Softwaretechnik

Systematisches Testen

Burkhardt Renz

Fachbereich MNI
Technische Hochschule Mittelhessen

Sommersemester 2012

Inhalt

Was ist Testen?

Klassische Aussage

Testing is the process of executing a program with the intent of finding errors.
Glenford J. Myers: The Art of Software Testing (1979)

Diskussion

- Begriffe: Testfall, Testpunkt, Testablauf
- Finden *effektiver* Testfälle
- Vergleich zu Techniken des Code-Reviews

Fragestellung

Welche Teilmenge aller möglichen Testfälle hat die höchste Wahrscheinlichkeit bisher nicht entdeckte Fehler zu finden?

Prinzipien des Testens I

1. A necessary part of a test case is a definition of the expected outcome or result.
2. A programmer should avoid attempting to test his or her own program.
3. A programming organization should not test its own programs.
4. Thorougly inspect the result of each test.

Diskussion

Prinzip 2 & 3 – Unittests und Test-First-Ansatz

Prinzipien des Testens II

5. Test cases must be written for invalid and unexpected, as well as valid and expected input conditions.
6. Examing a program to see if it does not do what it is supposed to do is only half of the battle. The other half is seeing whether the program does what it is not supposed to do.
7. Avoid throw-away test cases unless the program is truly a throw-away program.
8. Do not plan a testing effort under the tacit assumption that no errors will be found.
9. The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.

Arten des Testens

White-Box-Test

- Interne Struktur des Programms ist bekannt.
- Testfälle werden aus dem Kontroll- oder Datenfluss des Programms abgeleitet.
- Dynamischer Strukturtest, auch Glass-Box-Test genannt.

Black-Box-Test

- Interne Struktur des Programms ist nicht bekannt.
- Testfälle werden aus der Spezifikation hergeleitet.
- Funktionaler Test.

White-Box-Test – Überblick

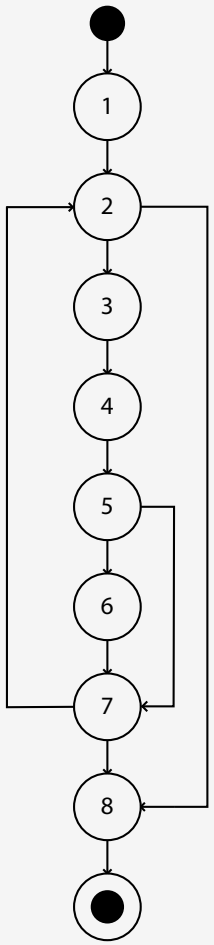
- Kontrollflussorientierter Test
  - Anweisungsüberdeckung
  - Zweigüberdeckung
  - Pfadüberdeckung
  - Bedingungsüberdeckung
- Datenflussorientierter Test
  - Defs/Uses-Verfahren

Black-Box-Test – Überblick

- Äquivalenzklassen
- Grenzwertanalyse
- Test spezieller Werte
- Erraten von Fehlern

Inhalt

Beispiel eines Kontrollfluss-Graphen




```
stmt

while ( cond_A )

{

  stmt_A;

  if ( cond_B )

    stmt_B;

}

stmt_C;
```

Kontrollfluss-Graph

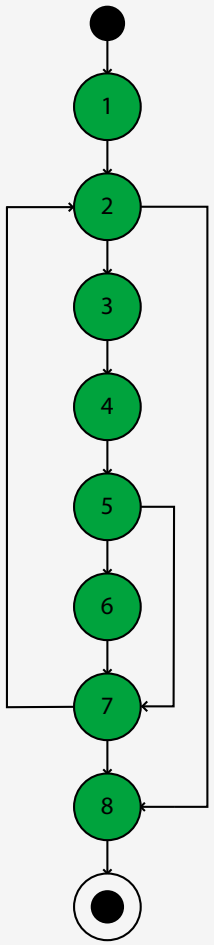
Knoten – Anweisungen/Zeilen des Programms

Kanten – Sprünge

Beispiel:

```
stmt;
while ( cond_A )
{
  stmt_A;
  if ( cond_B )
    stmt_B;
}
stmt_C;
```

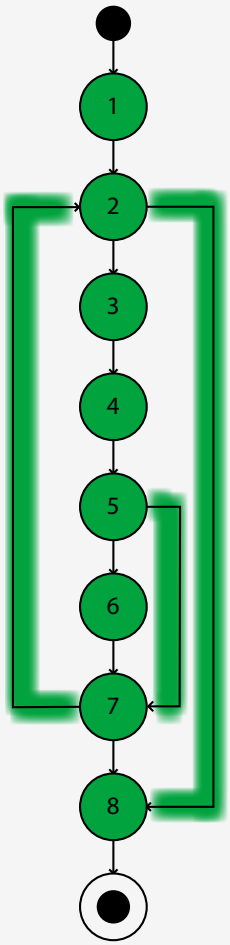
Anweisungsüberdeckung




```
stmt

while ( cond_A )

{

  stmt_A;

  if ( cond_B )

    stmt_B;

}

stmt_C;
```

- auch C_0-Test genannt (C für coverage).
- Schreibe Testfälle, so dass jede Anweisung mindestens einmal ausgeführt wird.
- Entdeckt „toten“ Code.
- Prüft Schleifen nicht.
- Prüft Bedingungen unzureichend.

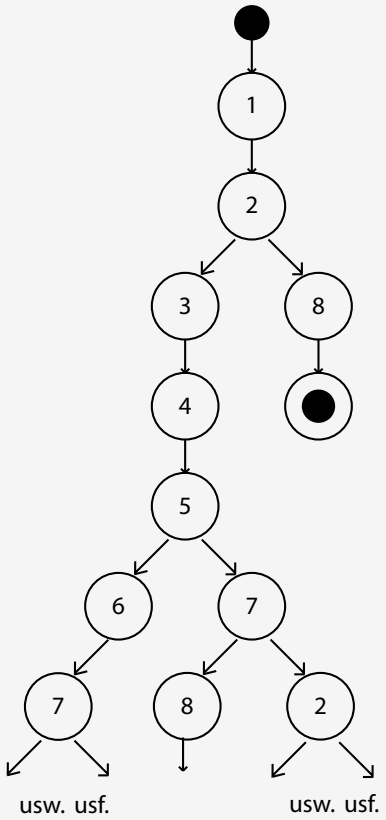
Zweigüberdeckung




```
stmt

while ( cond_A )

{

  stmt_A;

  if ( cond_B )

    stmt_B;

}

stmt_C;
```

- auch C_1-Test genannt.
- Schreibe Testfälle, so dass jeder Zweig mindestens einmal durchlaufen wird.
- Erweitert C_0-Test.
- Testet nicht Kombinationen von Zweigen
- Unzureichend bei komplexen Bedingungen.
- Schleifen werden nicht ausreichend getestet: terminiert eine Schleife stets?

Pfadüberdeckung



- Testfälle so dass alle möglichen Pfade durchlaufen werden.
- Problem 1: Je nach Konstruktion der Bedingungen ist nicht jeder denkbare Pfad durchführbar.
- Problem 2: Schleifen können zu „unendlich“ vielen Pfaden führen.
- Problem 3: Fehler, die von bestimmten Daten abhängen (datensensitive Fehler) werden nicht gefunden.

Boundary-Interior-Pfadtest

Teste Schleifen so:

- Boundary-Test
  - alle Pfade, die den Schleifenkörper nicht betreten
  - alle Pfade, die die Schleife betreten, aber nicht wiederholen
- Interior-Test
  - alle Pfade, so dass die Schleife wenigstens einmal wiederholt wird

Einfache Bedingungsüberdeckung

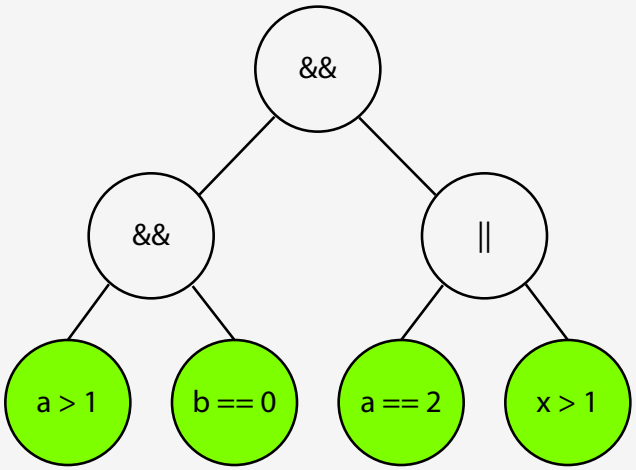
So viele Testfälle, dass jede atomare Bedingung

- mindestens einmal `true` und
- mindestens einmal `false` ist.

Beispiel

```
if (( a > 1 ) && ( b == 0 )) &&
   (( a == 2 ) || ( x > 1 )))
```

Beispiel - Syntaxbaum und Testfälle



	Testfall 1 a=2, b=1, x=1	Testfall 2 a=1, b=0, x=2
a>1	true	false
b==0	false	true
a==2	true	false
x>1	false	true

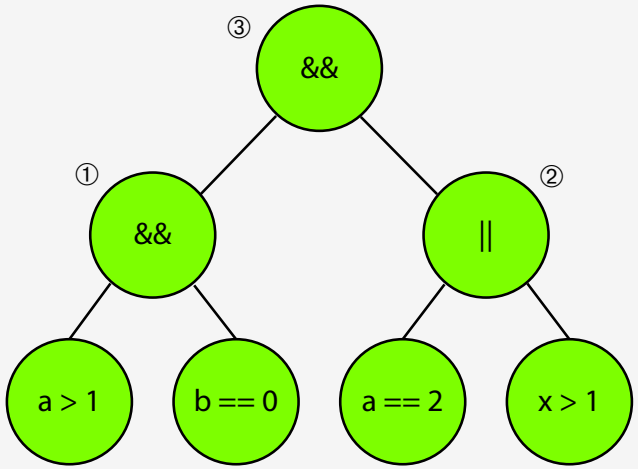
aber:
(a>1) && (b==0) in beiden Fällen false
(a==2) || (x>1) in beiden Fällen true

Minimale Mehrfach-Bedingungsüberdeckung

So viele Testfälle, dass jede Bedingung

- mindestens einmal `true` und
- mindestens einmal `false` ist.

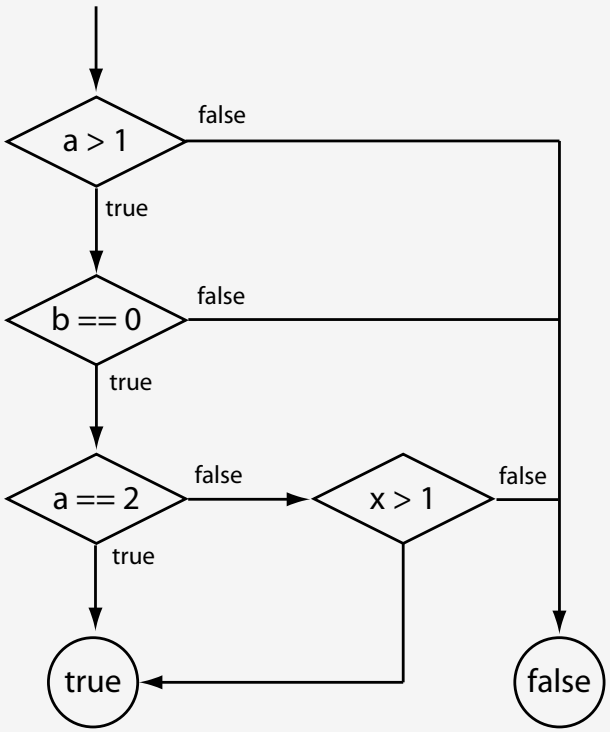
Unser Beispiel

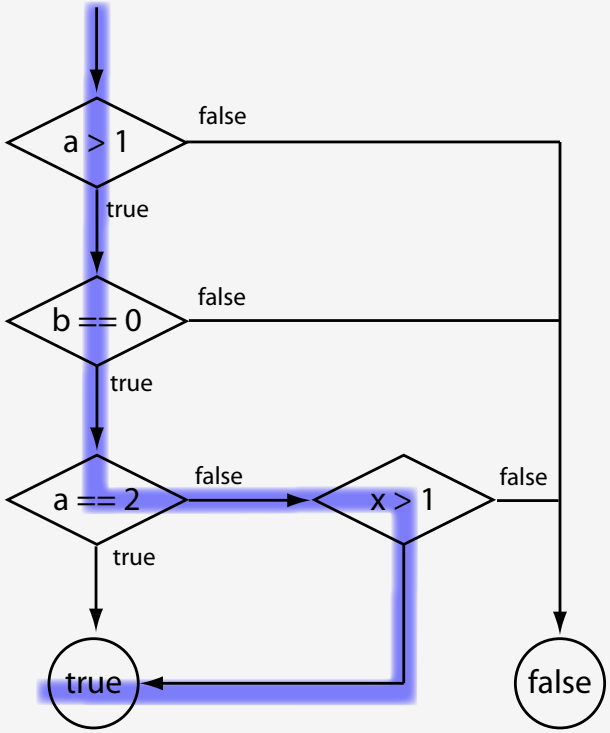


Testfälle für das Beispiel

	Testfall 1 a=2, b=1, x=1	Testfall 2 a=1, b=0, x=2	Testfall 3 a=2, b=0, x=2	Testfall 4 a=3, b=0, x=0
a>1	true	false	true	true
b==0	false	true	true	true
a==2	true	false	true	false
x>1	false	true	true	false
①	false	false	true	true
②	true	true	true	false
③	false	false	true	false

Diskussion





Der blau unterlegte Pfad wird bei keinem der bisherigen Testfälle durchlaufen.

Mehrfach-Bedingungsüberdeckung

So viele Testfälle, dass alle Kombinationen der Wahrheitswerte der atomaren Bedingungen vorkommen.

Diskussion

- n atomare Bedingungen ➜ 2^n Kombinationen.
- Darunter sind evtl. Kombinationen, die im Programm gar nicht auftreten können.
- Denkfehler in den Bedingungen werden auch dadurch nicht entdeckt.

Defs/Uses-Verfahren

Testfälle für alle Pfade zwischen einer Wertzuweisung und der Verwendung des Werts

Definition:

- Zuweisung (linke Seite)
- Lesen von Eingabe
- Schleifenvariablen

Verwendung:

- Zuweisung (rechte Seite)
- Berechnung
- Ausgabe von Werten
- Bedingung
- Parameter

Ziel: Mittelweg zwischen Zweigüberdeckung und Pfadüberdeckung.

Fazit White-Box-Test

Vorteile

- Zwingt Tester zur Analyse des Codes
- Findet Fehler in „verstecktem“ Code
- Hilft Programmierkenntnisse zu verbreiten

Nachteile

- Sehr aufwendig
- Tester vergisst evtl. Fälle, die im Code nicht vorkommen, wiederholt somit Denkfehler des Entwicklers
- Wird geprüft, ob die Funktion/Klasse das Richtige tut?

Inhalt

Äquivalenzklassenbildung

Testwerte finden

- Testfälle aus der Spezifikation ableiten
- Äquivalenzklassen bilden
- Mindestens einen Testfall mit einem Repräsentanten jeder Äquivalenzklasse

Beispiel

„Geben Sie den Monat als Zahl 1-12 ein“
Äquivalenzklassen:
`[1..12]` – gültige Werte
`[<= 0]` – ungültige Werte
`[>= 13]` – ungültige Werte
Repräsentanten:
`5, -3, 25`

Grenzwertanalyse

Vorgehen

- Testwerte an den Grenzen der Äquivalenzklassen wählen
- Maximum, Minimum, typische Werte, Fehlerwerte

an unserem Beispiel

Äquivalenzklassen	`[<=0]`	`[1..12]`	`[>=13]`
Testwerte	`0`	`1, 12`	`13`

Weitere Methoden

Test spezieller Werte

- Der Wert `0` oder `null`
- Spezielle Zeichen in Strings: Sonderzeichen, Unicode
- Fehlende Informationen
- Leere Datencontainer

Erraten von Fehlern

- Fußnoten in Fußnoten
- Riesige Eingabedateien
- Binärdaten, wo Texte erwartet werden
- ...

Fazit Black-Box-Test

Vorteile

- Spezifikation ist die Grundlage
- Unabhängigkeit vom Denken der Entwickler
- kann auch Mängel in der Spezifikation entdecken

Nachteile

- Überdeckung im Code kann nicht festgestellt werden
- Abgrenzung der Effekte der Umgebung gegenüber Effekten des Prüflings manchmal schwierig

Vorgehen in der Praxis

- Unittest durch die Entwickler
- Black-Box-Test unabhängig
- White-Box-Test unabhängig
- Ergebnisse fließen in Unittests ein
- wieder von vorne: Regressionstest im Unittest

Literatur

Bodo Igler, Nadja Krümmel: *Anleitung Testen*
Institut für SoftwareArchitektur
`http://homepages.thm.de/~hg11260/mat/testentwurf.pdf`