

Software-Test - Lernmaterial Modul F (Lernfeld 5)

1. Grundlagen des Software-Tests

Was ist Software-Testing?

Software-Testing ist der Prozess der Überprüfung und Validierung, ob eine Softwareanwendung fehlerfrei ist, die definierten Anforderungen erfüllt und die erwarteten Ergebnisse liefert.

Warum testen wir Software?

- Fehler frühzeitig erkennen und beheben
- Qualität sicherstellen
- Kosten reduzieren (je später ein Fehler gefunden wird, desto teurer wird die Behebung)
- Kundenvertrauen aufbauen
- Sicherheit gewährleisten

Die 7 Testprinzipien

- 1. Testen zeigt Fehler, nicht deren Fehlen** Tests können nur zeigen, dass Fehler vorhanden sind, aber nicht, dass keine Fehler existieren.
 - 2. Vollständiges Testen ist unmöglich** Es gibt zu viele mögliche Eingabekombinationen, um alle zu testen.
 - 3. Frühes Testen spart Zeit und Geld** Je früher Fehler gefunden werden, desto günstiger ist die Behebung.
 - 4. Häufung von Fehlern** Die meisten Fehler finden sich in wenigen Modulen (Pareto-Prinzip).
 - 5. Pestizid-Paradoxon** Immer die gleichen Tests finden irgendwann keine neuen Fehler mehr.
 - 6. Testen ist kontextabhängig** Unterschiedliche Projekte benötigen unterschiedliche Teststrategien.
 - 7. Trugschluss der Fehlerfreiheit** Auch fehlerfreie Software kann unbrauchbar sein, wenn sie nicht die Anforderungen erfüllt.
-

2. Black-Box-Testing (Funktionsorientiertes Testen)

Definition

Beim Black-Box-Testing wird die Software ohne Kenntnis der internen Struktur getestet. Der Tester kennt nur die Spezifikation und prüft, ob die Ein- und Ausgaben korrekt sind.

Vorteile und Nachteile

Vorteile:

- Keine Programmierkenntnisse erforderlich
- Unabhängig von der Implementierung
- Testet aus Anwendersicht
- Kann von externen Testern durchgeführt werden

Nachteile:

- Interne Fehler bleiben möglicherweise unentdeckt
 - Nicht alle Codepfade werden getestet
 - Schwer zu automatisieren ohne Kenntnis der Struktur
-

3. Black-Box-Testverfahren

3.1 Äquivalenzklassenbildung

Eingabedaten werden in Klassen unterteilt, bei denen alle Werte einer Klasse gleich behandelt werden.

Beispiel: Altersvalidierung (18-65 Jahre)

Äquivalenzklassen:

- Gültige Klasse: $18 \leq \text{Alter} \leq 65$
- Ungültige Klasse 1: $\text{Alter} < 18$
- Ungültige Klasse 2: $\text{Alter} > 65$

Testfälle:

- Testwert aus gültiger Klasse: 30
- Testwert aus ungültiger Klasse 1: 15
- Testwert aus ungültiger Klasse 2: 70

3.2 Grenzwertanalyse

Teste die Grenzwerte und die Werte direkt daneben, da hier häufig Fehler auftreten.

Beispiel: Altersvalidierung (18-65 Jahre)

Testwerte: 17, 18, 19, 64, 65, 66

Begründung: An den Grenzen (18 und 65) treten häufig Fehler auf, z.B. durch falsche Operatoren ($<$ statt $\leq$).

3.3 Zustandsbasiertes Testen

Testet Zustandsübergänge eines Systems.

Beispiel: Ampelsteuerung

Zustände: Rot → Rot-Gelb → Grün → Gelb → Rot

Testfälle:

- Teste alle erlaubten Übergänge
- Teste verbotene Übergänge (z.B. von Rot direkt zu Grün)

3.4 Entscheidungstabellen

Systematische Darstellung aller Kombinationen von Bedingungen und deren Aktionen.

Beispiel: Kreditvergabe

Bedingung	Regel 1	Regel 2	Regel 3	Regel 4
Einkommen > 2000€	Ja	Ja	Nein	Nein
Schufa positiv	Ja	Nein	Ja	Nein
Aktion				
Kredit bewilligt	Ja	Nein	Nein	Nein

4. White-Box-Testing (Strukturorientiertes Testen)

Definition

Beim White-Box-Testing wird die interne Struktur des Codes getestet. Der Tester kennt den Quellcode und testet die Logik, Bedingungen und Pfade.

Vorteile und Nachteile

Vorteile:

- Alle Codepfade können getestet werden
- Logikfehler werden erkannt
- Optimierung des Codes möglich
- Hohe Testabdeckung erreichbar

Nachteile:

- Erfordert Programmierkenntnisse
- Zeitaufwendig
- Fehlende Funktionalität wird nicht erkannt
- Abhängig von der Implementierung

5. Überdeckungsmaße (Coverage-Metriken)

5.1 Anweisungsüberdeckung (Statement Coverage)

Jede Anweisung im Code wird mindestens einmal ausgeführt.

Formel:

$$\text{Anweisungsüberdeckung} = (\text{Ausgeführte Anweisungen} / \text{Gesamtanweisungen}) \times 100\%$$

C#-Beispiel:

```
csharp

public string BewerteNote(int punkte)
{
    if (punkte >= 90)
    {
        return "Sehr gut";
    }
    else if (punkte >= 75)
    {
        return "Gut";
    }
    else
    {
        return "Befriedigend";
    }
}
```

Testfälle für 100% Anweisungsüberdeckung:

- Test 1: punkte = 95 → "Sehr gut"
- Test 2: punkte = 80 → "Gut"
- Test 3: punkte = 60 → "Befriedigend"

Alle drei Rückgabeanweisungen werden ausgeführt.

5.2 Zweigüberdeckung (Branch Coverage)

Jeder Zweig (wahr/falsch) einer Bedingung wird mindestens einmal durchlaufen.

Formel:

$\text{Zweigüberdeckung} = (\text{Durchlaufene Zweige} / \text{Gesamtzweige}) \times 100\%$

C#-Beispiel:

```
csharp

public bool IstRabattGueltig(int alter, bool mitglied)
{
    if (alter < 18 || mitglied)
    {
        return true;
    }
    return false;
}
```

Testfälle für 100% Zweigüberdeckung:

- Test 1: alter = 15, mitglied = false → true (erste Bedingung wahr)
- Test 2: alter = 25, mitglied = true → true (zweite Bedingung wahr)
- Test 3: alter = 25, mitglied = false → false (beide Bedingungen falsch)

5.3 Bedingungsüberdeckung (Condition Coverage)

Jede atomare Bedingung nimmt mindestens einmal den Wert wahr und falsch an.

C#-Beispiel:

```
csharp

if (a > 5 && b < 10)
{
    Console.WriteLine("Bedingung erfüllt");
}
```

Testfälle:

- Test 1: a = 6, b = 8 → beide Bedingungen wahr
- Test 2: a = 3, b = 8 → erste Bedingung falsch, zweite wahr
- Test 3: a = 6, b = 12 → erste Bedingung wahr, zweite falsch

Damit wurde jede Teilbedingung mindestens einmal wahr und einmal falsch.

5.4 Pfadüberdeckung (Path Coverage)

Alle möglichen Pfade durch den Code werden getestet.

C#-Beispiel:

```
csharp

public decimal BerechnePreis(int menge, bool premium)
{
    decimal preis = menge * 10;

    if (menge > 100)
    {
        preis = preis * 0.9m; // 10% Rabatt
    }

    if (premium)
    {
        preis = preis * 0.95m; // 5% Premium-Rabatt
    }

    return preis;
}
```

Mögliche Pfade:

1. $\text{menge} \leq 100$, $\text{premium} = \text{false}$
2. $\text{menge} \leq 100$, $\text{premium} = \text{true}$
3. $\text{menge} > 100$, $\text{premium} = \text{false}$
4. $\text{menge} > 100$, $\text{premium} = \text{true}$

Testfälle für 100% Pfadüberdeckung:

- Test 1: $\text{menge} = 50$, $\text{premium} = \text{false} \rightarrow \text{Preis} = 500$
- Test 2: $\text{menge} = 50$, $\text{premium} = \text{true} \rightarrow \text{Preis} = 475$
- Test 3: $\text{menge} = 150$, $\text{premium} = \text{false} \rightarrow \text{Preis} = 1350$
- Test 4: $\text{menge} = 150$, $\text{premium} = \text{true} \rightarrow \text{Preis} = 1282.50$

6. Kontrollflussgraph

Was ist ein Kontrollflussgraph?

Ein Kontrollflussgraph (CFG) visualisiert alle möglichen Ausführungspfade durch ein Programm.

Elemente eines Kontrollflussgraphen

- **Knoten:** Anweisungen oder Anweisungsblöcke
- **Kanten:** Kontrollfluss zwischen Anweisungen
- **Startknoten:** Einstiegspunkt des Programms
- **Endknoten:** Austritt aus dem Programm

Beispiel-Code

```
csharp

public string PruefeBewertung(int note)
{
    string ergebnis;

    if (note >= 1 && note <= 6) // Knoten 1
    {
        if (note <= 4) // Knoten 2
        {
            ergebnis = "Bestanden"; // Knoten 3
        }
        else
        {
            ergebnis = "Durchgefallen"; // Knoten 4
        }
    }
    else
    {
        ergebnis = "Ungültige Note"; // Knoten 5
    }

    return ergebnis; // Knoten 6
}
```

Zyklomatische Komplexität

Die zyklomatische Komplexität gibt die Anzahl der linear unabhängigen Pfade an.

Formel:

$$V(G) = E - N + 2$$

$$V(G) = \text{Anzahl der Entscheidungen} + 1$$

Wobei:

- E = Anzahl der Kanten
- N = Anzahl der Knoten

Für obiges Beispiel: $V(G) = 2 + 1 = 3$

Das bedeutet, wir benötigen mindestens 3 Testfälle für vollständige Pfadüberdeckung.

7. Zusammenfassung der Überdeckungsgrade

Überdeckungsart	Beschreibung	Aufwand	Effektivität
Anweisungsüberdeckung	Jede Anweisung einmal ausführen	Niedrig	Gering
Zweigüberdeckung	Jeden Zweig einmal durchlaufen	Mittel	Mittel
Bedingungsüberdeckung	Jede Bedingung wahr/falsch	Mittel	Mittel-Hoch
Pfadüberdeckung	Alle Pfade durchlaufen	Sehr hoch	Sehr hoch

Empfehlung in der Praxis:

- Mindestens: Anweisungsüberdeckung (70-80%)
- Standard: Zweigüberdeckung (80-90%)
- Kritische Systeme: Pfadüberdeckung (90-100%)

8. Praktische Tipps für Testfälle

Gute Testfälle erstellen

1. **Ein Testfall = Ein Aspekt:** Teste nur eine Sache pro Testfall
2. **Unabhängige Tests:** Tests sollten in beliebiger Reihenfolge ausführbar sein
3. **Wiederholbare Tests:** Gleiche Eingabe = Gleiche Ausgabe
4. **Aussagekräftige Namen:** z.B. `Test_BerechnePreis_MengeUeber100_GibtRabatt`

Testdaten auswählen

- **Normalwerte:** Typische Eingaben
- **Grenzwerte:** Min/Max-Werte
- **Fehlerhafte Eingaben:** Null, negativ, zu groß
- **Sonderfälle:** Leere Strings, null-Referenzen

Häufige Fehlerquellen

- Off-by-One-Fehler ($<$ statt $\leq$)
 - Null-Referenzen
 - Division durch Null
 - Integer-Überläufe
 - Falsche Bedingungsverknüpfungen (AND statt OR)
-

9. Unit-Testing in C# mit NUnit

Beispiel eines Unit-Tests

```

csharp

using NUnit.Framework;

[TestFixture]
public class PreisberechnungTests
{
    [Test]
    public void BerechnePreis_MengeUnter100_KeineErmaessigung()
    {
        // Arrange
        var rechner = new Preisrechner();

        // Act
        decimal preis = rechner.BerechnePreis(50, false);

        // Assert
        Assert.AreEqual(500, preis);
    }

    [Test]
    public void BerechnePreis_MengeUeber100_MitRabatt()
    {
        // Arrange
        var rechner = new Preisrechner();

        // Act
        decimal preis = rechner.BerechnePreis(150, false);

        // Assert
        Assert.AreEqual(1350, preis);
    }
}

```

10. Übungsaufgaben

Aufgabe 1: Äquivalenzklassen

Erstelle Äquivalenzklassen für eine Methode, die Rabatte vergibt:

- 0-10 Artikel: kein Rabatt
- 11-50 Artikel: 5% Rabatt
- 51-100 Artikel: 10% Rabatt
- Über 100 Artikel: 15% Rabatt

Aufgabe 2: Grenzwertanalyse

Definiere Testfälle mit Grenzwertanalyse für die Rabattmethode aus Aufgabe 1.

Aufgabe 3: Pfadüberdeckung

Analysiere folgenden Code und erstelle Testfälle für 100% Pfadüberdeckung:

```
csharp

public string BerechneVersandkosten(decimal warenwert, bool express)
{
    if (warenwert >= 50)
    {
        if (express)
        {
            return "5 Euro";
        }
        return "Kostenlos";
    }
    return "8 Euro";
}
```

Glossar

Black-Box-Testing: Testen ohne Kenntnis der internen Struktur

White-Box-Testing: Testen mit Kenntnis des Quellcodes

Coverage: Testabdeckung in Prozent

CFG: Kontrollflussgraph (Control Flow Graph)

Unit-Test: Test einer einzelnen Komponente/Methode

Testfall: Konkrete Eingabe mit erwarteter Ausgabe

Assertion: Überprüfung, ob das Ergebnis dem erwarteten Wert entspricht