Leibniz Universität Hannover

SOFTWARE ENGINEERING

Übung: Software-Qualität

Sommersemester 2016

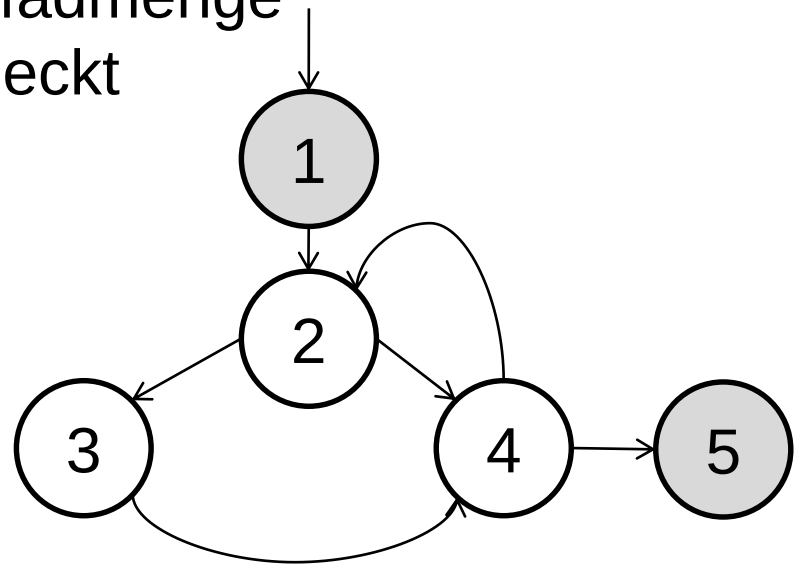
swq@se.uni-hannover.de

Edge-Pair-Coverage

- Variante von Pfadüberdeckung, bei der alle Kantenpaare überdeckt werden sollen

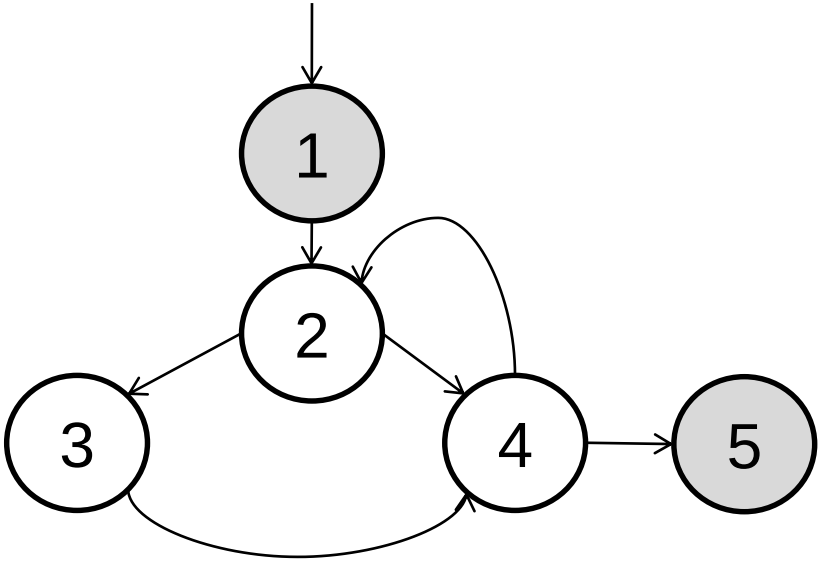
Aufgabe:

1) Stellen Sie eine minimale Testpfadmenge auf, die alle Kanten überdeckt
2) Stellen Sie eine minimale Testpfadmenge auf, die alle Kantenpaare überdeckt



Edge-Pair-Coverage

Aufgabe:
1) Stellen Sie eine minimale Testpfadmenge auf, die alle Kanten überdeckt
2) Stellen Sie eine minimale Testpfadmenge auf, die alle Kantenpaare überdeckt



Kanten:
(1,2)
(2,3)
(2,4)
(3,4)
(4,2)
(4,5)

Testpfade für Kantenüberdeckung:
(1,2,3,4,2,4,5)

Edge-Pair-Coverage

Kantenpaare:

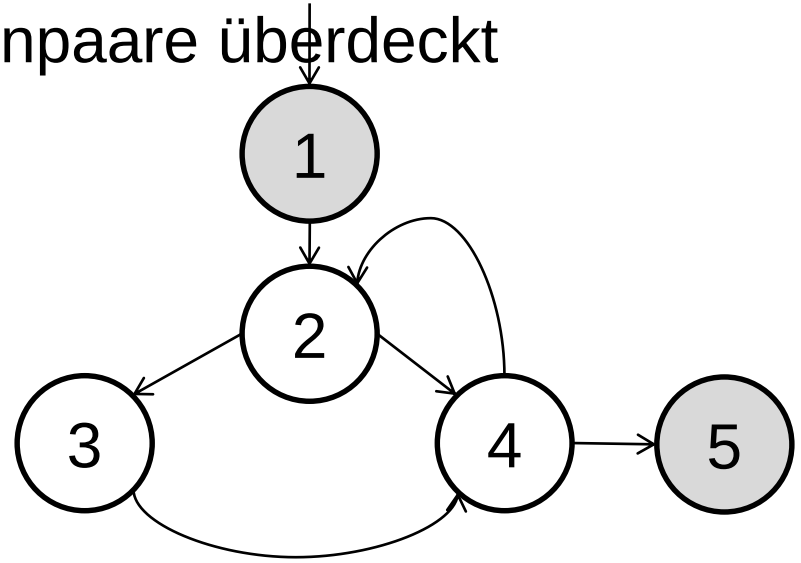
(1,2,3)
(1,2,4)
(2,3,4)
(2,4,2)
(2,4,5)
(3,4,2)
(3,4,5)
(4,2,4)
(4,2,3)

Edge-Pair-Coverage:

(1,2,3,4,2,4,5)
(1,2,4,2,3,4,5)

Aufgabe:

1) Stellen Sie eine minimale Testpfadmenge auf, die alle Kanten überdeckt
2) Stellen Sie eine minimale Testpfadmenge auf, die alle Kantenpaare überdeckt



Bedingungsüberdeckung

- **Einfache Bedingungsüberdeckung**: Alle atomaren Prädikate nehmen mindestens einmal beide Wahrheitswerte an
- **Minimale Bedingungsüberdeckung:** Alle atomaren und zusammengesetzten Prädikate nehmen mindestens einmal beide Wahrheitswerte an
- **Mehrfache Bedingungsüberdeckung:** Alle Kombinationen von Auswertungen der atomaren Prädikate getestet

Beispiel zur Überdeckung
100% Überdeckung nach verschiedenen Maßen

Aufgabe:
100% Überdeckung mit möglichst wenigen Testfällen

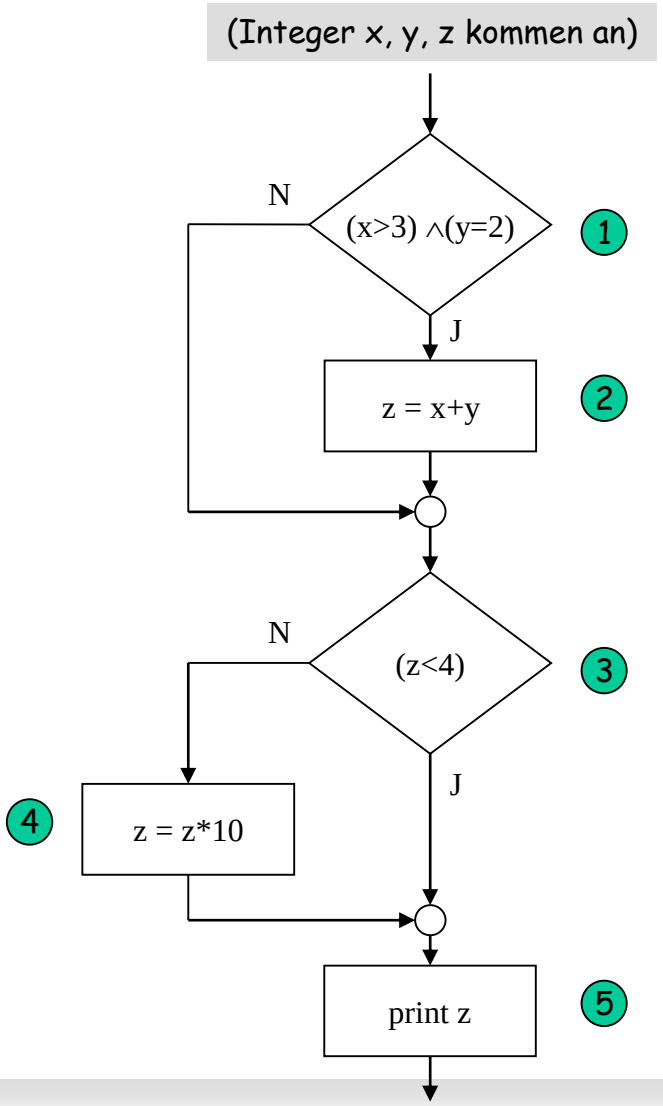
Anweisungsüberdeckung (statement coverage):
Eingabe (4,2,0) führt zu Pfad [1,2,3,4,5]

Zweigüberdeckung (branch coverage):
Eingabe (4,1,0) stimuliert oben Nein-Zweig, unten Ja-Zweig
Eingabe (4,2,0) stimuliert oben Ja-Zweig, unten Nein-Zweig

Mehrfache Bedingungsüberdeckung (condition coverage):
(x>3) wahr, (y=2) wahr durch (4,2,0)
(x>3) wahr, (y=2) falsch durch (4,1,0)
(x>3) falsch, (y=2) wahr durch (1,2,0)
(x>3) falsch, (y=2) falsch durch (1,1,0)
gleichzeitig wird (z<4) mit überdeckt durch (4,1,0), (4,2,0)

Pfadüberdeckung (path coverage):
erfordert Testfälle für folgende Pfade
[1,2,3,5], [1,3,5], [1,2,3,4,5], [1,3,4,5]

einen Testfall für [1,2,3,5] gibt es aber nicht!



Bedingungsüberdeckung

```
public static int getTriangleKind(int sideA, int
sideB, int sideC){
                    1
        if(sideA == sideB){
                        2
                if(sideB == sideC){
                        return 3;
                } else {
                        return 2;
                }
                        3                       3b
                        a
        } else if(sideA == sideC || sideB ==
sideC){
                return 2;

        } else {
                return 1;
        }
}
```

Besondere Kriterien führen dazu, dass nicht alle Kombinationen möglich sind:

1. 2 = 3b
2. 1 = 1 => erreichen 3a & 3b nie
3. 1 = 0 => erreichen 2 nie
4. 1 = 0 => (3a & 3b) darf nicht 1 sein

Bedingungsüberdeckung

Einfache Bedingungsüberdeckung

	1	2	3a	3b
T1	1	1		
T2	1	0		
T3	0		1	0
T4	0		0	1

Haben hier zwar einfache Bedingungsüberdeckung – aber keine Anweisungsüberdeckung!

```
public static int getTriangleKind(int sideA, int
sideB, int sideC){
                                    1
     if(sideA == sideB){
                                         2
          if(sideB == sideC){
               return 3;
          } else {
               return 2;
          }
                                   3a                  3b
     } else if(sideA == sideC || sideB == sideC){
          return 2;

     } else {
          return 1;
     }
}
```

Bedingungsüberdeckung

Minimale Bedingungsüberdeckung

	1	2	3a	3b	(3a || 3b)
T1	1	1			
T2	1	0			
T3	0		1	0	1
T4	0		0	1	1
T5	0		0	0	0

Checken hier, dass sowohl die einzelnen Teil-Prädikate als auch das zusammengesetzte Prädikat je beide Werte annehmen.

```
public static int getTriangleKind(int sideA, int
sideB, int sideC){

     if(sideA == sideB){ // 1

          if(sideB == sideC){ // 2
               return 3;
          } else {
               return 2;
          }

     } else if(sideA == sideC || sideB == sideC){ // 3a || 3b
          return 2;

     } else {
          return 1;
     }
}
```