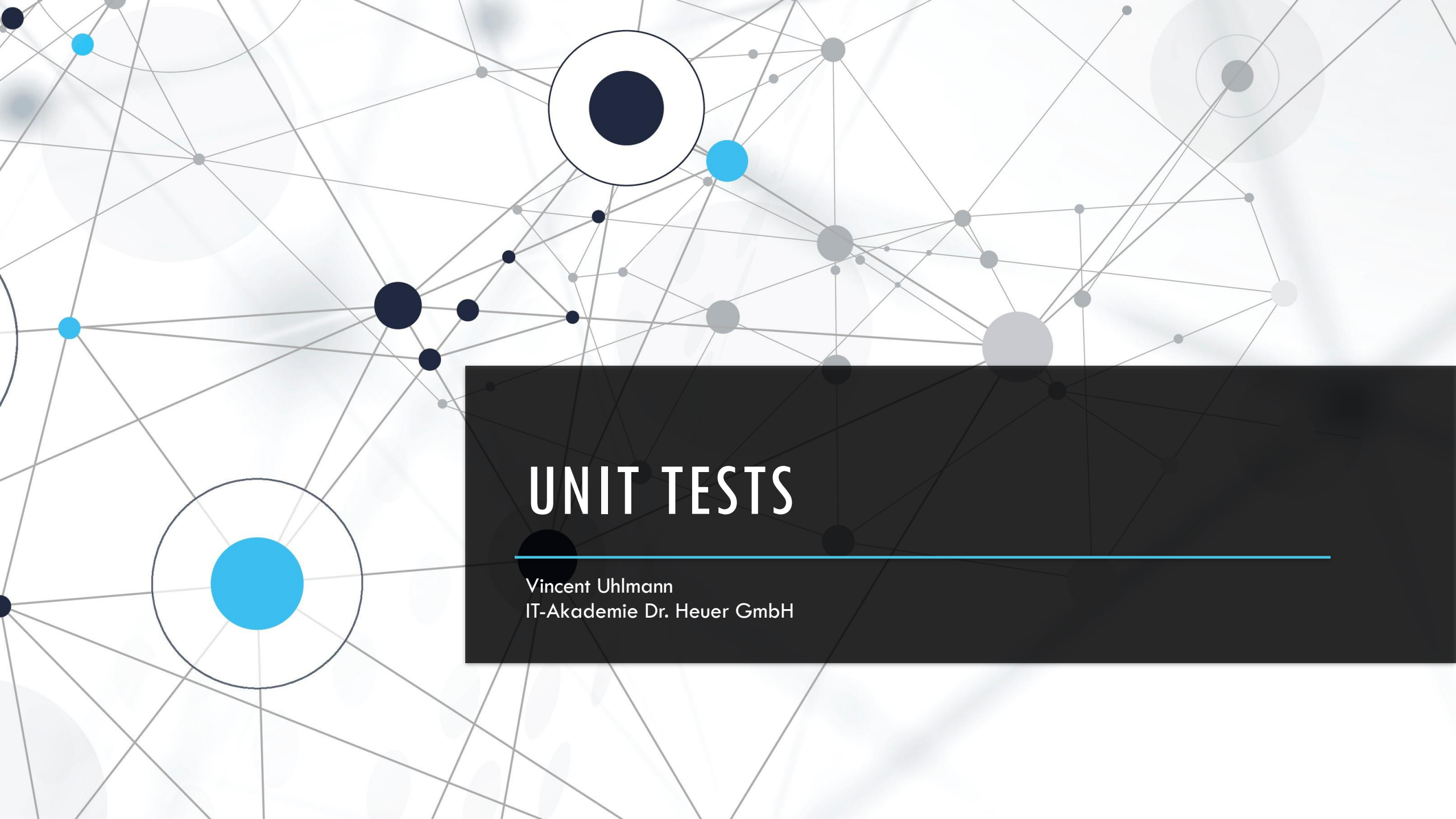

UNIT TESTS

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

ARTEN VON TESTS

- Komponententests (Unit tests)
- Testen isolierte Code-Einheiten (z. B. Methoden oder Klassen)

- Integrationstests (Integration tests)
- Ein Integrationstest unterscheidet sich von einem Komponententest darin, dass zwei oder mehr Komponenten gemeinsam getestet werden

- Auslastungstests (Load tests)
- Bei einem Auslastungstest wird ermittelt, ob ein System eine bestimmte Auslastung verarbeiten kann

- ...

UNIT TESTS

- Unit Tests sind automatisierte Tests, die kleinste, isolierte Einheiten ("Units") des Codes überprüfen

- Ziele
- Fehler frühzeitig erkennen
- Code-Qualität verbessern
- Refactoring erleichtern

MERKMALE VON UNIT TESTS

- Isoliert: Jeder Test fokussiert sich nur auf eine Komponente
- Deterministisch: Gleiche Eingaben liefern gleiche Ergebnisse

- Eine Methode, die zwei Zahlen addiert, sollte immer dasselbe Ergebnis liefern, wenn dieselben zwei Zahlen übergeben werden

BEST PRACTICES FÜR UNIT TESTS

- AAA-Pattern
- Arrange: Initialisiere Testdaten
- Act: Führe die zu testende Aktion aus
- Assert: Überprüfe die Ergebnisse

BEST PRACTICES FÜR UNIT TESTS

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

```
public class CalculatorTests
{
    [Fact]
    public void Add_ShouldReturnSum()
    {
        // Arrange
        var calculator = new Calculator();

        // Act
        var result = calculator.Add(2, 3);

        // Assert
        Assert.Equal(5, result);
    }
}
```

BEST PRACTICES FÜR UNIT TESTS

```
public int Divide(int a, int b)
{
    if (b == 0)
        throw new DivideByZeroException();

    return a / b;
}
```

```
[Fact]
public void Divide_WhenDenominatorIsZero_ShouldThrowException()
{
    var calculator = new Calculator();

    Assert.Throws<DivideByZeroException>(() => calculator.Divide(10, 0));
}
```

WIE WIRD GETESTET?

- Vorbereitung (Arrange): Initialisiere Testobjekte, Daten und Abhängigkeiten

```
var calculator = new Calculator();
```

- Durchführung (Act): Führe die Methode oder Funktion aus

```
var result = calculator.Add(2, 3);
```

- Prüfung (Assert): Vergleiche das Ergebnis mit den erwarteten Werten

```
Assert.Equal(5, result);
```

UNIT TESTS MIT XUNIT

- xUnit: Beliebtes Framework für Unit Tests in .NET

- Attribute
- [Fact]: Test ohne Parameter
- [Theory]: Test mit parametrisierten Daten
- [InlineData]: Bereitstellung von Testdaten für Theories
-

ASSERT-KLASSE

- Die Assert-Klasse stellt Methoden bereit, um Ergebnisse zu überprüfen

- Assert.Equal(expected, actual): Vergleicht zwei Werte
- Assert.NotEqual(notExpected, actual): Prüft, dass zwei Werte ungleich sind
- Assert.True(condition) / Assert.False(condition): Prüft boolesche Bedingungen
- Assert.Null(object) / Assert.NotNull(object): Prüft auf Nullwerte
- Assert.Throws<TException>(action): Prüft, ob eine Ausnahme geworfen wird
- ...

UNIT TESTS MIT XUNIT

```
[Fact]
public void Constructor_WhenValueIsNegative_ShouldThrowException()
{
    Assert.Throws<ArgumentOutOfRangeException>(() => new Calculator(-1));
}


[Theory]
[InlineData(1, 2, 3)]
[InlineData(5, 5, 10)]
[InlineData(-1, -1, -2)]
public void Add_ShouldReturnSum_ForMultipleInputs(int a, int b, int expected)
{
    var calculator = new Calculator();
    var result = calculator.Add(a, b);
    Assert.Equal(expected, result);
}
```

UNIT TESTS MIT XUNIT

- https://xunit.net/docs/getting-started/v2/netcore/cmdline