Qualitätssicherung von Software

Prof. Dr. Holger Schlingloff

Humboldt-Universität zu Berlin
und
Fraunhofer FIRST

Hinweis auf Umfrage

- wir, das Institut für Informatik der Humboldt-Universität, möchten mehr über Ihre Sorgen und Nöte im Studium herausfinden. Daher möchten wir Sie bitten, sich an der Umfrage zu beteiligen
  http://www.informatik.hu-berlin.de/fragebogen
- Studiendauer, Abbrecherquote: zu hoch?
- Wer sich nicht wehrt, lebt verkehrt!
- Abgabeendtermin ist der 3.12.2004

Kapitel 2. Testverfahren

2.3 strukturelle (white box) Tests

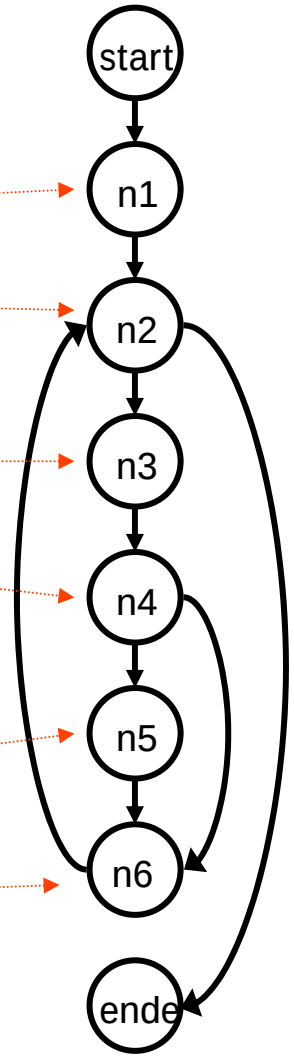
- kontrollflussorientiert
  - Grundlage: Programmtext, Kontrollflussgraph oder ausführbares Modell
  - Testfall = Pfad durch den Graphen

- datenflussorientiert
  - Grundlage: Programmtext, Zugriffe auf Variablen bzw. Parameter
  - Paare von schreibendem und lesendem Zugriff

! strukturelle Tests finden keine Auslassungsfehler !

Kontrollflussgraph

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



Beispiel: Liggesmeyer (f) / Balzert

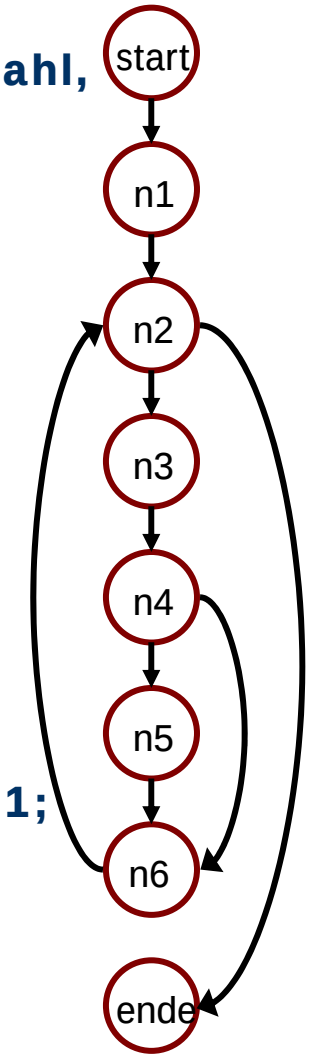
Kontrollflussorientierte Tests

- Überdeckungsmaße
  - Anweisungsüberdeckung
  - Zweigüberdeckung
  - Bedingungsüberdeckung
    - einfache Bedingungsüberdeckung
    - mehrfache Bedingungsüberdeckung
    - minimale Bedingungsüberdeckung
  - Pfadüberdeckung

Anweisungsüberdeckung

- auch C_0-Test genannt (statement coverage)
- jede Anweisung muss durch mindestens einen Testfall erfasst werden
- Beispiel: (A,1) ergibt Pfad (start,n1,n2,n3,n4,n5,n6, n2,ende)
- Kante (n4,n6) wird nicht durchlaufen

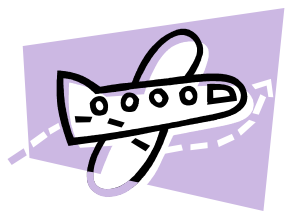
```
void ZaehleZchn (int& VokalAnzahl,
    int& Gesamtzahl){
    char Zchn;
    cin>>Zchn;
    while ((Zchn>=`A´) &&
     (Zchn<=`Z´) &&
     (Gesamtzahl<INT_MAX)){
        Gesamtzahl+=1;
        if ((Zchn==`A´)||
           (Zchn==`E´) ||
           (Zchn==`I´) ||
           (Zchn==`O´) ||
        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



Bewertung Anweisungsüberdeckung

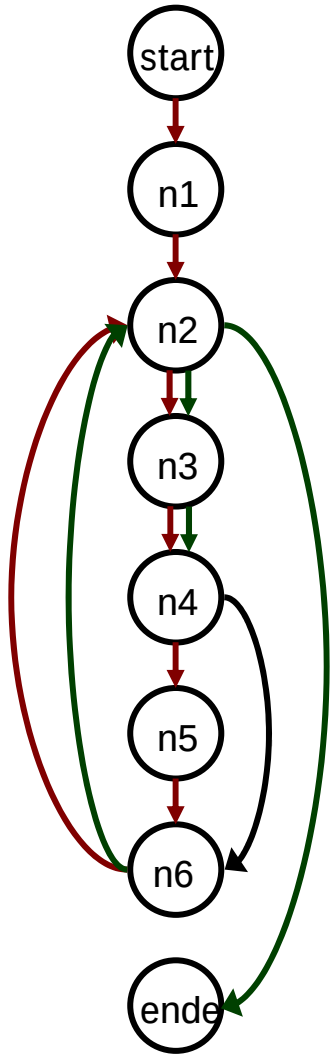
- Oft ist „*vollständige Anweisungsüberdeckung*“ das minimale Kriterium bei der Konstruktion einer Testsuite
- Überdeckungsgrad einer Testsuite: Prozentsatz der mindestens einmal ausgeführten Anweisungen (erstrebenswert: 100%)

- Minimum z.B. in DO-178B (ab Stufe C)
- oft als Überdeckungsmaß verwendet
- schwaches Kriterium (18% aufgedeckte Fehler)
- Beispiel: (x>5) statt (x>=5) wird nicht entdeckt



Zweigüberdeckung

- auch C_1-Test genannt (branch coverage)
- jede Kante muss durch mindestens einen Testfall erfasst werden
- Beispiel: (A,B,1) ergibt Pfad (start,n1,n2,n3,n4,n5,n6,n2, n3,n4,n6,n2,ende)
- Überdeckungsgrad: Prozentsatz der durchlaufenen Kanten



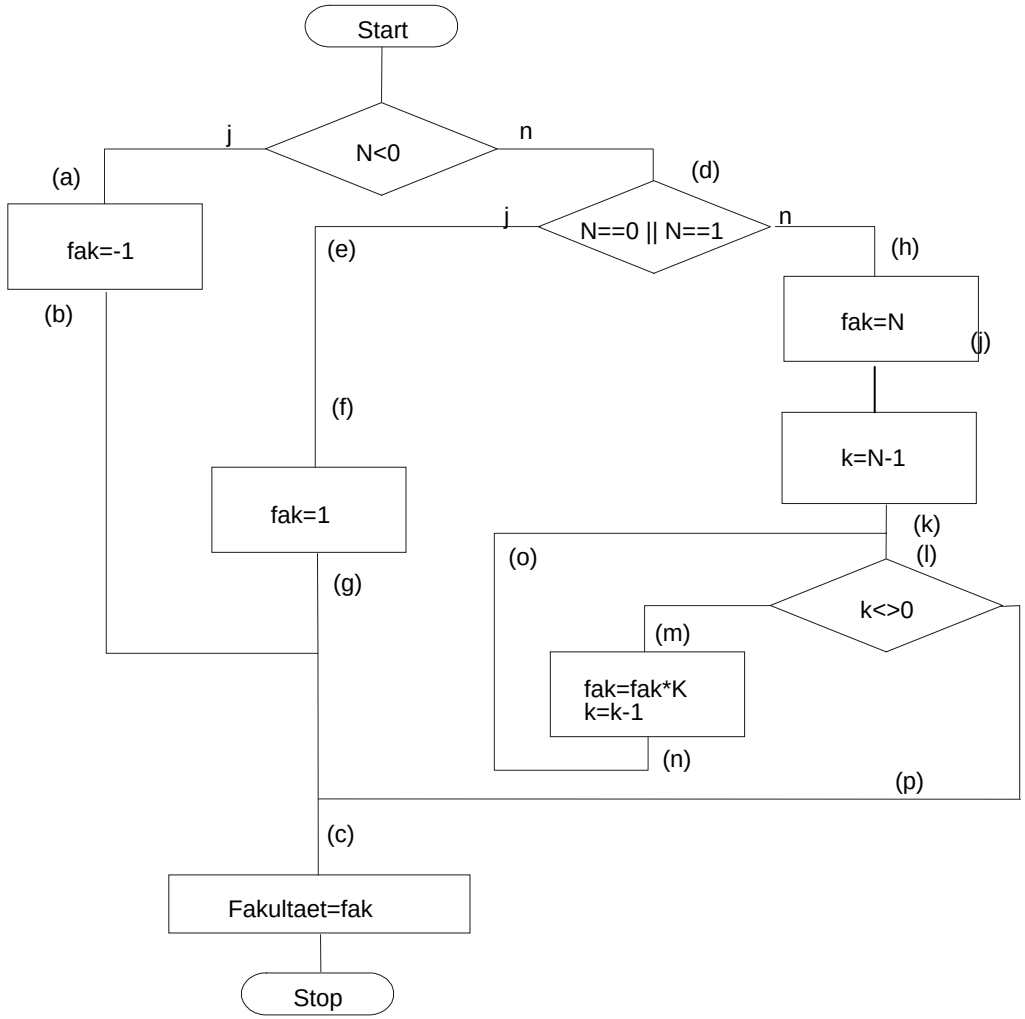
Bewertung Zweigüberdeckung

- subsumiert Anweisungsüberdeckung
- Schleifen werden ungenügend getestet (z.B. nur ein Mal durchlaufen)
- Problem der Gewichtung von Zweigen (Konvergenz der Überdeckungsrate mit Testfallanzahl, Fehleinschätzung)

- gut geeignet um logische Fehler zu finden, schlecht geeignet für Datenfehler
- Toolunterstützung durch Codeinstrumentierung

Übungsbeispiel

```
int Fakultaet (int N){
    if (N<0){
       fak=-1
    }
    else if (N==0 || N==1){
       fak=1
    }
    else {
       fak=N; K=N-1;
       while(k<>0){
         fak=fak*k;
         k=k-1
       }
    }
    Fakultaet=fak;
}
```



Bedingungsüberdeckung

- condition coverage test; Überprüfung der Entscheidungen im Programm
- Varianten:
  - einfache Bedingungsüberdeckung (C_2): Jede atomare Bedingung einmal wahr als auch einmal falsch
  - mehrfache Bedingungsüberdeckung (C_3 oder $C_2(M)$): Kombinationen atomarer Bedingungen
  - minimale Mehrfachbedingungsüberdeckung: Jede Entscheidung im Flussdiagramm wahr oder falsch

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

einfache Bedingungsüberdeckung

- jede atomare Bedingung mindestens in einem Testfall wahr und in einem Testfall falsch
- Problem: Ausführung teilweise compilerabhängig! (unvollständige Bedingungsauswertung)
- Problem, Programmfluss zur Bedingung zu steuern (Abhängigkeit der Variablenwerte)
- Problem, Gesamtentscheidung zu beeinflussen
- manchmal kombiniert mit Zweigüberdeckung zur Bedingungs/Entscheidungsüberdeckung (condition/decision coverage)

Mehrfachbedingungsüberdeckung

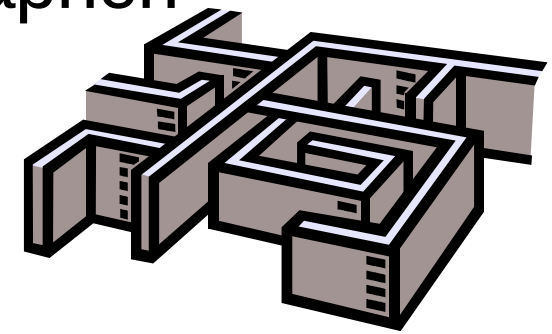
- alle Variationen der atomaren Bedingungen
- garantiert Variation der Gesamtentscheidung
- exponentiell viele Möglichkeiten
- Problem: Abhängigkeit von Variablen! (z.B. (Zchn==`A´)||(Zchn==`E´)) kann nicht beides wahr sein)
- Problem: kein vernünftiges Überdeckungsmaß

minimale Mehrfachbedingungsüberdeckung

- Evaluation gemäß Formelstruktur (jede Teilformel einmal wahr und einmal falsch)
- zusammengesetzte Entscheidungen werden zusammenhängend beurteilt
- Problem: ((A&&B)||C) wird durch (www) und (fff) vollständig überdeckt, aber nicht wirklich getestet
- Modifikation: zusätzlicher Nachweis, dass jede atomare Teilentscheidung relevant ist (z.B. durch positiven und negativen Testfall)
- z.B. für flugkritische Software erforderlich

Pfadüberdeckung

- Jeder Pfad durch den Kontrollflussgraphen
- Im Allgemeinen unendlich viele! (Überdeckungsmaß?)
- selbst falls endlich: „sehr viele“
- strukturierter Pfadtest: Äquivalenzklassen von Pfaden (ähnlich Grenzwertanalyse)
  - Schleife keinmal, einmal, mehr als zweimal ausgeführt (*boundary* oder *interior* condition)



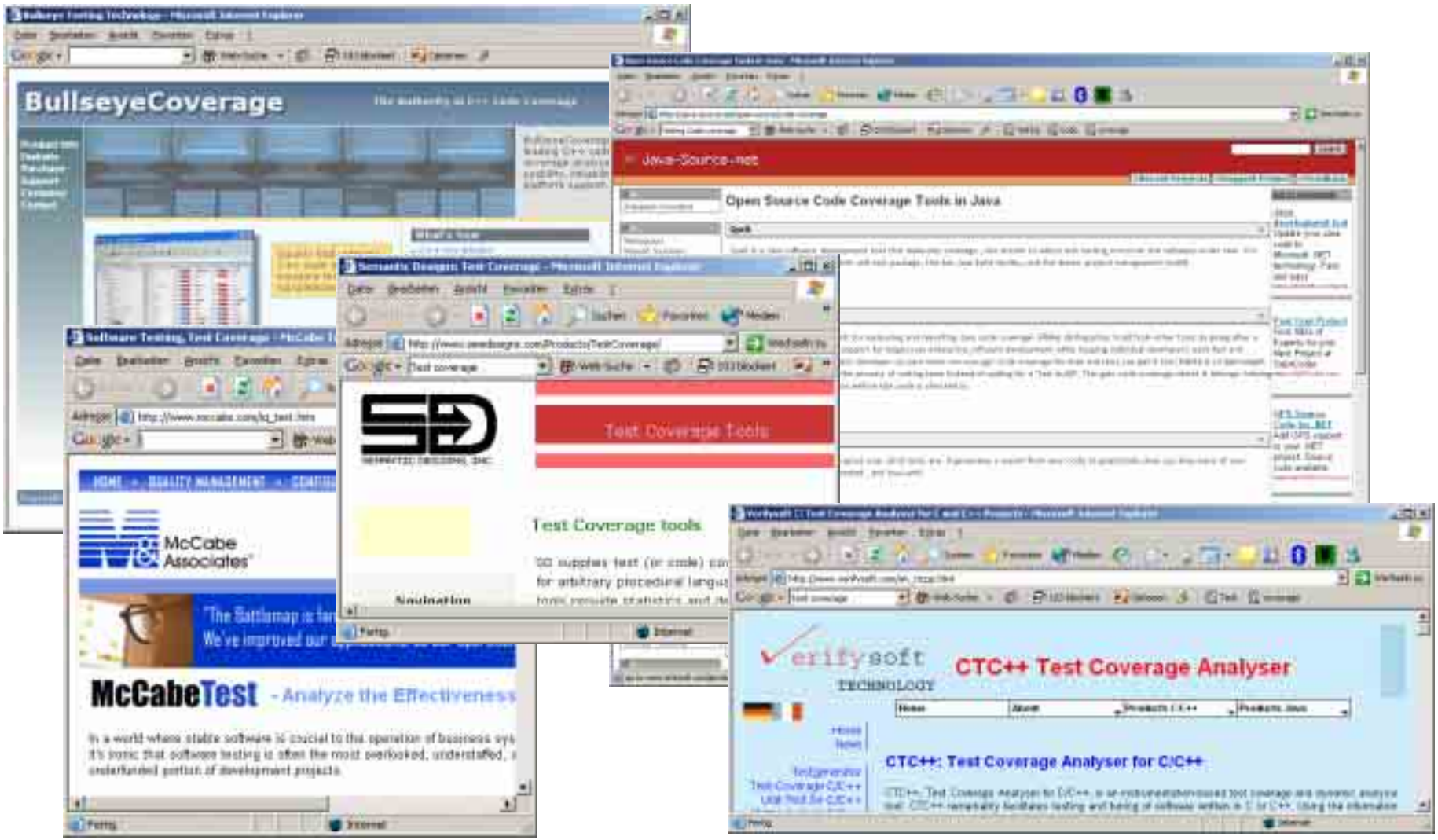
- zusätzlich intuitiv ermittelte Testfälle
- Werkzeugunterstützung?

modifizierter boundary-interior Test

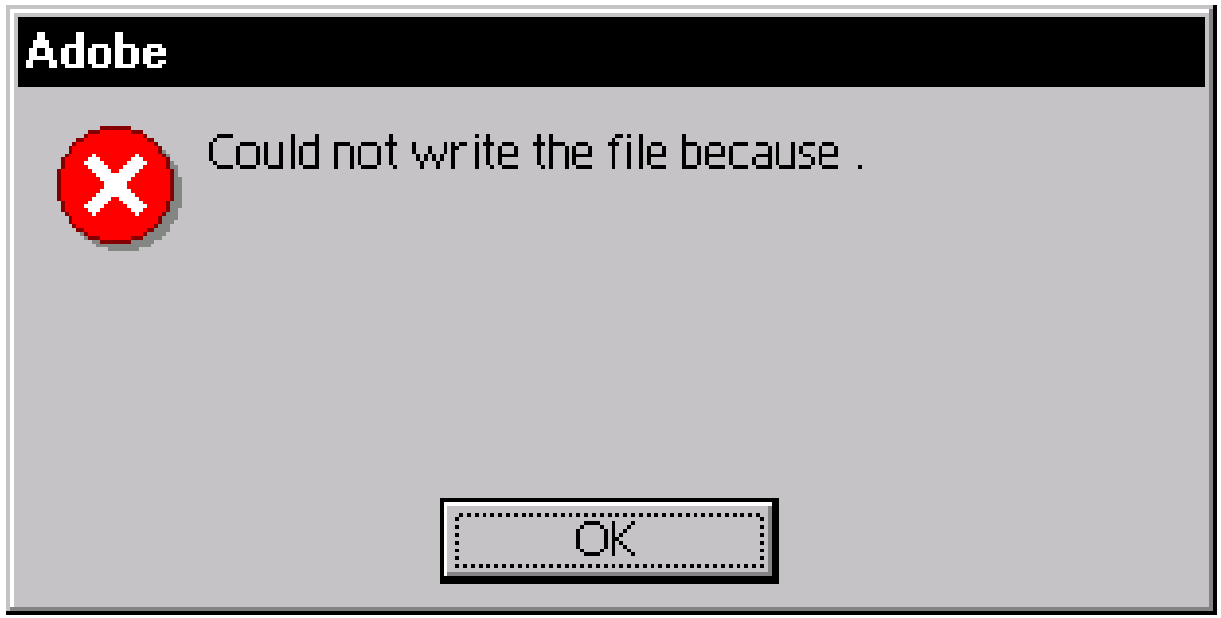
(nach Liggesmeyer)

- alle Pfade die Schleifen nicht betreten
- Schleifen einmal ausführen, Unterschiede im Rumpf (aber nicht in eingeschachtelten Schleifen) werden berücksichtigt
- Schleifen zweimal ausführen, wie oben
- jede Schleife separat beurteilen
- alle Zweige berücksichtigen

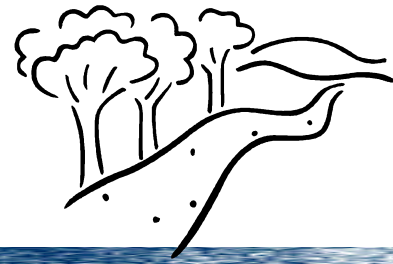
Coverage Tools



Pause!



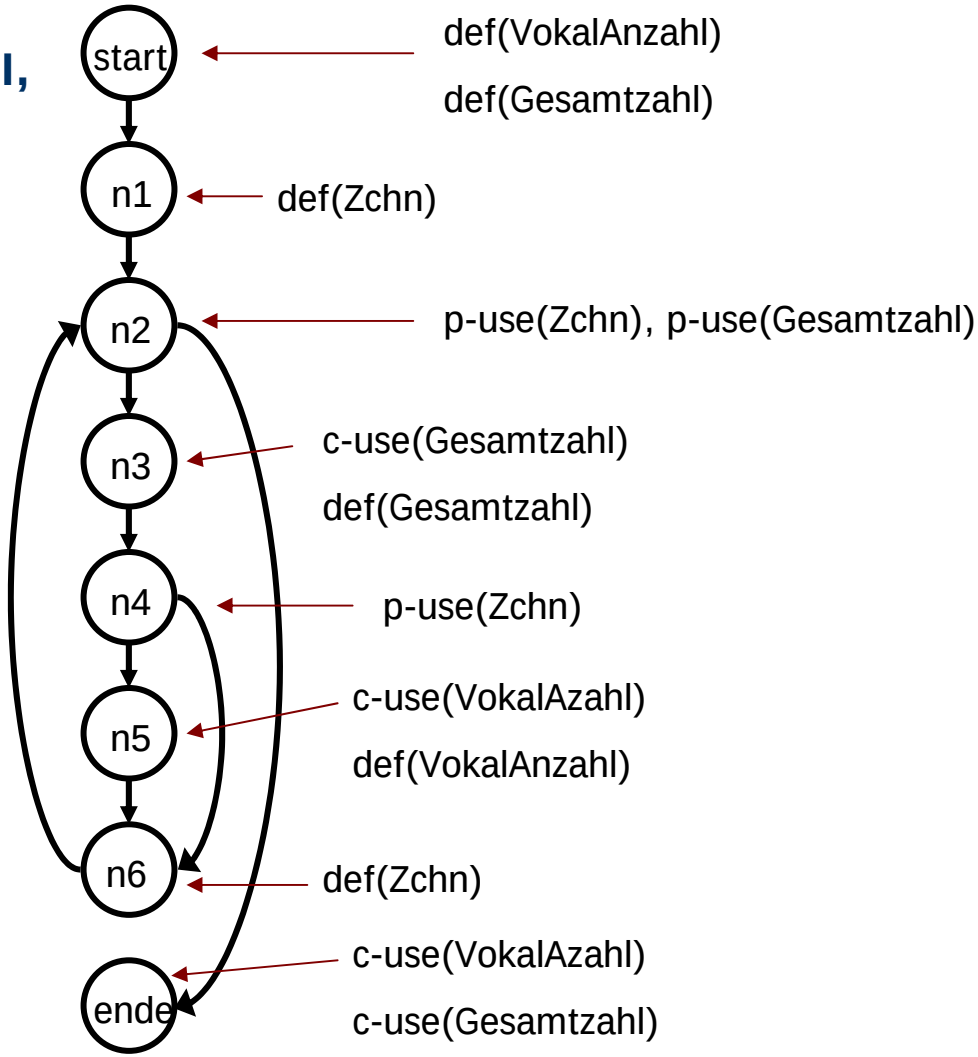
Datenflussorientierter Test



- Variablen und Parameter
  - Lebenszyklus:
    erzeugen – (schreiben – lesen*)* – vernichten
  - computational use versus predicate use
  - Zuordnung von Datenflussattributen zu den Knotens des Kontrollflussgraphen
- Berechnung der Variablenzugriffe
  - für jeden Definitionsknoten *n* für Variable *x* die Mengen *dcu(x,n)* und *dpu(x,n)* aller Knoten, in der *x* (berechnend oder prädikativ) verwendet wird

attributierter Kontrollflussgraph

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

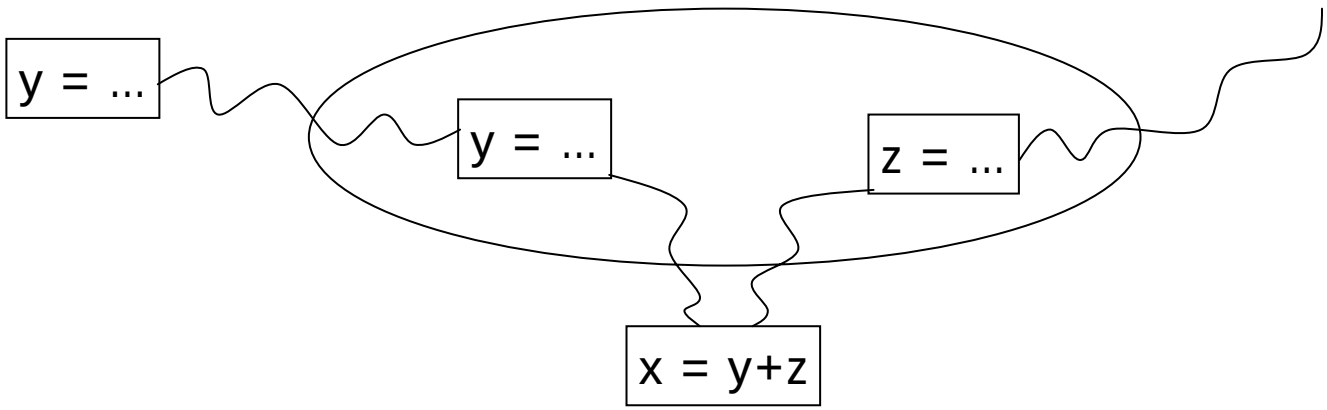


Defs/Uses-Kriterien zur Testabdeckung

- Testfallerzeugung
  - berechne Pfade zwischen Definition und Verwendung
  - ungenutzte Definitionen markieren Fehler
- Kriterien
  - *all defs:* Jeder Pfad von einer Definition zu mindestens einer Verwendung enthalten
  - *all p-uses:* Jeder Pfad von einer Definition zu irgendeiner prädikativen Verwendung
    - subsumiert Zweigüberdeckung
  - *all c-uses:* analog, zu computational use
  - *all c-uses/some p-uses:* jede berechnende oder mindestens eine prädikative Verwendung
  - *all uses:* jede Verwendung
  - *du-paths:* Einschränkung auf wiederholungsfreie Pfade

Datenkontexte

- Ein Datenkontext für einen Knoten ist eine Menge von Knoten, die für jede der im Knoten verwendeten Variablen eine Definition enthalten
- Datenkontext-Überdeckung: Alle Datenkontexte müssen vorkommen, d.h. jede Möglichkeit, den Variablen Werte zuzuweisen
- ggf. zusätzlich Berücksichtigung der Definitionsreihenfolge



Diskussion Datenflussüberdeckung

- Mächtiger als Kontrollflussverfahren
- Besonders für objektorientierte Programme
- *all c-uses* besser als *all p-uses* besser als *all-defs*
- Werkzeugunterstützung essenziell
- wenig Werkzeuge verfügbar

Bewertung strukturelle Tests

- Auslassungsfehler (auch: fehlende Ausnahmebehandlung usw.) prinzipiell nicht erfasst
- Konstruktion von Testfällen kann beliebig schwierig sein
- „dead code“ (der nie ausgeführt werden kann) wird normalerweise entdeckt
- Toolunterstützung
- Möglichkeit der Code-Optimierung durch Testergebnisse (häufig durchlaufene Programmteile verbessern); aber: Regressionstest erforderlich!

- Literaturempfehlung: *How to Misuse Code Coverage*; Brian Marick; http://www.testing.com/writings/coverage.pdf

Hausaufgabe

- Erstellen Sie Testsuiten, die die datenflussorientierten Abdeckungskriterien erfüllen!
- Wie unterscheiden sich die Testsuiten von den kontrollflussorientierten Testabdeckungen?