Structural testing

October 25th, 2023
Ana Paiva, José Campos

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

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 'Structural Testing', let's first prepare our machine.

1. Get the `jpacman` project's source code (available in here).
2. Collect all unit tests you developed in 'Category Partition' and 'Boundary Value Analysis' tutorial for the `jpacman` project.
3. Double check whether the JaCoCo library responsible for collecting code coverage is properly configured in the project's `pom.xml` file. For instance, the JaCoCo library should be configured as

```
Unset
<build>
 <plugins>

  ...
```

```
  <plugin>
   <groupId>org.jacoco</groupId>
   <artifactId>jacoco-maven-plugin</artifactId>
   <version>0.8.8</version>
   <executions>
    <execution>
     <id>default-prepare-agent</id>
     <goals>
      <goal>prepare-agent</goal>
     </goals>
    </execution>
    <execution>
     <id>default-report</id>
     <phase>test</phase>
     <goals>
      <goal>report</goal>
     </goals>
    </execution>
   </executions>
  </plugin>

  ...

 </plugins>
</build>
```

So that the JaCoCo `report` goal is executed during the Maven's test phase, i.e., when you run `mvn test`. The JaCoCo code coverage report will be generated to `target/site/jacoco/index.html`.

Now, run `mvn test` and assess which parts of the source of the code are not exercised by any test case, which parts of the source of the code are exercised by at least one test but not fully exercised, and which parts of the source of the code are fully exercised by all tests.

1. Line coverage exercise

1. Select a class with less than 100% line coverage.
2. Develop new test cases, for that class, to exercise all lines of code that have not been exercised.

2. Branch coverage exercise

1. Select a class with less than 100% branch coverage.
2. Develop new test cases, for that class, to exercise all branches of code that have not been *fully* exercised.

3. Cyclomatic Complexity exercise

Draw the control-flow graph of the method `getInheritance` in the `nl.tudelft.jpacman.level.CollisionInteractionMap` class. (Tip: you may use draw.io for example.)

```java
/**
 * A map of possible collisions and their handlers.
 *
 * @author Michael de Jong
 * @author Jeroen Roosen
 */
public class CollisionInteractionMap implements CollisionMap {

 // ...

 /**
   * Returns a list of all classes and interfaces the class
inherits.
   *
   * @param clazz
   *            The class to create a list of super classes and
interfaces
   *            for.
   * @return A list of all classes and interfaces the class
inherits.
   */
```

```
 @SuppressWarnings("unchecked")
 private List<Class<? extends Unit>> getInheritance(
   Class<? extends Unit> clazz) {
01   List<Class<? extends Unit>> found = new ArrayList<>();
02   found.add(clazz);
03
04   int index = 0;
05   while (found.size() > index) {
06     Class<?> current = found.get(index);
07     Class<?> superClass = current.getSuperclass();
08     if (superClass != null &&
Unit.class.isAssignableFrom(superClass)) {
09       found.add((Class<? extends Unit>) superClass);
10     }
11     for (Class<?> classInterface : current.getInterfaces()) {
12       if (Unit.class.isAssignableFrom(classInterface)) {
13         found.add((Class<? extends Unit>) classInterface);
14       }
15     }
16     index++;
17   }
18
19   return found;
 }

 // ...

}
```

1. Compute its cyclomatic complexity and write it in a txt file.
2. Identify its independent paths and write them in a txt file.
3. Write unit test cases using the JUnit framework to every single independent path identified in (2).

4. What should you submit/deliver?

Zip the
- Project's directory.

- Control-flow graph from Section 3 and the txt file.
and submit it here (M.EIC's moodle) or here (MESW's moodle).

Deadline: ~~End of the recitation class.~~ October 25, 2023, 11:59:00 pm.
Grades: available on November 1, 2023.

Miscellaneous

- JaCoCo library
- JaCoCo maven plugin doc
- JaCoCo maven usage example