



Mutationstests

Wieso, weshalb, warum?

Dehla Sokenou

Test- und Qualitätsmanagerin
Software-Architektin

Sprecherin der GI-Fachgruppe TAV



<https://www.wps.de/>



<https://fg-tav.gi.de/>

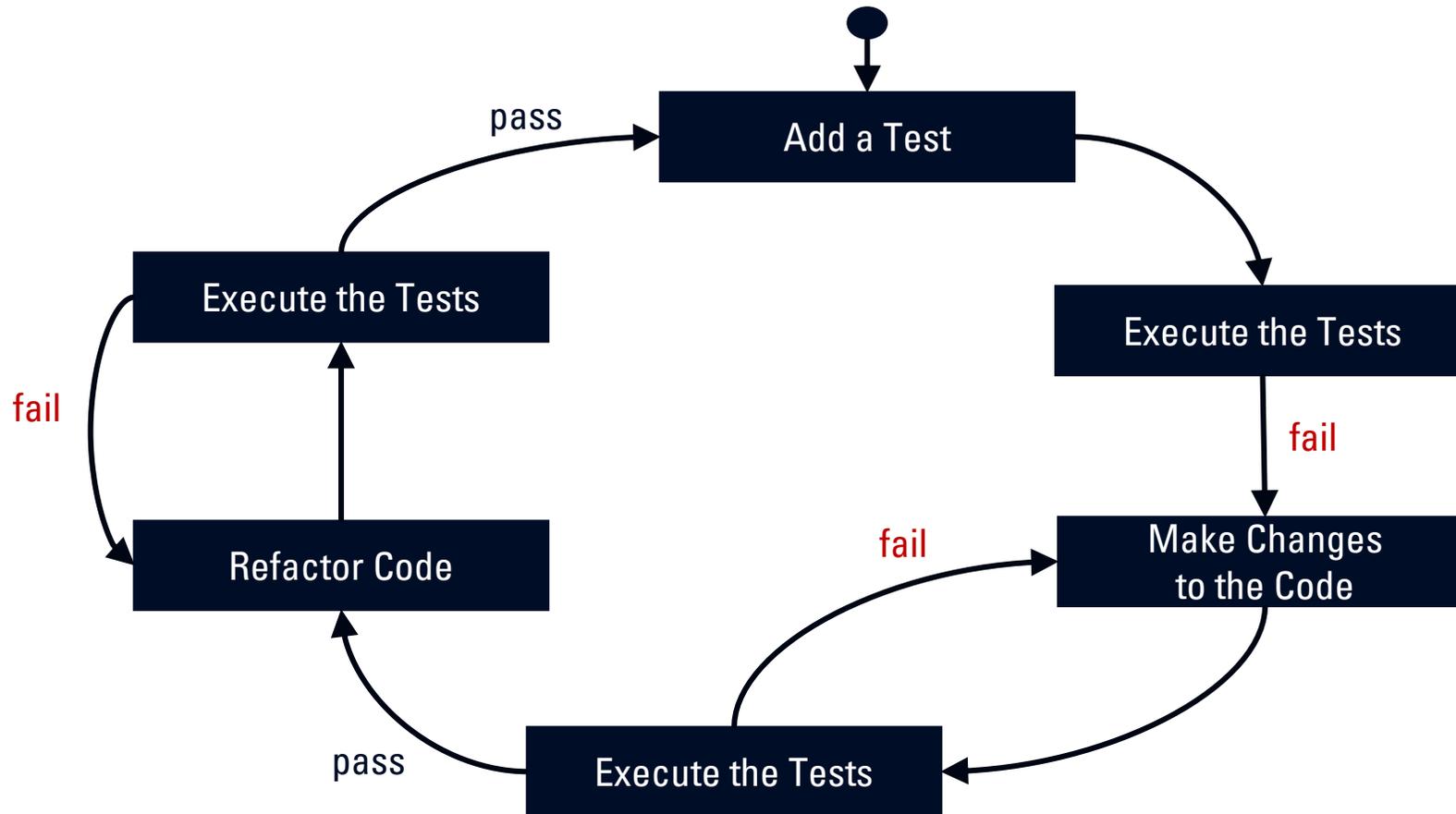


<https://dpsq.de/>

Ihr macht doch TDD – warum denn noch mehr?



➤ Der TDD-Zyklus und die Sicherheit...



➤ ABER:

- Machen wir zu viel?
- Machen wir das Richtige?



- Langlaufendes (sehr) agiles Projekt
- Backend und Frontend – Microservice-Architektur mit Micro-Frontends

- Teamübergreifendes Testkonzept:
 - Management-Vorgabe und Projektziel: > 90% Branch-Coverage
 - Test-Driven Development erwünscht, aber nicht gefordert
 - Aber unter uns: 90% Branch-Coverage ist mit TDD geschenkt 😊
 - Weitere Testaktivitäten u.a. End2End-Tests, Bingo-Bongo-Testsessions, ...
 - Mutationstests nach etwa einem halben Jahr Laufzeit eingeführt



1. The *competent programmer* hypothesis

- Programmierer implementieren in der Regel (fast) korrekte Programme
- Kleinere Fehler lassen sich trotzdem nicht vermeiden

2. The *coupling effect* hypothesis

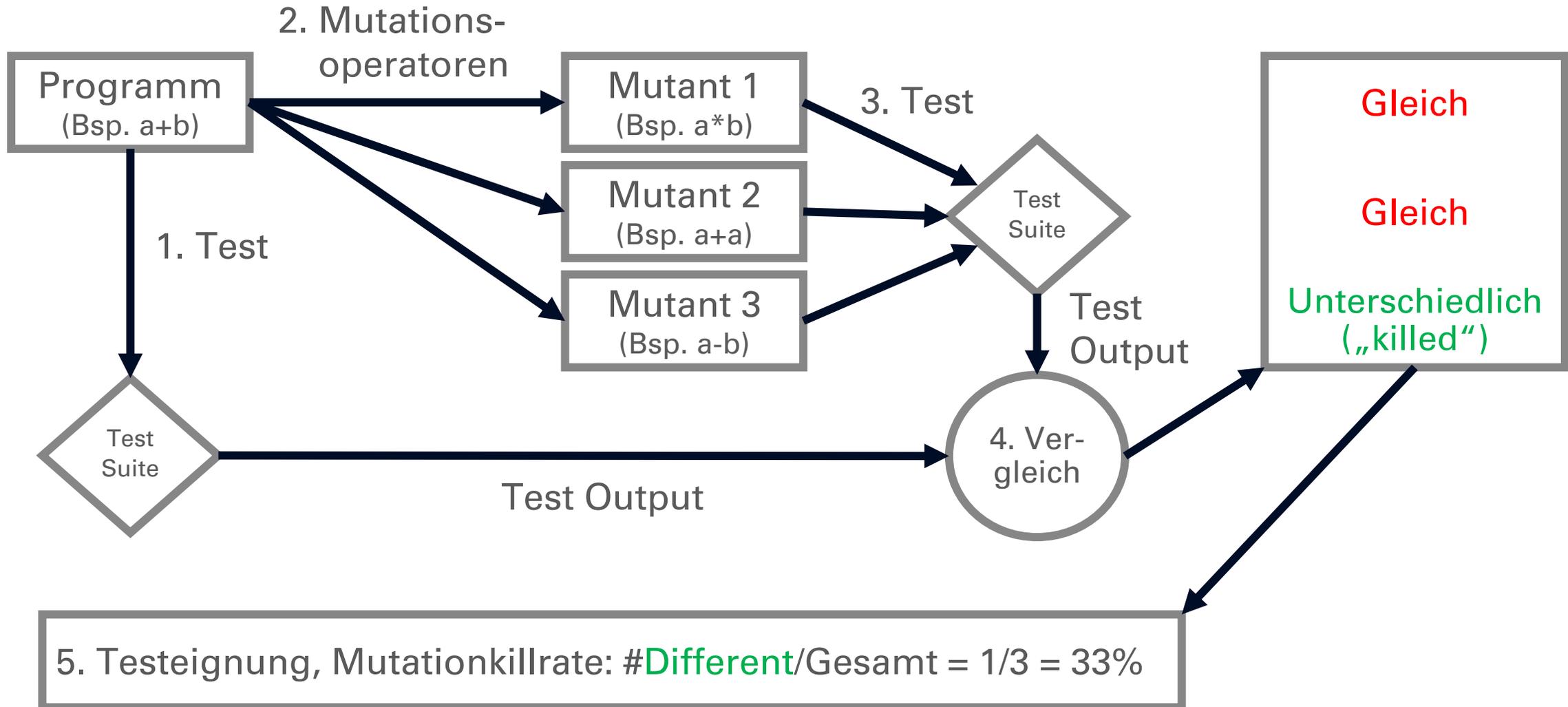
- Kleinere Fehler im Code können große Probleme verursachen

➤ Deshalb: Testen!

➤ ABER:

- Machen wir zu wenig?
- Machen wir zu viel?
- Machen wir das Richtige?

Mutationstests – wie funktioniert es?



Die Mutanten sind los!



- Mutationen sind typische Programmierfehler oder typische Fallstricke im Code
 - Leere Returns (null, leere Collection statt richtigem Ergebnis)
 - Negation einer Bedingung (true statt false)
 - Änderung einer bool'schen Operation (> statt >=) oder arithmetischen Operation (+ statt -)
 - Anweisung entfernen oder duplizieren
 - Methodenbody entfernen
 - Variablen vertauschen
- Abhängig von der verwendeten Programmiersprache
 - Beispiel: Annotation entfernen



Oder: warum ist es inzwischen wieder ein Thema?

- Idee vorgestellt im Jahr 1971 (!) von Richard Lipton
- 1. Implementierung eines Tools von Timothy Budd im Jahr 1980
- Dann war es lange ruhig
 - Da Mutationstests sehr rechenintensiv sind, fehlte lange einfach die Leistung
 - Was erst mal in der Schublade liegt, bleibt oft auch erst einmal dort
- Wiederbelebung erfolgt langsam
 - Seit den 2020ern vermehrt Artikel zu Thema
 - Obwohl es Werkzeuge wie PITest (PIT) schon länger gibt
 - Es reicht ein simples Integrieren in die Pipeline, z.B. als Maven-Plugin / npm-Paket

Test your tests...!



➤ Überdeckungsmessung zeigt Qualität der ACTs

- Habe ich alle Teile meines Source-Codes mit den Tests „getroffen“?

➤ Mutationstests zeigen Qualität der ACTs und ASSERTs

- Sind meine Tests gut genug?
- Aber die schnellsten sind es leider immer noch nicht...

Element	Missed Instructions	Cov.	Missed Branches	Cov.
	1	0 %		n/a
domain		54 %		36 %
web		0 %		0 %
common.domain		100 %		100 %
common.persistence		98 %	1	100 %
common.web		20 %		33 %
domain		89 %		79 %
persistence		2 %		0 %
web		56 %		43 %
Total	1,363 of 3,222		93 of 188	50 %

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
43	60% 483/806	51% 235/462	83% 235/284

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
domain	7	59% 63/106	38% 24/63	77% 24/31
web	2	0% 0/69	0% 0/32	0% 0/0
common.domain	2	100% 13/13	64% 14/22	78% 14/18
common.persistence	3	97% 67/69	89% 25/28	93% 25/27
common.web	7	31% 23/74	25% 14/57	100% 14/14
domain	5	95% 146/153	85% 100/117	90% 100/111
persistence	1	3% 1/34	0% 0/18	0% 0/0
web	5	62% 85/137	58% 44/76	73% 44/60

DISCLAIMER:
KEIN TDD-PROJEKT!

Was wir lernen mussten...



- JaCoCo vs. PIT
 - Test-Driven Development führt normalerweise zu guter Code-Überdeckung, aber:
 - 100% Branch-Coverage bedeutet nicht zwangsläufig gute Tests

Element	Missed Instructions	Cov.	Missed Branches	Cov.
...	1	0 %		n/a
... domain		54 %		36 %
... web		0 %		0 %
... common.domain		100 %		100 %
... common.persistence		98 %		100 %
... common.web		20 %		33 %
... domain		89 %		79 %
... persistence		2 %		0 %
... web		58 %		43 %
Total	1.363 of 3.222	57 %	93 of 188	50 %

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
43	60% 483/806	51% 235/462	83% 235/284

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
... domain	7	59% 63/106	38% 24/63	77% 24/31
... t.web	2	0% 0/69	0% 0/32	0% 0/0
... common.domain	2	100% 13/13	64% 14/22	78% 14/18
... common.persistence	3	97% 67/69	89% 25/28	93% 25/27
... common.web	7	31% 23/74	25% 14/57	100% 14/14
... domain	5	95% 146/153	85% 100/117	90% 100/111
... persistence	1	3% 1/34	0% 0/18	0% 0/0
... web	5	62% 85/137	58% 44/76	73% 44/60

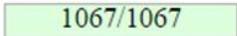
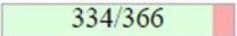
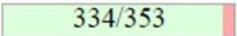
* Beispiel aus einem Non-TDD-Projekt

Und jetzt ein Beispiel aus einem TDD-Projekt

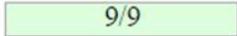
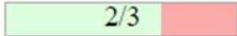
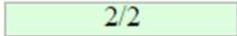
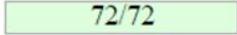
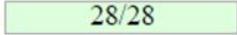
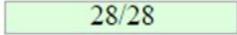
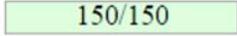
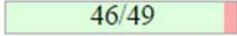
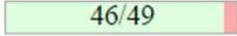
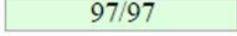
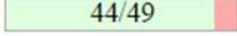
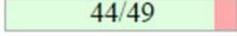
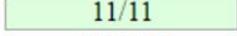
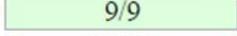
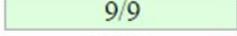
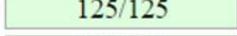
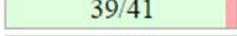
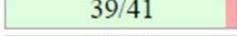
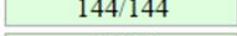
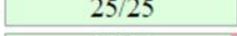
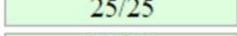
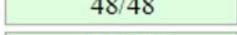
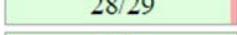
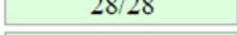
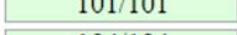
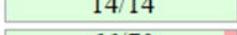
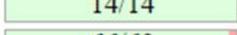
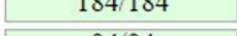
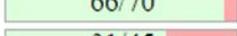
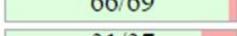
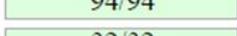
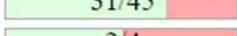
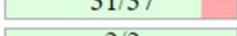
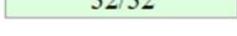
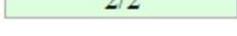


Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
61	100%  1067/1067	91%  334/366	95%  334/353

Breakdown by Package

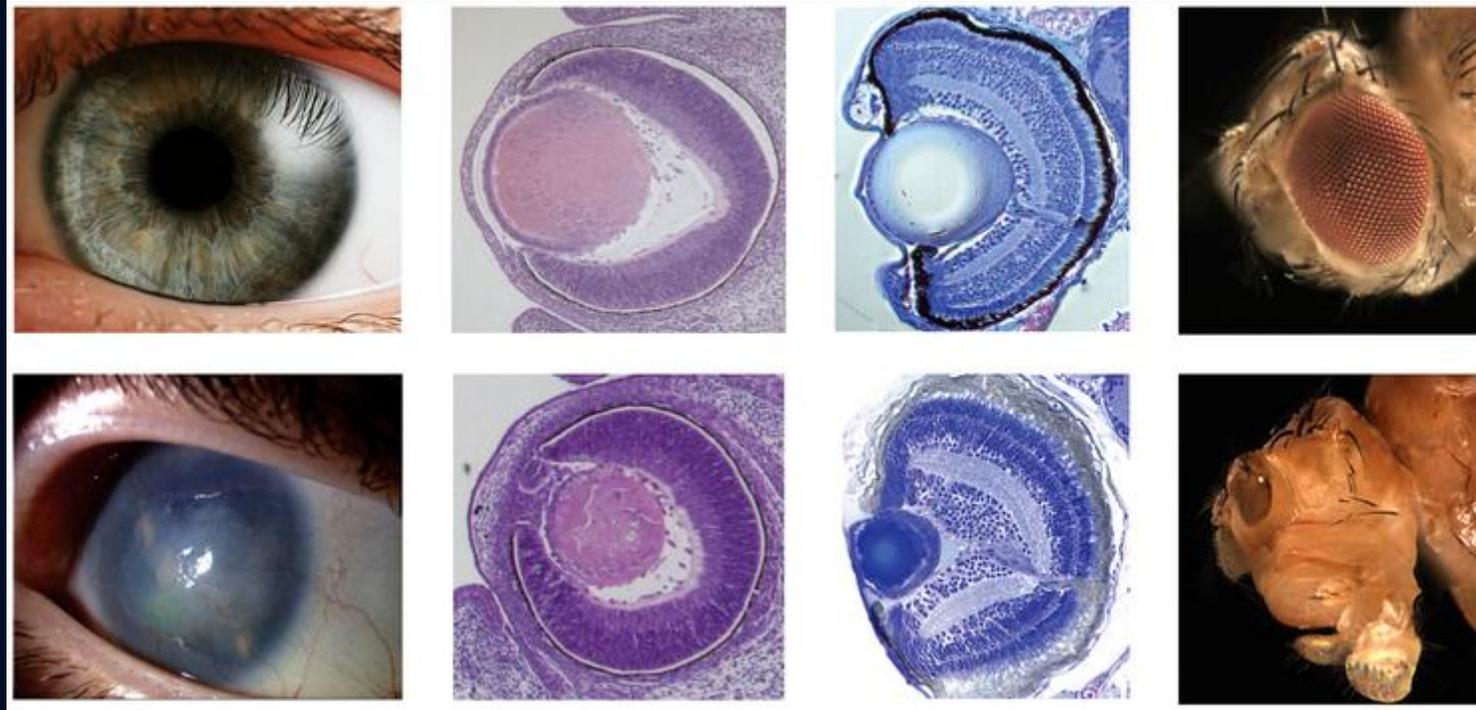
Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
config	3	100%  9/9	67%  2/3	100%  2/2
db.migration	5	100%  72/72	100%  28/28	100%  28/28
domain.model	5	100%  150/150	94%  46/49	94%  46/49
domain.model.value	21	100%  97/97	90%  44/49	90%  44/49
domain.model.wrapper	1	100%  11/11	100%  9/9	100%  9/9
domain.service	2	100%  125/125	95%  39/41	95%  39/41
service	2	100%  144/144	100%  25/25	100%  25/25
service.event	8	100%  48/48	97%  28/29	100%  28/28
service.mapper	2	100%  101/101	100%  14/14	100%  14/14
service.parser	3	100%  184/184	94%  66/70	96%  66/69
service.parser.model	7	100%  94/94	69%  31/45	84%  31/37
web.controller	2	100%  32/32	50%  2/4	100%  2/2

Ein genauer Blick lohnt sich auch hier

Was tun mit den Überlebenden?

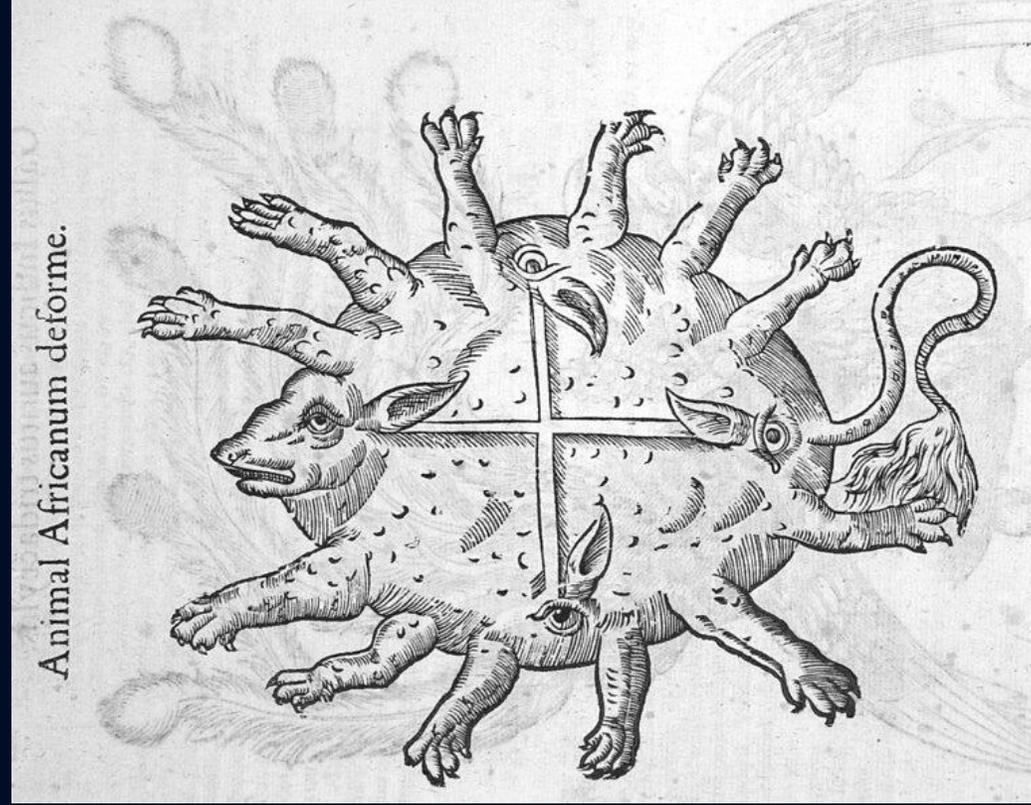


Original Source-Code	Mutant	Analyseergebnis	Aktion
PASS	FAILED	Mutant getötet	Gute Testqualität, keine Aktion
PASS	FAILED wegen eines Fehlers	Mutant invalide	Ignorieren, keine Aktion
PASS	PASS	Mutant hat überlebt	Verbessere die Tests
PASS	PASS	Mutant ist äquivalent	Als äquivalent markieren



Ein bisschen Code

https://github.com/dsokenou/public_mutationtesting



Einige Beispiele aus der Praxis

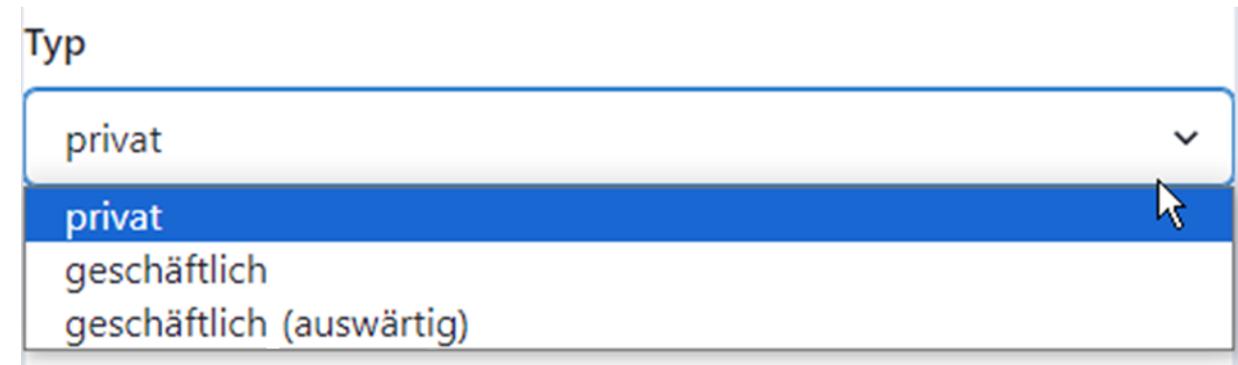


Beispiel: wenn das UI leer bleiben kann...

- Kein Test zur Prüfung der angezeigten Werte aus den Enums

```
enum class BookingItemType(val value: String) {  
  
    PRIVATE("privat"),  
    BUSINESS("geschäftlich"),  
    BUSINESS_ABROAD("geschäftlich (auswärtig)")  
  
}
```

replaced return value with ""



Beispiel: wenn nichts in der Datenbank landen kann



- Repository.save nicht aufgerufen – fehlendes verify

```
fun saveProject(projectValues: ProjectValues): Project {  
    var project = repository.loadProject(project)  
    project.updateWith(projectValues)
```

```
    project = repository.save(project)  
    return project  
}
```

```
removed call to ...
```

Beispiel: wenn Code gar nicht erreichbar ist



- Überflüssiger Code – weil subsummiert

```
fun checkProject(project: Project) {  
    checkProjectDates(project)  
    checkProjectTimes(project)  
}
```

```
fun checkProjectDates(project: Project) {  
    require(project.date.start <= project.date.end) {  
        "Der Starttag muss vor dem Endtag liegen oder gleich diesem sein."  
    }  
    checkProjectTimes(project)  
}
```

removed call to ...

Beispiel: wenn Grenzwertanalyse ein Fremdwort ist



- Grenzwerte nicht beachtet

```
class BookingItem(val start: LocalDate, val end: LocalDate) {  
  
    fun checkDateInFuture() {  
        require(start > LocalDate.now()) {  
            "Der Start muss in der Zukunft liegen."  
        }  
    }  
}
```

changed conditional boundary

Beispiel: wenn man manchmal eine Konstellation nicht herstellen kann



- Randomzahlenberechnung – ein Teil der Berechnung nie erreicht, warum nicht? Annahme falsch?

```
fun generateRandomValidPassword(length: Int): String {  
    var password = ""  
    do {  
        password = generateRandomPassword(length)  
    } while (!isValidPassword(password))  
    return password  
}
```

```
private fun isValidPassword(password: String): Boolean {  
    // some implementation  
}
```

replaced return value with true

Vorher vs. nachher 😊



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
61	100% 1067/1067	91% 334/366	95% 334/353

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
config	3	100% 9/9	67% 2/3	100% 2/2
db.migration	5	100% 72/72	100% 28/28	100% 28/28
domain.model	5	100% 150/150	94% 46/49	94% 46/49
domain.model.value	21	100% 97/97	90% 44/49	90% 44/49
domain.model.wrapper	1	100% 11/11	100% 9/9	100% 9/9
domain.service	2	100% 125/125	95% 39/41	95% 39/41
service	2	100% 144/144	100% 25/25	100% 25/25
service.event	8	100% 48/48	97% 28/29	100% 28/28
service.mapper	2	100% 101/101	100% 14/14	100% 14/14
service.parser	3	100% 184/184	94% 66/70	96% 66/69
service.parser.model	7	100% 94/94	69% 31/45	84% 31/37
web.controller	2	100% 32/32	50% 2/4	100% 2/2

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
67	100% 1169/1169	96% 382/396	98% 382/389

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
config	3	100% 9/9	100% 3/3	100% 3/3
db.migration	5	100% 72/72	100% 28/28	100% 28/28
domain.model	6	100% 156/156	98% 50/51	98% 50/51
domain.model.value	22	100% 106/106	100% 50/50	100% 50/50
domain.model.wrapper	1	100% 11/11	100% 9/9	100% 9/9
domain.service	2	100% 134/134	100% 44/44	100% 44/44
service	2	100% 145/145	100% 25/25	100% 25/25
service.event	10	100% 60/60	100% 37/37	100% 37/37
service.mapper	2	100% 117/117	100% 15/15	100% 15/15
service.parser	4	100% 216/216	98% 78/80	99% 78/79
service.parser.model	8	100% 109/109	82% 41/50	89% 41/46
web.controller	2	100% 34/34	50% 2/4	100% 2/2

* Zahlen unterscheiden sich leicht, weil die Entwicklung ja nicht stillgestanden hat



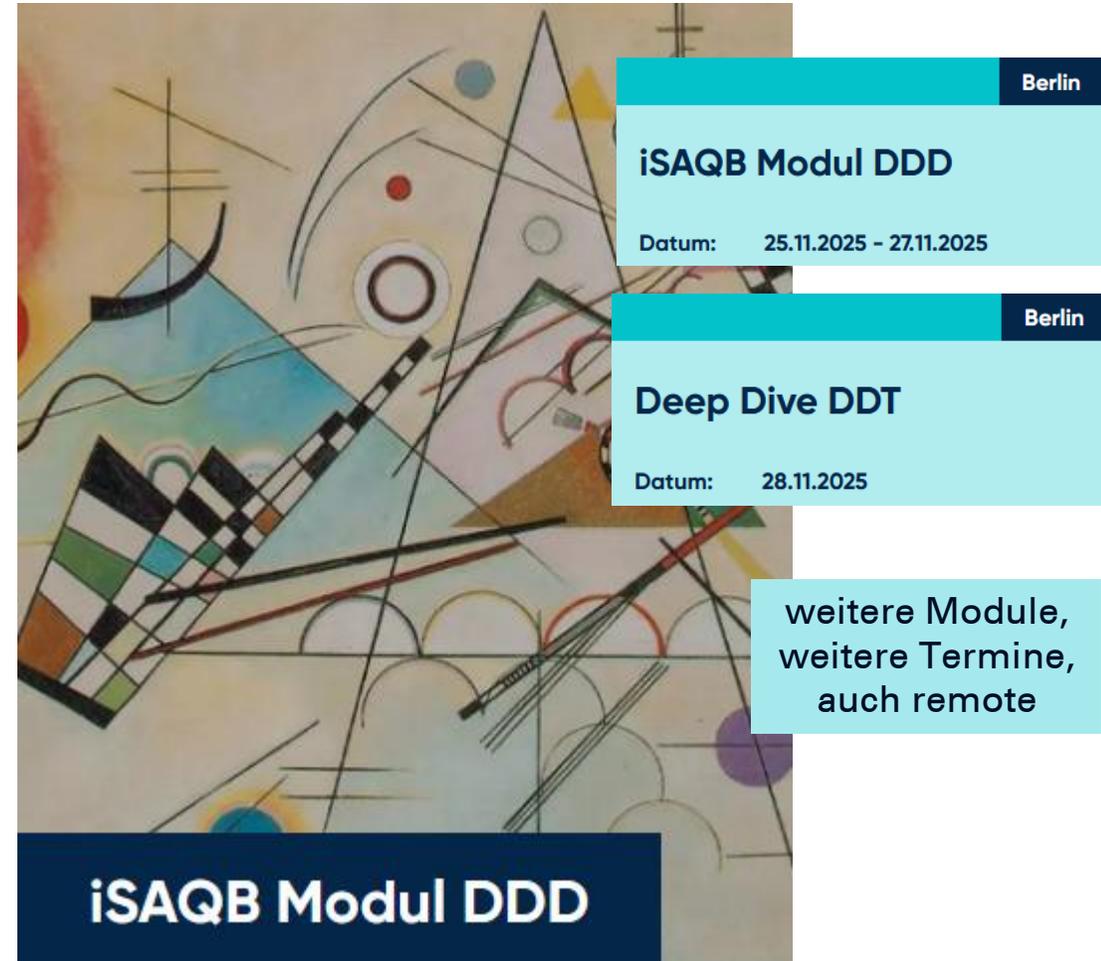
- ❖ Einbeziehen von Integrationstests
 - ❖ Brauchen im Gegensatz zu Unit-Tests sehr lange und bringen manchmal keinen Mehrwert
 - ❖ Müssen so implementiert werden, dass sie auch bei mehrfachem Lauf parallel ausgeführt werden können
- ❖ Reports
 - ❖ Mutation-Tests überhaupt ausführen und nicht in der Pipeline ignorieren
 - ❖ Richtig lesen, nicht nur auf die Zahlen starren
 - ❖ Problemstellen nicht ignorieren, sondern zeitnah beheben
- ❖ Im Backend oft leichter zu integrieren als im Frontend
 - ❖ Besonders bei Verwendung nicht so verbreiteter Technologien, bspw. NX-Workspace



- Machen 😊!
- Da es immer noch sehr rechenintensiv ist
 - Ein typischer Fall für einen Nightly- oder Weekly- oder Release-Build
 - Aber: Regelmäßiger Termin zum Auswerten der erstellten Reports
 - Tipp: Kein Deployment ohne ausreichende Mutation-Coverage
 - Notfalls Integrationstests ausschließen
 - Tipp: Nur bei Merge auf den Main-Branch werden alle Tests einbezogen
- Nicht nur für die Pipeline konfigurieren, sondern auch für lokale Entwicklung



Softwarearchitektur für KI



Domain-Driven Design konkret
Vertiefungstag Domain-Driven Transformation



❖ Mutationstest

- ❖ für JVM-basierte Sprachen: <https://pitest.org/>
- ❖ für Javascript / Typescript / C# / Scala: <https://stryker-mutator.io/>
- ❖ für Python: <https://github.com/boxed/mutmut>

❖ Code-Coverage

- ❖ für JVM-basierte Sprachen: <https://www.jacoco.org/>