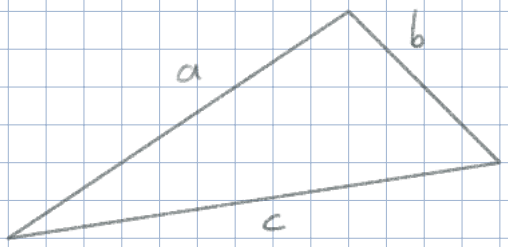
Ein Programm liest drei Zahlen a, b & c und soll feststellen ob sie die Seiten

- eines gleichseitigen Dreiecks
- eines gleichschenkligen Dreiecks
- eines rechtwinkligen Dreiecks
- eines gültigen Dreiecks
- eines ungültigen Dreiecks

beiden



Gültige Dreiecke

- gleichseitig: 3, 3, 3
- gleichschenklig: 5, 5, 3
- rechtwinklig: 3, 4, 3
- sonstiges: 3, 5, 7
- Permutationen davon

Ungültige Dreiecke

- $a + b < c$
- $a + b = c$
- neg. Seitenlängen:
- nur 2 Seitenlängen
- alle Seiten null

Mit welchen Eingaben testen?

> Testen heißt,
> ein Programm mit der Absicht,
> Fehler zu finden

Wann ist ein Test erfolgreich?

- Ein Test ist erfolgreich, wenn er Fehler findet
- Ein Test ist erfolglos, wenn er keine Fehler findet

Testen ist ein destruktiver Prozess

- Jedes Programm hat Fehler – Ziel ist es, sie zu finden
- Implementierer ist zu wenig destruktiv, daher sollte der Tester nicht der Implementierer sein
- Richtige Einstellung: versuchen, das Programm zu Fall zu bringen

Wert des nächsten gefundenen Fehlers

- Mit jedem gefundenen Fehler wird das Programm verteuert!
- Korrektheitsbeweis ist ein unerreichbares Ziel ⇒ frustriert den Tester
- "Noch einen Fehler zu finden" ist ein erreichbares Ziel, dem sich der Tester gewachsen fühlt.

Testen

- Fehler finden
- destruktive Tätigkeit

Debuggen

- Fehlerursache lokalisieren
- Korrektur überlegen
- Folgen der Änderung überlegen
- Änderungen durchführen
- Konstruktive Tätigkeit ⇒ Aufgabe des Implementierers

Statisches Testen

- Analyse des Programms, ohne es zu laufen zu lassen (manuell o. auto)
- Kann schon einsetzen, bevor das Programm fertig ist
- Techniken
  - Verifikation
  - Code-Inspektion / Walkthrough
  - Überprüfen von Codierstandards, Programmierstil, Komplexität

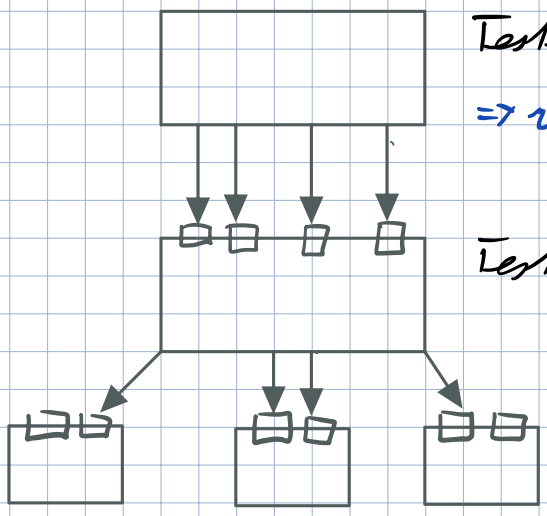
Dynamisches Testen

- Programm laufen lassen
- Geht erst nach der Implementierung
- Techniken
  - Unit Test / Integrationstest / Systemtest / Abnahmetest
  - Profiling & Performance-Analyse

Schrittweise Lesen eines Programms im Team

Warum ist das eine gute Idee?

- Programme werden zu wenig gelesen
  In anderen technischen Disziplinen unvorstellbar, dass ein Fabrikat nicht von anderen Personen kontrolliert wird
- Programmierer glaubt, über seinen Quellcode keine Rechenschaft schuldig zu sein
  "Hauptsache es funktioniert"
  Keine Rechenschaft => schlampiger Stil
- Programmierer entdeckt selbst Fehler, wenn er sein Programm anderen erklärt
  Muss alle seine Entscheidungen überdenken, rechtfertigen und erklären



Testtreiber
=> rufen Funktionalität auf (Des Testobj)

Testobjekt

Teststubs stellen Funktionalität zur Verfügung, die das Testobj. braucht.

Black Box Test

Auch: Unit Test, Funktionaler Test, Schnittstellentest

Testet des Input/Output - Verhalten

- Quellcode wird ignoriert (Black Box)
- Testfälle werden nut aus der Schnittstelle bzw. der Spezifikation abgeleitet
- Gelieferte Ausgabe wird mit erwarteter Ausgabe verglichen
- Testfälle werden so gewählt, dass:
  - 1 Testfall für jede gültige Eingabe (bzw. Eingabekategorie)
  - möglichst viele Testfälle für ungültige Eingaben
- Tester spielt mit dem Testobjekt

Vorteile

- Tester muss sich nicht in die Implementierung einarbeiten
- Tester ist unvoreingenommen (macht nicht die gleichen Denkfehler wie der Implementierer)

Nachteile

- Kann nie vollständig sein (unendlich viele Eingabekombinationen)
- Man weiß nicht, ob jede Anweisung (jeder Zweig) durchlaufen wurde

White Box Test

Auch: Strukturtest

Versucht, den Quellcode mit Testfällen abzudecken

- Quellcode des Mosuls wird analysiert (White Box)
- Testfälle werden gewählt, dass
  - jede Anweisung zumindest einmal ausgeführt wird.
  - jede Bedingung zumindest einmal true/false ergibt
  - jeder Pfad zumindest einmal ausgeführt

} "Abdeckungsarten"

Vorteile

- Man kann dafür sorgen, dass das Programm keinen ungelsenen Code enthält
- Kenntnis des Programms zeigt, wo besondersintensiv getestet werden muss

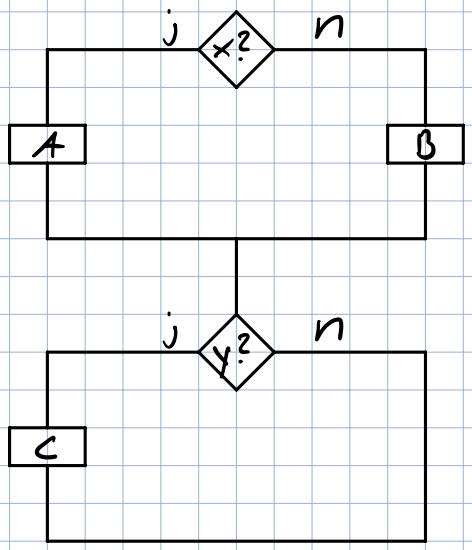
Nachteile

- Aufwändiger als Black Box Test
- Man übersieht Eingabefälle, die vom Programm nicht berücksichtigt werden

Anweisungsabdeckung

(statement coverage)

Jede Anweisung muss mindestens einmal ausgeführt werden.



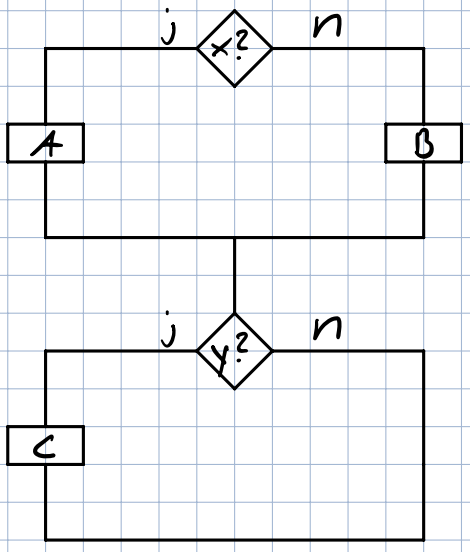
x & y : AC
!x & y : BC

schwächste Abdeckungsart!

Bedinungsabdeckung

(branch coverage)

Jede Abfrage muss meindestens einmal true und false ergeben

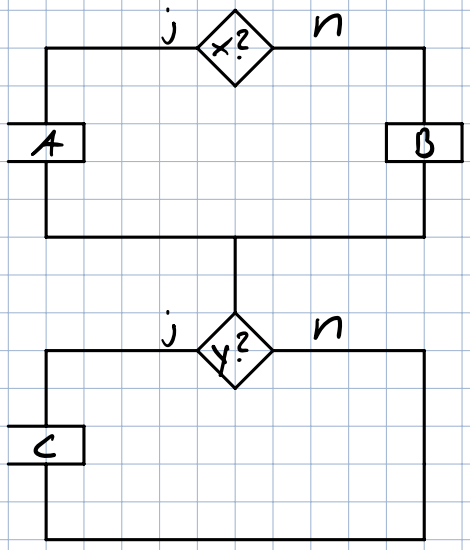


x & y : AC
!x & !y : B

Pfadabdeckung

(path coverage)

Jeder Pfad muss mindestens einmal durchlaufen werden



x & y: AC

!x & y: BC

x & !y: A

!x & !y: B

Stärkste Abdeckungsart