CS 520
In-class exercise 2
Software testing

Due: **Wednesday, March 17, 2021, 9:00 AM** via Moodle. This in-class exercise is a group submission. This means that **each group only needs to submit their solution once** and also that every student in a group will get the same grade. You will work with students within your group, but not with students from other groups. Multiple groups' submissions may not be created jointly. Late assignments will be accepted for extenuating circumstances.

Overview and goal

The high-level goal of this exercise is to learn how to systematically unit-test a program and how to assess test quality, using code coverage and mutation analysis.

What to do?

Forming groups

1. Team up in groups of size 3, 4, or 5. (If you cannot find enough members, raise your hand and ask the instructor.)
2. Create a new group on Moodle (see "In-class exercise 2: Group selection"), and add all group members.

Set up

1. Make sure that you have Git (v2.7.4 or later) and Java (v8 or later) installed.

   Git: `https://git-scm.com/`

   Java: `https://www.oracle.com/technetwork/java/javase/downloads/jdk13-downloads-5672538.html`

   USEFUL TIP: Make sure that the JDK comes early in the `PATH` environment variable. This can fix a lot of issues with modern and multiple JDK installations.
   For more information, see `https://www.java.com/en/download/help/path.xml`
2. Clone the cs520 git repository:

   ```
   git clone https://github.com/LASER-UMASS/cs520 inclass2
   ```
3. In a terminal, change into the triangle directory (`inclass2/triangle`).
4. Read the provided `README.md` file in that directory.
5. Test your set up by running `./test.sh`
6. Familiarize yourself with the triangle program (`src/triangle/Triangle.java`).
7. Familiarize yourself with the example test suite (`test/triangle/TriangleTest.java`).

8. We will be using the Cobertura (http://cobertura.github.io/cobertura/) tool to analyze coverage. Run the Cobertura code coverage analysis (`./coverage.sh`) and inspect the report it produces (`coverage_results/index.html`).

   USEFUL RESOURSES: How to read a Cobertura coverage report:

   https://github.com/cobertura/cobertura/wiki/Line-Coverage-Explained

   https://technology.amis.nl/2006/04/07/getting-started-with-cobertura/

9. Run the mutation analysis (`./mutation.sh`) and inspect the set of killed mutants it reports (`mutation_results/killed.csv`).

   USEFUL RESOURSE: The `README.md` file inside the `triangle` directory provides information on the mutation analysis.

   USEFUL TIP: If a mutant is labeled **LIVE** in the `mutation_results/killed.csv` file, your test suite did not catch (a.k.a. did not kill) that mutant. If a mutant is labeled **FAIL**, your test suite did catch (kill) that mutant.

Writing unit tests

Goal: Develop three (3) test suites that satisfy different (coverage) criteria.

1. Develop a test suite that satisfies the statement coverage criterion. (The coverage report calls this *Line Coverage*.)

2. Develop a test suite that satisfies the decision and condition coverage criterion. (The coverage report calls this *Branch Coverage*.)

3. Develop a test suite that is mutation adequate (i.e., it detect all detectable mutants).

Take notes about how you approach each step; in steps 2 and 3, you may extend the test suite developed in the previous step(s). Make sure to commit each test suite to your cloned repository with a proper commit message.

NOTE: You are required to submit three test suites, even if some of them are identical or one subsumes another.

Analyses

Goal: Interpret code coverage and mutation analysis results.

1. Determine the code coverage ratios and mutant detection rates for each of your three developed test suites.

2. Delete all assertions from your test suites and repeat step 1.

Questions

Using your notes and results, answer the following questions:

1. Did your approach to writing unit tests differ between developing a coverage-adequate test suite and developing a mutation-adequate test suite? Briefly explain why or why not.

2. Do your coverage-adequate test suites detect (i.e., kill) all detectable mutants? Do they cover all mutants (i.e., cover the mutated code)? Briefly explain why or why not?

3. For any given program, why are some mutants not detectable?

4. Consider your mutation-adequate test suite and the triangle program. For each undetected mutant, briefly explain why it is not detectable.

5. What changes in the code coverage ratios and mutant detection rates did you observe when deleting all assertions?

6. Create a definition of "test case redundancy" based on code coverage or mutation analysis. Given your definition of test case redundancy, are some of the test cases in your test suites redundant? Given your definition of test case redundancy, would you remove redundant test cases? Briefly explain why or why not.

Deliverables

Your submission, via Moodle, must be a single (one per group) archive (`.zip` or `.tar.gz`) file with name `<group name>-inclass2.<zip/tar.gz>`, containing:

1. `answers.txt`: A plain-text file with your answers to the above 6 questions. *List all group members at the top of this file*.

2. `inclass2`: Your copy of the `inclass2` repository, with your test suites committed. Note that this should **not** be the files in the `inclass2` working copy, but instead the **repository** (which is the `.git` directory in `inclass2`.) For example, on a Linux-based machine (e.g., MacOS), you can use the terminal from the `inclass2` directory and run the command
   `tar -vczf inclass2.repo.tar.gz .git`