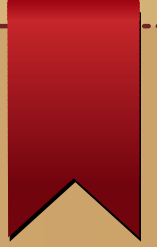
Mutation Testing

A 30-ish minute presentation

Contents



- Foreword
- General terms
- Types of code mutations
- Problems
- PIT
- Summary
- Discussion

Foreword

- Epic Battle: Zombies vs Mutants by Tomek Dubikowski at JBCNConf 2016

  (https://www.youtube.com/watch?v=ptlSsXU0Jdg)

General terms

- “Quis custodiet ipsos custodes?” – Test the tests!
- “Mutant” – intentional change in the code
- SURVIVED mutant – a code change which was not detected by the tests
- KILLED mutant – if (at least) one test has failed after the code change
- Mutation Coverage = |mutants killed| / |all mutants|

Types of code mutations

- Statement deletion or insertion
- Replacement of conditional expressions with true or false
- Replacement of arithmetic operations
- Replacement of boundary conditions
- Replacement of variables
- Replacement of return values

Problems

- Number of possible mutations
- Combinations of mutations
- Automation of creating even some of the simple mutation types is complex
- TIME

PIT

- http://pitest.org - Mutation testing tool for Java
- First runs the (selected) tests without mutations (smoke test)
- Generates mutants by manipulating the byte code of (selected) compiled classes (via ASM library)
- Runs tests against a particular mutant based on test coverage
- Optional: use history or scm to speed up execution

PIT [cont]

- Simple setup

  It is enough to add this to the pom.xml of the project (gradle alternative is also simple)

  (...)

```
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>LATEST</version>
 </plugin>
```

  (...)

PIT [cont]

- Simple configuration

  Select which classes can be mutated, which tests can be run and exclude some classes and/or tests.

```
<configuration>
        <targetClasses>
             <param>com.ocado.featureA.*</param>
        </targetClasses>
        <targetTests>
             <param>com.ocado.featureA.*</param>
        </targetTests>
   <excludedClasses>
          <param>com.ocado.featureA.pojos.*</param>
             <param>com.ocado.featureA.FailingTest.java</param>
   </excludedClasses>
  </configuration>
```

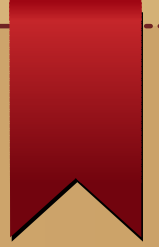
!NB: If some of the excluded tests are parts of a Test Suite, you must also exclude the Test Suite, otherwise they will be executed with it.

PIT [cont]

- Other configuration parameters:
  - `mutationTreshold` – fail the build if this level of mutation coverage is not reached
  - `lineTreshold` – similar to mutationTreshold
  - `withHistory` – keep (and use) history of previous runs in order to improve performance
  - `maxDependencyDistance` – a test that directly uses the mutated class has a distance of 1. A test that uses another class which itself uses the mutant as collaborator has a distance of 2. E.g, maxDependencyDistance=1 restricts possible candidates only to unit tests.

NB!: PIT fails to determine `maxDependencyDistance` if the reference is via an interface

PIT [cont]



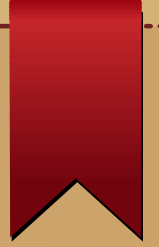
- And some more configuration options:
  - `threads` — increase to allow more simultaneous tests (default is 1)
  - `MaxMutationsPerClass` — set an upper limit (default is unlimited)
  - `jvmArgs` — pass Java parameters when executing tests

PIT [last]

- **Usage:**
  - mvn org.pitest:pitest-maven:mutationCoverage
  - mvn **-DwithHistory** org.pitest:pitest-maven:mutationCoverage
  - mvn org.pitest:pitest-maven:**scm**MutationCoverage
- **Results:**

  dist/pit/<timestamp>/

Summary



- PIT is very configurable
- Has some default optimizations, allows custom ones
- Also has pitfalls (e.g problems with interfaces)
- There are such tools for other languages:
  - https://stryker-mutator.github.io/ - for JavaScript
  - https://github.com/hcoles/sbt-pit - PIT for Scala (WiP)

Discussion

- Questions? (Sorry, I didn't bring pizza. Nor banitsa. Other questions?)

endif

```
assertTrue( msg.equals(“Thank you for your attention!”) );
```