Software-Qualitätsmanagement

Kernfach Angewandte Informatik

Sommersemester 2007

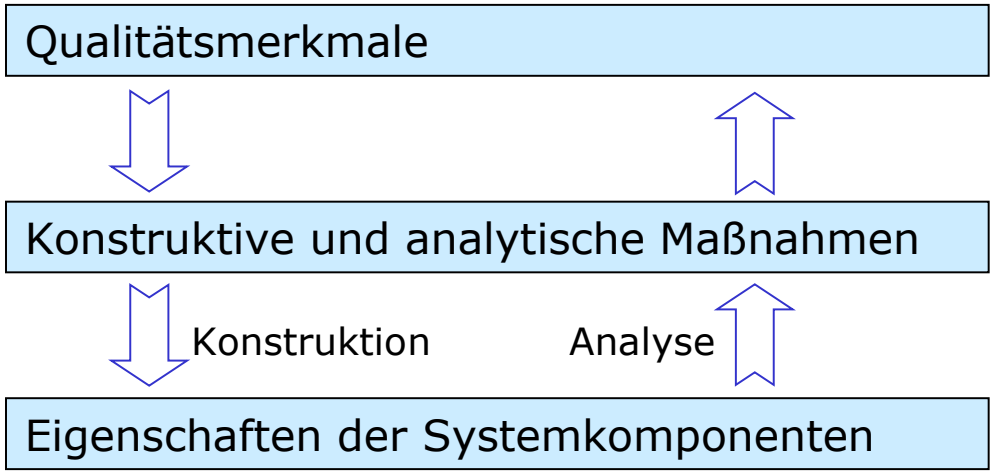
Prof. Dr. Hans-Gert Gräbe

http://bis.informatik.uni-leipzig.de/HansGertGraebe

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

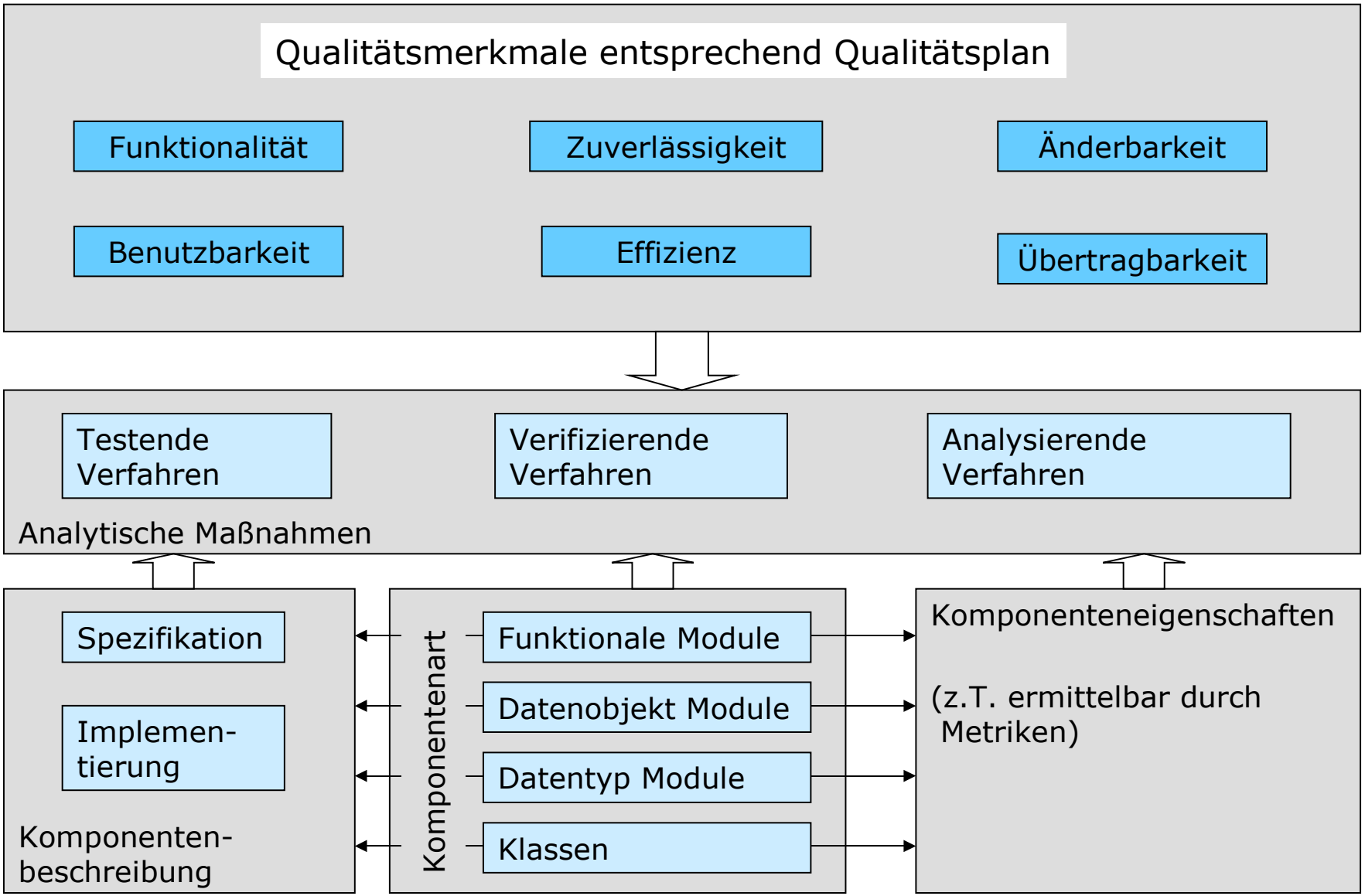
Software-Produktqualität abhängig von:

- Qualität der Systemkomponenten
- Qualität der Beziehungen zwischen den Komponenten



- Konstruktive Maßnahmen siehe VL „Software-Technik“
- analytische Maßnahmen beziehen sich im Wesentlichen auf Funktionalität, Zuverlässigkeit und evtl. Änderbarkeit

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Fehler als zentraler Begriff

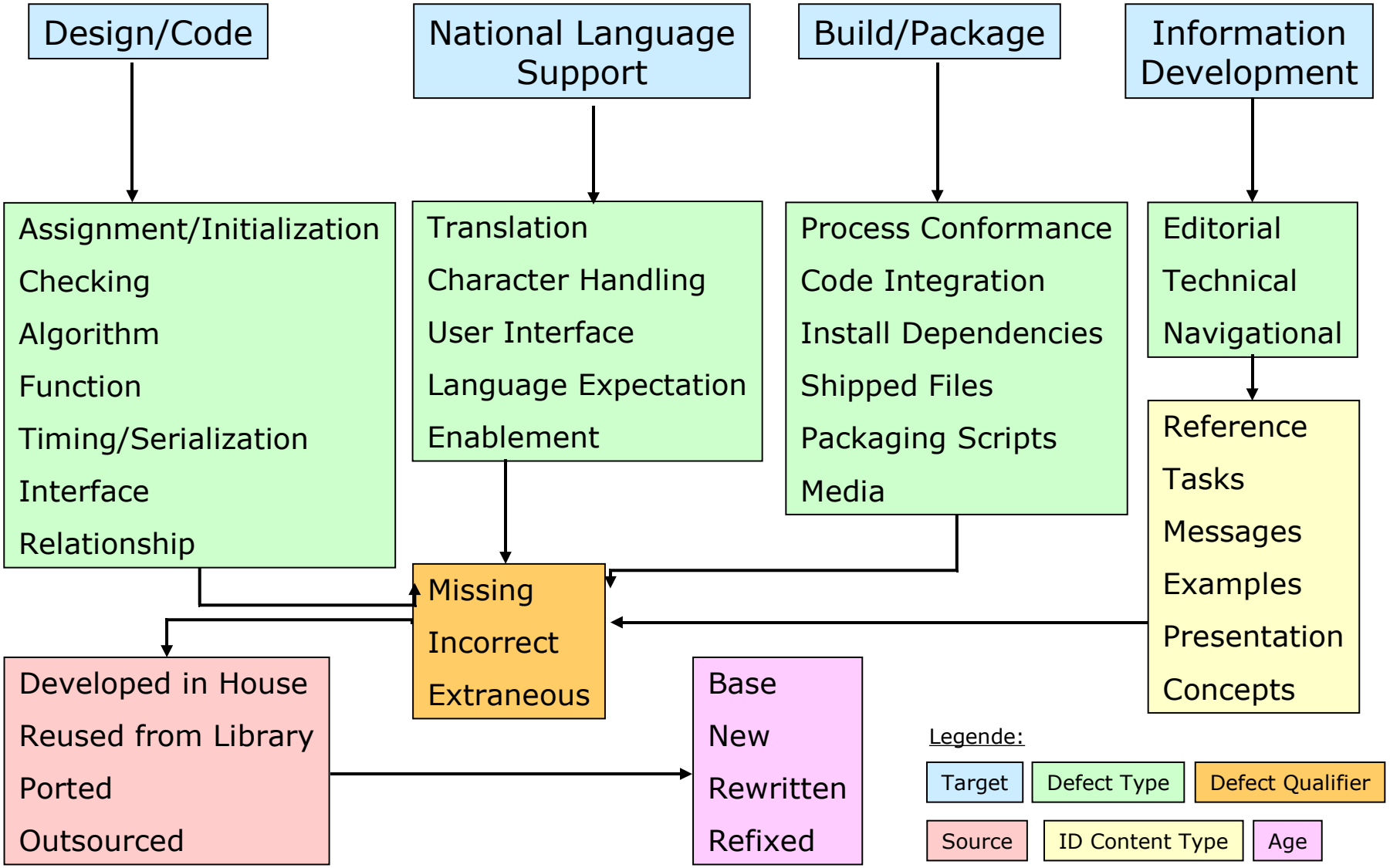
- Konstruktives Ziel: Entwicklung fehlerfreier Software-Komponenten
- Analytisches Ziel: Nachweis der Fehlerfreiheit von Software-Komponenten

Als **Fehler** wird
- jede Abweichung der tatsächlichen Ausprägung eines Qualitätsmerkmals von der vorgesehenen Soll-Ausprägung,
- jede Inkonsistenz zwischen der Spezifikation und der Implementierung und
- jedes strukturelle Merkmal des Programmtexts, das ein fehlerhaftes Verhalten des Programms verursacht,

bezeichnet.

2. Was sind Fehler?

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Analyseverfahren

Die Art und die Eigenschaften der Systemkomponenten bestimmen die Auswahl geeigneter analytischer Maßnahmen:

- Funktionale Module
- Datenobjekt-Module
- Datentyp-Module
- Klassen

Unterschied der eingesetzten analytischen Maßnahmen insbesondere zwischen Komponenten mit komplexen Kontrollstrukturen und Komponenten mit komplexen Datenstrukturen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **Testende Verfahren** haben das Ziel, Fehler zu erkennen
  - Dynamische Testverfahren
  - Statische Verfahren
- **Verifizierende Verfahren** sollen die Korrektheit einer Komponente beweisen
  - Verifikation
  - Symbolische Ausführung
- **Analysierende Verfahren** sollen Eigenschaften von Komponenten darstellen oder vermessen
  - Analyse der Bindungsart
  - Metriken
  - Grafiken und Tabellen
  - Anomalienanalyse

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Testende Verfahren - Prinzipieller Zugang

- Das tatsächliche Verhalten wird **stichprobenartig** an Hand einer Menge von **Testfällen** untersucht und die Ergebnisse mit den erwarteten Ergebnissen (Spezifikation, Normen) verglichen und dokumentiert.
- Testfälle werden entsprechend den Testzielen speziell ausgewählt.
- Einsatz um
  - Programmfehler aufzufinden (Bugs)
  - Wiederauftreten von Fehlern zu vermeiden (Regressionstests)
- Fehlerfreiheit ist plausibel, aber nicht garantiert.
- Abzugrenzen von
  - **Verifikation** als strengem Korrektheitsbeweis
  - **Ausprobieren** als einer Entwicklungsmethode (trial and error)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Begriff des Programms

- Programm = schrittweise Transformation einer Menge von Eingabedaten in eine Menge von Ausgabedaten nach einem vorgegebenen Algorithmus
- Black-Box-Betrachtung: f: X -> Y

  Spezifikation, funktionale Korrektheit
- Transformation = Abarbeiten einzelner Programmschritte, in denen die Daten entsprechend den angegebenen Instruktionen verändert werden.

  zustandsorientierte Betrachtung: Datenfluss

  übergangsorientierte Betrachtung: Kontrollfluss
- Programmstatus = Zustand der Gesamtheit der durch das Programm manipulierten Daten

  Anweisungen und Deklarationen

  Variablenbegriff als Wertcontainer
  - Sichtbarkeit und Lebensdauer
  - Compilezeit und Laufzeit

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Klassifikation testender Verfahren

- Dynamische Testverfahren
  - übersetztes und ausführbares Programm wird mit konkreten Eingabewerten ausgeführt
  - evtl. Instrumentierung des Programms
  - Test in realer Laufzeitumgebung
  - Stichprobenverfahren (Testfälle)
  - Ziele: Finden von Fehlern (debugging), Finden und Optimieren laufzeitkritischer Bereiche (profiling)
  - Korrektheit kann so nicht bewiesen werden
  - Klassifikation nach Herkunft der Testfälle
- Statische Testverfahren
  - Analyse des Quellcodes, evtl. testfallorientiertes Durchgehen
  - typische Verfahren: manuelle Prüfmethoden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Begriffe

Prüfling, **Testling** oder **Testobjekt**

- das zu testende Programm oder die Komponente (Prüfling als allgemeiner Begriff)

Testverfahren

- grundlegendes Verfahren, mit dem einzelne Eigenschaften eines Testlings durch eine geeignete Anzahl von Testfällen untersucht wird

Testfall

- Satz von Testdaten, der die vollständige Ausführung eines zu testenden Programms bewirkt

Testdatum

- Eingabewert, der einen Eingabeparameter des Testobjekts instanziiert

Testtreiber

- Testrahmen, mit dem eine nicht extern zugängliche Funktion interaktiv aufgerufen werden kann.
- Können durch Testwerkzeuge automatisch generiert werden.

Instrumentierung

- Der Quellcode des Testlings wird für die Analyse der Testfälle mit zusätzlichem Protokollcode versehen oder dieser aktiviert.
- Instrumentierter Testling wird übersetzt. Protokoll enthält Informationen über das Laufzeitverhalten des Testlings während des Abarbeitens der einzelnen Testfälle.

Überdeckungsgrad

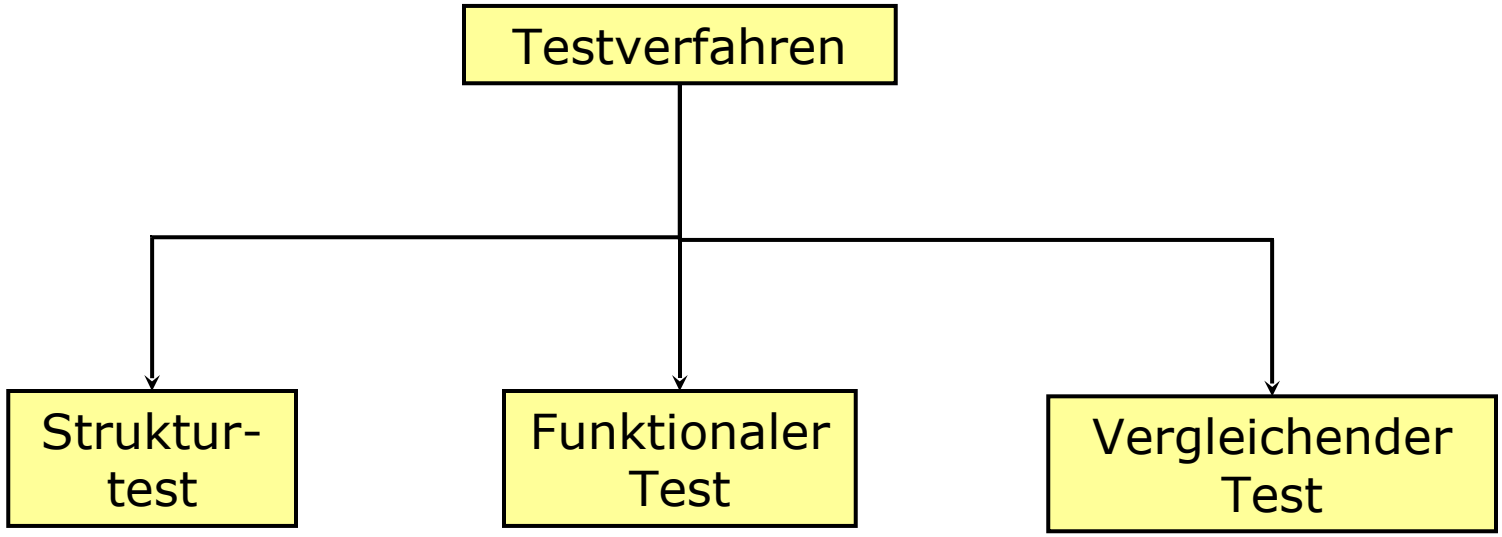
- Maß für den Grad der Vollständigkeit eines Tests bezogen auf ein bestimmtes Testverfahren

Regressionstest

- automatische werkzeuggestützte Neuausführung bereits durchlaufener Tests nach Änderungen am Testling.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Klassifikation testender Verfahren



kontrollfluss- oder datenflussorientiert

benötigt den Quellcode

Testwertauswahl nach funktionalen Gesichtspunkten

benötigt die Spezifikation

Vergleich der Ergebnisse verschiedener Programmversionen, etwa durch Einfügen künstlicher Fehler

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **Strukturtest**

  kontrollflussorientiert (Monitoring des Programmflusses)
  - Anweisungsüberdeckung
  - Zweigüberdeckung
  - Pfadüberdeckung (volle Version kombinatorisch exponentiell!)
  - Bedingungsüberdeckung

  datenflussorientiert (Monitoring der Programmdaten)

- **Funktionaler Test**

  funktionale Äquivalenz

  Grenzwertanalyse

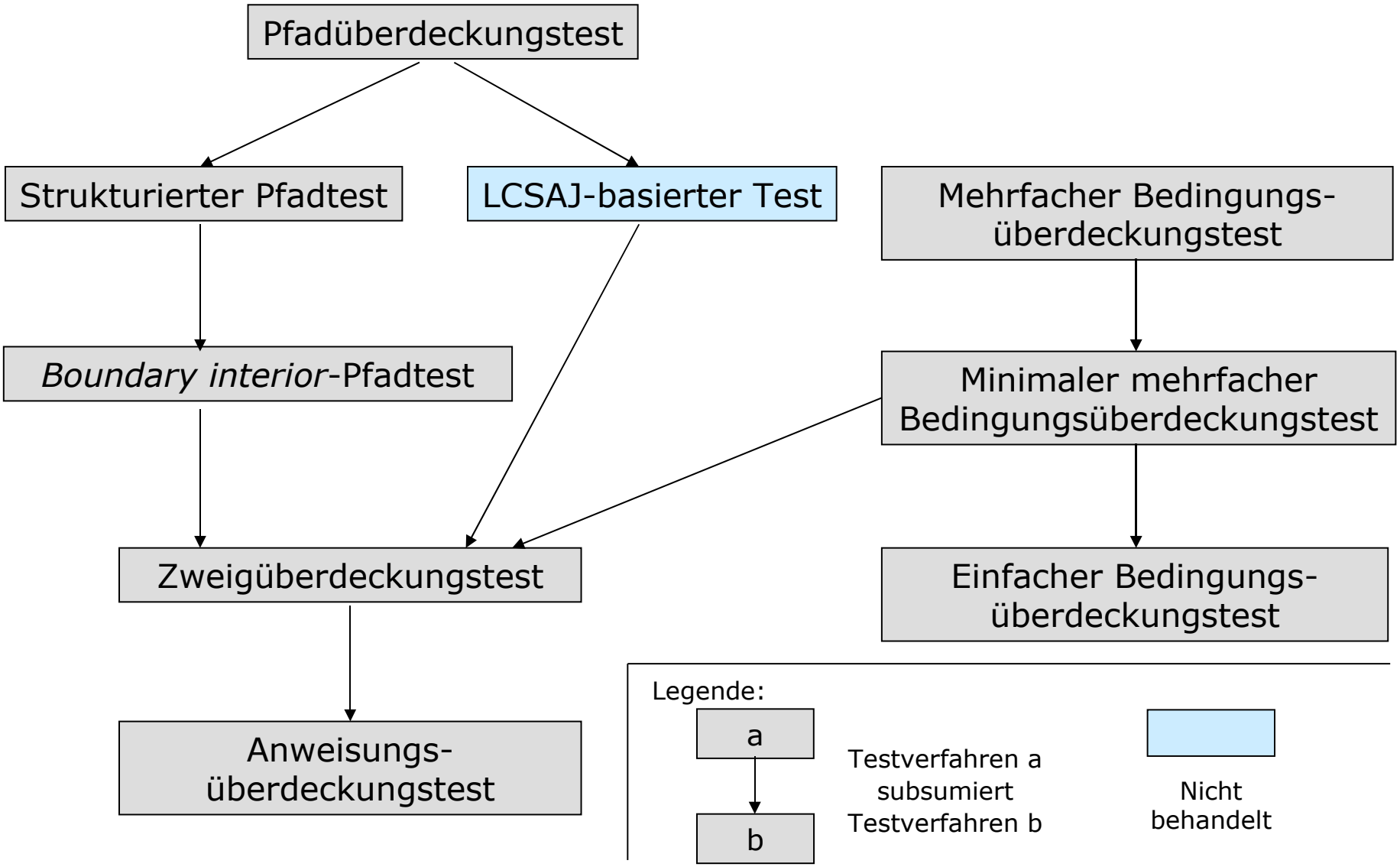
  Test spezieller Werte (Szenarios)

  Zufallstest

  zustandsübergangsgetriebene Tests

Überblick

- Basieren auf der Kontrollstruktur des zu prüfenden Programms
- Gehören zu den Strukturtest-, White Box- oder Glass Box-Verfahren
- Sind dynamische Testverfahren, d.h. das Programm wird ausgeführt
- **Ziel** ist, mit möglichst wenigen Testfällen alle Anweisungen, Zweige oder Pfade zu durchlaufen
- Sorgfältige Auswahl der Testfälle, um mögliche strukturelle Probleme genau zu überdecken.


Pfadüberdeckungstest
Strukturierter Pfadtest
LCSAJ-basierter Test
Mehrfacher Bedingungs-überdeckungstest
Boundary interior-Pfadtest
Minimaler mehrfacher Bedingungsüberdeckungstest
Zweigüberdeckungstest
Einfacher Bedingungs-überdeckungstest
Anweisungs-überdeckungstest
Legende:
a
b
Testverfahren a subsumiert Testverfahren b
Nicht behandelt

Ein Beispiel

/* **Programmname**: ZaehleZchn

Aufgabe: Die Prozedur ZaehleZchn liest solange Zeichen von der Tastatur, bis ein Zeichen erkannt wird, das kein Großbuchstabe ist, oder Gesamtzahl den größten durch den Datentyp int darstellbaren Wert INT_MAX erreicht.

Ist ein gelesenes ein Großbuchstabe zwischen A und Z, dann wird Gesamtzahl um eins erhöht. Ist der Großbuchstabe ein Vokal, dann wird auch VokalAnzahl um eins erhöht.

Ein-/Ausgabeparameter sind Gesamtzahl und VokalAnzahl.

Randbedingung: Das aufrufende Programm stellt sicher, dass Gesamtzahl stets größer oder gleich VokalAnzahl ist.

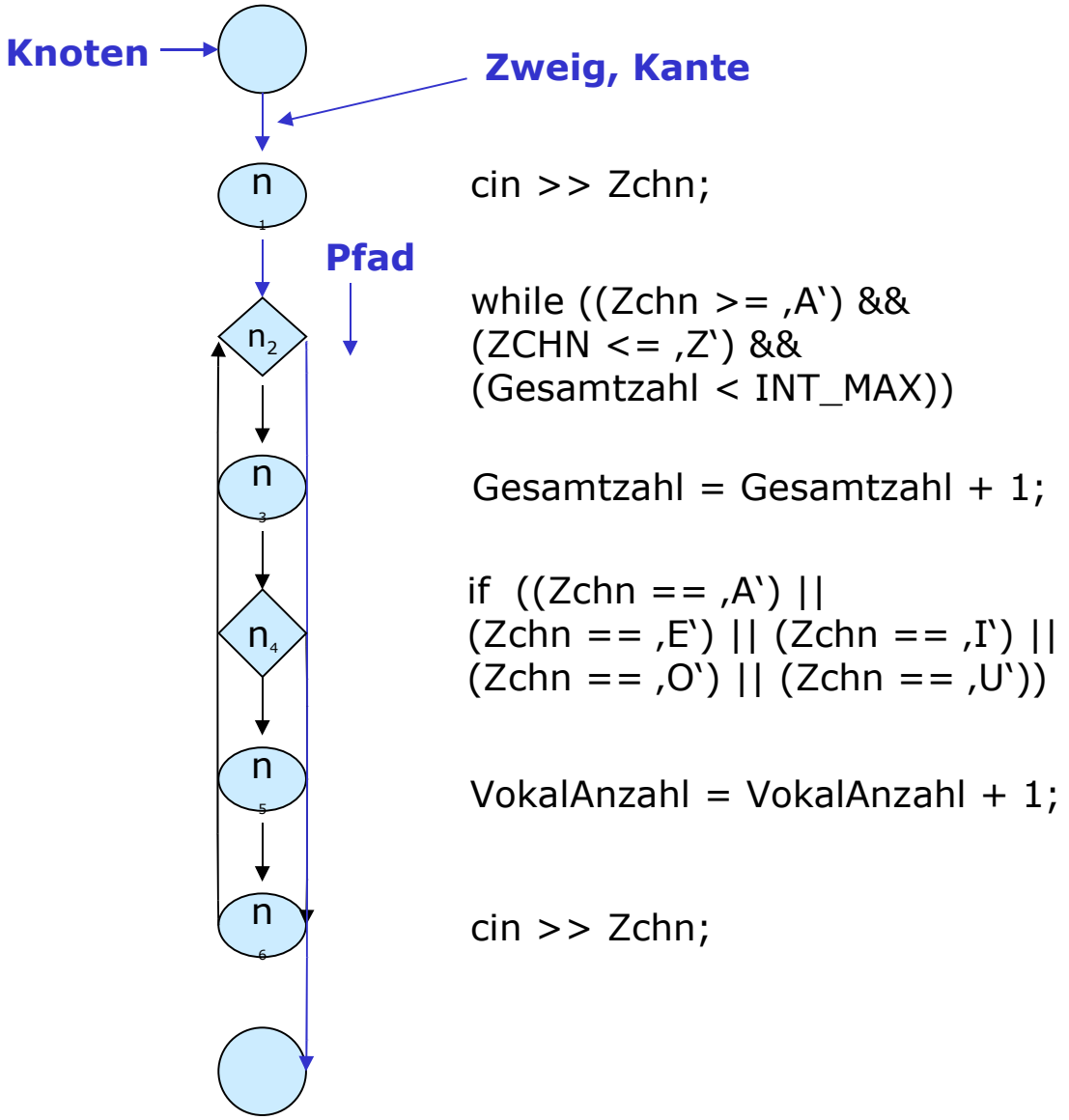
*/

```
void ZaehleZchn(int &VokalAnzahl, int &Gesamtzahl);

#include „ZaehleZchn.h“
#include <LIMITS.H>
#include <iostream.h>

void ZaehleZchn(int &VokalAnzahl, int &Gesamtzahl) {
    char Zchn;
    cin >> Zchn;
    while ((Zchn >= ‚A‘) && (ZCHN <= ‚Z‘) && (Gesamtzahl < INT_MAX)) {
        Gesamtzahl = Gesamtzahl + 1;
        if  ((Zchn == ‚A‘) || (Zchn == ‚E‘) || (Zchn == ‚I‘) ||
        (Zchn == ‚O‘) || (Zchn == ‚U‘)) {
            VokalAnzahl = VokalAnzahl + 1;
        }
        cin >> Zchn;
    }
}
```

```
void main()
{
    int AnzahlVokale = 0;
    int AnzahlZchn = 0;
    cout << „Programm ZaehleZchn“ << endl;
    cout << „Zeichen bitte eingeben:“ << endl;
    ZaehleZchn(AnzahlVokale, AnzahlZchn);
    cout << „Anzahl Vokale: “ << AnzahlVokale << endl;
    cout << „Anzahl Zeichen: “ << AnzahlZchn << endl;
}
```

Kontrollflussgraph

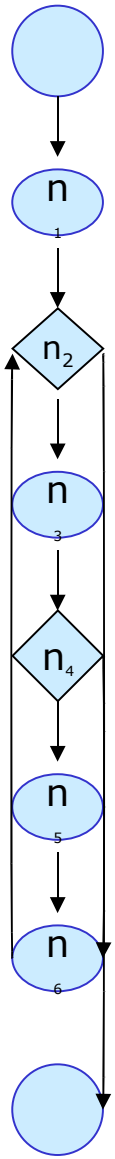
- auch Programm-ablaufplan
- Gerichteter Graph, bestehend aus Knoten und Kanten
- Besitzt einen Start- und einen Endknoten
- Folge von Knoten und Kanten vom Start- zum Endknoten heißt Pfad



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

2.1 Anweisungsüberdeckungstest

- Auch C_0 – Test (C = *Coverage*) genannt
- Verlangt Ausführung aller Anweisungen (**Knoten**)
- Testmenge: {[„A", „1"]}
- Ein Test reicht aus. Testpfad enthält alle Knoten, aber nicht alle Kanten




```
cin >> Zchn;

while ((Zchn >= ‚A') &&
(ZCHN <= ‚Z') &&
(Gesamtzahl < INT_MAX))

Gesamtzahl = Gesamtzahl + 1;

if  ((Zchn == ‚A') ||
(Zchn == ‚E') || (Zchn == ‚I') ||
(Zchn == ‚O') || (Zchn == ‚U'))

VokalAnzahl = VokalAnzahl + 1;

cin >> Zchn;
```

Zweig (n_4, n_6) wird nicht notwendig ausgeführt

2.1 Anweisungsüberdeckungstest

Eigenschaften:

- 100prozentige Überdeckung bedeutet: jede Anweisung wurde mindestens einmal ausgeführt
- Wesentliche Aspekte eines Programms werden nicht geprüft

Metrik: *Überdeckungsgrad =*

Zahl der ausgeführten Anweisungen / Gesamtzahl aller Anweisungen

Leistungsfähigkeit:

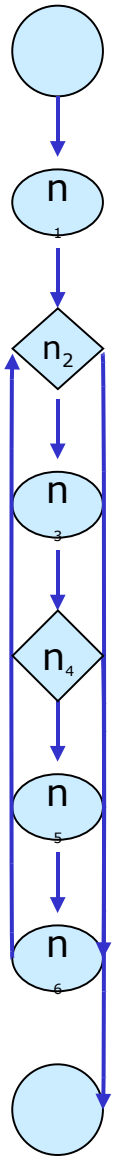
- Niedrigste Fehleridentifizierungsquote, 18 % der Fehler werden entdeckt

Bewertung:

- *Notwendiges*, aber nicht hinreichendes Testkriterium
- Nicht ausführbarer Code kann gefunden werden
- Eigenständig nicht geeignet, aber in Kombination mit anderen Verfahren

- Auch C_1 – Test genannt
- Verlangt Ausführung aller Zweige (**Kanten**)
- Testmenge: {[„A", „B", „1"]}
- Ein Test reicht aus. Testpfad enthält alle Kanten.

  Insbesondere sind die Kanten n_4--n_5--n_6 (Durchlauf mit „A") sowie n_4--n_6 (Durchlauf mit „B") abgedeckt
- Zweigüberdeckung wird auch als Entscheidungs-überdeckung bezeichnet




```
cin >> Zchn;

while ((Zchn >= ‚A') &&
(ZCHN <= ‚Z') &&
(Gesamtzahl < INT_MAX))

Gesamtzahl = Gesamtzahl + 1;

if  ((Zchn == ‚A') ||
(Zchn == ‚E') || (Zchn == ‚I') ||
(Zchn == ‚O') || (Zchn == ‚U'))

VokalAnzahl = VokalAnzahl + 1;

cin >> Zchn;
```

2.2 Zweigüberdeckungstest

Eigenschaften:

- 100prozentige Überdeckung bedeutet: jeder Zweig wurde mindestens einmal durchlaufen.
- Fehlende Zweige können nicht direkt entdeckt werden.

Metrik: *Überdeckungsgrad =*

Zahl der erfassten Kanten / Gesamtzahl aller Kanten

Leistungsfähigkeit:

- Höhere Fehleridentifizierungsquote als Anweisungsüberdeckung, ca. 34% der Fehler werden entdeckt, 79% der Kontrollflussfehler und 20% der Berechnungsfehler
- Leistungsfähigkeit schwankt in weitem Bereich zwischen 25% bis 75%

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bewertung:

- Gilt als *das* minimale Testkriterium
- Nicht ausführbare Zweige können gefunden werden
- Korrektheit des Kontrollflusses an Verzweigungen wird kontrolliert
- Gezielte Optimierung häufig durchlaufener Programmteile möglich

Nachteile:

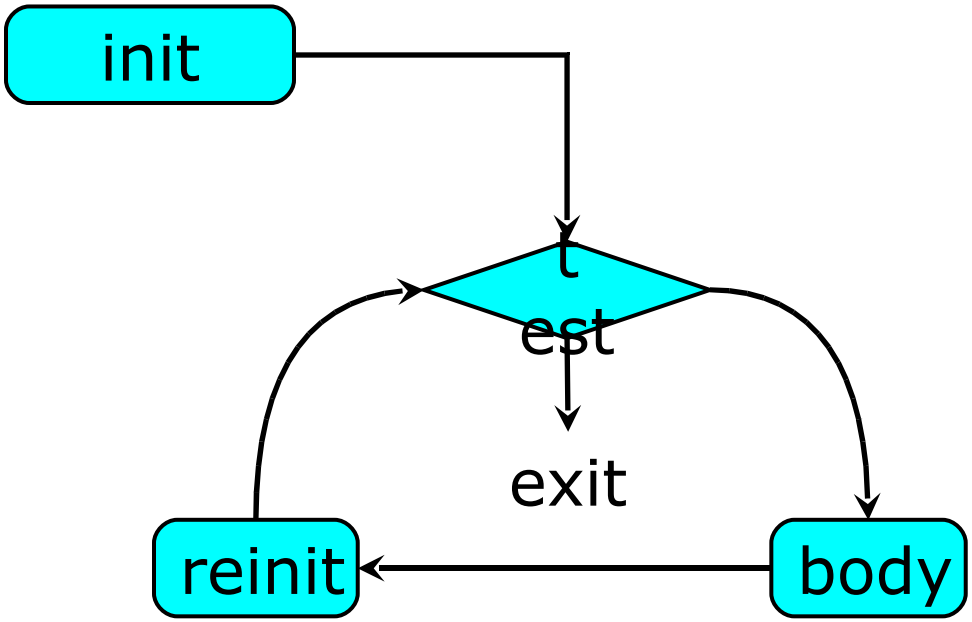
- Unzureichend für den Test von Schleifen
- Keine Berücksichtigung von Abhängigkeiten zwischen Zweigen
- Nicht geeignet für den Test komplexer Bedingungen

- Lösung der beiden ersten Nachteile: Pfadüberdeckungstest
- Lösung des letzten Nachteils: Bedingungsüberdeckungstests

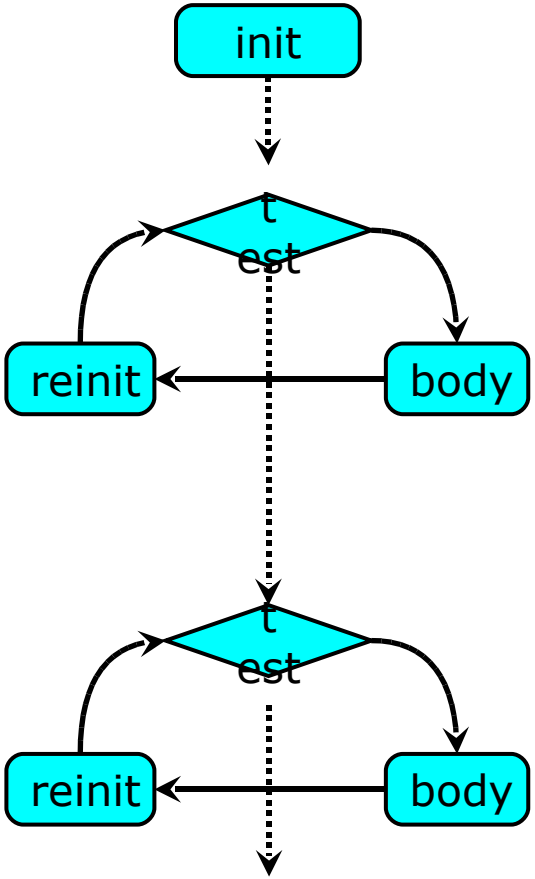
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Testen von Schleifen

- Anweisungs- und Zweigüberdeckung haben Probleme mit dem Test von Schleifen
- Typische Schleifenstruktur:



Problem des Wachstums der Anzahl der Pfade



(1) Zahl der Testfälle von while-Schleifen ist nicht vorab bekannt oder nicht beschränkt

(2) Zahl der Testfälle konsekutiver Schleifen ist multiplikativ

(3) Schleife mit Verzweigung im Körper: Ist N Schranke für Zahl der Schleifendurchläufe, so sind im worst case 2^N Testfälle erforderlich (exponentielles Wachstum)

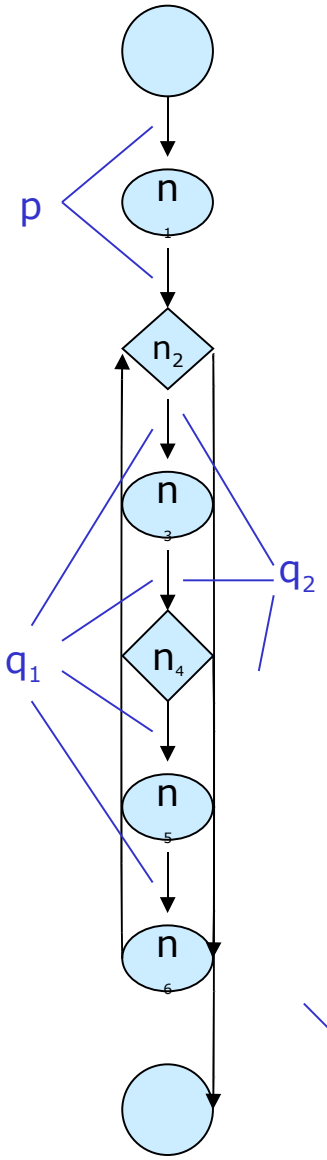
Brute force: Testbeispiele zur Ausführung **aller** unterschiedlichen Pfade im Programm

Beispiel:

Jeder Schleifendurchlauf trägt entweder zu Vokal oder zu Konsonant bei.

Pfade stehen in eineindeutiger Korrespondenz zu den Worten über dem Alphabet $\{q_1, q_2\}$.

Anzahl der Testpfade bei Beschränkung auf N Schleifendurchläufe ist also 2^N-1.




```
cin >> Zchn;

while ((Zchn >= ‚A‘) &&
(ZCHN <= ‚Z‘) &&
(Gesamtzahl < INT_MAX))

Gesamtzahl = Gesamtzahl + 1;

if  ((Zchn == ‚A‘) ||
(Zchn == ‚E‘) || (Zchn == ‚I‘) ||
(Zchn == ‚O‘) || (Zchn == ‚U‘))

VokalAnzahl = VokalAnzahl + 1;

cin >> Zchn;
```

Allgemeiner Pfadüberdeckungstest

Eigenschaften:

- Pfadanzahl bei unbestimmten Wiederholungen (while, …) nicht beschränkt.
- Ein Teil der konstruierbaren Pfade ist nicht ausführbar, da sich Bedingungen gegenseitig ausschließen können.

Leistungsfähigkeit:

- *Mächtigstes* kontrollflussorientiertes Testverfahren
- In einer vergleichenden Studie [Howden 78a, b, c] Erkennung von 18 von 28 Fehlern, um den Faktor 3 höhere Erkennung als Zweigüberdeckung
- Höhere Erfolgsquote nur durch Kombination mit anderen Verfahren

Bewertung:

- Praktische Bedeutung höchstens für Programmteile ohne Schleifen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Boundary Interior – Test

- **Idee**: Unterscheide Durchlauf nur keinmal, einmal, mehrmals
- Eingeschränkter Pfadüberdeckungstest, für Programme ohne Schleifen sogar identisch
- Zwei Gruppen von Pfaden für jede Schleife im Programm:
  1. Grenztest-Gruppe (*boundary tests*):
     Pfade, welche die Schleife nur einmal durchlaufen
     Testet Kombination init -- body
  2. Gruppe zum Test der Reinitialisierung (*interior tests*):
     Pfade, welche die Schleife mindestens einmal wiederholen
     Testet Kombination reinit -- body
- Praktisch anwendbar (im Gegensatz zum allgemeinen Pfadüberdeckungstest)
- **Strukturierter Pfadtest**

  Verallgemeinerung des *Boundary Interior* Tests

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bedingungsüberdeckungstest

- **Ziel:** Analysiert und überprüft die Testbedingungen in Schleifen und Verzweigungen aus struktureller Perspektive
- **Ansatz**: Bedingung ist logische Verknüpfung atomarer boolescher Funktionen. Testfälle sollen verschiedene Kombinationen dieser atomaren Werte überdecken. Analyse des Termbaums.
- 3 verschiedene Varianten:

  Einfache Bedingungsüberdeckung:

  Überdeckt alle atomaren Werte einzeln (im Extremfall 2 Testfälle)

  Mehrfach-Bedingungsüberdeckung:

  Überdeckt alle möglichen Kombinationen atomarer Werte (im Extremfall 2^n Testfälle)

  Minimale Mehrfach-Bedingungsüberdeckung:

  Jede Bedingung (ob atomar oder nicht) muss für sich überdeckt sein

Bewertung:

- Einfache Bedingungsüberdeckung:

  weder Zweig- noch Anweisungsüberdeckung enthalten

  sehr schwaches Kriterium

- Mehrfach-Bedingungsüberdeckung:

  Zweigüberdeckung ist enthalten, jedoch sehr aufwändig

- Minimale Mehrfach-Bedingungsüberdeckung:

  Wie einfache BÜ, aber beachtet die hierarchische Struktur

  Es müssen nicht nur die Blätter (Atome), sondern auch alle Teilbäume des Ausdrucksbaums überdeckt sein

  Sinnvolle Weiterentwicklung des Zweigtests (entspricht Überdeckung des Wurzelknotens)

Auswahl geeigneter kontrollflussorientierter Testverfahren

- Liegt Programm im Quellcode vor?
  - Nein: Kein Strukturtestverfahren möglich
- Besteht Programm nur aus Anweisungen?
  - Anweisungsüberdeckung sinnvoll
- .. nur Anweisungen und Verzweigungen mit atomaren Testbedingungen
  - Zweigüberdeckung sinnvoll
- .. Anweisungen, Verzweigungen und Schleifen mit atomaren Testbedingungen
  - Pfadüberdeckung, je nach Komplexität der Schleifensemantik doppelte oder mehrfache Schleifenüberdeckung
- ... komplexe Testbedingungen
  - Kopplung geeigneter Verfahren mit Bedingungsüberdeckung