Mutation Testing as a Safety Net for Test Code Refactoring

Ali Parsai
Alessandro Murgia
Quinten D. Soetens
Serge Demeyer
“Refactoring is changing a system to improve its internal structure without altering its external behavior.” – Fowler et al.
Production Code
Does refactoring preserve the behavior?

Perform refactoring
Run the test suite
Do all tests pass?
Perform refactoring

Test Code

Does refactoring preserve the behavior?
Solution

A metric to verify the preservation of the test suite behavior pre- and post- refactoring
Branch Coverage

Options

Mutation Coverage
Branch Coverage

Block 1

CS 1

Block 2

CS 2

Block 4

Block 5

Block 6

Block 3

CS 3

Block 7

Not covered
Covered

33%
Mutation Coverage

Procedure

Example

Test Suite

Software

Mutant Generation

Mutant

Test Execution

Survived

Killed

if (x > 0 && y <= 0) x++;

if (x <= 0 && y <= 0) x++;

if (x > 0 && y >= 0) x++;

if (x > 0 || y <= 0) x++;

if (x > 0 && y <= 0) x--;
Branch Coverage

+ Can be calculated fast
+ Very good tool support — Easily integrable
– Not accurate for defining behavior
Mutation Coverage

+ More accurate
+ Adaptable by customizing the fault model
  – Computationally intensive
  – Poor tool support, hard to integrate
Works on Java source code

Works regardless of build system and testing framework
Main

JavaRead
JavaParse
JavaMutate
Report Generator

Antlr-generated Java Parser

Customized Antlr4 Runtime
## Project Summary

<table>
<thead>
<tr>
<th>Number of Files</th>
<th>Mutation Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>62.0%</td>
</tr>
</tbody>
</table>

## Breakdown by File

<table>
<thead>
<tr>
<th>Name</th>
<th>Mutation Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>./main/java/com/addthis/codec/binary/BufferIn.java</td>
<td>0.0% 0/2</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/binary/BufferOut.java</td>
<td>0.0% 0/2</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/binary/CodecBin2.java</td>
<td>86.4% 89/103</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/config/ConfigNodeCursor.java</td>
<td>81.2% 13/16</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/config/ConfigTraversingParser.java</td>
<td>53.3% 16/30</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/config/Configes.java</td>
<td>32.6% 14/43</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CaseIgnoringEnumDeserializer.java</td>
<td>28.6% 2/7</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecBeanDeserializer.java</td>
<td>60.0% 6/10</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecBeanDeserializerModifier.java</td>
<td>50.0% 2/4</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecDeserializers.java</td>
<td>50.0% 1/2</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecIntrospector.java</td>
<td>100.0% 14/14</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecJackson.java</td>
<td>80.0% 4/5</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecTypeDeserializer.java</td>
<td>76.9% 20/26</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecType1dResolver.java</td>
<td>42.9% 3/7</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/CodecTypeResolverBuilder.java</td>
<td>100.0% 3/3</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/IndexReportingObjectArrayDeserializer.java</td>
<td>80.0% 8/10</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/Jackson.java</td>
<td>21.7% 5/23</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/KeyRecordingMapDeserializer.java</td>
<td>21.9% 7/32</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/MissingPropertyException.java</td>
<td>100.0% 1/1</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/jackson/UnderscorePropertyIgnorer.java</td>
<td>100.0% 1/1</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/plugins/PluginMap.java</td>
<td>95.1% 27/29</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/plugins/PluginRegistry.java</td>
<td>33.3% 1/3</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/plugins/Plugins.java</td>
<td>0.0% 0/1</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/reflection/CodableClassInfo.java</td>
<td>69.2% 9/13</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/reflection/CodableFieldInfo.java</td>
<td>47.8% 33/69</td>
</tr>
<tr>
<td>./main/java/com/addthis/codec/reflection/Fields.java</td>
<td>80.0% 16/20</td>
</tr>
</tbody>
</table>
LittleDarwin File Report

File Summary

<table>
<thead>
<tr>
<th>Number of Mutants</th>
<th>Mutation Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>81.2%</td>
</tr>
</tbody>
</table>

Detailed List

<table>
<thead>
<tr>
<th>Survived Mutant</th>
<th>Build Output</th>
<th>Killed Mutant</th>
<th>Build Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.java</td>
<td>1.txt</td>
<td>2.java</td>
<td>2.txt</td>
</tr>
<tr>
<td>4.java</td>
<td>4.txt</td>
<td>3.java</td>
<td>3.txt</td>
</tr>
<tr>
<td>14.java</td>
<td>14.txt</td>
<td>5.java</td>
<td>5.txt</td>
</tr>
<tr>
<td>6.java</td>
<td>6.txt</td>
<td>7.java</td>
<td>7.txt</td>
</tr>
<tr>
<td>8.java</td>
<td>8.txt</td>
<td>9.java</td>
<td>9.txt</td>
</tr>
<tr>
<td>10.java</td>
<td>10.txt</td>
<td>11.java</td>
<td>11.txt</td>
</tr>
<tr>
<td>12.java</td>
<td>12.txt</td>
<td>13.java</td>
<td>13.txt</td>
</tr>
<tr>
<td>15.java</td>
<td>15.txt</td>
<td>16.java</td>
<td>16.txt</td>
</tr>
</tbody>
</table>
Experiments

- Mutation testing tool $\rightarrow$ LittleDarwin
- Branch coverage tool $\rightarrow$ JaCoCo

- Toy example $\rightarrow$ Salary
- Real project $\rightarrow$ AddThis Codec
Toy Example

Salary class diagram
Toy Example

```java
package salary;

public class Engineer extends Employee {
    Engineer()
    {
        super();
    }
}

package salary;

public class Salesman extends Employee {
    Salesman()
    {
        super();
    }
    @Override
    int payAmount()
    {
        return defaultSalary - extra;
    }
}

package salary;

public class Manager extends Employee {
    Manager()
    {
        super();
    }
    @Override
    int payAmount()
    {
        return defaultSalary + extra;
    }
}
```
Toy Example

```java
public class salaryTest {
    boolean checkPayment(Employee x) {
        String type = x.getClass().getName();
        switch (type) {
            case "salary.Engineer":
                if (2000 <= x.payAmount()) return true;
                break;
            case "salary.Manager":
                if (2500 <= x.payAmount()) return true;
                break;
            case "salary.Salesman":
                if (1500 <= x.payAmount()) return true;
                break;
        }
        return false;
    }

    @Test
    public void salaryEmployeeTest() {
        Employee e = new Engineer(), m = new Manager(), s = new Salesman();

        try {
            assertTrue(checkPayment(e));
            assertTrue(checkPayment(m));
            assertTrue(checkPayment(s));
            System.out.println("Test Passed!");
        } catch (AssertionError err) {
            System.out.println("Test Failed!");
            fail();
        }
    }
}
```
Toy Example

```java
public class salaryTest {

  @Test
  public void salaryEngineerTest() {
    Employee e = new Engineer();
    try {
      assertTrue(2000 <= e.payAmount());
      System.out.println("EngineerTest Passed!");
    } catch (AssertionError err) {
      System.out.println("EngineerTest Failed!");
      fail();
    }
  }

  @Test
  public void salarySalesmanTest() {
    Employee s = new Salesman();
    try {
      assertTrue(1500 <= s.payAmount());
      System.out.println("SalesmanTest Passed!");
    } catch (AssertionError err) {
      System.out.println("SalesmanTest Failed!");
      fail();
    }
  }

  @Test
  public void salaryManagerTest() {
    Employee m = new Manager();
    try {
      // Erroneous assertion
      assertTrue(1500 <= m.payAmount());
      System.out.println("ManagerTest Passed!");
    } catch (AssertionError err) {
      System.out.println("ManagerTest Failed!");
      fail();
    }
  }
}
```

Post-Refactoring
Toy Example

- Branch coverage and percentage of passed tests remain the same
- Mutation coverage changes
Toy Example

```java
public class Manager extends Employee {
    Manager()
    {
        super();
    }

    @Override
    int payAmount()
    {
        return defaultSalary + extra;
    }
}
```

The mutant that makes a difference can be traced back to the problematic test
Real Project

Found a version with refactorings in the test code

Identified the refactorings using RefFinder

Backported the changes in the test code to the previous version
Real Project
Real Project

Mutation Coverage remains the same for all classes
Limitations

One can catch smaller fishes with a tighter net
Conclusion

- Mutation testing can act as a safety net for test suite refactoring by detecting behavior changes.

- Mutation testing cannot guaranty the preservation of the test suite behavior in every situation.

- This method can be used to verify automated refactoring tools.
Does refactoring preserve the behavior?

Perform refactoring

Test Code

Does the mutation coverage stay the same?

Little DARWIN
pre- and post-refactoring