

Tests und Code verbessern mit Mutation Testing



Modern Software Development
Würzburg

Gute Software

- Software-Qualität?
- *Quality is value to some person (who matters)*
- Wie können wir diesen Wert sicherstellen, auch wenn wir unsere Software immer wieder ändern?

Gute Tests

- Kontinuierlich durchgeführte Tests
- Agile Testing Quadrants
- Checking vs Exploring
- Basis: Gute Unit-Tests
- A-TRIP
- FIRST
- Wieviele Tests sind genug?

Code Coverage

- Soviele Tests, dass möglichst viel Code durchlaufen wird
- Ausreichend hohe Test-Abdeckung
- Wie hoch?
- 80 Prozent?
- 100 Prozent?
- Quality Gates?



Thierry de Pauw

@tdpauw

Folgen



Team has a 30% code coverage. After running mutation tests, drops to 3% coverage.
When a metric becomes a target you fool it.
Goodhart's Law

00:30 - 26. Juli 2017

29 Retweets 34 „Gefällt mir“-Angaben



3



29



34

<https://github.com/thinkinglabs/the-100percent-test-coverage-fallacy>

<https://martinfowler.com/bliki/AssertionFreeTesting.html>

Mutation Testing Grundidee

- Nehme eine kleine Code-Änderung vor:

Mutation

- Führe alle Tests aus
- Alle Tests immer noch grün
=> Mutation „überlebt“
- Mindestens ein Test rot
=> Mutation „getötet“
- Wiederhole



Mutationen



- Mutationen: kleine Veränderungen
- Mutations-Operatoren => Mutatoren

Beispiele für Mutatoren

Logische 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)

Beispiele für Mutatoren

Arithmetische Operatoren ersetzen

$a = b + c \Rightarrow a = b - c$

$a = b * c \Rightarrow a = b / c$

$a++ \Rightarrow a--$

Beispiele für Mutatoren

Rückgabe-Werte verändern

return **true** => return **false**

return **new Object()** => return **null**

return **null** => **throw new RuntimeException()**

Beispiele für Mutatoren

Anweisungen oder Blöcke entfernen

`doSomething()` => ~~`doSomething()`~~

Herausforderungen

Alle Tests grün

- Voraussetzung: Alle Tests müssen für den Original-Code erfolgreich durchlaufen
- Sonst fällt ein weiterer fehlschlagender Test nicht auf
- Nicht immer und überall selbstverständlich!

Herausforderungen

Endlosschleifen abbrechen

- Durch eine Mutation können Endlosschleifen erzeugt werden
- Häufige Strategie
 - Abbruch durch Timeout für den jeweiligen Test

Herausforderungen

Laufzeit im Rahmen halten

- Theoretische Gesamt-Laufzeit = Anzahl Mutationen * Laufzeit Gesamte Test-Suite
- Strategien
 - Zu mutierende Klassen einschränken (etwa nach Risiko oder nur bei aktuellen Änderungen)
 - Keine Tests ausführen für Mutationen in einer Codezeile ohne Testabdeckung
 - Nur bestimmte Tests ausführen

Herausforderungen

Semantisch äquivalente Mutationen erkennen

- Beispiele:
 - Klassische Refactorings (sind deshalb keine guten Kandidaten für Mutationen!)
 - Löschen einer Codezeile, die nichts tut
- Möglichkeit, „uninteressante“ Mutationen auszuschließen (z.B. von Log-Anweisungen)
- Generell: Schwierig zu erkennen!

Kennzahlen

- Mutationen in Kategorien einteilen
 - Entdeckt = {Getötet, Timeout, Fehler}
 - Nicht entdeckt = {Überlebt, Nicht abgedeckt}
- Effektivität der Testsuite: **Mutation Coverage**
 - Anzahl Entdeckt / Anzahl Gesamt
- Effektivität der vorhandenen Tests
 - Anzahl Entdeckt /
(Anzahl Gesamt – Anzahl Nicht abgedeckt)

Werkzeuge: Java und JVM

PIT

<http://pitest.org>

Aktive Weiterentwicklung

Minimum Java 7, Unterstützung bis Java 9

Interne und externe Plugins

Beispiel JUnit 5 Plugin



Werkzeuge: JavaScript

Stryker

<http://stryker-mutator.io>

Für JavaScript und TypeScript

Aktive Weiterentwicklung

Plugin-Architektur

Beispiel Transpiler-Plugins



Werkzeuge: C# und .NET

Visual Mutator

<https://visualmutator.github.io/web>

Ninja Turtles

<http://www.mutation-testing.net>

Seit mehreren Jahren keine Weiterentwicklung

Aktuell keine Empfehlung

Werkzeuge: C# und .NET



Seb Rose

@sebrose

Folgen



Please RT:
Any CLR gurus out there with an interest in
mutation testing?
Contact me.
It's time to create pitest for .NET

07:50 - 1. Juni 2017

33 Retweets **9** „Gefällt mir“-Angaben



Werkzeuge: PHP

~~Humbug~~

<https://github.com/humbug/humbug>

31.12.2017  This package is deprecated, check out [Infection](#) instead.

Infection

<https://infection.github.io>



Werkzeuge: Überblick

Java – PIT <http://pitest.org>

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

C# – aktuell keine Empfehlung

PHP – Infection <https://infection.github.io>

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

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

Hands-On

Sample project

- Domain: Payment Authorisation
- Users have limits for payments they initiate
- If the amount of a payment is above this limit then a supervisor has to approve the payment
- Help us testing the
PaymentAuthorisationChecker

Hands-On

- Sprache: Java
- Werkzeug: PIT



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

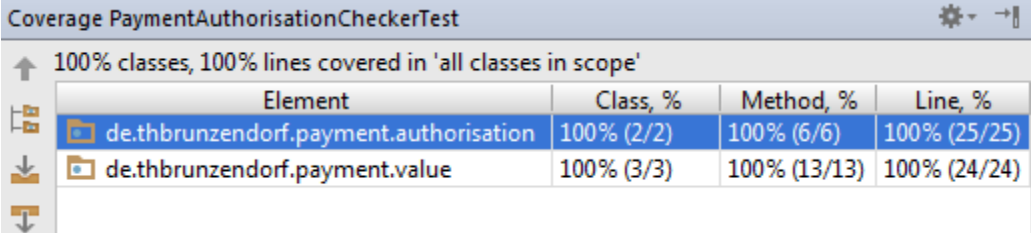
- Sprache: JavaScript
- Werkzeug: Stryker



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

Hands-On PIT

- Basis Code Coverage 100%



The screenshot shows the Coverage tool interface for the test `PaymentAuthorisationCheckerTest`. It displays a summary of 100% classes and 100% lines covered. Below this, a table lists the covered elements with their respective class, method, and line coverage percentages.

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)

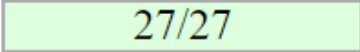
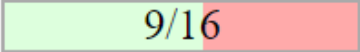
- mvn clean install
- mvn pitest:mutationCoverage
- Ergebnis unter target/pit-reports/...

Hands-On PIT

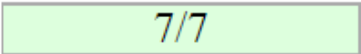
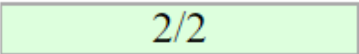
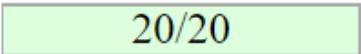
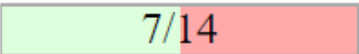
Pit Test Coverage Report

Package Summary

de.thbrunzendorf.payment.authorisation

Number of Classes	Line Coverage	Mutation Coverage
2	100% 	56% 

Breakdown by Class

Name	Line Coverage	Mutation Coverage
PaymentAuthorisation.java	100% 	100% 
PaymentAuthorisationChecker.java	100% 	50% 

Report generated by [PIT](#) 1.3.2

Hands-On PIT

```
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
17         1. checkFor : changed conditional boundary → SURVIVED
18         2. checkFor : negated conditional → SURVIVED
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
17 2. negated conditional → SURVIVED
18 1. removed call to de/thbrunzendorf/payment/authorisation/PaymentAuthorisation::setApprovalNeeded → SURVIVED
20 1. changed conditional boundary → SURVIVED
20 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
32 2. Changed increment from 1 to -1 → TIMED_OUT
33 3. negated conditional → SURVIVED
33 1. changed conditional boundary → SURVIVED
33 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 Stryker

- Basis Code Coverage 100%

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	100	100	100	100	
money.js	100	100	100	100	
payment.js	100	100	100	100	
paymentAuthorisation.js	100	100	100	100	
paymentAuthorisationChecker.js	100	100	100	100	
user.js	100	100	100	100	

- stryker run
- Ergebnis unter `reports/mutation/html/...`

Hands-On Stryker



All files - Stryker report

All files

File / Directory	Mutation score									Total detected	Total undetected	Total mutants
			# Killed	# Survived	# Timeout	# No coverage	# Runtime errors	# Transpile errors				
All files	<div><div>48.57 %</div></div>	48.57	16	18	1	0	0	0	17	18	35	
money.js	<div><div>57.14 %</div></div>	57.14	4	3	0	0	0	0	4	3	7	
payment.js	<div><div>100.00 %</div></div>	100.00	1	0	0	0	0	0	1	0	1	
paymentAuthorisation.js	<div><div>0.00 %</div></div>	0.00	0	2	0	0	0	0	0	2	2	
paymentAuthorisationChecker.js	<div><div>45.83 %</div></div>	45.83	10	13	1	0	0	0	11	13	24	
user.js	<div><div>100.00 %</div></div>	100.00	1	0	0	0	0	0	1	0	1	

Generated with stryker-html-reporter generator. Visit the [Stryker website](#)

paymentAuthorisationChecker.js - Stryker report



[All files](#) / paymentAuthorisationChecker.js

File / Directory	Mutation score	# Killed	# Survived	# Timeout	# No coverage	# Runtime errors	# Transpile errors	Total detected	Total undetected	Total mutants
paymentAuthorisationChecker.js	45.83 %	10	13	1	0	0	0	11	13	24

☒ Survived (13) ☒ Killed (10) ☒ TimedOut (1) [Expand all](#)

```
var Payment = require('../src/payment.js');
var PaymentAuthorisation = require('../src/paymentAuthorisation.js');
function checkFor(payment) {
  let paymentAuthorisation = new PaymentAuthorisation();
  let initiator = payment.initiator;
  let amount = payment.amount;
  let limit = initiator.limit;
  if (1 false 2 3 4 amount.compareTo(limit) <= 0) 5 {
    paymentAuthorisation.approvalNeeded = 6 false;
  }
  if (7 8 9 10 amount.compareTo(limit) > 0) 11 {
    paymentAuthorisation.approvalNeeded = 12 true;
    let approver = getPrimaryApprover(initiator, amount);
    paymentAuthorisation.primaryApprover = approver;
  }
  return paymentAuthorisation;
}
function getPrimaryApprover(initiator, amount) 13 {
  let supervisor = initiator.supervisor;
  let limit = supervisor.limit;
  const maxIterations = 10; // preventing infinite loops
  for (let i = 0; 14 15 16 i < maxIterations; 17 i++) 18 {
    if (19 20 21 22 amount.compareTo(limit) > 0) 23 {
      supervisor = supervisor.supervisor;
      limit = supervisor.limit;
    }
  }
}
```

#	Mutator	State	Location	Original	Replac
0	Block	Killed	2 : 27	{ ... ; }	{ }
1	IfStatement	Survived	7 : 8) <=	
2	IfStatement	Survived	7 : 8) <=	
3	BinaryExpression	Survived	7 : 8) <=
4	BinaryExpression	Survived	7 : 8) <=
5	Block	Survived	7 : 38	{ ... }	{ }
6	BooleanSubstitution	Killed	8 : 46		
7	IfStatement	Killed	10 : 8) >	
8	BinaryExpression	Survived	10 : 8) >
9	IfStatement	Killed	10 : 8) >	



Diskussion



Mutation Testing?

Eine gute Code Coverage ist

- eine notwendige
- aber nicht hinreichende

Bedingung für gute Tests

Mutation Testing?

Mutation Testing hilft, Lücken in den Tests zu finden

- Zusätzliche Tests mit anderen Daten, z.B. Grenzwerte
- Zusätzliche Assertions

Mutation Testing?

Mutation Testing hilft, überflüssigen Code zu finden

- Fallen uns keine Testfälle ein, die eine Mutation entdecken, ist der mutierte Code wahrscheinlich redundant

Mutation Testing?

Testgetriebene Entwicklung

- Produktiven Code nur schreiben bzw. ändern, wenn zuvor ein „Test“ als Spezifikation erstellt worden ist und fehlschlägt
- Reduziert die Überlebenschancen von Mutanten drastisch

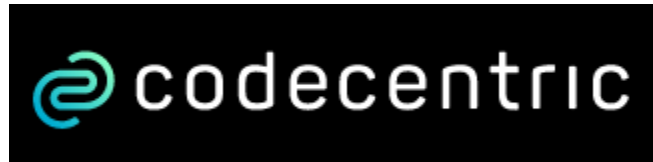
Mutation Testing?

Mit Mutation Testing können wir die
Effektivität
von Tests verbessern

Effektive Tests sind dort am wichtigsten,
wo die höchsten **Risiken** liegen

Kontakt

Thorsten Brunzendorf



@thbrunzendorf



thorsten.brunzendorf@codecentric.de

Vielen Dank!

Quellen

<http://secretsofconsulting.blogspot.de/2012/09/agile-and-definition-of-quality.html>
<https://www.a1qa.com/blog/interview-with-michael-bolton-part-1/>
<https://media.pragprog.com/titles/utj/StandaloneSummary.pdf>
<https://pragprog.com/magazines/2012-01/unit-tests-are-first>
<https://twitter.com/tdpauw/status/890112157450481664>
<https://github.com/thinkinglabs/the-100percent-test-coverage-fallacy>
<https://martinfowler.com/bliki/AssertionFreeTesting.html>
<https://pixabay.com/de/genmanipulation-mutant-mutation-549889>
<https://pixabay.com/de/katze-auge-ungerade-518306>
<http://pitest.org>
<http://stryker-mutator.io>
<https://visualmutator.github.io/web>
<http://www.mutation-testing.net>
<https://twitter.com/sebrose/status/870291435215630336>
<https://github.com/humbug/humbug>
<https://infection.github.io>
<https://github.com/mbj/mutant>
<http://cosmic-ray.readthedocs.io>

Abstract

Tests und Code verbessern mit Mutation Testing

Gute Software bleibt nachhaltig gut, wenn sie kontinuierlich gut getestet werden kann. Die Testabdeckung (Code Coverage) als Maß für die Güte einer Testsuite ist aber eine trügerische Kennzahl. Das Prinzip des "Mutation Testing" ist eine bessere Möglichkeit, die Effektivität von Tests nachzuweisen. Zunächst stelle ich die Idee des Mutation Testing sprach- und werkzeugunabhängig vor und gehe auch auf die Herausforderungen dabei ein. Es folgt ein Überblick über aktuelle Werkzeuge für gängige Sprachen bzw. Plattformen. In einer Hands-On Coding Session demonstriere ich dann den praktischen Umgang mit zwei dieser Tools, nämlich PIT für JVM-basierte Sprachen und Stryker für die JS-Welt und zeige konkret anhand eines kleinen Beispiels, wie wir damit Lücken in unseren Tests und auch Unschärfen im Produktivcode finden können.

Thorsten Brunzendorf

Consultant und Entwickler im Nürnberger Team der codecentric AG