Programmiertechnik (C++)

Prof. Dr. Stefan Enderle
NTA Isny

Verifikation / Validierung

- ***Verifikation*** ist der Nachweis, dass ein vermuteter oder behaupteter Sachverhalt wahr ist.
- **Validierung** ist die dokumentierte Beweisführung, dass ein System die Anforderungen in der Praxis erfüllt.
- Beispiel:
  - Ein Programm besteht aus mehreren Funktionen, die genau beschrieben sind.
  - Die Funktionen können einzeln verifiziert werden.
  - Das Programm wird in der Anwendung validiert.

7. Testverfahren

7.1 Unit Tests

- **Unit-Tests (Modultests, Komponententests)** dienen zur Verifikation der Korrektheit eines Moduls einer Software, z.B. einer Klasse.

- Bei der **testgetriebenen Entwicklung** (Test-first-programming) genannt, werden die Modultests parallel zum eigentlichen Quelltext erstellt und gepflegt.

QTestLib

- Qt bietet eine umfangreiche und leicht zu handhabende Bibliothek für Unit-Tests an.
- Idee:
  - Zu jeder zu testenden Funktion oder Klasse gibt es eine Testklasse, die die Tests der eigentlichen Klasse in einzelnen Methoden durchführt.
  - Ein Standard-Testprogramm ruft sämtliche Methoden der Test-Klasse auf und zeigt deren Resultat.

Beispiel

- Beispiel für den Test einer einzelnen Funktion:

1. Die zu testende Funktion:

```
int addValue(int a, int b)
{
  return a+b;
}
```

Beispiel

2. Die Testklasse:

```
#include <QtTest/QtTest>

class Test : public QObject
{
  Q_OBJECT
private slots:
  void initTestCase()          // wird vor jedem Test aufgerufen
  {  }

  void add()
  {
    QCOMPARE(addValue(3,5), 8);
  }
};
```

Beispiel

4. Die „Hauptprogramm“:

```
QTEST_MAIN(Test)
```

Beispiel

5. Das Qt Projekt (`unittest.pro`):

```
CONFIG        += qtestlib
CONFIG        += console
HEADERS        = unittest.h
SOURCES        = unittest.cpp
```

Beispiel

6. Testdatei erzeugen (auf Kommandozeile):
```
qmake unittest.pro
make
```

7. Testdatei ausführen (auf Kommandozeile):
```
./unittest
```

Beispiel

Ausgabe:

```
********* Start testing of Test *********
Config: Using QTest library 4.3.2, Qt 4.3.2
PASS   : Test::initTestCase()
PASS   : Test::add()
PASS   : Test::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped
********* Finished testing of Test *********
```

Beispiel

Ausgabe bei Fehler:

```
********* Start testing of Test *********
Config: Using QTest library 4.3.2, Qt 4.3.2
PASS   : Test::initTestCase()
PASS   : Test::add1()
FAIL!  : Test::add2() Compared values are not the
  same
   Actual (addValue(3,5)): 8
   Expected (9): 9
   Loc: [unittest.cc(28)]
PASS   : Test::cleanupTestCase()
Totals: 3 passed, 1 failed, 0 skipped
********* Finished testing of Test *********
```

Test-Makros

- Die wichtigsten Test-Makros:
  - `QVERIFY(condition)`
    Prüft, ob die Bedingung erfüllt ist.
  - `QCOMPARE(act,exp)`
    Prüft, ob das erwartete Ergebnis (exp) dem aktuellen Ergebnis (act) entspricht.
  - `QEXPECT_FAIL(dataIndex, comment, mode)`
    Zeigt an, dass der nächste Test schiefgehen wird.
    Beispiel:
    ```
    QEXPECT_FAIL("", "Will be fixed in next
    release", Continue);
    ```

7.2 Dynamische Testverfahren

- Unit-Testing gehört zu den sog. *Dynamischen Testverfahren“*
- Die dynamischen Verfahren sind das, was man klassisch unter „Testen“ versteht.

Dynamische Testverfahren

- Merkmale von dynamischen Testverfahren sind:
  - das Programm wird mit konkreten Testdaten ausgeführt
  - das Programm wird in der realen Umgebung getestet
  - es handelt sich um Stichprobenverfahren
  - die Korrektheit des getesteten Programms kann nicht bewiesen werden.

Dynamische Testverfahren

- Problem:
  In der Regel können nicht alle möglichen Eingaben geprüft werden.

- Ziel: Erzeugung einer Stichprobe der Eingaben, die
  - repräsentativ
  - fehlersensitiv (Fehler sollen entdeckt werden!)
  - redundanzarm und
  - ökomisch ist.

White-Box Tests

- White-Box Tests basieren auf der Kontrollstruktur des Programms, zu der die Testfälle konstruiert werden.

- Ziel: Eine bestimmte Menge von Programmpfaden soll beim Test durchlaufen werden. Zur Erläuterung der Verfahren kann der Kontrollflussgraph verwendet werden.

White-Box Tests

- Ziel:
  Alle möglichen Pfade einmal zu durchlaufen.

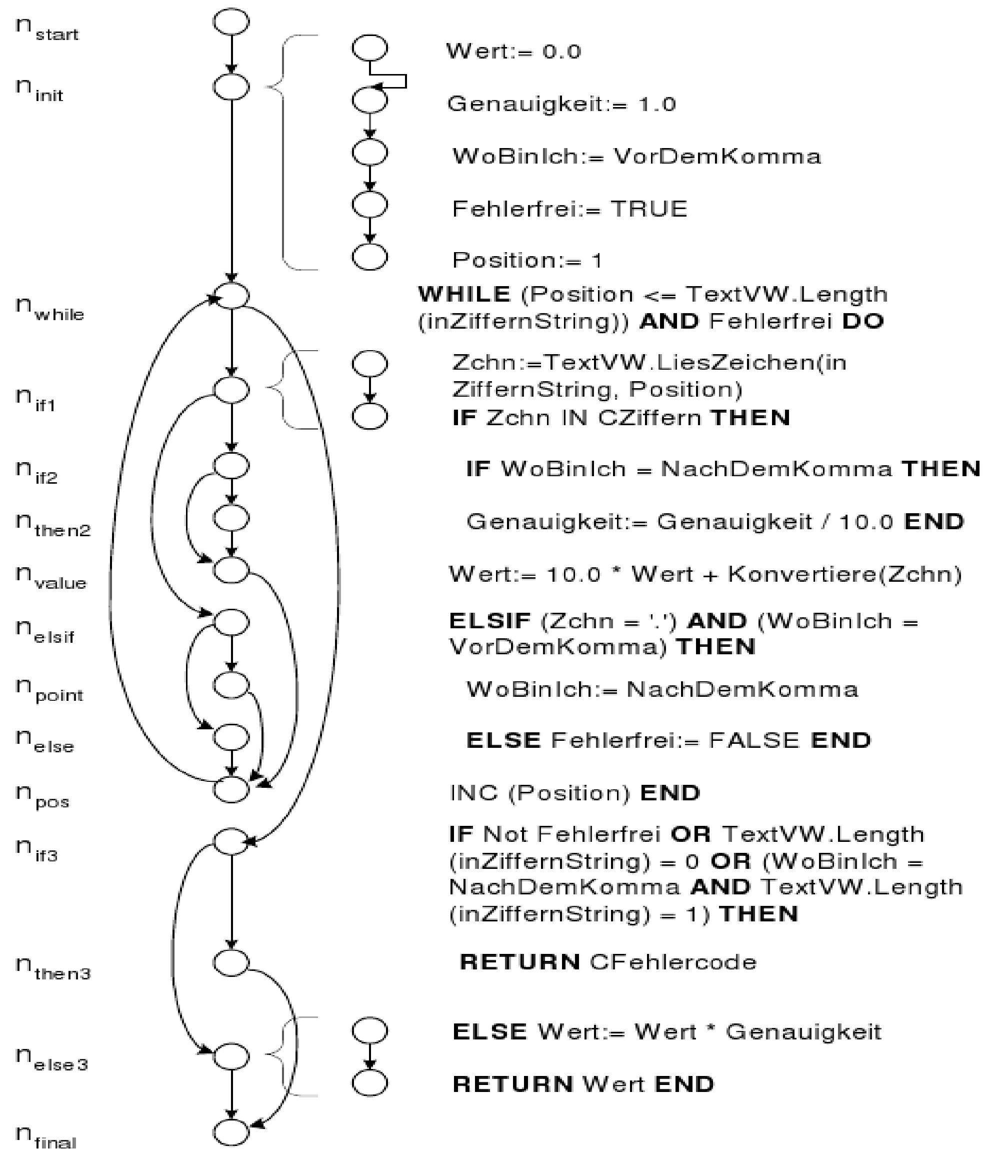
- Problem:
  Dies ist nur für kleine Beispiele möglich.

- Abhilfe:
  Es werden Näherungsverfahren verwendet, die nicht ALLE Pfade durchlaufen, aber dennoch viele Fehler finden sollen.

Beispiel

```
PROCEDURE WerteZiffernfolgeAus (inZiffernString : TText) : REAL;
TYPE    TWoBinIch         = (VorDemKomma, NachDemKomma);
        TZiffern          = SET OF ['0' .. '9'];
CONST   CFehlercode       = -1.0
        CZiffern          = TZiffern {0,1,2,3,4,5,6,7,8,9};
VAR     Zchn              : Char;
        Wert,
        Genauigkeit       : REAL;
        Position          : CARDINAL;
        WoBinIch          : TWoBinIch;
        Fehlerfrei        : BOOLEAN;
BEGIN
      Wert:= 0.0;
      Genauigkeit:= 1.0;
      WoBinIch:= VorDemKomma;
      Fehlerfrei:= TRUE;
      Position:=1;
      WHILE (Position <= TextVW.Length (inZiffernString)) AND Fehlerfrei DO
            Zchn:=TextVW.LiesZeichen(inZiffernString, Position)
            IF Zchn IN CZiffern THEN
                  IF WoBinIch = NachDemKomma THEN
                        Genauigkeit:= Genauigkeit / 10.0
                  END; (* IF *)
                  Wert:= 10.0 * Wert + Konvertiere(Zchn)
            ELSIF (Zchn = '.') AND (WoBinIch = VorDemKomma) THEN
                  WoBinIch:= NachDemKomma
            ELSE Fehlerfrei:= FALSE
            END; (* IF *)
            INC (Position)
      END; (* WHILE *)
      IF    NOT Fehlerfrei
      OR TextVW.Length (inZiffernString) = 0   (* leerer String *)
      OR (WoBinIch = NachDemKomma AND TextVW.Length(inZiffernString) = 1)   (* nur '.' *)
      THEN
            RETURN Cfehlercode
      ELSE
            Wert:= Wert * Genauigkeit;
            RETURN Wert
      END; (* IF *)
END WerteZiffernfolgeAus;
```

Kontrollflussgraph



White-Box Tests

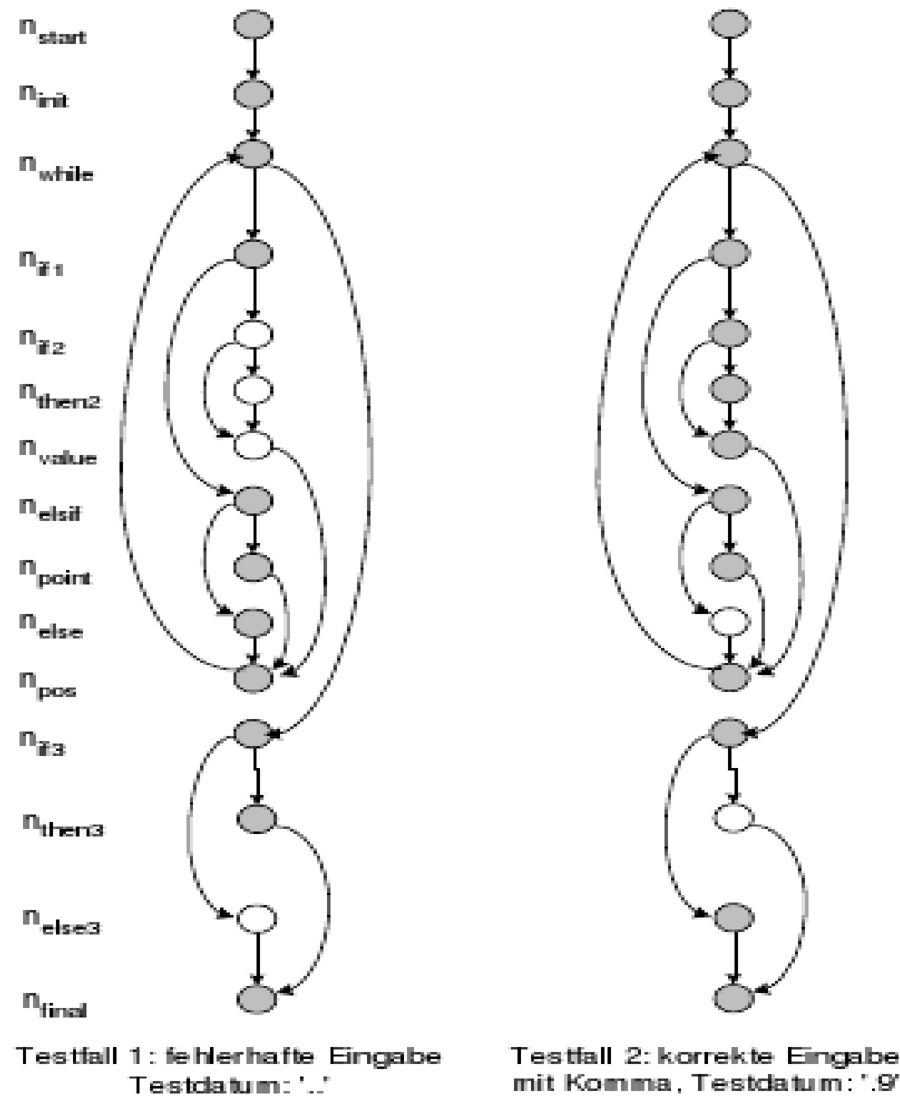
- Die wichtigsten Kriterien für die kontrollflussbezogenen Verfahren sind:
  - **Anweisungsüberdeckung:**
    Alle Anweisungen sollen mindestens einmal ausgeführt werden.
  - **Zweigüberdeckung:**
    Alle durch Selektion (if) oder Iteration (while) bedingten Verzweigungen sind mindestens einmal zu verfolgen.
  - **Bedingungsüberdeckung:**
    Bedingungen in Schleifen und Auswahlkonstruktionen werden getestet.
  - **Pfadüberdeckung:**
    Wie bereits erwähnt, sollen alle Pfade einmal durchlaufen werden. (Meist nicht möglich!)

Anweisungsüberdeckung: C_0-Test

- Der C_0 -Test ist die einfachste kontrollflussorientierte Testtechnik.
- Ziel: alle Anweisungen des Programms einmal ausführen, d.h. alle Knoten des Kontrollflussgraphen mindestens einmal besuchen.
- Testmaß: Überdeckungsgrad C_0

  C_0 = Anzahl besuchter Knoten / Anzahl aller Knoten
- Ziel: $C_0 = 1$
- Die Testfälle müssen so gewählt werden, dass die Anweisungsüberdeckung erreicht wird.

Beispiel



Bewertung C_0-Test

- Vorteile:
  - Einfach
  - geringe Anzahl von Eingabedaten für vollständige Überdeckung
  - nicht-ausführbare Programmteile werden erkannt

- Nachteile:
  - Überwiegen!
  - schlechte Testqualität, weniger als 20% der Fehler lassen sich entdecken
  - logische Aspekte werden nicht geprüft
  - Nicht alle Kanten werden besucht.
  - Datenabhängigkeiten zwischen Programmteilen werden nicht beachtet.

Zweigüberdeckung: C_1-Test

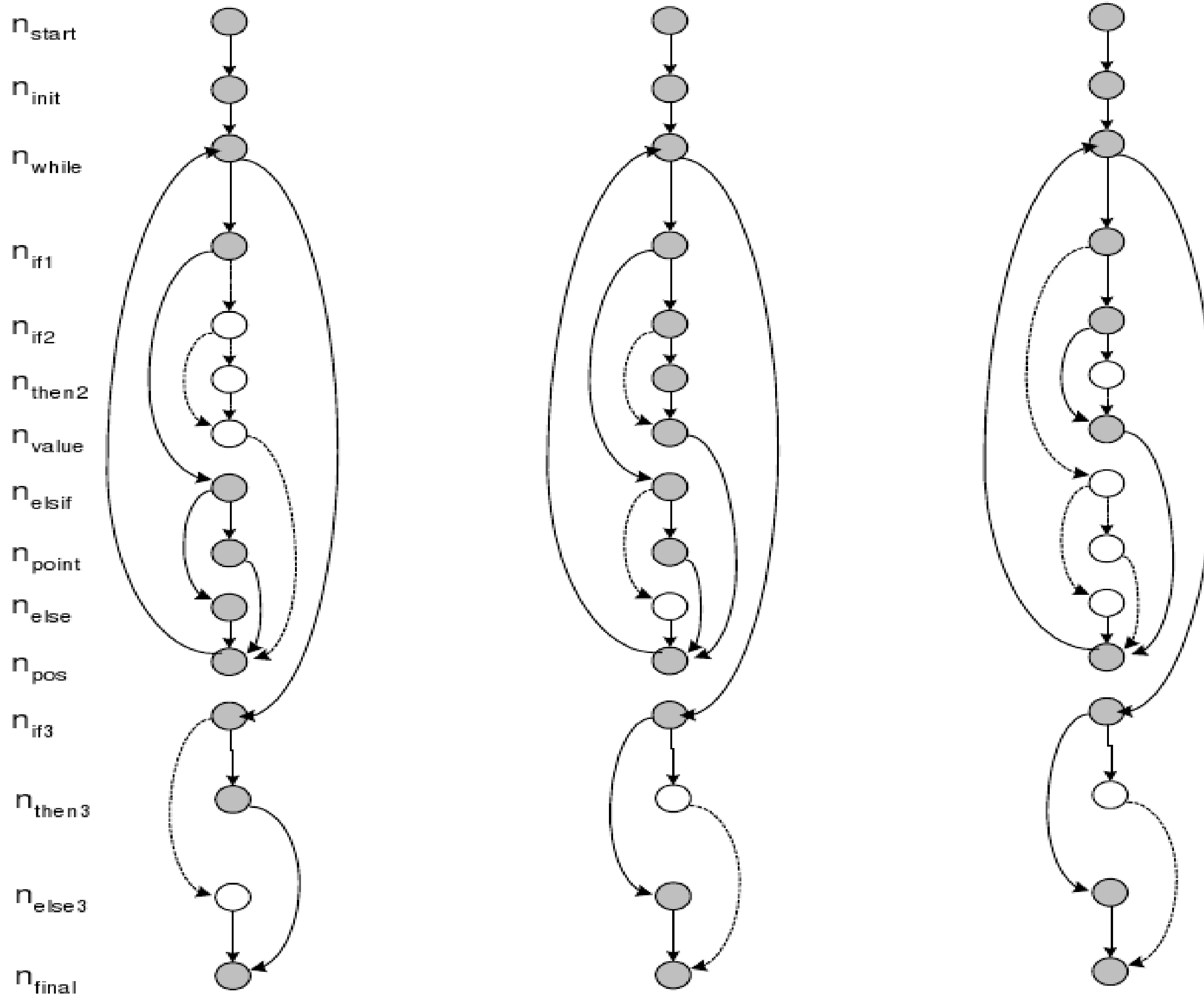
- Der C_1 -Test
  - konzentriert sich auf die Kanten des Kontrollflussgraphen
  - gilt allgemein als das minimale Testkriterium
  - enthält den C_0 -Test (Anweisungsüberdeckung) komplett.

- Testmaß: Überdeckungsgrad C_1

  C_1 = Anzahl besuchter Zweige / Anzahl aller Zweige

- Ziel: $C_1 = 1$

Beispiel



Testfall 1: fehlerhafte Eingabe
Testdatum: '..'

Testfall 2: korrekte Eingabe
mit Komma, Testdatum: '.9'

Testfall 3: korrekte Eingabe mit Ziffer
vor dem Komma, Testdatum: '9'

Bewertung C_1-Test

- Vorteile:
  - Nicht-ausführbare Programmzweige und etwa 25% der Fehler (vor allem Kontrollflussfehler) werden gefunden.

- Nachteile:
  - (Daten-/ Berechnungsfehler werden nicht entdeckt)
  - fehlende Zweige werden nicht automatisch entdeckt
  - Schleifen werden nicht ausreichend getestet
  - Kombination von Zweigen und Existenz von komplexen Bedingungen wird nicht getestet.

Bedingungsüberdeckung: C_2-Test

- Beim C_2-Test werden die Bedingungen in den Schleifen- und Auswahlkonstrukten werden zur Definition von Tests verwendet.
- Im Gegensatz zur Zweigüberdeckung werden alle Werte einzeln betrachtet! (nicht nur das Ergebnis)
- Man unterscheidet atomare Prädikate und verknüpfte Prädikate:
  - atomar: `Fehlerfrei` oder `(Zchn = '.')`
  - verknüpft: `(Zchn = '.') AND (WoBinIch = VorDemKomma)`
- Alle atomaren Prädikate müssen je einmal den Wert TRUE und FALSE erhalten

Beispiel

atomare Prädikate		Eingabedaten			
		'9'	'z'	'.9'	''
1	Position <= Length (inZiffernstring)	T, F	T, F	T, F	F
2	Fehlerfrei	T	T, F	T	T
3	Zchn in CZiffern	T	F	T	-
4	WoBinIch = NachDemKomma	F	-	T	-
5	Zchn = '.'	-	F	T	-
6	WoBinIch = VorDemKomma	-	T	T, F	-
(	Not Fehlerfrei (wird durch Fehlerfrei abgedeckt))				
7	Length (inZiffernString) = 0	F	F	F	T
8	Length (inZiffernString) = 1	T	T	F	-