Vorschlag eines Systems für Laborübungen für die Ausbildung mit neuronalen Netzwerken

von Anton Klüver und Rico Ukro

TODO

- [ ] Einpacken und Schleife drum machen

Agenda

- Aufgabenstellung

Aufgabenstellung

Implementierung von GPU-Workload

Schlagen Sie ein System für Laborübungen für die Ausbildung mit neuronalen Netzwerken vor, mit folgenden Anforderungen:

- 2 Seminargruppen á 25 Studenten gleichzeitig, Jupyter
- Ressourcen pro Student
  - 16GB Arbeitsspeicher
  - 4 CPU-Cores
  - 8GB GPU-Speicher
  - 250 GB HDD / SSD
- Nutzungshorizont: 5 Jahre

5x

https://www.ebay.de/itm/166955662024

Menge | Produkt | Preis | Link

- | - | - | -
  5 | NVIDIA Tesla H100 80GB SXM5 PCIE | 20.480,00 | Ebay (10.04.20205)

Start of old slides as example

Agenda

- Begriffsklärung und Bedeutung von Softwaretests
- Zweck und Notwendigkeit von Softwaretests
- Grundlagen der Testmethoden und -verfahren
- Durchführung von Softwaretests
- Offene Fragen und Diskussion
- Übungen

Begriffsklärung und Bedeutung von Softwaretests

> *„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

Zweck und Notwendigkeit von Softwaretests

Fallstudien: Historische 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

Fallstudien: Historische 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 an Softwaretests (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?

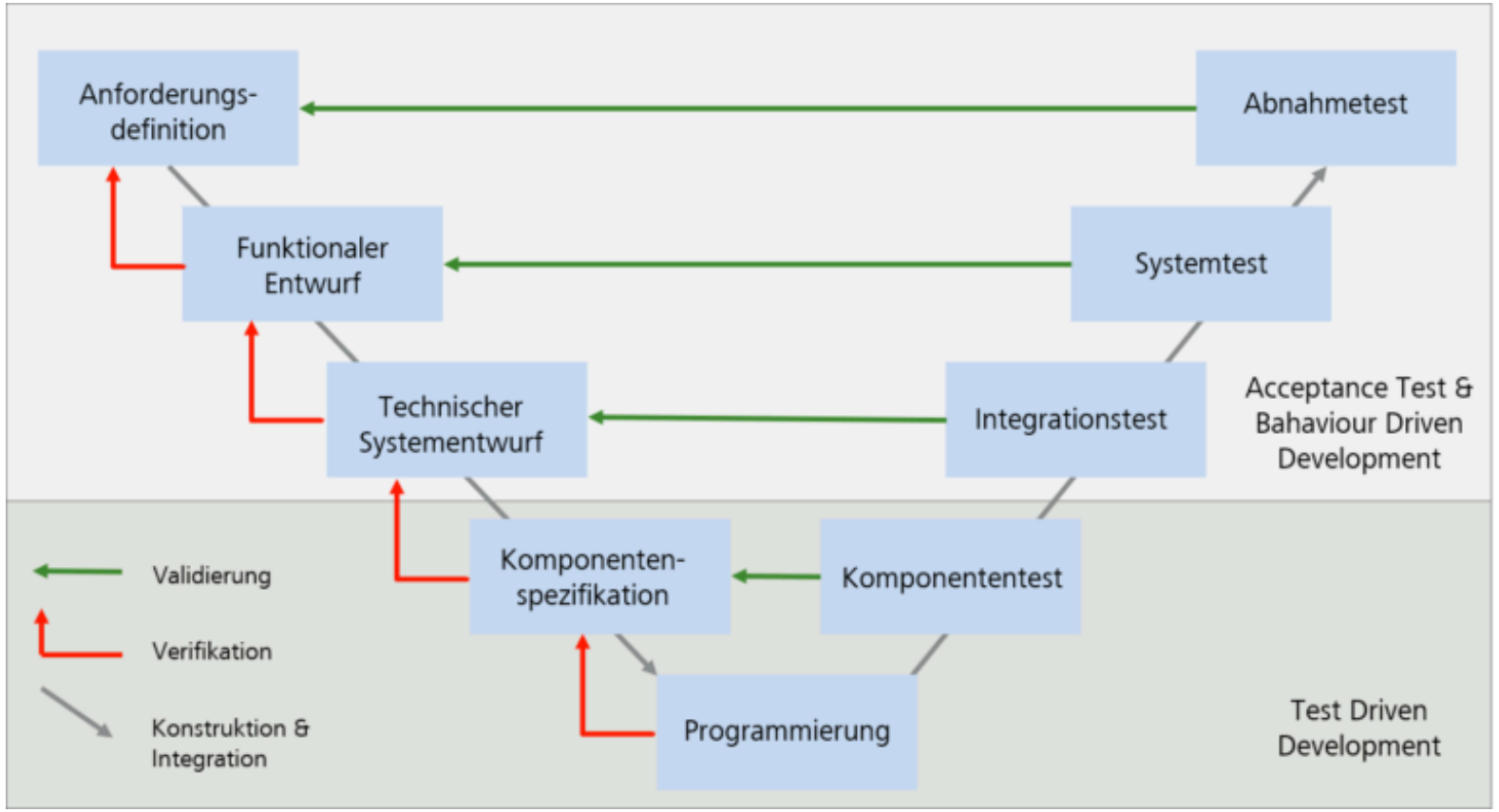
- Fehlererkennung: API antwortet korrekt, Fehler werden richtig gehandhabt
- Fehlertoleranz: Anwendung reagiert robust auf verschiedene Antwortcodes und Netzwerkausfälle
- Sicherheit: Sicherstellen, dass keine sensiblen Daten kompromittiert werden
- Zuverlässigkeit: API-Abfrage funktioniert konsistent,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

Grundlagen der Testmethoden und -verfahren

Testgetriebene Entwicklung (TDD - Test Driven Development)

- 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

Testgetriebene Entwicklung (TDD - Test Driven Development)



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

Übersicht über Testverfahren

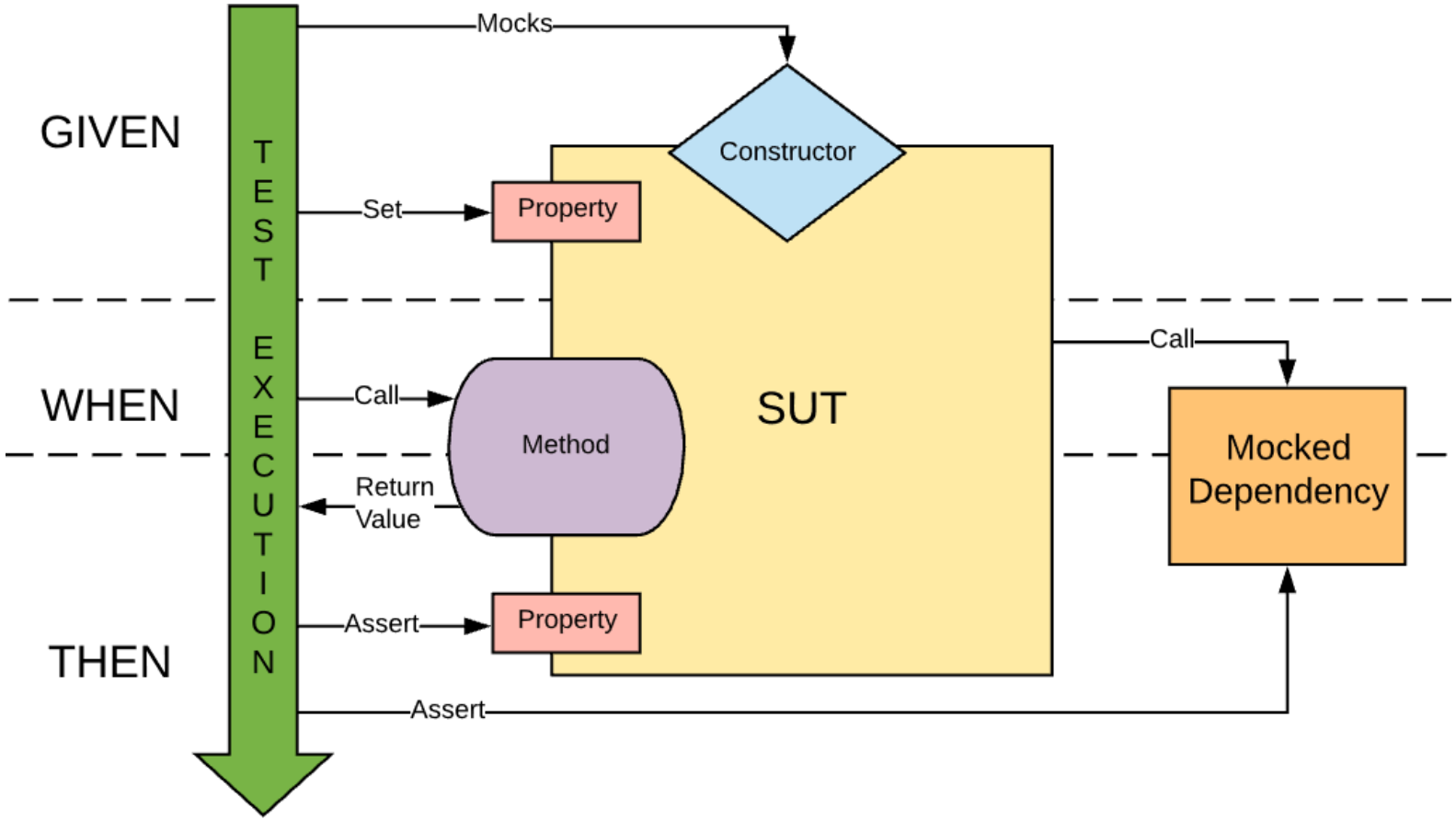
- Unit-Tests
- Integrationstests
- Systemtests
- Fuzz-Tests - Fuzzing
- Penetrationstests
- ...

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



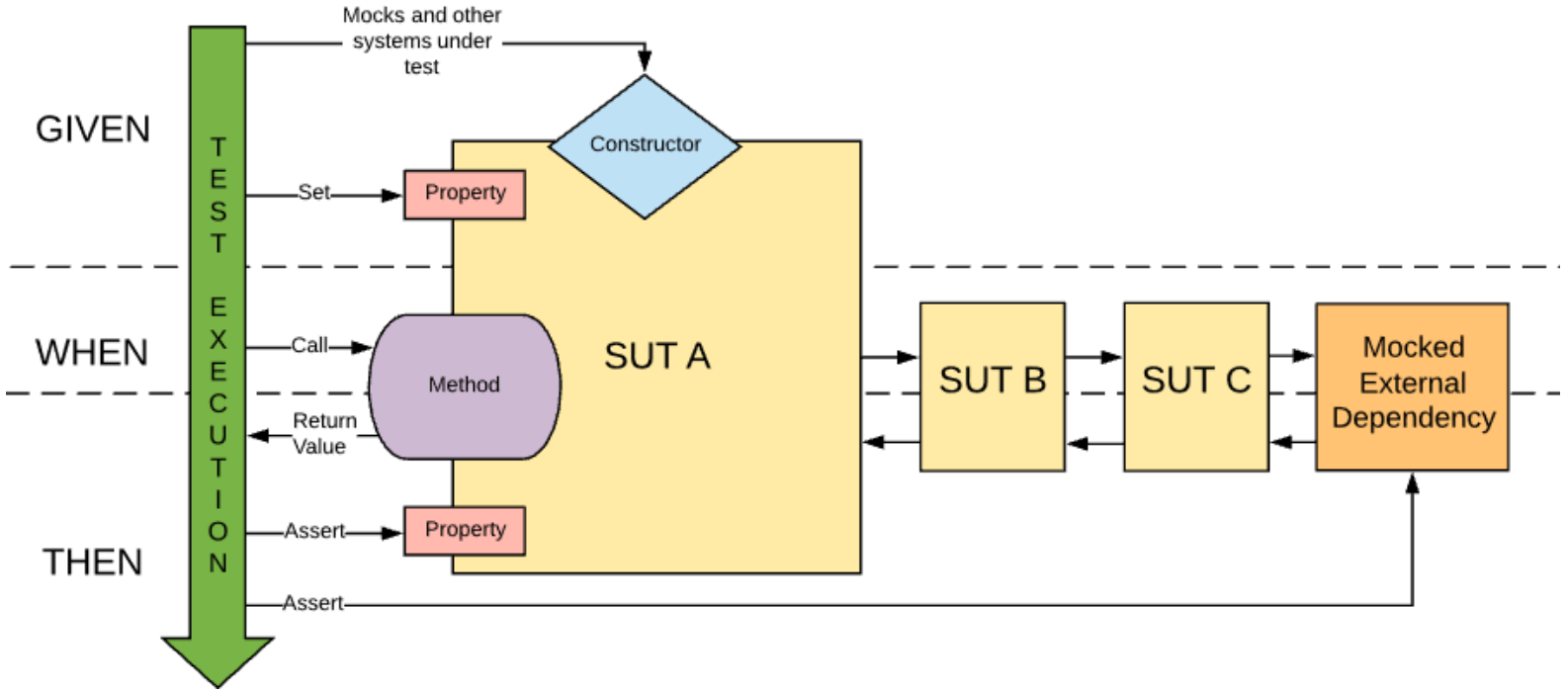
Quelle: https://dancerscode.com/posts/unit-tests/

Integrationstests

- 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,

Integrationstests



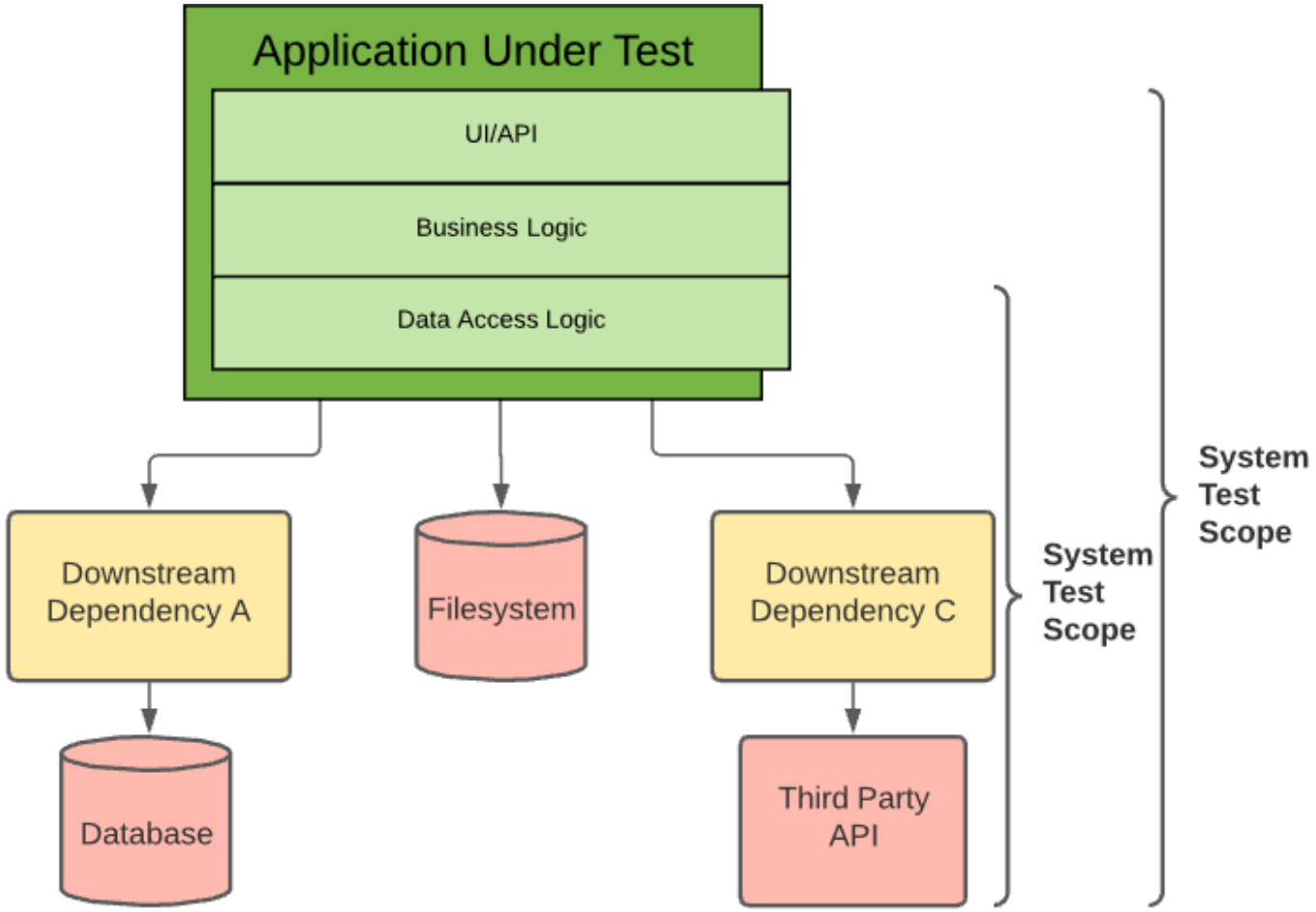
Quelle: https://dancerscode.com/posts/integration-tests/

Systemtests

- 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

Systemtests

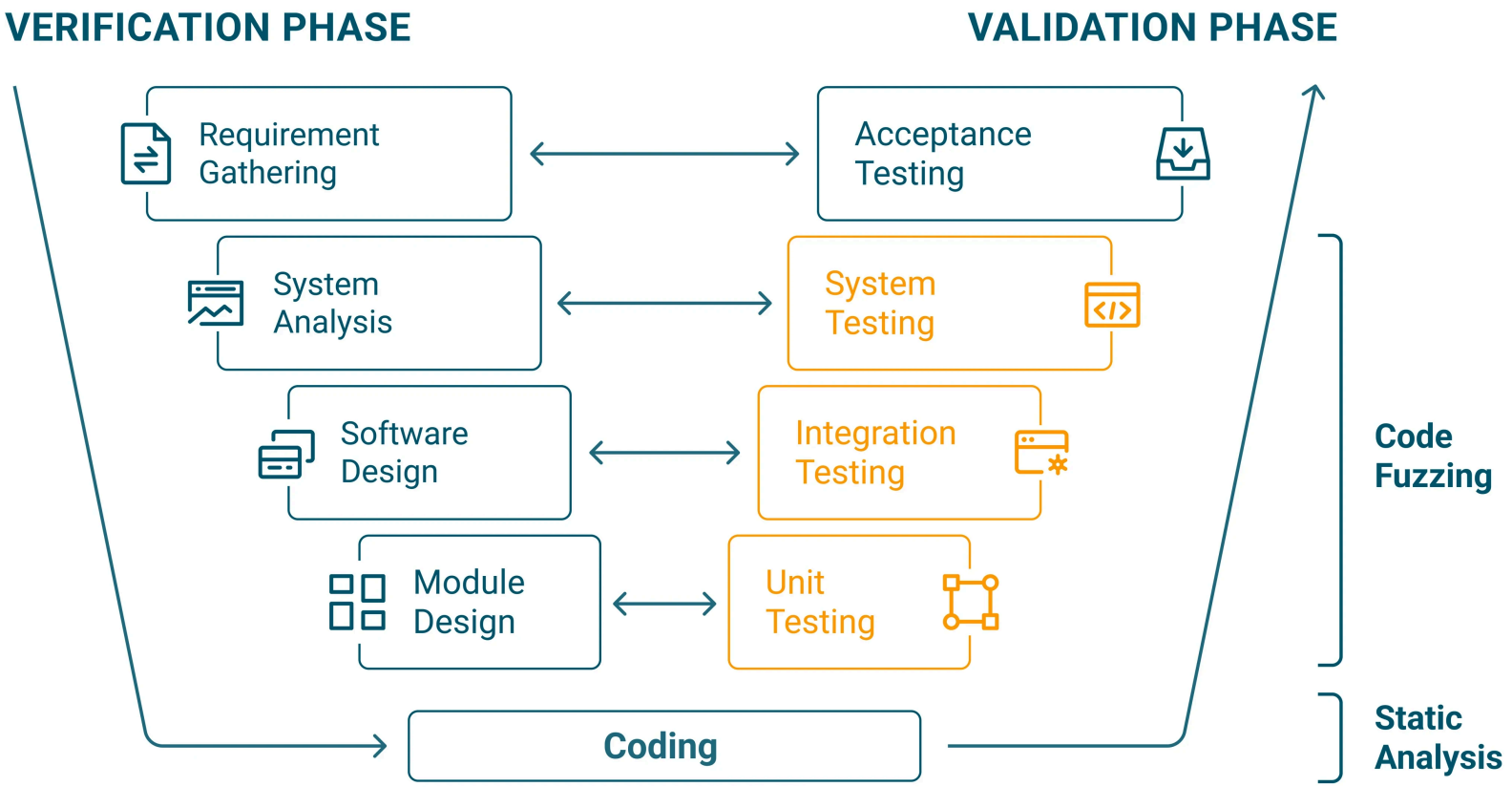


Quelle: https://dancerscode.com/posts/system-testing/

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



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

Penetrationstests

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

Penetrationstests

Unterschiedliche Angriffsvektoren:

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

Penetrationstests

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)

Weitere Testverfahren im Überblick

- **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 - Simulation externer Abhängigkeiten

- 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

Effizienzsteigerung in der Softwareentwicklung

Rolle der Testautomatisierung in einer CI/CD-Pipeline

- **Continuous Integration (CI)**:
  - Automatisiert Tests bei jedem Integrationsschritt
  - Früherkennung von Bugs
  - Führt Unit-Tests, Integrationstests und andere automatisierte Tests aus
- **Continuous Delivery/Deployment (CD)**:
  - Stellt sicher, dass jeder Build produktionsbereit ist
  - Automatisiert Regressionstests zur Sicherstellung der Stabilität
  - Reduziert manuellen Eingriff bei der Bereitstellung

Rolle der Testautomatisierung in einer CI/CD-Pipeline

- **Vorteile**:
  - Schnelleres Feedback zur Code-Qualität
  - Reduziert menschliche Fehler durch Automatisierung
  - Beschleunigt Entwicklungs- und Release-Zyklen
  - Verbessert Softwarequalität und Zuverlässigkeit

Nutzenanalyse von Tests (Test Benefit Analysis)

- **Fehlererkennung:**
  - Frühzeitige Erkennung von Fehlern reduziert spätere Korrekturkosten
  - Höhere Testabdeckung minimiert das Risiko von unentdeckten Fehlern
- **Qualitätssteigerung:**
  - Regelmäßige Tests verbessern die Code-Qualität und Systemstabilität
  - Sicherstellung, dass neue Features keine alten Funktionen beeinträchtigen

Nutzenanalyse von Tests (Test Benefit Analysis)

- **Effizienz:**
  - Automatisierte Tests beschleunigen Entwicklungsprozesse
  - Spart Zeit und Ressourcen durch frühzeitige Validierung
- **Vertrauen in Software:**
  - Gut getestete Software schafft Vertrauen bei Entwicklern, Testern und Nutzern
  - Höhere Zuverlässigkeit und geringeres Risiko von Produktionsausfällen

Durchführung von Softwaretests

Testdatenmanagement und Datenschutzkonformität

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

Entwicklung und Implementierung von Tests

Beispiel: Addierer

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

Entwicklung und Implementierung von 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

Offene Fragen und Diskussion

Lernkontrolle und Wiederholung mit praktischen Übungen zur Implementierung von Tests

Zu den Übungsfragen und Aufgaben

Quellen

1. Myers, Glenford J., Corey Sandler, and Tom Badgett. *The Art of Software Testing*. John Wiley & Sons, 2011.
2. Pajankar, Ashwin. *Python Unit Test Automation: Automate, Organize, and Execute Unit Tests in Python*. Apress, 2022.
3. GeeksforGeeks. „Software Testing Basics.“ *GeeksforGeeks*, https://www.geeksforgeeks.org/software-testing-basics/. Abgerufen am 15. Sep. 2024.
4. Zeiss. „Test Driven Development (TDD).“ *Digital Innovation Blog*. Zeiss, https://blogs.zeiss.com/digital-innovation/de/test-driven-development/. Abgerufen am 18. Sep. 2024.
5. Purplesec. „Types of Penetration Testing.“ *PurpleSec*, https://purplesec.us/learn/types-penetration-testing/. Abgerufen am 19. Sep. 2024.
6. Microsoft. *Unit Testing: Mocking in Automated Testing*. Code With Engineering Playbook, https://microsoft.github.io/code-with-engineering-playbook/automated-testing/unit-testing/mocking/. Abgerufen am 17. Sep. 2024.
7. Code Intelligence. „What is Fuzz Testing?“ *Code Intelligence*, https://www.code-intelligence.com/what-is-fuzz-testing. Abgerufen am 20. Sep. 2024.
8. Dancer´s Code „Role of Test Automation in a CI/CD Pipeline“ *Dancer´s Code*, https://dancerscode.com/posts/role-of-test-automation-in-a-ci-cd-pipeline/. Abgerufen am 24. Sep. 2024.
9. Dancer´s Code „Test Benefit Analysis“ *Dancer´s Code*, https://dancerscode.com/posts/test-benefit-analysis/. Abgerufen am 24. Sep. 2024.