

Software- Qualitätsmanagement

Kernfach Angewandte Informatik

Sommersemester 2006

Prof. Dr. Hans-Gert Gräbe



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.

Ein Beispiel

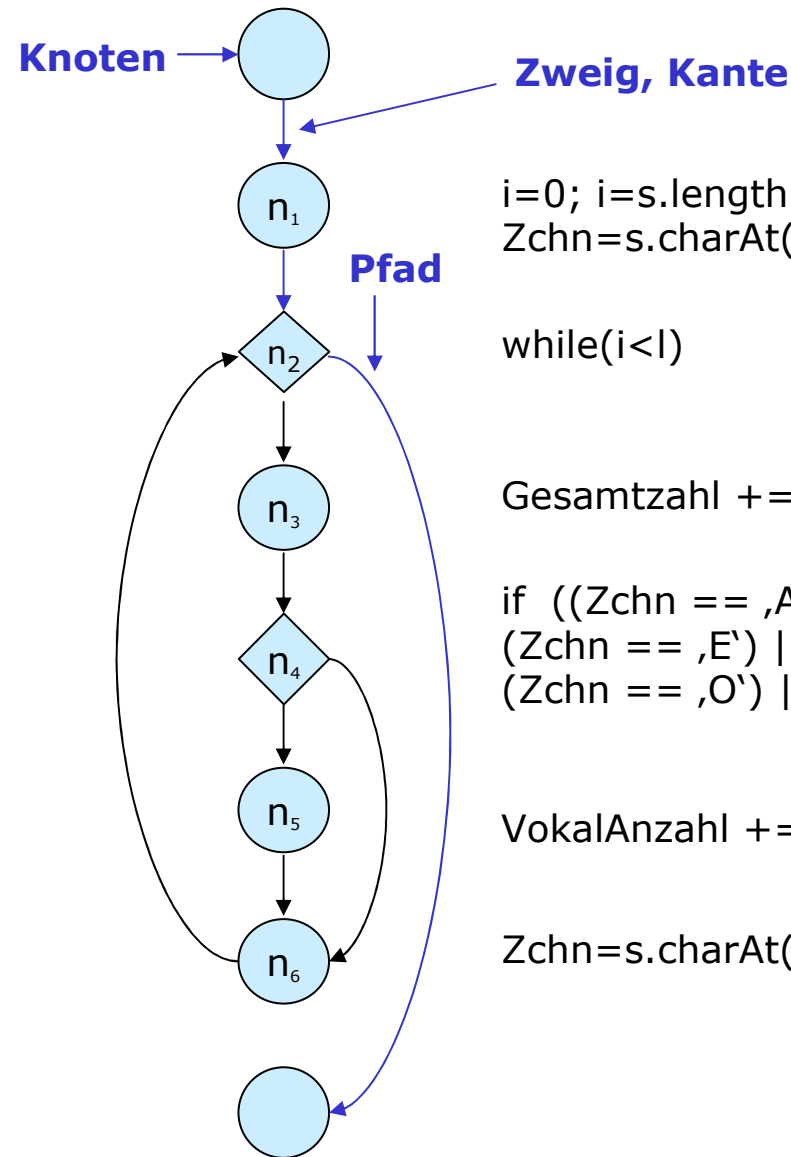
Die Funktion **zaehleZeichen** erwartet eine Zeichenkette in Großbuchstaben als Eingabe, zählt die Zeichen und die Vokale (keine Umlaute) und kumuliert das Ergebnis auf den statischen Variablen **Gesamtzahl** und **VokalAnzahl**

```
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");  
}
```

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

Kontrollflussgraph

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



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

```
while(i<l)
```

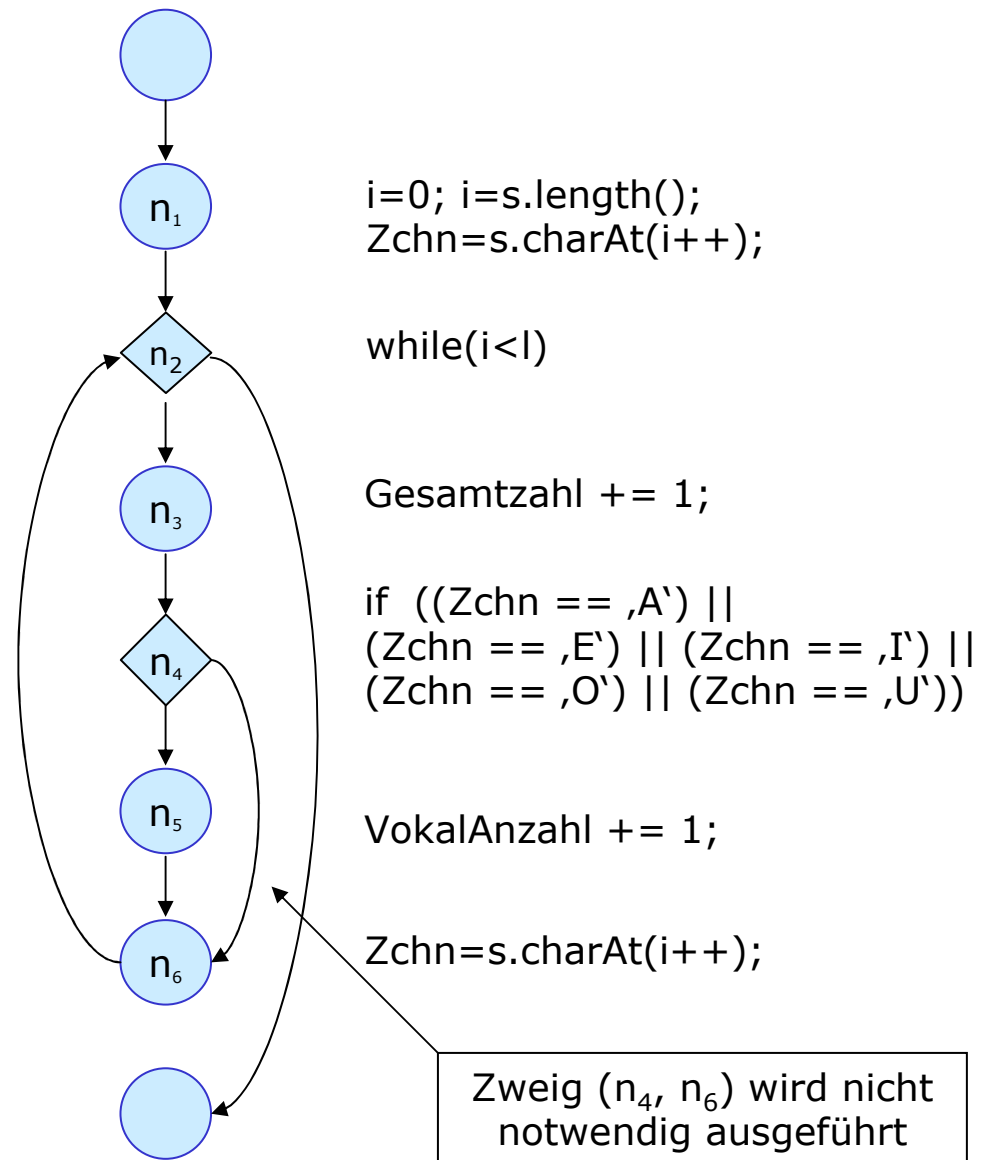
```
Gesamtzahl += 1;
```

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

```
VokalAnzahl += 1;
```

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

- Auch C_0 – Test ($C = \text{Coverage}$) genannt
- Verlangt Ausführung aller Anweisungen (**Knoten**)
- Testmenge: „A1“
- Ein Test reicht aus.
Testpfad enthält alle Knoten, aber nicht alle Kanten



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

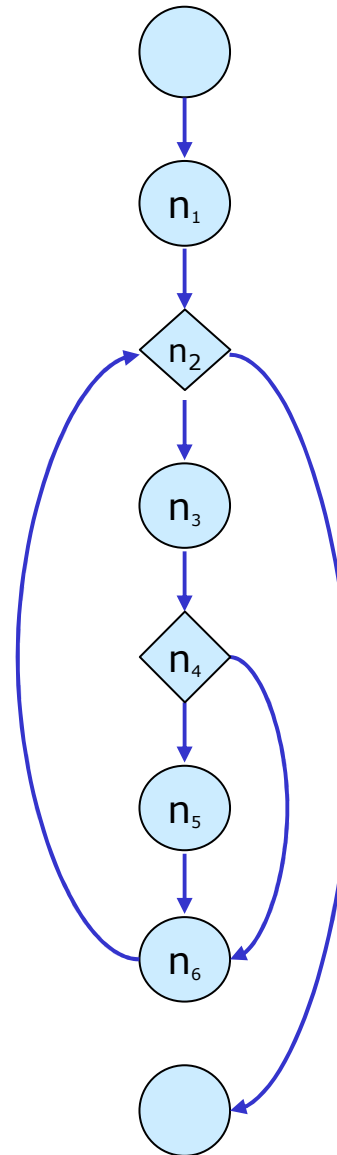
Bewertung:

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

5. Testende Verfahren

2.2 Zweigüberdeckungstest

- Auch C_1 – Test genannt
- Verlangt Ausführung aller Zweige (**Kanten**)
- Testmenge: „AB1“
- Ein Test reicht aus.
Testpfad enthält alle Kanten.
Insbesondere sind die Kanten
 $n_4 \rightarrow n_5 \rightarrow n_6$ (Durchlauf mit „A“) sowie
 $n_4 \rightarrow n_6$ (Durchlauf mit „B“) abgedeckt
- Zweigüberdeckung wird auch als Entscheidungsüberdeckung bezeichnet



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

```
while(i<l)
```

```
Gesamtzahl += 1;
```

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

```
VokalAnzahl += 1;
```

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


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%

5. Testende Verfahren

2.2 Zweigüberdeckungstest

Bewertung:

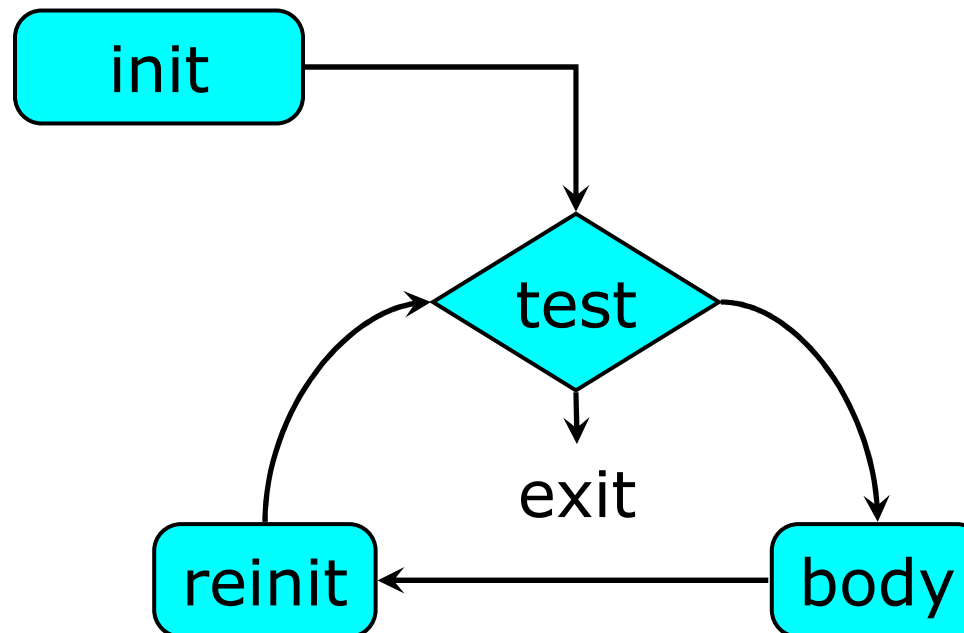
- Gilt als *das* minimale Testkriterium
- Nicht ausführbare Zweige können gefunden werden
- Korrektheit des Kontrollflusses an Verzweigungen wird kontrolliert
- Gezielte Optimierung häufig durchlaufener Programmteile möglich

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

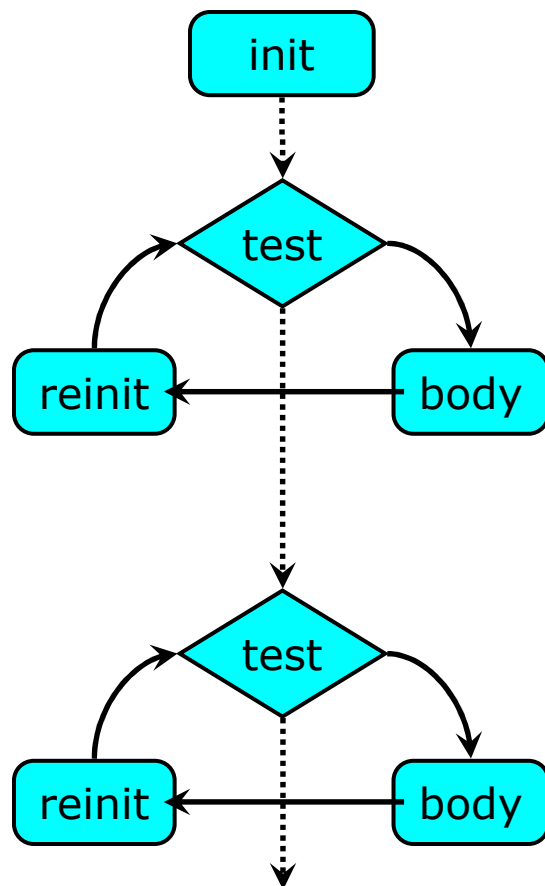
- Anweisungs- und Zweigüberdeckung haben Probleme mit dem Test von Schleifen
- Typische Schleifenstruktur:



5. Testende Verfahren

2.3. Pfadüberdeckungstests

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)

5. Testende Verfahren

2.3 Pfadüberdeckungstest

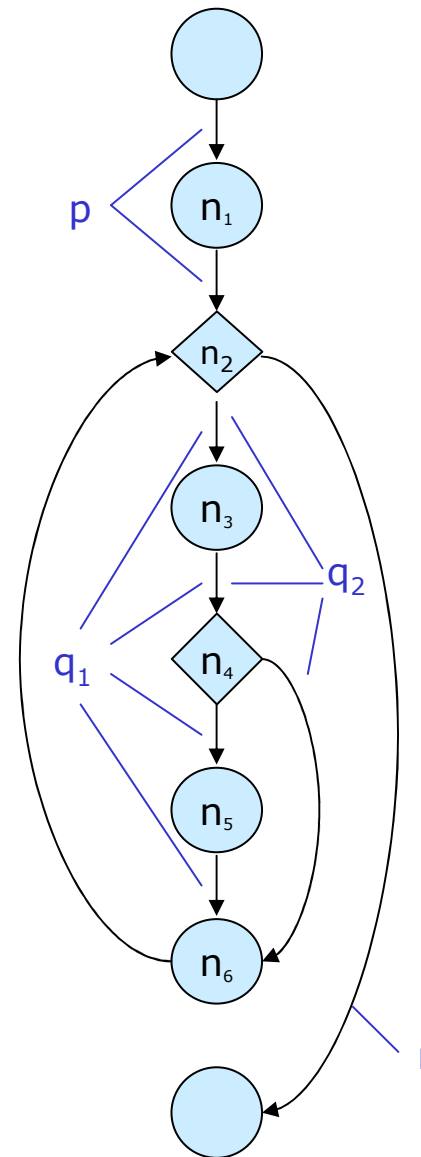
Brute force: Testbeispiele zur Ausführung **aller** unterschiedlichen Pfade im Programm

Beispiel:

Jeder Schleifendurchlauf trägt entweder zu Vokal oder zu Konsonant bei.

Pfade stehen in eindeutiger Korrespondenz zu den Worten über dem Alphabet $\{v,k\}$.

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



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

```
while(i<l)
```

```
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
- 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

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*

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

5. Testende Verfahren

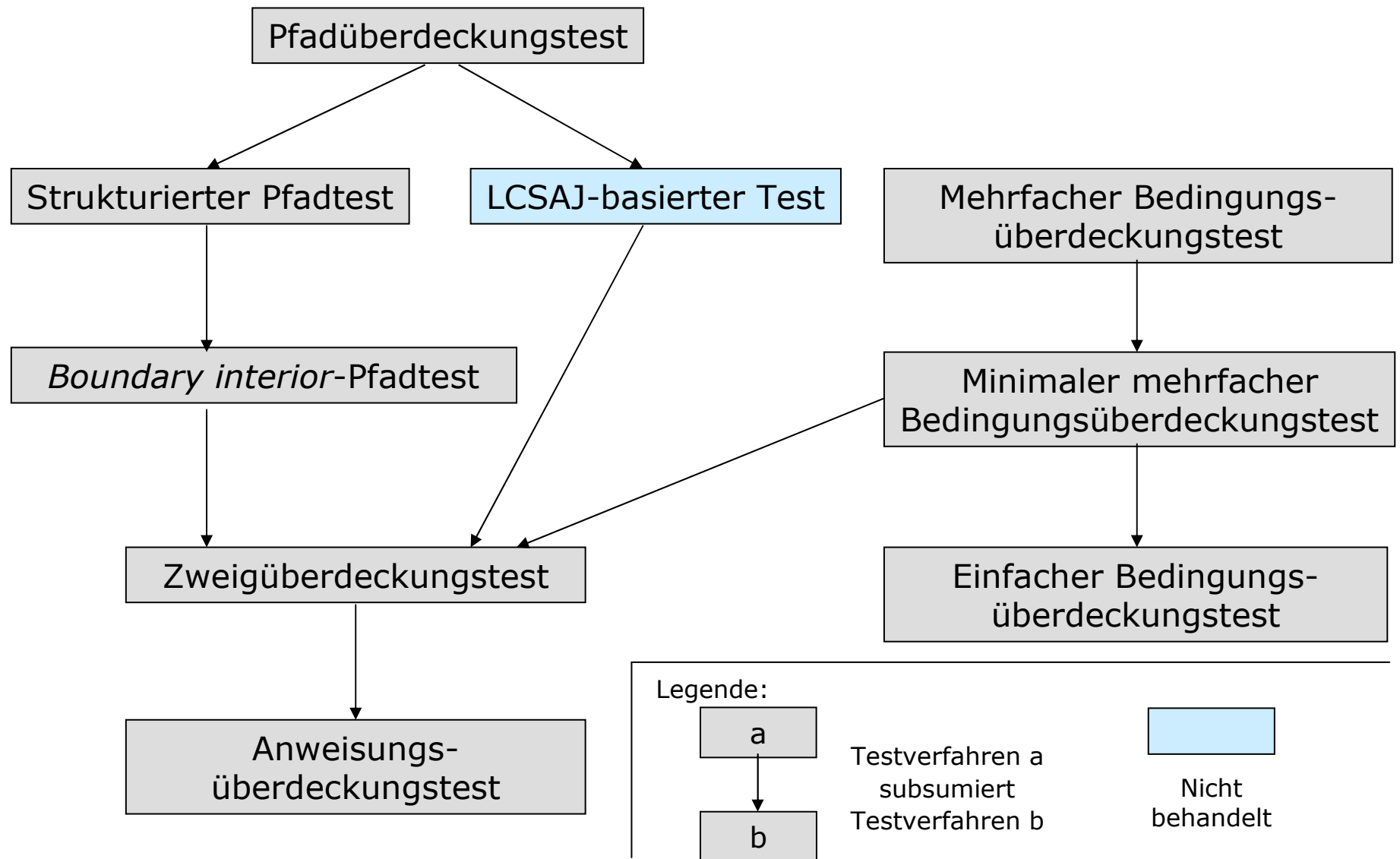
2.4 Bedingungsüberdeckungstest

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)

5. Testende Verfahren

2.5. Auswahl geeigneter Testverfahren



Auswahl geeigneter kontrollflussorientierter Testverfahren

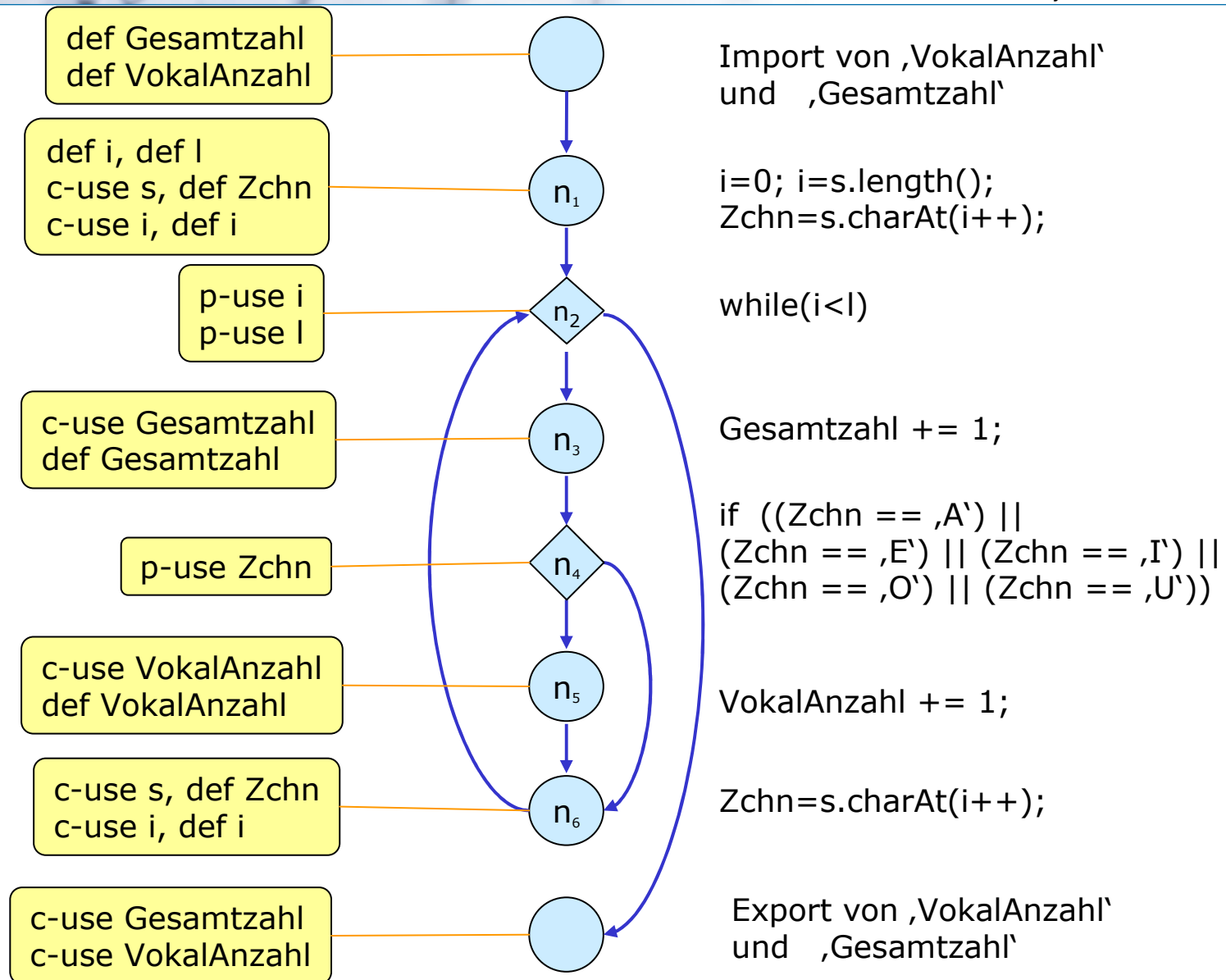
- Liegt Programm im Quellcode vor?
 - Nein: Kein Strukturtestverfahren möglich
- Besteht Programm nur aus Anweisungen?
 - Anweisungsüberdeckung sinnvoll
- .. nur Anweisungen und Verzweigungen mit atomaren Testbedingungen
 - Zweigüberdeckung sinnvoll
- .. Anweisungen, Verzweigungen und Schleifen mit atomaren 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.

5. Testende Verfahren

3. Datenflussorientierte Strukturtests

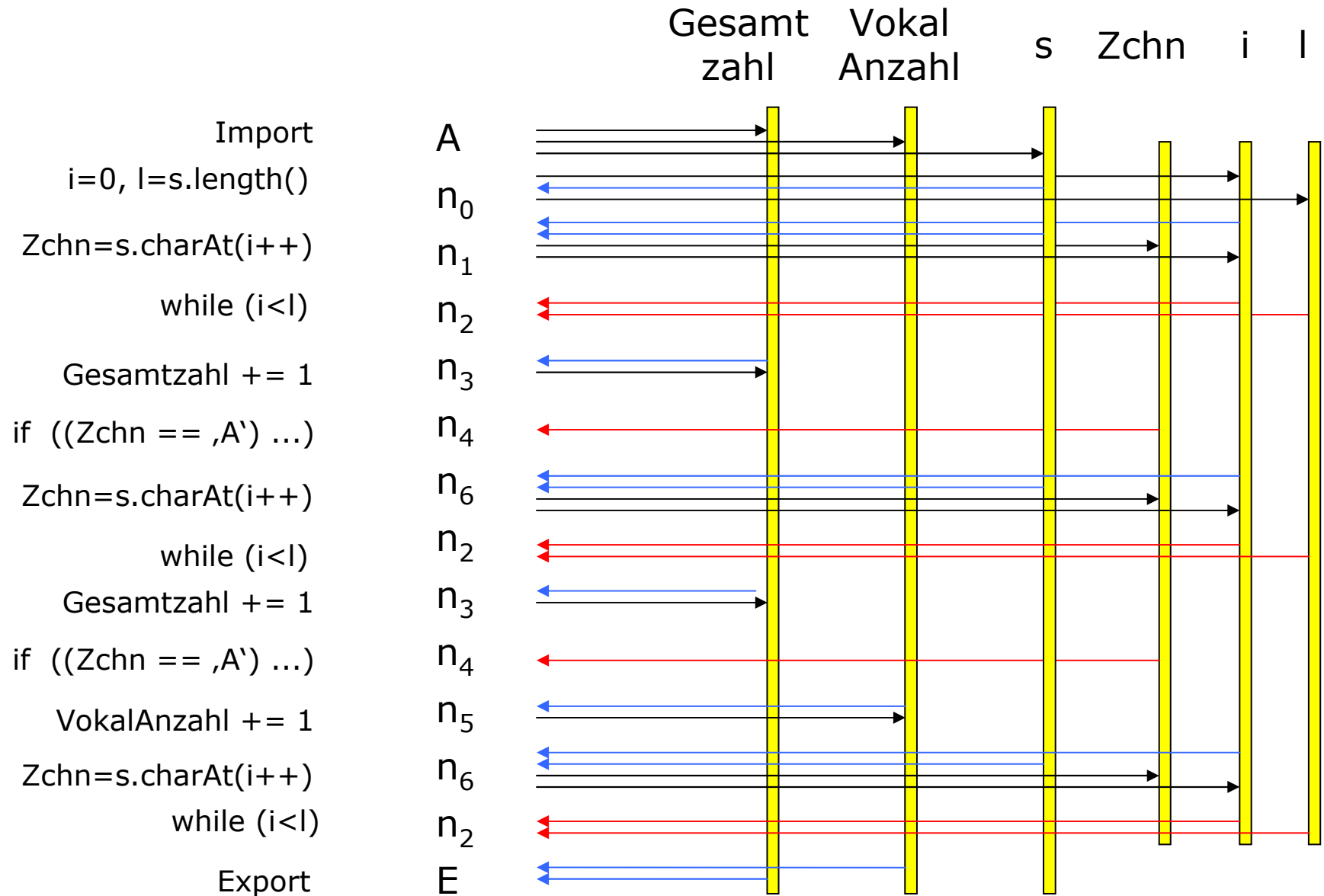


Defs/Uses-Verfahren

- 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)
- Zur Datenflussanalyse muss Zusammenhang zwischen Wertzuweisung und -benutzung analysiert werden
 - Ein Pfad heißt **definitionsfrei** für eine Variable x, wenn in dem Pfad keine Wertzuweisung an x erfolgt.
 - Variablenstelle ohne hinführende Pfeile im Datenflussgraphen

5. Testende Verfahren

3. Datenflussorientierte Strukturtests



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

5. Testende Verfahren

3. Datenflussorientierte Strukturtests

- ***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

5. Testende Verfahren

3. Datenflussorientierte Strukturtests

- ***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