



Mutation Testing

Michael Inden

Freiberuflicher Consultant, Buchautor und Trainer

Coding Guidelines – Agenda



- Mutation Testing im Kurzüberblick
 - PiTest installieren und nutzen
 - Weitere Infos
-



Mutation Testing im Kurzübersicht



Was ist Mutation Testing? Warum ist es hilfreich?

- Wir haben JUnit 5 und AssertJ zum Schreiben verständlicher, wartbarer Tests kennengelernt
 - Zudem können wir mit verschiedenen Tools die Code Coverage analysieren
 - Was fehlt?
 - Woher wissen wir eigentlich, dass die Tests auch (alle) Fehler aufdecken?
-



Wir wollen schauen, ob die Tests das richtige Testen

- Dazu wird der Sourcecode an gewissen Stellen mutiert und geschaut, ob die Tests diese Änderungen aufdecken
- Entdecken die Tests die Änderung, dann spricht man davon, dass die Mutation gekilled wurde, ansonsten ist die Mutation live und man sollte die Tests korrigieren oder ergänzen
- Grenzwerte: etwa $<$ durch $<=$ ersetzen

Original	Mutation
$<$	$<=$
$<=$	$<$
$>$	$>=$
$>=$	$>$



- Rückgabewerte werden gemäß der Tabelle mutiert, etwa false -> true, true -> false

Original	Mutation
Boolean value	!value
Int, short, byte value	0 -> 1, alle anderen Werte -> 0
Long value	Value + 1
Float, double	Value + 1.0, Spezialfall NaN -> 0



PiTest



PiTest hat gegenüber anderen Werkzeugen einige Vorteile:

- Freier Download unter <https://pitest.org/>
 - Es wird aktiv weiterentwickelt
 - im Vergleich zu früheren Generationen von Tools um ein Vielfaches schneller
 - Dadurch endlich praxistauglich

 - leichte Integrierbarkeit in Build-Tools und Entwicklungsumgebungen
 - ein aktiv entwickeltes [SonarQube-Plug-in](#)
 - umfangreiche Möglichkeiten zur Konfiguration
 - unterstützt inkrementelle Analysen
-

PiTest Maven Dependencies



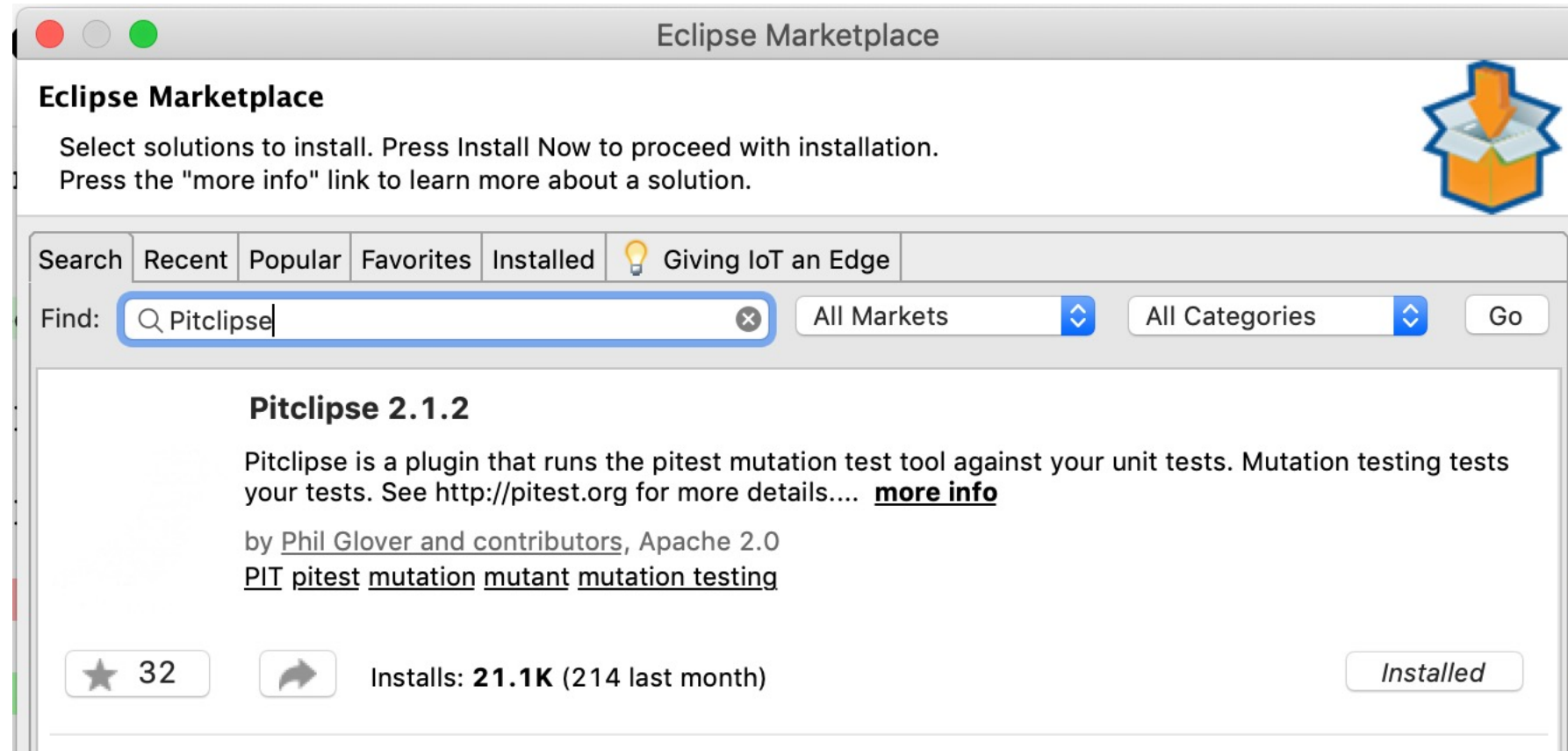
```
<plugin>
  <groupId>org.pitest</groupId>
  <artifactId>pitest-maven</artifactId>
  <version>1.6.6</version>
  <dependencies>
    <dependency>
      <groupId>org.pitest</groupId>
      <artifactId>pitest-junit5-plugin</artifactId>
      <version>0.14</version>
    </dependency>
  </dependencies>
</plugin>
```

```
mvn org.pitest:pitest-maven:mutationCoverage
```

PiTest Eclipse Plugin



Installation über Eclipse Marketplace



Beispiel



```
class SpecialCounter
{
    private int count;

    public void countIfHundredOrAbove(final int value)
    {
        if (value >= 100)
        {
            count++;
        }
    }

    public void reset()
    {
        count = 0;
    }

    public int currentCount()
    {
        return count;
    }
}
```

Problem «Code Coverage 100 %»



```
public class SpecialCounterTest
{
    // VERY BAD TEST ... 100% Coverage, aber KEINE semantische Prüfung
    @Test
    @DisplayName("Boss says he wants 100% coverage. Here you go!")
    public void veryBadTrickyAssertNothing()
    {
        SpecialCounter counter = new SpecialCounter();

        counter.reset();
        counter.countIfHundredOrAbove(111);
        counter.countIfHundredOrAbove(99);

        counter.currentCount();
    }
}
```

**«WIR KÖNNEN UNS NICHT AUF DIE
CODE COVERAGE VERLASSEN!»**

**SIE ZEIGT UNS NICHT DEN CODE DER
GETESTET WURDE, SONDERN NUR
DER DABEI DURCHLAUFEN WURDE!**

Abhilfe sinnvolle Testfälle



```
@Test
void startsWithEmptyCount()
{
    SpecialCounter counter = new SpecialCounter();

    assertEquals(0, counter.currentCount());
}
```

```
@Test
void countsLargeNumbersCorrectly()
{
    SpecialCounter counter = new SpecialCounter();

    counter.countIfHundredOrAbove(111);
    counter.countIfHundredOrAbove(333);

    assertEquals(2, counter.currentCount());
}
```

Abhilfe sinnvolle Testfälle



```
@Test
void doesNotCountIntegersBelowHundred()
{
    SpecialCounter counter = new SpecialCounter();

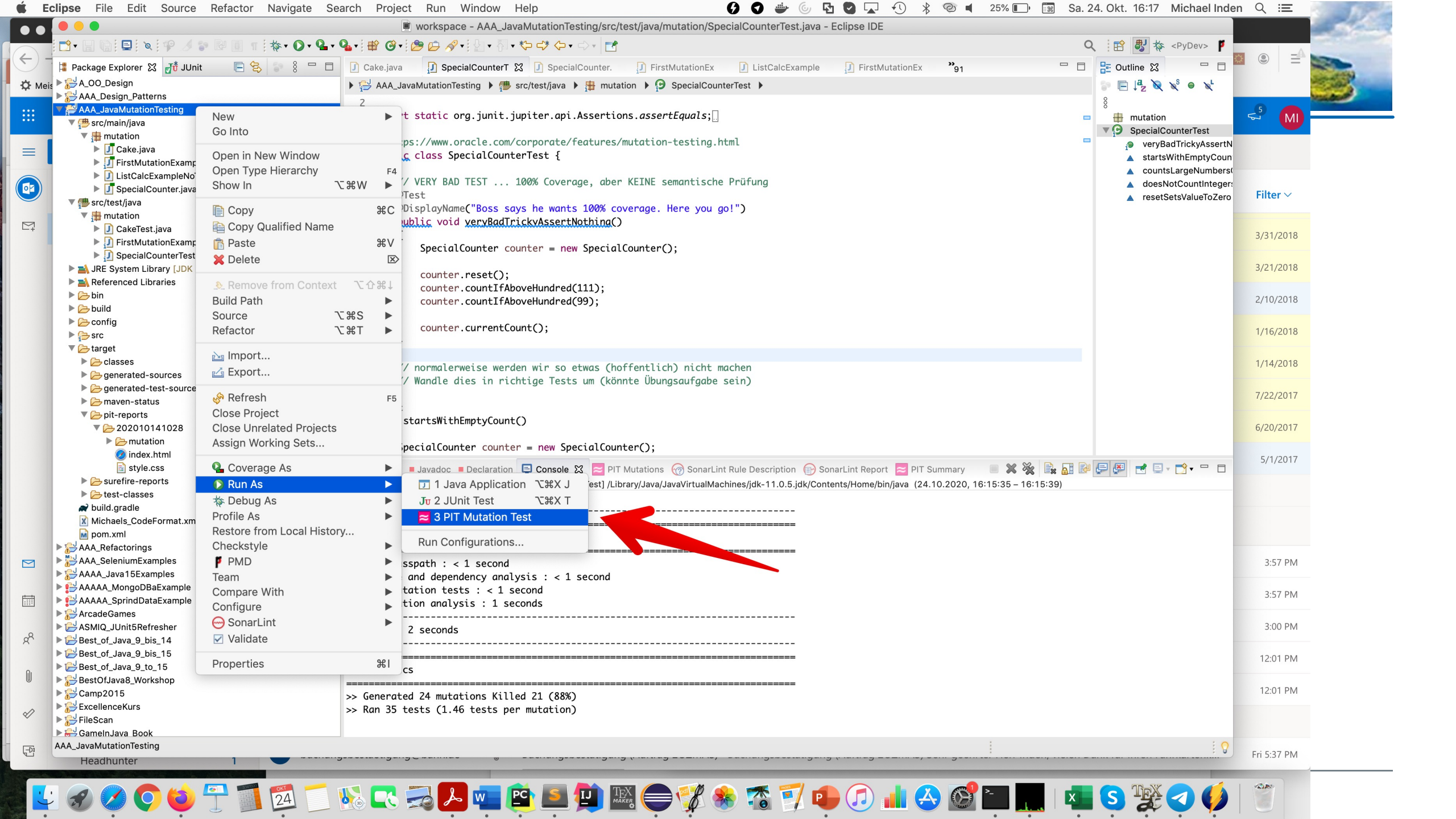
    counter.countIfHundredOrAbove(77);

    assertEquals(0, counter.currentCount());
}
```










```
@Test
void resetSetsValueToZero()
{
    SpecialCounter counter = new SpecialCounter();

    counter.countIfHundredOrAbove(420);
    counter.reset();

    assertEquals(0, counter.currentCount());
}
```





 Problems  Javadoc  Declaration  Console  PIT Mutations  SonarLint Rule Description  SonarLint Report  PIT Summary 

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
3	98% <div><div>41/42</div></div>	88% <div><div>21/24</div></div>

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
mutation	3	98% <div><div>41/42</div></div>	88% <div><div>21/24</div></div>

Report generated by [PIT](#) 1.4.11



Problems Javadoc Declaration Console PIT Mutations SonarLint Rule Description SonarLint Report PIT Summary

Pit Test Coverage Report

Package Summary

mutation

Number of Classes	Line Coverage	Mutation Coverage
3	98% 41/42	88% 21/24

Breakdown by Class

Name	Line Coverage	Mutation Coverage
Cake.java	100% 31/31	94% 15/16
FirstMutationExample.java	75% 3/4	75% 3/4
SpecialCounter.java	100% 7/7	75% 3/4

Report generated by [PIT](#) 1.4.11



```
4 class SpecialCounter
5 {
6     private int count;
7
8     public void countIfHundredOrAbove(final int value)
9     {
10         if (value >= 100)
11         {
12             count++;
13         }
14     }
15
16     public void reset()
17     {
18         count = 0;
19     }
20
21     public int currentCount()
22     {
23         return count;
24     }
25 }
```

Mutations

- 10 1. changed conditional boundary → SURVIVED
- 2. negated conditional → KILLED
- 12 1. Replaced integer addition with subtraction → KILLED
- 23 1. replaced int return with 0 for mutation/SpecialCounter::currentCount → KILLED



// MUTATION TESTING FINDET DEN FEHLENDEN TEST FÜR DIE GRENZE

```
@Test
void countsIntegersCorrectlyForMinimumOf100()
{
    SpecialCounter counter = new SpecialCounter();
    counter.countIfHundredOrAbove(100);

    assertEquals(1, counter.currentCount());
}
```

```

4 class SpecialCounter
5 {
6     private int count;
7
8     public void countIfHundredOrAbove(final int value)
9     {
10         if (value >= 100)
11         {
12             count++;
13         }
14     }
15
16     public void reset()
17     {
18         count = 0;
19     }
20
21     public int currentCount()
22     {
23         return count;
24     }
25 }

```



Mutations

- [10](#) 1. changed conditional boundary → KILLED
2. negated conditional → KILLED
- [12](#) 1. Replaced integer addition with subtraction → KILLED
- [23](#) 1. replaced int return with 0 for mutation/SpecialCounter::currentCount → KILLED

Weitere Infos



- <https://pitest.org/quickstart/>
 - <https://www.codeaffine.com/2015/10/05/what-the-heck-is-mutation-testing/>
 - <https://jaxenter.de/mutant-testing-pit-java-84437>
 - <https://www.baeldung.com/java-mutation-testing-with-pitest>
 - <https://blog.scottlogic.com/2017/09/25/mutation-testing.html>
 - <https://www.oracle.com/corporate/features/mutation-testing.html>
 - <https://dzone.com/articles/mutation-testing-covering-your-code-with-right-tes>
 - <https://dzone.com/articles/mutation-testing-covering-your-code-with-right-tes-1>
-



Thank You
