# Software Engineering

## Test 2 / 2

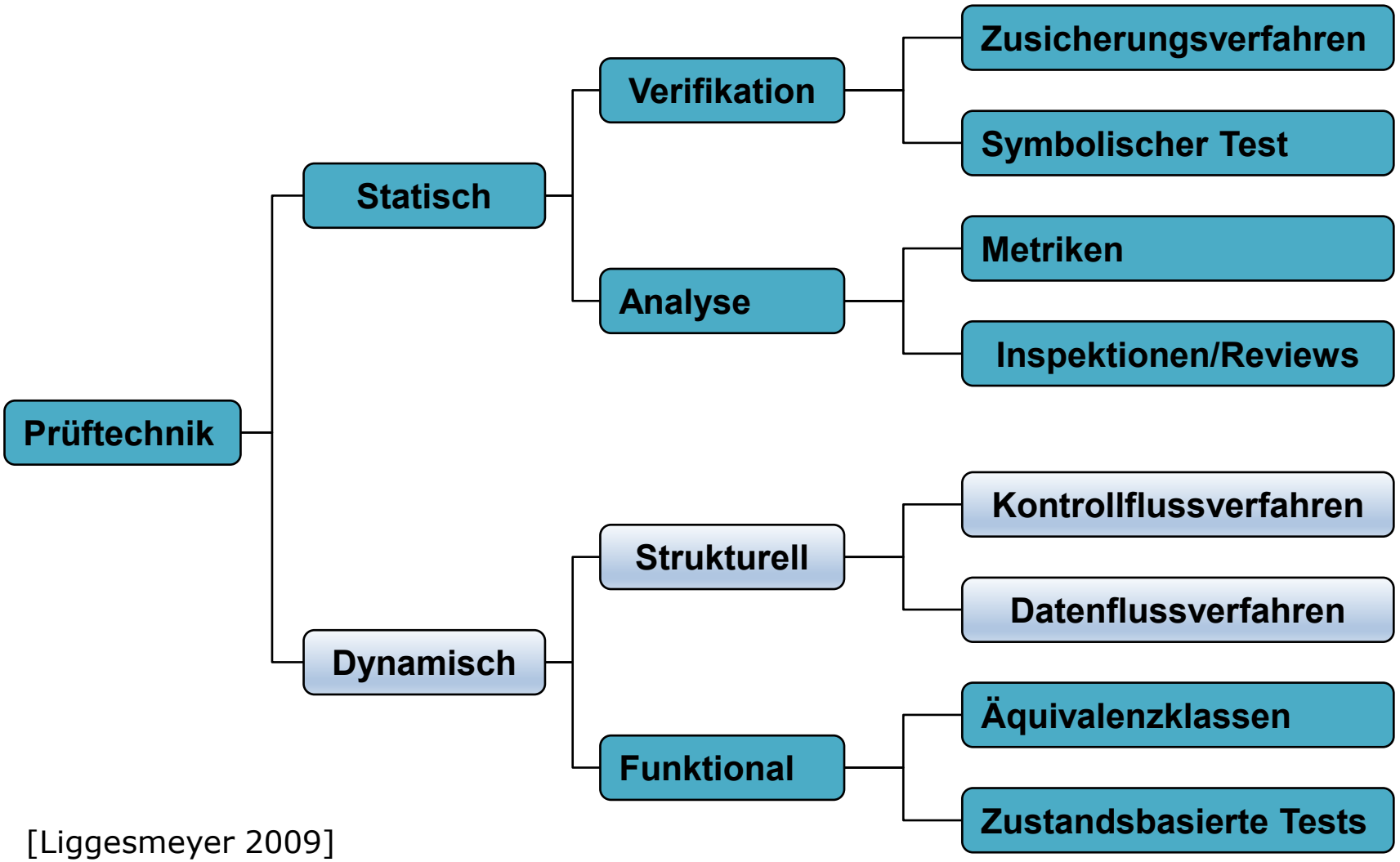
Prof. Dr. Bodo Kraft

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

# Agenda

- **Begriffsklärung / Motivation**
- **Testverfahren**
  - Black-Box
  - Whitebox
- **Test Driven Development (TDD)**

# Übersicht Verfahren der analytischen QS




[Liggesmeyer 2009]

# Strukturtests/Whitebox
## Dynamisch, strukturelle Prüfverfahren

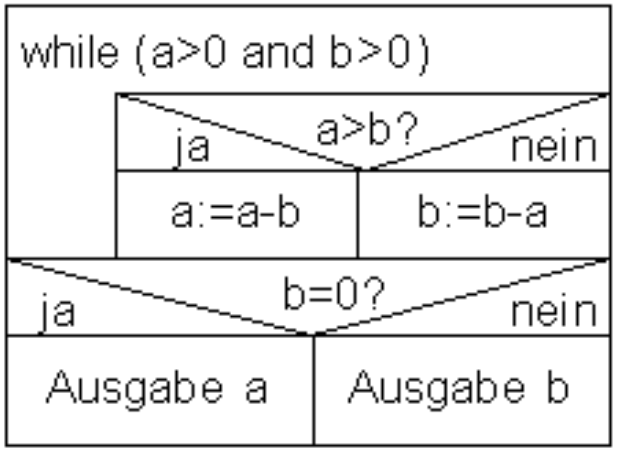


**Charakteristisch:**

- Quellcode des Programms ist einsehbar
- Testen anhand der bekannten Programmstruktur
- Kontrollfluss-orientiert

**Zur Wdh.: Nassi-Shneiderman**

- Methode zur strukturierten Programmierung in Entwurfsphase
- Top-Down-Prinzip
- Diagrammtyp: Struktogramm
  - Zeigt Programmablaufplan
  - kein UML-Diagrammtyp! (aber ISO 66261)
- In der Praxis selten im Einsatz
  - Stattdessen: UML-Flussdiagramme
  - setzen auf Kontrollflussgraph



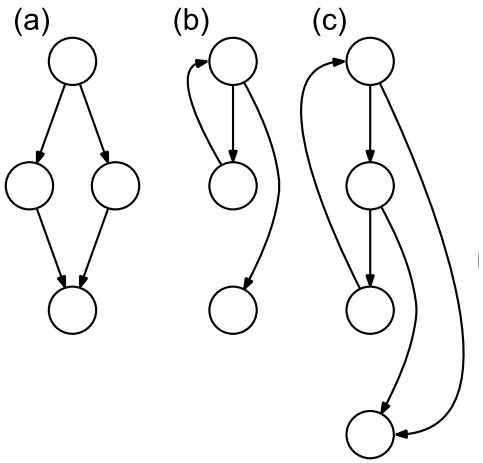

Wer kennt den Algorithmus?

# Kontrollflussgraph
## Dynamisch, strukturelle Prüfverfahren



**Eigenschaften:**

- Gerichteter Graph
- Beschreibt Kontrollfluss eines Computerprogramms
- Visualisiert mögliche Programmflüsse
- Einsatz zur Programmoptimierung oder Qualitätssicherung
  - ➢ Erreichbarkeit von Knoten/Kanten
- Hauptunterschied zu Ablaufdiagrammen der UML: sehr codenah




**Beispiele:**

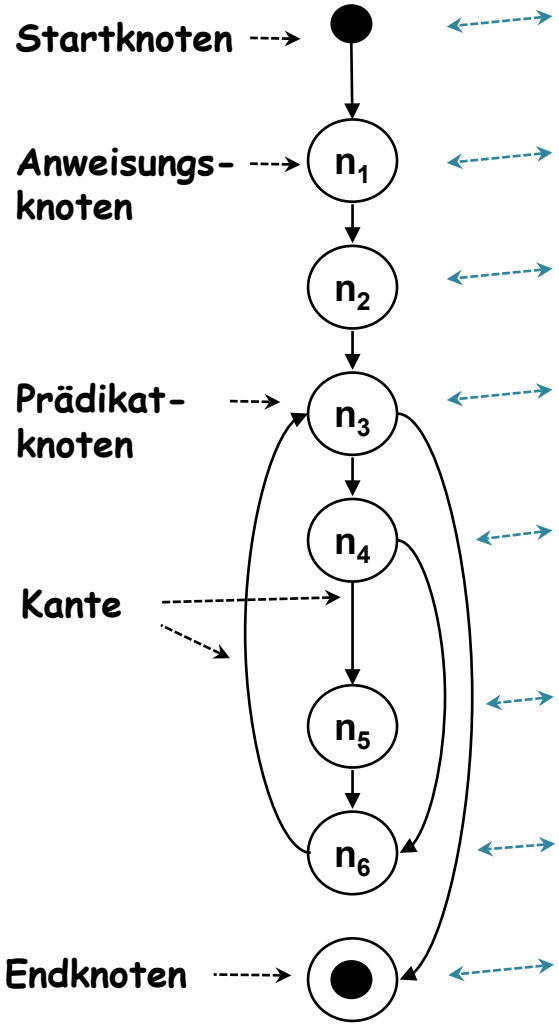
*a)* *if-then-else* - Bedingung

*b)* *while* - Schleife

*c)* *zwei Abbruchbedingungen* in Schleife

z.B.: "while"-Schleife, die ein "if...break" im Rumpf enthält

# Kontrollflussgraph
## Dynamisch, strukturelle Prüfverfahren




```
public static int zahlVokale(InputStreamReader isr)
throws IOException
{

    int nummer = 0;


    int zeichen = isr.read();


    while (('A' <= zeichen) && (zeichen <= 'Z'))
    {


        if (Arrays.asList('A','E','I','O','U')
            .contains(zeichen))
        {

            nummer++;

        }

        zeichen = isr.read();
    }

    return nummer;

}
```

# Grundlagen Überdeckungsverfahren
## Dynamisch, strukturelle Prüfverfahren



Der Kontrollflussgraph wird betrachtet und auf verschiedene Überdeckungsarten analysiert.

Gängige Metriken zur Messung der Überdeckung:

1) Anweisungsüberdeckung (C0)
2) Zweigüberdeckung (C1)
3) Bedingungsüberdeckung (C2 oder C3)
    1) Einfache Bed.-Überdeckung (C3a)
    2) Mehrfache Bed.-Überdeckung (C3b)
    3) Minimal mehrfache Bed.-Überdeckung (C3c)
4) Pfadüberdeckung (C2 oder C4)
5) Und weitere …

**Zur Information:**

Die Abkürzungen C0-C4 werden in der Literatur teilweise (unterschiedlich) verwendet.

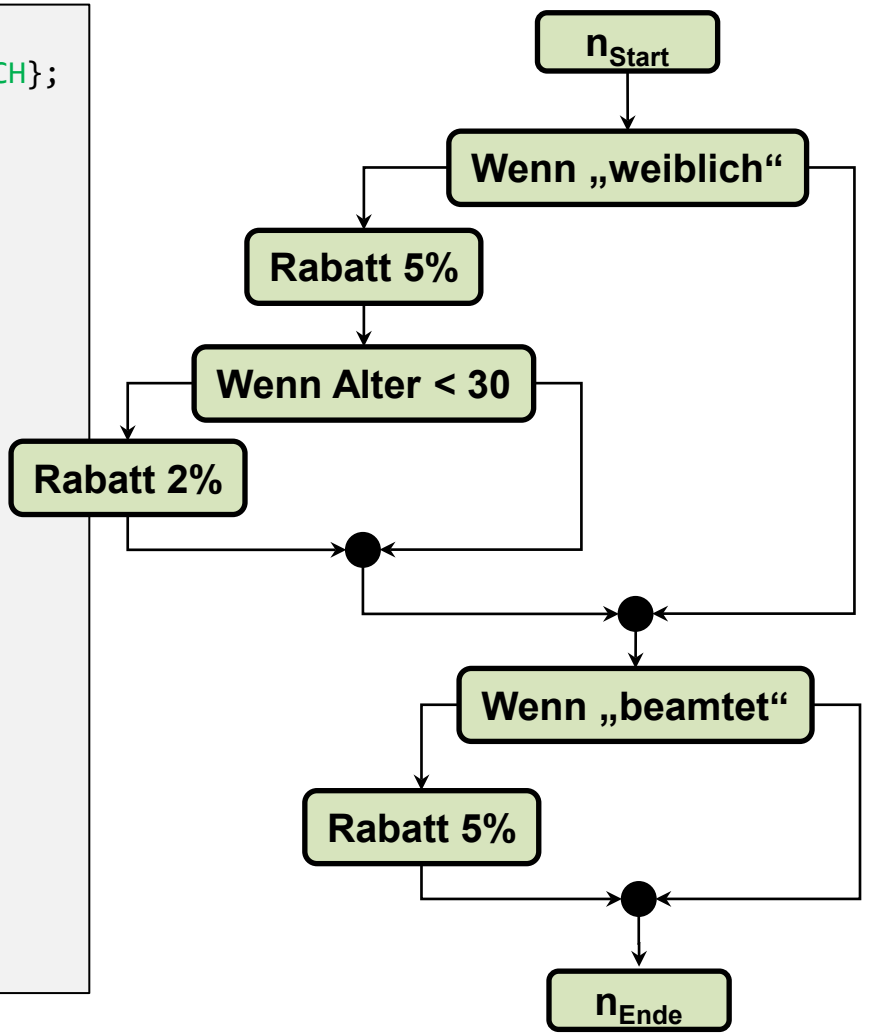
Wir verwenden hier ausschließlich die sprechenden Bezeichnungen.

# Anweisungsüberdeckung
## Dynamisch, strukturelle Prüfverfahren

```java
public class Person {
    public enum GESCHLECHT {MÄNNLICH, WEIBLICH};

    GESCHLECHT genus;
    int alter;
    boolean istBeamter;
    double rabatt = 0.0;

    /*
     * Berechnet den Rabatt auf den
     * Eintrittspreis eines Diskobesuchs
     */
    public double berechneRabatt(){
        if (genus==GESCHLECHT.WEIBLICH){
            rabatt += 0.05;
            if (alter < 30){
                rabatt += 0.02;
            }
        }
        if (istBeamter){
            rabatt += 0.05;
        }
        return rabatt;
    } // …
}
```



# Anweisungsüberdeckung
## Dynamisch, strukturelle Prüfverfahren

100% Anweisungsüberdeckung =

Jede Anweisung wird mindestens einmal ausgeführt (Anweisungsknoten)

- Anzahl der notwendigen Testfälle: 1
- Eingabedaten:

  genus=weiblich,

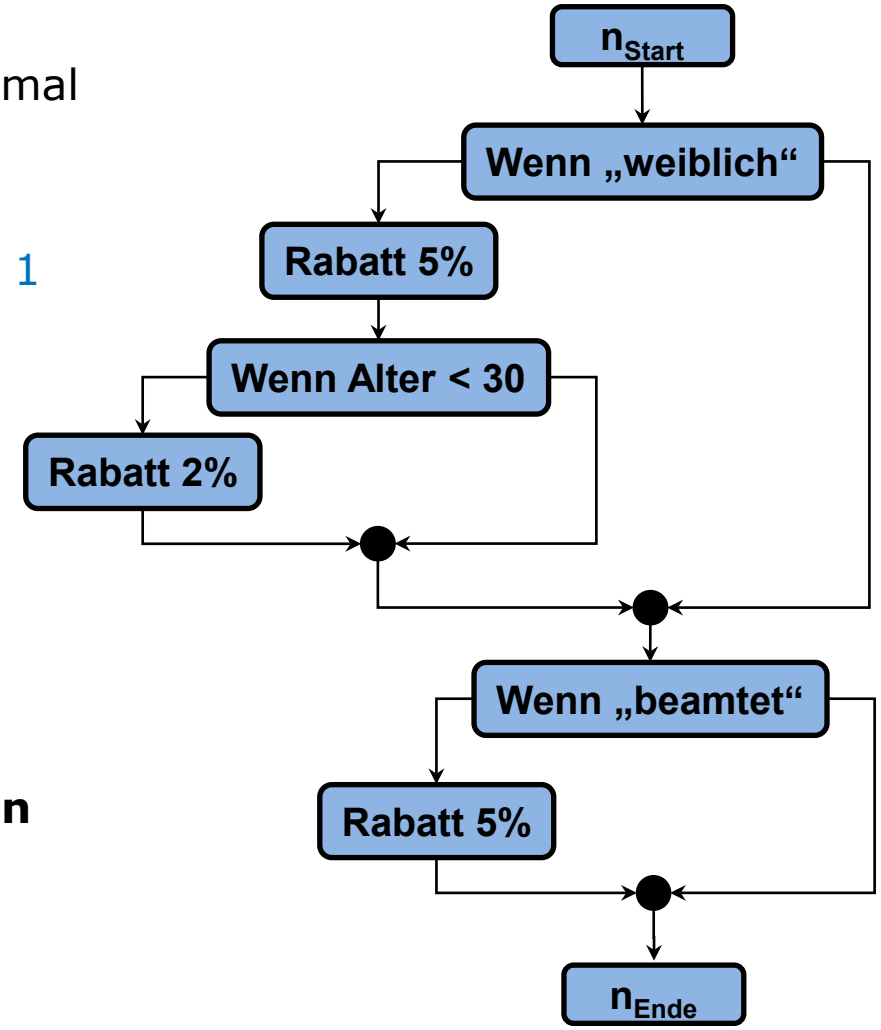
  alter=29,

  istBeamtet=true,
- Erwartetes Ergebnis:

  Rabatt 12%

Vorteil:

- **nicht erreichbare Anweisungen** können erkannt werden
- liefert Verhältnis getesteter Anweisungen zu Gesamtanzahl Anweisungen



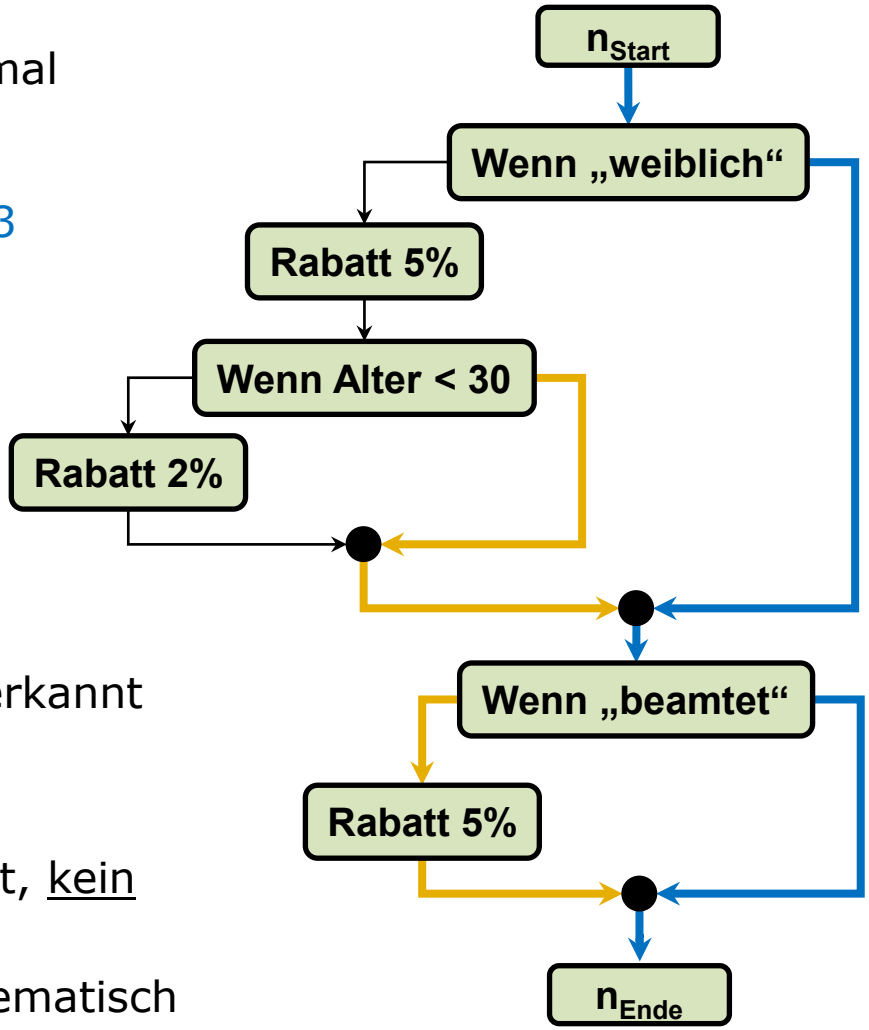
# Zweigüberdeckung
## Dynamisch, strukturelle Prüfverfahren

100% Zweig-Abdeckung =

Jeder **Zweig** wird mindestens einmal ausgeführt (alle Pfeile)

- Anzahl der notwendigen Testfälle: 3

| Genus | W | W | M |
|---|---|---|---|
| Alter | 29 | 30 | 30 |
| istBeamter | Ja | Ja | Nein |
| Erwarteter Rabatt | 12 % | 10% | 0% |




**Vorteil:**

- nicht erreichbare Zweige können erkannt werden

**Nachteil:**

- Jeder wird Zweig isoliert betrachtet, kein Kontext
- komplexe Bedingungen sind problematisch

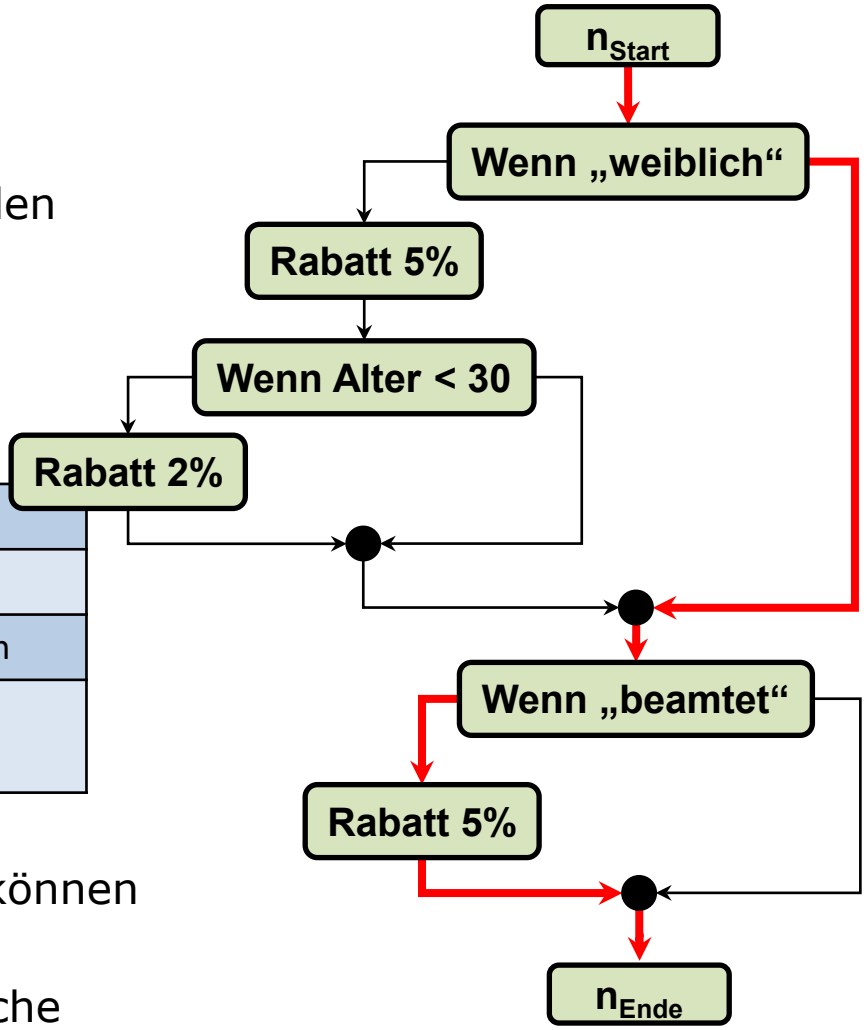
# Pfadüberdeckung
## Dynamisch, strukturelle Prüfverfahren

100% Pfad-Abdeckung =

- Jeder **Pfad** wird mindestens einmal ausgeführt
- Also **alle möglichen Wege** durch den Graphen

- Anzahl der notwendigen Testfälle: 6
- Testdaten:

| Genus | W | W | M | W | **M** | W |
|---|---|---|---|---|---|---|
| Alter | 29 | 30 | 30 | 30 | **31** | 29 |
| istBeamter | Ja | Ja | Nein | Nein | **Ja** | Nein |
| Erwarteter Rabatt | 12 % | 10 % | 0% | 5% | **5%** | 7% |




Vorteil:

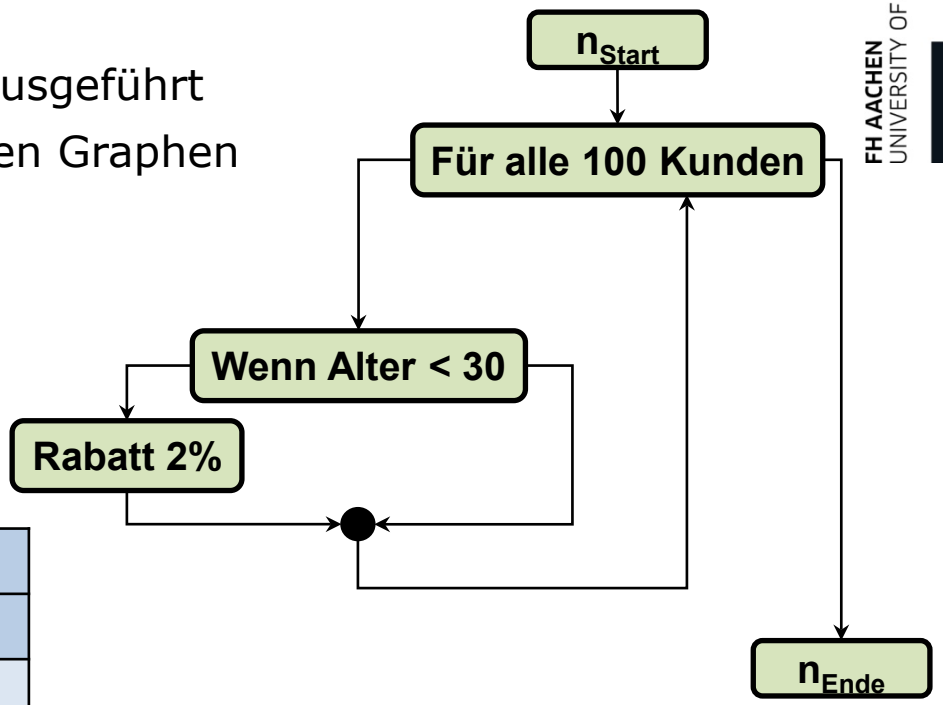
- **nicht erreichbare Kombination** können erkannt werden
- Sehr sehr aufwendig, kombinatorische Explosion

# Problem Pfadüberdeckung: kombinatorische Explosion

## Dynamisch, strukturelle Prüfverfahren

100% Pfad-Abdeckung =

- Jeder Pfad wird mindestens einmal ausgeführt
- Also **alle möglichen Wege** durch den Graphen




- Testdaten für **einen** Pfad:

| Kunde | 1 | 2 | 3 | 4 | … | 100 |
|---|---|---|---|---|---|---|
| Alter | 29 | 30 | 29 | 30 | | 30 |
| Erwarteter Rabatt | 2 | 0 | 2 | 0 | | 0 |

- Anzahl Testdaten für **alle Pfade: 2 hoch 100**

→ Schon bei einfachem Code nicht mehr berechenbar

# Boundary-Interior-Pfadüberdeckung
## Dynamisch, strukturelle Prüfverfahren

**Idee:** Man schränkt den vollständigen Pfadüberdeckungstest ein, auf ***maximal zwei Durchläufe pro Schleife***

**Vorgehen:**

Betrachte alle Pfade bei

- Keiner Schleifenausführung („Abweisung“)
- Einmaligen Durchlauf („Grenzpfad“, „boundary path“)
- Zweimaligen Durchlauf („innerer Pfad“, „interior path“)

# Boundary-Interior-Pfadüberdeckung
## Dynamisch, strukturelle Prüfverfahren



**Vorteile**

- Besser für Schleifen geeignet als Kantenüberdeckung
- In vielen Fällen praktikabel

**Probleme**

- 100 % Überdeckung nicht erreichbar in folgenden Fällen
  - Toter Code
  - Nicht erreichbare Kombinationen von Subpfaden
- Schlechter geeignet bei fixer Zahl von Schleifendurchläufen (for-Schleifen) bspw. bei
  - Vektorprodukt
  - Matrixmultiplikation

# Bedingungsüberdeckung (einfach)
## Dynamisch, strukturelle Prüfverfahren



**Einfache Bedingungsüberdeckung:**

Die Testfälle sind so zu bestimmen, dass die Auswertung jeder atomaren Teilentscheidung/Bedingung (mindestens) einmal wahr und einmal falsch ergibt. Gemessen wird unabhängig voneinander.

```
int a, x, y;

if (a > 0){
    //… Bsp1
}

if (x>0 & y<0){
    //... Bsp2
}

if (x>0 && y<0){
    //… Bsp3
}
```

| Testfall | a>0 | x>0 | y<0 |
|---|---|---|---|
| I | -4(f) | -4(f) | -4(t) |
| II | +4(t) | +4(t) | +4(f) |

Atomare Teilentscheidungen sind:

- a>0, x>0, y<0

Für die zwei Testfälle gilt:

- Bsp1 (2/2, 100% Coverage)
- Bsp2 (4/4, 100% Coverage)
- Bsp3 (3/4, 75% Coverage, y=-4 wird nicht ausgeführt)

**Allgemein Vorsicht bei Kurzschluss-Operatoren:**
(x && y) ist x bereits FALSE wird y nicht ausgeführt
(x & y) führt immer beide Teile der Bedingung aus

**Bemerke:** Der Rumpf von Bsp2 wird nicht erreicht, trotz 100% Coverage → Kriterium nicht ausreichend

# Bedingungsüberdeckung (mehrfach)
## Dynamisch, strukturelle Prüfverfahren



**Mehrfache Bedingungsüberdeckung:**

Jede Kombination von Wahrheitswerten der elementaren Bedingungen muss zusätzlich zu Einfacher Bedingungsüberdeckung abgedeckt werden.

```
int x, y, z;

if ((A && B) || C){
    //... Bsp
}
```

| Testfall | A | B | C | A && B | (A && B) \|\| C |
|---|---|---|---|---|---|
| I | **T** ✓ | **T** ✓ | **T** ✓ | **T** ✓ | **T** ✓ |
| II | T | T | **F** ✓ | T | F ✓ |
| III | T | **F** ✓ | T (skip) | **F** ✓ | F (skip) |
| IV | T | F | F (skip) | F | F (skip) |
| V | **F** ✓ | T (skip) | T (skip) | F (skip) | F (skip) |
| VI | F | T (skip) | F (skip) | F (skip) | F (skip) |
| VII | F | F (skip) | T (skip) | F (skip) | F (skip) |
| VIII | F | F (skip) | F (skip) | F (skip) | F (skip) |

Für n elementare Ausdrücke gilt:

- Minimal n+1 Testfälle, Maximal 2^n Testfälle

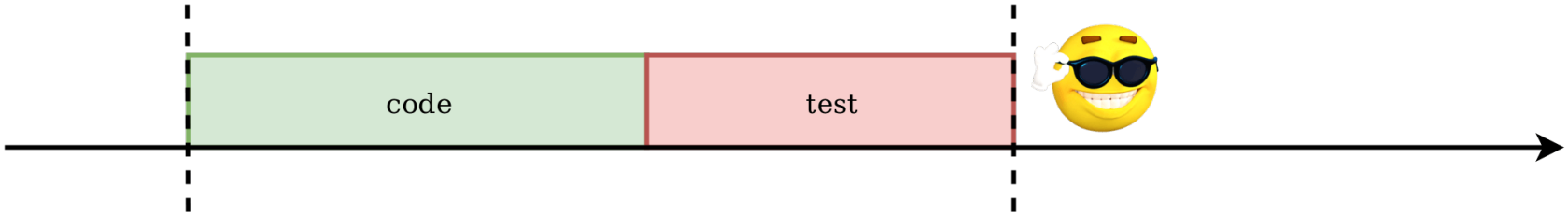
# Agenda

- **Begriffsklärung / Motivation**
- **Testverfahren**
  - Black-Box
  - Whitebox
- **Test Driven Development (TDD)**

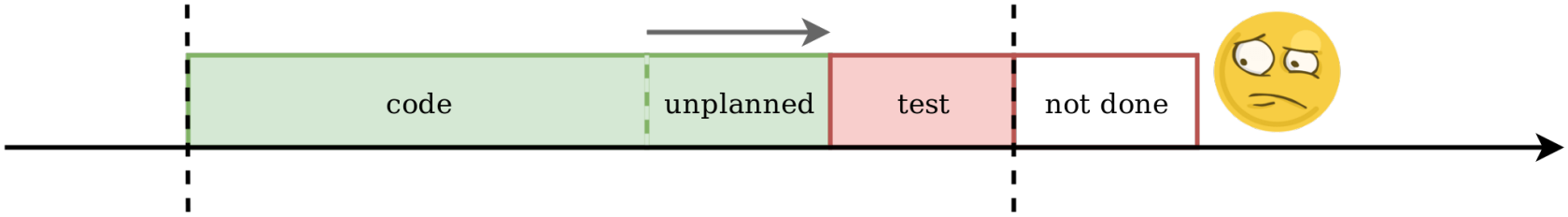
# Motivation für Testgetriebene Entwicklung
## TDD und Modultest



- In klassischen Vorgehensmodellen werden Tests meist **nach** der Entwicklung der Software geschrieben.




- Was passiert bei längerer Entwicklungsdauer, unerwarteten neuen Features, …?
- Entweder wird die Deadline nach hinten verschoben oder die Tests gestrichen.




➔ Umfang der **Auslieferung ist fix** und die **Qualität variabel**

[https://blog.codecentric.de/en/2019/06/test-driven-development-theory-practice/]

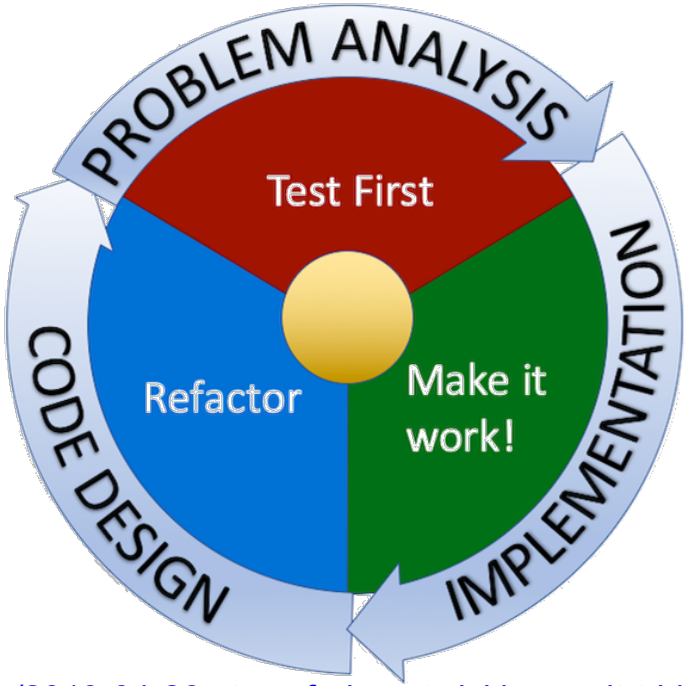
# Idee und Konzept
## TDD und Modultest

### Was ist TDD?

- Test-driven development / Testgetriebene Entwicklung
- Tests werden vor den zu testenden Komponenten erstellt
- Programmierung erfolgt in Mikroiterationen (nur wenige Minuten)

### Schritte

1. Problem verstehen und Tests formulieren (Kernfunktionalität und Grenzfälle)
2. Einfachsten Implementierungs-ansatz wählen, damit die Tests durchlaufen
3. Code mit Entwurfsmustern, etc. refactoren, um Wartbarkeit und Erweiterbarkeit sicherzustellen




[https://www.agiledojo.de/2019-04-26-stressfreie-entwicklung-mit-tdd/]

# Idee und Konzept umgesetzt in Microiterationen
TDD und Modultest

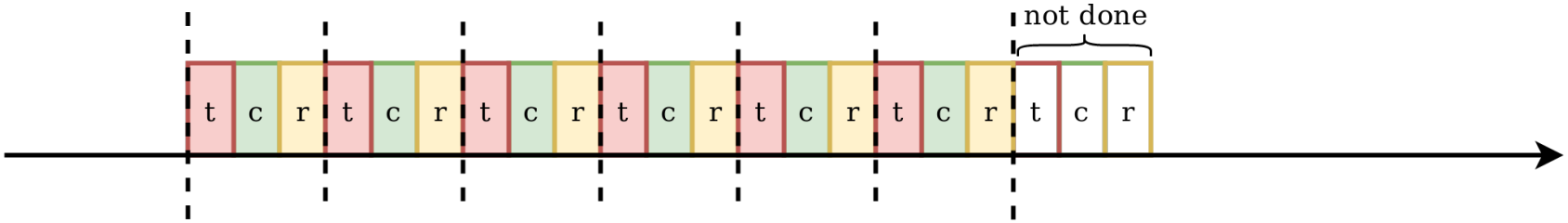


**Neues Konzept**

➔ Umfang der **Auslieferung ist variabel** und die **Qualität fix**

Bei jeder Deadline gilt:

- Fertige, vollständig getestete Features werden ausgeliefert
- Ein nicht fertiges oder nicht getestetes Feature wird nicht ausgeliefert
- Das ist akzeptabel, weil
  - das Feature in der nächsten Zeit nachgeliefert wird.
  - es besser ist, ein sicher korrekt implementiertes Feature auszuliefern, als alle Features ohne entsprechend sichergestellte Qualität.




[https://blog.codecentric.de/en/2019/06/test-driven-development-theory-practice/]

# TDD im Kontext der agilen Softwareentwicklung
## TDD und Modultest



**XP (Kent Beck)**

- Test-first-Ansatz ist eine der 12 Kernpraktiken von eXtreme Programming

**Scrum (Ken Schwaber)**

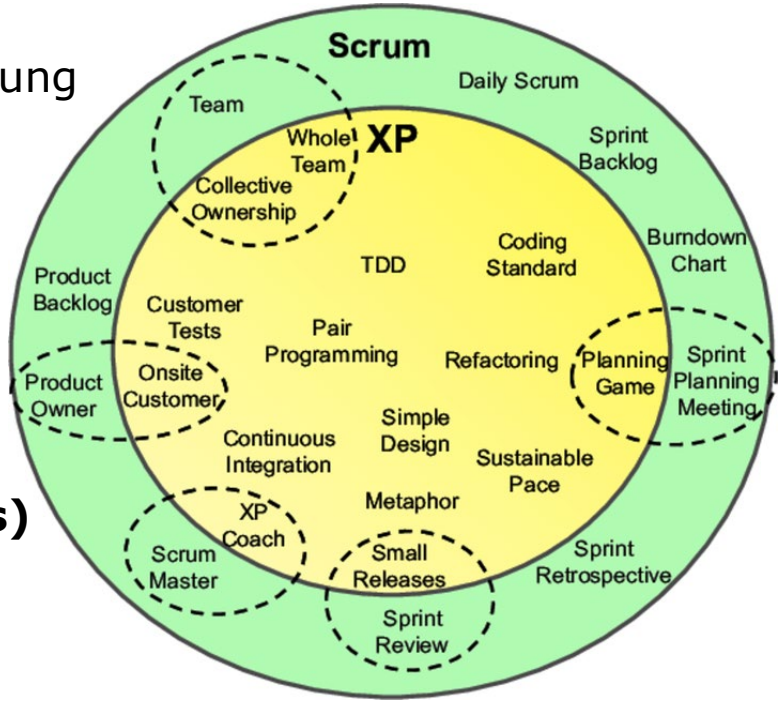
- Grundlage für inkrementelle Softwareentwicklung

**Refactoring (Martin Fowler)**

- Grundlage für Umbau von Software sind automatisierte Test zur Absicherung der Qualität

**Konfigurationsmanagement (Linus Torvalds)**

- Grundlage für kollaborative Software-Entwicklung in Branches




Quelle: https://medium.com/agile-outside-the-box/better-together-xp-and-scrum-c69bf9bffcff

**CI/CD (Jez Humble, Paul M. Duvall)**

- Testabdeckung ist Garant für lauffähigen Code

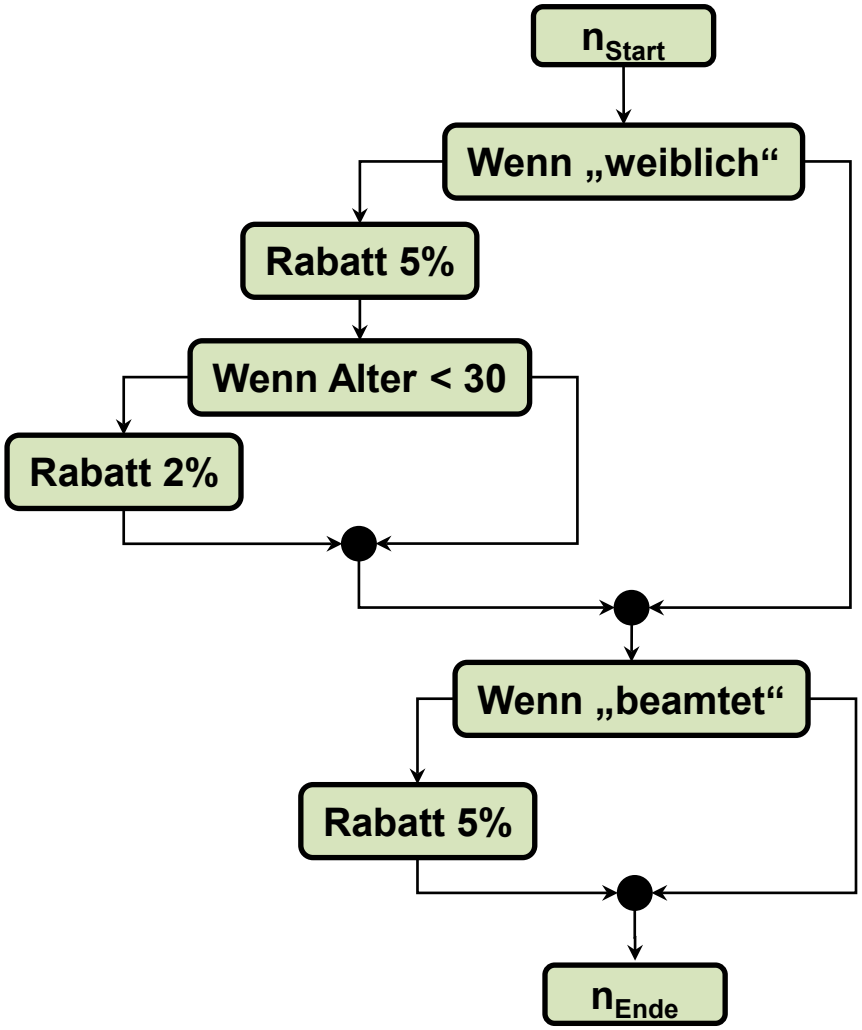
# Anwendungsbeispiel – fachliche Einführung
TDD und Modultest

Für die Betragsberechnung soll der **Rabattsatz** berechnet werden.

Aus der Anforderungsanalyse erhalten wir die folgende Spezifikation:

**Was sind mögliche Schritte ?**

1. Unterscheidung Genus
2. Unterscheidung Alter
3. Unterscheidung Anstellung



# Anwendungsbeispiel – Rahmenbedingungen
## TDD und Modultest



**Start mit dem Rahmen der Klasse**

```
/**
Klasse zur Speicherung der personenbezogenen Eigenschaften mit Berechnung
der persönlichen Rabattsätze
*/

public class Person {

  public Person() {
  }


}
```

# Anwendungsbeispiel – Erste Microiteration
## TDD und Modultest



```
/**
Klasse zur Speicherung der personenbezogenen Eigenschaften mit Berechnung
der persönlichen Rabattsätze
*/

public class Person {

  public Person() {
  }

  public double berechneRabatt() {
    double rabatt = 0.0;
    return rabatt;
  }
}
```

**Minimale Implementierung**

```
@Test
void berechneRabatt_withEinfachePerson_shouldReturn0() {
    Person person = new Person();

    double rabatt = person.berechneRabatt();
    assertEquals(0.0, rabatt);
}
```

**Erster Testfall**

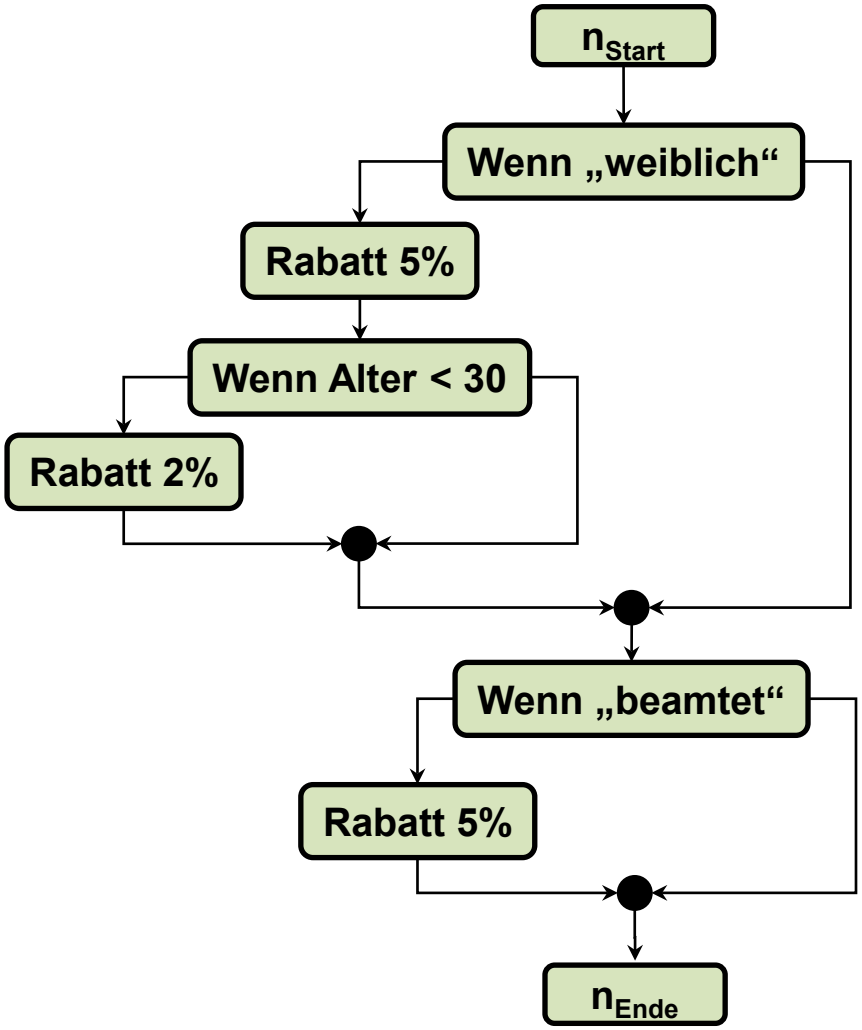
# Anwendungsbeispiel – Zweite Microiteration
## TDD und Modultest

Für die Betragsberechnung soll der **Rabattsatz** berechnet werden.

Aus der Anforderungsanalyse erhalten wir die folgende Spezifikation:

**Was sind mögliche Schritte ?**

1. Unterscheidung Genus
2. Unterscheidung Alter
3. Unterscheidung Anstellung



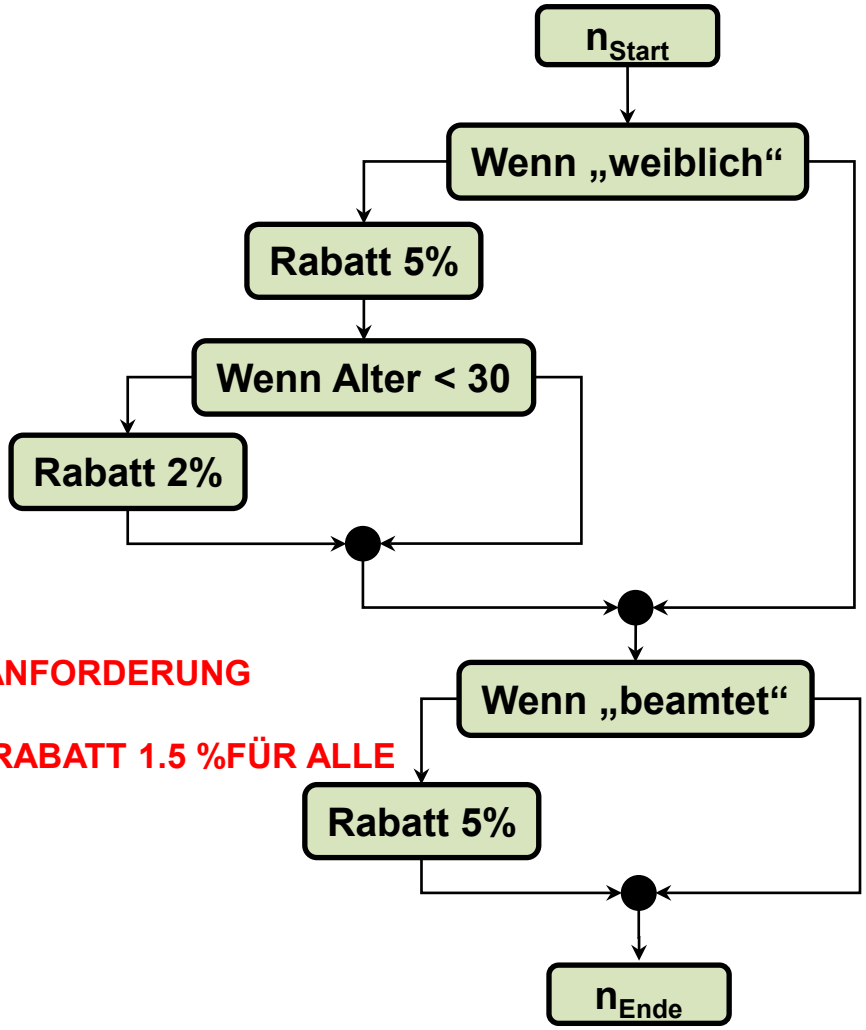
# Anwendungsbeispiel – Zweite Microiteration
## TDD und Modultest

Für die Betragsberechnung soll der **Rabattsatz** berechnet werden.

Aus der Anforderungsanalyse erhalten wir die folgende Spezifikation:

**Was sind mögliche Schritte ?**

1. Unterscheidung Genus
2. Unterscheidung Alter
3. Unterscheidung Anstellung




Neue ANFORDERUNG

GENERELLER RABATT 1.5 %FÜR ALLE