

# **Tests und Code verbessern mit Mutation Testing**



**Hands-On Coding  
Developer Camp 2017  
Nürnberg**

# Beyond Code Coverage

- Unsere Tests prüfen unseren Code
- Wer prüft unsere Tests?
- Hohe Code Coverage => Hohe Güte der Tests?
- 100% Code Coverage, aber keine Assertions ...

# Mutation Testing

Einfache Grundidee:

- `SOLID` original
- Alle Tests grün als Voraussetzung
- `SOLD` mit Mutation
- Alle Tests immer noch grün => Mutation „überlebt“ 
- Mindestens ein Test rot => Mutation „getötet“ 
- Güte der Tests = Anzahl getöteter Mutationen / Anzahl aller Mutationen

# Beispiele für Mutatoren

- Relationale Operatoren ersetzen
  - Mit anderen Grenzen     `if (a >= b) => if (a > b)`
  - Durch das Gegenteil     `if (a >= b) => if (a < b)`
  - Durch Konstanten     `if (a >= b) => if (true)`
- Arithmetische Operatoren ersetzen
  - `a = b + c => a = b - c`
- Rückgabe-Werte verändern
  - `return x => return x != null ? null : throw new RuntimeException()`
- Anweisungen entfernen

# Probleme

- Alle Tests müssen für Original-Code grün sein
  - Nicht immer und überall selbstverständlich
- Endlosschleifen durch Mutation erzeugt
  - Abbruch durch Timeout für jeden Test
- Laufzeit der Tests im Rahmen halten
  - Nur bestimmte Tests ausführen
  - Zu mutierende Klassen und Tests einschränken
- Semantisch äquivalente Mutationen erkennen
  - Schwierig!

# Werkzeug-Auswahl

Java - PIT <http://pitest.org>

C# - Visual Mutator <https://visualmutator.github.io/web>

JavaScript – Stryker <https://stryker-mutator.github.io>

Ruby – Mutant <https://github.com/mbj/mutant>

PHP – Humbug <https://github.com/humbug/humbug>

Python – Cosmic Ray <http://cosmic-ray.readthedocs.io>

# Hands-On

## Übungsprojekt

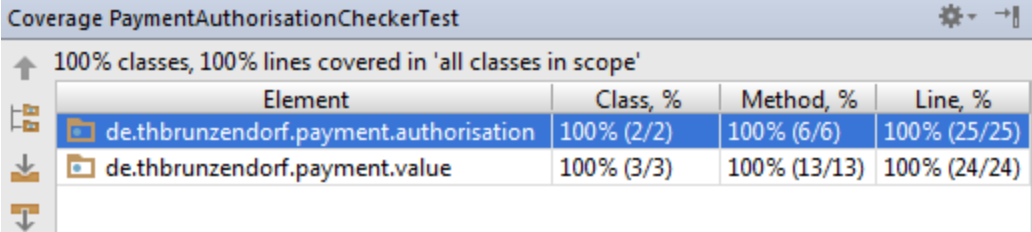
- Sprache: Java
- Werkzeug: PIT
- Domain: Payment Authorisation



<https://github.com/thbrunzendorf/mutation-testing-sample>

# Hands-On

- Code Coverage 100%



The screenshot shows a code coverage tool window titled "Coverage PaymentAuthorisationCheckerTest". It displays a summary of coverage for the test, indicating that 100% of classes and 100% of lines are covered. Below this, a table lists the elements being tested, their class coverage, method coverage, and line coverage.

Element	Class, %	Method, %	Line, %
de.thbrunzendorf.payment.authorisation	100% (2/2)	100% (6/6)	100% (25/25)
de.thbrunzendorf.payment.value	100% (3/3)	100% (13/13)	100% (24/24)

- mvn clean install
- mvn org.pitest:pitest-maven:mutationCoverage
- Ergebnis unter target/pit-reports/...



# Hands-On

## Pit Test Coverage Report

### Package Summary

**de.thbrunzendorf.payment.authorisation**

Number of Classes	Line Coverage	Mutation Coverage
2	100% <div>27/27</div>	56% <div>9/16</div>

### Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">PaymentAuthorisation.java</a>	100% <div>7/7</div>	100% <div>2/2</div>
<a href="#">PaymentAuthorisationChecker.java</a>	100% <div>20/20</div>	50% <div>7/14</div>

---

Report generated by [PIT](#) 1.2.0

# Hands-On

```
10 public class PaymentAuthorisationChecker {
11
12     public PaymentAuthorisation checkFor(Payment payment) {
13         PaymentAuthorisation paymentAuthorisation = new PaymentAuthorisation();
14         User initiator = payment.getInitiator();
15         Money amount = payment.getAmount();
16         Money limit = initiator.getLimit();
17         if (amount.compareTo(limit) <= 0) {
18             paymentAuthorisation.setApprovalNeeded(false);
19         }
20         if (amount.compareTo(limit) > 0) {
21             paymentAuthorisation.setApprovalNeeded(true);
22             User approver = getPrimaryApprover(initiator, amount);
23             paymentAuthorisation.setPrimaryApprover(approver);
24         }
25         return paymentAuthorisation;
26     }
27
28     private User getPrimaryApprover(User initiator, Money amount) {
29         User supervisor = initiator.getSupervisor();
30         Money limit = supervisor.getLimit();
31         int maxIterations = 10; // preventing infinite loops
32         for (int i = 0; i < maxIterations; i++) {
33             if (amount.compareTo(limit) > 0) {
34                 supervisor = supervisor.getSupervisor();
35                 limit = supervisor.getLimit();
36             }
37         }
38         return supervisor;
39     }
40 }
```

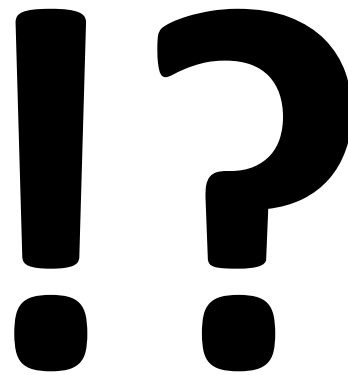
## Mutations

```
17 1. changed conditional boundary → SURVIVED
   2. negated conditional → SURVIVED
18 1. removed call to de/thbrunzendorf/payment/authorisation/PaymentAuthorisation::setApprovalNeeded → SURVIVED
20 1. changed conditional boundary → SURVIVED
   2. negated conditional → KILLED
21 1. removed call to de/thbrunzendorf/payment/authorisation/PaymentAuthorisation::setApprovalNeeded → KILLED
23 1. removed call to de/thbrunzendorf/payment/authorisation/PaymentAuthorisation::setPrimaryApprover → KILLED
25 1. mutated return of Object value for de/thbrunzendorf/payment/authorisation/PaymentAuthorisationChecker::checkFor to ( if (x != null) null else throw new RuntimeException ) → KILLED
32 1. changed conditional boundary → SURVIVED
   2. Changed increment from 1 to -1 → TIMED_OUT
   3. negated conditional → SURVIVED
33 1. changed conditional boundary → SURVIVED
   2. negated conditional → KILLED
38 1. mutated return of Object value for de/thbrunzendorf/payment/authorisation/PaymentAuthorisationChecker::getPrimaryApprover to ( if (x != null) null else throw new RuntimeException ) → KILLED
```

# Hands-On

**Kill the mutants!**

# Diskussion



# Mutation Testing?

- Eine gute Code Coverage ist notwendige, aber nicht hinreichende Bedingung für gute Tests
- Mutation Testing hilft, fehlende Testfälle zu finden
  - Zusätzliche Testdaten, z.B. Grenzwerte
  - Zusätzliche Assertions
- Mutation Testing hilft, überflüssigen Code zu finden
  - Fallen uns keine Testfälle ein, die eine Mutation töten, ist der mutierte Code wahrscheinlich redundant
- Pro Tip: Testgetriebene Entwicklung reduziert die Überlebenschancen von Mutanten drastisch

# Fragen?

Thorsten Brunzendorf  
@thbrunzendorf

## Vielen Dank!