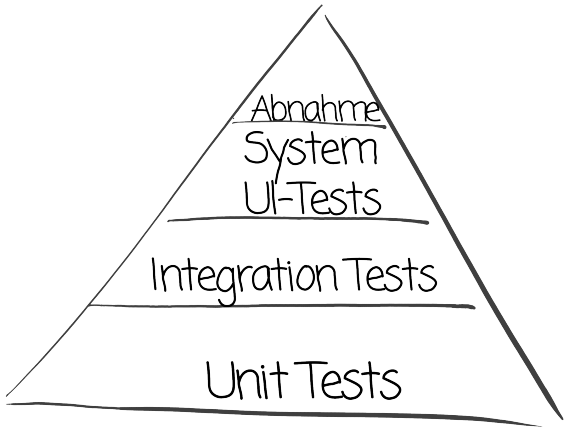
T 03 Modul 450: Teststufen

Softwaretests sind ein unverzichtbarer Bestandteil des Entwicklungsprozesses moderner Anwendungen. Sie helfen, Fehler frühzeitig zu erkennen, die Qualität von Systemen sicherzustellen und Risiken zu minimieren.

Inhaltsverzeichnis

1. Teststufen: Hierarchische Ansätze

1.1. Testpyramide

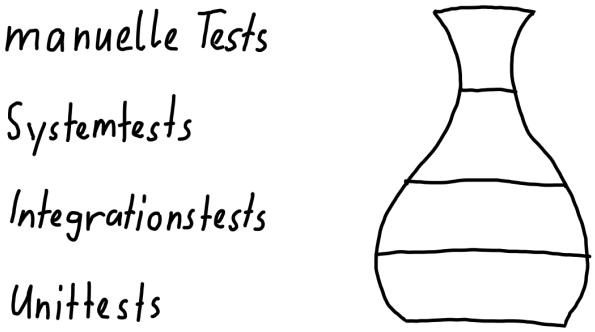


Die Testpyramide ist ein bewährtes Modell, das eine klare Struktur für die Priorisierung von Tests vorgibt. Sie basiert auf der Idee, dass

- **Unittests** die Basis bilden und am häufigsten ausgeführt werden sollten, da sie schnell, günstig und isoliert durchführbar sind.
- **Integrationstests** die Zusammenarbeit zwischen Komponenten überprüfen und weniger häufig als Unittests ausgeführt werden.
- **Systemtests** an der Spitze stehen, da sie aufwendig, langsam und teuer sind, jedoch das gesamte System als Ganzes testen.

Dieses Modell visualisiert die Effizienz und Effektivität verschiedener Testebenen und wird anhand eines anschaulichen Schaubilds präsentiert.

1.2. Testvase



Die Testvase steht im Kontrast zur klassischen Testpyramide und reflektiert häufige realistische Situationen in der Softwareentwicklung. Hier liegt der Fokus verstärkt auf höherstufigen Tests wie Systemtests, während Unittests und Integrationstests weniger stark berücksichtigt werden. Diese "verformte" Pyramidenform (die an eine Vase erinnert) verdeutlicht, dass in der Praxis oft mehr Zeit und Ressourcen in komplexere Tests investiert werden, obwohl diese kostspieliger und weniger effizient sind.

2. Testarten im Detail

Für jede Testart werden Ziele, Methoden, Vor- und Nachteile ausführlich beschrieben, um die jeweiligen Einsatzgebiete und Grenzen zu verstehen.

2.1. Unittests

Definition: Testen einzelner, isolierter Einheiten wie Klassen, Methoden oder Funktionen.

Ziele: Sicherstellen, dass die kleinsten Bausteine eines Systems korrekt funktionieren.

Vorteile:

- Schnell: Tests laufen direkt im Arbeitsspeicher, ohne externe Abhängigkeiten.
- Einfach zu kontrollieren: Der Test überprüft spezifische Eingaben und Ausgaben.
- Einfach zu schreiben: Kein Aufbau komplexer Testumgebungen nötig.

Nachteile:

- Unrealistisch: Fehler, die durch Interaktionen entstehen, werden nicht erkannt.
- Unvollständig: Tests berücksichtigen nicht alle Abhängigkeiten.

2.2. Integrationstests

Definition: Testen der Interaktion zwischen mehreren Komponenten eines Systems.

Ziele: Sicherstellen, dass Komponenten korrekt miteinander kommunizieren und Daten austauschen.

Vorteile:

- Realistisch: Simuliert reale Bedingungen, in denen Komponenten zusammenarbeiten.
- Vollständig: Erfasst mögliche Fehler an Schnittstellen.

Nachteile:

- Aufwändig: Testumgebungen und Abhängigkeiten müssen vorbereitet werden.
- Langsam: Zugriffe auf externe Systeme (z. B. Datenbanken oder APIs) dauern länger.
- Zerbrechlich: Kleine Fehler in einer Komponente können viele Tests fehlschlagen lassen.

2.3. Systemtests

Definition: Überprüfung des gesamten Systems unter produktionsähnlichen Bedingungen.

Ziele: Sicherstellen, dass alle Komponenten als Ganzes korrekt funktionieren.

Vorteile:

- Vollständig: Deckt das gesamte System ab.
- Aussagekräftig: Ergebnisse repräsentieren die reale Benutzererfahrung.

Nachteile:

- Langsam: Das Aufsetzen und Starten von Systemtests dauert lange.
- Schwierig: Zusätzlicher Code und Aufwand sind nötig, um das System zu testen.
- Unzuverlässig: Künstliche Testumgebungen können instabil sein.

2.4. Manuelle Tests

Definition: Tests, die von Menschen durchgeführt werden, um Systeme auf bestimmte Aspekte wie Benutzerfreundlichkeit oder spezielle Szenarien zu überprüfen.

Vorteile:

- Menschliche Intuition: Entdeckung von Details, die Maschinen übersehen könnten.
- Fachwissen: Tester können sich auf geschäftskritische Aspekte konzentrieren.

Nachteile:

- Langsam: Manuelle Tests benötigen Zeit und Aufmerksamkeit.
- Fehleranfällig: Menschen machen eher Fehler als automatisierte Prozesse.

3. Weitere Testarten / Spezialmethoden

Neben den klassischen Testarten gibt es weitere spezialisierte Ansätze:

- **End-to-End-Tests**: Simulation kompletter Nutzerabläufe.
- **Usability-Tests**: Fokus auf Benutzerführung und Barrierefreiheit.
- **Penetrationstests**: Sicherheitsprüfungen zur Erkennung von Schwachstellen.
- **Lasttests**: Testen des Systemverhaltens unter hoher Last.
- **Benchmarks**: Messungen der Systemleistung unter vergleichbaren Bedingungen.
- **Fuzzying**: Einsatz zufälliger Daten zur Erkennung unerwarteten Verhaltens.
- **Property-based Tests**: Prüfung allgemeiner Eigenschaften wie mathematische Gesetzmässigkeiten.