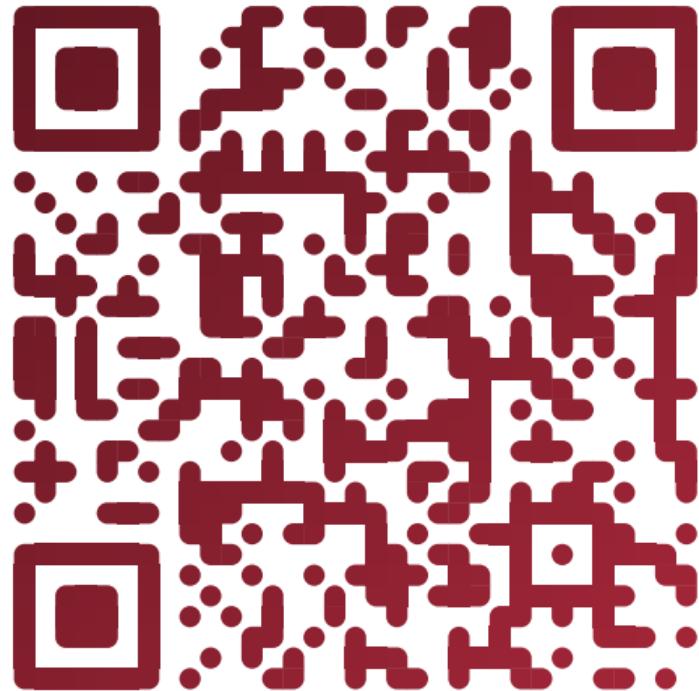


Fragen? Fragen!



**Klont schon mal das  
*exercise-tests*  
Repository!**

A photograph of several mushrooms growing in a dense forest undergrowth. The mushrooms have white caps and dark, fuzzy stems. They are surrounded by green leaves and some fallen brown leaves.

It is. It really is.

# Test.... it is important!

## 3. Großübung

Ulm University | Florian Sihler, Raphael Straub und Matthias Tichy | 7. Mai 2024



Software Engineering  
Programming Languages



universität  
**ulm**



# Hehooo!

Die Zweite!

Gibt es vorab Wünsche?

# Themenübersicht

**1. Einführung**

2. Theorie

5. Die Welt ist groß, die Welt ist schön

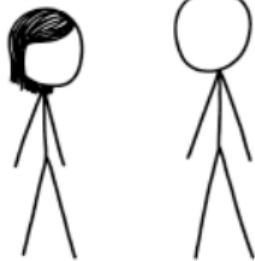
3. Unit-Tests

4. Property-Based Testing

<https://xkcd.com/2928/>

SO, DO YOU AND THE OTHER  
QA ENGINEERS HAVE ANY FUN  
PLANS FOR THE HOLIDAY?

YEAH, ASSUMING THE  
SCHEDULING SYSTEM  
DOESN'T CRASH.



SOFTWARE TESTING DAY IS A HOLIDAY  
CELEBRATED EVERY -1 YEARS ON  
JANUARY 0<sup>TH</sup> AT 25:71 PM.



# Themenübersicht

1. Einführung

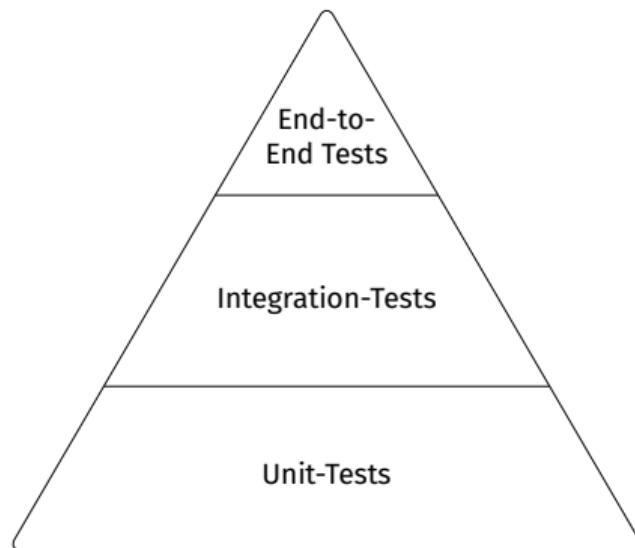
2. Theorie

3. Unit-Tests

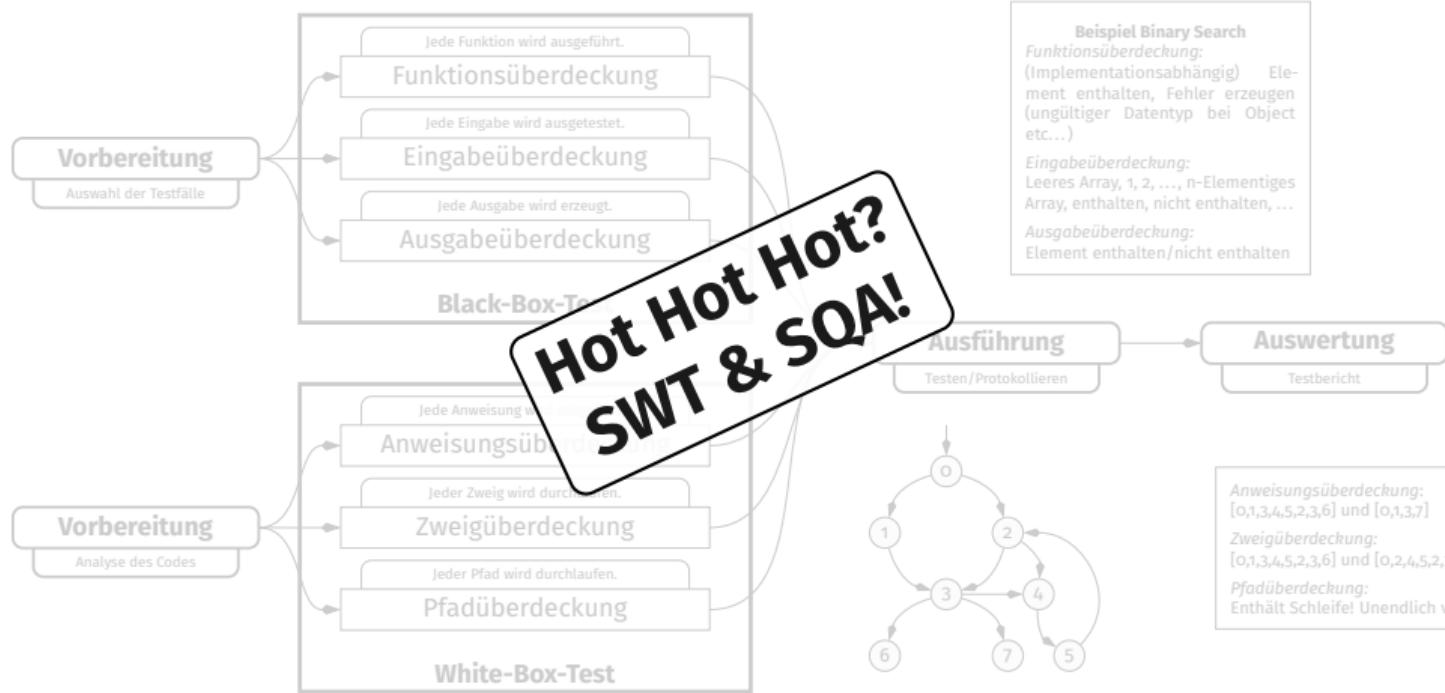
4. Property-Based Testing

5. Die Welt ist groß, die Welt ist schön

# Test-Theorie (nicht klausurrelevant)



# Test-Theorie (nicht klausurrelevant)



# Themenübersicht

1. Einführung

2. Theorie

**3. Unit-Tests**

Test-Coverage

Parameterized Tests

5. Die Welt ist groß, die Welt ist schön

4. Property-Based Testing

# Determinanten Berechnen

1a)

Zunächst soll eine Funktion `double simpleDeterminant(int[][] matrix)` geschrieben werden, welche die Determinante einer  $2 \times 2$ -Matrix berechnet. Stellen Sie zu Beginn sicher, dass es sich tatsächlich um eine Matrix der entsprechenden Größe handelt. Falls nicht, werfen Sie eine `RuntimeException` mittels `throw new RuntimeException()`.

Die Determinante einer  $2 \times 2$ -Matrix  $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$  ist definiert als  $a \cdot d - b \cdot c$ .

```
public static int simpleDeterminant(int[][] m) {  
    return m[0][0] * m[1][1] - m[0][1] * m[1][0];  
}
```

# Ein erster Unit-Test

```
public static int simpleDeterminant(int[][][] m) {  
    return m[0][0] * m[1][1] - m[0][1] * m[1][0];  
}
```

```
import org.junit.jupiter.api.Test;  
  
class MyTest {  
    @Test  
    void test1() {  
        int[][] matrix = {{1, 2}, {3, 4}};  
        assertEquals(-2, Matrix.simpleDeterminant(matrix), "Nachricht");  
    }  
}
```

Annotations: (Unit-)Test

Erwarte: -2

Tatsächlicher Wert

Implementieren Sie *zwei* weitere unit-tests für *simpleDeterminant*!

# Mögliche Tests

- Weitere In-/Output-Tests  
*Für die Eingabe X erwarte ich Y.*
- Randfälle (Edge-Cases)  
*Was passiert, wenn die Matrix Nullen enthält?*
- Fehlerfälle  
*Was passiert, wenn die Matrix nicht die richtige Größe hat?  
Was passiert, wenn die Zahlen z.B. überlaufen?*
- Performance-Tests  
*Wie lange dauert die Berechnung der Determinante?*
- ...

## Ein paar Kommentare...



- Nicht immer sind (Standard-)Tests sinnvoll!
- Es ist wichtig nicht nur “positiv”-Fälle zu testen!
- Tests können auch dokumentieren, wie Funktionen (nicht) verwendet werden sollen.
- Wählt gute und klare Testnamen!

# Test-Coverage



The screenshot shows three Java test coverage extensions available on the Microsoft Store:

- Extension Pack for Java** by Microsoft: Popular extensions for Java development that provides Java IntelliSense... (28ms)
- Gradle for Java** by Microsoft: Manage Gradle Projects, run Gradle tasks and provide better Gradle fil... (28ms)
- Coverage Gutters** by ryanluker: Display test coverage generated by lcov or xml - works with many lang... (81ms)

- Im Beispielprojekt ist die `build.gradle` schon vorkonfiguriert
- Die Coverage gibt an, wie viele Zeilen des Codes durch Tests abgedeckt sind

Hohe Coverage  $\implies$  Besserer Code?

**Nein!**

# Parameterized Tests

```
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.ValueSource;  
  
class MyTests {  
    @ParameterizedTest  
    @ValueSource(ints = {1, 2, 3, 4, 5})  
    void test2(int value){  
        assertTrue(value > 0 && value < 6);  
    }  
}
```

Annotation: Parameterized Test  
Zu verwendende Parameter  
Parameter

Implementieren Sie *einen* parameterized test für *simpleDeterminant*!

# Themenübersicht

1. Einführung

2. Theorie

3. Unit-Tests

4. Property-Based Testing

5. Die Welt ist groß, die Welt ist schön

# Property-Based Testing

QuickCheck revisited

- Wir spezifizieren *Eigenschaften* statt konkreter Testfälle
- Das Test-Framework (hier: *jwqik*) generiert automatisch Testfälle

```
import net.jqwik.api.Property;
import net.jqwik.api.ForAll;
import net.jqwik.api.constraints.IntRange;

class MyTests {
    @Property
    void test3(
        @ForAll @IntRange(min=1, max=100) int a,
        @ForAll @IntRange(min=1, max=100) int b
    ) {
        int[][] matrix = {{a, b}, {a, b}};
        assertEquals(0, Matrix.simpleDeterminant(matrix));
    }
}
```

Annotations and their descriptions:

- `@Property` → Annotation: Eigenschaft
- `@ForAll` → Zu generierende Testwerte
- `@IntRange(min=1, max=100)` → Wertebereich

Implementieren Sie *zwei* weitere property-based tests für *simpleDeterminant*!

# Noch mehr Kommentare...

- Property-Based Tests können Fälle abdecken, die manuell schwer zu finden sind
- **Aber:** Alles hängt am “Orakel”  
*Woher wissen wir, dass die generierte Lösung korrekt ist?*
- Auch Test-Code ist Code!  
*⇒ er kann genau so Fehler beinhalten!*

# Themenübersicht

1. Einführung

2. Theorie

**5. Die Welt ist groß, die Welt ist schön**

3. Unit-Tests

4. Property-Based Testing

# Die Welt ist groß, die Welt ist schön...

(leider) nicht klausurrelevant

- Es gibt ja sooooooooo viel mehr! 🎉
- Weitere Annotationen  
*@DisplayName, @Order, @TestFactory, ...*
- Weitere Test-Varianten  
*Metamorphic-, Regression-, Mutation-, Visual Gui-, Security-Testing, ...*
- Weitere Techniken  
*Mocks, Stubs, Fixtures, Contracts, Test-Driven Development, CI/CD, ...*
- Probleme und Lösungen  
*Test smells (z.B. Flakey Tests), ...*