



GTB

German Testing Board

Software. Testing. Excellence.



Basiswissen Softwaretest Certified Tester Dynamischer Test – White-Box

HS@GTB
2019
Version 3.1



Nach dieser Vorlesung sollten Sie ...

- Die Grundidee der White-Box-Testverfahren erläutern können
- White-Box-Testverfahren zur Ermittlung von Testfällen charakterisieren und voneinander abgrenzen können
- Die Anweisungsüberdeckung und die Entscheidungsüberdeckung für den Kontrollflusstest kennen und auf einfache Beispiele anwenden können
- Unterschiedliche Arten der Bedingungsüberdeckung kennen und auf einfache Beispiele anwenden können
- Intuitive Testfallermittlung, exploratives Testen und checklistenbasiertes Testen als erfahrungsbasierte Testverfahren kennen und charakterisieren können
- Dynamische Tests bezüglich ihres Einsatzbereiches erläutern und mit anderen Testverfahren zu einer Teststrategie kombinieren können

Lernziele für den Abschnitt Testverfahren

(nach Certified Tester Foundation Level Syllabus,
deutschsprachige Ausgabe, Version 2018)



4.3 White-Box-Testverfahren

- FL-4.3.1 (K2) Anweisungsüberdeckung erklären können
- FL-4.3.2 (K2) Entscheidungsüberdeckung erklären können
- FL-4.3.3 (K2) Die Bedeutung von Anweisungs- und Entscheidungsüberdeckung erklären können

4.4 Erfahrungsbasierte Testverfahren

- FL-4.4.1 (K2) Die intuitive Testfallermittlung erklären können
- FL-4.4.2 (K2) Exploratives Testen erklären können
- FL-4.4.3 (K2) Checklistenbasiertes Testen erklären können



Kategorien von Testfallentwurfsverfahren



Photo: Fotolia / Dan Race



Source: <http://www.operation.de/knie/>

- **Black-Box-Testverfahren**

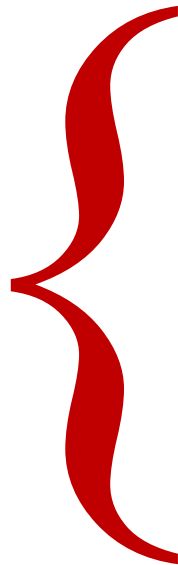
- Spezifikationen sind Grundlage für den Testentwurf, aber nicht der Programm Quelltext oder sein innerer Aufbau
- z.B. Äquivalenzklassenbildung und Grenzwertanalyse

- **White-Box-Testverfahren**

- Programm Quelltext liegt vor und ist Grundlage für den Testentwurf
- z.B. Entscheidungsüberdeckung oder Anweisungsüberdeckung

- **Erfahrungsbasierte Testverfahren**

4.2
Dynamischer
Test
White-Box



Idee der White-Box-Testverfahren

Kontrollflusstest

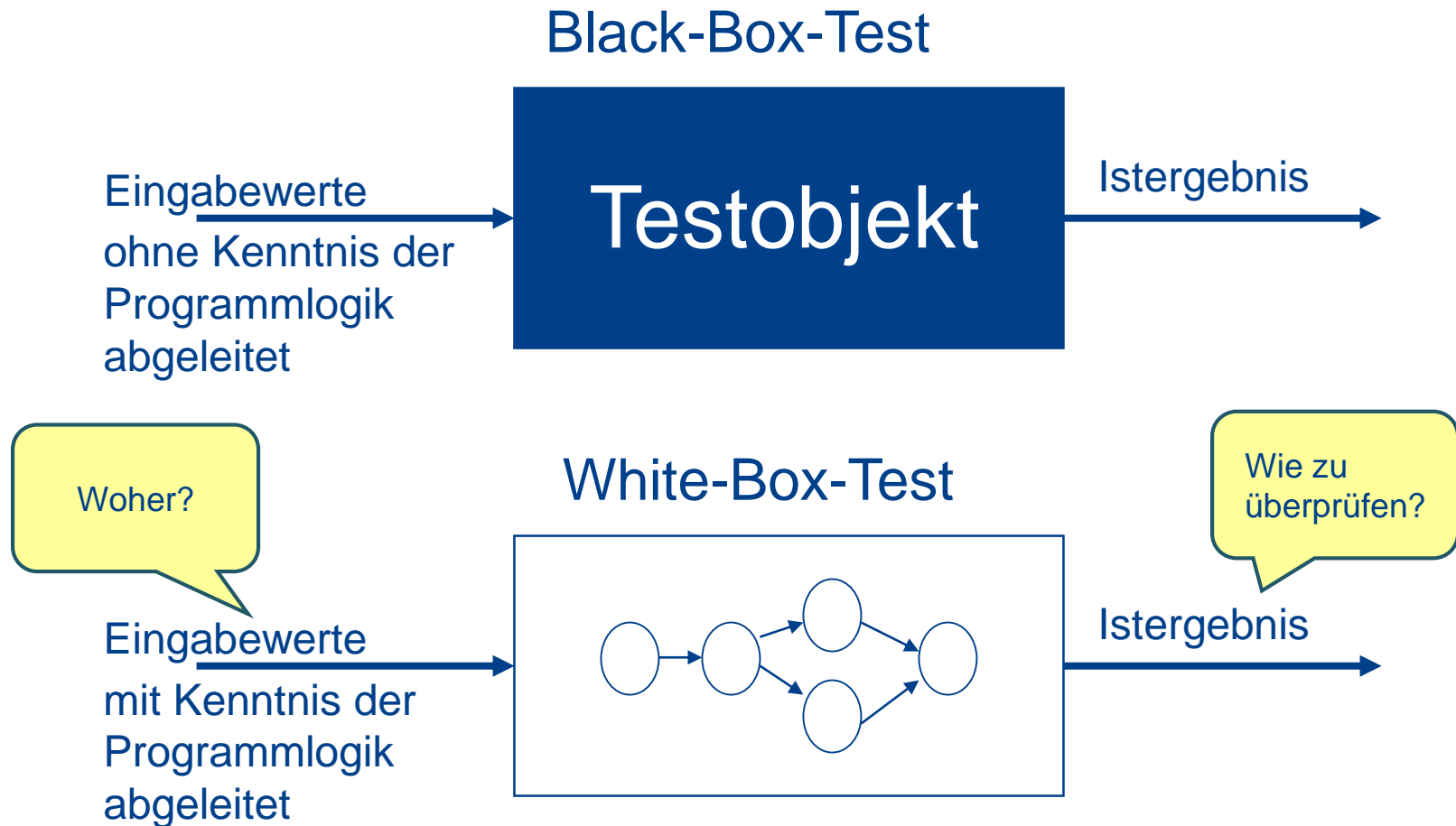
Exkurs: Datenflusstest

Exkurs: Bedingungstest

Erfahrungsbasierte Testverfahren

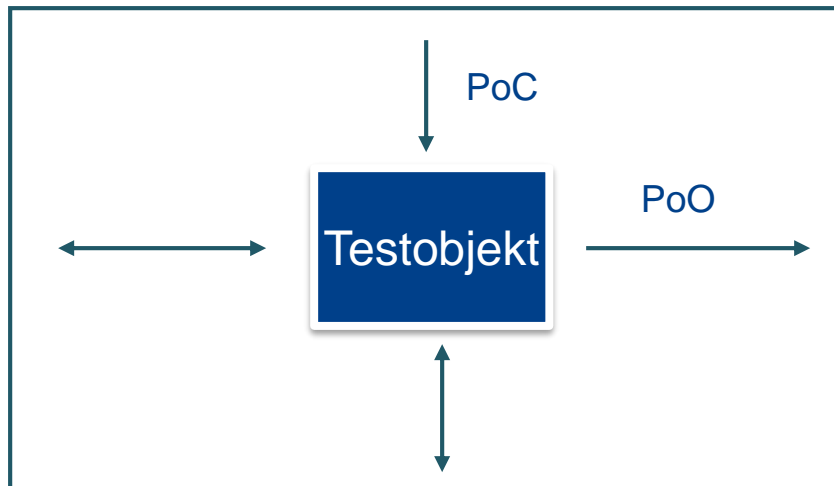
Dynamischer Test: Testverfahrensauswahl
und Zusammenfassung

Zur Erinnerung: Black-Box-Test vs. White-Box-Test (1)

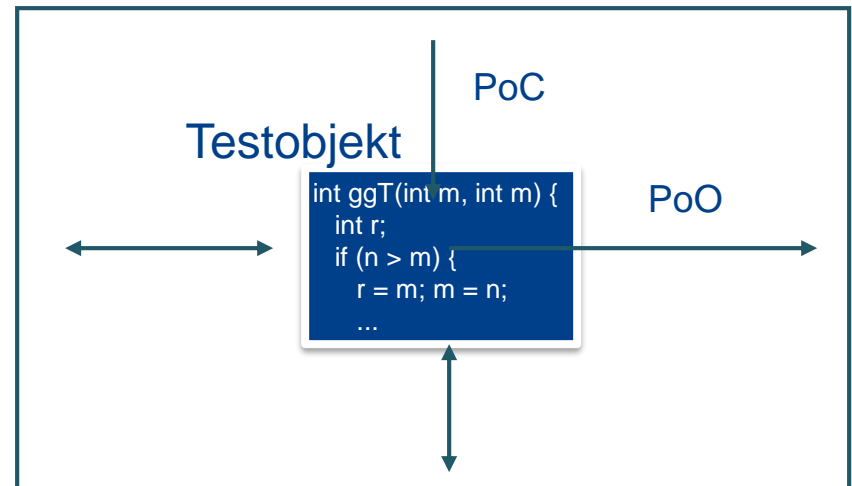


Zur Erinnerung: Black-Box-Test vs. White-Box-Test (2)

Black-Box-Test



White-Box-Test



PoC = Point of Control
PoO = Point of Observation



White-Box-Testverfahren – Begriffe

White-Box-Test

Ein Test, der auf der Analyse der internen Struktur einer Komponente oder eines Systems basiert.

White-Box-Testverfahren

Ein Verfahren zur Herleitung und Auswahl von Testfällen, basierend auf der internen Struktur einer Komponente oder eines Systems.

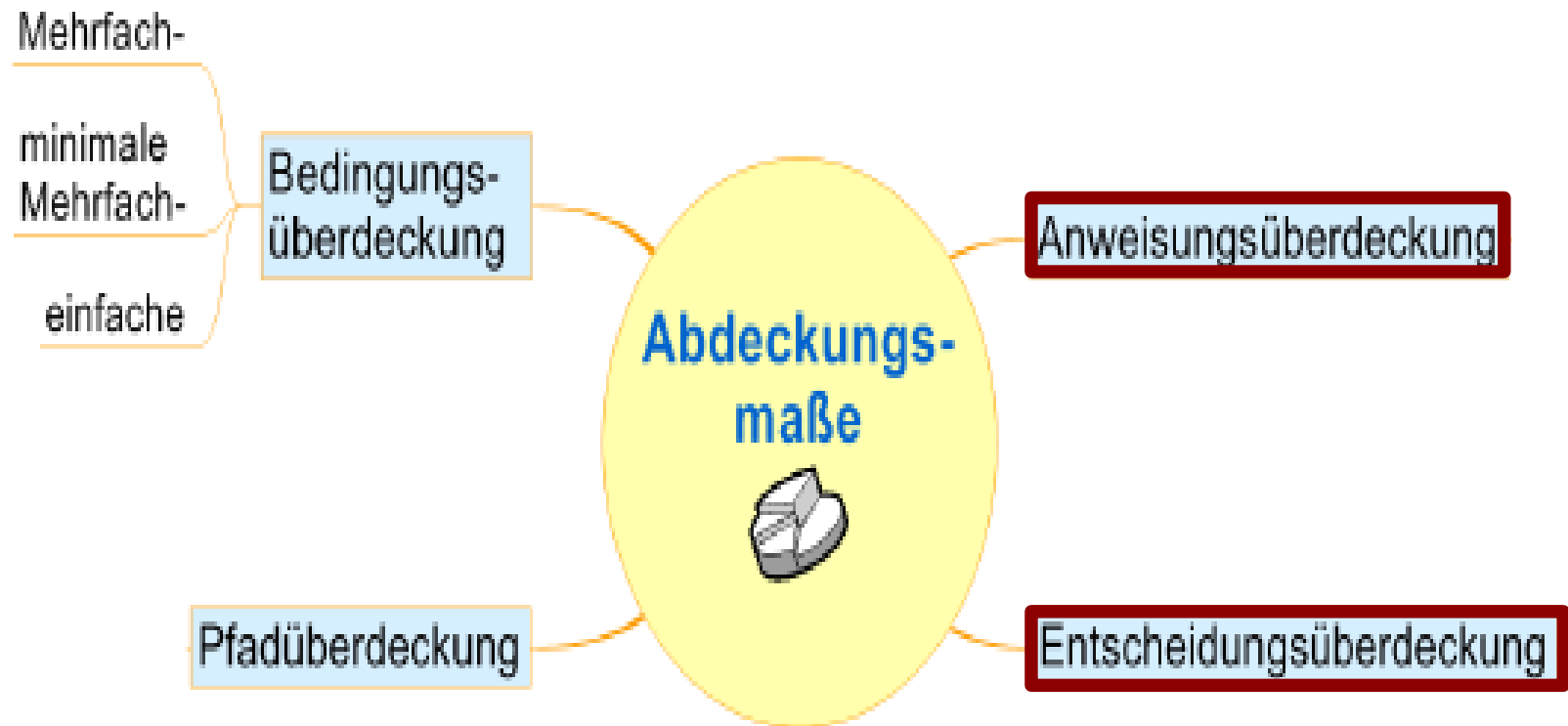
- White-Box-Test ist Stichprobe aller möglichen Programmabläufe und Datenverwendungen
- White-Box-Testverfahren verwenden innere Struktur des Testobjekts (z.B. Kontrollfluss, Datenfluss)
 - zur Herleitung oder Auswahl der Testfälle
 - zum Prüfen der Vollständigkeit (Überdeckung)
 - auch strukturorientierte, strukturbezogene oder strukturelle Testverfahren genannt
 - Testbasis: Quelltext, Architekturmodelle, Schnittstellen, etc.

(ISTQB Glossar V3.2)

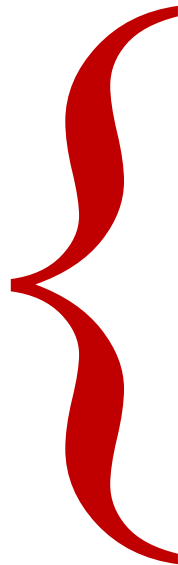
Strukturelles Testen von Programmen

- Kontrollflusstest
 - Anweisungsüberdeckung (C0-Überdeckung, alle Knoten des Kontrollflussgraphen)
 - Entscheidungsüberdeckung (C1-Überdeckung, bei Knoten mit mehr als einer ausgehenden Kante alle diese Kanten; auch: alle Zweige des Kontrollflussgraphen, Zweigüberdeckung)
 - Grenze-Inneres-Überdeckung (Cgi, alle Schleifen einmal abgewiesen, einmalig ausgeführt und wiederholt ausgeführt)
 - Pfadüberdeckung (C_{∞} -Überdeckung, alle Pfade)
- Datenflusstest
 - Alle Definitionen (all defs)
 - Alle Definition-Verwendungspaare (all def-uses)
- Bedingungstest
 - Einfache Bedingungsüberdeckung
 - Mehrfachbedingungsüberdeckung
 - Minimal bestimmende Mehrfachbedingungsüberdeckung

Abdeckungsmaße



4.2
Dynamischer
Test
White-Box



Idee der White-Box-Testverfahren

Kontrollflusstest

Exkurs: Datenflusstest

Exkurs: Bedingungstest

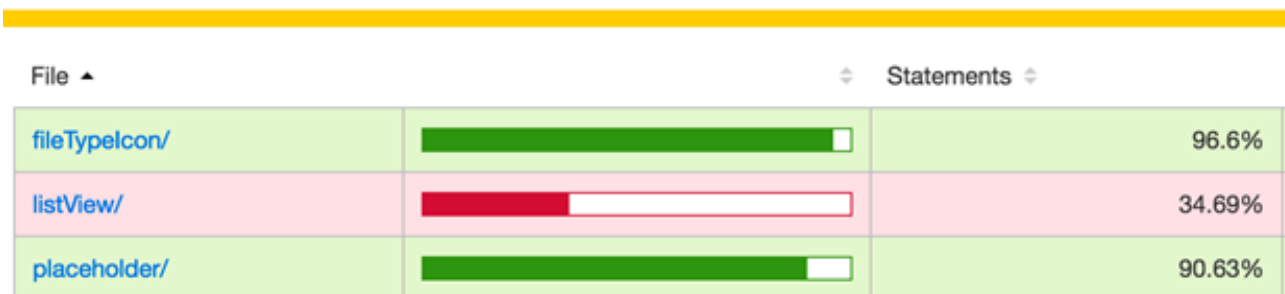
Erfahrungsbasierte Testverfahren

Dynamischer Test: Testverfahrensauswahl
und Zusammenfassung

Kontrollflusstest

- Testverfahren, bei dem Testfälle auf Basis von Kontrollflüssen im Testobjekt entworfen werden
- Verschiedene Überdeckungsformen, z.B.
 - Anweisungsüberdeckung (*statement coverage*)
 - Entscheidungsüberdeckung (*decision coverage*)
 - Exkurs: Zweigüberdeckung (*branch coverage*)
 - Exkurs: Grenze-Inneres-Test (*boundary interior coverage*)
 - Exkurs: Pfadüberdeckung (*path coverage*)

74.01% Statements 285/277 62.42% Branches 93/149 64.81% Functions 35/54 75.48% Lines 197/261





Beispiel: Funktion **ggT** ()

- Bestimmung des größten gemeinsamen Teilers (ggT) zweier ganzer Zahlen m und n (*greatest common divisor, gcd*)
 - Beispiele: $\text{ggT}(4,8)=4$; $\text{ggT}(5,8)=1$; $\text{ggT}(15,35)=5$, $\text{ggT}(35, 35) = 35$
- Spezifikation in UML / Java

```
public int ggt(int m, int n) {  
    // pre: m > 0 and n > 0  
    // post: return > 0 and m@pre.mod(return) = 0 and  
    //       n@pre.mod(return) = 0 and  
  
    //       forall(i : int | i > return implies  
  
    //           (m@pre.mod(i) > 0 or n@pre.mod(i) > 0)  
}
```



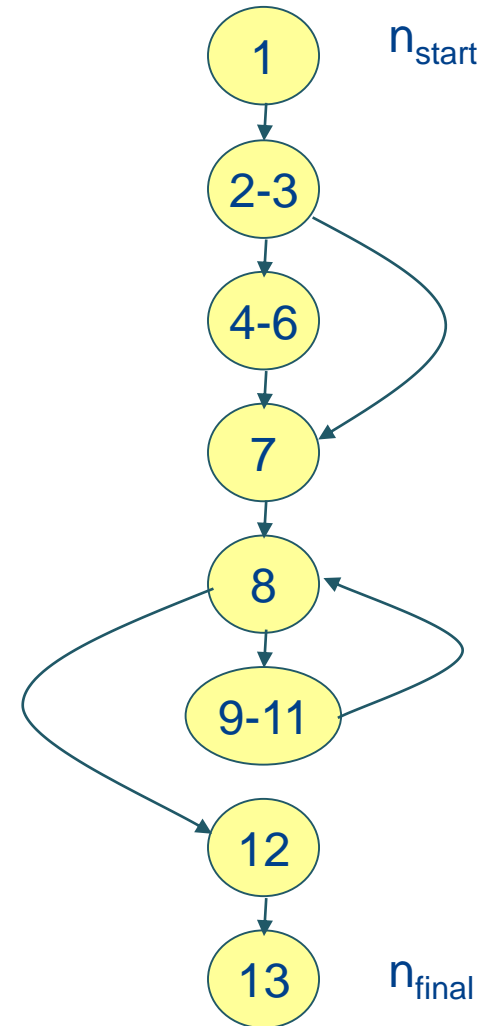
Beispiel: Kontrollflussgraph von `ggT()`

```
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.     }  
8.     r = m % n;  
9.     while (r != 0) {  
10.        m = n;  
11.        n = r;  
12.        r = m % n;  
13.    }  
14.     return n;  
15. }
```

Block

Block

Block



Anweisungsüberdeckung

- Testfälle anhand von Pfaden durch den Kontrollflussgraphen bestimmen
- Testfall durchläuft auf dem Pfad liegende Kanten des Graphen und führt die auf dem Pfad liegenden Anweisungen (Knoten) aus

$$\text{Anweisungsüberdeckung} = \frac{\text{Anzahl ausgeführter Anweisungen}}{\text{Gesamtzahl ausführbarer Anweisungen}}$$

- Für Anweisungsüberdeckung nur wichtig, ob Anweisung überhaupt durchlaufen; Häufigkeit der Ausführung egal
- Vor- / Nachbedingungen, erwartete Ergebnisse / erwartetes Verhalten *nicht* aus dem Code ableiten, sondern aus einer *anderen* Testbasis (z.B. Anforderungsspezifikation)

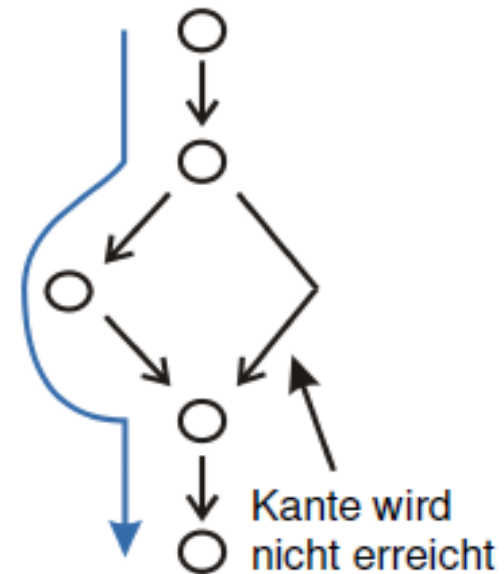


Anweisungsüberdeckung

Anweisungsüberdeckung

- Ziel: Alle **Anweisungen** im Testobjekt werden **mindestens einmal** ausgeführt.
- Nachteil:
 - Kontrollfluss wird nicht berücksichtigt
 - Abhängigkeiten zwischen Daten werden nicht berücksichtigt

$$C_0 = \frac{\# \text{ ausgeführte Anweisungen}}{\# \text{ alle Anweisungen}}$$



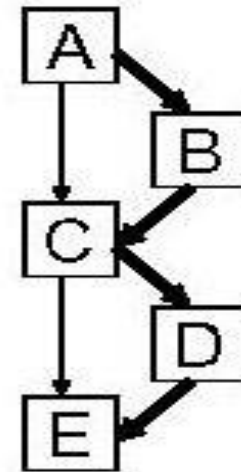


Anweisungsüberdeckung (Beispiel)

- Jede Anweisung wird einmal durchlaufen

```
void CoverMe (int a, int b)
{
    printf("A");
    if (a < 1)
        printf("B");
    printf("C");
    if (b < 2)
        printf("D");
    printf("E");
}
```

**Welche minimale Anzahl von Testfällen
wird benötigt
für eine 100% Anweisungsüberdeckung?**



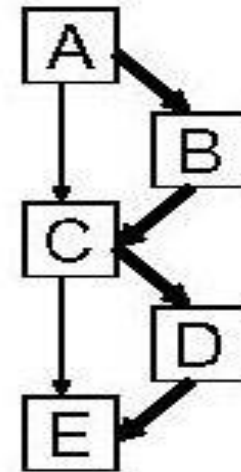


Anweisungsüberdeckung (Beispiel)

- Jede Anweisung wird einmal durchlaufen

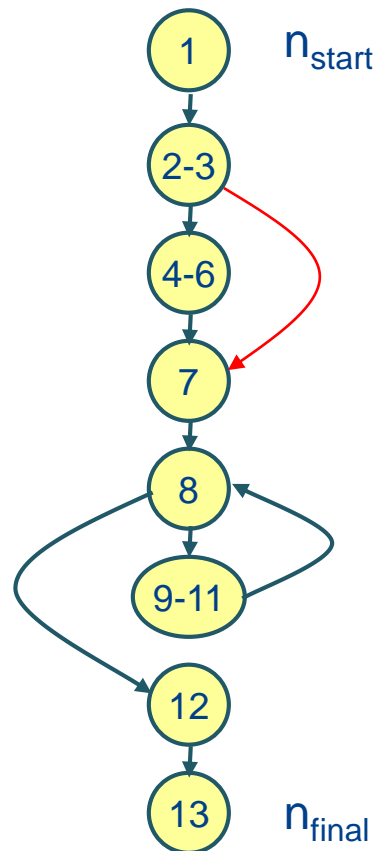
```
void CoverMe (int a, int b)
{
    printf("A");
    if (a < 1)
        printf("B");
    printf("C");
    if (b < 2)
        printf("D");
    printf("E");
}
```

1 Testfall reicht aus:
a=0, b=1 => ABCDE





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.    }  
8.    while (r != 0) {  
9.        m = n;  
10.       n = r;  
11.       r = m % n;  
12.    }  
13.    return n;  
13. }
```

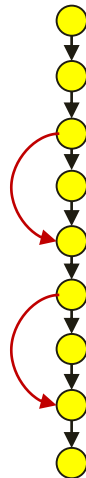
Pfad = (1, 2-3, 4-6, 7, 8, 9-11, 8, 12, 13)

Diskussion der Anweisungsüberdeckung

- **Anweisungsüberdeckung ist vergleichsweise schwaches Kriterium**
 - Ignoriert „leere“ Kanten, die ein oder mehrere Knoten „überbrücken“, z. B.
 - (ELSE-)Kante (zwischen IF und ENDIF) mit leerem ELSE-Teil
 - CONTINUE für Rücksprung zum Anfang einer Schleife
 - BREAK zum Abbruch einer Schleife
 - RETURN zum Abbruch eines Unterprogramms
 - Fehlende Anweisungen werden nicht erkannt

Beispiel:

```
BEGIN
  Read Time
  IF Time < 12 Then
    Print (Time, „am“)
  ENDIF
  IF Time > 12 Then
    Print (Time -12, „pm“)
  ENDIF
END;
```



- **Anweisungsüberdeckung von 100% nicht immer erreichbar**
 - z.B. wenn Ausnahmebedingungen im Testobjekt vorkommen, die während des Testens nur mit erheblichem Aufwand / gar nicht herstellbar sind
 - z.B. wenn nicht erreichbare Anweisungen (dead code) vorhanden sind (ggf. statische Analyse durchführen)

Entscheidungstest

- Testfälle anhand von Pfaden durch den Kontrollflussgraphen bestimmen
- Jede Entscheidung im Kontrollfluss zu allen Fällen (Entscheidungsausgängen) auswerten, dazu passende Pfade im Kontrollfluss finden
 - IF-/WHILE-/FOR-Bedingungen zu wahr und falsch
 - SWITCH/CASE zu allen Alternativen, inkl. DEFAULT

$$\text{Entscheidungsüberdeckung} = \frac{\text{Anzahl getesteter Entscheidungsausgänge}}{\text{Gesamtzahl Entscheidungsausgänge}}$$

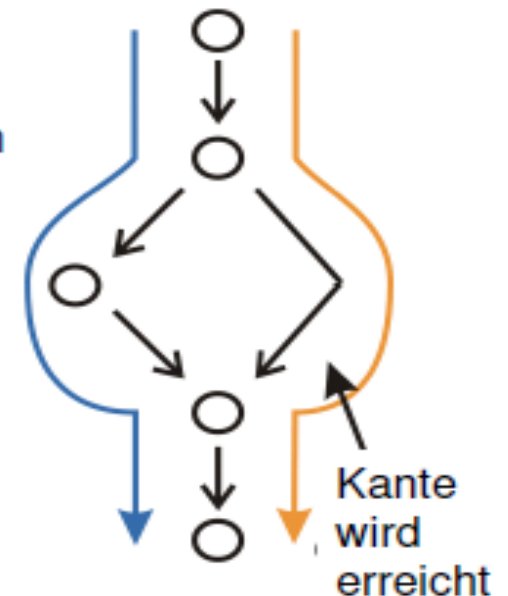
- Für Entscheidungsüberdeckung nur wichtig, ob Entscheidungsausgang überhaupt durchlaufen; Häufigkeit des Durchlaufens egal
- 100% Entscheidungsüberdeckung \Rightarrow 100% Anweisungsüberdeckung
- Vor- / Nachbedingungen, erwartete Ergebnisse / erwartetes Verhalten *nicht* aus dem Code ableiten, sondern aus einer *anderen* Testbasis (z.B. Anforderungsspezifikation)



Entscheidungstest

Entscheidungsüberdeckung

- Nachteile:
 - Schleifen müssen nur einmal durchlaufen werden
 - Komplexe Bedingungen werden nicht berücksichtigt
- Subsumiert Anweisungsüberdeckung



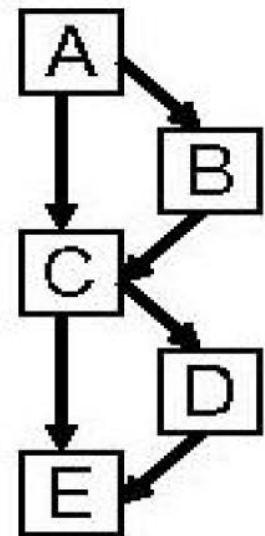
$$C_1 = \frac{\# \text{ zu Wahr und zu Falsch ausgewertete Entscheidungen}}{\# \text{ alle Entscheidungen}}$$



Beispiel: Entscheidungstest

```
void CoverMe (int a, int b)
{
    printf("A");
    if (a < 1)
        printf("B");
    printf("C");
    if (b < 2)
        printf("D");
    printf("E");
}
```

**Welche minimale Anzahl von Testfällen
wird benötigt
für eine 100% Entscheidungsüberdeckung?**

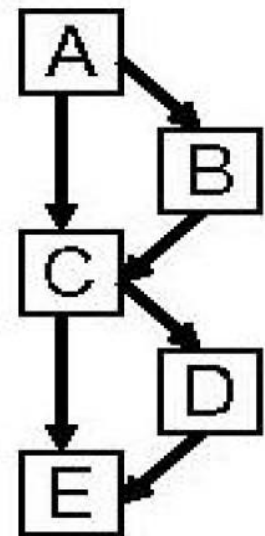




Beispiel: Entscheidungstest

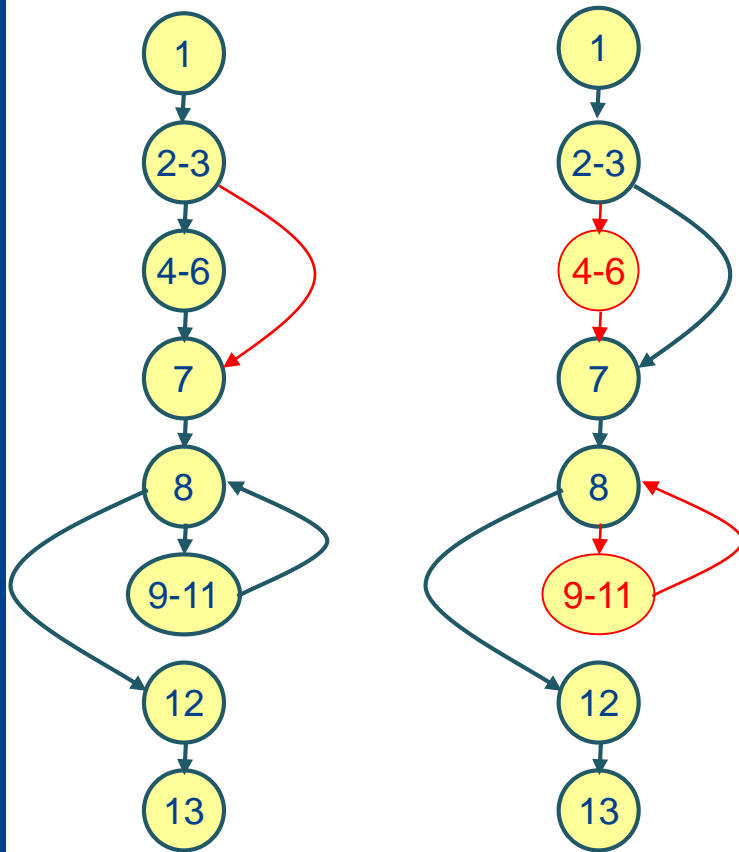
```
void CoverMe (int a, int b)
{
    printf("A");
    if (a < 1)
        printf("B");
    printf("C");
    if (b < 2)
        printf("D");
    printf("E");
}
```

2 Testfälle sind nötig:
a=0, b=1 => ABCDE
a=1, b=2 => ACE





Beispiel: Entscheidungsüberdeckung für `ggt()`



Pfad 1 = (1, 2-3, 4-6, 7, 8, 9-11, 8, 12, 13)
Pfad 2 = (1, 2-3, 7, 8, 12, 13)

```
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.     }
8.     while (r != 0) {
9.         m = n;
10.        n = r;
11.        r = m % n;
12.    }
13.    return n;
}
```

Entscheidung



Exkurs: Zusammenhang mit Zweigüberdeckung

- Entscheidungsüberdeckung betrachtet die Ausgänge von Entscheidungen; diese sind mindestens einmal zu allen Fällen auszuwerten.
- Zweigüberdeckung zielt darauf ab, alle Kanten (Zweige) im Kontrollflussgraphen abzudecken. Durch die Testfälle müssen alle Kanten des Graphen durchlaufen werden (auch die „leeren“ Kanten).

$$\text{Zweigüberdeckung} = \frac{\text{Anzahl durchlaufener Zweige}}{\text{Gesamtzahl Zweige}}$$

- Jeder Entscheidungsausgang hat eine Kante (Zweig) im Kontrollflussgraphen, daher gilt:
 - 100% Entscheidungsüberdeckung \Rightarrow 100% Zweigüberdeckung
 - 100% Zweigüberdeckung \Rightarrow 100% Entscheidungsüberdeckung

Diskussion der Entscheidungsüberdeckung

- Entscheidungsüberdeckung ist stärkeres Kriterium als Anweisungsüberdeckung
 - Entscheidungsüberdeckung berücksichtigt bei einer Verzweigung des Kontrollflusses *alle* Möglichkeiten
 - Entscheidungsüberdeckung kann fehlende Anweisungen (z.B. in leeren ELSE-Zweigen) erkennen, Anweisungsüberdeckung nicht
- Empfehlung: 100% Entscheidungsüberdeckung anstreben (aber analog 100% Anweisungsüberdeckung oft nicht zu erreichen)
- Blinder Fleck: Die Entscheidungsausgänge innerhalb der Testfälle werden unabhängig voneinander betrachtet
 - keine Abdeckung der Kombinationen der Entscheidungsausgänge verschiedener Entscheidungen gefordert
 - spezielle Fehlerzustände können somit „übersehen“ werden



Exkurs: Grenze-Inneres-Überdeckung

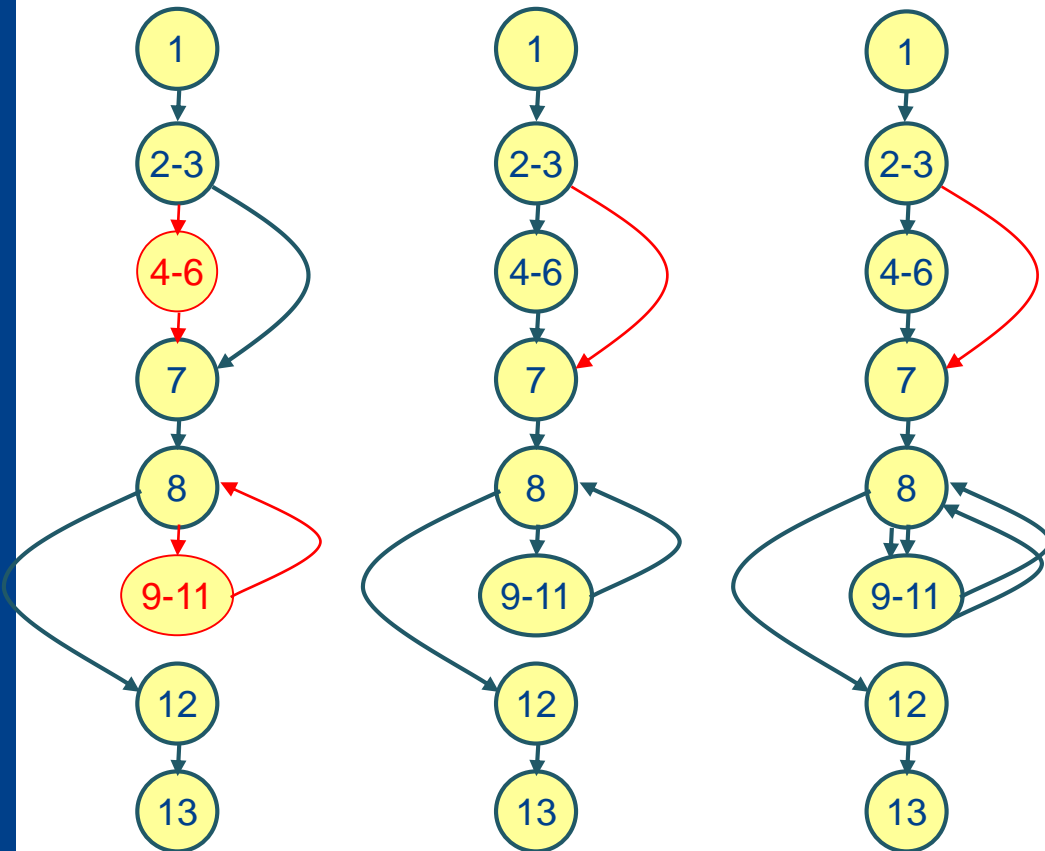
- 100% Grenze-Inneres-Überdeckung bedeutet:
jede Schleife in mindestens einem Testfall
 - gar nicht (nur bei abweisenden Schleifen wie WHILE, FOR möglich),
 - genau einmal und
 - mehr als einmal ausgeführt

$$\text{Grenze-Inneres-Überdeckung} = \frac{\text{Anzahl grenze-inneres-getesteter Schleifen}}{\text{Gesamtzahl Schleifen}}$$

- Testfälle anhand der Pfade durch den Kontrollflussgraphen bestimmen



Exkurs: Grenze-Inneres-Überdeckung für `ggt()`



Pfad 1 = (1, 2-3, 7, 8, 12, 13)
Pfad 2 = (1, 2-3, 4-6, 7, 8, 9-11, 8, 12, 13)
Pfad 3 = (1, 2-3, 4-6, 7, 8, 9-11, 8, 9-11, 8, 12, 13)

```
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;
13. }
```



Exkurs: Diskussion der Grenze-Inneres-Überdeckung

- Grenze-Inneres-Überdeckung auf Schleifen beschränkt
 - Verlangt nur, Schleifen auf bestimmte Art zu testen
 - Berücksichtigt keine Programmteile außerhalb von Schleifen
 - Erkennt z.B. fehlende Anweisungen in „leeren“ Zweigen nicht
- Daher in der Praxis höchstens als ergänzendes Kriterium verwenden
- Die einzelnen Schleifen werden unabhängig voneinander betrachtet
- Für verschachtelte Schleifen gibt es spezialisierte, in ihrer Stärke abgestufte Überdeckungsmaße



Exkurs: Pfadüberdeckung

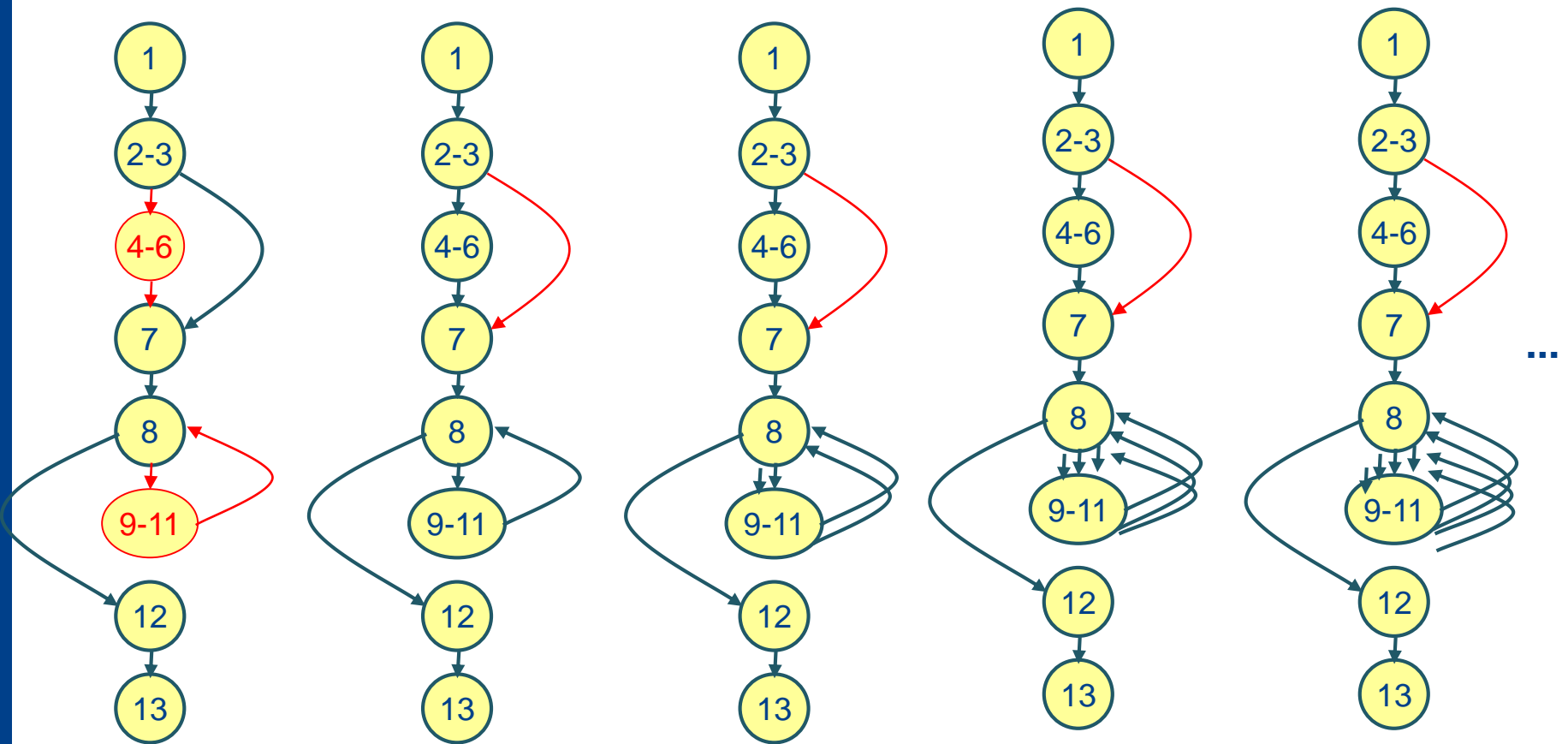
- 100% Pfadüberdeckung (path coverage) bedeutet:
jeder Pfad im Kontrollflussgraphen mindestens einmal ausgeführt

$$\text{Pfadüberdeckung} = \frac{\text{Anzahl durchlaufene Pfade}}{\text{Gesamtzahl Pfade}}$$

- Bei zyklischen Kontrollflussgraphen potenziell unendlich viele Pfade
 - Aber: Obere Grenzen für die Anzahl ggf. aus Spezifikation oder aus technischen Einschränkungen ableitbar
 - bei Schleifen alternativ Grenze-Inneres-Überdeckung verwenden
- In der Praxis nicht zu 100% erreichbar, eher als theoretisches Vergleichsmaß verwenden



Exkurs: Beispiel Pfadüberdeckung für `ggt()`





Exkurs: Von Pfaden zu Testfällen

- Bisher lediglich Pfade durch den Kontrollflussgraphen bestimmt, die durch die Testfälle zur Ausführung gebracht werden sollen
- Frage: Mit welchen Eingaben werden diese Pfade erzwungen?
- Idee: Die Bedingungen der kontrollflussbestimmenden Anweisungen betrachten
- Damit Aussagen über Programmvariablen berechnen (typischerweise rückwärts im Kontrollfluss)

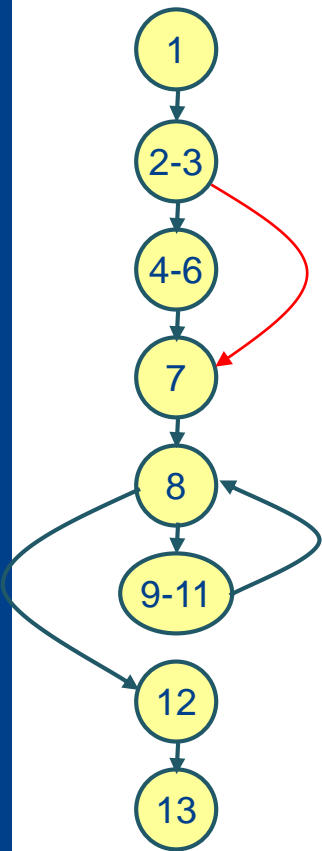


Testfall zur Anweisungsüberdeckung

Pfad: (1, 2-3, 4-6, 7, 8, 9-11, 8, 12, 13)

Logischer Testfall: $\{ n > m \wedge n \% m \neq 0 \wedge n \% (n \% m) = 0 ;$
ggt(m, n) }

Konkreter Testfall: { m = 4, n = 6 ; 2 }



```
1. public int ggt(int m, int n) {
2.     int r;
3.     if (n > m) {
4.         r = m;
5.         m = n;
6.         n = r;
7.     }
8.     while (r != 0) {
9.         m = n;
10.        n = r;
11.        r = m % n;
12.    }
13.    return n;
14. }
```

m	n	r
4	6	?

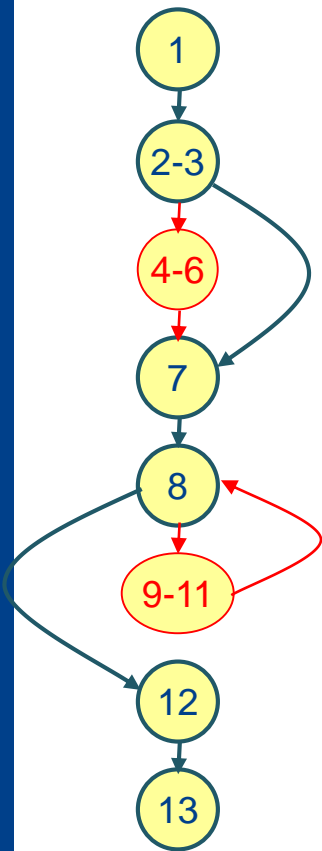


Testfall zur Entscheidungsüberdeckung

Pfad: (1, 2-3, 7, 8, 12, 13)

Logischer Testfall: $\{n \leq m \wedge m \% n = 0 ; \text{ggt}(m, n) \}$

Konkreter Testfall: $\{ m = 4, n = 4 ; 4 \}$



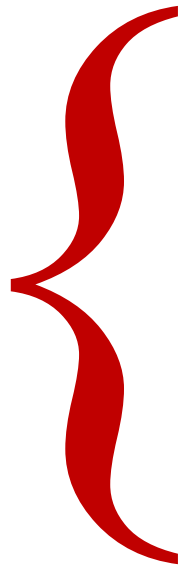
```
1. public int ggt(int m, int n) {
2.     int r;
3.     if (n > m) {
4.         r = m;
5.         m = n;
6.         n = r;
7.     }
8.     r = m % n;
9.     while (r != 0) {
10.        m = n;
11.        n = r;
12.        r = m % n;
13.    }
14.    return n;
15. }
```

m	n	r
4	4	?

White-Box-Tests und Instrumentierung

- White-Box-Testverfahren fordern, dass
 - bestimmte Programmteile zur Ausführung kommen
 - Bedingungen unterschiedliche Wahrheitswerte annehmen
- Instrumentierung durch Werkzeuge
 - Messen der durch Testfälle erreichten Anweisungen
 - Messen der durch Testfälle abgedeckten Entscheidungsausgänge
 - Werkzeuge liefern Metriken für Abdeckung

4.2
Dynamischer
Test
White-Box



Idee der White-Box-Testverfahren

Kontrollflusstest

Exkurs: Datenflusstest

Exkurs: Bedingungstest

Erfahrungsbasierte Testverfahren

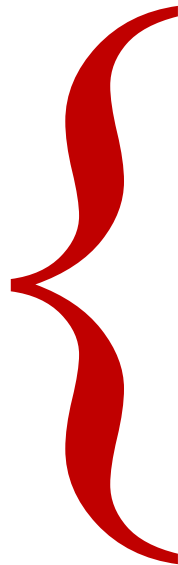
Dynamischer Test: Testverfahrensauswahl
und Zusammenfassung



Exkurs: Datenflusstest

- Testfälle unter Berücksichtigung der Datenverwendungen im Testobjekt herleiten
- Vollständigkeit der Prüfung (Überdeckung) anhand der Datenverwendung bewerten
- Test bezüglich der Variablen-/Objektverwendung
- Wertzuweisung, zustandsverändernd
 - z.B. $r = m$ oder $r = 5$
 - Definitional use, $\text{def}(r)$
- Benutzung in Ausdrücken, zustandserhaltend
 - z.B. $r = m \% n$ oder $r = \text{opl}(m, n)$
 - Computational use, c-use (m, n) und $\text{def}(r)$
- Benutzung in Bedingungen, zustandserhaltend
 - z.B. $\text{while } (r \neq 0)$ oder $\text{if } (r \neq 0)$
 - Predicative use, p-use (r)

4.2
Dynamischer
Test
White-Box



Idee der White-Box-Testverfahren

Kontrollflusstest

Exkurs: Datenflusstest

Exkurs: Bedingungstest

Erfahrungsbasierte Testverfahren

Dynamischer Test: Testverfahrensauswahl
und Zusammenfassung



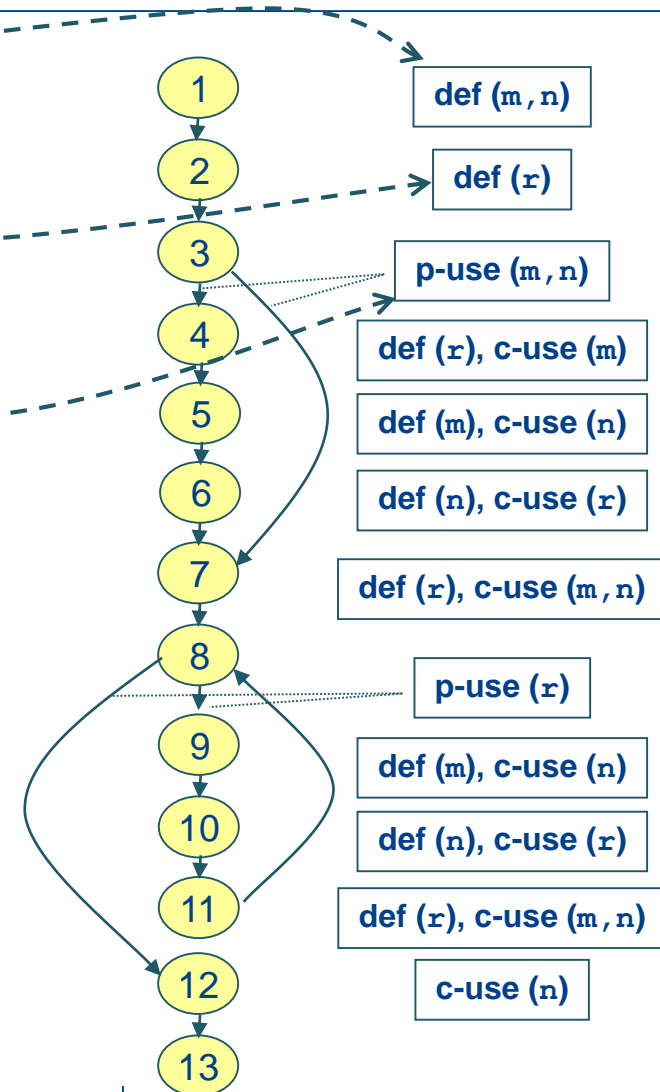
Exkurs: Bedingungen in Programmen und Spezifikationen

- Wahrheitswerte: `false`, `true` (oft auch `0`, `1` oder falsch, wahr)
- Atomare (Teil-)Bedingung (*condition*)
 - Variablen vom Typ `boolean`
 - Operationen mit Rückgabewert vom Typ `boolean`
 - Vergleichsoperationen
 - Z.B. `flag; isEmpty(); size > 0`
- Zusammengesetzte Bedingung (*compound condition*)
 - Verknüpfung von (Teil-)Bedingungen mit booleschen Operatoren
 - Basis-Operatoren sind **und** (\wedge , \cap), **oder** (\vee , \cup), **nicht** (\neg)
- Entscheidung ist (zusammengesetzte) Bedingung, die den Programmablauf steuert
- In Java
 - `&`, `|`, `^` Bitweise und, oder und exklusiv-oder-Verknüpfung
 - `&&`, `||` Wie oben, aber lazy evaluation, z.B. `a && b` entspricht `a ? b : false`
 - z.B. **if** `((size > 0) && (inObject != null)) {...} else {...}`



Exkurs: Kontrollflussgraph mit Datenflussannotation

```
1. public int ggt (int m, int n) {  
  // pre: (m > 0) and (n > 0)  
  // post: ...  
2.   int r;  
3.   if (n > m) {  
4.     r = m;  
5.     m = n;  
6.     n = r;  
7.   }  
8.   r = m % n;  
9.   while (r != 0) {  
10.    m = n;  
11.    n = r;  
12.    r = m % n;  
13.  }  
14.  return n;  
15. }
```

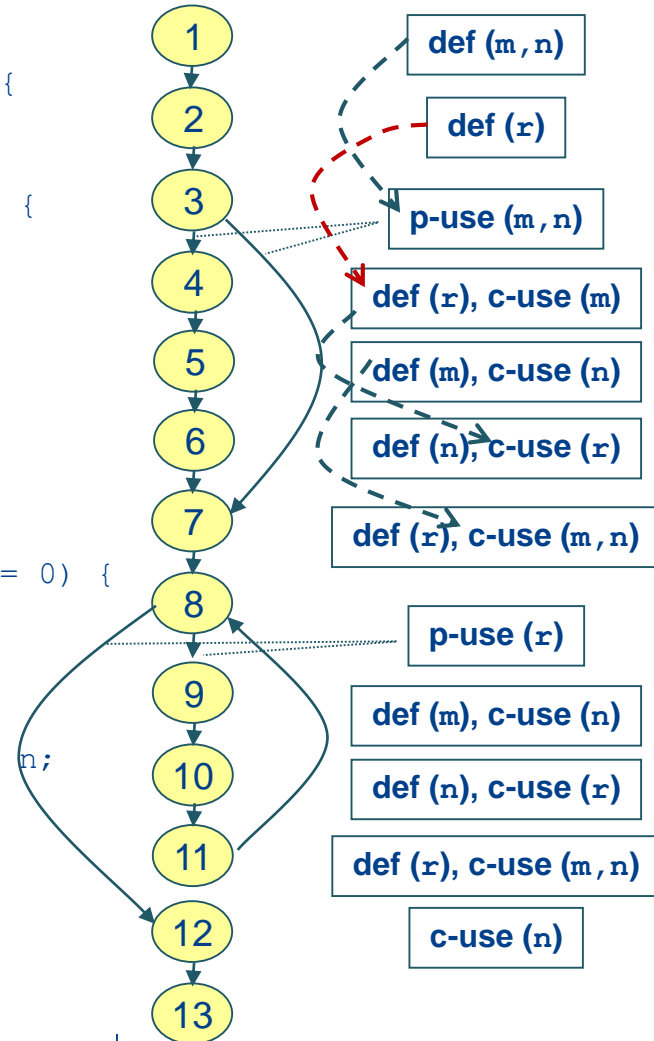




Exkurs: Datenflussbezogene Überdeckungskriterien

- Hypothese: fehlerhafte Datenverwendung!
- All-defs Kriterium: Jede Definition mindestens einmal ohne dazwischen liegendes erneutes def in einem c-use oder p-use verwendet
- Beispiel:
 - All-defs:
 - 1, def m: p-use 3-4
 - 1, def n: p-use 3-4
 - 4, def r: c-use 6
 - 5, def m: c-use 7
 - 6, def n: c-use 7
 - 7, def r: p-use 8-9
 - 9, def m: c-use 11
 - 10, def n: c-use 11
 - 11, def r: p-use 8-9
- Weitere Kriterien:
All p-uses / All c-Uses / All uses / ...

```
1. public int ggt
   (int m, int n) {
2.     int r;
3.     if (n > m) {
4.         r = m;
5.         m = n;
6.         n = r;
7.     }
8.     r = m % n;
9.     while (r != 0) {
10.        m = n;
11.        n = r;
12.        r = m % n;
13.    }
14.    return n;
15. }
```





Exkurs: Bedingungstest

Bedingungstest Ein White-Box-Testentwurfsverfahren, bei dem Testfälle so entworfen werden, dass Bedingungsaustritte zur Ausführung kommen.

- Entscheidungsüberdeckung berücksichtigt ausschließlich Ergebnis-Wahrheitswert der Gesamtentscheidung
- Problem:
Strukturelle Komplexität aus logisch verknüpften Teilbedingungen wird nicht berücksichtigt
- Bedingungstest ermöglicht Abstufungen der Testintensität mit Berücksichtigung der zusammengesetzten (Teil-)Bedingungen



Exkurs: Bedingungstest

- Überdeckungskriterien sind Verhältnisse zwischen den bereits erreichten und geforderten Wahrheitswerten der (Teil-)Bedingungen
- Wenn Komplexität der Bedingungen im Mittelpunkt der Prüfung, dann vollständige Überdeckung (100%) anstreben
- Wir betrachten:
 - (Einfache) Bedingungsüberdeckung
 - *condition coverage*
 - Mehrfachbedingungsüberdeckung
 - *multiple condition coverage*
 - Minimal bestimmende Mehrfachbedingungsüberdeckung
 - *condition determination coverage*



Exkurs: Einfacher Bedingungstest

Einfacher Bedingungstest Ein White-Box-Testentwurfsverfahren, bei dem Testfälle so entworfen werden, dass Bedingungsaustritte zur Ausführung kommen.

- Überdeckung der atomaren Teilbedingungen einer Entscheidung mit wahr und falsch gefordert
 - Teste jede atomare Bedingung einmal zu wahr und einmal zu falsch
- Bei n atomaren Bedingungen mindestens 2, höchstens $2n$ Testfälle

$$\text{Bedingungsüberdeckung} = \frac{\text{Anzahl zu wahr und falsch getesteten atom. Bed.}}{\text{Gesamtzahl atomarer Bedingungen}}$$



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



Exkurs: Beispiele zur einfachen Bedingungsüberdeckung

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

2 Bedingungen, 2 Testfälle

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

2 Bedingungen, 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

3 Bedingungen, 2 Testfälle



Exkurs: Mehrfachbedingungsüberdeckung

Mehrfachbedingungstest Ein White-Box-Testentwurfsverfahren, das die Überdeckung der atomaren Teilbedingungen einer Entscheidung mit WAHR und FALSCH in allen Kombinationen fordert. (ISTQB Glossar V3.2)

- Bei n atomaren Bedingungen ist die Testfall-Anzahl $= 2^n$
 - Wächst exponentiell mit der Anzahl unterschiedlicher atomarer Ausdrücke!

$$\text{Mehrfachbedingungsüberdeckung} = \frac{\text{Anzahl getesteter Kombinationen atom. Bed.}}{2^{\text{Gesamtzahl atomarer Bedingungen}}}$$

- Bei der Auswertung der Gesamtbedingung ergeben sich i.d.R. auch beide Wahrheitswerte (sonst: Tautologie!)
 - Die Mehrfachbedingungsüberdeckung erfüllt somit auch die Kriterien der Anweisungs- und Entscheidungsüberdeckung
 - Sie ist ein umfassenderes Kriterium, da sie auch die Komplexität bei zusammengesetzten Bedingungen berücksichtigt
- Problem: Manche Kombinationen sind nicht durch konkrete Testfälle realisierbar, wenn Teilbedingungen voneinander abhängig sind
 - Z.B. bei $(x > 0) \ \&\& \ (x < 5)$ ist (falsch, falsch) nicht realisierbar





Exkurs: Beispiel: Mehrfachbedingungsüberdeckung

Teste jede Kombination der Wahrheitswerte aller atomarer Bedingungen!

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

2 Bedingungen, 4 Testfälle

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

2 Bedingungen, 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

3 Bedingungen, 8 Testfälle

Exkurs: Minimal bestimmende Mehrfachbedingungsüberdeckung

Minimal bestimmende Mehrfachbedingungsüberdeckung Der Anteil aller einfachen Bedingungsergebnisse, die von einer Testsuite ausgeführt wurden und unabhängig voneinander einen Entscheidungsausgang beeinflussen.

- Teste die (Gesamt-)Bedingung einmal zu wahr und einmal zu falsch sowie jede Kombination von Wahrheitswerten, bei denen die Änderung des Wahrheitswertes einer atomaren Bedingung den Wahrheitswert der (Gesamt-)Bedingung ändern kann (MM-Kombinationen)!

$$\text{Minimalbest. Mehrfachbed.überdeckung} = \frac{\text{Anzahl getesteter MM-Kombinationen}}{\text{Gesamtzahl MM-Kombinationen}}$$

- 100% minimal bestimmende Mehrfachbedingungsüberdeckung
⇒ 100% Entscheidungsüberdeckung
 - Voraussetzung: (Gesamt-)Ausdruck jeweils mindestens einmal zu wahr und falsch ausgewertet (notwendig, falls keine MM-Kombination existiert)

Exkurs: Beispiel: Minimal bestimmende Mehrfachbedingungsüberdeckung

Teste die (Gesamt-)Bedingung einmal zu wahr und einmal zu falsch sowie jede Kombination von Wahrheitswerten, bei denen die Änderung des Wahrheitswertes einer atomaren Bedingung den Wahrheitswert der (Gesamt-)Bedingung ändern kann!

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

2 Bedingungen, 3 Testfälle

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

2 Bedingungen, 3 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

3 Bedingungen, 4 Testfälle

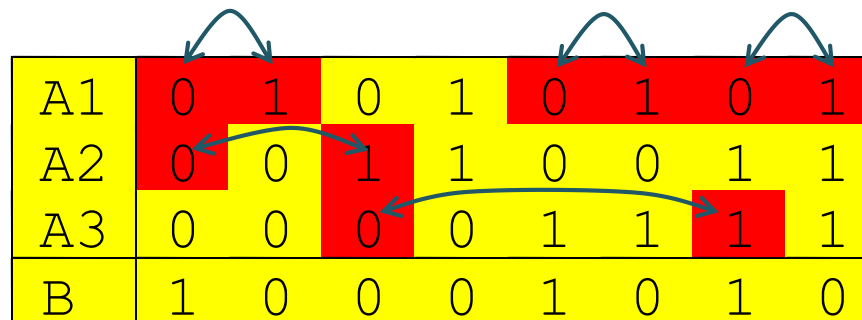
Exkurs: Übung: Minimal bestimmende Mehrfachbedingungsüberdeckung

- Seien A1, A2, A3 atomare Bedingungen und $B=F(A1, A2, A3)$ die durch die unten abgebildete Wahrheitstabelle definierte (Gesamt)Bedingung
- Frage: Welche Werte-Kombinationen von A1, A2 und A3 sind nach der minimal bestimmenden Mehrfachbedingungsüberdeckung mit Testfällen zu bewirken?

A1	0	1	0	1	0	1	0	1
A2	0	0	1	1	0	0	1	1
A3	0	0	0	0	1	1	1	1
B	1	0	0	0	1	0	1	0

Exkurs: Lösung: Minimal bestimmende Mehrfachbedingungsüberdeckung

- Antwort: Mit Testfällen alle Kombinationen (Spalten) bewirken, in denen mindestens eine rote Markierung ist



The diagram shows a 4x8 matrix with rows labeled A1, A2, A3, and B. The columns contain binary values (0 or 1). Cells are colored red or yellow. Arrows indicate dependencies between columns: a curved arrow from column 1 to 2, a curved arrow from column 5 to 6, a curved arrow from column 7 to 8, a straight arrow from column 1 to 3, and a curved arrow from column 3 to 7.

A1	0	1	0	1	0	1	0	1
A2	0	0	1	1	0	0	1	1
A3	0	0	0	0	1	1	1	1
B	1	0	0	0	1	0	1	0



Exkurs: Hinweis

- Die in der DO-178C (Software Considerations in Airborne Systems and Equipment Certification) und ISO 26262 (Road Vehicles – Functional Safety) geforderte *Modified Condition/Decision Coverage* (MC/DC) ist ähnlich der Minimal bestimmenden Mehrfachbedingungsüberdeckung (*condition determination coverage*)
 - Fordert aber nur, für jede atomare Bedingung in mindestens einem Fall zu zeigen, dass sie alleine die (Gesamt-)Bedingung zu wahr und falsch beeinflussen kann
 - Es werden zwei MM-Kombinationen pro atomarer Bedingung benötigt
 - Bei n atomaren und/oder/nicht-verknüpften Bedingungen ist die Testfall-Anzahl bei MC/DC mindestens $n+1$ und höchstens $2n$ (also nur linear wachsend!)

A1	0	1	0	1	0	1	0	1
A2	0	0	1	1	0	0	1	1
A3	0	0	0	0	1	1	1	1
B	1	0	0	0	1	0	1	0



Exkurs: Bewertung der Bedingungstestentwurfverfahren (1 von 2)

- Intensiver dynamischer Test komplexer Bedingungen verzichtbar, wenn die Korrektheit durch Code-Review nachgewiesen
- Weitere Alternative: Entscheidungen mit komplexen zusammengesetzten Bedingungen in äquivalente, verschachtelte Entscheidungen mit einfachen Bedingungen umwandeln und einen Entscheidungstest durchführen
- Nachteil der Bedingungsüberdeckungen: boolesche Ausdrücke nur innerhalb einer Entscheidung betrachtet
 - Wird die Bedingung *vor* der Entscheidung berechnet, könnte sie aus mehreren Teilbedingungen zusammengesetzt sein, weshalb dann die minimal bestimmende Mehrfachbedingungsüberdeckung anzuwenden wäre, z.B. bei
`flag = a || (b && c); if (flag) {...}`
 - Daher alle booleschen Ausdrücke im Code bei der Testfallermittlung betrachten!



Exkurs: Bewertung der Bedingungstestentwurfverfahren (2 von 2)

- Herausforderung: Erreichen von 100% Überdeckung der Teilbedingungen
 - Einige Programmiersprachen und Compiler verkürzen die Auswertung von booleschen Ausdrücken, sobald das Ergebnis feststeht (lazy evaluation / short circuit evaluation)
 - Beispiel: Bei $a = \text{false}$ ist die Bedingung $a \text{ AND } b$ auch false, egal welchen Wert b hat, also wird b gar nicht ausgewertet
 - Einige Compiler ändern die Reihenfolge der Auswertung der booleschen Operatoren, um möglichst schnell ein Endergebnis zu erhalten und weitere Teilbedingungen nicht auswerten zu müssen
 - Testfälle für eine Überdeckung von 100% sind zwar ausführbar, wegen der Verkürzung der Auswertung ist die Überdeckung allerdings nicht nachweisbar



Exkurs: Bedingungsüberdeckung und Lazy Evaluation

- Lazy Evaluation bedeutet, dass Bedingungen nur so lange geprüft werden, bis der Wahrheitswert feststeht
 - Im Programm: `if (A && B) then op1(); op2() ...`
 - Im Objectcode: `if (A) then if (B) then op1(); op2() ...`
- Hier müssen bei der einfachen BÜ drei Fälle verwendet werden, da sonst B=1 nicht wirklich im Programm ausgewertet wird (Abbruch, sobald A=0 erkannt ist)
 - Damit wird auch der Gesamtausdruck zu 0 und 1 ausgewertet
 - Mehrfach-BÜ und MM-BÜ sind nicht erreichbar

Abbruch der
Auswertung

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

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

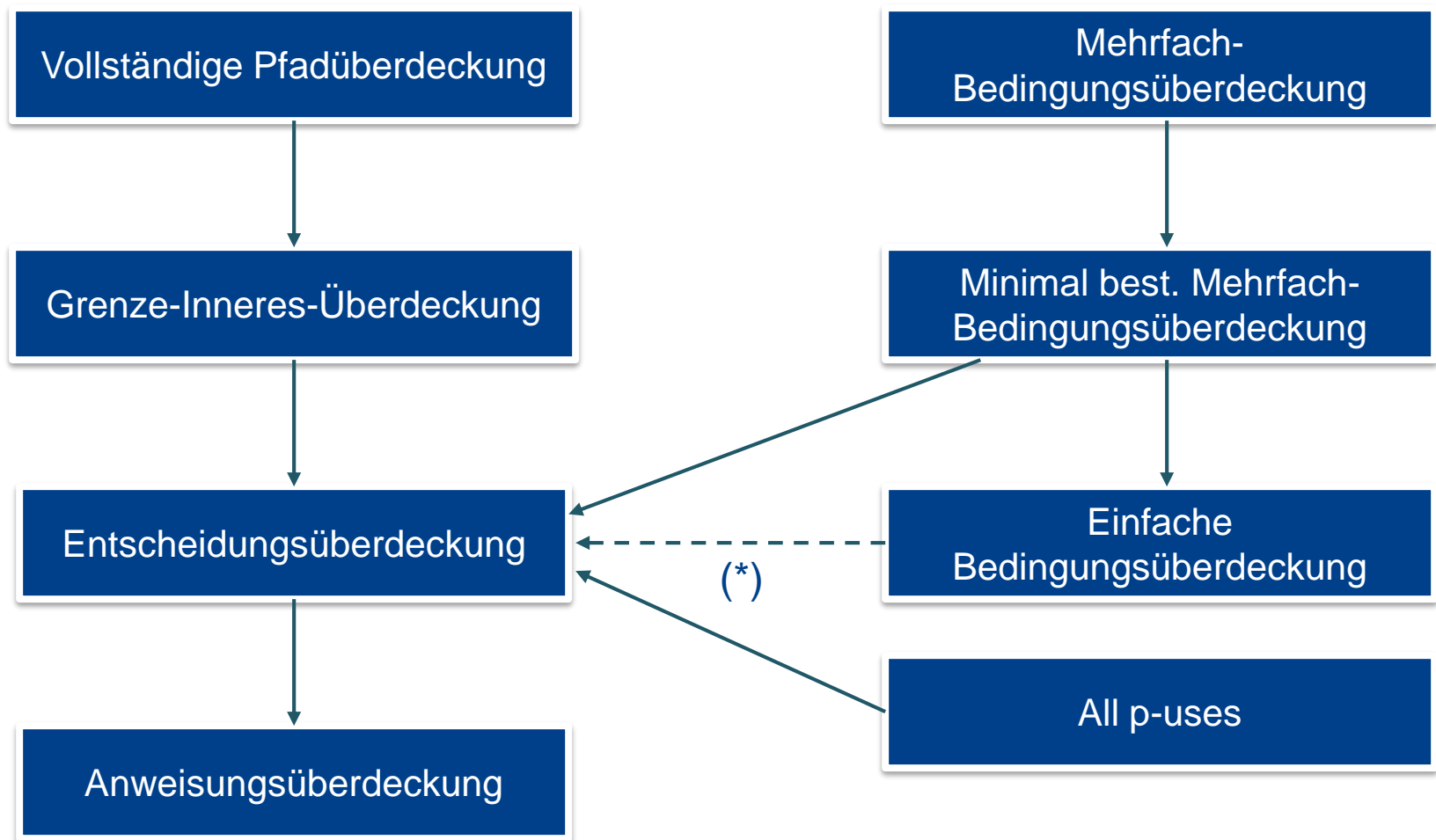


Zusammenfassung: White-Box-Testverfahren

- Grundlage aller White-Box-Testverfahren ist die Programmstruktur
- Gemäß der Komplexität der Programmstruktur passende Testverfahren auswählen
 - Entscheidungsüberdeckung bei IF-Anweisungen mit leeren ELSE-Zweigen
 - Minimal bestimmende Mehrfachbedingungsüberdeckung bei komplexen Bedingungen
 - Intensität des Tests anhand Programmstruktur und ausgewählter Testverfahren festgelegt
 - Empfehlung: Entscheidungsüberdeckung als minimales Kriterium verwenden
- White-Box-Testverfahren eher für die unteren Teststufen geeignet
- Problem: Nicht vorhandener Programmcode bleibt unberücksichtigt
 - Nicht umgesetzte Anforderungen fallen durch White-Box-Testverfahren nicht auf
 - Nur im Programm umgesetzte Anforderungen können bei den White-Box-Testverfahren überprüft werden
- Zur Instrumentierung Werkzeuge verwenden!
- Bei den White-Box-Testverfahren soll die Struktur des Testobjektes die Grundlage bei der Auswahl der Testverfahren sein – sie sind z.B. das adäquate Testverfahren, um fehlerhafte Bedingungen zu erkennen

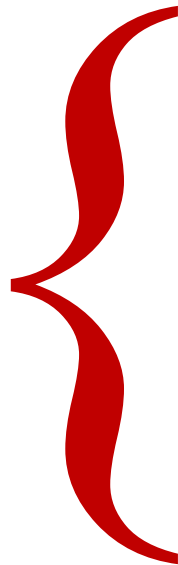


Mächtigkeit der White-Box-Testverfahren



(*) Bei lazy evaluation

4.2
Dynamischer
Test
White-Box



Idee der White-Box-Testverfahren

Kontrollflusstest

Exkurs: Datenflusstest

Exkurs: Bedingungstest

Erfahrungsbasierte Testverfahren

Dynamischer Test: Testverfahrensauswahl
und Zusammenfassung



Kategorien von Testfallentwurfsverfahren



Photo: Fotolia / Dan Race



Source: <http://www.operation.de/knie/>

- **Black-Box-Testverfahren**

- Spezifikationen sind Grundlage für den Testentwurf, aber nicht der Programm Quelltext oder sein innerer Aufbau
- z.B. Äquivalenzklassenbildung und Grenzwertanalyse

- **White-Box-Testverfahren**

- Programm Quelltext liegt vor und ist Grundlage für den Testentwurf
- z.B. Entscheidungsüberdeckung oder Anweisungsüberdeckung

- **Erfahrungsbasierte Testverfahren**

Erfahrungsbasierte Testverfahren (1 von 2)

- Der Erfolg systematischer Testverfahren hängt stark von der Qualität der Testbasis ab
 - Testbasis muss überhaupt verfügbar sein!
 - schwierig z.B. bei Legacy-Systemen, oft bei „agiler“ Entwicklung
 - Testbasis muss hinreichend korrekt, lesbar, vollständig, eindeutig, widerspruchsfrei und testbar sein
 - Formalisierungsgrad und Granularität muss ausreichen, um formale bzw. systematische Testverfahren anzuwenden
- Erfahrungsbasierte Testverfahren ergänzen systematische Testverfahren
 - Kompensieren fehlende Testbasis oder Mängel der Testbasis
 - Zeigen Fehlerwirkungen auf, die systematischer Test typischerweise übersieht

Erfahrungsbasierte Testverfahren (2 von 2)

- Grundidee: Testbasis sind auch Erfahrung, Kenntnisse und Intuition der Tester, Entwickler und Benutzer, z.B.
 - erwartete Nutzung der Software und ihrer Umgebung
 - mögliche Fehlerzustände und ihre Verteilung
- Beispiele für Erfahrung (Heuristiken)
 - Neue Funktionalität fehleranfälliger als reife Funktionalität
 - Geänderte Funktionalität fehleranfälliger als reife Funktionalität
 - Späte Änderungen verursachen Fehler
 - Fehler häufig an Grenzwerten
- Wichtige erfahrungsbasierte Testverfahren
 - Intuitive Testfallermittlung (error guessing)
 - Exploratives Testen (exploratory testing)
 - Checklistenbasiertes Testen

Intuitive Testfallermittlung: Charakterisierung

- Tester vermuten aufgrund von Erfahrung, Intuition und Kenntnissen über das Testobjekt, welche Fehlerwirkungen oder Fehlerzustände im Testobjekt aufgrund von Fehlhandlungen vorkommen
 - Leiten Testfälle ab, die diese Fehler aufdecken können
 - Qualität der Testfälle abhängig von Qualität der Tester!
- Beispiele für Erfahrung / Kenntnisse
 - Wie hat das Testobjekt früher funktioniert? Welche Fehlerwirkungen gab es damals?
 - Welche Fehlerwirkungen gab es in ähnlichen Testobjekten?
 - Welche Fehlhandlungen machen Entwickler üblicherweise?

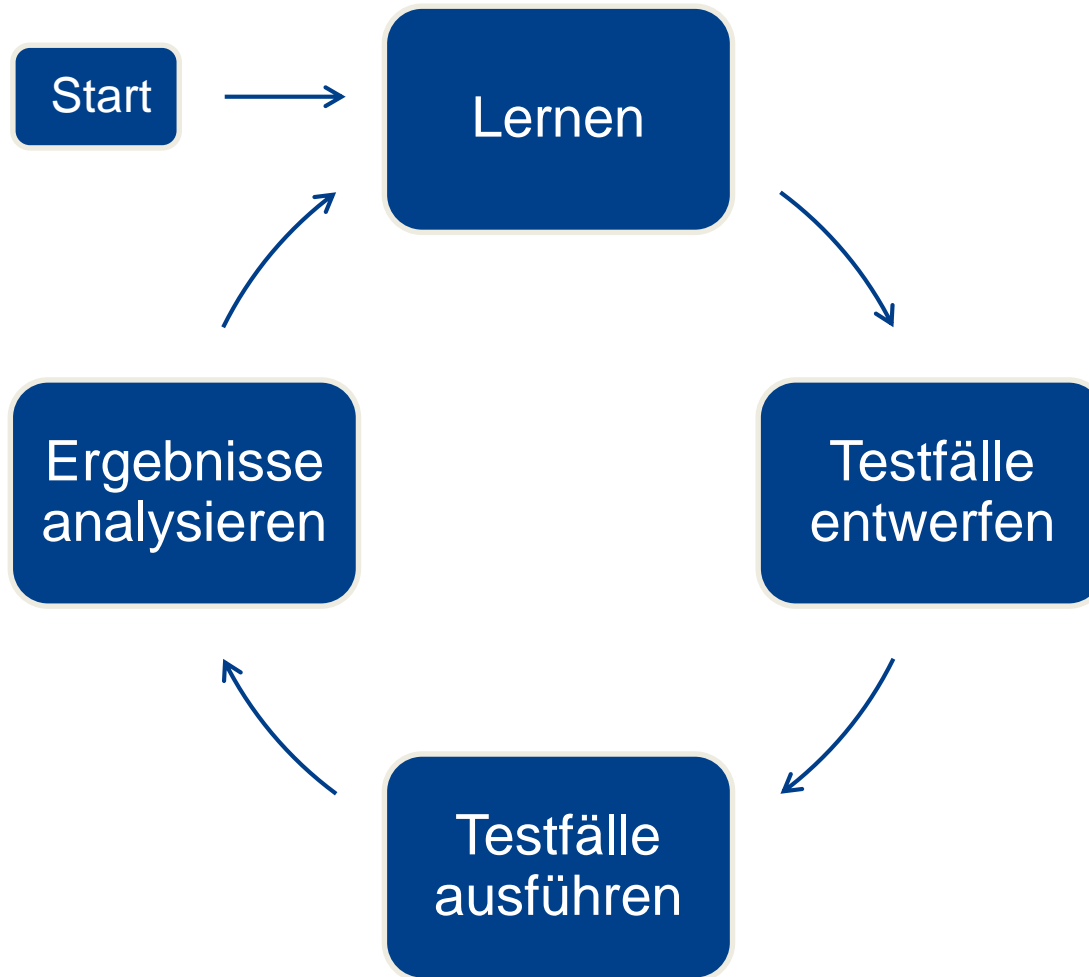
Intuitive Testfallermittlung: Vorgehensweise

- Strukturierte Herangehensweise
 1. Liste möglicher Fehlhandlungen, Fehlerzustände und Fehlerwirkungen erstellen (Fehlerkatalog)
 2. Daraus Testfälle ableiten, die auf diese Fehler abzielen
- Basis für Fehlerkatalog
 - Erfahrung der Tester
 - Verfügbaren Daten über Fehlerzustände und Fehlerwirkungen, z.B.
 - Fehlerstatistiken aus früheren Projekten
 - Liste von Software-Sicherheitsschwachstellen
 - Allgemeinwissen zu Software-Fehlern, z.B.
 - Division durch null, Null-Zeiger, Stapelüberlauf, Speicherleck, Deadlock
- Fehlerkatalog sollte
 - firmen- oder projektspezifisch sein
 - regelmäßig aktualisiert werden

Exploratives Testen: Charakterisierung

- Informelles Testen ohne vorbereitete Testfälle, insb. ohne Sollergebnisse
- Testfallanalyse, Testentwurf, Testrealisierung und Testdurchführung laufen parallel
 - Tester lernt beim Testen das Testobjekt kennen, testet immer besser
 - Erfahrung und Heuristiken unterstützen die spontane Testfallfindung
 - Häufig sitzungsbasiert, d.h. Test innerhalb eines festen Zeitfensters
 - Meistens mit Test-Charta mit Testziele und Testideen
- Geeignet, wenn
 - zu wenig oder ungeeignete Spezifikationen vorhanden
 - Test unter hohem Zeitdruck steht
 - andere, formale Testverfahren zu unterstützen/ergänzen sind

Exploratives Testen: Vorgehen



Exploratives Testen: Testvorbereitung

Einheiten

Abgeschlossene Teile des Produkts

Testaufwand 1-2 Tage

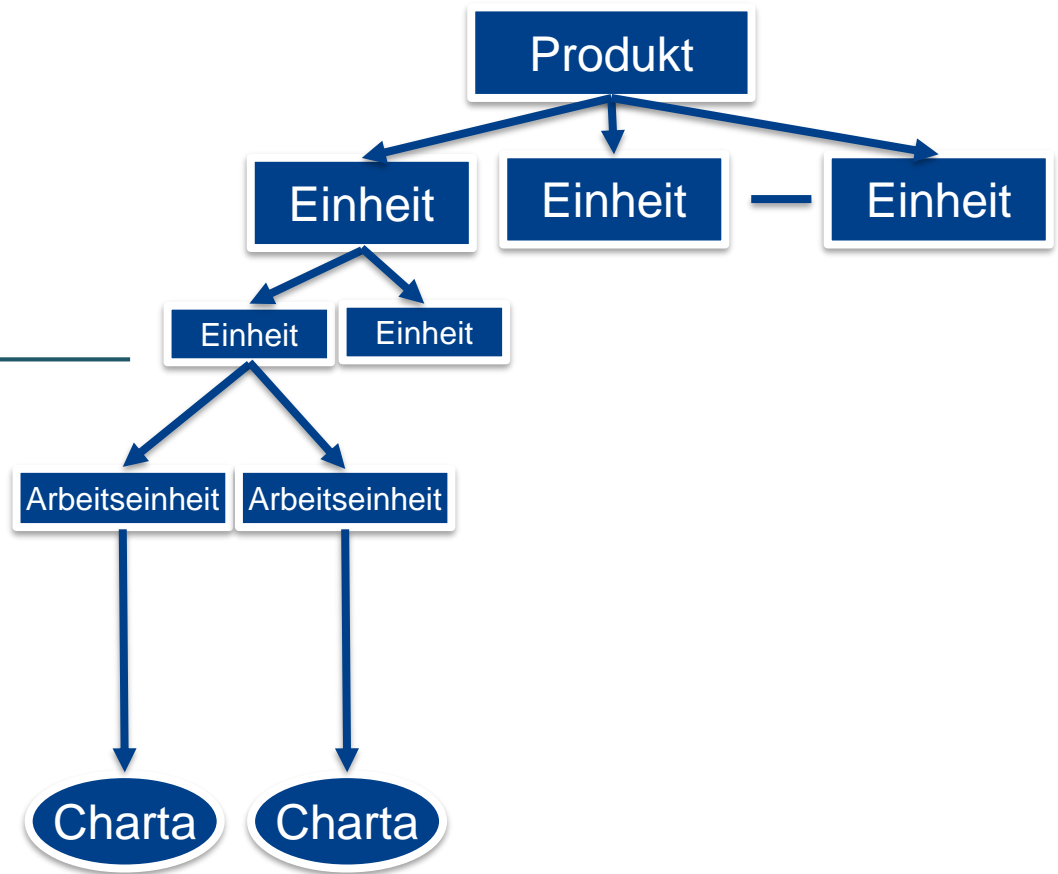
Arbeitseinheit

Innerhalb einer Sitzung abgearbeitet

Testaufwand 1-2 Stunden

Charta

Gibt Ziel und Aufgabe
einer Sitzung vor



Exploratives Testen: Test-Charta

- **Gibt Ziel und Aufgabe einer Sitzung vor**
 - Warum soll die entsprechende Einheit getestet werden?
 - Was soll getestet werden?
 - Wie getestet werden soll (Testverfahren)?
 - Welche Probleme sollen adressiert werden?
- **Anfangs allgemein zum Kennenlernen des Testobjekts**
 - Experimentelle Überprüfung der Grundfunktionalität
 - Beispiel Textverarbeitung: Analysiere Grafik-Einfüge-Funktion
- **Später detaillierter**
 - Detaillierter Test der Funktionalität
 - Test von Qualitätsanforderungen wie Performanz oder Gebrauchstauglichkeit
 - Beispiel: Teste Einfügen Clipart aus Katalog, Einfügen Bild aus Datei

Exploratives Testen: Testdurchführung

- **Zeitlich begrenzte Sitzung**
 - Kurz: 60 min, normal: 90 min, lang: 120 min
 - Basiert typischerweise auf einer Test-Charta
 - Flexibel auf Testergebnisse reagieren
 - Ohne Unterbrechung (z.B. Email, Telefon)
- **Sitzungsprotokoll**
 - Tester, Beginn/Ende der Sitzung
 - Test-Charta, Testobjekte
 - Testdaten
 - Notizen (Was gemacht? Warum? Hinweise, Fragen, Anomalien, aufgetretene Schwierigkeiten, ...)
 - Fehlerberichte (genug Details, um Tests wiederholen zu können)
- **Testauswertung nach der Sitzung: Lessons Learned?**

- **Vorteile**

- Kurzfristig einsetzbar, erfordert keine lange Vorplanung
- Anwendbar, wenn wenig Dokumentation oder Domänenwissen vorhanden (Kennenlernen des Produktes, Exploration)
- Fördert Kreativität und Spontaneität
- In Gruppen Synergien zwischen erfahrenen und unerfahrenen Testern

- **Schwächen**

- Verfall in bestimmte Denkmuster behindert das Aufdecken von Fehlern
- Anhängig von Wissen und Erfahrung der Tester
- Nicht automatisierbar

Checklistenbasiertes Testen: Charakterisierung

- **Tests entwerfen, realisieren und ausführen, um Testbedingungen aus einer Checkliste abzudecken**
 - Interpretation der eher abstrakten Checklisten-Punkte anhand der sonstigen vorhandenen Testbasis zum Testobjekt
 - Checkliste bereits vorhanden oder im Rahmen der Testanalyse zu erstellen
 - Ähnliche Vorgehensweise zur Erstellung der Checkliste wie beim Fehlerkatalog für das intuitive Testen:
eigene Erfahrung und Kenntnisse zum Testobjekt auswerten
 - Checkliste formalisiert Kenntnisse und Erfahrung
 - Checkliste stellt eine gewisse Testüberdeckung sicher
- **Testendekriterium = vollständige Abarbeitung der Checkliste**

Checklistenbasiertes Testen: Inhalte Checkliste

- Eine abstrakte, verallgemeinerte Liste mit Punkten, die beachtet und geprüft werden müssen
- Kann auch ein Satz von Vorschriften und Kriterien sein, gegen die das Testobjekt geprüft werden muss
- Quellen für Checklisten-Punkte sind z.B.
 - Erfahrung
 - Wissen über typische Nutzung des Testobjekts
 - Typische Fehlhandlungen und Fehlerzustände
 - Standards und Normen
- Beispiel: Checkliste auf Basis einer Richtlinie für Benutzungsschnittstellen
 - Meldungsdialoge: Schaltflächen bei allen festgelegten Bildschirmauflösungen lesbar?
- Tipp: Checklisten über mehrere Testprojekte hinweg weiterentwickeln, um neue Erfahrungen einzubringen und Veraltetes zu entfernen



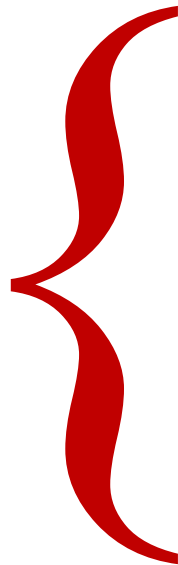
Checklistenbasiertes Testen: Bewertung

- Vorteile
 - Checkliste gibt Testern Orientierung und hilft, nichts zu vergessen
 - Allgemeine Natur der Checkliste positiv für Testüberdeckung, wenn unterschiedliche Tester daraus unterschiedliche Testfälle ableiten
 - Checkliste über Testprojekte hinweg wiederverwendbar
 - Unterstützt verschiedene Testarten, z.B. funktionalen Test oder nicht-funktionalen Test
- Schwächen
 - Erfahrene Tester nötig, um Checkliste zu interpretieren
 - Allgemeine Natur der Checkliste schlecht für Wiederholbarkeit
 - Checklisten neigen über die Zeit zum Wachsen → regelmäßig überarbeiten
 - Vorhandene Checkliste muss nicht zum Testobjekt passen

Bewertung erfahrungsbasierter Testverfahren

- Eher in den höheren Teststufen anzuwenden
 - da auf den niedrigeren meist ausreichende Informationen für die anderen Testverfahren zur Verfügung stehen, beispielsweise der Programmtext oder eine detaillierte Spezifikation
- Erfahrungsbasierte Testverfahren unabhängig vom Testobjekt mit gutem Erfolg einsetzbar
- Vor- und Nachbedingungen, erwartete Ausgaben und erwartetes Verhalten des Testobjektes oft schwierig festzulegen
- Kriterium der Intensität/Vollständigkeit der Testfallermittlung fehlt
 - klares Endekriterium wie bei den systematischen Testverfahren fehlt
 - aber: auf Basis Fehlerkatalog oder Checkliste Vollständigkeit prinzipiell überprüfbar
- Fazit: Nicht als primäre Testverfahren einsetzen, sondern systematische Testverfahren damit abrunden / unterstützen

4.2
Dynamischer
Test
White-Box



Idee der White-Box-Testverfahren

Kontrollflusstest

Exkurs: Datenflusstest

Exkurs: Bedingungstest

Erfahrungsbasierte Testverfahren

Dynamischer Test: Testverfahrensauswahl
und Zusammenfassung

Worin unterscheiden sich Testverfahren?

TESTSTART	Welche Art von Tests werden durchgeführt?	Funktionale Anforderungen, Nicht-funktionale Anforderungen
TESTSTUFE	Für welche Testobjekte wird der Test spezifiziert?	Komponententest, Integrationstest, Systemtest, Abnahmetest
TESTER	WER testet?	Entwickler, (erfahrene) Tester, Benutzer, ...
TESTÜBER-DECKUNG	WAS wird übergedeckt?	Anforderung, Modell, Anweisung, Entscheidung, ...
POTENZIELLE FEHLER	Welche potenziellen Fehler sollen identifiziert werden?	Behandlung von Grenzwerten, Behandlung von Ausnahmen, ...
ARTEFAKT	Was ist Grundlage für Auswahl und Herleitung der Testfälle?	Anforderungen → Black-Box, Code → White-Box, ...
DOMÄNE / PARADIGMA	Für welche spezielle Domäne bzw. Entwicklungsparadigma ist das Verfahren zugeschnitten?	OOP, Web-basiert, DB-basiert, Automotive, Sicherheitskritische SW, ...

Kriterien zur Auswahl eines Testverfahrens

- Testart, Teststufe, Tester, Abdeckung, potenzielle Fehler, Artefakte, Charakteristika des Produkts (Domäne, Technologie)
- Verfügbare Dokumentation und ihre Qualität
- Qualifikation/Erfahrung der Tester
- Regulatorische Anforderungen / Standards, Kunden- / Vertragsanforderungen
- Testziele, Projekt- und Produktrisiken
- Angestrebter Automatisierungsgrad
- Testbudget (Zeit und Geld)
- Zu findende Arten von Fehlerwirkungen
- Empfehlung: Unterschiedlicher Testverfahren kombinieren
 - verschiedene Verfahren finden unterschiedliche Fehler
 - insgesamt bessere Abdeckung des Testobjekts



Zusammenfassung dynamischer Test (1 von 2)



- Ziel: Mit wenig Aufwand ausreichend unterschiedliche Testfälle zu erzeugen, die mit gewisser Wahrscheinlichkeit vorhandene Fehlerzustände zur Wirkung bringen
- Systematische Testverfahren sind die Basis für die Testfallermittlung
 - Black-Box: Testfälle abgeleitet aus den Anforderungen
 - White-Box: Testfälle abgeleitet aus der inneren Struktur
- Unterschiedliche Testverfahren verwenden, kein Testverfahren kann alle Aspekte gleich gut abdecken
- Testverfahren und die Intensität der Durchführung anhand der Kritikalität und des zu erwartenden Risikos im Fehlerfall festlegen
- Erfahrungsbasierte Testverfahren ergänzend einsetzen, um weitere Fehler aufzudecken



Zusammenfassung dynamischer Test (2 von 2)



- Empfehlungen
 - Ausreichende Prüfung der Funktionalität des Testobjektes gewährleisten
 - Testfälle mit erwarteten Ergebnissen / Reaktionen des Testobjektes versehen, damit Auswertung der durchgeführten Testfälle möglich
 - White-Box-Testverfahren eher auf den unteren Teststufen einsetzen, auf den oberen Teststufen eher Black-Box-Testverfahren
- Vorgehen auf unteren Teststufen: erst Black-Box, dann White-Box
 - Äquivalenzklassenbildung in Kombination mit der Grenzwertanalyse zur Erstellung der Testfälle einsetzen
 - Haben unterschiedliche Zustände einen Einfluss auf das Verhalten des Testobjekts, zusätzlich zustandsbasierten Test durchführen
 - Bei Ausführung dieser Testfälle die erreichte Anweisungs- und Entscheidungsüberdeckung messen
 - darauf achten, dass Schleifen auch mehr als einmal durchlaufen werden
 - Bisher nicht ausgeführte Teile des Testobjekts einem White-Box-Testverfahren unterziehen



Folgende Fragen sollten Sie jetzt beantworten können

- Was bedeutet der Begriff Anweisungsüberdeckung?
- Worin unterscheiden sich Anweisungs- und Entscheidungsüberdeckung?
- Nach welcher Formel wird die erreichte Anweisungsüberdeckung berechnet?
- Wozu dient die Instrumentierung beim White-Box-Test?
- Exkurs: Worauf zielt die Bedingungsüberdeckung ab?
- Exkurs: Worin unterscheiden sich die einfache Bedingungsüberdeckung und die Mehrfachbedingungsüberdeckung?
- Was ist unter erfahrungsbasiertem Testen zu verstehen? Nennen Sie drei Beispiele für Verfahren zum erfahrungsbasierten Testen.
- Warum sollten systematische und erfahrungsbasierte Testverfahren kombiniert werden?



Muster-Prüfungsfragen

Testen Sie Ihr Wissen...



Frage 1

21. Die folgende Aussage bezieht sich auf Entscheidungsüberdeckung:

"Wenn der Code nur eine einzige IF-Anweisung und keine Schleifen oder CASE-Anweisungen enthält und auch sonst durch den Test nicht geschachtelt aufgerufen wird, dann wird bei jedem einzelnen Testfall, der ausgeführt wird, eine Entscheidungsüberdeckung von 50% erreicht."

Welcher der folgenden Aussagen ist zutreffend? [K2]

a)	Die Aussage ist wahr. Jeder einzelne Testfall bietet eine 100% Anweisungsüberdeckung und daher 50% Entscheidungsüberdeckung.	<input type="checkbox"/>
b)	Die Aussage ist wahr. Bei jedem einzelnen Testfall würde der Entscheidungsausgang der IF-Anweisung entweder wahr oder falsch sein.	<input type="checkbox"/>
c)	Die Aussage ist falsch. Ein einzelner Testfall kann in diesem Fall nur eine Entscheidungsüberdeckung von 25% garantieren.	<input type="checkbox"/>
d)	Die Aussage ist falsch. Die Aussage ist zu weit gefasst. Sie kann abhängig von der getesteten Software richtig sein oder nicht.	<input type="checkbox"/>

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board; 16.02.2019



Frage 1 - Lösung

21. Die folgende Aussage bezieht sich auf Entscheidungsüberdeckung:

"Wenn der Code nur eine einzige IF-Anweisung und keine Schleifen oder CASE-Anweisungen enthält und auch sonst durch den Test nicht geschachtelt aufgerufen wird, dann wird bei jedem einzelnen Testfall, der ausgeführt wird, eine Entscheidungsüberdeckung von 50% erreicht."

Welcher der folgenden Aussagen ist zutreffend? [K2]

a)	Die Aussage ist wahr. Jeder einzelne Testfall bietet eine 100% Anweisungsüberdeckung und daher 50% Entscheidungsüberdeckung.	<input type="checkbox"/>
b)	Die Aussage ist wahr. Bei jedem einzelnen Testfall würde der Entscheidungsausgang der IF-Anweisung entweder wahr oder falsch sein.	<input checked="" type="checkbox"/>
c)	Die Aussage ist falsch. Ein einzelner Testfall kann in diesem Fall nur eine Entscheidungsüberdeckung von 25% garantieren.	<input type="checkbox"/>
d)	Die Aussage ist falsch. Die Aussage ist zu weit gefasst. Sie kann abhängig von der getesteten Software richtig sein oder nicht.	<input type="checkbox"/>

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board; 16.02.2019



Frage 2

22. Welcher der folgenden Punkte ist eine Beschreibung für Anweisungsüberdeckung? [K2]

a)	Es handelt sich um eine Metrik zur Berechnung und Messung des prozentualen Anteils der ausgeführten Testfälle.	<input type="checkbox"/>
b)	Es handelt sich um eine Metrik zur Berechnung und Messung des prozentualen Anteils von ausgeführten Anweisungen eines Quelltextes.	<input type="checkbox"/>
c)	Es handelt sich um eine Metrik zur Berechnung und Messung der Anzahl von Anweisungen eines Quellcodes, die durch Testfälle ausgeführt wurden, die keine Fehlerwirkung aufgedeckt haben.	<input type="checkbox"/>
d)	Es handelt sich um eine Metrik, die eine wahr/falsch-Bestätigung gibt, ob alle Anweisungen abgedeckt sind oder nicht.	<input type="checkbox"/>

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board; 16.02.2019



Frage 2 - Lösung

22. Welcher der folgenden Punkte ist eine Beschreibung für Anweisungsüberdeckung? [K2]

a)	Es handelt sich um eine Metrik zur Berechnung und Messung des prozentualen Anteils der ausgeführten Testfälle.	<input type="checkbox"/>
b)	Es handelt sich um eine Metrik zur Berechnung und Messung des prozentualen Anteils von ausgeführten Anweisungen eines Quelltextes.	<input checked="" type="checkbox"/>
c)	Es handelt sich um eine Metrik zur Berechnung und Messung der Anzahl von Anweisungen eines Quellcodes, die durch Testfälle ausgeführt wurden, die keine Fehlerwirkung aufgedeckt haben.	<input type="checkbox"/>
d)	Es handelt sich um eine Metrik, die eine wahr/falsch-Bestätigung gibt, ob alle Anweisungen abgedeckt sind oder nicht.	<input type="checkbox"/>

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board; 16.02.2019



Frage 3

23. Welche Aussage über den Zusammenhang zwischen der Anweisungsüberdeckung und der Entscheidungsüberdeckung ist wahr? [K2]

a)	100% Entscheidungsüberdeckung schließt 100% Anweisungsüberdeckung ein.	<input type="checkbox"/>
b)	100% Anweisungsüberdeckung schließt 100% Entscheidungsüberdeckung ein.	<input type="checkbox"/>
c)	50% Entscheidungsüberdeckung schließt 50% Anweisungsüberdeckung ein.	<input type="checkbox"/>
d)	Entscheidungsüberdeckung kann nie 100% erreichen.	<input type="checkbox"/>



Frage 3 - Lösung

23. Welche Aussage über den Zusammenhang zwischen der Anweisungsüberdeckung und der Entscheidungsüberdeckung ist wahr? [K2]

a)	100% Entscheidungsüberdeckung schließt 100% Anweisungsüberdeckung ein.	<input checked="" type="checkbox"/>
b)	100% Anweisungsüberdeckung schließt 100% Entscheidungsüberdeckung ein.	<input type="checkbox"/>
c)	50% Entscheidungsüberdeckung schließt 50% Anweisungsüberdeckung ein.	<input type="checkbox"/>
d)	Entscheidungsüberdeckung kann nie 100% erreichen.	<input type="checkbox"/>



Frage 4

24. Für welche der folgenden Situationen ist der Einsatz von explorativem Testen geeignet? [K2]

a)	Wenn unter Zeitdruck die Durchführung bereits spezifizierter Tests beschleunigt werden muss.	<input type="checkbox"/>
b)	Wenn das System inkrementell entwickelt und keine Test-Charta vorhanden ist.	<input type="checkbox"/>
c)	Wenn Tester zur Verfügung stehen, die über ausreichende Kenntnisse von ähnlichen Anwendungen und Technologien verfügen.	<input type="checkbox"/>
d)	Wenn bereits ein vertieftes Wissen über das System vorhanden ist und der Nachweis erbracht werden soll, das besonders intensiv getestet werden soll.	<input type="checkbox"/>

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board; 16.02.2019



Frage 4 - Lösung

24. Für welche der folgenden Situationen ist der Einsatz von explorativem Testen geeignet? [K2]

a)	Wenn unter Zeitdruck die Durchführung bereits spezifizierter Tests beschleunigt werden muss.	<input type="checkbox"/>
b)	Wenn das System inkrementell entwickelt und keine Test-Charta vorhanden ist.	<input type="checkbox"/>
c)	Wenn Tester zur Verfügung stehen, die über ausreichende Kenntnisse von ähnlichen Anwendungen und Technologien verfügen.	<input checked="" type="checkbox"/>
d)	Wenn bereits ein vertieftes Wissen über das System vorhanden ist und der Nachweis erbracht werden soll, das besonders intensiv getestet werden soll.	<input type="checkbox"/>

Quelle: ISTQB Foundation Level Sample Paper; SET A; 2018; Deutschsprachige Fassung/ Lokalisierung; German Testing Board; 16.02.2019