Software-Qualitätsmanagement

Vorlesung im Modul 10-202-2319 Software-Management

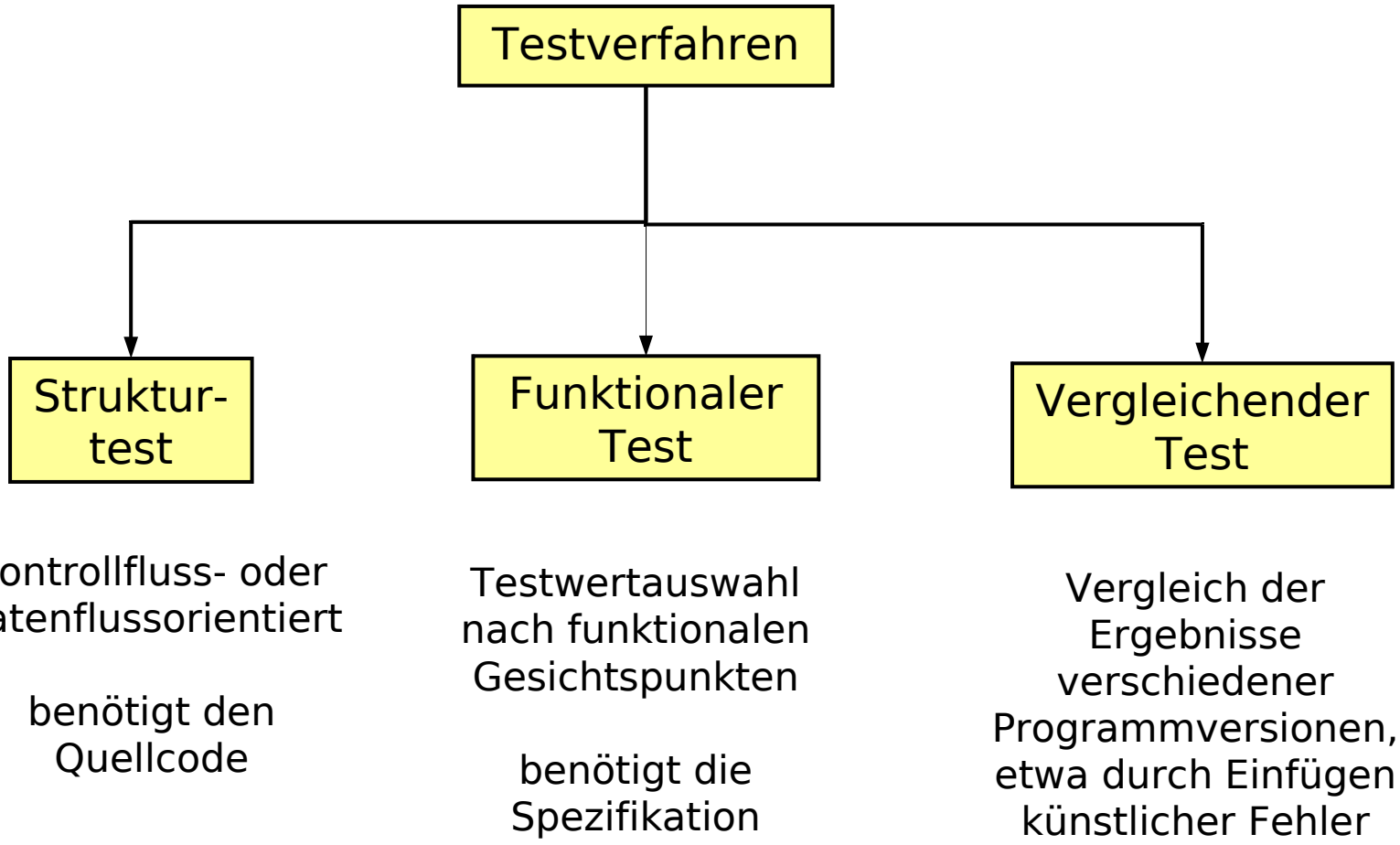
Sommersemester 2012

Prof. Dr. Hans-Gert Gräbe

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Klassifikation testender Verfahren



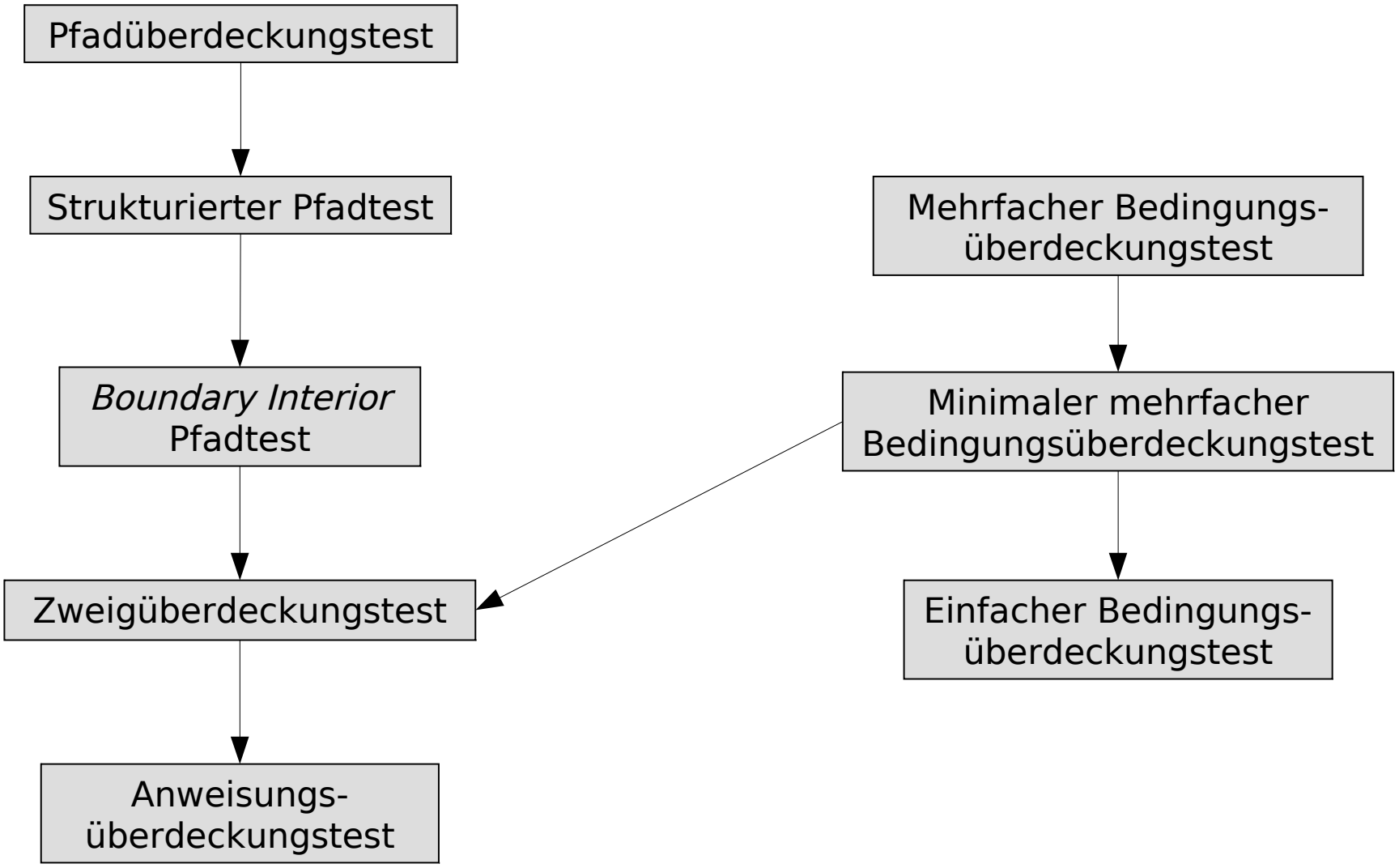
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **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
- **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
Boundary Interior Pfadtest
Zweigüberdeckungstest
Anweisungs-überdeckungstest
Mehrfacher Bedingungs-überdeckungstest
Minimaler mehrfacher Bedingungsüberdeckungstest
Einfacher Bedingungs-überdeckungstest

Ein Beispiel

/* **Funktion**: ZaehleZchn

Aufgabe: Die Prozedur ZaehleZchn zählt die Zeichen sowie Vokale in einem String. Dazu werden die Klassenvariablen *Gesamtzahl* und *VokalAnzahl* der Klasse *count* inkrementiert.

Randbedingung: Die Funktion stellt sicher, dass Gesamtzahl stets größer oder gleich VokalAnzahl ist. */

```
public static void main(String[] argv) {
        Gesamtzahl=0;
        VokalAnzahl=0;
        for(int i=0; i<argv.length; i++)
            zaehleZeichen(argv[i].toUpperCase());
        System.out.println("Argumente enthalten "+Gesamtzahl+
            " Zeichen, davon "+VokalAnzahl+" Vokale");
   }
}
```

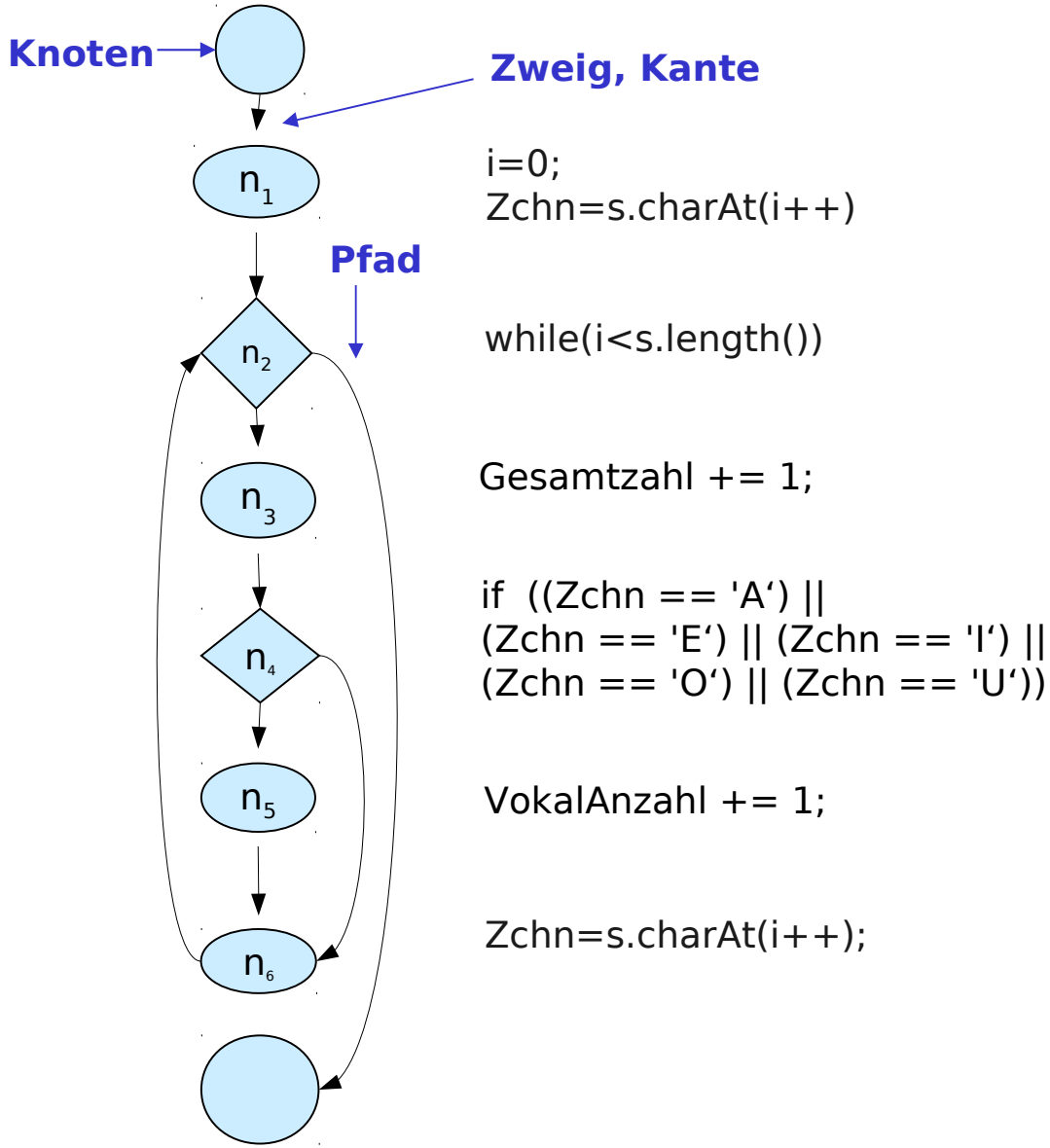
```
public class count {

    static int Gesamtzahl;
    static int VokalAnzahl;

    static void zaehleZeichen(String s) {
            int i=0;
            char Zchn=s.charAt(i++);
            while(i<s.length()) {
                Gesamtzahl+=1;
                if ((Zchn == 'A') || (Zchn == 'E') || (Zchn == 'I')
                        || (Zchn == 'O') || (Zchn == 'U'))
                        VokalAnzahl+=1;
                Zchn=s.charAt(i++);
            }
    }
```

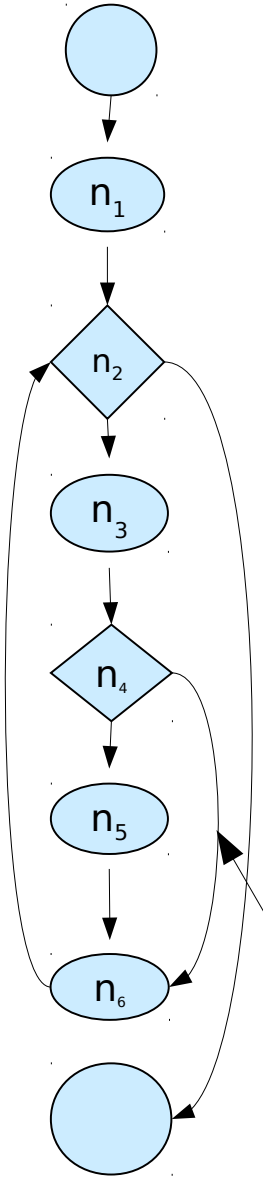
Kontrollflussgraph

- 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



2.1 Anweisungsüberdeckungstest

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




```
i=0;
Zchn=s.charAt(i++)

while(i<s.length())

Gesamtzahl += 1;

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

VokalAnzahl += 1;

Zchn=s.charAt(i++);
```

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

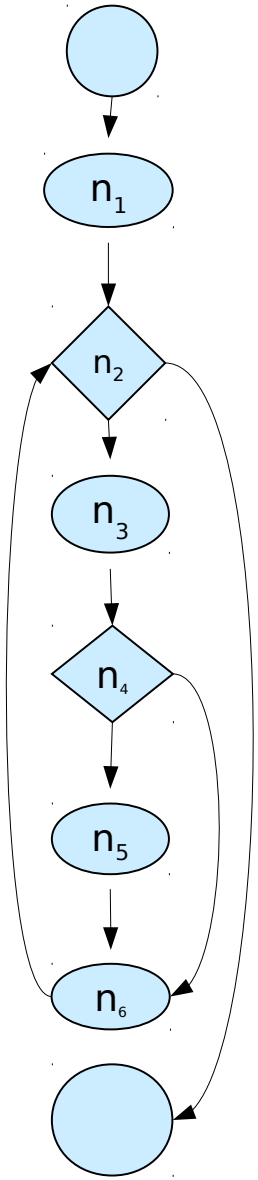
Bewertung:

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

2.2 Zweigüberdeckungstest

- Auch C_1-Test genannt
- Verlangt Ausführung aller Zweige (**Kanten**)
- Testmenge: {„AB“}
- Zwei Zeichen reichen 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




```
i=0;
Zchn=s.charAt(i++)

while(i<s.length())

Gesamtzahl += 1;

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

VokalAnzahl += 1;

Zchn=s.charAt(i++);
```

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

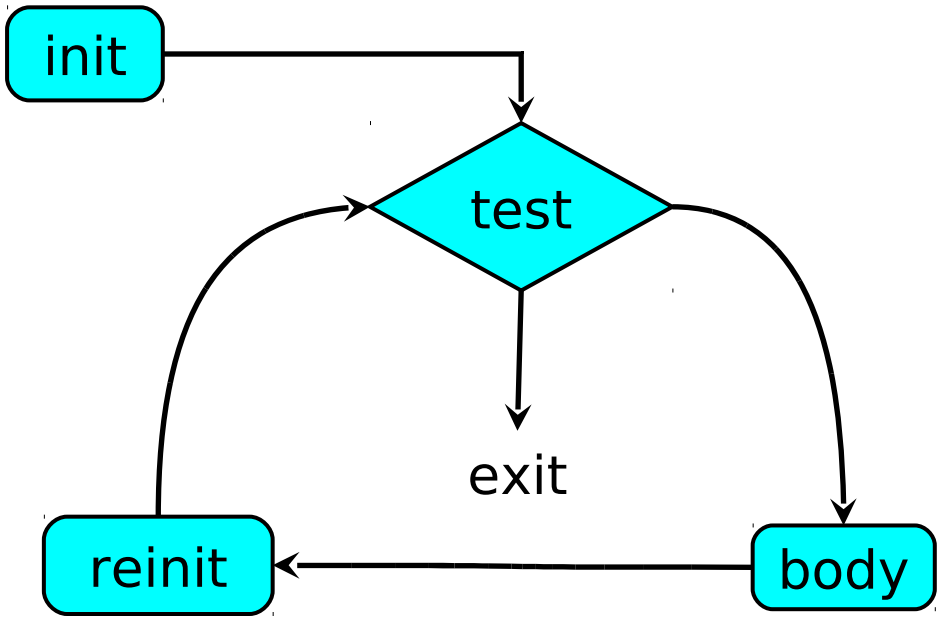
Nachteile:

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

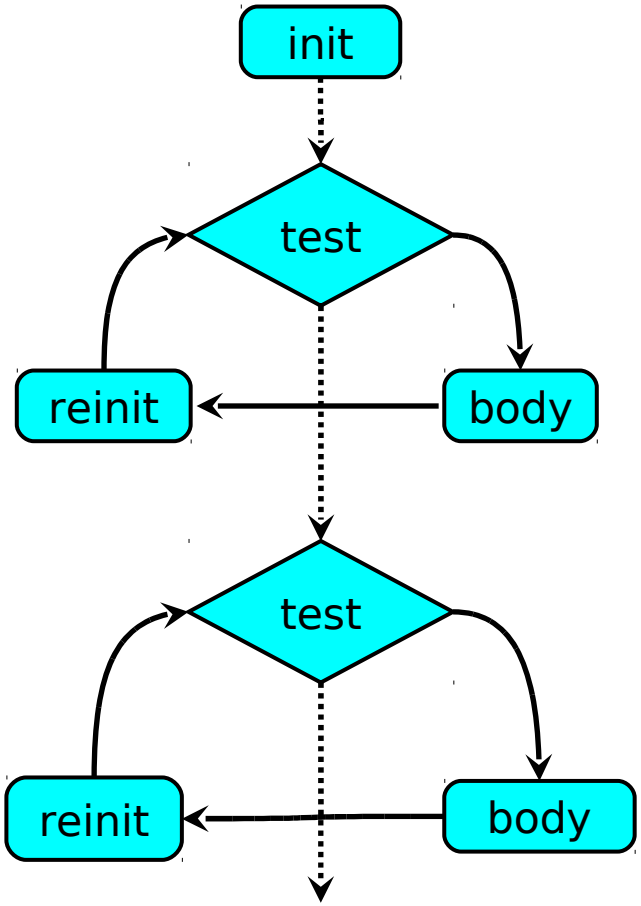
- Lösung der beiden ersten Nachteile: Pfadüberdeckungstest
- Lösung des letzten Nachteils: Bedingungsüberdeckungstests

Testen von Schleifen

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



Problem des Wachstums der Anzahl der Pfade



- Zahl der Testfälle von while-Schleifen ist nicht vorab bekannt oder nicht beschränkt
- Zahl der Testfälle konsekutiver Schleifen ist multiplikativ: $N=N_1*N_2$
- Schleife mit Verzweigung im Körper: Ist N Schranke für Zahl der Schleifendurchläufe, so sind im worst case 2^N Testfälle erforderlich (exponentielles Wachstum) – siehe das Beispiel auf der nächsten Folie

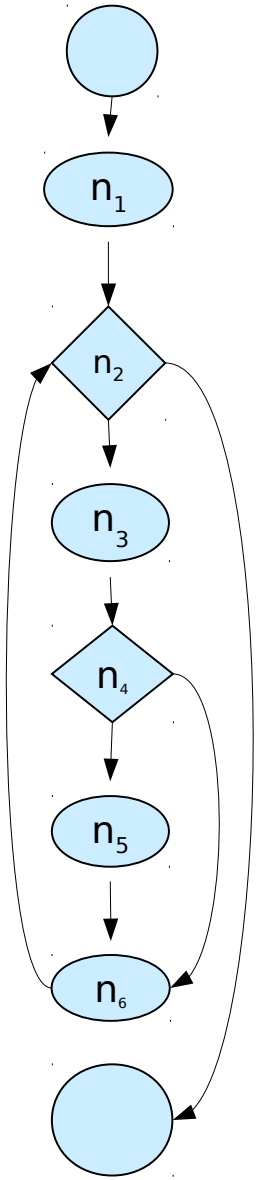
Brute force: Testbeispiele zur Ausführung **aller** unterschiedlichen Pfade im Programm

Beispiel:

Jeder Schleifendurchlauf trägt entweder zu Vokal oder zu Konsonant bei.

Pfade stehen in eineindeutiger Korrespondenz zu den Worten über dem Alphabet {V,C}.

Anzahl der Testpfade bei genau N Schleifendurchläufen ist also 2^N, der Anzahl der Worte der Länge N über dem Alphabet {V,C}.




```
i=0;
Zchn=s.charAt(i++)

while(i<s.length())

Gesamtzahl += 1;

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

VokalAnzahl += 1;

Zchn=s.charAt(i++);
```

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
- 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
- Schleifenabweisungstest sowie zwei Gruppen von Pfaden für jede Schleife im Programm:
  - Grenztest-Gruppe (*boundary tests*):
    Pfade, welche die Schleife nur einmal durchlaufen
    Testet Kombination init – body
  - 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 Pfadüberdeckungstest als Verallgemeinerung

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 2b Testfälle, b = Anzahl der Blätter im Termbaum)

  Volle Mehrfach-Bedingungsüberdeckung:

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

  Minimale Mehrfach-Bedingungsüberdeckung:

  Jede Bedingung (ob atomar oder nicht) muss für sich überdeckt sein (im Extremfall 2k Testfälle, k = Anzahl der Knoten im Termbaum)

Bewertung:

Einfache Bedingungsüberdeckung:

- weder Zweig- noch Anweisungsüberdeckung enthalten
- sehr schwaches Kriterium

Volle 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

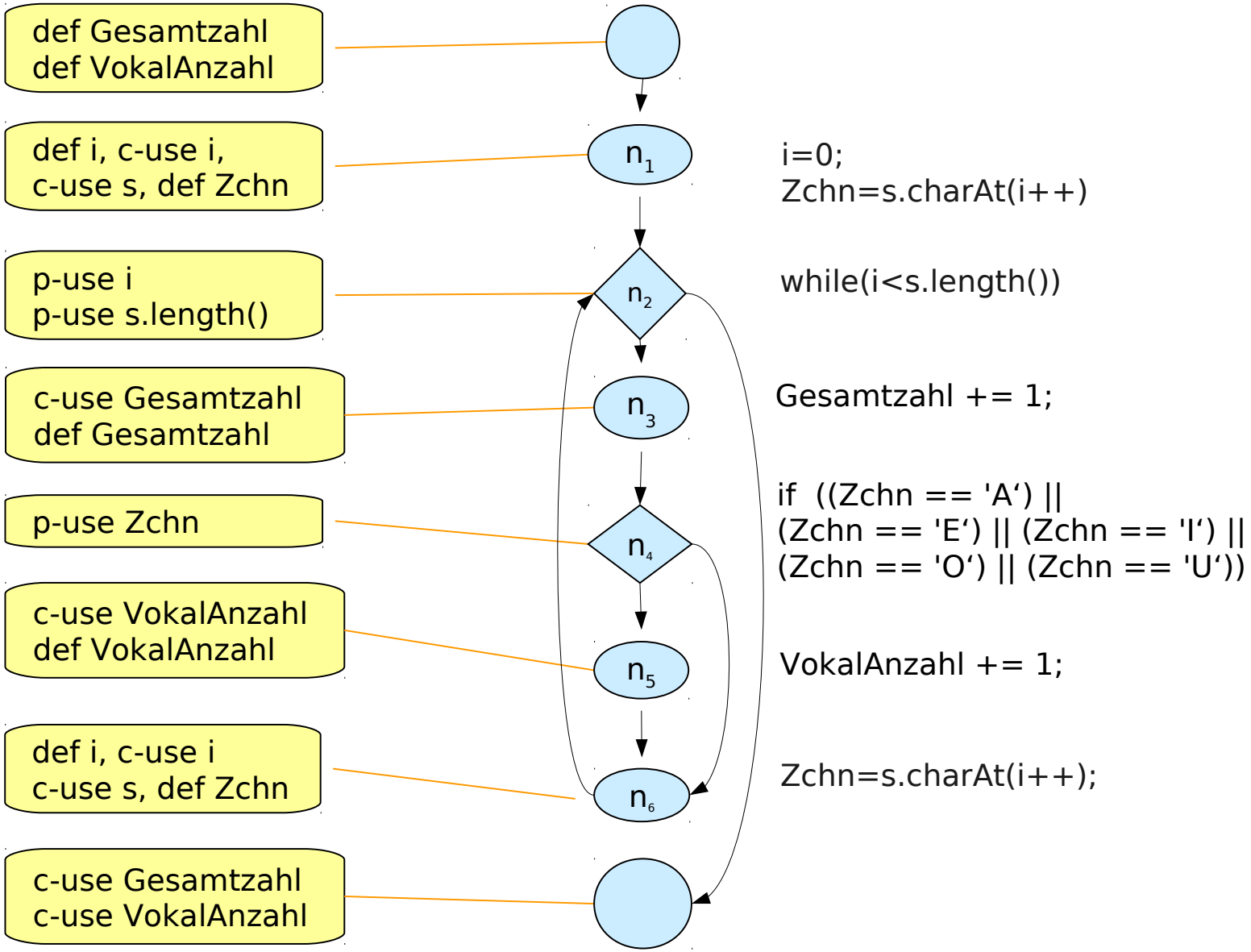
Datenflussorientierte Strukturtestverfahren

- Ebenfalls dynamisches Strukturtestverfahren
- Im Gegensatz zu kontrollflussorientierten Verfahren werden Datenbenutzungen und damit eher globale Aspekte getestet.
- Analyse der Programmdynamik an Hand der Dynamik der in Variablenwerten gespeicherten Programmzustände
  - Aufstellen des Datenflussdiagramms
    - Sichtbarkeit und Lebensdauer von Bezeichnern
    - Variablenidentitäten: Gültigkeitskontext und Kontrollfluss
    - lesende und schreibende Zugriffe auf Variablen (use, def)
    - Unterscheidung lesender Zugriffe in Anweisungen und Bedingungen (c-use, p-use)
- Eignen sich besonders für den Test von Datenobjekt- und Datentypmodulen sowie Klassen.
- Nur wenige Testwerkzeuge vorhanden.

Defs/Uses-Verfahren

Klassifikation der Variablenzugriffe nach

- Zuweisung (set-Methode, Definition, *def*)
  - Variablenwert wird an einer solchen Stelle geändert
- Zugriff zur Berechnung von anderen Werten (*computational-use, c-use*)
  - Auswirkung auf Wert anderer Variablen (Programmzustand)
- Zugriff zur Berechnung von Wahrheitswerten in Bedingungen (*predicate-use, p-use*)
  - Auswirkung auf Wert der Testbedingung (Kontrollfluss)


def Gesamtzahl
def VokalAnzahl
def i, c-use i,
c-use s, def Zchn
p-use i
p-use s.length()
c-use Gesamtzahl
def Gesamtzahl
p-use Zchn
c-use VokalAnzahl
def VokalAnzahl
def i, c-use i
c-use s, def Zchn
c-use Gesamtzahl
c-use VokalAnzahl
n1
n2
n3
n4
n5
n6
i=0;
Zchn=s.charAt(i++)
while(i<s.length())
Gesamtzahl += 1;
if ((Zchn == 'A') ||
(Zchn == 'E') || (Zchn == 'I') ||
(Zchn == 'O') || (Zchn == 'U'))
VokalAnzahl += 1;
Zchn=s.charAt(i++);

Datenflussgraph

- Knoten: jede Verwendung jeder Variablen B im Quelltext:
  (def B) (c-use B) (p-use B)
- Kanten: von (def B) zu allen nachfolgenden (use B), gelegentlich auch eine Kante pro möglichem Kontrollfluss
- Kantenmarken: Angabe des / der möglichen Kontrollflüsse (als Pfade im Kontrollflussgraphen)

Datenflussorientierte Test-Verfahren

- ***all defs* Kriterium**
  - Testfälle sind so zu wählen, dass jeder Wertzuweisung an eine Variable auch eine Wertbenutzung folgt.
  - Zu jeder Variablen gibt es einen Testfall, in welchem die Variable wenigstens einmal geschrieben und gelesen wird.
  - **Idee**: Jede Variable hat einen „Zweck“, eine Semantik, die wenigstens einmal exemplarisch geprüft wird.
- Spezialfall: Kontrolle, ob alle definierten Variablen auch verwendet werden (statischer Test, den der Compiler durchführen kann)
  - Charakterisiert durch fehlende abgehende Pfeile im DFD.
  - Variablen, die nicht benutzt werden, weisen auf Programmfehler hin.
  - Problem fällt beim Aufstellen der Testfälle auf.

- ***all p-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder prädikativen Nutzung dieses Variablenwerts überdeckt wird
  - Zu jedem Kontrollfluss, in dem eine Variable geschrieben und später in einer Bedingung gelesen wird, gibt es einen Testfall, welcher diesen Kontrollfluss abdeckt.
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle relevanten Bedingungsknoten
    - Fokus auf die bedingungsrelevanten Variablen
  - beinhaltet Zweigüberdeckung

- ***all c-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder berechnenden Nutzung dieses Variablenwerts überdeckt wird
  - Zu jedem Kontrollfluss, in dem eine Variable geschrieben und später in einer Berechnung gelesen wird, gibt es einen Testfall, welcher diesen Kontrollfluss abdeckt.
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle kausal davon abhängenden Ausdrücke.
    - Fokus auf die berechnungsrelevanten Variablen

- ***all c-uses / some p-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder berechnenden Nutzung dieses Variablenwerts überdeckt wird.
  - Falls kein berechnender Zugriff existiert, so muss der Wert in mindestens einem Prädikat benutzt werden.
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle kausal davon abhängenden Ausdrücke und exemplarischer Test von nur bedingungsrelevanten Variablen

- ***all p-uses / some c-uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder prädikativen Nutzung dieses Variablenwerts überdeckt wird.
  - Falls kein prädikativer Zugriff existiert, so muss der Wert in mindestens einer Berechnung benutzt werden.
  - **Idee**: Prüfen der Auswirkung einer Wertzuweisung an eine Variable auf alle davon abhängenden Bedingungen und exemplarischer Test von nur berechnungsrelevanten Variablen

- ***all uses* Kriterium**
  - Testfälle sind so zu wählen, dass jede Kombination einer Wertzuweisung mit jeder Nutzung dieses Variablenwerts überdeckt wird.
  - komplettester und damit aufwändigster datenflussorientierter Test

Leistungsfähigkeit nach Studie [Girgis, Woodward 86]

Vergleich *all defs, all p-/c-uses:*

- *all c-uses:* 48% der Fehler, insbesondere Berechnungsfehler,
- *all p-uses:* 34% und entdeckt Kontrollflussfehler,
- *all defs:* 24% der Fehler, aber keine Kontrollflussfehler

Weitere Verfahren

- **Idee**: Überdeckung längerer Sequenzen aus Zuweisung und Nutzung
  - *Required k-Tuples Test*
- **Idee**: Orientierung nicht an abgehenden, sondern an ankommenden Pfeilen im DFG
  - *Datenkontextüberdeckung*: jede mögliche Herkunft eines Werts wird überdeckt.
  - *geordnete Datenkontextüberdeckung*: zusätzliche Beachtung der Zuweisungsreihenfolge