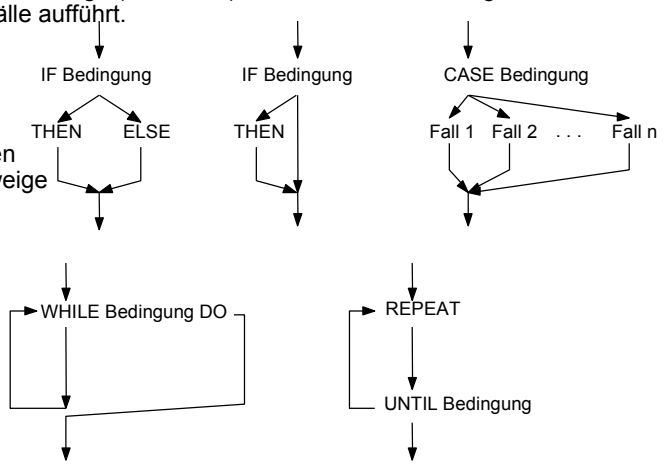
- **Programm wird als Flußdiagramm betrachtet:**
  - Anweisung: Knoten
  - Zweig: Kante nach Bedingung
  - Pfad: Weg vom Anfangs- bis zum Endknoten
- **Überdeckungen können nur rechnerunterstützt ermittelt werden:**
  - Ein Testinstrumentierer fügt Zähler in Knoten und Kanten ein
  - Der Überdeckungsgrad wird über mehrere Läufe kumuliert

```
IF a > b THEN
  Statement1;
ELSE
  Statement2;
END;
IF a >= b THEN
  Statement3;
END;
```



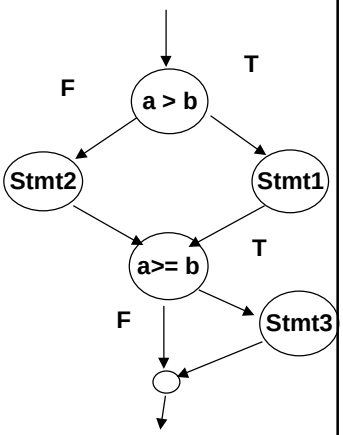
Zweige / Pfade im Programm

- **Bestimmung der Programmzweige:**
  - Betrachtung von Verzweigungen und Schleifen. Bei Programmiersprachen mit geschlossenen Ablaufkonstrukten (z.B. Pascal) haben jede IF-Anweisung und jede Schleife je zwei Zweige (siehe Bild). Eine CASE-Anweisung hat soviele Zweige, wie sie Fälle aufführt.
- **Bestimmung der Pfade (Wege):**
  - Alle Kombinationen aller Programmzweige bei maximalem Durchlauf aller Schleifen



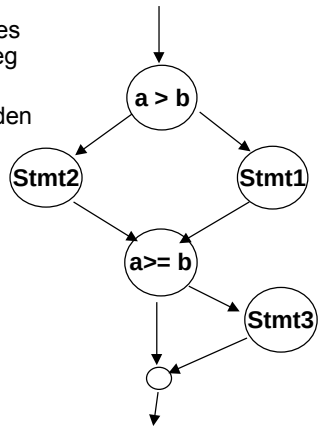
Anweisungsüberdeckung

- **Co-Test ist eine einfache kontrollflußorientierte Testmethode**
- **Definititon**
  - Eine Testfallmenge T erfüllt das Co-Kriterium, wenn es für jede Anweisung A des Programms P einen Testfall gibt, der die Anweisung A ausführt.
- **Überdeckungsgrad** $\frac{\text{ausgeführte Statements}}{\text{Anzahl aller Statements}}$
- **Black-Box-Testfälle produzieren ca. 60-70% Anweisungsüberdeckung.**
- **Angestrebt werden 95-100%**
- **Bewertung**
  - notwendiges aber schwaches Kriterium
  - hilft dead-code zu finden
  - gewisse Kontrollflußfehler werden nicht (immer) gefunden



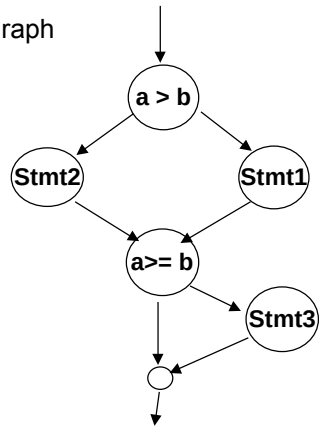
Zweigüberdeckung

- **C1-Test ist eine dynamische kontrollflußorientierte Testmethode**
- **Definition**
  - Eine Testfallmenge T erfüllt das C1-Kriterium, wenn es für jede Kante k im Kontrollflußgraph von P einen Weg in Wege(T,P) gibt, zu dem k gehört.
  - D.h. wenn alle Entscheidungskanten ausgeführt werden
- **Überdeckungsgrad** $\frac{\text{ausgeführte Kanten}}{\text{Anzahl aller Kanten}}$
- **Black-Box-Testfälle produzieren ca. 80% Zweigüberdeckung.**
  - Angestrebt werden 100%
- **Bewertung**
  - minimales Testkriterium
  - hilft dead-code zu finden
  - berücksichtigt nicht komplexe Bedingungen



Pfadüberdeckung

- **C2-Test ist die intensivste kontrollflußorientierte Testmethode**
- **Ziel**
  - Alle unterschiedlichen Pfade sollen einmal ausgeführt werden.
  - Pfad: Sequenz von Knoten im Ablaufgraph
- **Problem**
  - Pfadüberdeckung ist für reale Programme unrealistisch, da die Anzahl der Pfade durch Schleifen astronomisch hoch sein kann.



Eingeschränkte Pfadüberdeckung

- **Problem**
  - Schleifen führen zu sehr vielen/unbekannt vielen Pfaden
  - Dieser Überdeckungsgrad wird nur bei sicherheitskritischer Software angestrebt.
- **Idee**
  - Nicht alle Pfade durch Schleifen werden berücksichtigt.
  - Nur die Pfade, die neue Aspekte ansprechen.
- **Beschränkung auf 5 Aspekte pro Schleife:**
  - 0 Iterationen: die Schleife wird nicht betreten
  - 1 Iteration: zeigt häufig Initialisierungsfehler
  - 2 Iterationen: kann ebenfalls Initialisierungsfehler zeigen
  - typ. Anzahl Iterationen: Normalfall muß auch geprüft werden
  - max. Iterationen: zeigt Fehler beim Abbruchkriterium

Beispiel: eing. Pfadüberdeckung

```
:
Kandidat := 1;
WHILE (Kandidat < Anzahl) AND (Elemente [Kandidat] <> Suchwert) DO
   Kandidat := Kandidat + 1;
END;
```

- **0 Iterationen:**
- **1 Iteration:**
- **2 Iterationen:**
- **typ. Anzahl Iterationen:**
- **max. Iterationen:**

Testen - Sollergebnis

- **Testen setzt voraus, daß die erwarteten Ergebnisse bekannt sind**
  - Entweder muß gegen eine ***Spezifikation*** oder gegen vorhandene Testergebnisse
  - (z.B. bei der Wiederholung von Tests nach Programm-Modifikationen) getestet werden (sogenannter Regressionstest)
- **Unvorbereitete und undokumentierte Tests sind sinnlos!**
- **Mit Testen können nicht alle Eigenschaften eines Programms geprüft werden**
  - Wartbarkeit, etc.
- **Mit Testen werden nur Fehlersymptome,**
  - nicht aber die ***Fehlerursachen*** gefunden!

Prinzipien des Testens

- **Vollständiges Testen ist unmöglich!**
- **Zu jedem Testfall gehört ein Soll-Resultat**
  - Nur ein auffällig falsches Resultat springt ins Auge!
- **Teste niemals dein eigenes Programm**
  - Kein Programmierer will zeigen, daß sein Progamm Fehler hat
- **Testfälle müssen auch für ungültige und unerwartete Eingaben definiert werden!**
- **Vermeide Wegwerftestfälle, es sei denn das Programm ist wirklich ein Wegwerfprogramm.**
- **Ein erfolgreicher Testfall ist dadurch gekennzeichnet, daß er einen bisher unbekannten Fehler entdeckt.**
- **Ein Test ist nur so gut wie seine Testfälle!**

Vor- und Nachteile von Tests

- **Vorteile**
  - Testen ist ein ***natürliches*** Prüfverfahren
  - Tests sind ***reproduzierbar*** (=> objektiv)
  - investierter Aufwand ***mehrfach*** nutzbar
  - ***Zielumgebung*** wird mitgeprüft
  - Systemverhalten wird sichtbar gemacht
- **Nachteile**
  - Ergebnisse werden ***überschätzt*** (Korrektheitsaussage unmöglich)
    - Tests sind Stichprobenverfahren
  - ***nicht alle*** Programm-Eigenschaften sind testbar
  - nicht alle ***Anwendungssituationen*** sind nachbildbar
  - Test zeigt die Fehlerursache ***nicht***