LINGI2251
Software Quality Assurance
Assignment 1
Spring 2019

Charles Pecheur

March 5, 2019

Due Mon Mar 25

The object of this assignment is to produce and apply tests to a Java program, using different tools. The program to be tested, and the tools to be used, are available online and are referenced from the website of the course.

This assignment can be performed in **groups of two students**. Individual submissions are also accepted. Interaction between students and groups is allowed but plagiarism will not be tolerated.

Assignment results will be returned as an archive file (`.zip` or `.gz`) whose base name is the last names of the students (e.g. `Pecheur_Martin.zip`, no accents please). The archive will at least contain a **report** covering all tasks below. If more files are provided, their use will be explained at the beginning of the report. Make sure to **include proper identification** (course, year, assignment number, student names) at the beginning of all documents.

Submit your deliverables in the *Assignments* section of the Moodle website, before the deadline stated above. Late submissions will not be accepted.

Please post any question regarding this assignment on the forum on the course website. For personal issues you may contact *Charles Pecheur* or *Igor Zavalyshyn*.

The subject: Jodd Template Parser

The software to be tested is part of *Jodd*, a collection of tools, frameworks and utilities for Java programs (`http://jodd.org/`). Specifically,

you will work on the *String Template Parser* utility, provided by the class `jodd.util.StringTemplateParser`. The main method is `parse()`. The class is documented at `http://jodd.org/doc/stringtemplateparser.html`. Note that `parse()` calls other methods from other classes within Jodd.

Tools

The following software tools will be used in this assignment.

Eclipse — You will have to work within the Eclipse IDE[1], as several tools that you will be using come as Eclispe plug-ins.

JUnit — Your will develop tests using the JUnit framework[2]. JUnit is readily supported and available in Eclipse.

PICT — You will use PICT to produce functional test specifications[3]. PICT produces pairwise test cases given a combinatorial specification of test categories and classes. PICT is a command-line tool.

Note that PICT only produces a list of combinations of classes, not the actual Java test cases. It would be possible to read and feed PICT's output into JUnit tests, but that capability is not provided here.

EclEmma — You will use EclEmma to measure structural coverage of our tests[4]. EclEmma is an Eclipse plug-in, available on the Eclipse marketplace. EclEmma uses JaCoCo to perform structural coverage analysis. JaCoCo performs analysis on-the-fly, on the Java bytecode, as an agent observing the Java VM.

Pitclipse — You will use PitClipse to perform mutation testing[5]. Pitclipse is an eclipse plug-in available on the Eclipse marketplace. Pitclipse is a wrapper over PITest, which produces and tests the mutants.

[1] `https://www.eclipse.org/`
[2] `http://junit.org/junit4/`
[3] `https://github.com/microsoft/pict`
[4] `http://www.jacoco.org/index.html`
[5] `https://github.com/philglover/pitclipse`

Tasks

Have the tools downloaded, compiled and/or installed in Eclipse as needed.

Download the sources of Jodd (from the Jodd website or from Moodle) and unpack the archive[6].

Functional testing

Based on the specification, identify the **parameters** and **environment elements** of `StringTemplateParser`. Define **categories** (characteristics) and **choices** (classes of values) over those parameters. The categories may relate to a single function parameter, several parameters, or part of a parameter (the input string); discuss the need for each. Specify **constraints** between the categories and choices; in particular, identify value choices corresponding to error or singular situations, for which a single test case need be generated.

Specify your values and constraints into a *PICT model* file[7]. Execute PICT (with default pairwise generation) to obtain a table of test case specifications.

Report: the list of categories and choices (or the PICT model file) with explanations; statistics on the generated test cases.

Develop **JUnit tests** corresponding to the test cases that you have obtained. One test run may cover several test cases in sequence but keep test cases multiple and short, so that failed test cases remain informative. Run your tests and report results, in particular if a failed test reveals a fault in the program.

Report: general description of tests; full source code in appendix; test results.

Structural Testing

Perform structural coverage analysis on your tests with EclEmma. Consider coverage within the `StringTemplateParser` class only. Report the different coverage statistics (instruction, branch, complexity, method)[8]. Analyse missing coverage cases; justify if coverage is unachievable, and add test cases to achieve coverage otherwise. Report any new fault uncovered through additional test cases.

[6] `jar xvf <file>.jar`

[7] `https://github.com/Microsoft/pict/blob/master/doc/pict.md`

[8] selectable from pull-down menu in coverage window

Report: coverage statistics with initial and with completed test suite; discussion of unachievable coverage cases; added test cases.

Fault-Based Testing

Perform mutation analysis on `StringTemplateParser` using Pitclipse, with the default set of mutators[9]. Report statistics of mutants killed. Analyse unkilled mutants; justify if killing is unachievable, and add test cases that kill those mutants otherwise. Report any new fault uncovered through additional test cases.

Report: coverage statistics with initial and with completed test suite; discussion of unkillable mutants; added test cases.

[9] Configurable in the preferences panel.