SAARLAND UNIVERSITY
COMPUTER SCIENCE

Universität des Saarlandes
Software Engineering Chair
Prof. Andreas Zeller
Gordon Fraser
David Schuler

1. Project: Code Coverage

The task of this project is to (a) write a tool that measures statement coverage, (b) create tests that satisfy a coverage goal, and (c) use coverage information for fault localization.

1 Requirements

You need the following tools for the project:

- Java SE Development Kit (JDK), version $\geq$ 1.5.
- Ant `http://ant.apache.org/`
- Maven `http://maven.apache.org/`
- Subversion `http://subversion.apache.org/`

2 Set up the project

You can check out the project from our SVN repository. Your user name is your matriculation number, and the password was sent to you by e-mail. The location of your repository for the first project is:
`https://prog2.cs.uni-saarland.de/debugging/students/project1/<MATR>`
where `<MATR>` has to be replaced with your matriculation number. The command for checking out the initial working copy is :

```
svn co --username <MATR>
https://prog2.cs.uni-saarland.de/debugging/students/project1/<MATR>
```

At the end of the project (13th May, at 23:59) the version in the repository will be graded.

The project can be built using maven, with the command `mvn compile`. Maven can also create a project suitable for importing into eclipse with the command: `mvn eclipse:eclipse`. For more details see the documentation on maven.

3 Code Coverage Tool

Your code coverage tool must work in 3 phases:

1. First, the source code is instrumented such that the execution of statements is logged.
2. Then, the instrumented code is compiled and the tests are run. At the end of the run the coverage data is written out.

3. Finally, the data from the previous step is read and analyzed.

The following sections give an overview over the project, for more details see also the Javadoc comments, and the lecture on *Foundations of Testing 2*.

3.1 Instrumenting files

The class *CoverageTransformer* receives a list of Java source files that should be instrumented. For these files statements have to be added to the source code that log each source statement that is executed. Then, the transformed files should be written to a directory that is specified via the system property *target.dir*.

In this step, it should also be determined which of the statements can be covered. Therefore, the system property *coverage.dir* gets passed, which gives a location where the data should be written.

For parsing and instrumenting the source code we recommend to use the Eclipse AST (Abstract Syntax Tree) parser. There are several tutorials that describe how to work with the AST, for example:

`http://tinyurl.com/y7zjoj7`

3.2 Running the instrumented files

When the instrumented classes are run, the executed statements are logged, and at the end of the run the collected data is written to disk. Therefore, the system property *coverage.dir* gets passed to the program. This property gives a location (directory) where the coverage information should be written.

3.3 Tracing test names

For JUnit 4 Tests, it should also be traced which test executes which statements. Therefore, the tests have to be run with a customized test runner. This test runner is *CoverageTestRunner*. It should run the tests, trace the tests that are run, and log whether a test passed or failed. This functionality is also tested in the integration test of the fault localizer.

3.4 Reading the coverage data

The class *CoverageDataReader* reads the data that was written in the previous step and returns a *CoverageData* object. The location of the coverage file is also determined via the system property *coverage.dir*.

3.5 What needs to be logged?

In general every statement in a method or constructor should be logged. However, there are some exceptions for statements that should not be logged and are not expected to be coverable, e.g. else, case, catch, finally, super constructor call, and this constructor call.

In the package *testing.debugging.project1.coverage.integration* there are several tests that describe how different statements are expected to be covered. Each class has some code that is

exercised by the main method, and two methods that return the data that is expected from the coverage tool. The method *getExpectedCoverage()* returns an array that describes which lines are expected to be covered, and the method *getCoverableLines()* returns an array with all lines that are expected to be coverable.

The tests can be run via ant with the command: *ant check-coverage*. Note: Before the first invocation *mvn assembly:assembly* has to be run once.

4 Reaching a coverage goal

Using the coverage tool, the percentage of covered statements for a class can be determined. To this end, the tests in the package *testing.debugging.project1.coveragegoal* should be enhanced such that 100 % code coverage is reached for the classes:
org.apache.commons.math.stat.descriptive.moment.Variance and
org.apache.commons.math.fraction.ProperFractionFormat from the COMMONS-MATH project as they appear in revision 801855. The svn command to check out this version is:
`svn checkout -r 801855`
`http://svn.apache.org/repos/asf/commons/proper/math/trunk commons-math`
It is allowed to use existing tests from COMMONS-MATH to reach the coverage goal.

5 Fault Localization

Using the coverage data of passing and failing tests, allows to apply fault localization techniques as described in the lecture on *Comparing Coverage*. Your task is to implement the Tarantula fault localization technique as defined in the paper *Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique* [1] , which uses the following formula to calculate the suspiciousness for a line:

$$suspiciousness(e) = \frac{\frac{failed(e)}{totalfailed}}{\frac{passed(e)}{totalpassed} + \frac{failed(e)}{totalfailed}}$$

The *FaultLocalizerFactory* should return a *FaultLocalizer* that implements the Tarantula technique. An integration test for the fault localizer can be run with the command *ant check-fault-localization*.

6 Hints

- In order to run a process on exit of the virtual machine, you can use the method *addShutdownHook()* in class *java.lang.Runtime*
- You are allowed to use several third party libraries, e.g. commons-lang, commons-io, google-collections. For details see the `pom.xml`. If you want to use a library not listed there, write us a mail and explain why you want to use this library.

[1] The paper can be obtained from our web site. The user name and password is *tad10*

- There is a bug in the *rewrite()* method of class *org.eclipse.jdt.core.dom.CompilationUnit* that either results in a *java.lang.StringIndexOutOfBoundsException*, or in wrong output for the test case *NoBracesElseIfCoverageTest*. A workaround is to use the *toString* method of *CompilationUnit* instead.

Enjoy!