BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Datenschutz/-sicherheit

und die Notwendigkeit von Tests

von Kevin Böhme und Rico Ukro

- [ ] Check Abbreviations
- [ ] Discuss:
  - Use Mentimeter (or similar) opinion poll?

  Questions: Do you regularly test your software? Will you change your testing habits after this presentation?
- [ ] Collect all inline quotes
- [x] Make example solutions for all exercises as files
- [x] Number the exercises
- [x] Choose online platform to propagate for exercises
  - onlinegdb.com
  - ...
- [ ] Maybe, Add offset for slide content, because of `fixed-top` heading
- [x] Add example exercise link in the last slide
  - Difficult - match the presentation with markdown or the generated pdf?
- [ ] Vote: Add this topic: [Role of Test Automation in a CI/CD

Agenda

- Was sind Tests?
- Warum Tests?
- Test Driven Development (TDD)
- Grundlagen von Tests und Testverfahren
  - Unit Tests
  - Integration Tests
  - System Tests
  - Fuzzing
  - Penetration Tests
  - Weiterführende Testverfahren
- Mocking
- Umsetzung von Tests
- Testverfahren für Datensicherheit im Detail
- Datenschutz: Besonderheiten beim Testen
- Quellen

Was sind Tests?

> *„Software testing is the process of evaluating and verifying that a software product or application does what it’s supposed to do. The benefits of good testing include preventing bugs and improving performance.“*
>
> Quelle: https://www.ibm.com/topics/software-testing

Warum Tests?

Negativ Beispiele

- 1985: Kanadische Strahlentherapie Therac-25
  - Softwarefehler führte zu tödlicher Strahlendosis
  - 3 Verletzte, 3 Tote
- 1994: Airbus A300 der China Airlines
  - Absturz aufgrund eines Softwarefehlers
  - 264 Tote

Negativ Beispiele

- 1996: US-Bank Softwarefehler
  - 823 Kunden fälschlicherweise 920 Millionen US-Dollar gutgeschrieben
- 2015: Bloomberg-Terminal Absturz in Londom
  - Softwarefehler legte 300.000 Händler auf den Finanzmärkten lahm
  - Britische Regierung musste den Verkauf von 3 Mrd. Pfund Staatsanleihen verschieben

Gesetzliche Anforderungen (DSGVO)

- Geeignete technische und organisatorische Maßnahmen
- Schutz personenbezogener Daten durch Tests
- Sicherstellung der System- und Datensicherheit

Artikel 5 – Grundsätze der Verarbeitung

- Angemessene Sicherheit sicherstellen
- Schutz vor unbefugter Verarbeitung und Datenverlust

> *„in einer Weise verarbeitet werden, die eine angemessene Sicherheit der personenbezogenen Daten gewährleistet [...] durch geeignete technische und organisatorische Maßnahmen („Integrität und Vertraulichkeit“);“* Quelle: Artikel 5 Abs. 1(f) DSGVO

Artikel 25 – Datenschutz durch Technikgestaltung

- **Privacy by Design**: Datenschutz in der Entwicklung berücksichtigen
- Systeme vor Einführung testen

> *„[...] trifft der Verantwortliche [...] geeignete technische und organisatorische Maßnahmen – wie z. B. Pseudonymisierung –, die dafür ausgelegt sind, die Datenschutzgrundsätze wie etwa Datenminimierung wirksam umzusetzen und die notwendigen Garantien in die Verarbeitung aufzunehmen, um den Anforderungen dieser Verordnung zu genügen und die Rechte der betroffenen Personen zu schützen.“* Quelle: Artikel 25 Abs. 1 DSGVO

Artikel 32 – Sicherheit der Verarbeitung

- Regelmäßige Tests vorgeschrieben
- Vertraulichkeit, Integrität und Verfügbarkeit sicherstellen
- Systeme und Prozesse evaluieren

> *„Ein Verfahren zur **regelmäßigen** Überprüfung, Bewertung und Evaluierung der Wirksamkeit der technischen und organisatorischen Maßnahmen zur Gewährleistung der Sicherheit der Verarbeitung.“* Quelle: Artikel 32 Abs. 1(f) DSGVO

Aus technischer Sicht am Beispiel

```
response = requests.get("https://api.example.com/data")

if response.status_code != 200:
    # Handle error
else:
    # Handle response
```

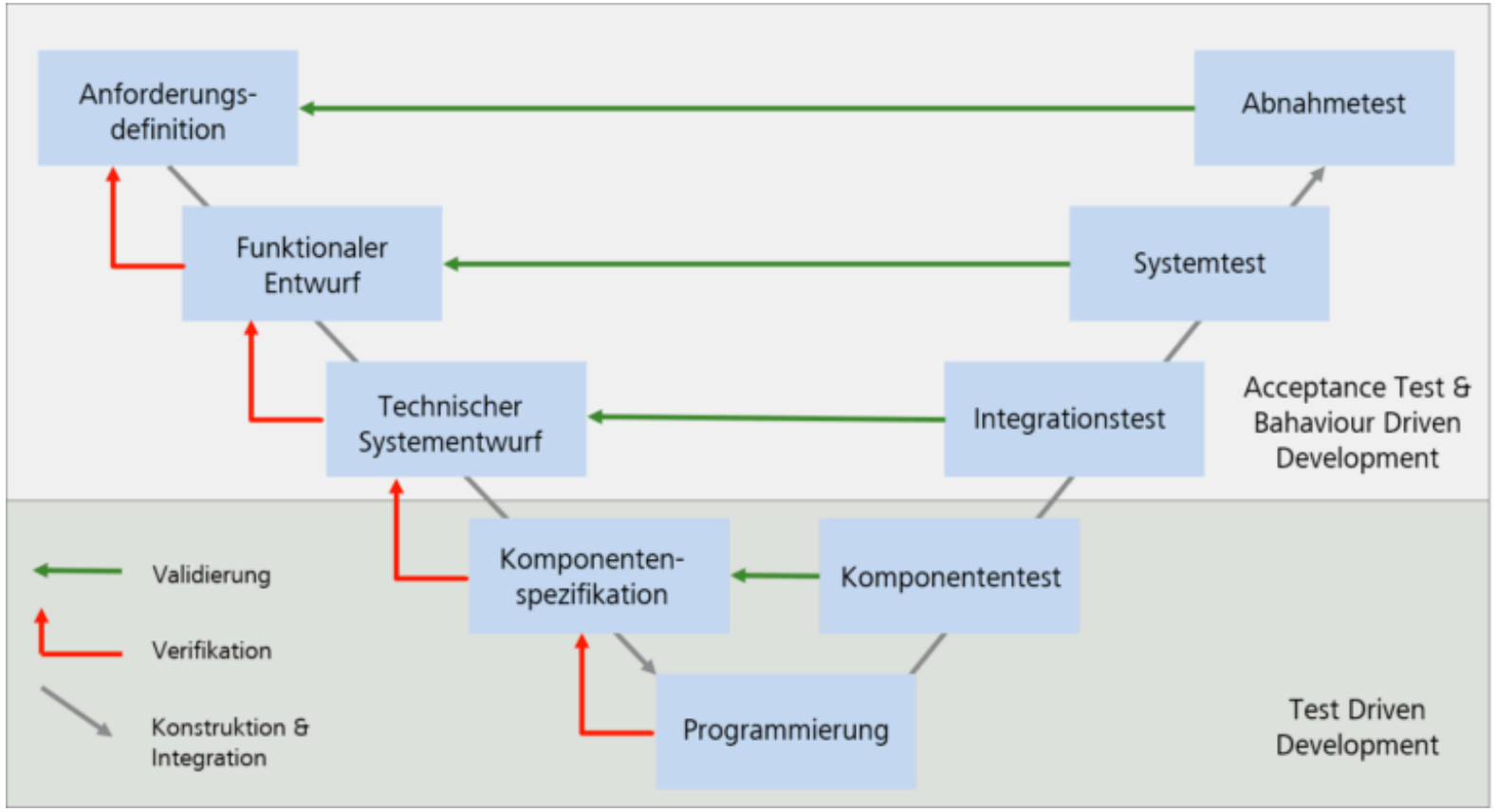
Warum ist Testing in diesem Kontext wichtig?

- Fehlererkennung: API antwortet korrekt, Fehler werden richtig gehandhabt
- Fehlertoleranz: Anwendung reagiert robust auf verschiedene Antwortcodes und Netzwerkausfälle
- Sicherheit: Sicherstellen, dass keine sensiblen Daten kompromittiert werden
- Zuverlässigkeit: API-Abfrage funktioniert konsistent, auch bei hoher Last oder langsamen Netzwerken
- Validierung der Logik: Richtig implementierte Logik für verschiedene Statuscodes und Inhalte

Grundlagen von Tests und Testverfahren

Test Driven Development (TDD)

- Wasserfallmodell: Testen erst am Projektende vorgesehen
- V-Modell: Zeitlich klar definierte Abfolge der Testphasen
- Agiles Umfeld:
  - Tests müssen häufig und unter gleichen Bedingungen durchführbar sein
  - Geringer Aufwand für Testausführung erforderlich
  - Tests sollten zeitnah zur Umsetzung der Funktionalität bereitstehen
  - Ziel: Tests müssen mit dem ständigen Wandel Schritt halten



Quelle: https://blogs.zeiss.com/digital-innovation/de/test-driven-development/

Überblick über Testarten

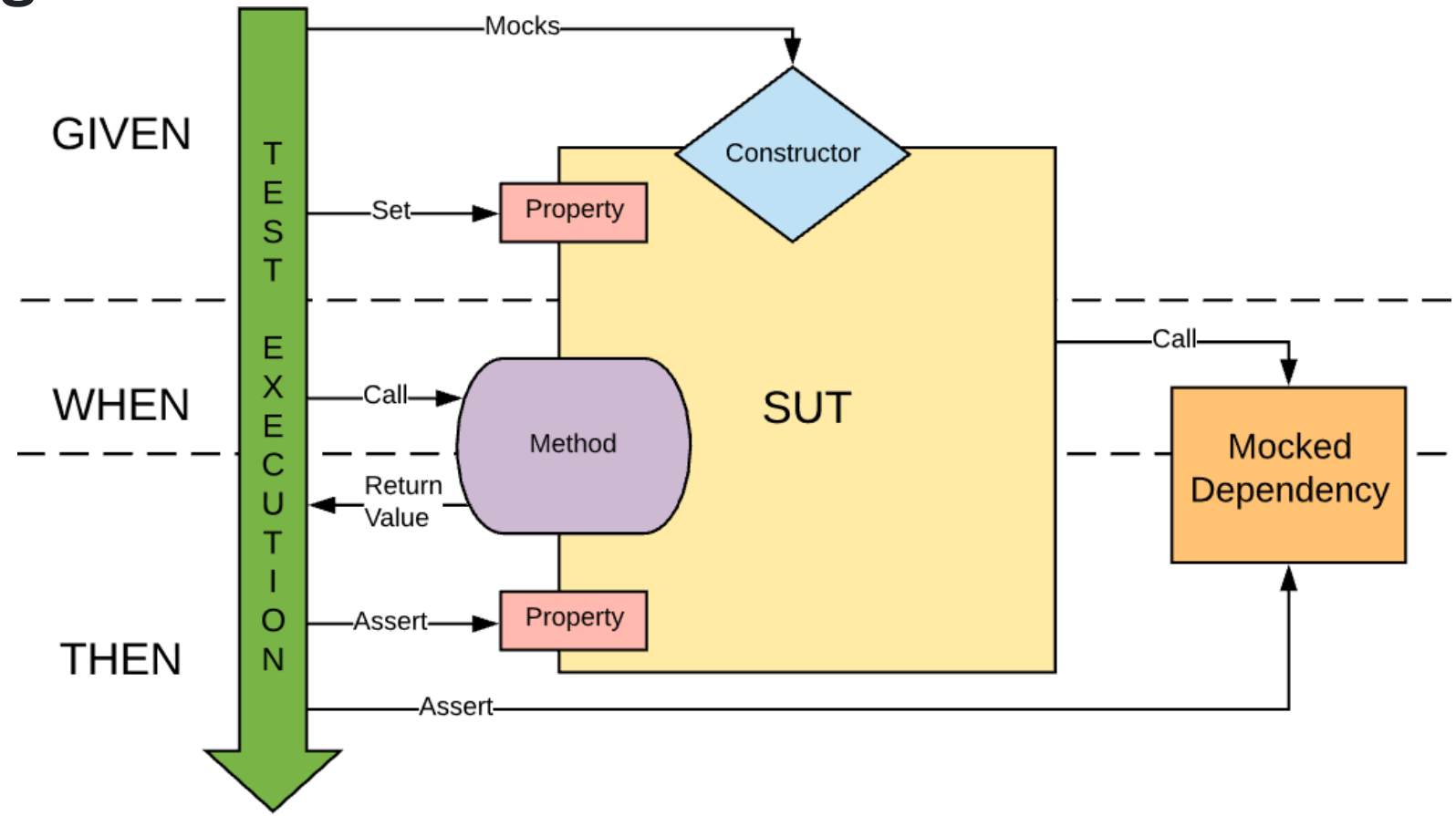
- Unit Tests
- Integration Tests
- Penetration Tests
- System Tests
- Fuzzing

Unit Tests

- Testen einzelner Softwarekomponenten
- Ziel: Sicherstellen, dass jede Komponente isoliert korrekt funktioniert
- Wichtig für Datenschutz: Vermeidung unsicherer Funktionen/Klassen in Modulen

> *„Software unit testing is a process that includes the performance of test planning, the acquisition of a test set, and the measurement of a test unit against its requirements.“* Quelle: IEEE Standard for Software Unit Testing

Unit Tests



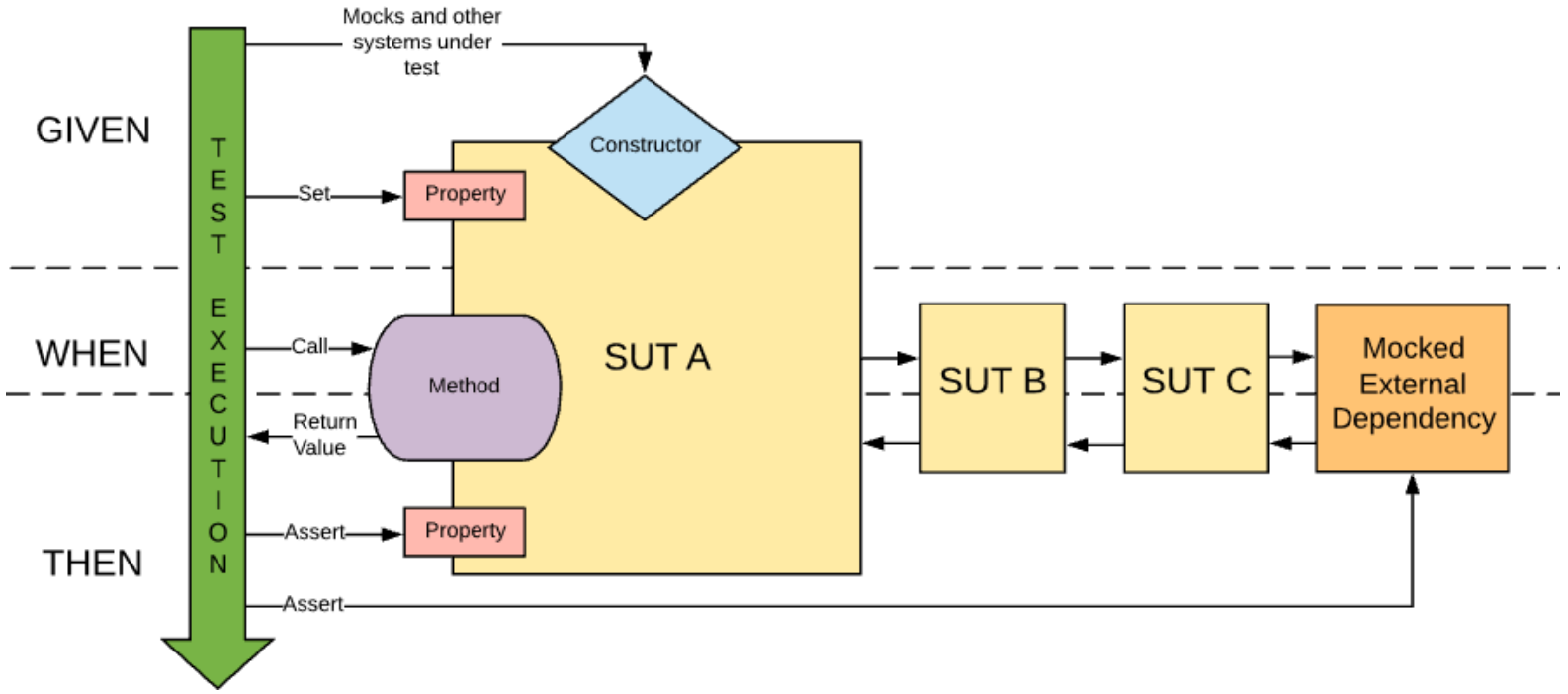
Quelle: https://dancerscode.com/posts/unit-tests/

Integration Tests

- Testen das Zusammenspiel mehrerer Komponenten
- Ziel: Sicherstellen, dass die Schnittstellen und Datenflüsse korrekt funktionieren
- Im Kontext von Datensicherheit/-schutz: Prüfen, ob sensible Daten korrekt zwischen Modulen übermittelt werden

> *„Software Integration V&V ensures that software components are validated as they are incrementally integrated“* Quelle: IEEE/ISO/IEC International Standard - Software and systems engineering--Software testing--Part 4: Test techniques,

Integration Tests



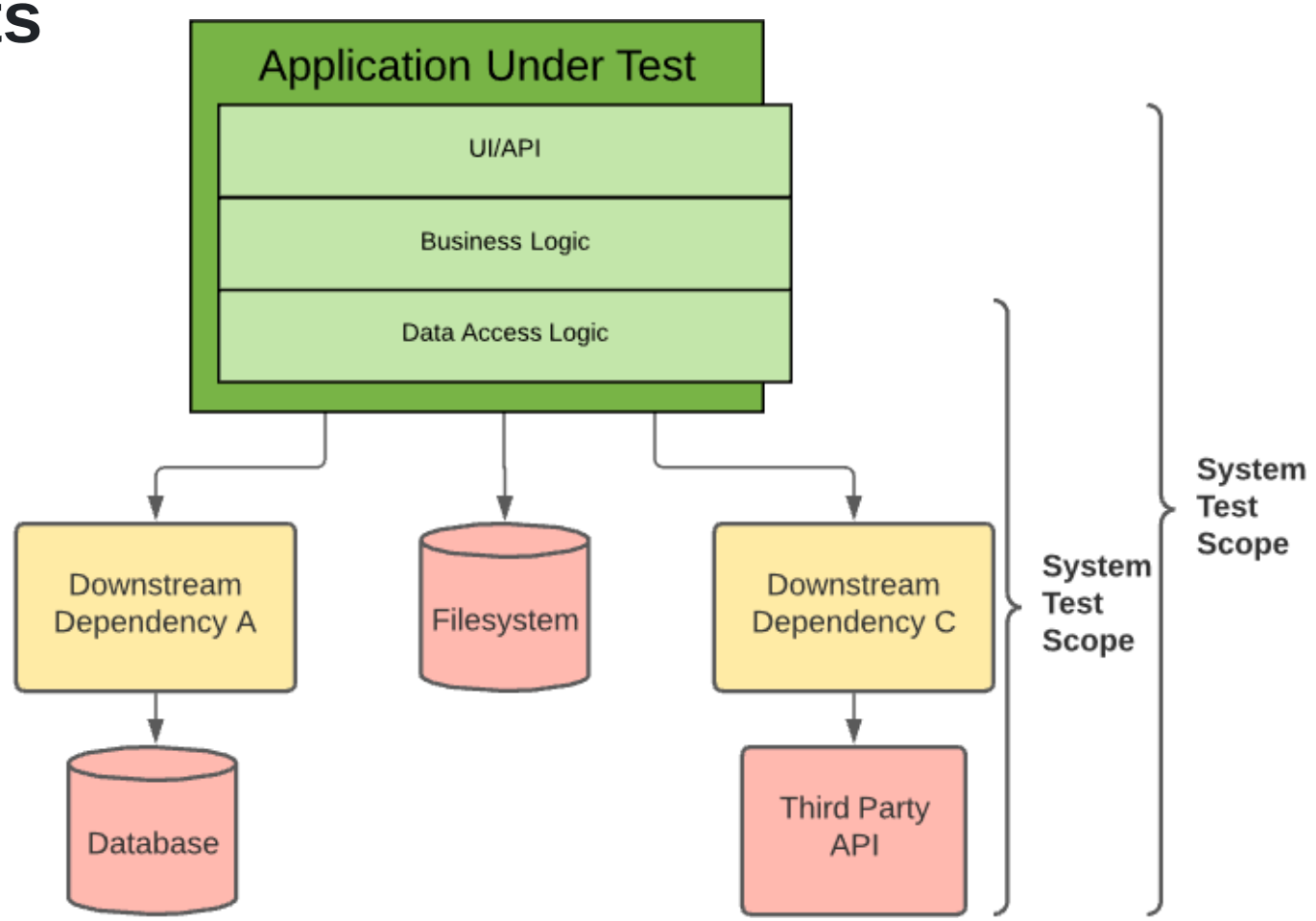
Quelle: https://dancerscode.com/posts/integration-tests/

System Tests

- Testen das gesamte System als Ganzes
- Ziel: Sicherstellen, dass die Software als Gesamtsystem funktioniert und sicher ist
- Relevant für Datenschutz: Überprüfung des gesamten Datenflusses und der Einhaltung von Sicherheitsstandards.

> *„The objective of the system testing is to find defects in features of the system compared to the way it has been defined in the software system requirements.“*
>
> Quelle: ISO/IEC/IEEE International Standard - Software and systems engineering —Software testing —Part 1:Concepts and definitions

System Tests

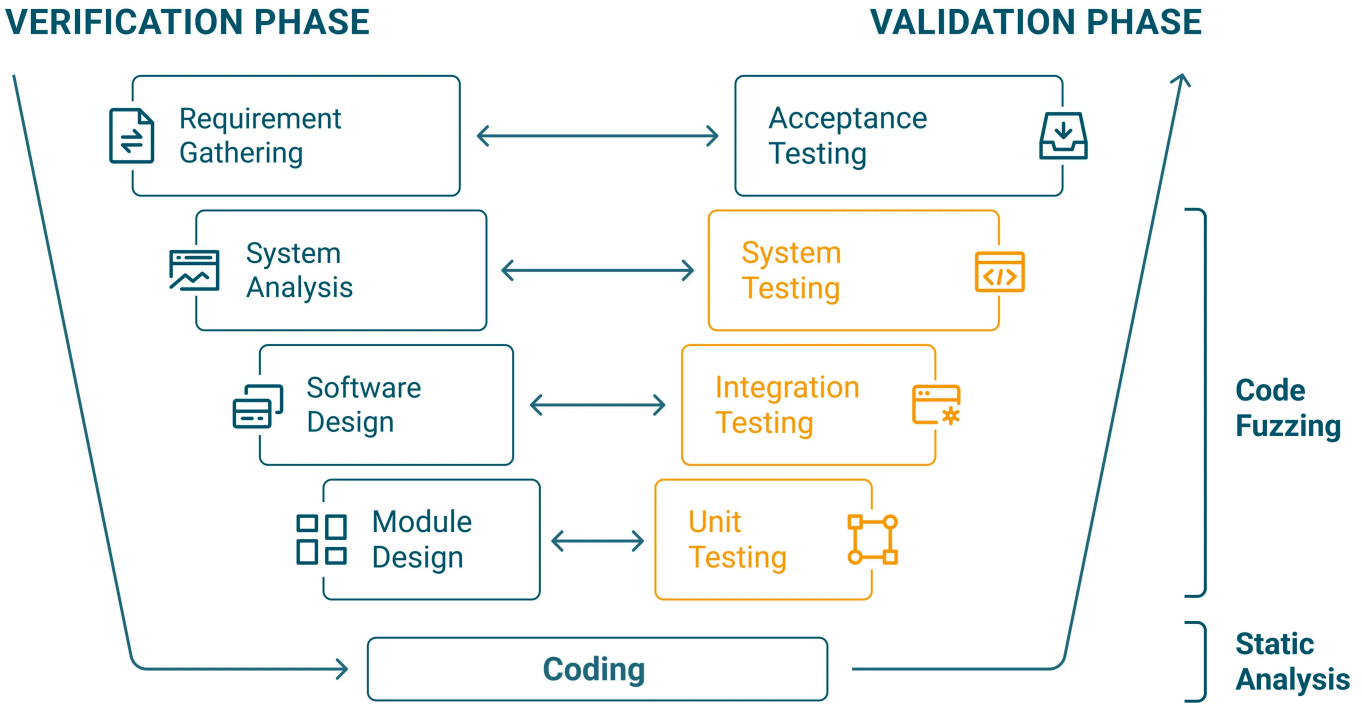


Quelle: https://dancerscode.com/posts/system-testing/

Fuzz Tests - Fuzzing

- Testmethode, bei der zufällige Daten an das System gesendet werden
- Ziel: Entdecken von Schwachstellen durch ungewöhnliche oder unerwartete Eingaben.
- Datenschutzrelevant: Aufdecken von Schwachstellen, die zu unbefugtem Zugriff auf personenbezogene Daten führen könnten.

Fuzz Tests - Fuzzing



Static Analysis and Code Fuzzing in the V-model
Quelle: https://dancerscode.com/posts/system-testing/

Fuzz Tests - Fuzzing

Arten von Fuzz-Tests:

- **Dumb Fuzzers**: Generieren zufällige Eingaben
- **Smart Fuzzers**: Erzeugen gezielte Eingaben
- **Mutationsbasiert**: Verändern bestehende Eingaben in semivalide Varianten
- **Generationsbasiert**: Erzeugt eingaben aus bekannten Strukturen
- **Black-Box**: Kein Wissen über die Programminterne Struktur
- **White-Box**: Kennt die Programmstruktur
- **Gray-Box**: Teilweise Kenntnis der Struktur
- **Abdeckungsgesteuert**: Optimiert Mutationen für maximale Codeabdeckung

Penetration Tests

- Simulierte Angriffe auf ein System, um Sicherheitslücken zu identifizieren
- Auch bekannt als *ethical hacking*
- Besonders wichtig für Datenschutz und Datensicherheit, um Schwachstellen frühzeitig zu erkennen
- Verschiedene Arten von Penetration Tests:
  - White Box
  - Black Box
  - Grey Box

Penetration Tests

Unterschiedliche Angriffsvektoren:

- Network
- Web Application
- Client Side
- Wireless
- Social Engineering
- Physical Penetration Testing

Penetration Tests

Phasen eines Penetration Tests:

1. Reconnaissance (Information Gathering)
2. Scanning (z.B. Port-Scanning)
3. Exploitation (Ausnutzung gefundener Schwachstellen)
4. Post-Exploitation (z.B. Aufrechterhaltung des Zugangs)
5. Reporting (Erstellung eines Berichts mit Empfehlungen)

Überblick: weitere Testverfahren

- **Regressionstests**: Prüfen auf neue Fehler nach Code-Änderungen
- **Load Tests**: Testen, ob das System unter Last stabil bleibt
- **End-to-End Tests**: Prüfen des gesamten System-Workflows
- **Smoke Tests**: Schnelltests nach einem Build zur Grundfunktionsprüfung
- **Sanity Tests**: Prüfen neuer Änderungen auf Korrektheit
- **Acceptance Tests (UAT)**: Überprüft Nutzeranforderungen
- **Performance Tests**: Misst Reaktionszeit und Stabilität
- **Usability Tests**: Überprüfung der Benutzerfreundlichkeit der Software
- **Alpha/Beta Tests**: Frühe/späte Testphasen mit internen/externen Nutzern

Mocking

- Mocking: Simuliert das Verhalten von Objekten oder Komponenten
- Ziel: Unabhängig von externen Abhängigkeiten testen
- Verwendung:
  - Imitiert APIs, Datenbanken oder andere Dienste
  - Ermöglicht gezieltes Testen von isolierten Funktionen
- Vorteile:
  - Schneller und zuverlässiger als echte externe Ressourcen
  - Erlaubt das Testen von Szenarien, die in der realen Umgebung schwer zu reproduzieren sind

Umsetzung von Tests

Testdaten und Datenschutz

Erforderlich bei Tests von Software, die personenbezogene Daten verarbeitet:

- **Testdaten anonymisieren** oder pseudonymisieren, um DSGVO-Anforderungen zu erfüllen.
- **Automatisierte Tools** nutzen, um sichere und DSGVO-konforme Testdaten zu generieren.
- **Zugriffsbeschränkungen** für Testumgebungen einführen, um Missbrauch von Testdaten zu verhindern.

> *„Personenbezogene Daten müssen dem Zweck angemessen und erheblich sowie auf das für die Zwecke der Verarbeitung notwendige Maß beschränkt sein („Datenminimierung“);“* Quelle: Artikel 5 Abs. 1(c) DSGVO

Und wie erstelle ich Tests?

Beispiel: Addierer

```python
# adder.py
def add(a, b):
    # Addiert zwei Zahlen
    return a + b
```

Und wie erstelle ich Tests?
Dazugehöriger Test

```python
# test_adder.py    # Bei online IDEs: main.py
import unittest # Importiere das Testframework
from adder import add # Importiere die Funktion add aus adder.py

class TestAdder(unittest.TestCase):
    def test_add_small_positiv_numbers(self): # Testfall 1
        self.assertEqual(add(1, 2), 3) # Erwartetes Ergebnis: 3

    def test_add_small_negativ_numbers(self): # Testfall 2
      self.assertEqual(add(-1, -2), -3) # Erwartetes Ergebnis: -3

if __name__ == '__main__':
  unittest.main() # Starte die Testausführung
```

Zum ausprobieren z.B. auf: onlinegdb.com

Ehmm.. Noch Fragen oder Anmerkungen?

Link zu den Aufgaben

Gehe zu den Aufgaben

Quellen

- https://www.geeksforgeeks.org/software-testing-basics/
- https://blogs.zeiss.com/digital-innovation/de/test-driven-development/
- https://purplesec.us/learn/types-penetration-testing
- https://www.code-intelligence.com/what-is-fuzz-testing
- https://microsoft.github.io/code-with-engineering-playbook/automated-testing/unit-testing/mocking/