Mutation testing

November 15th, 2023
Ana Paiva, José Campos

In this recitation class, we are going to explore 'Mutation Testing', *a white-box testing technique*, in the `jpacman` project, using the Pitest library.

Please make sure your machine is configured properly, i.e.:

- Java installed on your machine and available through the command line. Disclaimer: this tutorial has been validated under Java-11. It may or may not work on other versions of Java. Let us know whether it does not work under Java-X, where X is a version higher than 11.
- Apache Maven to be installed on your machine and available through the command line. In case Maven is not installed, please follow the following steps:
    - Download `apache-maven-3.9.4-bin.zip`
    - Extract `apache-maven-3.9.4-bin.zip`
    - On Windows, augment your environment variables with the full path to the `<extracted directory>/bin`. On Linux/MacOS, run `export PATH="<extracted directory>/bin:$PATH"`. (You might have to run the `export` everytime you restart the computer. For a more permanent solution, please consider adding that command to your bash profile.)

0. Setup

Before trying to perform 'Mutation Testing', let's first prepare our machine.

1. Get the `jpacman` project's source code (available here).
2. Collect all unit tests you developed in previous recitation classes for the `jpacman` project.
3. Double check whether the Pitest library responsible for mutation `jpacman` project's source code and running all test cases is properly configured in the project's `pom.xml` file. For instance, the Pitest library should be configured as

```
Unset
<build>
 <plugins>

  ...
```

```
  <plugin>
   <groupId>org.pitest</groupId>
   <artifactId>pitest-maven</artifactId>
   <version>1.9.11</version>
   <dependencies>
    <!-- The following dependency must be included to support JUnit 5
-->
    <dependency>
     <groupId>org.pitest</groupId>
     <artifactId>pitest-junit5-plugin</artifactId>
     <version>1.1.0</version>
    </dependency>
   </dependencies>
   <configuration>
    <excludedTestClasses>
     <!-- The following test classes must be excluded as they fail on
Pit -->
     <param>nl.tudelft.jpacman.LauncherSmokeTest</param>

<param>nl.tudelft.jpacman.e2e.framework.startup.StartupTest</para
m>

<param>nl.tudelft.jpacman.integration.StartupSystemTest</param>
     <param>nl.tudelft.jpacman.npc.ghost.NavigationTest</param>
     <param>nl.tudelft.jpacman.sprite.SpriteTest</param>
    </excludedTestClasses>
   </configuration>
  </plugin>

  ...

 </plugins>
</build>
```

and then run mutation analysis with the following command:

```
Unset
mvn test-compile org.pitest:pitest-maven:mutationCoverage
```

If successfully, Pitest will generate a report that you can find in `target/pit-reports/index.html`.

Note that since release 1.4.0 Pitest requires Java 8 or above, earlier releases require Java 5. Regarding JUnit, version 4.6 or above is supported (note JUnit 3 tests can be run using JUnit 4 so JUnit 3 tests are supported). JUnit 5 is not fully supported out of the box.

1. Assess current mutation score

Open the report generated by Pitest (`target/pit-reports/index.html`) and select three classes for which you have *some* mutation coverage (aka mutation score), i.e., classes for which you have a mutation coverage greater than 0% and less than 100%.

Few tips to read the generated report:

- `Line coverage` is the ratio of line statements covered by the tests to the all number of line statements potentially to be executed in your class.
- `Mutation coverage` is the ratio of the number of mutants killed by the tests to the all number of mutants (regardless of whether it was covered by a test or not).
- `Test strength` is the ratio of the number of mutants killed by the tests to the number of all mutants covered by the tests.

Explore the report and try to figure out why some of the mutants were not killed by your test suite.

2. Exercise: attempt to reach 100% mutation score

For the three classes you selected in (1), derive new tests that could kill more mutants. Re-run `mvn test-compile org.pitest:pitest-maven:mutationCoverage`, assess your mutation score, derive more test cases using the JUnit framework, …

Tip: To speed-up repeated analysis of the same codebase set the `withHistory` parameter to true as

```
Unset
mvn -DwithHistory test-compile
org.pitest:pitest-maven:mutationCoverage
```

3. Exercise: enable all mutation operators available

Configure the project's `pom.xml` file as suggested below to enable all mutation operators available.

```
Unset
<build>
 <plugins>

  ...

  <plugin>
   <groupId>org.pitest</groupId>
   <artifactId>pitest-maven</artifactId>
   <version>1.9.11</version>
   <dependencies>
    <!-- The following dependency must be included to support JUnit 5
-->
    <dependency>
     <groupId>org.pitest</groupId>
     <artifactId>pitest-junit5-plugin</artifactId>
     <version>1.1.0</version>
    </dependency>
   </dependencies>
   <configuration>
    <!-- The following test classes must be excluded as they fail on Pit
-->
    <excludedTestClasses>
     <param>nl.tudelft.jpacman.LauncherSmokeTest</param>

<param>nl.tudelft.jpacman.e2e.framework.startup.StartupTest</para
m>
```

```
<param>nl.tudelft.jpacman.integration.StartupSystemTest</param>
     <param>nl.tudelft.jpacman.npc.ghost.NavigationTest</param>
     <param>nl.tudelft.jpacman.sprite.SpriteTest</param>
    </excludedTestClasses>

    <!-- All mutation operators -->
    <mutators>
     <mutator>CONDITIONALS_BOUNDARY</mutator>
     <mutator>INCREMENTS</mutator>
     <mutator>INVERT_NEGS</mutator>
     <mutator>MATH</mutator>
     <mutator>NEGATE_CONDITIONALS</mutator>
     <mutator>VOID_METHOD_CALLS</mutator>
     <mutator>EMPTY_RETURNS</mutator>
     <mutator>FALSE_RETURNS</mutator>
     <mutator>TRUE_RETURNS</mutator>
     <mutator>NULL_RETURNS</mutator>
     <mutator>PRIMITIVE_RETURNS</mutator>
     <mutator>CONSTRUCTOR_CALLS</mutator>
     <mutator>INLINE_CONSTS</mutator>
     <mutator>NON_VOID_METHOD_CALLS</mutator>
     <mutator>REMOVE_CONDITIONALS</mutator>
     <mutator>REMOVE_INCREMENTS</mutator>
     <mutator>EXPERIMENTAL_ARGUMENT_PROPAGATION</mutator>
     <mutator>EXPERIMENTAL_BIG_INTEGER</mutator>
     <mutator>EXPERIMENTAL_NAKED_RECEIVER</mutator>
     <mutator>EXPERIMENTAL_MEMBER_VARIABLE</mutator>
     <mutator>EXPERIMENTAL_SWITCH</mutator>
    </mutators>
   </configuration>
  </plugin>

  ...

 </plugins>
```

```
</build>
```

then run mutation analysis with the following command:

```
Unset
mvn test-compile org.pitest:pitest-maven:mutationCoverage
```

and finally assess your mutation coverage and test strength. Has your mutation coverage decreased with more mutation operators? Can you derive more test cases to kill even more mutants generated by non-default mutation operators?

Miscellaneous

- Guide to Configuring Maven Plug-ins
- JUnit framework
- Learn how to write unit tests
- JUnit 5 User Guide
- Parameterized Tests and JUnit 5 Tutorial: Writing Parameterized Tests
- Pitest
- Pitest – maven
- Pitest – mutation operators