

---

# Vorlesung „ Methoden des Software Engineering“

Block D „Qualitätssicherung“

## **Black-Box-Test und White-Box-Test**

Martin Wirsing

Einheit D.2, 21.12.2004

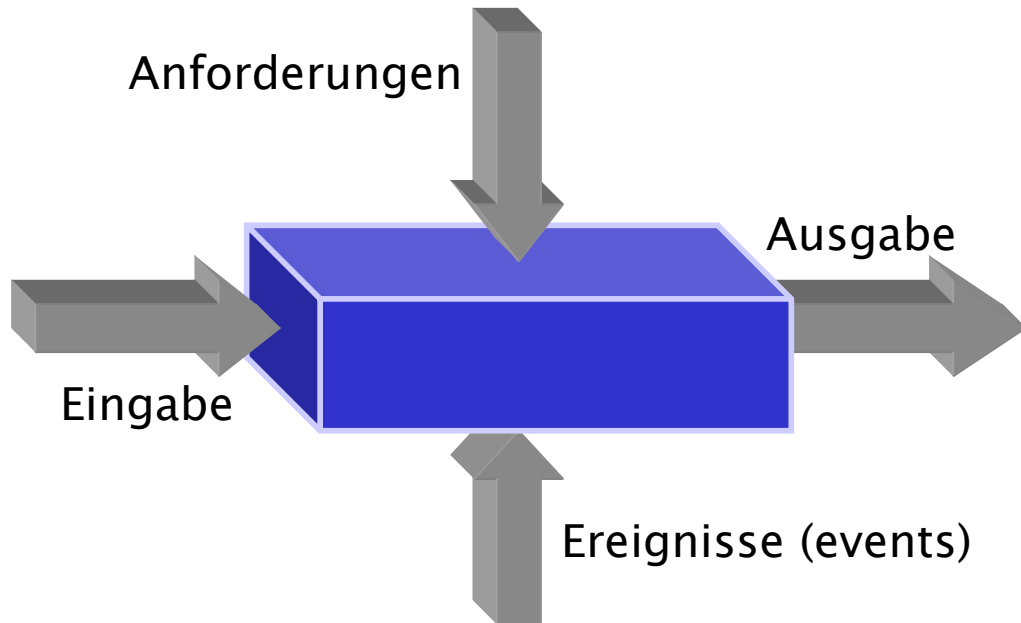
Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## **Ziele**

- 
- Techniken des Black-Box-Testens und des White-Box-Testens kennen lernen

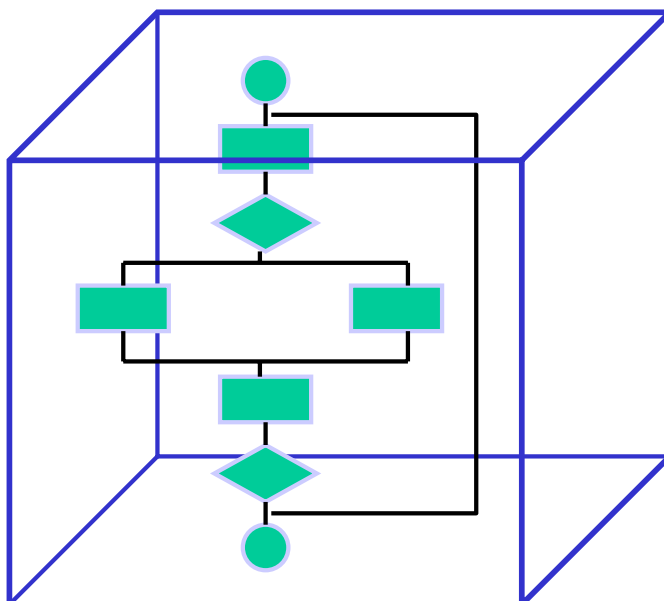
# Black-Box-Test

---



# White-Box-Test

---



... Ziel ist, dass alle Anweisungen, Bedingungen und Pfade mindestens einmal ausgeführt wurden...

# Testen

---

Black-Box-Test	White-Box-Test
<ul style="list-style-type: none"> <li>• Testfälle gehen von der Spezifikation aus</li> <li>• Interna des Testobjekts sind bei der Ermittlung der Testfälle unbekannt</li> <li>• Testüberdeckung wird an Hand des spezifizierten Ein/Ausgabeverhaltens gemessen</li> </ul>	<ul style="list-style-type: none"> <li>• Testfälle ausgehend von der Struktur des Testobjekt</li> <li>• Testfälle werden vom Entwickler beschrieben</li> <li>• Testüberdeckung wird an Hand des Codes gemessen</li> </ul>

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Testen

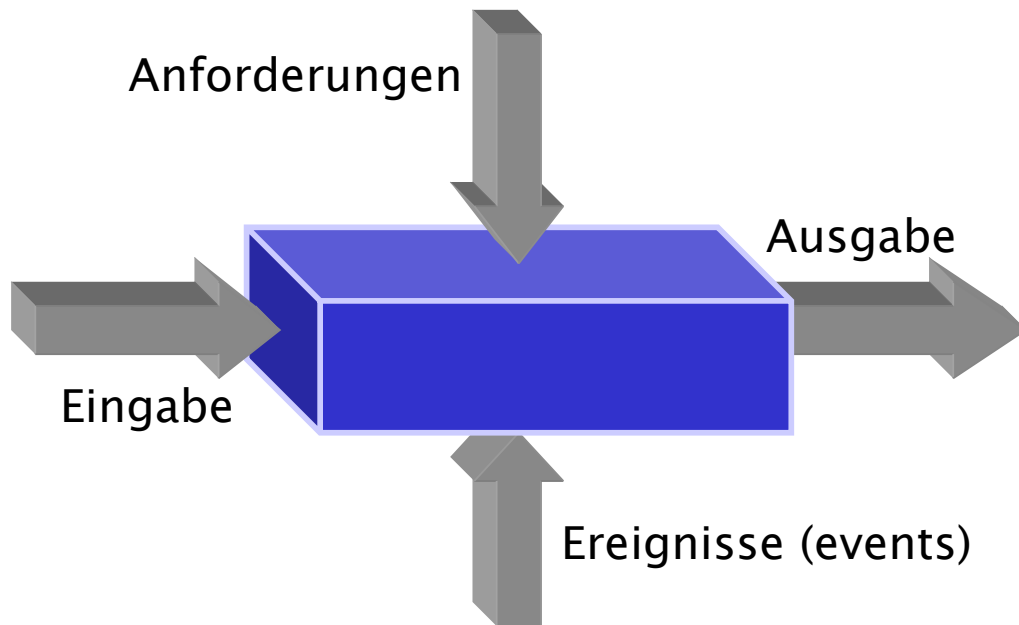
---

- **Methoden des Black-Box-Test**
  - Äquivalenzklassenmethode
  - Methode der Grenzwertanalyse
  - Test von Zustandsautomaten (*wird später behandelt*)
- **Methoden des White-Box-Test**
  - kontrollflussorientierte Verfahren
    - Anweisungsüberdeckungsverfahren
    - Kantenüberdeckungsverfahren
    - Bedingungsüberdeckungsverfahren
    - Pfadüberdeckungsverfahren
  - datenflussorientierte Verfahren (*wird nicht behandelt*)

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Black-Box-Test

---



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Fehlerklassen beim Black-Box-Testen

---

- Inkorrekte oder fehlende Funktionen
- Schnittstellenfehler
- Fehler in Datenstrukturen oder externem Datenbankzugriff
- Performanzfehler
- Fehler bei Initialisierung und Terminierung von Abläufen

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

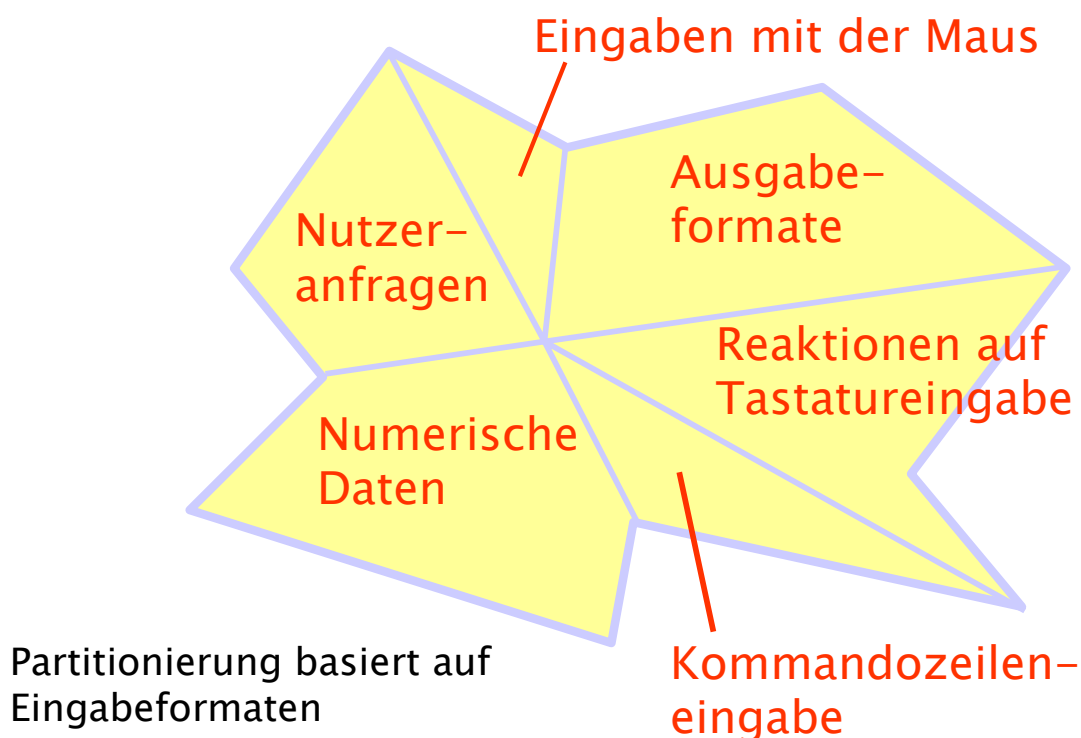
# Beispiel

```
int abs (int x)
```

```
{  
  if (x >= 0) return x;  
  if (x < 0)  return x;  
}
```

- Fehlverhalten für alle  $x < 0$ . Der konkrete Wert von  $x$  ist nicht wichtig!
- ☞ Basis für Äquivalenzklassenbildung

# Äquivalenzklassentest: Äquivalenzpartitionierung



# Äquivalenzklassen

---

- **Gültige Daten**
  - Kommandos der Benutzer
  - Antworten auf Systemausgaben
  - Dateinamen
  - Datenstrukturen des Programms
    - physikalische Parameter
    - Grenzwerte
    - Initialisierungswerte
    - initiation values
  - Kommandos zur Formatierung von Ausgabedaten
  - Antworten auf Fehlermeldungen
  - Graphische Daten (z.B. Mausclicks)
- **Ungültige Daten**
  - Daten außerhalb der Programmgrenzen
  - Physikalisch unmögliche Daten
  - Richtige Werte, die an falscher Stelle eingegeben werden

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Äquivalenzklassenmethode und Grenzwertanalyse

---

- **Äquivalenzklassenmethode (heuristisches Verfahren)**
  - Eine Äquivalenzklasse ist eine Menge von Eingabewerten, die auf ein Programm eine gleichartige Wirkung ausüben.
  - Es werden Äquivalenzklassen gültiger und ungültiger Werte gebildet, welche den Eingabebereich abdecken.
  - die Testfälle entsprechen (Kombinationen von) Äquivalenzklassen
  - aus jeder Äquivalenzklasse wird mindestens ein Testdatum gewählt
- **Grenzwertanalyse**
  - basierend auf der Äquivalenzklassenmethode
  - aus jeder Äquivalenzklasse werden Testdaten gewählt, welche Grenzwerte der Äquivalenzklassen abdecken

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Äquivalenzklassenmethode: Beispiel 1

Funktion zur Bestimmung der Anzahl der Tage in einem Monat  
 Eingabe: int monat, int jahr

- **monat: Äquivalenzklassen gültiger Werte**
  - $C_{m,31}$  = Monate mit 31 Tagen = {1,3,5,7,8,10,12}
  - $C_{m,30}$  = Monate mit 30 Tagen = {4,6,9,11}
  - $C_{m,feb}$  = {2}
  - Ungültige Werte:** Negative Zahlen, Zahlen > 12
  
- **jahr: Äquivalenzklassen gültiger Werte**
  - $C_{j,schaltjahr}$  = Menge der Schaltjahre
  - $C_{j,nicht-schaltjahr}$  = Menge der Nicht-Schaltjahre
  - Ungültige Werte:** Negative Zahlen

# Äquivalenzklassenmethode: Beispiel 1

Ableitung von Testfällen gültiger Eingaben aus den Äquivalenzklassen

Testfall	Äquivalenzklasse	Ausgabe	ausgewähltes Testdatum		
			monat	jahr	Ausgabe: Soll
T <sub>1,1</sub>	$C_{m,31}$ und $C_{j,nicht-schaltjahr}$	31	7	1901	31
T <sub>1,2</sub>	$C_{m,31}$ und $C_{j,schaltjahr}$	31	7	1904	31
T <sub>2,1</sub>	$C_{m,30}$ und $C_{j,nicht-schaltjahr}$	30	6	1901	30
T <sub>2,2</sub>	$C_{m,30}$ und $C_{j,schaltjahr}$	30	6	1904	30
T <sub>3,1</sub>	$C_{m,feb}$ und $C_{j,nicht-schaltjahr}$	28	2	1901	28
T <sub>3,2</sub>	$C_{m,feb}$ und $C_{j,schaltjahr}$	29	2	1904	29

# Äquivalenzklassenmethode: Beispiel 2

Spezifikation (noch nicht ganz konsistent) zur Ableitung des technischen Eintrittsalters einer Person in einen Versicherungsvertrag:

<b>Eingabe:</b> vertragsbeginn, geburtsdatum	
<b>Hilfsvariable</b> diff_Monat := Monat(vertragsbeginn) – Monat (geburtsdatum) diff_Jahr := Jahr (vertragsbeginn) – Jahr (geburtsdatum)	
<b>technisches_Eintrittsalter</b>	<b>Bedingung</b>
Fehler	vertragsbeginn < geburtsdatum
diff_Jahr	vertragsbeginn >= geburtsdatum und -5 <= diff_Monat <= 6
diff_Jahr + 1	vertragsbeginn >= geburtsdatum und diff_Monat >= 6
diff_Jahr - 1	vertragsbeginn >= geburtsdatum und diff_Monat < - 5

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Äquivalenzklassenmethode: Beispiel 2

Test-fall			Ausgewähltes Testdatum		
	Äquivalenzklasse	Ausgabe	Geburtsdatum	Vertragsbeginn	Ausgabe : Soll
T1	1 Vertragsbeginn vor Geburtsdatum	Fehler	01.02.2001	01.01.2001	Fehler
T2	2 diff_Monat im Intervall [-5, 6]	diff_Jahr	01.06.1975	01.08.2001	26
T3	3 diff_Monat >= 6	diff_Jahr + 1	01.06.1975	01.12.2001	27
T4	4 diff_Monat < - 5	diff_Jahr - 1	01.10.1975	01.01.2001	25

**Klasse 1 ist eine Klasse ungültiger Werte**

**Weitere ungültige Klassen:**

**Tag, Monat, Jahr nicht im gültigen Wertebereich**

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Äquivalenzklassenmethode: Beispiel 2

Ti-j : Testfall in Äquivalenzklasse i an der Grenze zu Klasse j

Testfall	Eingabe	Ausgabe	Ausgewähltes Testdatum		
			Geburts-datum	Vertrags-beginn	Ausgabe : Soll
T1-2	Vertragsbeginn 1 Tag vor Geburtsdatum	Fehler			
T2-1	Vertragsbeginn = Geburtsdatum	0			
T2-3	diff_Monat = 6	diff_Jahr			
T2-4	diff_Monat = - 5				
T3-2	diff_Monat = 6	diff_Jahr + 1			
T4-2	diff_Monat = - 6	diff_Jahr - 1			

**Nebeneffekt:**

Es wurde eine Inkonsistenz in der Spezifikation gefunden

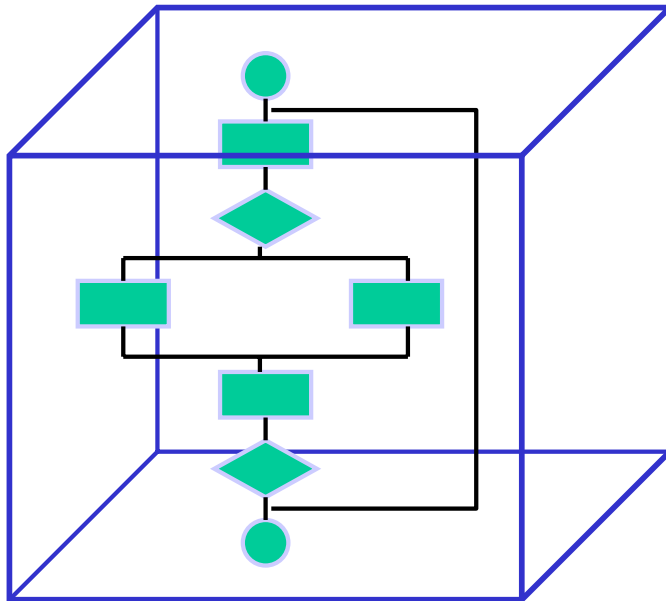
# Grenzen der Black-Box-Tests

- **Methoden des Black-Box-Test alleine sind nicht ausreichend, da die Spezifikation ein höheres Abstraktionsniveau besitzt wie die**

## Implementierung

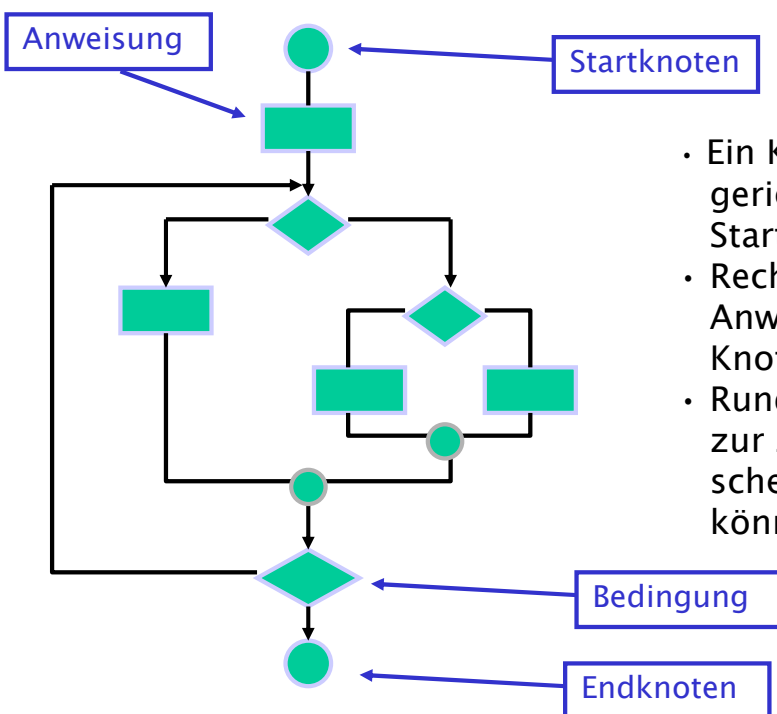
- nicht alle Elemente einer funktionalen Äquivalenzklasse werden in der Implementierung „intern“ gleich behandelt
- in der Implementierung kann ein Zustand im Automaten mehreren internen Zuständen entsprechen

# White-Box-Test



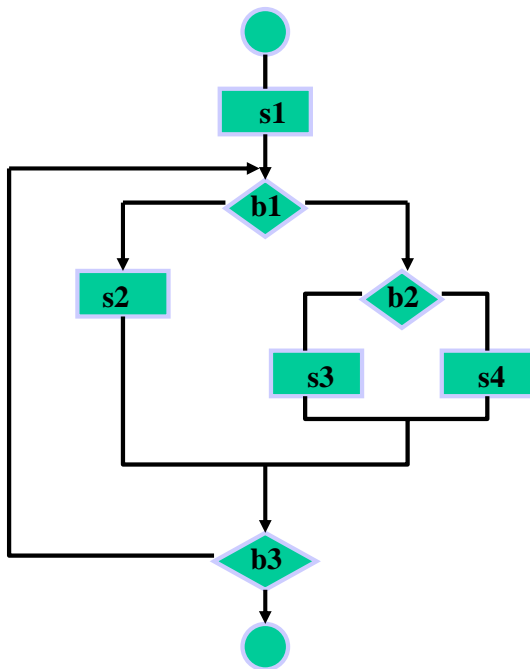
... Ziel ist, dass alle Anweisungen, Bedingungen und Pfade mindestens einmal ausgeführt werden...

# Kontrollflussgraph



- Ein Kontrollflussgraph ist ein gerichteter Graph mit genau einem Start- und genau einem Endknoten.
- Rechteckige Knoten stellen Anweisungen dar, rautenförmige Knoten Bedingungen.
- Runde Knoten im Inneren dienen nur zur Zusammenführung von Fallunterscheidungen und können weggelassen werden.

# Beispiel Kontrollflussgraph



Kontrollflussgraph

```

s1;
do {
  if ( b1 ) s2;
  else {
    if ( b2 ) s3;
    else s4;
  }
} while (b3)
  
```

Programmsegment

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

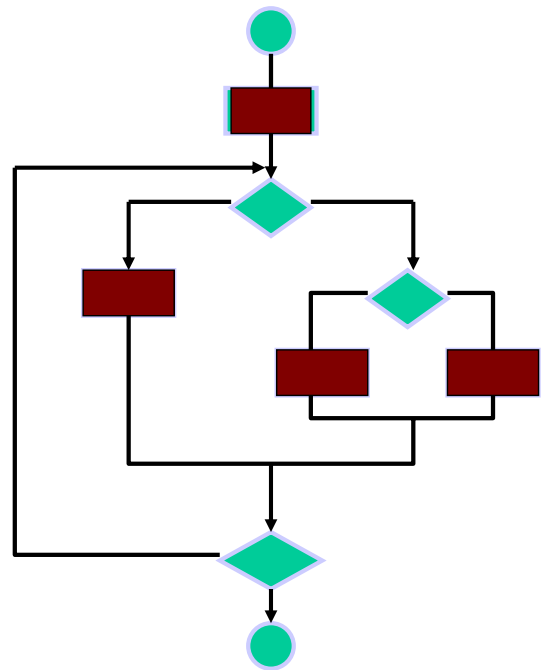
# Kontrollfluß-orientierte Testverfahren

- **Kontrollfluß-orientierte Testverfahren** orientieren sich am Kontrollflußgraphen des Programms
- Man unterscheidet folgende Typen von Verfahren:
  - Überdeckung von Anweisungen (C0)
  - Kantenüberdeckung (C1)
  - Bedingungsüberdeckung (C2, C3)
  - Pfadüberdeckung (C4)

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Anweisungsüberdeckungsverfahren (C0-Test)

- Kriterium: Wähle eine Testmenge, die **alle Anweisungen** des Testobjekts ausführt.
- Es gibt **Werkzeuge** zur Messung der C0-Überdeckung
- **Schwächen** des C0-Verfahrens
  - es müssen nicht alle Wege, die zu einer Anweisung führen überprüft werden: einer genügt
  - unzureichend für den Test von Schleifen



**Vollständige C0-Überdeckung**

# Beispiel eines fehlerhaften Programmfragments (J. Coldewey)

C0- Überdeckung bei **Testdaten**

**T1: (a < b und a + b < 10)**

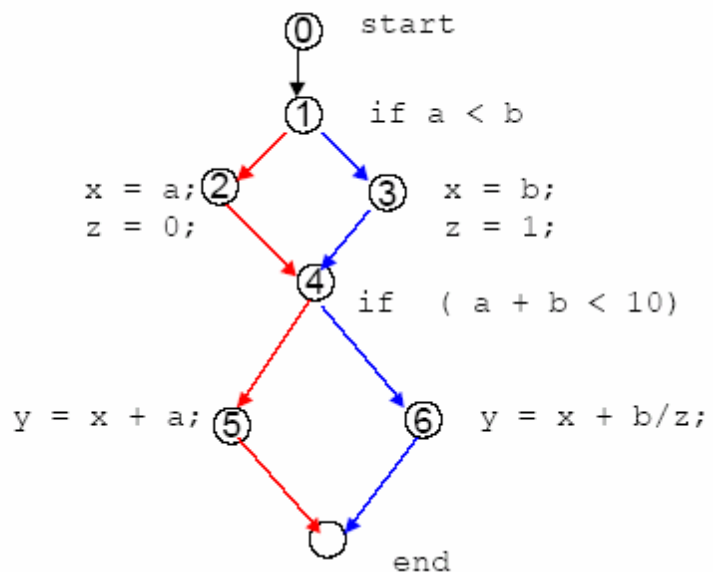
und

**T2: (a >= b und a + b >= 10 )**

Fehler im Pfad

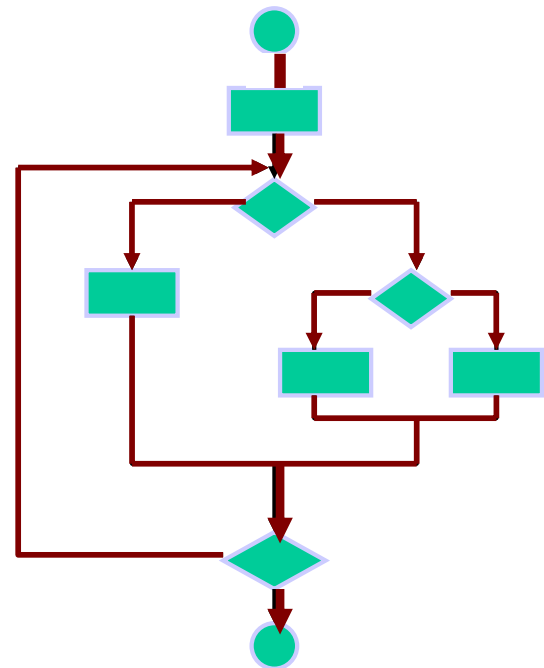
0; 1; 2; 4; 6

wird dabei nicht entdeckt



# Zweigüberdeckungstest (C1-Test)

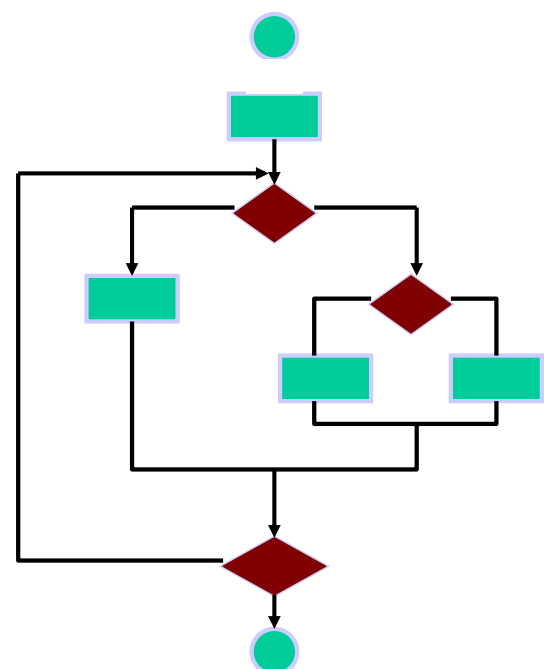
- **Kriterium:** Wähle eine Testmenge, die **alle Kanten** des Testobjekts ausführt.
- C1-Überdeckung enthält Anweisungsüberdeckungstest und gilt als **Minimalkriterium** für kontrollflussorientiertes Testen (vorgeschrieben vom Luftfahrtstandard RTCA DO-178B ab Stufe B).
- Es gibt **Werkzeuge** zur Messung der C1-Überdeckung
- **Schwächen** des C1-Verfahrens
  - Wie bei C0 müssen nicht alle Wege, die zu einer Anweisung führen überprüft werden;
  - unzureichend für den Test von Schleifen.



**Vollständige C1-Überdeckung**

# Bedingungsüberdeckungstest (C2/C3-Test)

- **Kriterium:** Wähle eine Testmenge, die **alle Bedingungen** überdeckt.
- Beim **einfachen Test (C2)** werden alle atomaren Teilformeln von Bedingungen auf true und false geprüft.
- Beim **Mehrfach-Bedingungs-Überdeckungstest (C3)** werden auch zusammengesetzte Teilformeln auf true und false geprüft.



**Vollständige C2/C3-Überdeckung**

# Beispiel Bedingungsüberdeckung

Bedingung der Form:  $(A||B) \&\& C$

Einfacher Bedingungsüberdeckungstest:

	A	B	C	A  B	(A  B)&&C
1,3	t	-	t	t	t
2,4	t	-	f	t	f
5	f	t	t	t	t
6	f	t	f	t	f
7,8	f	f	-	f	f

- **Sequenzielle Auswertung:** Die Testfälle 1,3 werden zusammengefasst, da die Belegung von B keine Rolle spielt.
- Die drei Testfälle (1,3), 6 und (7,8) bilden eine **vollständige C2-Überdeckung**, da A,B, C jeweils mit true und false belegt werden.
- Sie bilden auch eine **Überdeckung für den mehrfachen Bedingungsüberdeckungstest**, da unterschiedliche Werte für A||B vorkommen.

# Beispiel Bedingungsüberdeckung

Bedingung der Form:  $(A||B) \&\& C$

Modifizierter mehrfacher Bedingungsüberdeckungstest:

	A	B	C	A  B	(A  B)&&C
1	t	t	t	t	t
2	t	t	f	t	f
3	t	f	t	t	t
4	t	f	f	t	f
5	f	t	t	t	t
6	f	t	f	t	f
7	f	f	t	f	f
8	f	f	f	f	f

- Der modifizierte C3-Test fordert, dass **jede Teilformel unabhängig von anderen** die Gesamtentscheidung beeinflussen kann.
- Dies führt zu **n+1 Testfällen** bei n Teilformeln.
- Dieser Testtyp ist **vorgeschrieben vom Luftfahrtstandard RTCA DO-178B Stufe A**.

# Bedingungsüberdeckungstest (C2 / C3-Test)

---

## Bemerkungen:

- Bei sequentieller (unvollständiger Belegung) subsumiert der C2-Test den C1-Test.
- Vollständige Belegung beim C3-Test führt zu exponentiellem Anstieg der Testfälle (im Vergleich zur Anzahl der Teilformeln einer Bedingung).
- Bedingungsüberdeckungstests sind vor allem interessant, wenn eine komplizierte Verarbeitungslogik vorliegt, die zu kompliziert aufgebauten Entscheidungen führt. Empfohlen wird in diesem Fall der modifizierte C3-Test.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Pfadüberdeckungsverfahren

---

## Kriterium:

**Wähle eine Testmenge, die alle Pfade vom Eingangsknoten bis zum Endknoten durchläuft:**

- Nicht realistisch für while-Programme,
- Boundary-interior Pfadtest:

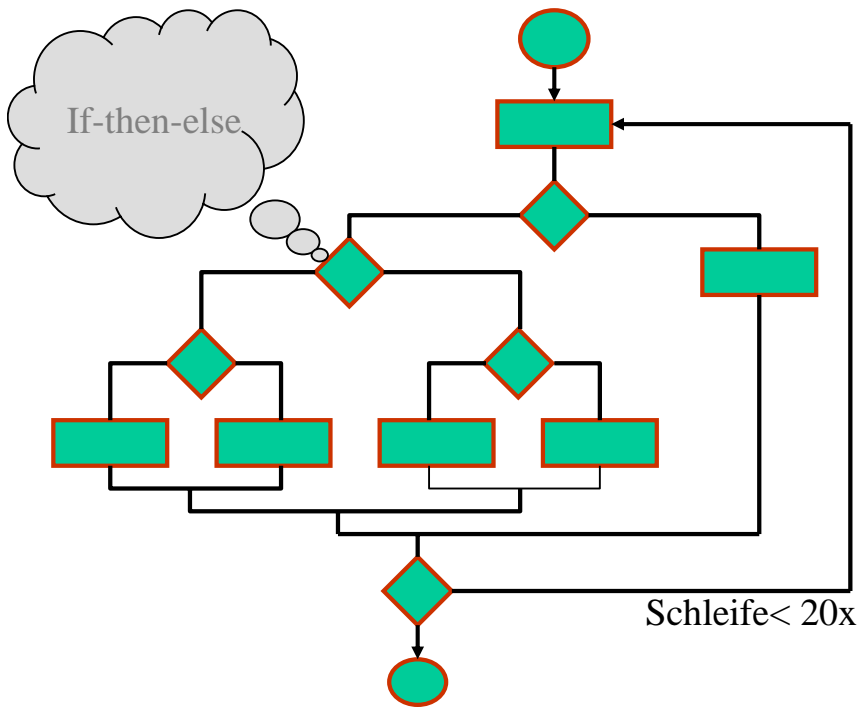
Die Anzahl der Wiederholungen in Schleifen wird eingeschränkt.

## Warum alle Pfade überdecken?

- Logische Fehler und inkorrekte Annahmen sind umgekehrt proportional zu der Wahrscheinlichkeit der Ausführung eines Pfads
- Entwickler **glauben** häufig, dass ein bestimmter Pfad nicht ausgeführt wird; die Realität verhält sich häufig anders.
- Tippfehler sind zufällig; sehr wahrscheinlich enthalten ungetestete Pfade Tippfehler.

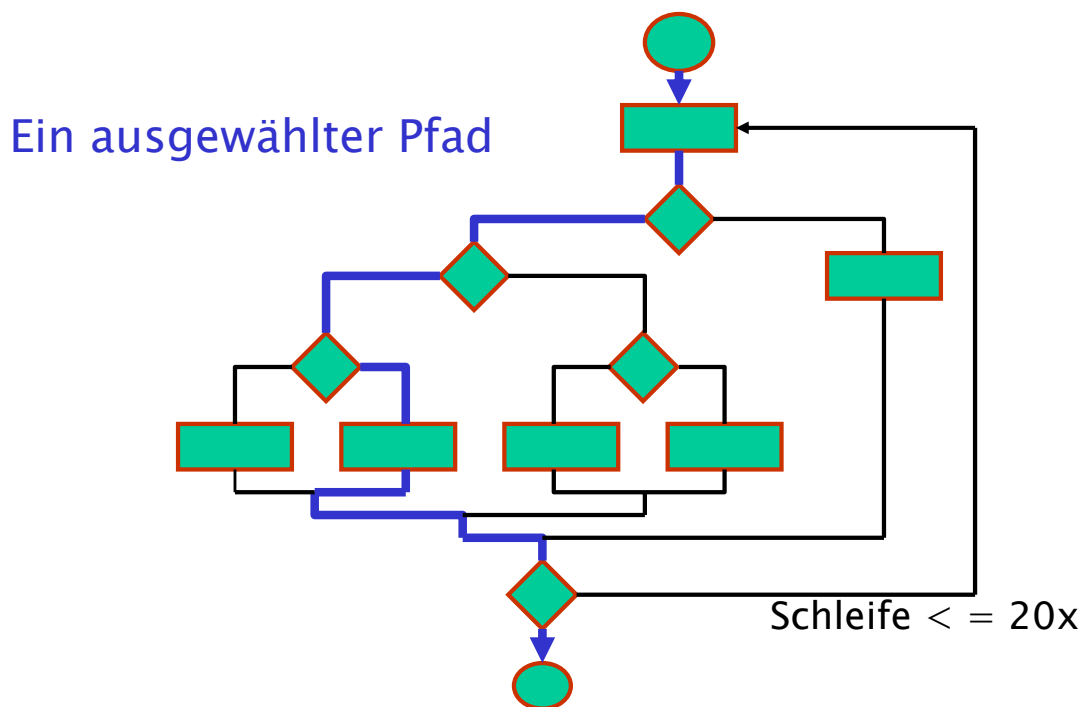
Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Garantierte Überdeckung: Alle Pfade

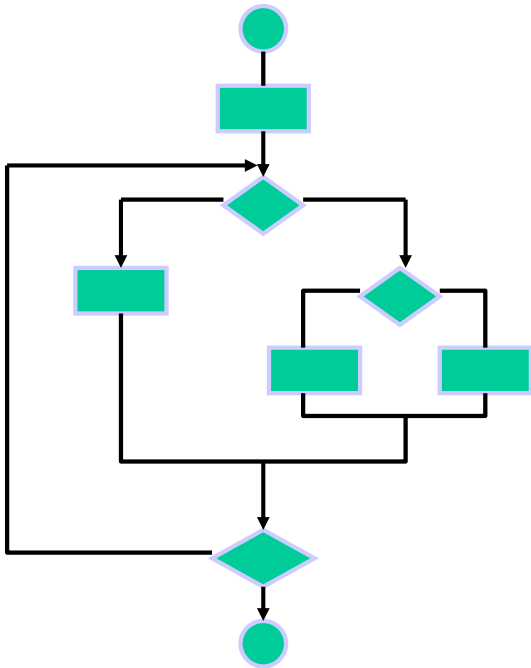


Aber:  
Es gibt ca.  
 $5^{20} = 10^{14}$   
mögliche Pfade!

# Besser: Selektives Testen



# Zyklomatische Komplexität: Ein Maß für Pfadüberdeckungstesten



Berechnung der zyklomatischen Komplexität  $V(G)$  :

Anzahl der Transitionen - Anzahl der Knoten + 2

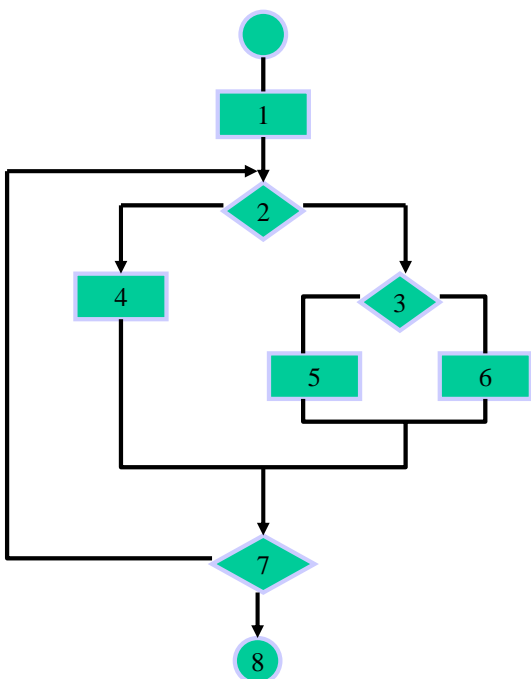
bzw. Anzahl der Bedingungen + 1

In diesem Fall,  $V(G) = 4$

$V(G)$  gibt die Anzahl der **linear unabhängigen Zyklen** von  $G$  an (wobei Start- und Endknoten miteinander verbunden werden).

$V(G)$  liefert eine **obere Schranke** für die Anzahl der Testfälle, die nötig sind, um alle Anweisungen zu überdecken.

## Pfadüberdeckungstesten: Beispiel



Wg  $V(G) = 4$  gibt es 4 Pfade:

Pfad 1: 1,2,3,6,7,8

Pfad 2: 1,2,3,5,7,8

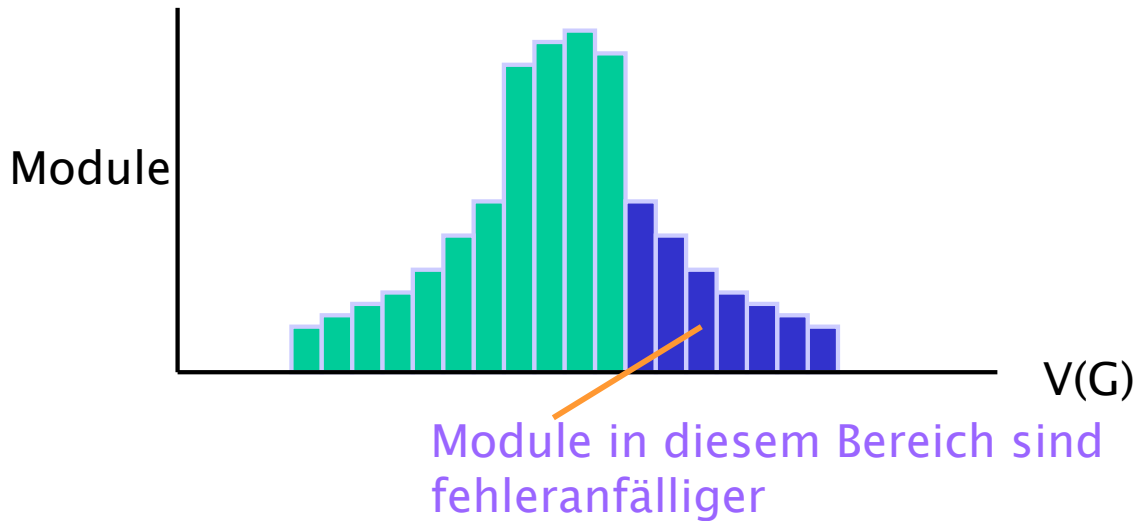
Pfad 3: 1,2,4,7,8

Pfad 4: 1,2,4,7,2,4...7,8

*Entwickle daraus Testmengen durch Angabe von geeigneten Inputs.*

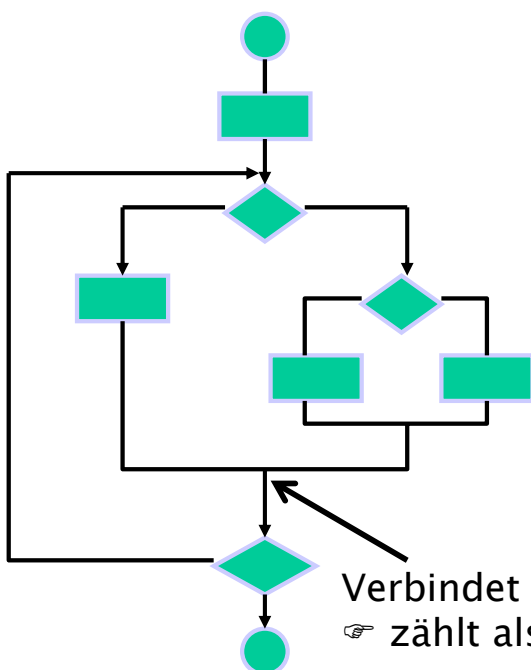
# Zyklomatische Komplexität

Eine Reihe von Industriestudien haben gezeigt, dass die Fehlerwahrscheinlichkeit bei großem  $V(G)$  ansteigt.



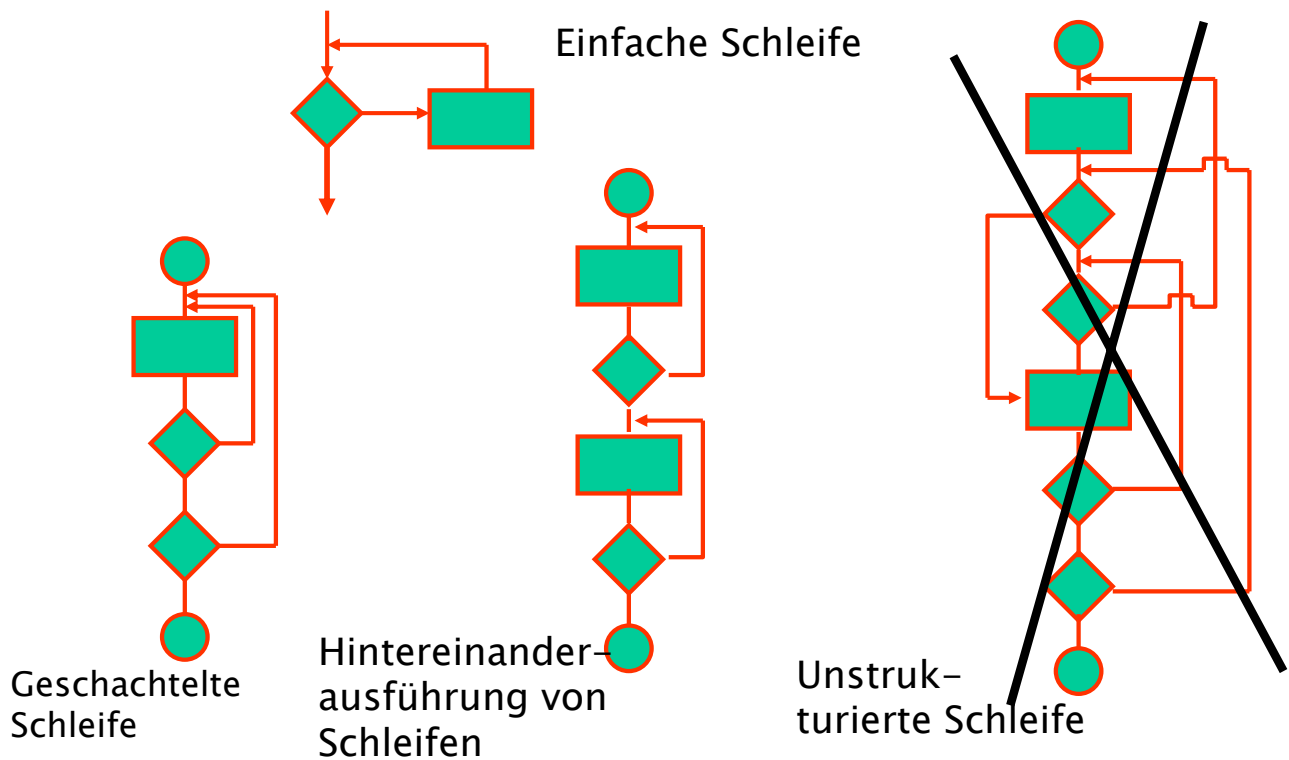
Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Pfadüberdeckungstesten: Bemerkungen



- Pfadüberdeckungstesten sollte bei kritischen Modulen angewendet werden.
- Kontrollflussgraphen sind nicht notwendig, aber hilfreich um die Pfade zu definieren.
- Jede Verbindung zwischen den Kästchen zählt als eine Transition.

# Testen von Schleifen

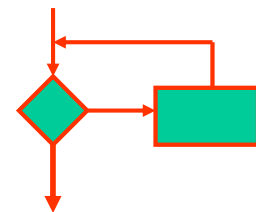


Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Testen von Schleifen : Einfache Schleifen

## Minimalbedingungen – einfache Schleifen:

1. KEIN Schleifendurchlauf
2. Nur ein Schleifendurchlauf
3. Zwei Schleifendurchläufe
4. m Schleifendurchläufe mit  $m < n$
5.  $(n-1)$ ,  $n$ , and  $(n+1)$  Schleifendurchläufe

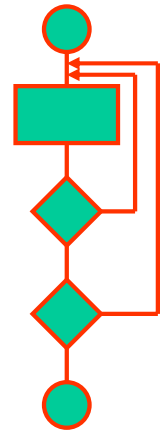


wobei  $n$  die maximale Anzahl erlaubter Schleifendurchläufe ist

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Geschachtelte Schleifen

- Naïve Erweiterung des Verfahrens für einfache Schleifen:
  - ☞ exponentielle Zunahme der Testfälle
- Reduktion der Testanzahl:
  - Starte mit der innersten Schleife; setze alle anderen Schleifen auf Minimalwerte
  - Führe Tests für einfache Schleife durch mit zusätzlichen Testfällen für Werte außerhalb der Definitionsbereichs und unmögliche Werte
  - Führe Tests für weiter außen liegende Schleifen durch, wobei die inneren Schleifen auf typischen Werten gehalten werden.
  - Fahre so lange fort, bis alle Schleifen getestet sind



Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

# Strategien zum White-Box-Testen

- Kontrollfluss-orientierte Tests können nur für kleine Programmsegmente von Hand durchgeführt werden. Sie sind deshalb nur für Unit Tests gut geeignet.
- Metrik-basierte Werkzeuge machen es möglich, White-Box-Tests auch beim Integrations- und Systemtest einzusetzen.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Metrik-gesteuerte Systemtestfall-Auswahlstrategie

---

- Verwende Äquivalenztests und Grenzwertanalyse für eine vollständige Testsuite von Black-Box-Tests.
- Führe die Black-Box-Testsuite aus und messe den Überdeckungsgrad. Normalerweise wird dieser bei 60% bis 70% Anweisungsüberdeckung liegen.
- Wähle White-Box-Tests so, dass eine bestimmte Überdeckung erreicht wird, wie z.B.
  - >95% Anweisungsüberdeckung und
  - >90% Kantenüberdeckung.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München

## Zusammenfassung

---

- Das Black-Box-Testen geht von der Spezifikation aus und die Interna des Testobjekts sind nicht bekannt. Das White-Box-Testen ergänzt das Black-Box-Testen durch systematisches Explorieren der Struktur des Testobjekts.
- Wichtige Methoden des Black-Box-Testens sind die Äquivalenzklassenmethode, die Methode der Grenzwertanalyse und der Test von Zustandsautomaten.
- Wichtige Techniken des White-Box-Testens sind kontrollflussorientierte Verfahren wie etwa Anweisungsüberdeckungsverfahren, Kantenüberdeckungsverfahren, Bedingungsüberdeckungsverfahren, Pfadüberdeckungsverfahren sowie datenflussorientierte Verfahren.

Methoden des Software Engineering (c) 2004, Koch, Störrle, Wirsing, LMU München