Softwaretesting

Prof. Dr. Raphael Herding

Qualitätssicherstellung der Softwareentwicklung

- Zugehörige Fragen:
  - Entwickeln wir das richtige Produkt?
  - Entwickeln wir (das Produkt) richtig?
- Ziel der Qualitätssicherung ist es, beide Fehlerklassen (produkt- bzw. prozessbezogen)

  aufzudecken, zu beheben und gleichzeitig die anderen Qualitätsziele wie Wartbarkeit,

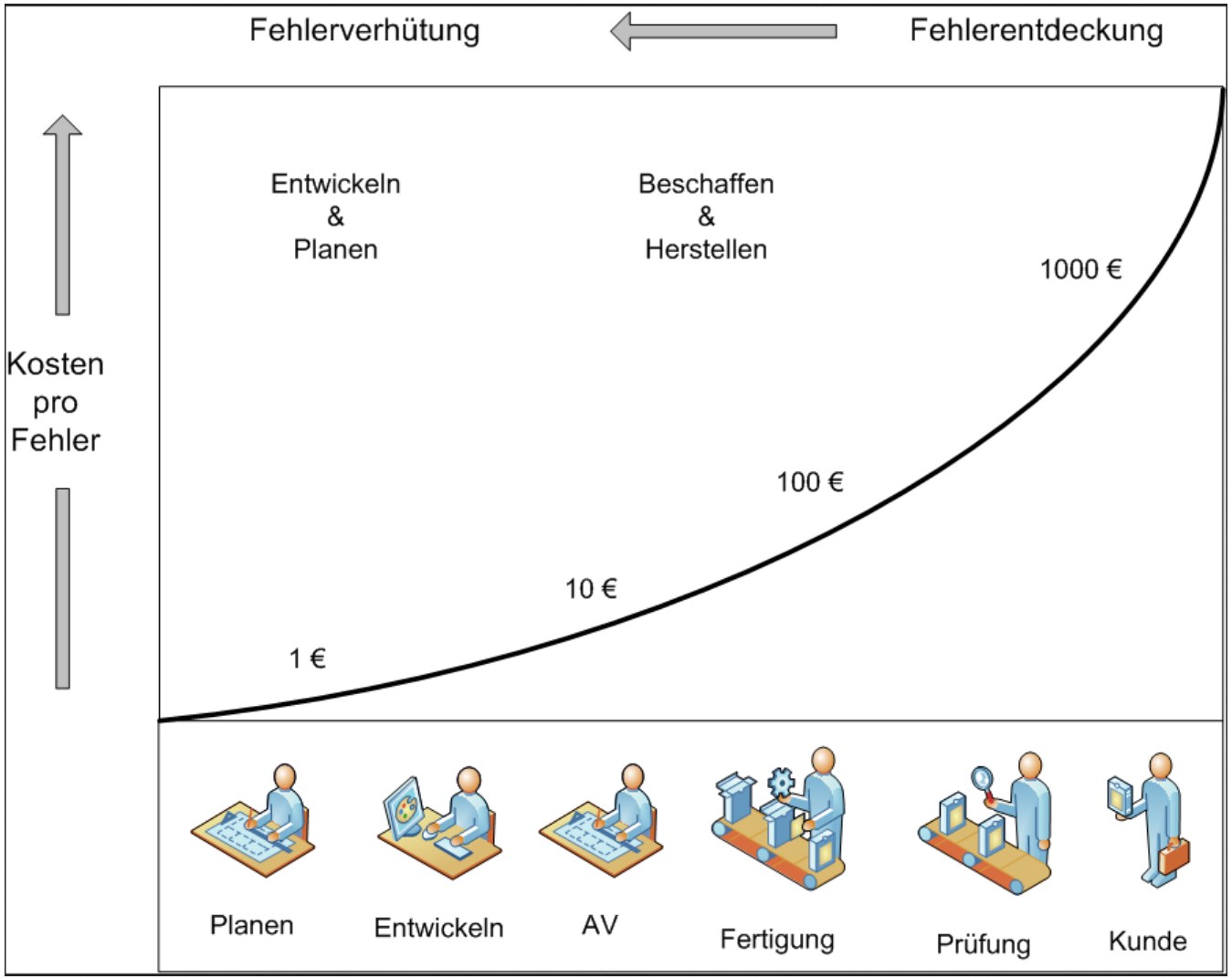
  Robustheit und Effizienz sicherzustellen.
- Schlussfolgerungen:
  - Qualitätssicherung muss phasenübergreifend betrieben werden.
  - Qualitätssicherung muss geplant und kontrolliert (messbar) werden.
  - Qualitätssicherung benötigt Ressourcen (Personal, Zeit, Geld).

Software Testing

- Testen soll zeigen, dass ein Programm das tut, was es tun soll.
- Programmfehler aufdecken, bevor es in Produktion geht.
- Im Test werden zumeist künstlichen Daten genutzt.
- Ergebnisse werden auf Fehler, Anomalien oder auf nichtfunktionale Eigenschaften geprüft.

- Sie können das Vorhandensein von Fehlern aufdecken, NICHT aber deren Abwesenheit.
- Keine Software kann auf vollständige Fehlerfreiheit getestet werden.

Software Testing



Software Testing - Ziele

- **Validation Tests:** Entwickler und Kunden zeigen, dass die Softwareanforderungen erfüllt sind.
  - Es muss mindestens einen Test für jede Anforderung im Anforderungsdokument geben.
  - Es muss Tests für alle Systemfunktionen sowie für Kombinationen dieser geben.

- **Defect Tests:** Aufdecken von Situationen, in denen das Verhalten der Software falsch oder nicht spezifikationskonform ist.
  - Es geht darum, unerwünschtes Systemverhalten wie Systemabstürze, unerwünschte Interaktionen mit anderen Systemen, fehlerhafte Berechnungen und Datenbeschädigungen aufzudecken.

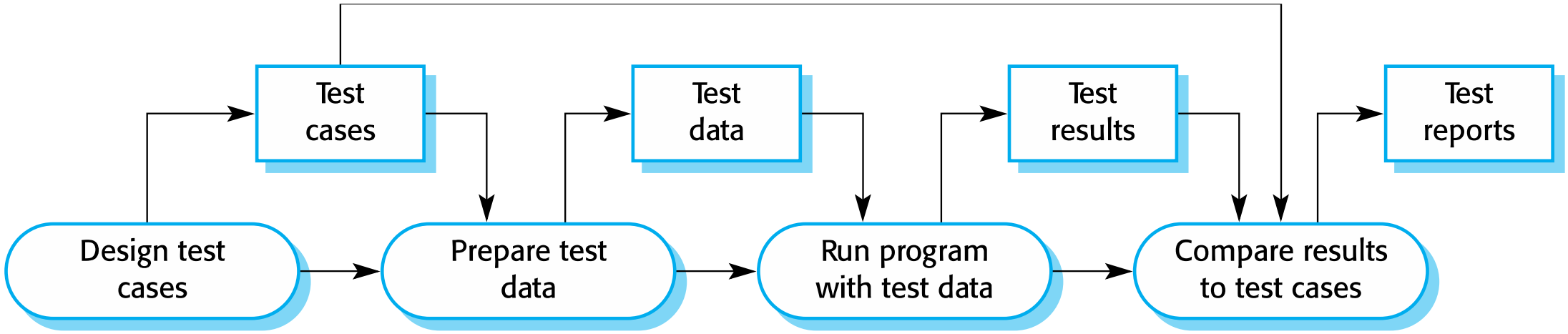
Software Testing - Verifizierung und Validierung

- **Verifizierung -** „Entwickeln wir das Produkt richtig“
- **Validierung -** „Entwickeln wir das richtige Produkt“
- Ziel V & V: Vertrauen zu schaffen, dass das System "fit for purpose" ist.
- Grad des „Vertrauens“ hängt vom Einsatzzweck ab
  - **Zweck der Software** – wie wichtig ist die Software für eine Organisation
  - **Erwartungen der Benutzer** – Benutzererwartungen variieren stark
  - **Marketing-Umgebung** - Markteinführung kann wichtiger sein als das Auffinden von Fehlern im Programm

Software Testing – statische und dynamische Tests

- **Software-Inspektionen** (**statische Verifikation**) - statische Systemanalyse um Probleme zu entdecken.
  - Programm wird nicht ausgeführt
  - Code Reviews um Anomalien und Fehler frühzeitig zu finden
  - Kann auf verschiedenen Abstraktionsebenen angewendet werden (Code, Architektur, Skripte, …)
  - Kann zur Qualitätsverbesserung eingesetzt werden
- **Softwaretests (dynamische Verifikation)** - Befasst sich mit dem Testen und Beobachten des Systemverhaltens.
  - Programm wird mit Testdaten ausgeführt
- Beides ist nötig. Sie ergänzen sich.

Software Testing - Verifizierung und Validierung



Software Testing - Klassifikation

- **Statische Verfahren** – Programm wird nicht ausgeführt
- **Dynamische Verfahren** – Programm wird mit Testdaten ausgeführt

- **Analysierende Verfahren** – Messen von Codequalität
- **Verifizierende Verfahren** – Beweis der Systemkorrektheit

- **White-Box-Verfahren** – in den Programmcode „hineinschauen“
- **Black-Box-Verfahren** – Aussensicht (Interface Testing)

Software Testing Klassifikation

- **Statische Verfahren** – Programm wird nicht ausgeführt
  - Häufig kommen Code-Analysetools zum Einsatz (SonarCube, Checkstyle, …)
- **Dynamische Verfahren**
  - Programm wird mit Testdaten ausgeführt
  - Aufgrund einer systematischen Auswahl von Testdaten sollen möglichst zahlreiche Fehler aufgedeckt werden

- Statische und Dynamische Testverfahren werden in der Praxis häufig eingesetzt

Software Testing - Klassifikation

- In der Praxis kommt häufig Unit-Testing zum Einsatz (dynamisches Testen).
- In Java durch JUnit –Framework unterstützt.

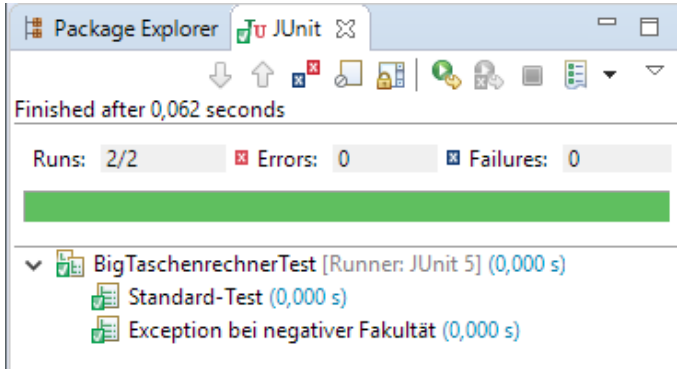
```java
class BigTaschenrechnerTest {

    @Test
    @DisplayName("Standard-Test")
    void testFakultaet() {
        assertEquals(BigInteger.ONE, BigTaschenrechner.fakultaet(0));
        assertEquals(BigInteger.ONE, BigTaschenrechner.fakultaet(1));
        assertEquals(BigInteger.valueOf(2), BigTaschenrechner.fakultaet(2));
        //assertEquals(78865786736479050355236321393218506229513597768717326329474253324435944
    }

    @Test
    @DisplayName("Exception bei negativer Fakultät")
    void testFakultaetException() {
        assertThrows(IllegalArgumentException.class, () -> {BigTaschenrechner.fakultaet(-1);});
    }
}
```

```java
import java.math.BigInteger;

public class BigTaschenrechner {
    public static BigInteger fakultaet(int zahl) {
        if (zahl < 0) throw new IllegalArgumentException();
        if (zahl == 0) return BigInteger.ONE;
        BigInteger ergebnis = BigInteger.ONE;
        for (int i=1;i<=zahl;i++) {
            ergebnis = ergebnis.multiply(BigInteger.valueOf(i));
        }
        return ergebnis;
    }
}
```



Software Testing - Klassifikation

- **Analysierende Verfahren**
  - Versuchen aufgrund einer rein syntaktischen Analyse Aussagen über die Qualität zu machen
  - Verfahren sind häufig automatisierbar (Metriken, Anomalie-Analyse) oder werden manuell durchgeführt (Review, Walkthrough)
  - **Automatisierte Verfahren** - liefern kostengünstig erste Hinweise auf Problembereiche
  - **Reviews** - lassen sich immer in allen Phasen der Softwareentwicklung einsetzen und helfen, grobe Fehler und Probleme frühzeitig zu entdecken

- **Verifizierende Verfahren**
  - Beweisen die *formale Korrektheit* von Analyse-, Entwurfs-, oder Programmdokumenten oder wichtigen Eigenschaften der Softwaredokumente
  - Aufwendig und komplex
  - Nicht automatisierbar
  - Wird in sicherheitskritischen Bereichen angewendet

Software Testing - Klassifikation

- **White-Box-Verfahren**
  - Setzen Kenntnis der inneren Strukturen des Softwaresystems voraus.
  - Nutzt diese Strukturkenntnisse beim Testen.
- **Black-Box-Verfahren**
  - Reine Außensicht
  - Innere Struktur ist unwichtig
  - Nur offene Schnittstellen werden berücksichtigt
  - Interface-Testing

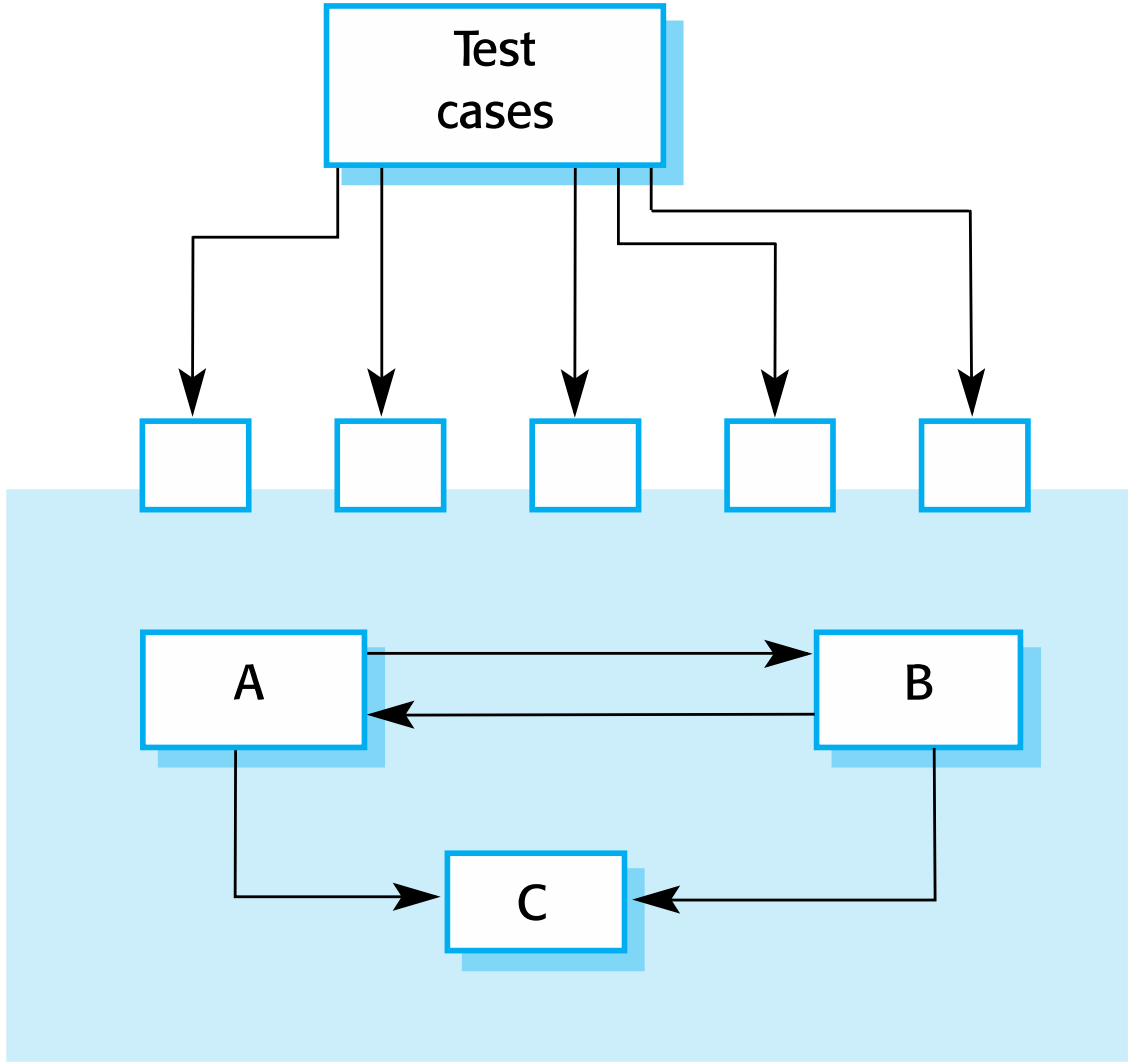
Interface Testing

- Schnittstellenfehler oder ungültige Annahmen über Schnittstellen sollen erkannt werden.

- Interface-Typen
  - **Parameter Interface -** Daten, die von einer Methode oder Prozedur an eine andere übergeben werden.
  - **Shared-Memory-Schnittstellen** - Ein Speicherblock wird von Prozeduren oder Funktionen gemeinsam genutzt.
  - **Prozedurale Schnittstellen** - Teilsystem kapselt eine Reihe von Prozeduren, die von anderen Teilsystemen aufgerufen werden.
  - **Schnittstellen zur Nachrichtenübermittlung** - Subsysteme fordern Dienste von anderen Subsystemen an.

- Je nach Anwendungstyp existieren verschiedene Interface-Typen.

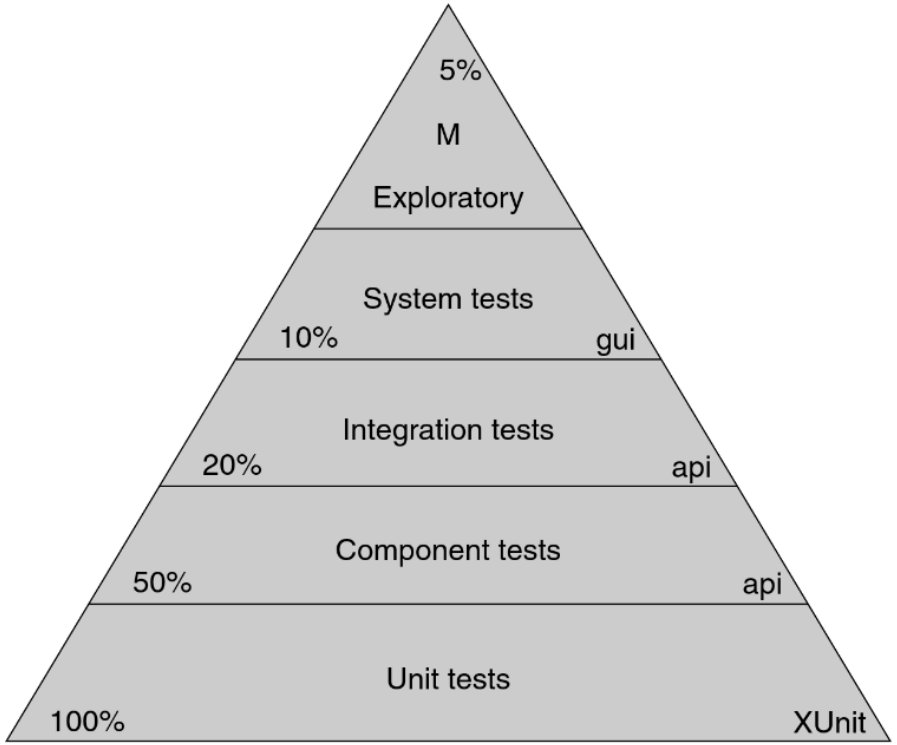
Interface Testing



Interface Testing – Probleme und Fehler

- Schnittstellenmissbrauch
  - Eine Komponente ruft eine andere Komponente auf und macht einen Fehler bei der Verwendung ihrer Schnittstelle (z. B. Parameterreihenfolge falsch).
- Schnittstellenmissverständnis
  - Eine Komponente macht falsche Annahmen über das Verhalten einer aufzurufenden Komponente (z.B. Missverständnis über Datenfelder und deren inhaltliche Bedeutung).
- Timing-Errors (Zeitliche Fehler)
  - Aufrufende und aufgerufene Komponente arbeiten mit unterschiedlichen Geschwindigkeiten.

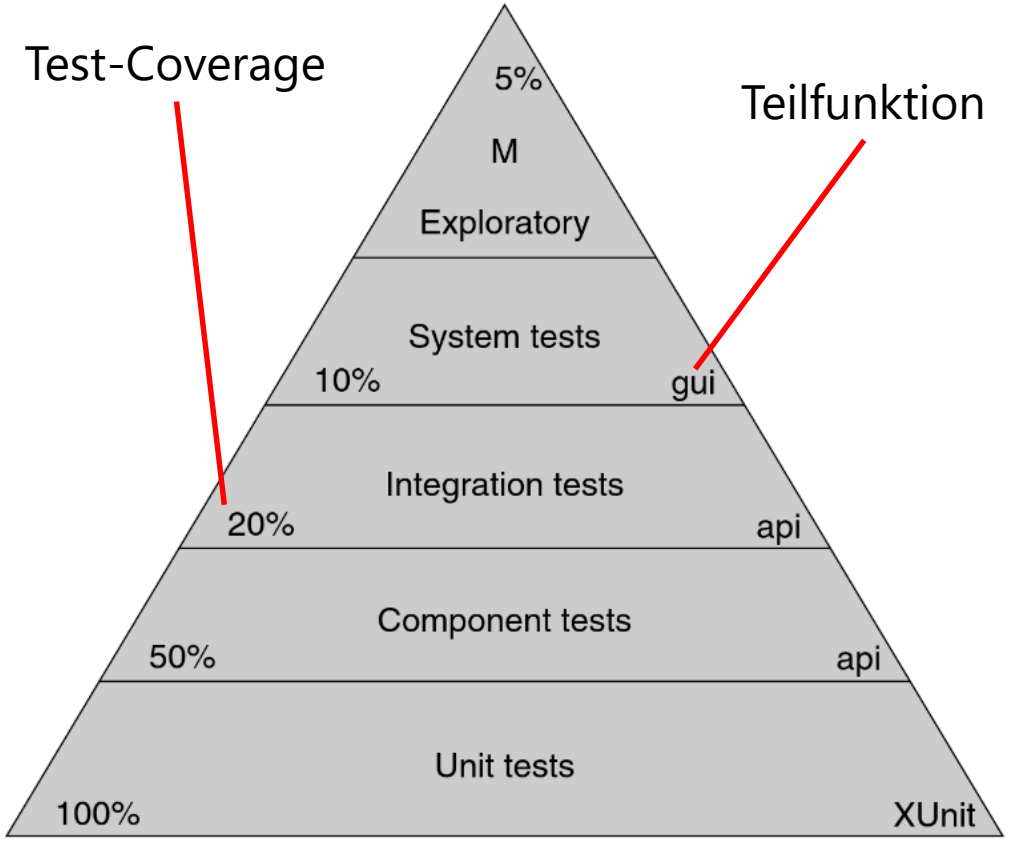
Software Testing - Testpyramide



- Mehrere Testebenen sind erforderlich, um eine gute Testabdeckung sicherzustellen
- Ziel: möglichst hohe Testabdeckung, um Software Qualität zu erhöhen
- Es gibt ja nach Anwendungsfall verschiedene Pyramiden (hier Robert C. Martin)
- Test, wann immer möglich, automatisieren

Software Testing - Testpyramide

- **Unit-Tests** – kleine Einheit im Programmcode (einzelne Klasse oder Methode/n)
- **Component-Tests** – Zusammenstellung aus mehreren Units (Logischer Klassenverbund)
- **Integration Tests** – Software vollständig erstellt und lauffähig. Integration aller Komponenten miteinander wird getestet. End-To-End Tests, Security Tests und Performance Tests werden hier durchgeführt



Software Testing - Testpyramide

- **System Tests –** Testen des „installierten“ Gesamtsystems. Testen der Oberfläche Ende-zu-Ende.

- **Exploratory (Erkundung, Sondierung) –** Manuelle Tests, nachdem das System installiert wurde. Letzte Qualitätssicherungsmaßnahmen. „Wie fühlt sich Produkt an“.