06

INTERNATIONALE HOCHSCHULE

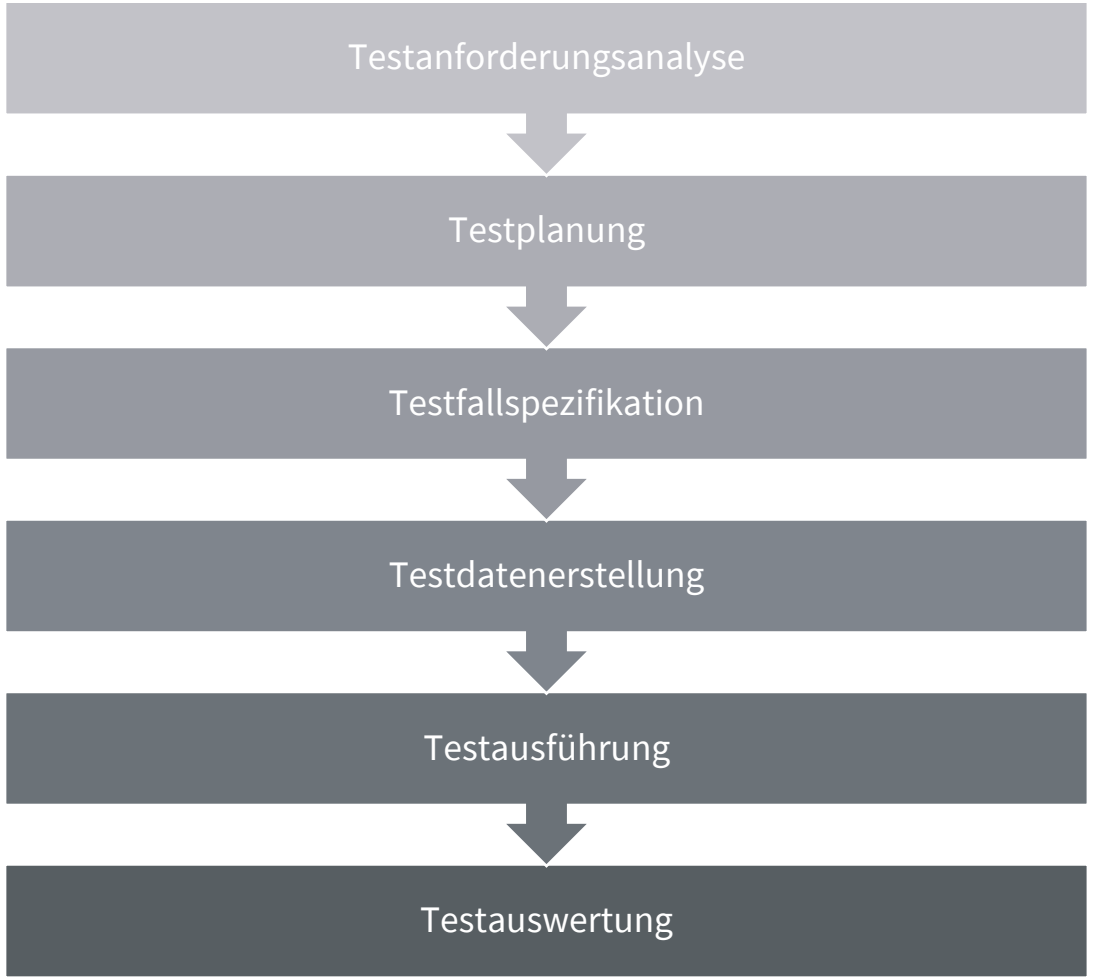
# SYSTEMATISCHES TESTEN VON SOFTWARE

# LERNZIELE

1. Aktivitäten zum methodischen Testen
2. Komponententests, inkl. automatische Unit Tests und Test Driven Development
3. Integrationstests, Integrationsstrategien
4. System- und Abnahmetests

# AKTIVITÄTEN ZUM METHODISCHEN TESTEN

- Aktivitäten sind in allen Teststufen vorhanden
- Anpassungen je Teststufe bei Inhalt und Ausgestaltung




Quelle

# DIE TESTPLANUNG: WELCHE FRAGEN MÜSSEN GEKLÄRT WERDEN

| | | |
|---|---|---|
| Der Test | **Was?** | Welche Objekte und Funktionen sind zu testen? |
| | **Warum?** | Welche Ziele verfolgt der Test? |
| | **Wann?** | Welche Termine sind einzuhalten? |
| | **Wo?** | Wo soll getestet werden? |
| | **Wie?** | Unter welchen Bedingungen soll getestet werden? |
| | **Womit?** | Mit welche Mitteln und Werkzeugen soll getestet werden? |
| | **Wer?** | Welche Mitarbeiter führen den Test durch? |

# REMINDER: TESTFALLSPEZIFIKATION: GRUNDGRÖßE TESTFALL

Handlungsanweisung zur Durchführung von Softwaretests

besteht (wenigstens) aus folgenden Elementen

| Beschreiben Daten zum Testfall (z.B. Name, ID, …) | Vorbedingun-gen | Testaktionen | Testdaten | Nachbedin-gungen |
|---|---|---|---|---|

Schiffmann, J. (2022). Qualitätssicherung im Softwareprozess – Dynamische Qualitätssicherung. IU.

# REMINDER: BEISPIEL TESTFALLDEFINITION 1

| Template für konkrete Testfälle | |
|---|---|
| ID des Testfalls | Eindeutige ID |
| Testgegenstand | Offizieller Name der zu testenden Systemkomponente oder der Schnittstelle oder der Funktion des Systems/der Komponente |
| Getestetes Detail | Name der Klasse(n) oder Komponente(n), die bei diesem Test getestet werden |
| Methode(n), die getestet werden | Detaillierter Name der Methode/der Funktionen/der Schnittstelle und deren Parameterliste |
| Kurzbeschreibung des Tests | Kurze Beschreibung, was mit dem Test geprüft bzw. sichergestellt werden soll |
| Testdaten/Eingabe | Welche Eingabedaten werden benötigt, um den Testfall durchzuführen? Mit welchen Parametern sollen die zu testenden Funktionen aufgerufen werden? Falls es sich um Objekte handelt: Welche Attribute sollen sie enthalten? Falls es sich um Dateien handelt: Welchen Inhalt haben sie? |
| Erwartetes Ergebnis/ Erwartete Systemreaktion | Welche Ausgaben oder Systemreaktionen werden zu den Testdaten erwartet? |
| Vorbereitung/Vorbedingung | Konkrete Schritte, um das System/die Komponente für den Test vorzubereiten:<br>• Erzeugen oder Einspielen von Testdaten in die Datenbank<br>• Aufrufen bestimmter Methoden, damit das System in den Zustand gelangt, in dem getestet werden kann (Bsp.: Anmelden von Nutzern am System)<br>• Sicherstellen, dass bestimmte Daten in der Datenbank vorliegen |

Schiffmann, J. (2022). Qualitätssicherung im Softwareprozess – Dynamische Qualitätssicherung. IU.

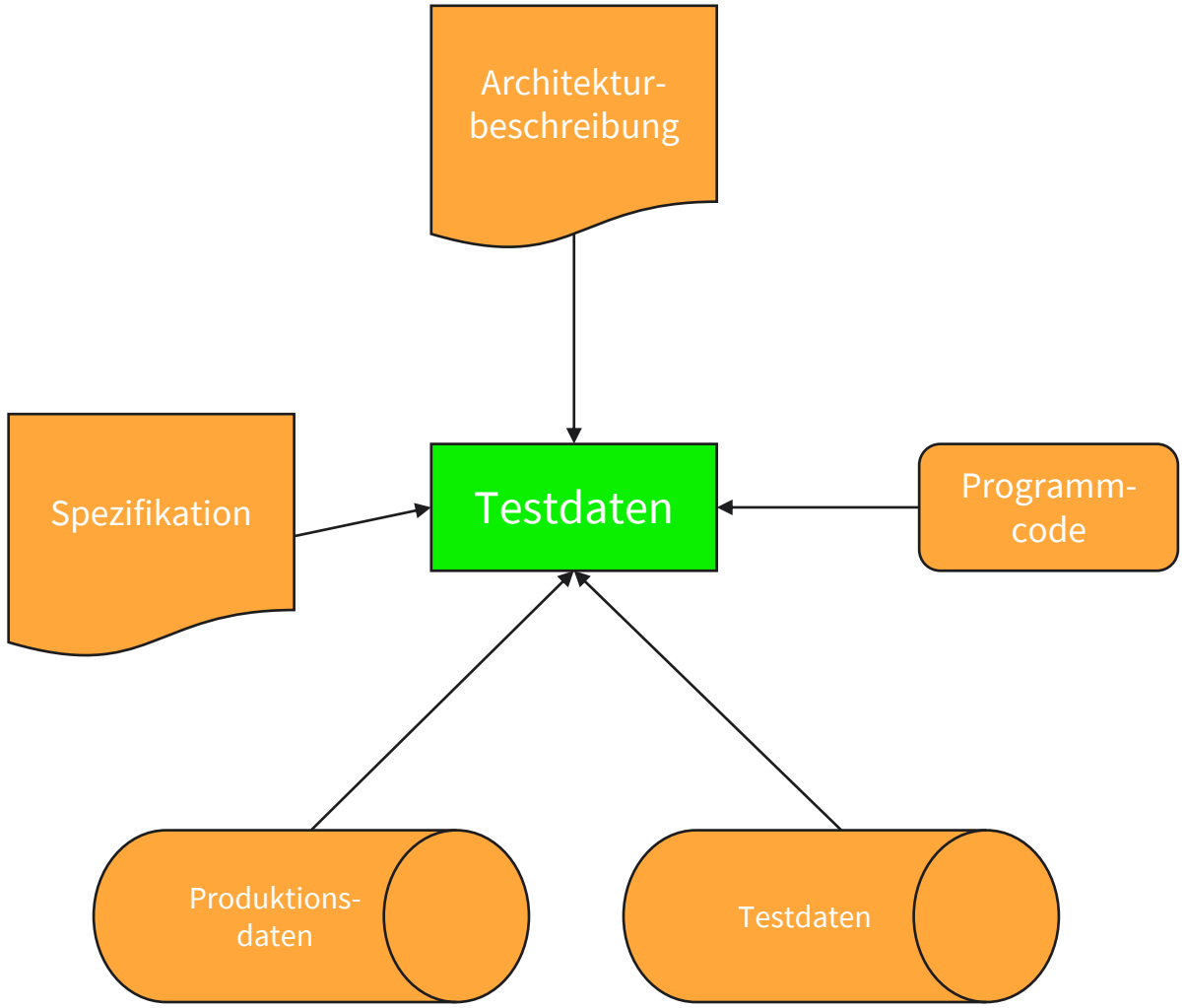
# REMINDER: BEISPIEL TESTFALLDEFINITION 2

| Template für konkrete Testfälle | |
|---|---|
| ID des Testfalls | Eindeutige ID |
| Durchführung | Welche Systemfunktion(en)/Methode(n) muss/müssen beim Test ausgeführt werden?<br>Bei komplexeren Testfällen mit mehreren Aufrufen: In welcher Reihenfolge müssen die Methoden/System-funktionen ausgeführt werden? |
| Überprüfung/Nachbe-dingung | Geben Sie hier die konkreten Bedingungen eines bestandenen Tests an.<br>Welches Systemverhalten/welche Ausgaben werden vom System erwartet?<br>Welche Kriterien müssen die im System gespeicherten Daten erfüllen?<br>Bei der Erfüllung welcher Kriterien ist der Test erfolg-reich? Wie werden die Kriterien gemessen? |

Schiffmann, J. (2022). Qualitätssicherung im Softwareprozess – Dynamische Qualitätssicherung. IU.

# TESTDATENERSTELLUNG

- Synthetische oder reale Testdaten
- Erstellung
- Management des Testdatenbestands



# ERSTELLUNG VON (ZUFALLS-)TESTDATEN

Gruppenarbeit (15 Min.):

Erklärt an einem Beispiel eurer Wahl:

- Was sind Charakteristika beim Arbeiten mit Testfällen bei eurem Projekt → Welche Ausprägungen (Funktionen und Situationen) kann bei eurer Softwarelösung getestet werden?

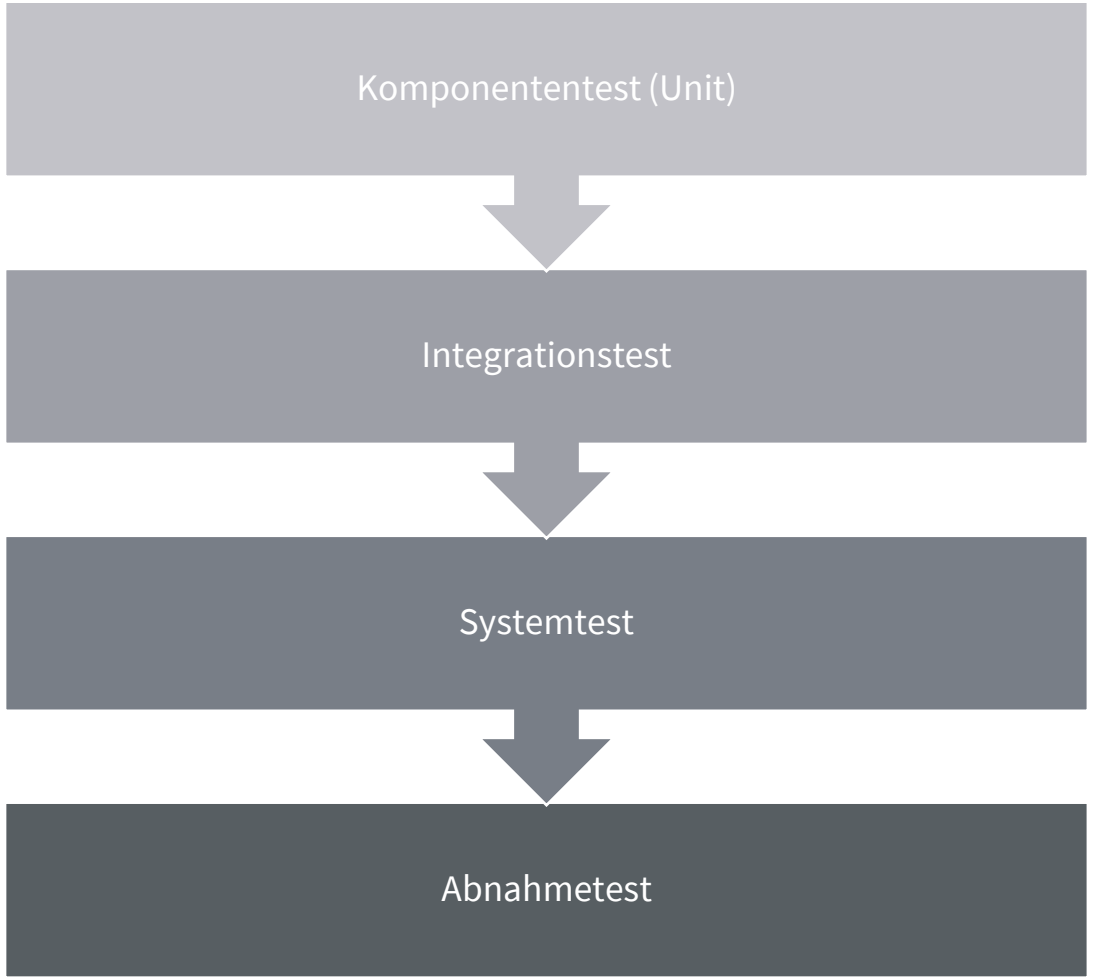
- Bei Systemen mit Datenverwaltung: Aus welchen Quellen können Testdaten für den Testfall stammen? Wie können diese für euer Beispiel erstellt werden? (grobe Codebeispiele → ggf. Spezifikation für ChatGPT)

# ERGEBNISSE

# TESTSTUFEN | QUALITÄTSSICHERUNG VON SOFTWARE IN MEHREREN TESTSTUFEN

- Testen erfolgt (soll erfolgen) entwicklungsbegleitend
- Teststufen orientieren sich an Entwicklungsstufen

- Welche Eigenschaften getestet werden hängt **auch** von den Teststufen ab



# KOMPONENTENTEST

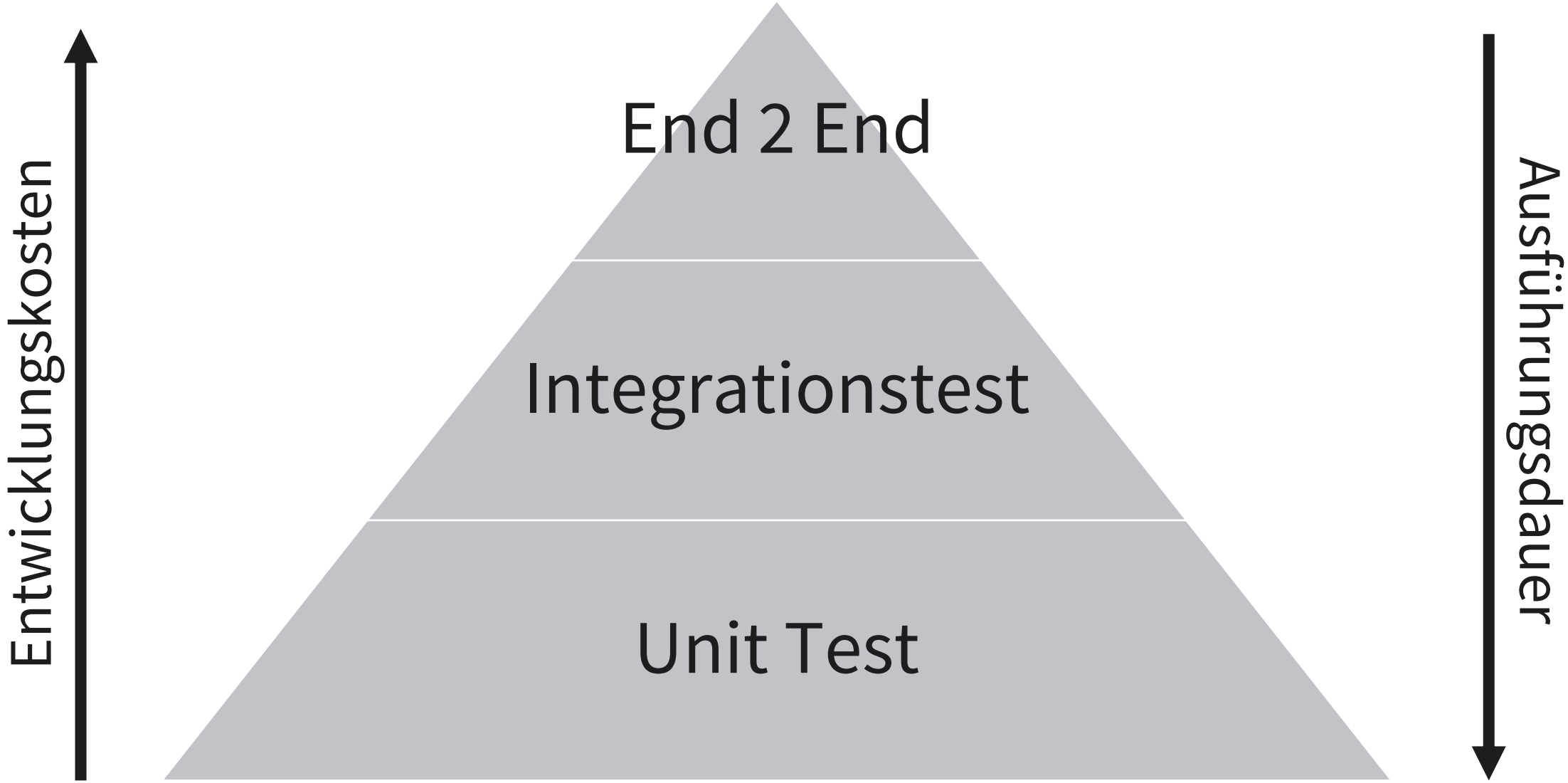
Eigenschaften:

- Reproduzierbarkeit
- Unabhängigkeit
- Regressionsfähigkeit

Scope: logische Komponente

- Komponenten werden einzeln getestet (vor Integration)
- Hoher Automatisierungsgrad (Frameworks für gängige Programmiersprachen)
- Whitebox-Test

# BERÜCKSICHTIGUNG VON TESTKOSTEN




Lenger, M. (o. D.). Was ist ein guter Unit-Test? – Und wie entwickelt man ihn? Informatik Aktuell. https://www.informatik-aktuell.de/entwicklung/methoden/was-ist-ein-guter-unit-test-und-wie-entwickelt-man-ihn.html

# WAS IST EIN UNITTEST?

Ein einem Unit-Test wird eine Einheit im Programm durch einen Test abgedeckt: Wie groß diese Einheit ist, ist nicht genau definiert:

- Im klassischen Sinne ist dieses eine Methode, ein Modul oder eine Klasse (für jede Klasse wird eine Testklasse geschrieben).

- Es können auch mehrere Methoden oder ein größeres Modul sein → es sollte ein abgrenzbarer Teil eines Programms sein, welcher eigenständig Daten verarbeitet.

# WAS BEINHALTET EIN UNIT TEST?

- Es werden Testfälle mit Akzeptanzkriterien beschrieben.
  - Es kann aber auch auf Erfüllung oder auf Fehler (Exception) getestet werden: Eine Exception wird erwartet, aber nicht ausgeliefert --> Der Test ist fehlgeschlagen

- Bei Testfällen sollte man die verschiedenen Abläufe und Zustände in einem Programm (oder auch in einer Unit) berücksichtigen und dieses Testen.

# WELCHE ABLÄUFE KÖNNEN HIER EINTRETEN?

## Beispielprogramm:

```
while (B) {
  A
  }
C
```

# WELCHE ABLÄUFE KÖNNEN HIER EINTRETEN?

INTERNATIONALE HOCHSCHULE

## Beispielprogramm:

```
while (B) {
  A
  }
C
```

C = Abbruch

AC = Die Schleife wird einmal durchlaufen

AAC = Die Schleife wird mehrfach durchlaufen

# DER UNTERSCHIED ZWISCHEN UNIT- UND INTEGRATIONSTESTS

- Ein Unit-Test überprüft einzelne Komponente oder Einheit einer Anwendung isoliert von anderen Komponenten.
- Ein Integrationstest überprüft, wie verschiedene Komponenten oder Einheiten einer Softwareanwendung zusammenarbeiten.
- Integrationstests werden nach den Unit-Tests durchgeführt
- Unit-Tests werden von Entwicklern geschrieben und sind in der Regel schnell auszuführen.

# DISKUSSION ZUM START VON TEST DRIVEN DEVELOPMENT

- Warum kann Testautomatisierung bei der Softwareentwicklung wichtig sein?
  - Überlegt auch in diesem Kontext:
    - Welche Systeme zur Testautomatisierung kennt und nutzt ihr?
    - Programmiert ihr defensiv (Arbeiten mit Tests – Tests first) oder offensiv (Tests second oder gar nicht?)?

# TEST DRIVEN DEVELOPMENT (TDD)

- Einsatz in der agilen Softwareentwicklung:
  - „With incremental development, there is no system specification that can be used by an external testing team to develop system tests.“

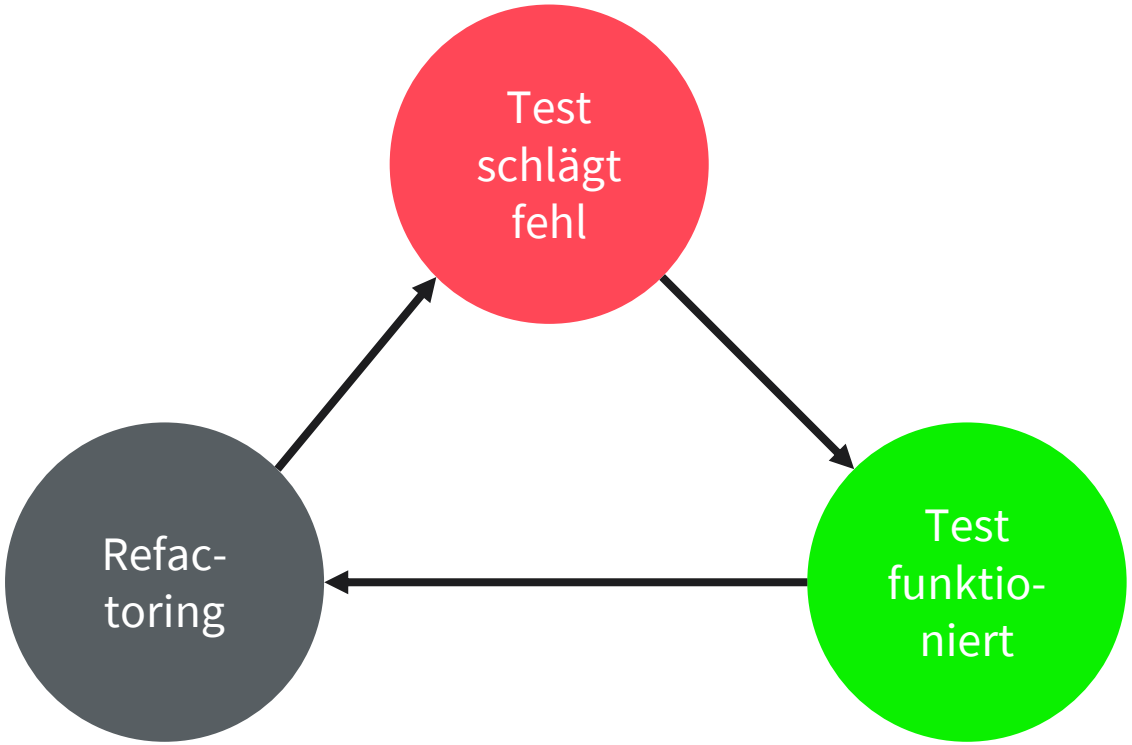
- TDD lenkt den Fokus früh auf ggf. notwendige Interfaces (Schnittstellen), was Missverständnisse reduziert (Wann arbeite ich Wo mit Daten? Wann werden diese Wo übergeben?).

- Der fachliche Input des Kunden fließt in die Entwicklung des Tests ein, sodass dieser zur Spezifikation der Anforderung wird.

Sommerville, Ian (2016): Software Engineering. Tenth Edition. Boston, Columbus, Indianapolis u. a.: PEARSON. Harlow: Pearson Education Limited. 2016. S. 81-82.

# TEST DRIVEN DEVELOPMENT (TDD)

- Zuerst werden anhand der Anforderung (mehrere und passende) Testfälle entwickelt: Die Implementation erfolgt so lang, bis die Testfälle durch den Code erfüllt werden



# TEST DRIVEN DEVELOPMENT (TDD)

- TDD erfordert den Einsatz automatisierter Testframeworks, z. B. JUnit (Java) oder NUnit (.net). Diese Frameworks bieten Funktionen wie Assertions und Testläufer.

- Assertions sind Anweisungen, die in einem Unit-Test verwendet werden, um zu überprüfen, ob das Ergebnis einer Funktion dem erwarteten Ergebnis entspricht.

- Ein Beispiel für eine Assertion in Java ist:

```
assertEquals(erwartetesErgebnis, tatsächlichesErgebnis);
```

Sommerville, Ian (2016): Software Engineering. Tenth Edition. Boston, Columbus, Indianapolis u. a.: PEARSON. Harlow: Pearson Education Limited. 2016. S. 81-82.

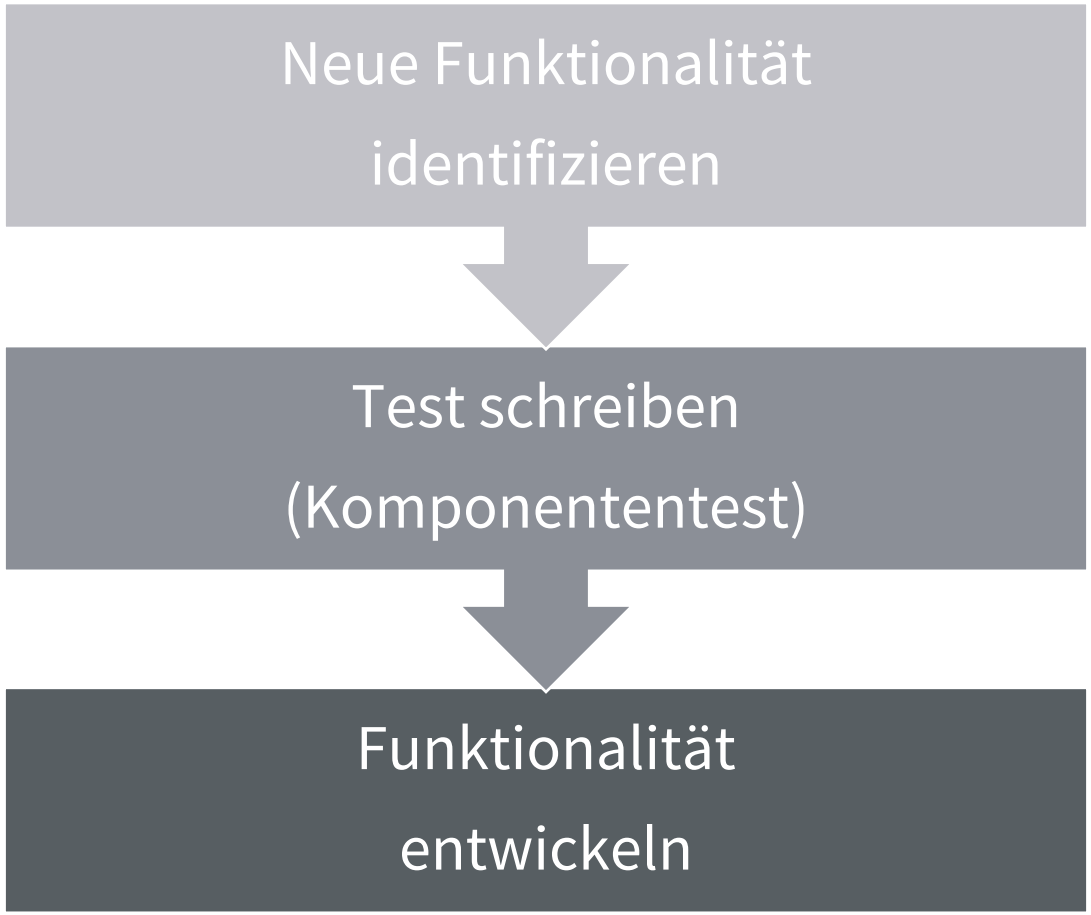
# TEST DRIVEN DEVELOPMENT (TDD)

- **Testläufer** sind Programme, die die Tests ausführen, indem sie die Assertions ausführen und das Ergebnis auswerten.
  - Sie können mehrere Tests gleichzeitig ausführen und die Ergebnisse analysieren, um zu sehen, ob die Tests erfolgreich waren oder nicht.

- Tests vor jedem Commit beugt Problemen vor dem Release vor

Sommerville, Ian (2016): Software Engineering. Tenth Edition. Boston, Columbus, Indianapolis u. a.: PEARSON. Harlow: Pearson Education Limited. 2016. S. 81-82.

# TEST DRIVEN DEVELOPMENT (TDD)



# BEISPIEL TEST JUNIT

```
import org.junit.jupiter.api.Test;
import static
org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {
  @Test
  public void testAddition() {
    Calculator calculator = new Calculator();
    int result = calculator.add(2, 3);
    assertEquals(5, result);
  }
}
```

@Test ist ein Annotationstyp, der in der JUnit-Bibliothek verwendet wird, um eine Methode als Testmethode zu markieren.

In unserem Fall wird die Methode testAddition() als Testmethode markiert, um zu überprüfen, ob die Methode add() des Calculators korrekt funktioniert.

# HERAUSFORDERUNGEN DES TEST DRIVEN DEVELOPMENT (TDD)

- Teilweise wird von der Regel abgewichen, zuerst den Testfall zu entwickeln
- Mitunter sind Tests ineffizient, da sie nicht die volle Funktionalität testen
- Das User-Interface lässt sich schwer testen (Sommerville, S. 83)

Sommerville, Ian (2016): Software Engineering. Tenth Edition. Boston, Columbus, Indianapolis u. a.: PEARSON. Harlow: Pearson Education Limited. 2016. S. 81-82.

# ÜBUNG ZU TEST-DRIVEN-DEVELOPMENT

Das Projekt IT-Kaffeebohne steht kurz vor dem Abschluss. Ihr müsst kurzfristig als Entwickler einspringen. Die Anforderung, die umzusetzen ist, lautet wie folgt:

> Entwickelt eine Funktion um für eine gegebene Temperatur zu entscheiden, ob der Kaffeevollautomat auf den Becher einen Warnhinweis „Vorsicht heiß“ druckt. Der Warnhinweis soll ausgegeben werden, wenn die Temperatur 50°C überschreitet.

Die Aufgabe:

- Wie kann ein Softwaretest für dieses Beispiel aussehen? (ChatGPT ist zugelassen)
- 15 Minuten in den Gruppen (versucht verschiedene Programmiersprachen)

# BEISPIELLÖSUNGEN

# BEISPIELLÖSUNG TEST-DRIVEN-DEVELOPMENT

```
public static void hotCoffeeWarning(double temperature) {
   if (temperature > 50) {
      System.out.println("Vorsicht heiß!");
   }
}
import org.junit.Test;
public class HotCoffeeWarningTest {
   @Test
   public void testHotCoffeeWarning() {
      double temperature = 60;
      HotCoffeeWarning.hotCoffeeWarning(temperature);
      assertEquals("Vorsicht heiß!", outContent.toString());
   }
}
```

# INTEGRATIONSTEST

Scope: Zusammenspiel der Komponenten

- Sobald zwei Komponenten fertig gestellt sind, können diese in einen Integrationstest treten
- Treiber und Dummies als Testmöglichkeiten
- Integrationsstrategien

Integrationsstrategien

- Bottom-Up
- Top-Down
- By Value

# INTEGRATIONSTEST: TREIBER UND DUMMIES

- Als Treiber werden dabei die Softwarefragmente bezeichnet, die das Aufrufen anderer Komponenten simulieren.
- Dummies hingegen simulieren Komponenten, die durch andere Komponenten aufgerufen werden.

# SYSTEMTESTS

**Ziel des Systemtests ist die Überprüfung, ob das System als ganzes die spezifizierten Anforderungen erfüllt**

Herausforderungen:

- möglichst originalgetreue Nachbildung der Produktivumgebung des Kunden
- Bereitstellung von originalgetreuen Datensätzen
- Realistisch Auslastung und Nutzerverhalten

# TESTARTEN SIND MANNIGFALTIG


Düsterbeck, F. 2014, Testen mit, durch und in Scrum, abgerufen am 25.03.2023 von https://de.slideshare.net/fduesterbeck/frank-dsterbeck-testen-mit-durch-und-in-scrum-asqf

# TESTARTEN IM SYSTEMTEST

Testarten unterscheiden sich nach

- zu prüfendem Qualitätsmerkmal
- Testzielen
- Testobjekt (z.B. System oder Dokumentation)

# TESTARTEN IM SYSTEMTEST

Mögliche Testarten im Systemtest:

- funktionaler Systemtest
- Performancetest
- Lasttest
- Usability-Test
- Wiederinbetriebnahmetest

# SYSTEMTEST: LAST- UND PERFORMANCETEST

**Lasttest (auch Stresstest):** Das Ziel ist explizit das Testen des Systemverhaltens unter besonders hohen Anforderungen an die zur Verfügung stehenden Ressourcen.

## Performancetest:

| Kontext | Definition | Beispiel |
|---|---|---|
| **Latenz** | Zeitspanne, die von der Anfrage an das System bis zur Beantwortung vergeht. | Die Zeit, die das System bei der Bestellung in einem Onlineshop vom Klick auf den OK-Button bis zur Anzeige des Bestätigungsdialogs benötigt. |
| **Durchsatz** | Verarbeitungsgeschwindigkeit von Funktionen oder Daten. | Die Anzahl der verarbeiteten Bestellvorgänge pro Minute. |
| **Transaktionsrate** | Maß zur Verarbeitungsgeschwindigkeit von definierten, komplexen Änderungen am Datenbestand. | Die Anzahl der gebuchten Überweisungen im Kernsystem einer Direktbank. |

# SHORTDISCUSSION

INTERNATIONALE HOCHSCHULE

## Blitzlicht:

Warum ist es nicht sinnvoll Testes direkt miteinander zu kombinieren?

Wieso kann man keinen Lasttest während des Integrationstest durchführen?

# REGRESSIONSTESTS

Regressionstests = Testfälle, die bereits erfolgreich durchgeführt worden sind und nach einer Anpassung des Systems erneut durchgeführt werden.

Die Eigenschaft Regressionsfähigkeit von Testfällen bezeichnet daher die Fähigkeit zur wiederholten Durchführung des Tests.

# ZIEL DES REGRESSIONSTEST

- „Ziel des Regressionstests ist nachzuweisen, dass Modifikationen von Software keine unerwünschten Auswirkungen auf die Funktionalität besitzen.

- Durch Software-Modifikationen, z. B. durch Funktionalitätserweiterungen oder Fehlerkorrekturen können neue Fehler in zuvor fehlerfreie Teile eingefügt werden.“

Liggesmeyer, Peter (2009): Software-Qualität. Testen, Analysieren und Verifizieren von Software. 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, Springer. S. 192.

# ZIEL DES REGRESSIONSTEST

- „Der Regressionstest zielt auf die Erkennung derartiger Seiteneffekte.
- Weil nicht davon ausgegangen werden kann, dass korrekt abgearbeitete Testfälle auch nach Änderungen der Software weiterhin korrekt abgearbeitet werden, reicht die einmalige Durchführung von Testfällen nicht aus.
- Streng genommen, müssten alle bisherigen Testfälle nach Modifikationen wiederholt werden.“

Liggesmeyer, Peter (2009): Software-Qualität. Testen, Analysieren und Verifizieren von Software. 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, Springer. S. 192.

# ZIEL DES REGRESSIONSTEST

- Ein Regressionstest besteht aus der Wiederholung von bereits durchgeführten Testläufen:
  - Die zu testende Software ist pro durchzuführendem Testfall in den gleichen Zustand zu versetzen wie bei der vorhergehenden Durchführung dieses Testfalls.
  - Es  sind identische Eingaben vorzunehmen.
  - Die neu erzeugten Ausgaben sind mit den Ausgaben der Vorläuferversion zu vergleichen.
  - Falls keine Unterschiede auftreten, ist der Regressionstestfall erfolgreich absolviert.
  - Falls Unterschiede erkannt werden, so ist zu prüfen, ob diese gewünscht sind, oder ob sie fehlerhafterweise aufgetreten sind.“

Liggesmeyer, Peter (2009): Software-Qualität. Testen, Analysieren und Verifizieren von Software. 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, Springer. S. 192f.

# REGRESSIONSTESTS UND TESTAUTOMATISIERUNG

- „Der so genannte Regressionstest prüft das Verhalten einer aktuellen Software Version gegen das Verhalten der Vorläuferversion. Der Regressionstest besitzt eine hohe praktische Bedeutung. “

- „Regressionstests sind möglichst zu automatisieren und zwar einschließlich des Soll-/Ist-Ergebnisvergleichs.“

Liggesmeyer, Peter (2009): Software-Qualität. Testen, Analysieren und Verifizieren von Software. 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, Springer. S. 179, 436.

# GROUPWORK

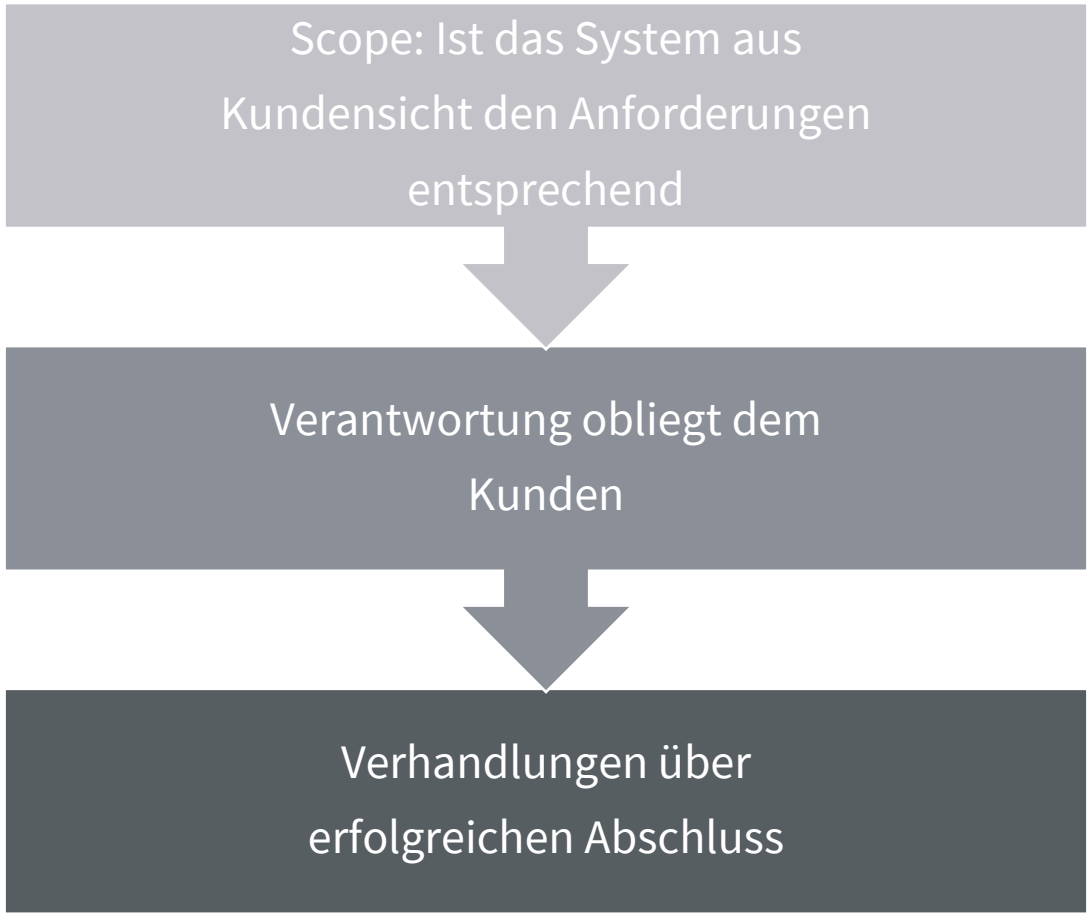
Gruppenarbeit, 20 Minuten:

Wählt ein praxisnahes Beispiel aus:

- Zeigt an diesem Beispiel den Unterschied zwischen einem Last- und Performancetest
- Zeigt an eurem Beispiel Möglichkeiten für standardisierte (Unit) Tests und einen Regressionstest

# ERGEBNISSE

# ABNAHMETESTS



INTERNATIONALE HOCHSCHULE

## Kurzes Reminder-Blitzlicht:

Warum sollte es eine Trennung von Fehlersuche und Fehlerbehebung geben?

Warum kann dieses Vorgehen wichtig und sinnvoll sein?