

Dominik Meyer

Software Engineering

## Software Testing



WEITER WISSEN →

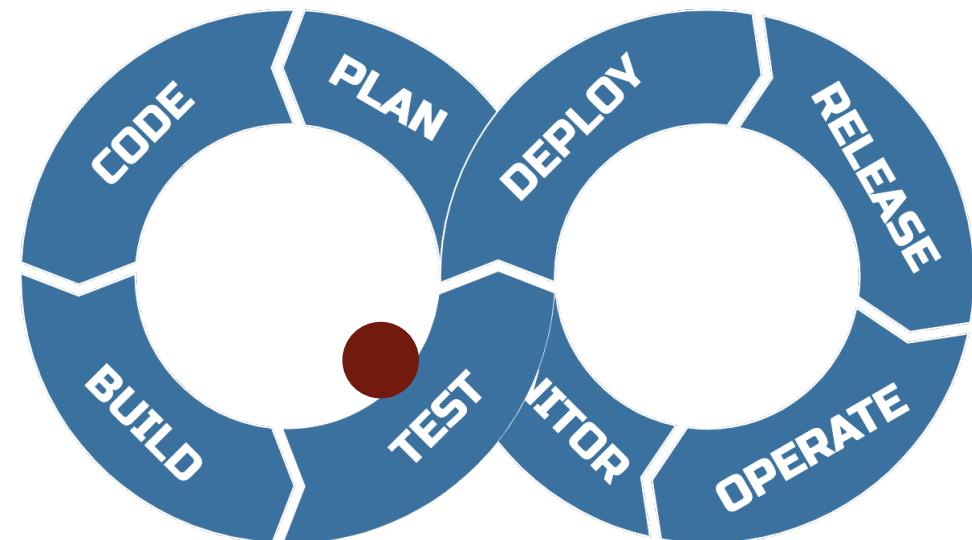
# Ziel

Nach der Lektion wenden die Studierenden verschiedene Test Frameworks korrekt an.

Nach der Lektion beschreiben die Studierenden die verschiedenen Testlevel und deren Vernetzung mit dem Software Development Lifecycle SDLC.

# Software Testing

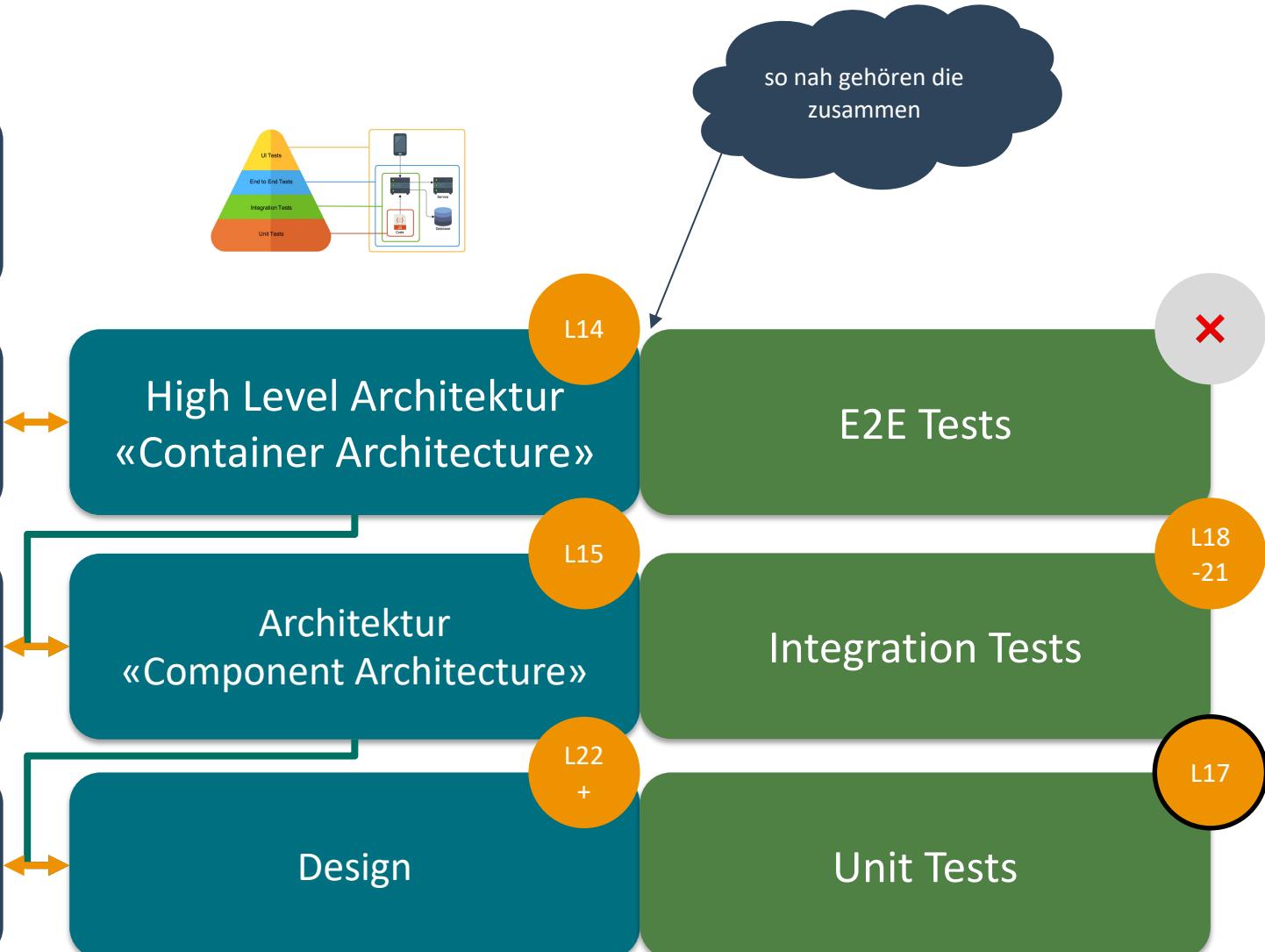
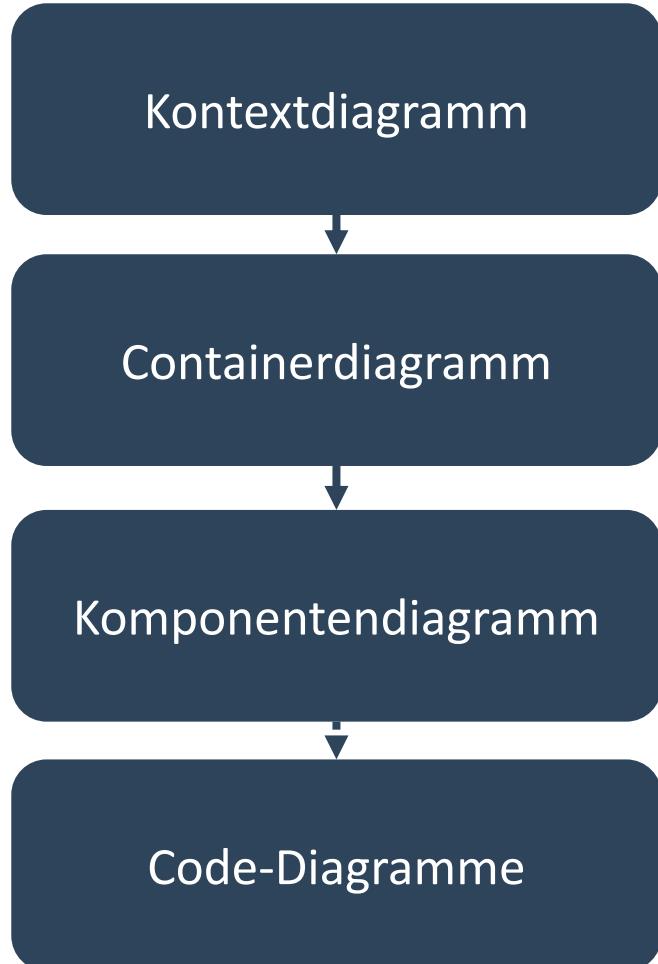
- Verschiedene Flughöhen
  - Unit Tests und deren Java Framework JUnit
  - GitHub Actions um Tests zu Automatisieren
- 
- Resultate der Entwicklung, Artefakte
  - Werkzeuge pro Flughöhe
  - Werkzeuge für die Modulararbeit



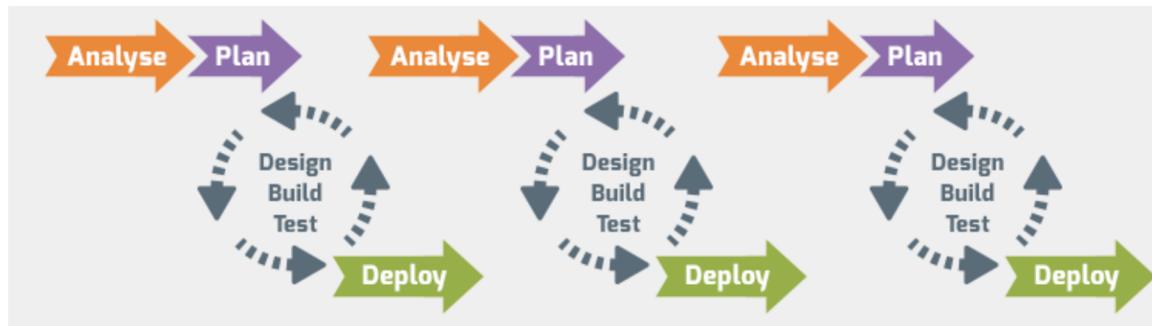
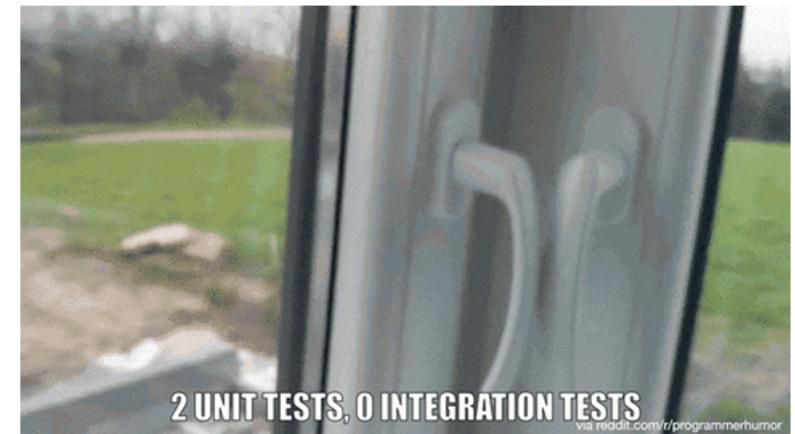
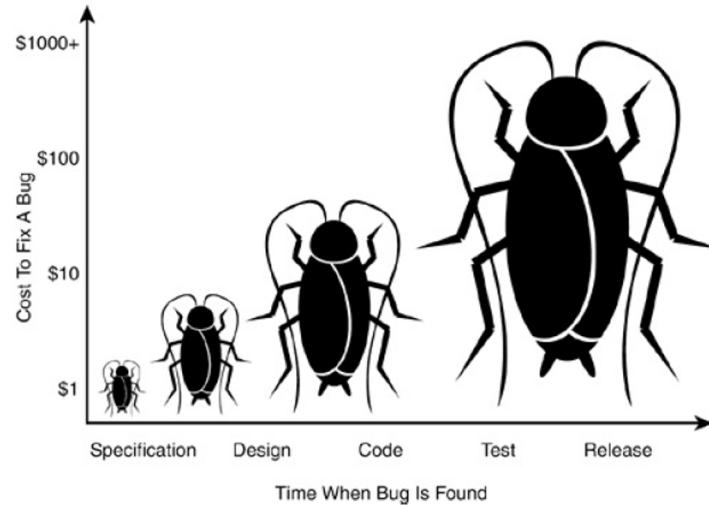
# Agenda

- Ziel
- Abholung Flughöhen
- Warum überhaupt?
- Test-«Konzepte»
- Werkzeuge pro Flughöhe
- Artefakte
- Artefakte pro Flughöhe
- Zielkontrolle

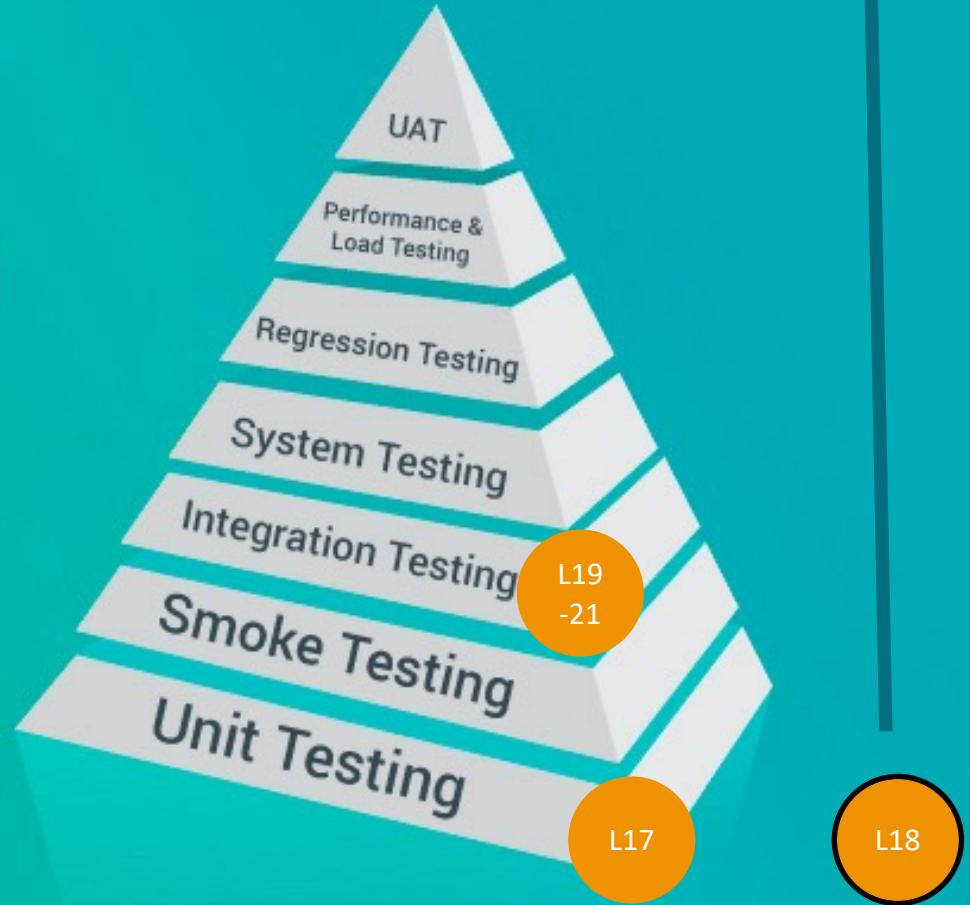
# Erinnerung Flughöhen (L14 Architektur)



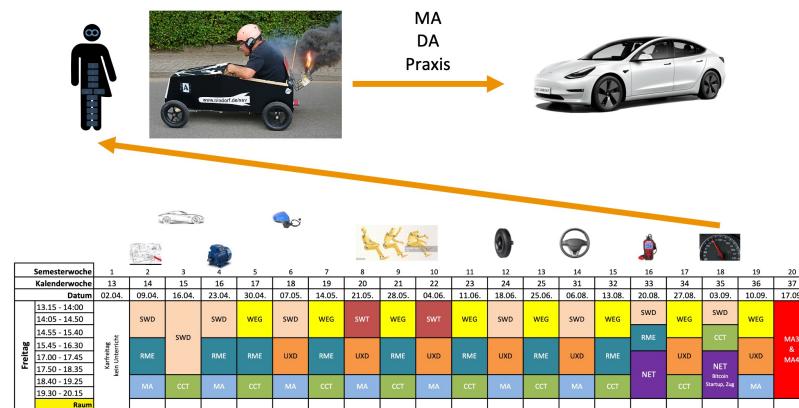
## Wozu?



# Types of **Software Testing**



# Könnte eigener Studiengang sein



**Test Policy** – High level description of principles, approach and major objectives of the organization regarding testing.

**Test Strategy** – High level description of test levels for an organization or programme, and the testing to be performed within each level.

- Wir wollen das nötigste, praxis-orientierte um zu Starten
- uns ist klar, das Testing bis zum Exzess konzipiert werden kann, z.B. mit ISO29119

# Die allerwichtigste Regel

# Alles muss «traceable» sein

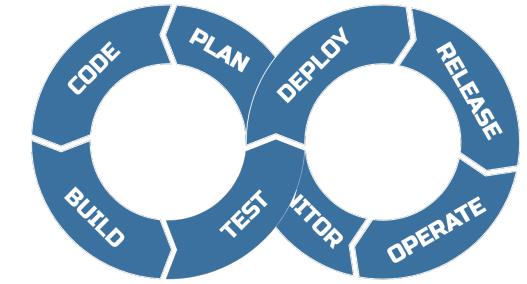
- Wann wurde, was, wie, unter welchen Bedingungen, mit welchem Resultat von wem getestet?
- Welche Anforderung deckt dieser Test (Case) ab?
- Welches Ticket hatte diese Änderung umgesetzt, welcher Commit?
- Wer hat den Test enabled/disabled?
- Für welche Releases wurde dieser Test mit welchem Resultat von wem ausgeführt?
- Welches Build-Resultat (z.B. Docker Container) wurde dafür verwendet?

Die meisten Fragen können Logs (u.a. von GHA) beantworten.

Wer nicht automatisiert muss den Preis von Hand bezahlen...

# Erstes Ziel (unter Einhaltung Regel 1): Automation

- Nur wer automatisiert kann DevOps leben, kontinuierlich releases und deployen
- Diese Vorlesung zeigt einfachste Automation vor
  - zudem sehr effektiv
- Am Anfang (oder z.B. MA und DA) tut es auch ein Mix aus Automation und manueller Arbeit
- Alles manuell → schlussendlich mehr Aufwand (denke an Tracing)
- Manuelle, lästige Tasks, DevOps «Toil»
- Dieser Ratschlag kann getrost ignoriert werden, die Rechnung folgt dann später einmal..., in MA, DA oder dem Leben 



# Pro Level

UAT

E2E Tests

Integration Tests

Unit Tests

Pro Level gibt es grundsätzlich

- Daten
- Fälle
- Bedingungen
- Frameworks oder Tools
  - u.a. Runner (wie JUnit)
- Resultate in Form von Artefakten

- Schlussendlich gibt es Entscheidungen
  - Release, kein Release
  - Deployment, kein Deployment

# Runner (Profile)

Suite

Case

Preconditions



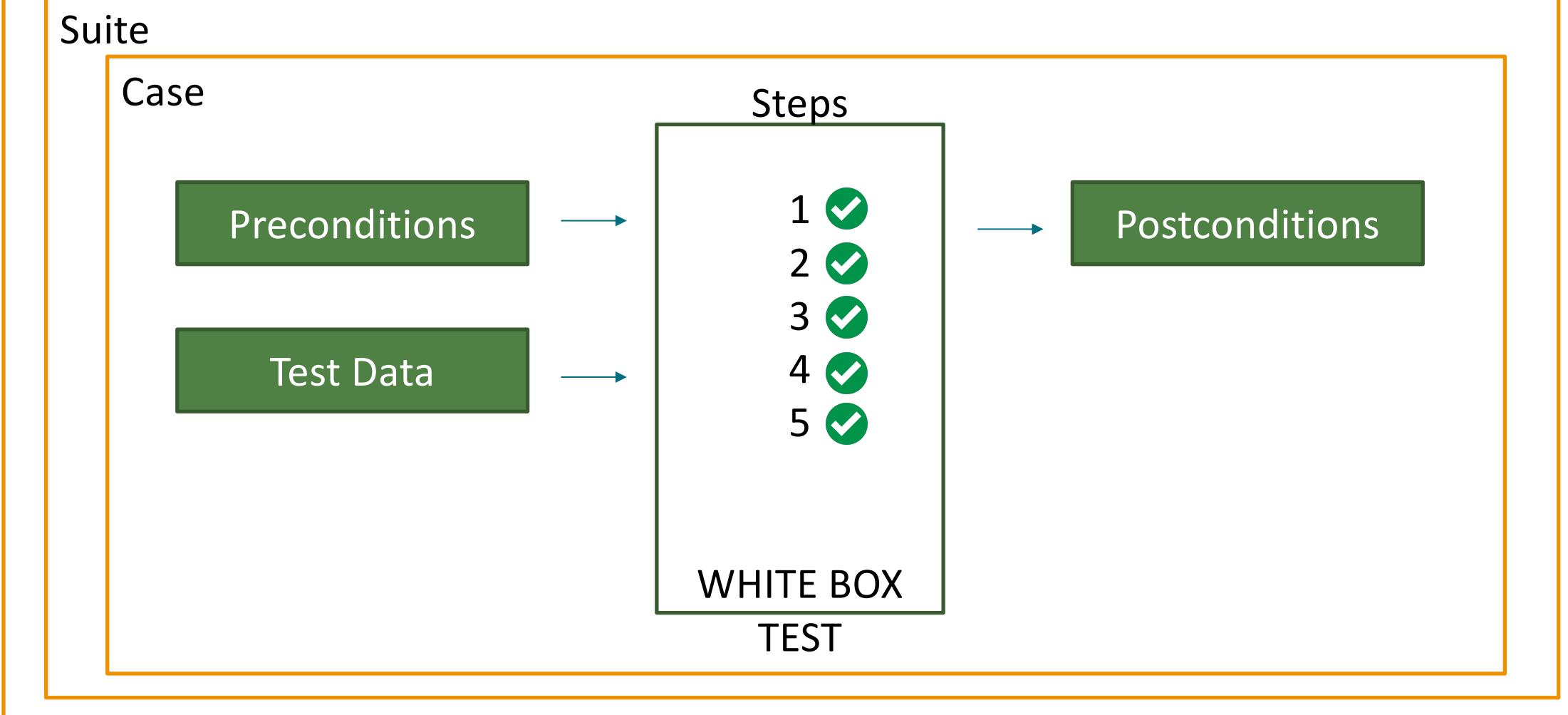
Postconditions



Test Data



# Runner (Profile)



# Sichtbarkeiten

- Black Box Testing
  - Input geben, Output checken, «Weg» egal
- White Box Testing
  - Input geben, Output checken, Weg auf Korrektheit prüfen
- Gray Box Testing
  - Beide mischen
- Ad hoc Testing
  - Testen ohne Plan und Dokumentation, was wir möglichst vermeiden wollen
- Wir nutzen für die MA primär Black Box Tests
  - Komplexität kleiner
  - Software Design für «Testability» einfacher

# Level: Unit

Hard-codiert, als Code

*Daten*

- Logs
- Coverage Report
- Success/Failure

*Artefakte*

- Fälle*
- Äquivalenzklassen
  - Rand-Fälle
  - Spezielle Kombinationen
  - Fehler-Fälle

- Tools*
- J: JUnit
  - P: PHPUnit
  - G: Go hat eigenes
  - N: Node Mocha oder Jest
  - Java Helper: Hamcrest
  - Node Helper: Sinon

# Level: Integration

Daten

- Hard-codiert, als Code
- Könnten aus einer Seed-Datenbank geladen werden
- Könnten dynamisch generiert werden (denn Unit spielt keine Rolle mehr)

Artefakte

- Logs
- Coverage Report
- Success/Failure

Fälle

- Use Cases
- Rand-Fälle
- Fehler-Fälle
- Spezielle Kombinationen

Tools

- J: Rest-Assured, Mockito  
P: 😊
- G: Gorilla Mux
- N: Node Supertest, Jest, Nock
- Java Helper: Hamcrest
- Node Helper: Sinon
- Test-Container

# Klein Anfangen

Unsere Werkzeuge

Unit Test: JUnit 5

Integration Test: Rest-Assured und Test-Containers

Ausführen: Lokal und mit GitHub Actions

Artefakte: GitHub Action Logs

Fälle:

- Happy Flow Use Cases
- Randfälle oder Äquivalenzklassen

# Fälle finden

```
public int multiply(int a, int b){  
    return a*b;  
}
```

## Äquivalenzklassen

- Gruppen für Fälle
- Bsp:
  - Zahlen um 0
  - Zahlen zwischen 0 und 1000
  - resp. -1000 und 0
  - Zahlen bis kurz vor Int.max
  - Zahlen kurz nach Int.min
  - 0
  - Int.max und Int.min

## Randfälle

- Spezielle Fälle
- Bsp:
  - 0
  - Int.max
  - Int.max-1
  - Int.min
  - Int.min-1
  - Normale Zahlen (3\*4)

Auswerten

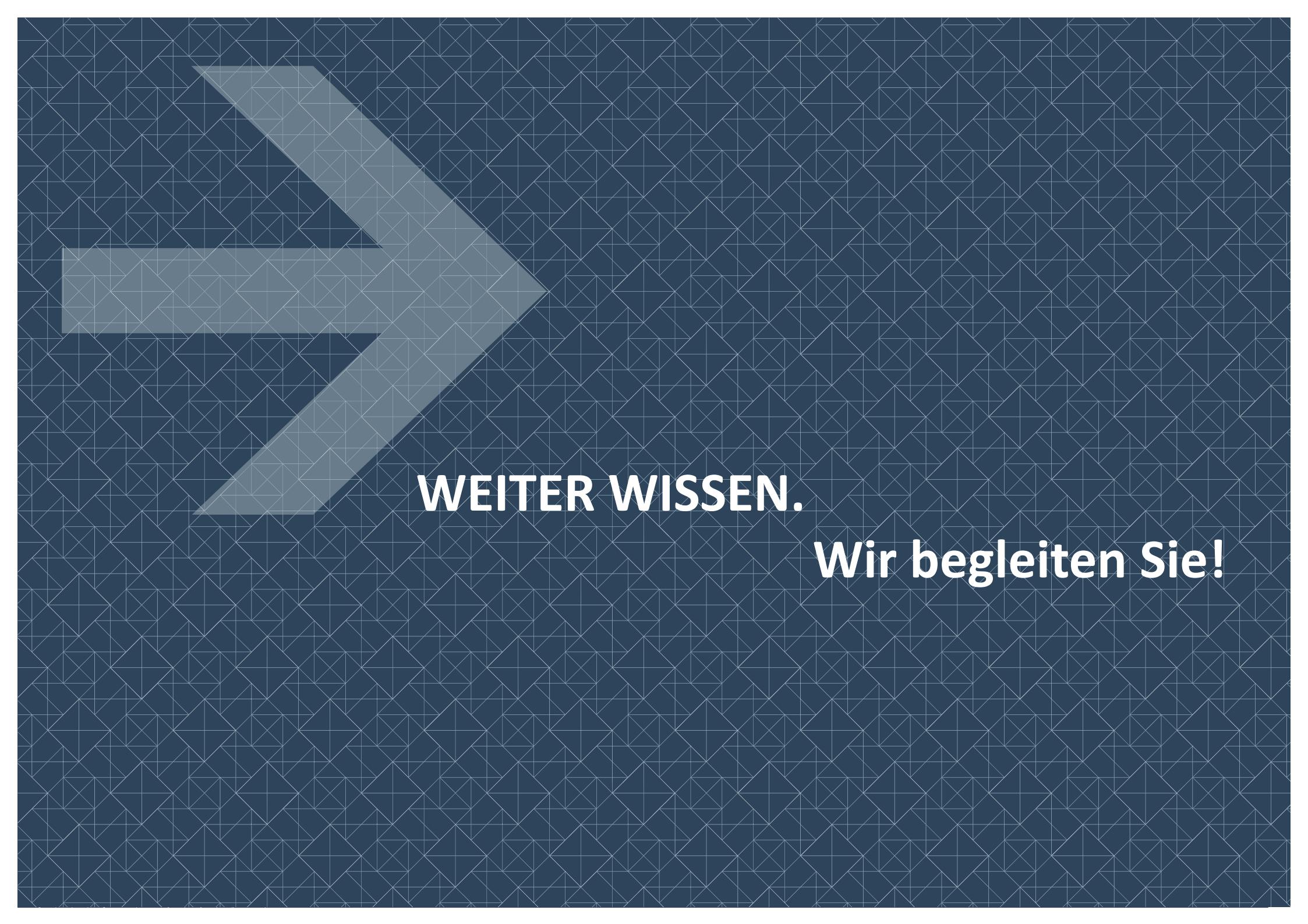
# Zielkontrolle

AUFTAG  
⏰ 5min

<https://forms.office.com/r/dB1riM6vL4>

🔒 ABB Login





**WEITER WISSEN.**

**Wir begleiten Sie!**