ZooRoyal IT

Unit Tests

Sebastian Knott

10. April 2018

Überblick

Grundlagen

- Einordnung
- Lebenszyklus Tests

Unit Tests

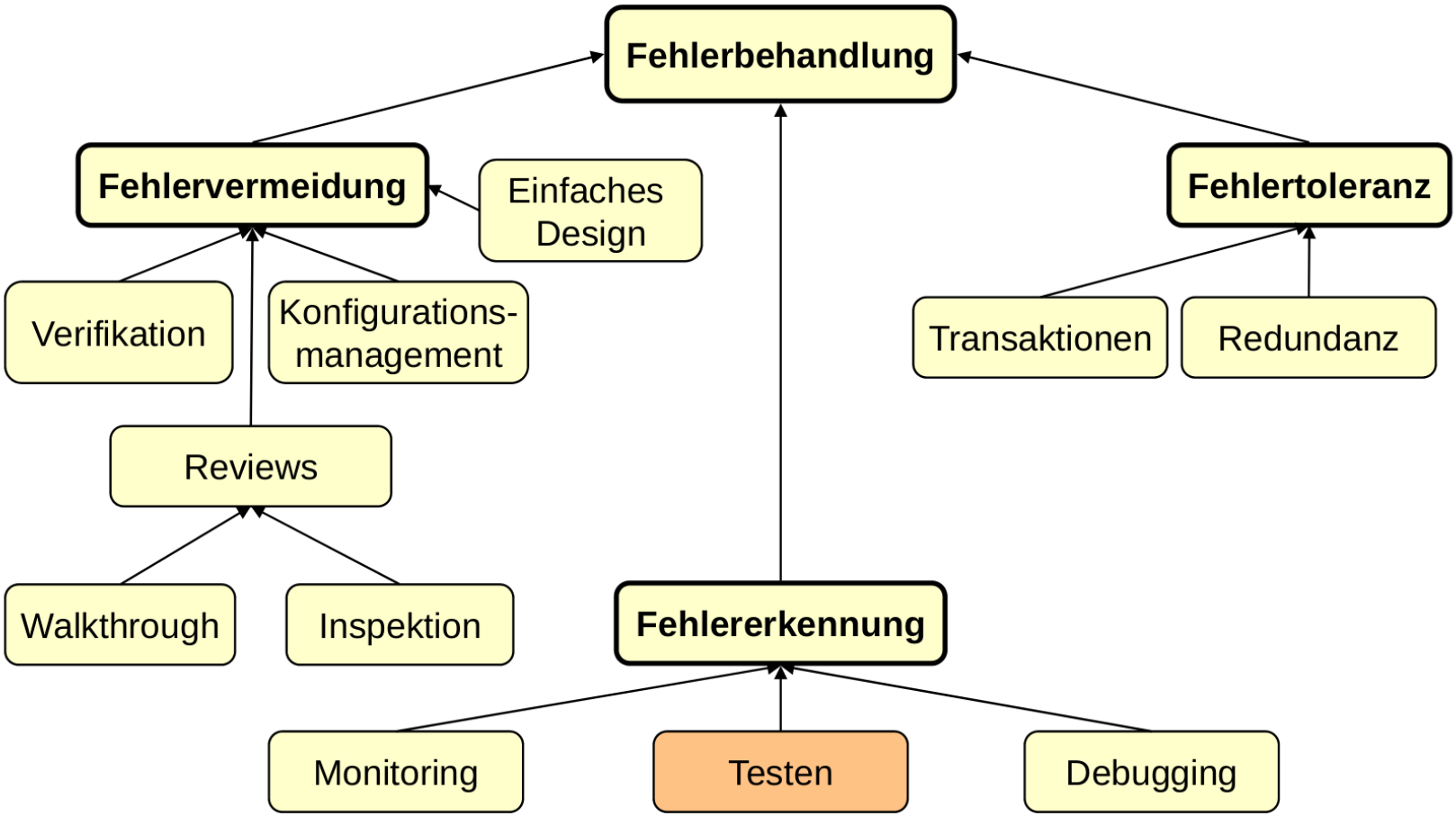
- Begriffsklärung
- Testfallauswahl

Code Dojo

- Das Dojo
- Code Kata
- Das Tennis Kata

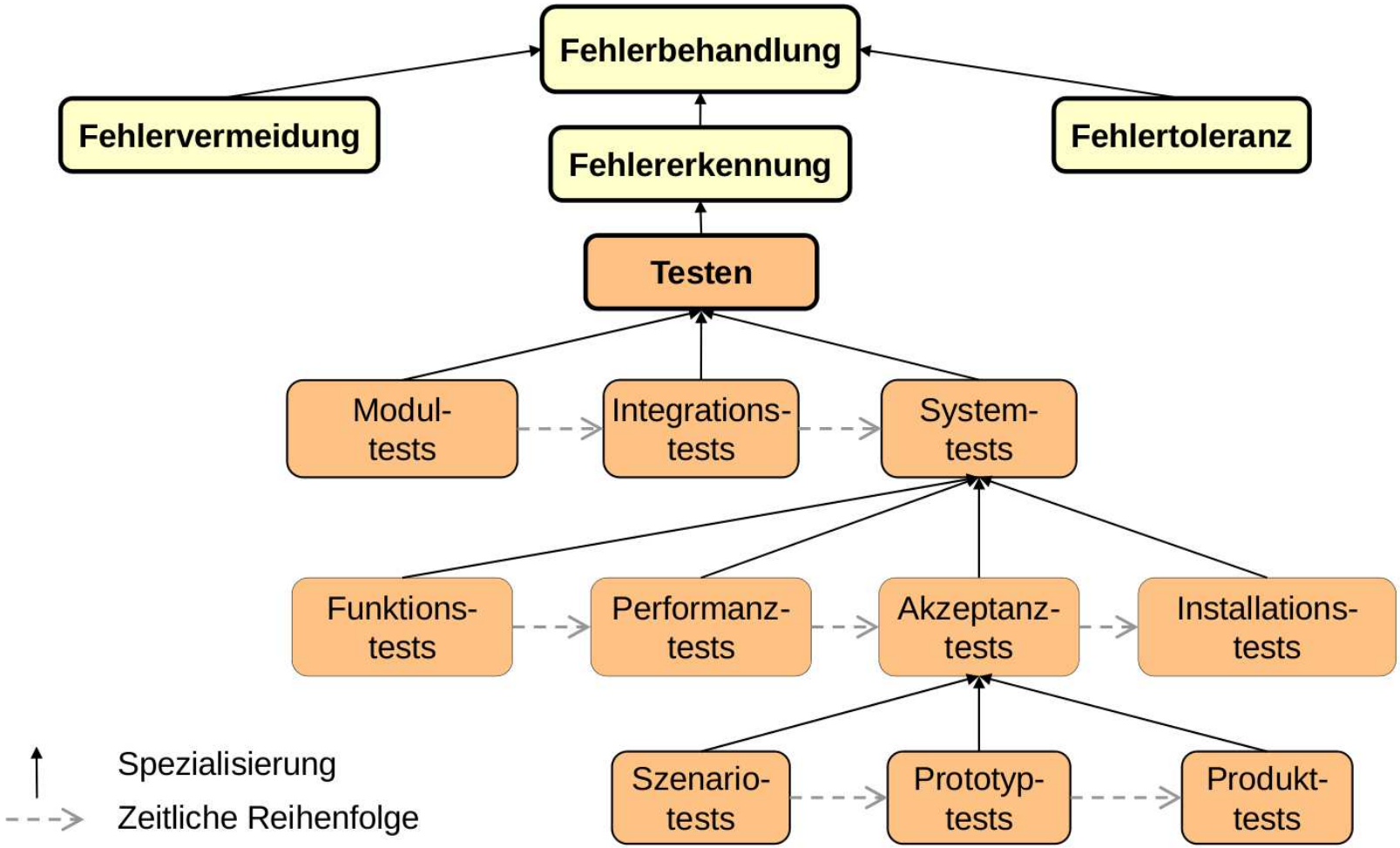
Einordnung

Übersicht Fehlerbehandlung



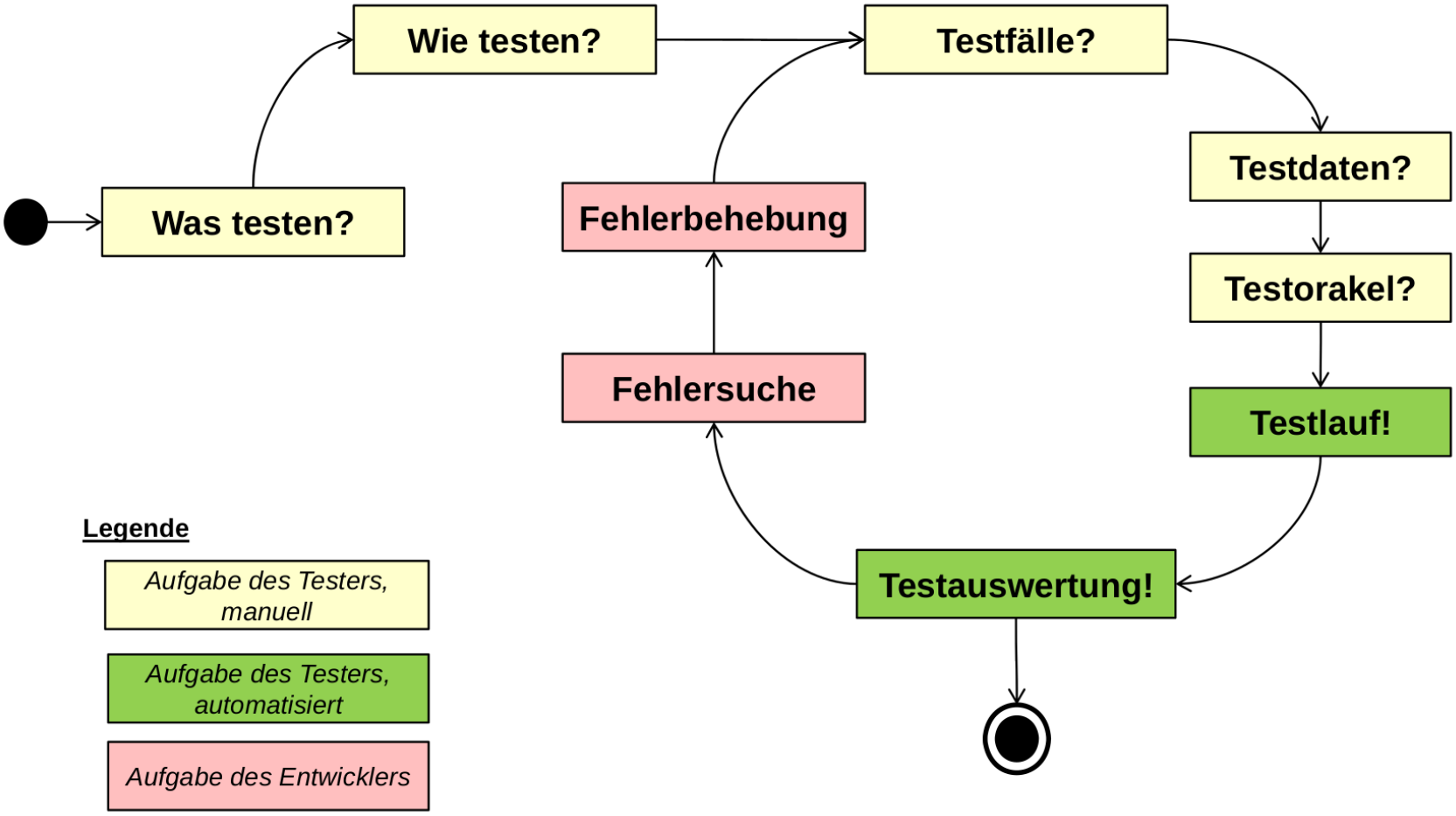
Einordnung

Übersicht Fehlererkennung

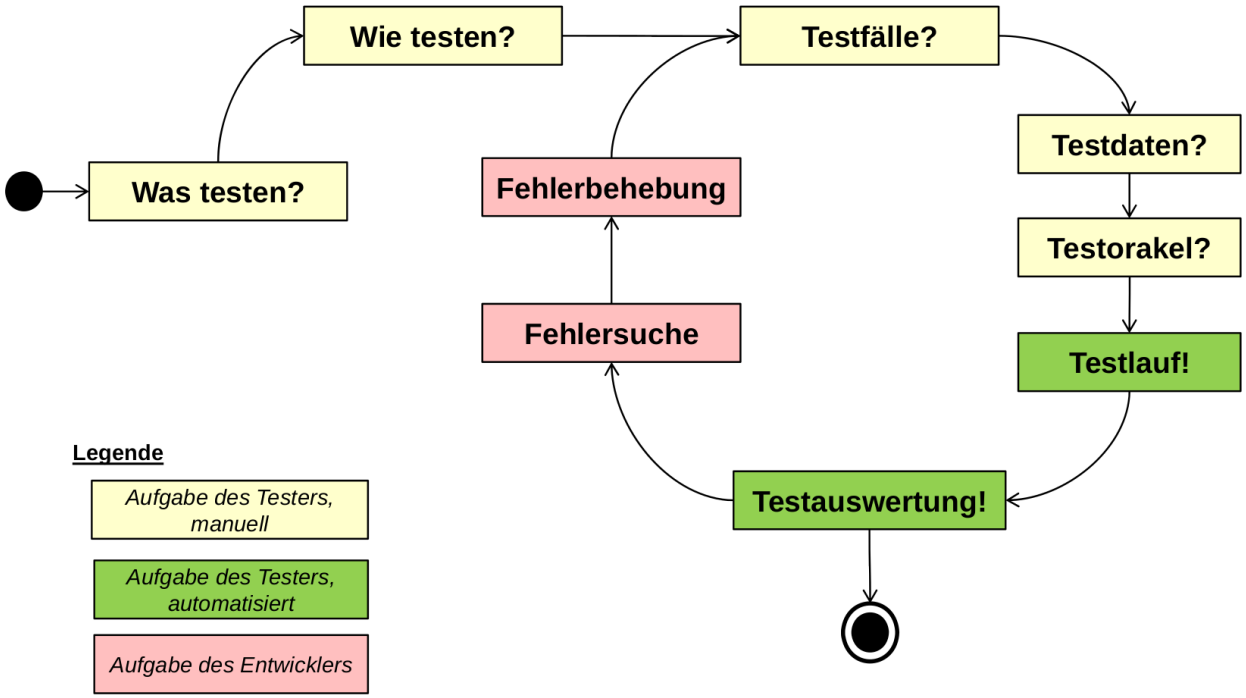


Lebenszyklus Tests

…in 9 Schritten oder weniger



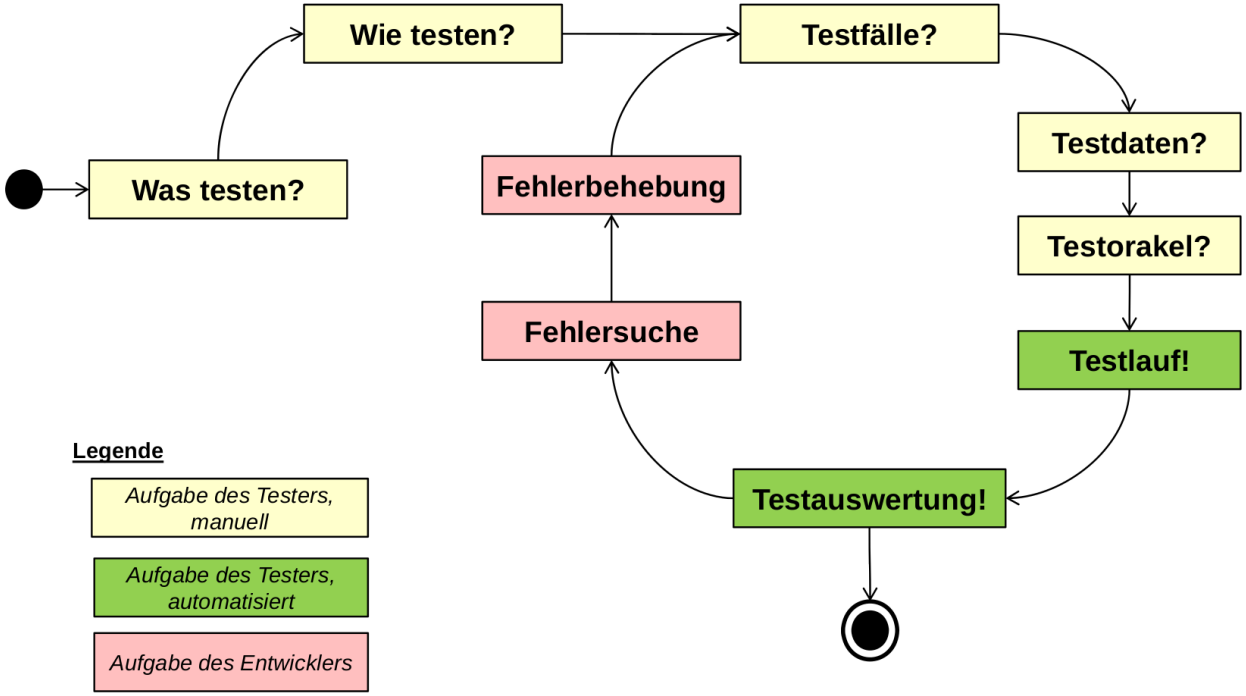
Lebenszyklus Tests
1. Was testen?



Testgegenstand: Bestimme was getestet werden soll

- Vollständigkeit der Anforderungen, …
- Testen des Codes auf Zuverlässigkeit, …

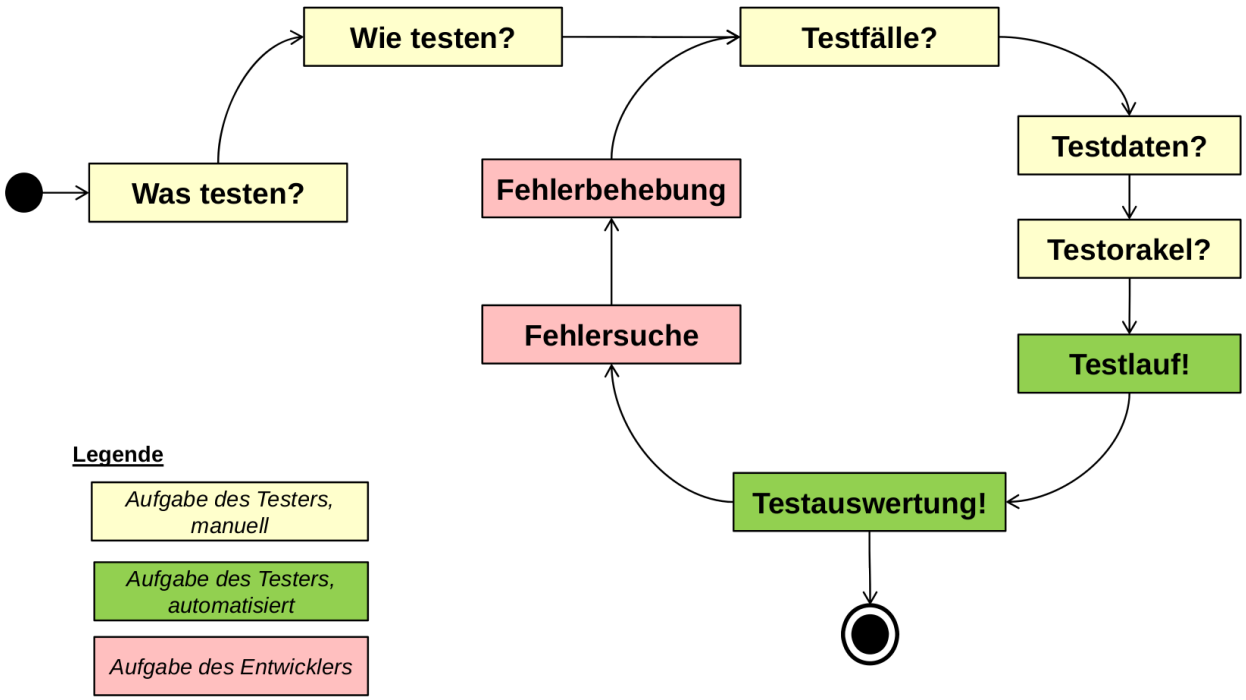
Lebenszyklus Tests
2. Wie testen?



Testverfahren: Entscheide wie getestet wird

- Inspektion des Codes, …, Beweise
- Black-box, White-box
- Strategie für Integrationstests

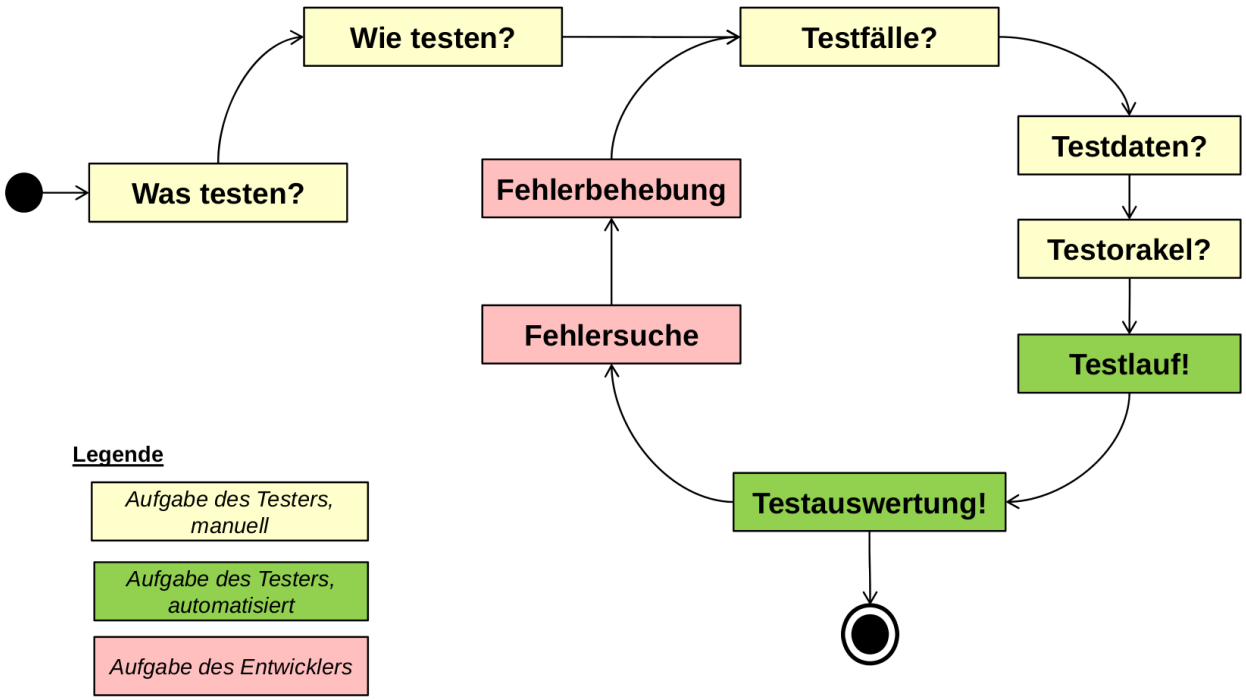
Lebenszyklus Tests
3. Testfälle



Testfälle: Entscheide was genau getestet wird

- Ein Testfall ist eine Menge von Testdaten oder Situationen, die benutzt werden um die zu testende Einheit (Code, Modul, System) oder das gemessene Attribut zu überprüfen

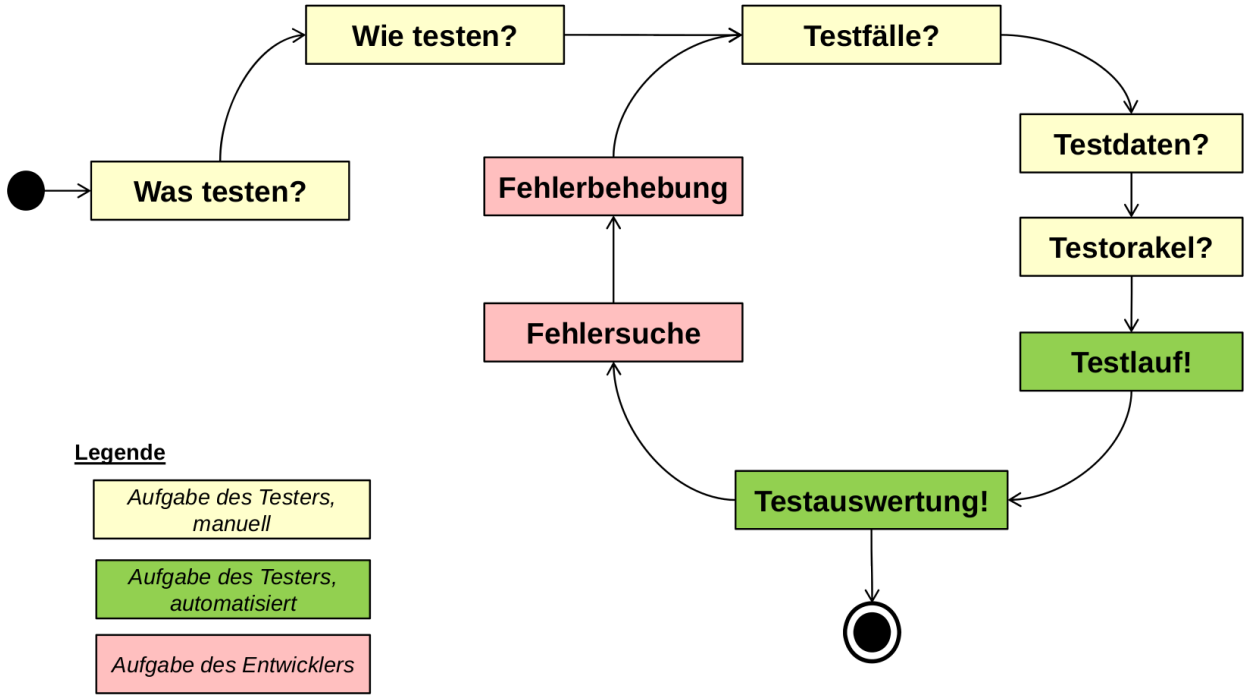
Lebenszyklus Tests
4. Testdaten



Testdaten: Entscheide wie der Testfall charakterisiert werden kann

- Daten, die einem bestimmten Testfall entsprechen

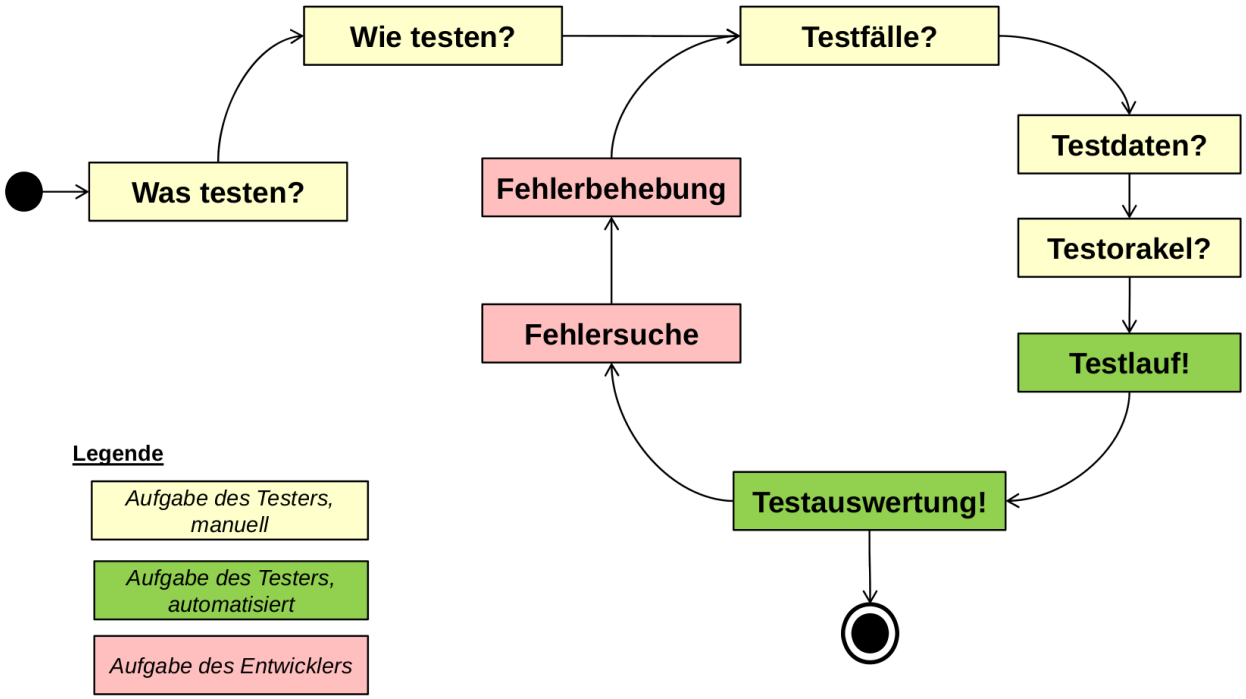
Lebenszyklus Tests
5. Testorakel



Testorakel: Entscheide anhand wovon der Testerfolg geprüft wird

- Ein Orakel besteht aus den vorhergesagten Ergebnissen für eine Menge von Testfällen
- Das Testorakel muss geschrieben werden, bevor das eigentliche Testen stattfindet

Lebenszyklus Tests
6. Testlauf

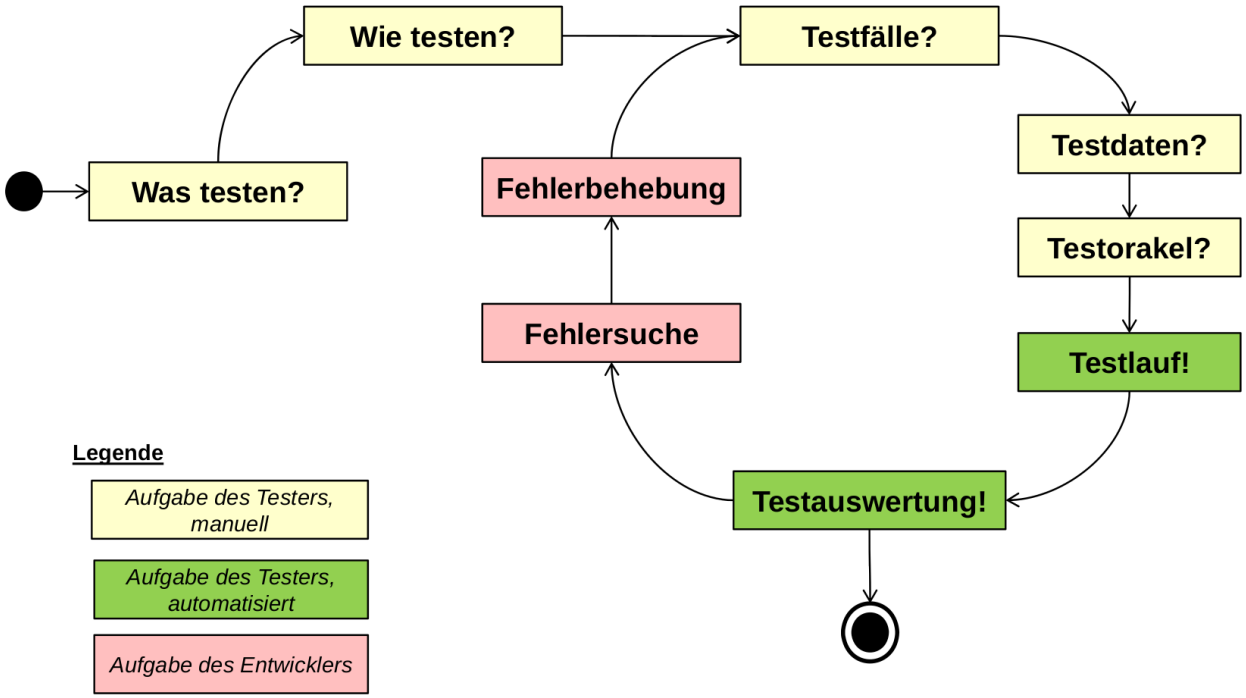


Testlauf

- Die eigentliche Ausführung des Tests und das Sammeln der Testergebnisse

Lebenszyklus Tests

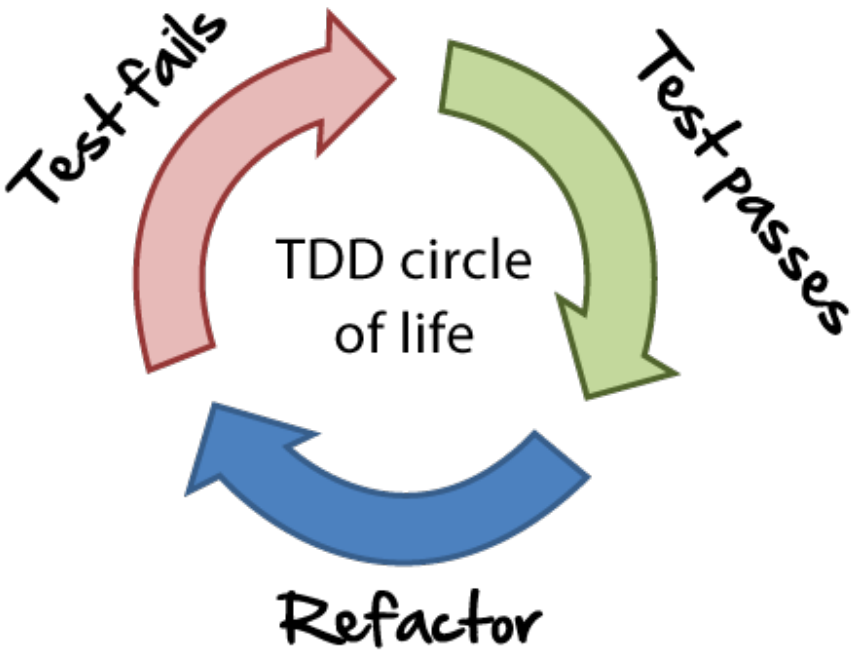
7. Testauswertung



Testauswertung

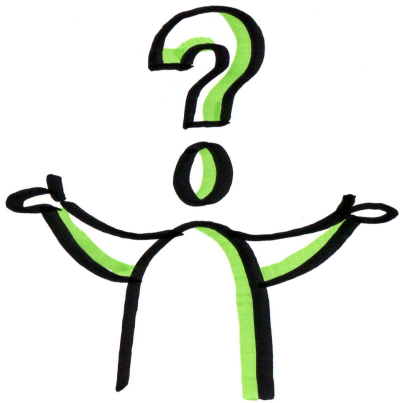
- Vergleich der Testergebnisse mit dem Orakel

Lebenszyklus Tests
Test Driven Development



- Schreibe einen neuen Tests…
  - der zur Spezifikation passt
  - rot wird
- Fix den Code im Sinne der Spezifikation bis alle Tests Grün werden
- Passe die Tests an
  - Äquivalenzklassen zusammenfassen
  - Gemeinsam genutzte Logik extrahieren

Fragen
Fragen?



Definition (Unit Test)

Ein Unit Test (auch Modultest oder Komponententest) wird in der Softwareentwicklung angewendet, um die funktionalen Einzelteile (Module) von Computerprogrammen zu testen, d. h., sie auf korrekte Funktionalität zu prüfen.

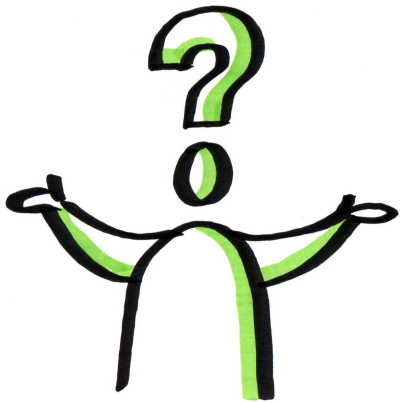
Begriffsklärung
Eigenschaften Unit Tests

- Konsistent
- Eindeutig
- Testet ein Subjekt
- Testet einen Testfall
- Ohne eigene Logik oder Schleifen
- Präzise Annahmen
- Selbsterklärend

Begriffsklärung
Wirkung der Eigenschaften

- Begrenzt komplex durch Fokus auf Modul
- Klar definierte Schnittstellen
- Senkt die Komplexität von Integrationstests
  - Integrationstest können stichprobenartig durchgeführt werden
- Es stehen verschiedene Strategien zur Ermittlung von Testfällen zur Verfügung

Fragen
Fragen?



Black-box Tests
Eigenschaften Black-box Tests

- Testfallauswahl anhand von Analysewissen über funkt. Anforderungen
  - Use cases
  - Erwartete Eingabedaten
  - Ungültige Eingabedaten
- Fokus: Ein-/Ausgabeverhalten
  - Das Modul besteht den Test, wenn für jede gegebenen Eingabe die Ausgabe vorhersagbar ist
  - Es ist meist unmöglich, jede mögliche Eingabe zu erzeugen („test cases“)

Black-box Tests
Partitionierung

- Reduzierung der Anzahl der Testfälle durch Partitionierung
  - Zerlege Eingabe in Äquivalenzklassen
  - Wähle Testfälle für jede der Äquivalenzklassen. (Beispiel: Wenn ein Objekt negativen Zahlen akzeptieren soll, reicht es aus, mit einer negativen Zahl zu testen)

Wahl der Partitionierung (Richtlinie)

- Gültige Eingabewerte liegen in einem Intervall. Wähle Testfälle aus drei Äquivalenzklassen
  - Unterhalb des Intervalls
  - Im Intervall
  - Oberhalb des Intervalls
- Gültige Eingabewerte bilden eine diskrete Menge. Wähle Testfälle aus zwei Äquivalenzklassen:
  - Gültiger diskreter Wert
  - Ungültiger diskreter Wert

Black-box Tests
Beispiel

Contract

Verbindet die beiden Werte mit Hilfe des Operators und gibt das Ergebnis zurück.

```php
class Calculator
{
  /**
  * @param float $a
  * @param float $b
  * @param Operator $operator
  *
  * @return mixed
  */
  public function calculateBinaryOperation($a, $b, Operator $operator)
}
```

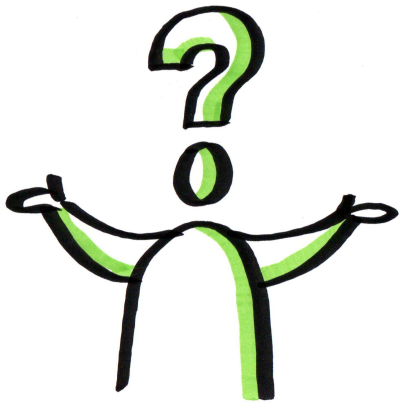
Black-box Tests
Beispiel

Contract

Diese Klasse implementiert die Division.

```php
class Calculator
{
  /**
  * @param float $a
  * @param float $b
  *
  * @return float
  */
  public function division($a, $b)
}
```

Fragen
Fragen?



Definition (White-box Tests)

Der Begriff White-Box-Test (seltener auch Glass-Box-Test) bezeichnet eine Methode des Software-Tests, bei der die Tests mit Kenntnissen über die innere Funktionsweise des zu testenden Systems entwickelt werden.

White-box Tests
Ziel von White-box Tests

Wissen über Entwurf und Implementierung nutzen um …

- Anzahl von Testfällen zu begrenzen
- Gründliche Testabdeckung zu gewährleisten

White-box Tests
Ziel von White-box Tests

Testfallauswahl

- Entwurfswissen über Systemaufbau, Algorithmen, Datenstrukturen
- Kontrollstrukturen: Verzweigungen, Schleifen, …
- Datenstrukturen: Datensätze, Felder, Arrays, …

Definition (Testabdeckung)

Abdeckung ist prozentuales Maß der vom Test durchlaufenen Teile der getesteten Komponente im Verhältnis zur Gesamtanzahl solcher Teile.

$$\text{Abdeckung} = \frac{\text{AnzahlgetesteterTeile}}{\text{AnzahlallerTeile}}$$

Zu testende Teile

- jede Anweisung rightarrow Anweisungsabdeckung
- jede Verzweigung rightarrow Verzweigungsabdeckung
- jede Schleife rightarrow Schleifenabdeckung
- jeder Pfad rightarrow Pfadabdeckung

Ein „Teil“ gilt als getestet wenn es während des Testlaufs mindestens einmal durchlaufen wurde

White-box Tests
Anweisungsabdeckung

Nicht einfach, 100% Anweisungsabdeckung zu erreichen

- Fehlerbedingungen / seltene Ereignisse:

```
if (param > 20) {
  print("Dies␣sollte␣nie␣geschehen!");
}
```

- Exceptions inmitten einer Anweisungsfolge:

```
doThis(); # throws exception
doThat(); # never reached
```

Eine Anweisung ist abgedeckt, wenn sie ausgeführt wird.
Anweisung != Codezeile

White-box Tests
Verzweigungsabdeckung

- Prüft alle Alternativen einer Verzweigung
- Schützt vor Fehlern, die dadurch entstehen, dass bestimmte Anforderungen in einem Zweig nicht erfüllt sind.

```
$divisor = 0;

if ($flag){$divisor = 1;}
else {     $divisor = 0;}

echo 3 / $divisor;
```

Der Code wirft eine Exception, falls `$flag` gleich false ist.

White-box Tests
Verzweigungsabdeckung

Fehlendes else ist auch ein Zweig der getestet werden muss!

```
$divisor = 0;
if ($flag){$divisor = 1;}
echo 3 / $divisor;
```

Fun Fact

100% Verzweigungsabdeckung impliziert 100% Anweisungsabdeckung, sofern das Programm keine Exception wirft.

White-box Tests
Grenzen der Verzweigungsabdeckung

Es gibt Fehlerklassen, die durch Verzweigungsabdeckung nicht erkannt werden.

```
$value = null;
if ($a) { $value = new DateTime(); }
if ($b) { $value->format('Y');}
```

- 100% Verzweigungsabdeckung ist gegeben, wenn ($a, $b) mit (true, true) und (false, false) belegt wird
- Für ($a, $b) = (false, true) gibt‘s einen Zugriff auf null → Exception

White-box Tests
Pfadabdeckung

Ziel der Pfadabdeckung ist das Durchlaufen des Programmes auf jedem mögliche Pfad.

```
$value = null;
if ($a) { $value = new DateTime(); }
if ($b) { $value->format('Y');}
```

- Verfügbare Pfade `(\$a, \$b)= (true,true),(false,true),(true, false),(false,false)`

Pfad ist jeder Weg von Startpunkt zum Endpunkt im Kontrollflussgraphen des analysierten Codes

- Return-Anweisungen haben Kanten zum Endpunkt
- Exceptions nicht $\rightarrow$ sie stellen kein ”normales“ Ende dar!
- Fehlende else-Anweisungen zählen als Kanten im Graphen
  - In obigem Beispiel ist gerade der ”fehlende“ Zweig a=false kritisch!

Halteproblem

Problem: Zyklen können zu einer unendlichen Anzahl von Pfaden führen.

Lösung: Pfade durch ein und den selbe Zyklus kollabieren zu einer Äquivalenzklasse.

Zu viele Pfade

Problem: Der Programmcode ist zu komplex.

Folgerung: Anderes Testverfahren wählen oder Komplexität reduzieren.

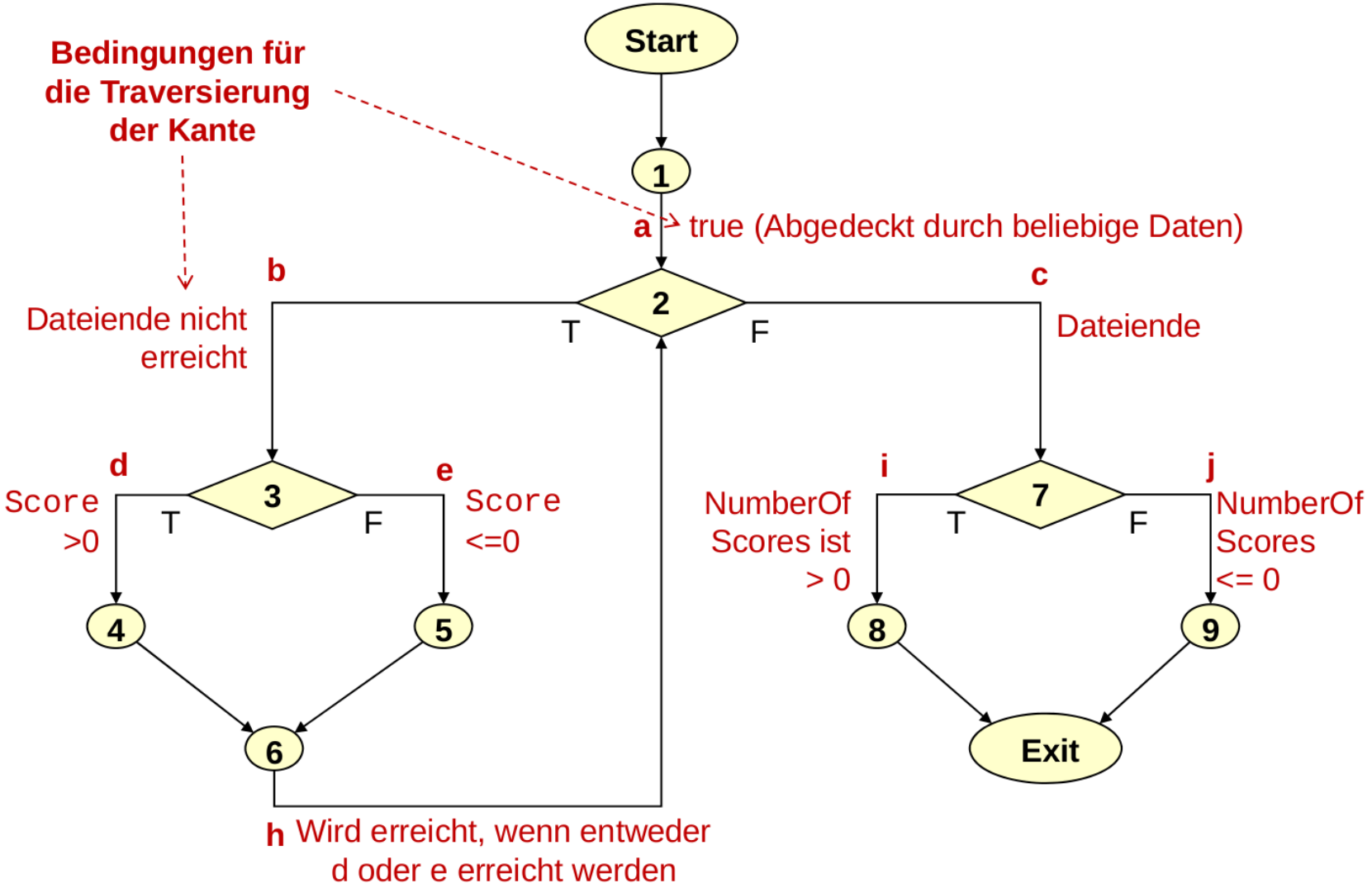
Fun Fact

100% Pfadabdeckung
impliziert
100% Verzweigungsabdeckung
und
100% Anweisungsabdeckung

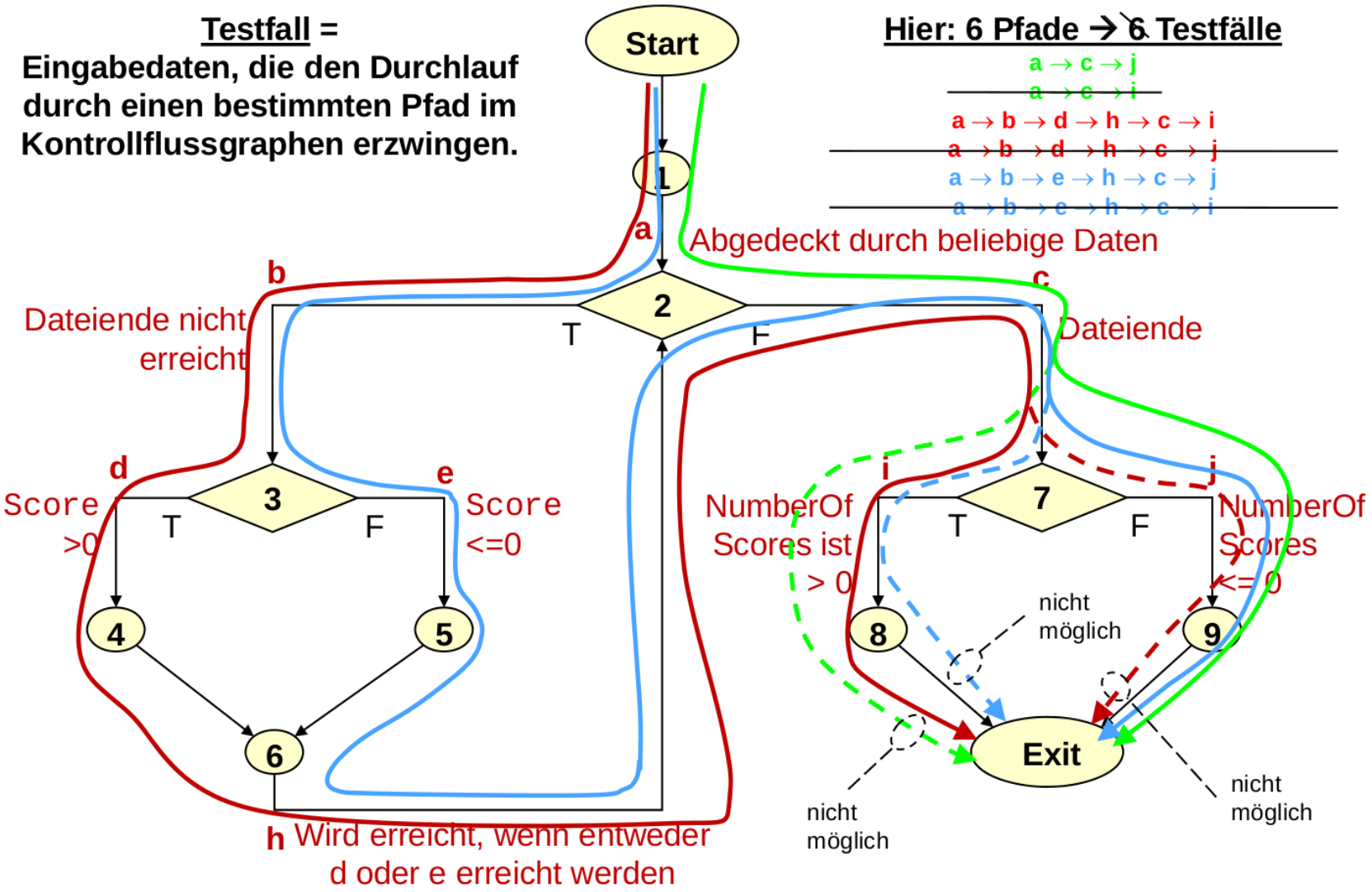
White-box Tests
Verzweigungsabdeckung


```
1  FindMean (FILE ScoreFile) {
2    int NumberOfScores = 0; //1
3    float SumOfScores = 0.0; //1
4    float Mean = 0.0; //1
5    float Score; //1
6    Read(ScoreFile, Score); //1
7    while ( !EOF(ScoreFile) ) { //2
8      if (Score > 0.0 ) { //3
9        SumOfScores = SumOfScores + Score; //4
10       NumberOfScores++; //4
11     } //5
12     Read(ScoreFile, Score); //6
13   }
14   /* Compute the mean and print the result */
15   if (NumberOfScores > 0) { //7
16     Mean = SumOfScores / NumberOfScores; //8
17     printf(" The mean score is %f\n", Mean); //8
18   } else printf ("No scores found in file\n"); //9
19 }
```

White-box Tests
Verzweigungsabdeckung



Testfall =
Eingabedaten, die den Durchlauf durch einen bestimmten Pfad im Kontrollflussgraphen erzwingen.

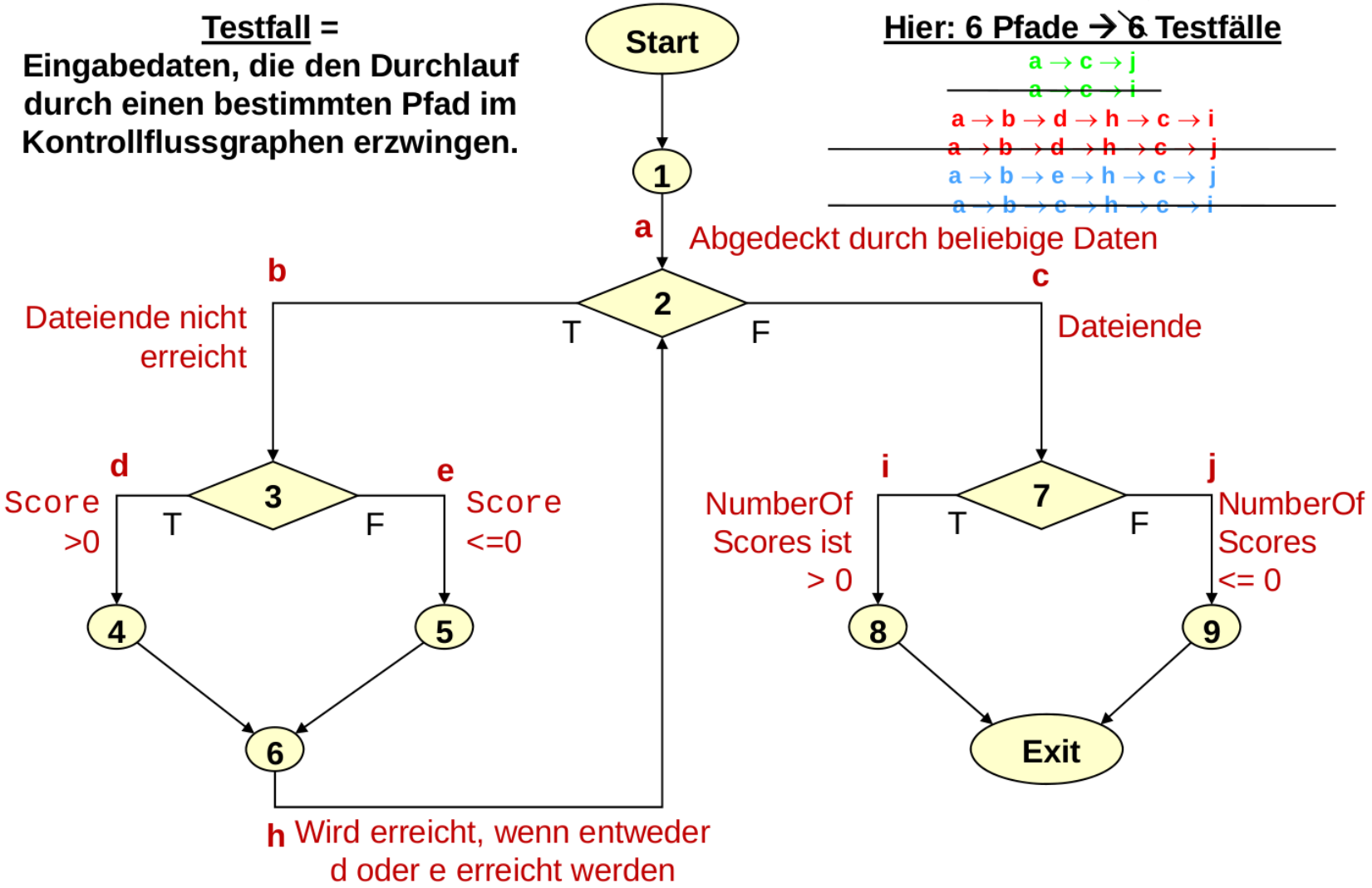


Hier: 6 Pfade → ~~6~~ Testfälle

a → c → j
~~a → c → i~~
a → b → d → h → c → i
~~a → b → d → h → c → j~~
a → b → e → h → c → j
~~a → b → e → h → c → i~~

White-box Tests
Verzweigungsabdeckung

Testfall =
Eingabedaten, die den Durchlauf durch einen bestimmten Pfad im Kontrollflussgraphen erzwingen.



Hier: 6 Pfade → 6 Testfälle

a → c → j
~~a → c → i~~
a → b → d → h → c → i
~~a → b → d → h → c → j~~
a → b → e → h → c → j
~~a → b → e → h → c → i~~

Abgedeckt durch beliebige Daten

White-box Tests
Verzweigungsabdeckung

Testfall =
Eingabedaten, die den Durchlauf durch einen bestimmten Pfad im Kontrollflussgraphen erzwingen.

Hier: 6 Pfade → ~~6~~ Testfälle

a → c → j
a → c → i
a → b → d → h → c → i
a → b → d → h → c → j
a → b → e → h → c → j
a → b → e → h → c → i

Relevante Kantenbedingungen

- **c:** Dateiende
- **i:** NumberOfScores > 0 ≅ positive Zahlen gelesen
- **j:** NumberOfScores <= 0 ≅ keine positive Zahlen gelesen
- **b:** Dateiende nicht erreicht
- **d:** `Score` >0 ≅ positive Zahl gelesen
- **e:** `Score` <=0 ≅ keine positive Zahlen gelesen

Minimale Testfälle / Testdaten

1. Leere Datei
2. Datei mit positiver Zahl
3. Datei mit negativer Zahl

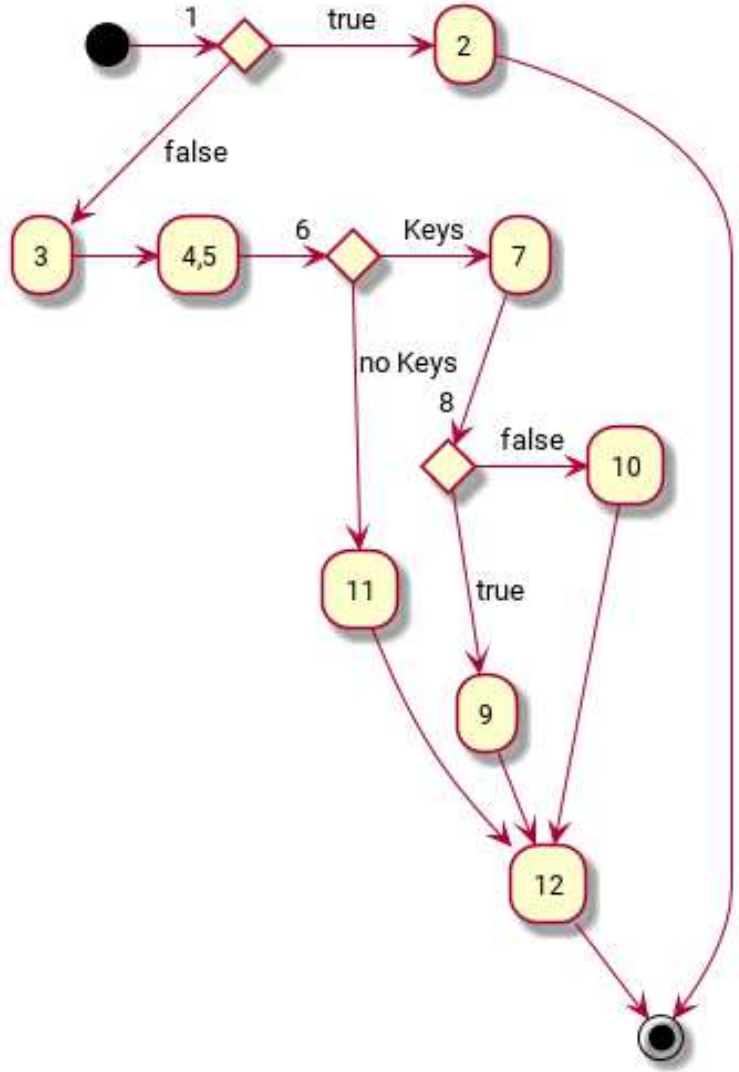
White-box Tests
Verzweigungsabdeckung

1. Kontrollflussgraphen erstellen
2. Kanten beschriften
3. Pfade bestimmen
4. Unmögliche Pfade eliminieren
5. Kanten die keine Bedingung mit sich tragen eliminieren
6. Kanten, deren Bedingung durch die vorangegangenen Schritte erfüllt wurden eliminieren
7. Eingabedaten bestimmen, die die Bedingungen der restlichen Kanten erfüllen
8. Testdatensatz erzeugen
9. Testorakel dafür erzeugen
10. Kontrollflussgraphen erstellen

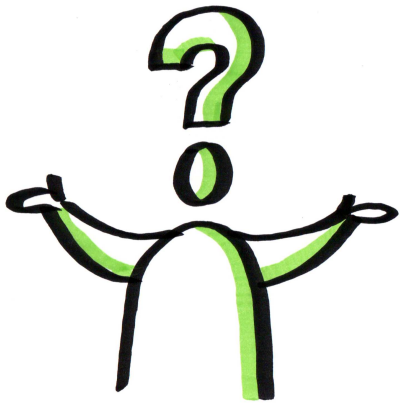
White-box Tests
Übung


```
2   public function findIntersection(&$firstSet, &$secondSet) {
3     if (!is_array($firstSet) || !is_array($secondSet)) { #1
4       return $firstSet === $secondSet; #2
5     } #3
6
7     $commonKeys   = array_intersect(array_keys($firstSet),
      ↪  array_keys($secondSet)); #4
8     $returnValue = array(); #5
9     foreach ($commonKeys as $key) { #6
10      $intersection = $this->findIntersection($firstSet[$key],
        ↪  $secondSet[$key]); #7
11      if ($intersection !== array()) { #8
12        $returnValue[$key] = $intersection; #9
13      } #10
14    } #11
15    return $returnValue; #12
16  }
```

White-box Tests
Verzweigungsabdeckung



Fragen
Fragen?



White-box vs Black-box
Zusammenfassung

White-box

- Potentiell unendliche Anzahl von Pfaden muss getestet werden
- Getestet wird anhand der tatsächlich anstatt des erwarteten Verhaltens
- Keine Erkennung fehlender Use Cases

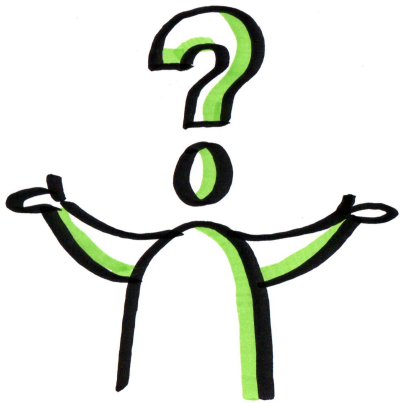
Black-box

- Potentielle kombinatorische Explosion der Testfälle (gültige & ungültige Daten)
- Oft ist es unklar, ob die gewählten Testfälle einen bestimmten Fehler entdecken
- Keine Entdeckung belangloser Use Cases

Fact

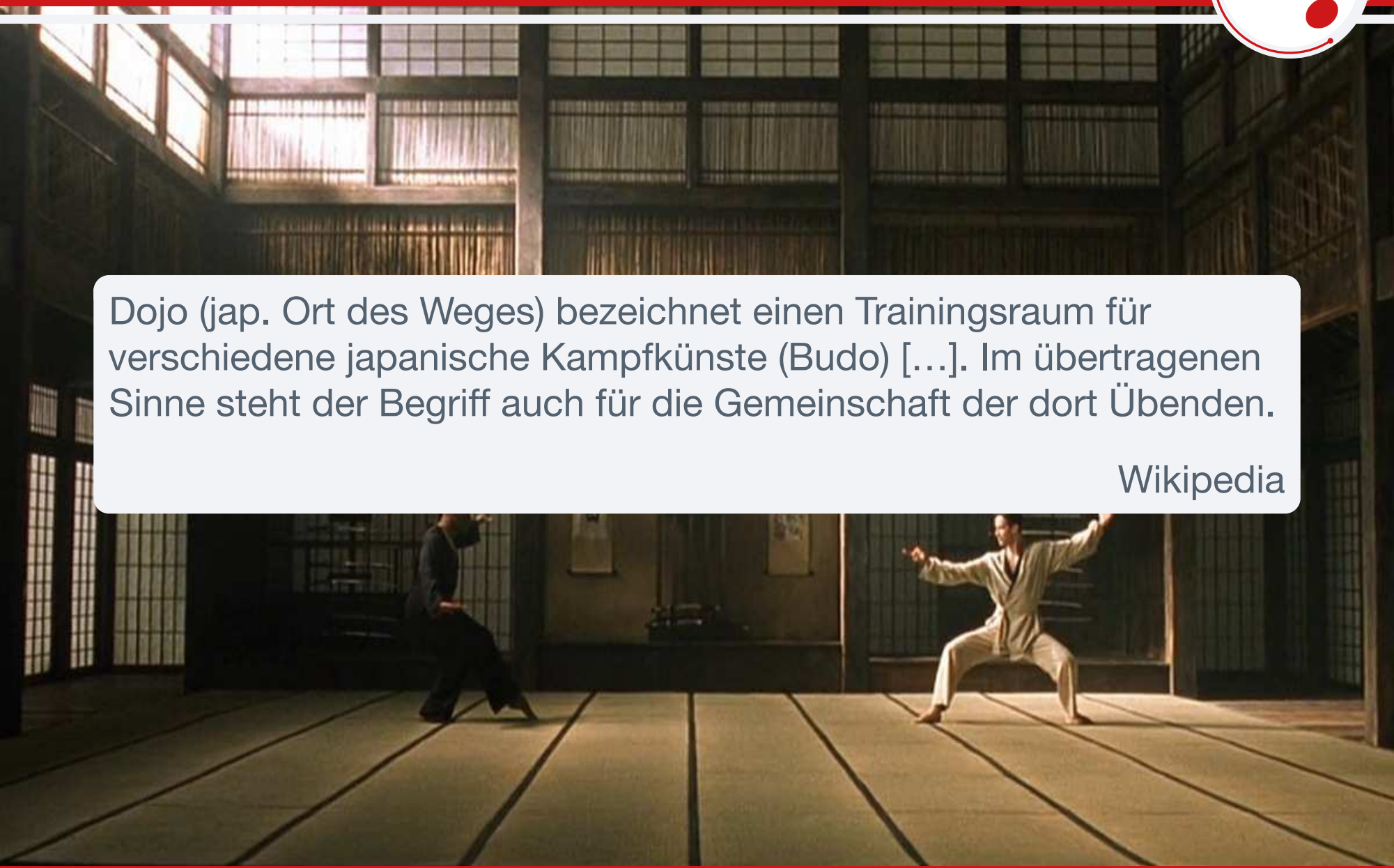
White-box-Tests und Black-box-Tests sind die beiden Extreme des Modultest-Kontinuums. Beide Testtypen sind erforderlich.

Fragen
Fragen?



Code Dojo

Was ist ein Dojo?



Dojo (jap. Ort des Weges) bezeichnet einen Trainingsraum für verschiedene japanische Kampfkünste (Budo) […]. Im übertragenen Sinne steht der Begriff auch für die Gemeinschaft der dort Übenden.

Wikipedia

Code Dojo
Was ist ein Dojo?

Die Gemeinschaft führt gemeinsam Übungen – so genannte Katas – durch um ihr Wissen in der entsprechenden Disziplin zu vervollkommnen.

Code Kata
Was ist ein Code Kata?

- Kleine, unabhängige, fokussierte, in sich geschlossene Übung
- Übt die Ausführung und Herangehensweise
- Bietet Raum für gemeinsames Lernen
- Lösung der Aufgabe erklärtes Nicht-Ziel

Fokus

- Pair Programming + TDD = TDD-Game
- Keine PHPMD Regel darf gebrochen werden
- Absichtsvolles Testen
  - Test wird zuallererst dem Pair-Partner erklärt, dann programmiert
  - Auswahl des Zwecks (use case, Erwartete Eingaben …)
  - Auswahl der Kategorie (Black-, White-box)
  - Gegebenenfalls Auswahl der Äquivalenzklasse.

Anforderung

- Eine Klasse muss die folgenden public Methoden haben
  - addPointToPlayer(PlayerName:string):null - Soll aufgerufen werden wenn ein Spieler einen Punkt erzielt. Wirft eine Exception, wenn das Spiel vorbei ist
  - getCurrentScore():string - Gibt eine Übersicht über alle gespielten Sätze, den aktuellen Punktestand und ob ein Spieler gewonnen hat aus. Es gelten die Tennisregeln mit 2 Gewinnsätzen.
- Die Standardregeln von PHPMD müssen eingehalten werden.