Softwaretechnik II

Sommersemester 2014

Grundlagen des Softwaretestens II

Stefan Berlik

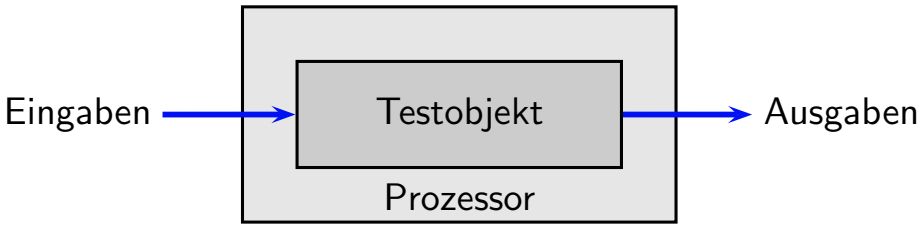
Fachgruppe Praktische Informatik
Fakultät IV, Department Elektrotechnik und Informatik
Universität Siegen

24. April 2014

Gliederung

Statischer Test vs. dynamischer Test

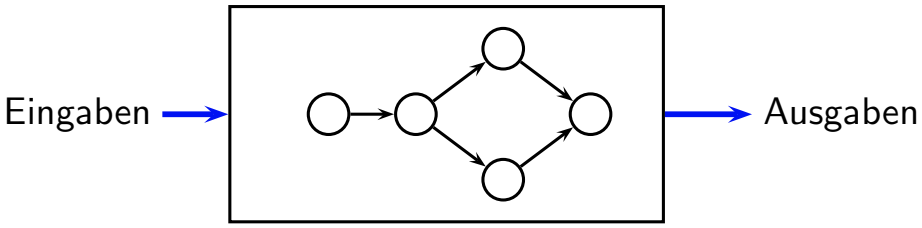
- Programme sind statische Beschreibungen dynamischer Prozesse
- Der **statische Test** prüft das Testobjekt an sich, d.h. dessen Beschreibung
  - Artefakte des Entwicklungsprozesses
    (informelle Texte, Modelle, formale Texte, Programmcode, . . .)
- Der **dynamische Test** prüft den aus der Beschreibung des Testobjekts resultierenden Prozess, d.h. dessen ‚Interpretation'
  - Das Testobjekt wird auf einem Prozessor **ausgeführt**
    → Bereitstellen von Eingangsdaten
    → Beobachten der Ausgangsdaten



Black-box Test vs. White-box Test (1)

- **Black-box Test**
  - Arbeitet **ohne** Kenntnis der Programmlogik

- **White-box Test**
  - Arbeitet **unter** Kenntnis der Programmlogik



Black-box Test vs. White-box Test (2)

- **Black-box Test**
  - Testobjekt ist ‚**undurchsichtig**', d.h. es liegen keine Informationen über den Programmtext und den inneren Aufbau vor
  - Verhalten des Testobjekts wird **von außen beobachtet** (Point of Observation (PoO) liegt außerhalb des Testobjekts)
  - **Ablauf** des Testobjektes wird nur **durch** die Wahl der **Eingaben gesteuert** (Point of Controll (PoC) liegt außerhalb des Testobjektes)
  - Testfälle basieren auf der Spezifikation bzw. den Anforderungen an das Testobjekt → **‚funktionales' Testverfahren**

Black-box Test vs. White-box Test (3)

- **White-box Test**
  - Testobjekt ist ‚**durchsichtig**', d.h. es wird auf den Programmtext und den inneren Aufbau zurückgegriffen
  - Bei Ausführung der Testfälle wird der innere **Ablauf im Testobjekt analysiert** (Point of Observation (PoO) liegt innerhalb des Testobjekts)
  - **Eingriff in den Ablauf** im Testobjekt ist **möglich**, z.B. wenn durch Negativtests die zu provozierende Fehlbedienung über die Komponentenschnittstelle nicht auslösbar ist (Point of Controll (PoC) kann innerhalb des Testobjektes liegen)
  - Ergänzende Testfälle können auf Grund der Programmstruktur des Testobjektes gewonnen werden → **‚strukturelles' Testverfahren**

Begriffe

- Black-box Test
  - Arbeitet **ohne** Kenntnis der Interna des Testobjekts
- Black-box Verfahren
  - Testverfahren, die zur Herleitung oder Auswahl der Testfälle keine Information über die innere Struktur des Testobjekts benötigen

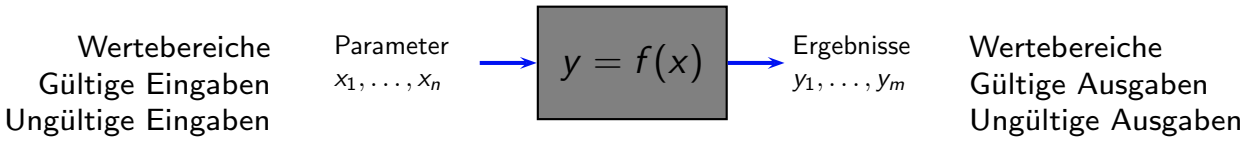
- Woher stammen die Eingaben?
- Wie können die Ausgaben überprüft werden?

Funktionale Testverfahren

- **Funktionaler Test**
  - Dynamischer Test, bei dem die Testfälle auf Basis der funktionalen Spezifikation des Testobjekts hergeleitet werden
  - Vollständigkeit der Prüfung (**Überdeckungsgrad**) wird ebenfalls anhand der funktionalen Spezifikation bewertet

- **Funktionalität**
  - Spezifiziert das Verhalten, das das System erbringen muss, d.h. beschreibt ‚**was**' das System leisten soll
  - Umsetzung ist Voraussetzung dafür, dass das System überhaupt einsetzbar ist
  - Merkmale der Funktionalität nach ISO 9126: **Angemessenheit, Richtigkeit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit**

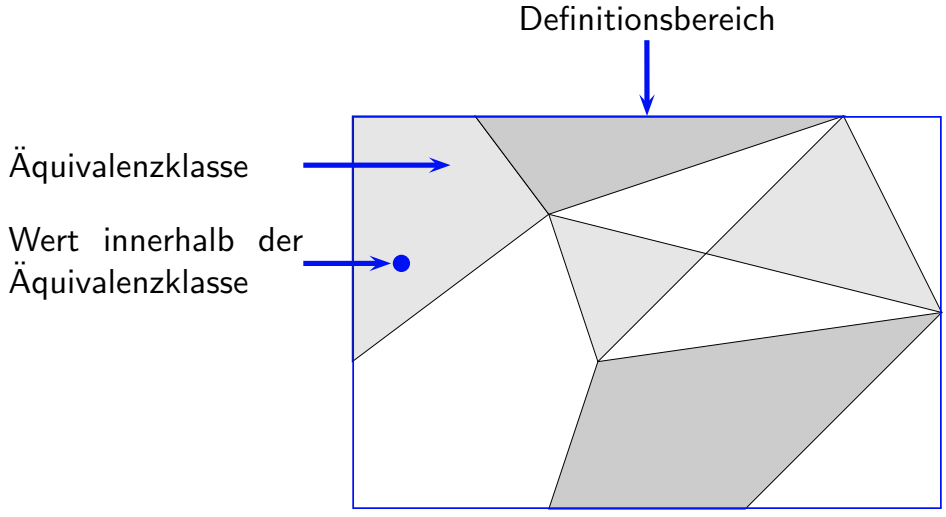
Funktionale Testfall- und Testdatenermittlung



- Äquivalenzklassenbildung
  - Repräsentative Eingaben
  - Gültige Eingaben
  - Ungültige Eingaben
- Grenzwertanalyse
  - Wertebereiche
  - Wertebereichsgrenzen
- Zustandstest
  - Komplexe (innere) Zustände
  - Zustandsübergänge

Idee der Äquivalenzklassenbildung

- **Partitioniere den Definitionsbereich der Ein- und Ausgaben in Teilbereiche**, so dass alle Werte eines Teilbereichs äquivalentes Verhalten des Prüflings ergeben
  - Die Teilbereiche werden als **Äquivalenzklassen** (ÄK) bezeichnet
  - Deckt ein Wert der ÄK einen **Fehler** auf, wird erwartet dass auch jeder andere Wert der ÄK diesen Fehler aufdeckt → 100% Fehlerrate
  - Deckt ein Wert der ÄK **keinen Fehler** auf, wird erwartet dass auch kein anderer Wert der ÄK einen Fehler aufdeckt → 0% Fehlerrate



Vorgehensweise

- Aufstellen der **Definitionsbereiche** der Ein-und Ausgaben aus der Spezifikation
- Arten von **Beschränkungen** bei der **Äquivalenzklassenbildung**
  - Beschränkung die einen **Wertebereich** spezifiziert
  - Beschränkung die eine **Anzahl von Werten** spezifiziert
  - Beschränkung die eine **Menge von Werten** spezifiziert, die ggf. **unterschiedlich behandelt** werden
    insgesamt eine ungültige ÄK
  - Beschränkung die eine **zwingend zu erfüllende Situation** spezifiziert
- Werden **Werte** einer ÄK vermutlich **nicht gleichwertig** behandelt, wird die **ÄK in kleinere ÄKs aufgespalten**

Beispiele (1)

- Beschränkung die einen **Wertebereich** spezifiziert
  $\rightarrow$ Eine gültige und zwei ungültige ÄK

Beispiel

In der **Spezifikation** des Testobjekts ist festgelegt, dass ganzzahlige Eingabewerte zwischen 1 und 100 möglich sind.

Wertebereich der Eingabe:	$\mathbf{1 \leq x \leq 100}$
Gültige Äquivalenzklasse:	$\mathbf{1 \leq x \leq 100}$
Ungültige Äquivalenzklassen:	$\mathbf{x < 1}$, $\mathbf{x > 100}$ und **NaN** (not a number)

Beispiele (2)

- Beschränkung die eine **Anzahl von Werten** spezifiziert
  → Eine gültige und zwei ungültige ÄK

Beispiel

Laut Spezifikation muss sich ein Mitglied eines Sportvereins mindestens einer Sportart zuordnen. Jedes Mitglied kann an maximal drei Sportarten aktiv teilnehmen.

Gültige Äquivalenzklasse:	$\mathbf{1 \leq x \leq 3}$ (1 bis 3 Sportarten)
Ungültige Äquivalenzklassen:	$\mathbf{x = 0}$ und $\mathbf{x > 3}$ (keiner bzw. mehr als 3 Sportarten zugeordnet)

Beispiele (3)

- Beschränkung die eine **Menge von Werten** spezifiziert, die ggf. **unterschiedlich behandelt** werden
  → für jeden Wert dieser Menge eine eigene gültige ÄK und zusätzlich insgesamt eine ungültige ÄK

Beispiel

Laut Spezifikation gibt es im Sportverein folgende Sportarten: Fußball, Hockey, Handball, Basketball und Volleyball.

Gültige Äquivalenzklassen:	Fußball, Hockey, Handball, Basketball, Volleyball
Ungültige Äquivalenzklassen:	alles andere, z.B. Badminton

Beispiele (4)

- Beschränkung die eine **zwingend zu erfüllende Situation** spezifiziert
  → eine gültige und eine ungültige ÄK

Beispiel

Laut Spezifikation erhält jedes Mitglied im Sportverein eine eindeutige Mitgliedsnummer. Diese beginnt mit dem ersten Buchstaben des Familiennamens des Mitglieds.

Gültige Äquivalenzklassen:	erstes Zeichen ein Buchstabe
Ungültige Äquivalenzklassen:	erstes Zeichen kein Buchstabe (z.B. eine Ziffer oder ein Sonderzeichen)

Testfälle für jeden Parameter tabellarisch notieren

- Eindeutige Kennzeichnung jeder Äquivalenzklasse (gÄK$_n$, uÄK$_n$)

	TF_1	TF_2	...	TF_n
gÄK$_1$	x			
gÄK$_2$		x		
...				
uÄK$_1$				
uÄK$_2$				
...				x

- **Pro Parameter** mindestens **zwei Äquivalenzklassen**
  - Eine mit **gültigen** Werten
  - Eine mit **ungültigen** Werten
- Bei n Parametern mit m_i Äquivalenzklassen ($i = 1, \ldots, n$) gibt es

$$\prod_{i=1,\ldots,n} m_i$$

unterschiedliche Kombinationen (Testfälle)

Anzahl der Testfälle per Heuristiken minimieren

- Testfälle aus allen **Repräsentanten kombinieren** und anschließend nach ‚Häufigkeit' sortieren (‚**Benutzungsprofile**')
  - Testfälle dann in dieser Reihenfolge priorisieren
  - Nur mit ‚**benutzungsrelevanten**' Testfällen testen
  - Testfälle bevorzugen, die **Grenzwerte** oder Grenzwert-Kombinationen enthalten
- Sicherstellen, dass **jeder Repräsentant** einer Äquivalenzklasse **mit jedem Repräsentanten** jeder anderen Äquivalenzklasse in einem Testfall zur Ausführung kommt
  - d.h. **paarweise Kombination** statt vollständiger Kombination
- **Minimalkriterium**: Mindestens ein Repräsentant jeder Äquivalenzklasse in mindestens einem Testfall
- Repräsentanten **ungültiger Äquivalenzklassen** nicht mit Repräsentanten anderer ungültiger Äquivalenzklassen kombinieren ($\rightarrow$ Fehlermaskierung)

Vor- und Nachteile der Äquivalenzklassenbildung

- **Vorteile**
  - Anzahl der Testfälle **kleiner** als bei unsystematischer Fehlersuche
  - Geeignet für Programme mit **vielen verschiedenen Ein- und Ausgabebedingungen**
- **Nachteile**
  - Betrachtet Bedingungen für **einzelne** Ein- oder Ausgabeparameter
  - Beachtung von **Wechselwirkungen** und **Abhängigkeiten** von Bedingungen **sehr aufwändig**
- **Empfehlung**
  - Zur Auswahl wirkungsvoller Testdaten: **Kombination** der ÄK-Bildung **mit fehlerorientierten** Verfahren, z.B. Grenzwertanalyse

Idee der Grenzwertanalyse

- **Werte die Grenzbereiche von Verzweigungs- und Schleifenbedingungen aus**, für welche eine Bedingung **gerade noch** oder **gerade nicht mehr zutrifft**
  - Diese **Fallunterscheidungen** sind fehlerträchtig ('off by one')
  - Testdaten, die solche **Grenzwerte prüfen**, decken Fehlerwirkungen mit höherer Wahrscheinlichkeit auf als Testdaten, die dies nicht tun
- Grenzen der ÄK testen
- Jeder ‚Rand' einer ÄK muss in einer Testdatenkombination vorkommen

Grundlagen der Grenzwertanalyse

- In der Regel werden der **Grenzwert** selbst, sowie die Werte **unmittelbar über bzw. unter** dem Grenzwert getestet
- **Atomare (geordnete) Bereiche** (integer, real, char)
  - Werte auf den Grenzen
  - Werte ‚rechts bzw. links neben' den Grenzen (ungültige Werte, kleiner bzw. größer als Grenze)
- **Mengenwertige Bereiche** (z.B. bei Datenstrukturen, Beziehungen)
  - Kleinste und größte gültige Anzahl
  - Zweitkleinste und zweitgrößte gültige Anzahl
  - Kleinste und größte ungültige Anzahl
- Fallen bei Äquivalenzklassen für geordnete Bereiche obere und untere Grenze zweier ÄK zusammen, dann auch die entsprechenden Testfälle

Grenzwertanalyse: Vorgehen (1)

- Grenzen des **Eingabebereichs**
  - Beispiel: abgeschlossener Bereich: $[-1.0; +1.0]$
    $\rightarrow$ Testdaten: -1.001; -1.0; +1.0; +1.001
  - Beispiel: offener Bereich: $]-1.0; +1.0[$
    $\rightarrow$ Testdaten: -1.0; -0.999; +0.999; +1.0
- Grenzen der **erlaubten Anzahl von Eingabewerten**
  - Beispiel: Eingabedatei mit 1 bis 365 Sätzen
    $\rightarrow$ Testfälle: 0, 1, 365, 366 Sätze
- Grenzen des **Ausgabebereichs**
  - Beispiel: Programm errechnet Beitrag, der zwischen €0,00 und €600 liegt
    $\rightarrow$ Testfälle: €0,00; €600 und möglichst auch für Beiträge $<$€0; $>$€600
- Grenzen der erlaubten Anzahl von **Ausgabewerten**
  - Beispiel: Ausgabe von 1 bis 4 Daten
    $\rightarrow$ Testfälle: Für 0, 1, 4 und 5 Ausgabewerte

Grenzwertanalyse: Vorgehen (2)

- **Erstes und letztes Element** bei geordneten Mengen beachten
  - Beispiel: Sequentielle Datei, lineare Liste, Tabelle
- Bei komplexen Datenstrukturen **leere Mengen** testen
  - Beispiel: Leere Liste, Null-Matrix
- Bei numerischen Berechnungen
  - **eng zusammen** und **weit auseinander liegende Werte** wählen

Grenzwertanalyse: Vor- und Nachteile

- **Vorteile**
  - An den Grenzen von Äquivalenzklassen sind **häufiger Fehler** zu finden als innerhalb dieser Klassen
  - Effizient in **Kombination** mit anderen Verfahren, die Freiheitsgrade in der Wahl der Testdaten lassen
  - „Die Grenzwertanalyse ist bei richtiger Anwendung eine der nützlichsten Methoden für den Testfallentwurf“
    [Myers, Glenford J.: Methodisches Testen von Programmen Oldenbourg, 7. Auflage, 2001]

- **Nachteile**
  - Rezepte für die **Auswahl** von Testdaten schwierig anzugeben
  - Bestimmung aller **relevanten** Grenzwerte schwierig
  - **Kreativität** zur Findung erfolgreicher Testdaten gefordert
  - Oft nicht effizient genug angewendet, da sie **zu simpel erscheint**

Black-box Tests: Bewertung

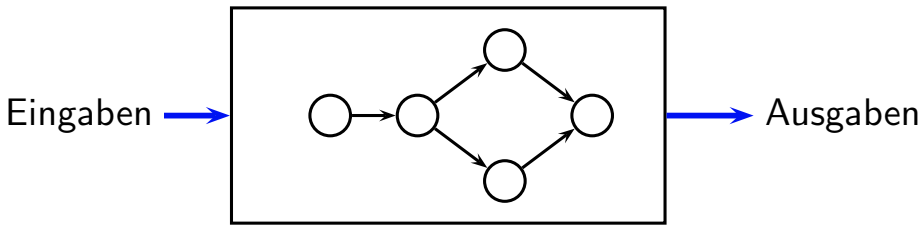
- Grundlage aller Black-box Verfahren sind die Anforderungen sowie die Spezifikation des Systems bzw. der einzelnen Komponenten und ihres Zusammenwirkens
  - **Fehlerhafte Anforderungen** oder Spezifikationen werden **nicht erkannt**
  - Testobjekt verhält sich so, wie die Spezifikation (bzw. die Anforderungen) es fordert, auch wenn diese fehlerhaft ist
- **Nicht geforderte Funktionalität** wird nicht erkannt
  - Zusätzliche Funktionen sind weder spezifiziert noch vom Kunden gefordert
  - Testfälle, die diese zusätzlichen Funktionen zur Ausführung bringen, werden – wenn überhaupt – nur zufällig durchgeführt
  - **Überdeckungskriterien** ausschließlich auf der Grundlage der Anforderungen bzw. der Spezifikation und nicht auf der Grundlage von nicht beschriebenen und nur vermuteten Funktionen

Black-box Tests: Bewertung (2)

- Im Mittelpunkt aller Black-box Verfahren steht die Prüfung der **Funktionalität** des Testobjektes
  - Das **korrekte Funktionieren** eines Softwaresystems hat **höchste Priorität** und somit sind auch Black-box Testverfahren stets einzusetzen

White-box Test

- White-box Test
  - Test unter Nutzung von Informationen über Interna des Testobjekts
- White-box Verfahren
  - Alle Verfahren, die zur **Herleitung oder Auswahl der Testfälle** Informationen über die innere Struktur des Testobjekts benötigen (auch **struktureller Test**)



- Woher stammen die Eingaben?
- Wie können die Ausgaben überprüft werden?

Prinzip des strukturellen Tests

- **Dynamischer** Test, bei dem die Testfälle unter Berücksichtigung der **Struktur** des Testobjekts hergeleitet werden und die Vollständigkeit der Prüfung (**Überdeckungsgrad**) anhand von **Strukturelementen** (z.B. Zweige, Pfade, Daten) bewertet wird
- Gesucht werden **‚fehleraufdeckende' Stichproben** der möglichen **Programmabläufe** und **Datenverwendungen**

Strukturelles Testen von Programmen

- **Kontrollflussbasiert**
  - Anweisungsüberdeckung (C0-Überdeckung, alle Knoten)
  - ...
  - Pfadüberdeckung (C∞-Überdeckung, alle Pfade)
- **Datenflussbasiert**
  - Alle Definitionen
  - ...
  - Kontextüberdeckung
- **Bedingungsbasiert**
  - Einfache Bedingungsüberdeckung
  - Mehrfachbedingungsüberdeckung
  - ...

Kontrollflussbezogene Überdeckungskriterien

- **Anweisungsüberdeckung** (statement coverage)
- **Zweigüberdeckung** (branch coverage)
- **Grenze-Inneres-Test** (boundary interior coverage)
- **Pfadüberdeckung** (path coverage)

Anweisungsüberdeckung (1)

- Dynamisches, kontrollflussbasiertes Testverfahren, das die mindestens einmalige **Ausführung aller Anweisungen** des Testobjekts fordert
- Auch als **C0-Maß** bezeichnet (C für engl. coverage = Überdeckung)

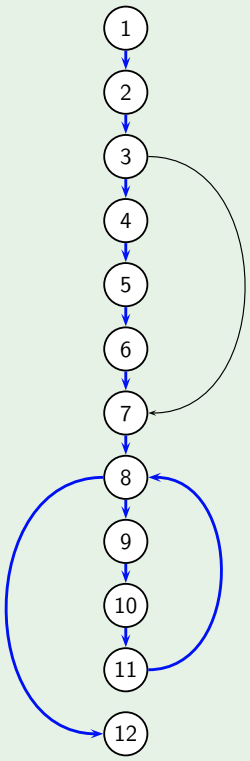
Definition

$$\text{C0-Überdeckung} = \frac{\text{Anzahl durchlaufene Anweisungen}}{\text{Gesamtzahl Anweisungen}} * 100\%$$

Anweisungsüberdeckung (2)

- Jeder **Testfall** wird anhand eines **Pfads** durch den Kontrollflussgraphen bestimmt
  - Bei dem Testfall müssen die auf dem Pfad liegenden Kanten des Graphen durchlaufen werden, d.h. die **Anweisungen** (**Knoten**) in der entsprechenden Reihenfolge zur **Ausführung** kommen.
  - Bei der Berechnung wird nur gezählt, ob eine **Anweisung** bei der Ausführung **überhaupt durchlaufen** wurde, die Häufigkeit der Ausführung spielt keine Rolle.
  - Für die einzelnen **Testfälle** sind, neben den **Vor- und Nachbedingungen**, auch die **erwarteten Ergebnisse** und das erwartete Verhalten des Testobjektes vorab zu bestimmen und danach mit dem **tatsächlichen Ergebnis** bzw. Verhalten zu **vergleichen**.
- Wenn der zuvor festgelegte **Überdeckungsgrad erreicht** ist, wird der **Test** als **ausreichend** angesehen und beendet

Beispiel (Anweisungsüberdeckung für ggt())




```
 1. public int ggt(int m, int n) {
    // pre: m > 0 and n > 0
    // post: return> 0 and
    // m@pre.mod(return) = 0 and
    //..
 2.   int r;
 3.   if (n > m) {
 4.     r = m;
 5.     m = n;
 6.     n = r;
      }
 7.   r = m % n;
 8.   while(r != 0) {
 9.     m = n;
10.     n = r;
11.     r = m % n;
      }
12.   return n;
    }
```

$\rightarrow \text{Pfad}_1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 8, 12)$

Diskussion der Anweisungsüberdeckung

- Die **C0-Überdeckung** ist ein in der Aussage **schwaches** Kriterium
  - Für die Anweisungsüberdeckung ist eine **leere** (`else`-) **Kante** (zwischen `if` und `endif`) ohne Bedeutung.
  - Möglicherweise **fehlende Anweisungen** in dem enthaltenden Programmteil werden **nicht erkannt**!

- Eine **100%ige Überdeckung** der Anweisungen ist in der Praxis (und auch Theorie) **nicht immer erreichbar**
  - Z.B. können **Ausnahmebedingungen** im Programm vorkommen, die während der Testphase nur mit erheblichem Aufwand oder gar nicht herzustellen sind
  - Möglicherweise auch Hinweis auf nicht erreichbare Anweisungen (‚**deadcode**‘) (ggf. statische Analyse durchführen)

Zweigüberdeckung (1)

- Dynamisches, kontrollflussbasiertes Testverfahren, das die **Überdeckung aller Zweige** des Kontrollflussgraphen einer Komponente fordert

- Auch als **C1-Maß** bezeichnet

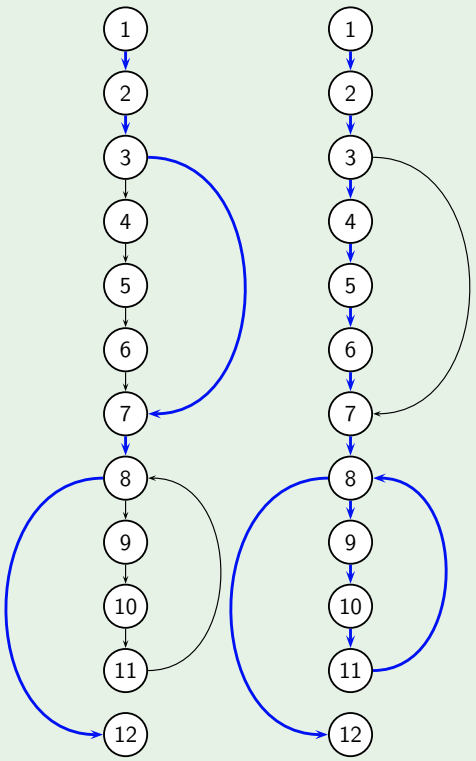
Definition

$$\text{C1-Überdeckung} = \frac{\text{Anzahl durchlaufene Zweige}}{\text{Gesamtzahl Zweige}} * 100\%$$

Zweigüberdeckung (2)

- Jeder **Testfall** wird wieder anhand eines **Pfads** durch den Kontrollflussgraphen bestimmt
  - Bei dem **Testfall** müssen die auf dem Pfad liegenden **Kanten** des Graphen durchlaufen werden.
  - Die **Häufigkeit** der Ausführung eines Zweiges spielt auch hier bei der Berechnung **keine Rolle**.
  - Für die einzelnen **Testfälle** sind auch hier neben den **Vor- und Nachbedingungen** wieder die **erwarteten Ergebnisse** und das erwartete Verhalten des Testobjektes vorab zu bestimmen und danach mit dem **tatsächlichen Ergebnis** bzw. Verhalten zu **vergleichen**.

Beispiel (Zweigüberdeckung für ggt())




```
1.  public int ggt(int m, int n) {
    // pre: m > 0 and n > 0
    // post: return> 0 and
    // m@pre.mod(return) = 0 and
    //..
2.    int r;
3.    if (n > m) {
4.      r = m;
5.      m = n;
6.      n = r;
      }
7.    r = m % n;
8.    while(r != 0) {
9.      m = n;
10.     n = r;
11.     r = m % n;
      }
12.   return n;
    }
```

$\rightarrow \text{Pfad}_1 = (1, 2, 3, 7, 8, 12)$ (neu)

$\rightarrow \text{Pfad}_2 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 8, 12)$

Diskussion der Zweigüberdeckung (1)

- Die **C1-Überdeckung** ist ein in der Aussage **stärkeres** Kriterium
  - Die Zweigüberdeckung verlangt, dass bei einer **Verzweigung** des Kontrollflusses (bei einer Bedingung) beide bzw. bei einer case-Anweisung alle Möglichkeiten und bei **Schleifen** der **Rücksprung** zum Schleifenanfang zu berücksichtigen sind.
  - Mit der Zweigüberdeckung können fehlende Anweisungen in **leeren Zweigen** im Gegensatz zur Anweisungsüberdeckung erkannt werden!
- Die Zweigüberdeckung wird oft als **minimales** Kriterium in der Praxis angewendet

Diskussion der Zweigüberdeckung (2)

- Die einzelnen **Zweige** werden **unabhängig** voneinander betrachtet und es werden **keine** bestimmten **Kombinationen** der einzelnen Zweige gefordert
- Die **Abfolge** der Zweige erfolgt relativ **willkürlich**
- Ergo:
  - Eine **100%ige Zweigüberdeckung** ist **anzustreben**, ist aber – wie bei der Anweisungsüberdeckung – in der Praxis (und auch Theorie) **nicht immer erreichbar**.
  - Nur wenn neben allen Anweisungen auch **jede** mögliche **Verzweigung** des Kontrollflusses in der Testphase berücksichtigt wird, kann der **Test** als **ausreichend** eingestuft werden.

Pfadüberdeckung (1)

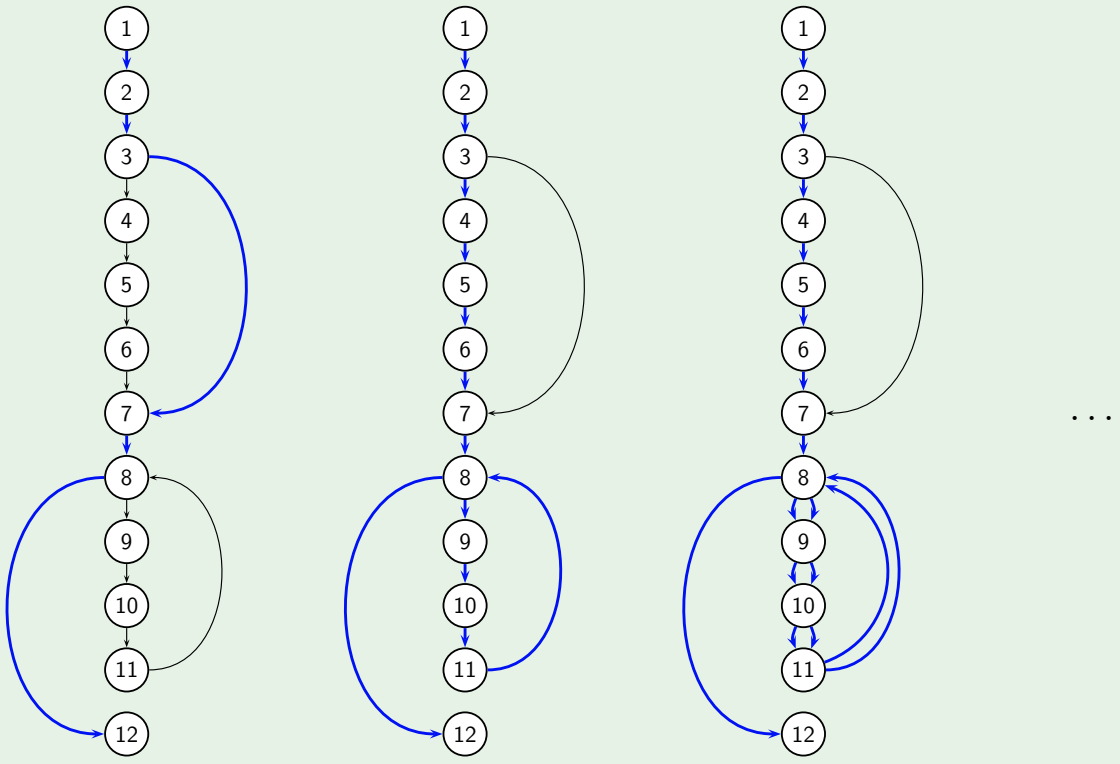
- Dynamisches, kontrollflussbasiertes Testverfahren, das die mindestens einmalige **Ausführung aller Pfade** des Kontrollflussgraphen einer Komponente fordert
- Auch als **C∞-Maß** bezeichnet

Definition

$$\text{C}\infty\text{-Überdeckung} = \frac{\text{Anzahl durchlaufener Pfade}}{\text{Gesamtzahl Pfade}} * 100\%$$

- Bei **zyklischen** Kontrollflussgraphen potenziell **unendlich** viele Pfade
  - **Obere Grenzen** für die Anzahl ggf. aus Spezifikation oder aus technischen Einschränkungen
- In der Praxis nicht erreichbar, eher als **theoretisches Vergleichsmaß** wichtig

Beispiel (Pfadüberdeckung für ggt())



Kontrollflussbasierter Test und objektorientierte Software

- Sowohl die **Anweisungs-** als auch die **Zweigüberdeckung** ist für objektorientierte Systeme nur **unzureichend** geeignet
  - **Methoden** in den Klassen sind normalerweise wenig umfangreich und **von geringer Komplexität**
  - Die geforderten C0-und C1-Überdeckungen lassen sich dann mit **wenig Aufwand** erreichen
- Die **Komplexität** bei objektorientierten Systemen ist meist **in** den **Beziehungen** zwischen den Klassen verborgen
- Wenn eine **Werkzeugunterstützung** zur Ermittlung der **Überdeckungswerte** vorhanden ist, kann diese allerdings genutzt werden, um **nicht aufgerufene Methoden** oder Programmteile zu erkennen

Bedingungen in Programmen (und Spezifikationen)

- **Wahrheitswerte**: `true, false` (oft auch 1 oder 0)
- **Atomare Teilbedingungen**
  - Variablen vom Typ `boolean`
  - Operationen mit Rückgabewert vom Typ `boolean`
  - Vergleichsoperationen (z.B. `flag; isEmpty(); size > 0`)
- **Zusammengesetzte Bedingungen**
  - Verknüpfen atomare Teilbedingungen mit booleschen Operatoren
  - Basis-Operatoren sind `und` ($\wedge$) , `oder` ($\vee$), `nicht` ($\neg$)
- **Entscheidungen** sind zusammengesetzte Bedingungen, die den Programmablauf steuern

Test der Bedingungen

- Bei der **Zweigüberdeckung** wird **ausschließlich** der ermittelte **Ergebnis-Wahrheitswert** einer Bedingung berücksichtigt
  - Anhand dieses Wertes wird entschieden, welche Verzweigung im Kontrollflussgraphen verfolgt wird bzw. welche Anweisung als nächste im Programm zur Ausführung kommt
- **Problem**: Setzt sich eine Bedingung aus **mehreren Teilbedingungen** zusammen, die über logische Operatoren miteinander verknüpft sind, so muss im Test die strukturelle **Komplexität der Bedingung** berücksichtigt werden
- Hierbei werden **unterschiedliche Anforderungen** und damit auch Abstufungen der Testintensität mit Berücksichtigung der **zusammengesetzten** Bedingungen unterschieden

Bedingungsüberdeckungstests

- Als **Überdeckungskriterien** werden Verhältnisse zwischen den bereits erreichten und allen geforderten Wahrheitswerten der (Teil-)Bedingungen gebildet
- Bei den Verfahren, welche die Komplexität der Bedingungen im Programmtext des Testobjekts in den Mittelpunkt der Prüfung stellen, ist es sinnvoll, eine **vollständige Prüfung (100% Überdeckung)** anzustreben

Im Folgenden

- **Einfache** Bedingungsüberdeckung (*branch condition testing*)
- **Mehrfach**bedingungsüberdeckung (*branch condition combination testing*)
- **Minimale Mehrfach**bedingungsüberdeckung (*modified branch condition decision testing*)

Einfache Bedingungsüberdeckung

- Auch als **C2-Maß** bezeichnet (Engl.: *branch condition testing*)
- Kontrollflussbasiertes, dynamisches Testverfahren, das die **Überdeckung** der **atomaren Teilbedingungen/Ausdrücke (a.A.)** einer Entscheidung mit ‚wahr' und ‚falsch' fordert.
  - *Teste jeden atomaren Ausdruck einmal zu wahr und einmal zu falsch*
- Bei n atomaren Ausdrücken **mindestens 2, höchstens 2n** Testfälle

Definition

$$\text{C2-Überdeckung} = \frac{\text{Anzahl zu wahr und falsch getesteter a.A.}}{\text{Gesamtzahl a.A.}} * 100\%$$

- Achtung: Die einfache Bedingungsüberdeckung ist ein **schwächeres Kriterium als die Anweisungs- oder auch Zweigüberdeckung**, da **nicht** verlangt ist, dass unterschiedliche Wahrheitswerte bei der Auswertung der **gesamten** Bedingung im Test zu berücksichtigen sind.

Beispiele (Einfache Bedingungsüberdeckung)

- Teste jeden atomaren Ausdruck einmal zu wahr und einmal zu falsch

- 2 Atomare Ausdrücke
  → 2 Testfälle

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

- 3 Atomare Ausdrücke
  → 2 Testfälle

A	B	C	$A \wedge B \wedge C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Mehrfachbedingungsüberdeckung

- Auch als **C3-Maß** bezeichnet (Engl.: *branch condition combination testing*)
- Teste **jede Kombination** der Wahrheitswerte aller atomaren Ausdrücke
- Bei n atomaren Ausdrücken (a.A.) ist die Testfall-Anzahl $= 2^n$
  - Wächst exponentiell mit der Anzahl atomarer Ausdrücke

Definition

$$\text{C3-Überdeckung} = \frac{\text{Anzahl getestete Kombinationen a.A.}}{2^{\text{Gesamtzahl a.A.}}} * 100\%$$

- Bei der Auswertung der Gesamtbedingung ergeben sich auch beide Wahrheitswerte
  - Die **Mehrfachbedingungsüberdeckung erfüllt** somit auch die Kriterien der Anweisungs- und **Zweigüberdeckung**
  - Sie ist ein **umfassenderes Kriterium**, da sie auch die Komplexität bei zusammengesetzten Bedingungen berücksichtigt
- **Problem**: Manche Kombinationen sind nicht durch konkrete Testfälle realisierbar, z.B. wenn Teilbedingungen voneinander abhängig sind

Beispiele (Mehrfachbedingungsüberdeckung)

- Teste jede Kombination der Wahrheitswerte aller atomaren Ausdrücke

- 2 Atomare Ausdrücke
  $\rightarrow$ 4 Testfälle

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

- 3 Atomare Ausdrücke
  $\rightarrow$ 8 Testfälle

A	B	C	$A \wedge B \wedge C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Minimale Mehrfachbedingungsüberdeckung

- Engl.: *modified branch condition decision testing*
- Teste **jede Kombination** von Wahrheitswerten, bei denen die **Änderung** des Wahrheitswertes **eines atomaren Ausdrucks** den Wahrheitswert des **zusammengesetzten Ausdrucks ändern** kann (**MM-Kombinationen**)

Definition

$$\text{MMB-Überdeckung} = \frac{\text{Anzahl getestete MM-Kombinationen}}{\text{Gesamtzahl MM-Kombinationen}} * 100\%$$

- **Gesamtzahl der MM-Kombinationen** ist **bei reinen and-** bzw. **or-Bedingungen** mit n atomaren Ausdrücken nur $n + 1$

Beispiele (Minimale Mehrfachbedingungsüberdeckung)

- Teste jede Kombination von Wahrheitswerten, bei denen die Änderung des Wahrheitswertes eines atomaren Ausdrucks den Wahrheitswert des zusammengesetzten Ausdrucks ändern kann

- 2 Atomare Ausdrücke
  → 3 Testfälle

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

- 3 Atomare Ausdrücke
  → 4 Testfälle

A	B	C	$A \wedge B \wedge C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Bewertung der Bedingungstestverfahren (1)

- Auf die intensive **Prüfung** oder Aufteilung von komplexen Bedingungen kann möglicherweise ganz **verzichtet** werden, **wenn** diese vor dem dynamischen Test einem **Code-Review** unterzogen werden und deren Korrektheit dort nachgewiesen wird
- Es kann sinnvoll sein, **komplexe** zusammengesetzte **Bedingungen in** verschachtelte, **einfache Abfragen aufzuteilen** und für diese Abfolge von Abfragen dann einen **Zweigüberdeckungstest** durchzuführen

Bewertung der Bedingungstestverfahren (2)

- Ein **Nachteil** der Bedingungsüberdeckungen ist, dass sie boolesche Ausdrücke beispielsweise **nur innerhalb einer** `if`-**Anweisung** prüfen
  - Manchmal wird z.B. nicht erkannt, dass die `if`-Bedingung eine aus mehreren Teilbedingungen zusammengesetzte ist und die minimale Mehrfachbedingungsüberdeckung angewendet werden sollte
  - Beispiel: `flag = a || (b && c); if(flag) {...}`
  - → **Alle booleschen Ausdrücke** für die Erstellung der Testfälle **heranziehen**

Bewertung der Bedingungstestverfahren (3)

- **Problem**: **Messung der Überdeckung** der Teilbedingungen
  - Einige Programmiersprachen und **Compiler verkürzen die Auswertung** von booleschen Ausdrücken, sobald das Ergebnis feststeht
  - Beispiel: Ist bei einer `and`-**Verknüfung** von zwei Teilbedingungen für eine Teilbedingung der Wert `false` ermittelt, dann ist die Gesamtbedingung ebenfalls `false`, egal welchen Wert die zweite Teilbedingung liefert
  - Einige **Compiler ändern auch die Reihenfolge** der Auswertung in Abhängigkeit von den booleschen Operatoren, um möglichst schnell ein Endergebnis zu erhalten und die weiteren Teilbedingungen nicht auswerten zu müssen

Datenflussbasierter Test

- Dynamischer Test, bei dem die Testfälle unter Berücksichtigung der **Datenverwendungen** im Testobjekt hergeleitet werden und die Vollständigkeit der Prüfung (**Überdeckungsgrad**) anhand der Datenverwendung bewertet wird
- Test bezüglich der **Variablen/Objektverwendung**
- **Wertzuweisung, zustandsverändernd**
  - Beispiel: `r = m` oder `r = 5`
  - Definition [definitional use, **def**(r)]
- **Benutzung in Ausdrücken, zustandserhaltend**
  - Beispiel: `r = m % n` oder `r = op(m,n)`
  - Berechnungsreferenz [computational use, **c-use**(m,n)]
- **Benutzung in Bedingungen, zustandserhaltend**
  - Beispiel: `while(r != 0)` oder `if (r == 0)`
  - Entscheidungsreferenz [Predicative use, **p-use**(r)]

Datenflussbezogene Überdeckungskriterien

- Kriterium **alle Definitionen (all-defs)**: Jede Definition mindestens einmal ohne dazwischenliegendes erneutes def in einem c-use oder p-use verwendet
- Kriterium **alle DR-Interaktionen**: jedes Paar def/ref (ohne dazwischenliegendes erneutes def) auf irgendeinem Weg ausführen
- Weitere typische Kriterien
  - **alle B-Referenzen (all-c-uses)**
  - **alle E-Referenzen (all-p-uses)**
  - **3-DR-Interaktionen**
  - **Kontextüberdeckung**

Bewertung des datenflussbasierten Tests

- Testverfahren unterscheiden sich nach dem Aufwand:
  - Alle Definitionen ist am einfachsten (O(n)Testdaten)
  - Alle E-Referenzen, alle B-Referenzen und Kontextüberdeckung sind am aufwandigsten (O(n^2)Testdaten bei n Segmenten im Programmteil)
- Testverfahren unterscheiden sich nach der Fehleraufdeckungsfähigkeit:
  - Alle B-Referenzen fand bspw. bis zu 88% aller Berechnungsfehler
  - Alle E-Referenzen fand bspw. 100% aller Bereichsfehler
  - folgende Fehler werden dagegen schlecht aufgedeckt:
    - fehlende Pfade
    - Bereichsfehler durch falsch platzierte Anweisungen und falsche arithmetische Operatoren
    - Berechnungsfehler bei speziellen Werten
- Ca. 9% aller Fehler wurden nur mit datenflussbezogenen Methoden gefunden

Kontrollfragen

Nach dieser Vorlesungseinheit sollten Sie folgende Fragen beantworten können

- Was ist der wesentliche Unterschied zwischen dynamischen und statischen Tests?
- Wie unterscheiden sich Black-box Verfahren und White-box Verfahren zur Testfallermittlung?
- Was versteht man unter Äquivalenzklassenbildung?
- Welches Ziel verfolgt die Grenzwertanalyse?
- Wie arbeiten kontrollflussbasierte Tests?
- Worin unterscheiden sich Anweisungs- und Zweigüberdeckung?
- Worauf zielt die Bedingungsüberdeckung ab?
- Worin unterscheiden sich die einfache Bedingungsüberdeckung und die (minimale) Mehrfachbedingungsüberdeckung?