UNIVERSITY OF ANTWERP
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Software Testing Lab

Assignment 5- Mutation Testing

Submission Deadline: **31 March 2025, 20:00**

Refer to "Assignments General" document for introductory information.

MUTATION TESTING

Mutation testing is a technique to measure the quality of a test suite by assessing its fault detection capabilities. Mutation testing starts with a *green* test suite, i.e. a test suite in which all the tests pass. First, a faulty version of the software is created by introducing faults into the system *(Mutation)*. This is done by applying a known transformation *(Mutation Operator)* on a certain part of the code. After generating the faulty version of the software *(Mutant)*, it is passed onto the test suite. If there is an error or failure during the execution of the test suite, the mutant is marked as killed *(Killed Mutant)*. If all tests pass, it means that the test suite could not catch the fault, and the mutant has survived *(Survived Mutant)*.

Mutation testing allows software engineers to monitor the fault detection capability of a test suite by means of mutation coverage (see Equation 1). If the output of a mutant for all possible inputs is the same as the original program, it is called an *equivalent mutant*. It is not possible to create a test case that passes for the original program and fails for an equivalent mutant, because the equivalent mutant has the same semantics as the original program. A test suite is said to achieve *full mutation test adequacy* whenever it can kill all the non-equivalent mutants, thus reaching a mutation coverage of 100%. Such test suite is called a *mutation-adequate test suite*.

$$Mutation\ Coverage = \frac{Number\ of\ killed\ mutants}{Number\ of\ all\ non\text{-}equivalent\ mutants} \quad (1)$$

In this assignment we will use LittleDarwin. LittleDarwin is a mutation testing tool to provide mutation testing within a continuous integration environment. It is designed to have a loose coupling with the test infrastructure, instead relying on the build system to run the test suite.

- **Exercise 1.** Use LittleDarwin to analyse the original version of JPacMan. LittleDarwin and its manual can be seen and downloaded from `https://github.com/aliparsai/LittleDarwin`. Explain the results. Now, use LittleDarwin to analyse *your* version of JPacMan. What differences can you see? **(Required, 10 points)**

- **Exercise 2.** Repeat the previous exercise, but this time use branch coverage (with JaCoCo) instead of mutation coverage. Look for a class with 100% branch coverage and less than 100% mutation coverage. Why did this happen? Look for similar examples (interesting differences between the coverages of the two techniques) and explain the difference between two results. **(Required, 10 points)**

- **Exercise 3.** Repeat the first exercises using PITest **or** Javalanche. Explain the difference in the results. **(Only when late, -10 to 0 points)**

- **Exercise 4.** Find a killed mutant. Explain why, where, and how it was killed? **(Required, 5 points)**

- **Exercise 5.** Take a survived mutant for class `Engine`, and write a test that kills it. Repeat the process until all survived mutants from the `Engine` class are covered. Rerun LittleDarwin to confirm. Note that you can run LittleDarwin for specific classes/files. This drastically cuts down on run time. **(Required, 15 points)**

- **Exercise 6.** Can you find an example of an equivalent mutant? How do you know (proof) it is equivalent? **(Required, 5 points)**

- **Exercise 7.** Make a mutation-adequate test suite for the model package of JPacMan. How many tests did you have to write? (Count them!) How many equivalent mutants did you find? Explain why each of them is equivalent. Note that you can run LittleDarwin for specific classes/files. This drastically cuts down on run time. **(Required, 50 points)**

- **Exercise 8.** Make a mutation-adequate test suite for the controller package of JPacMan. How many tests did you have to write? (Count them!) How many equivalent mutants did you find? Explain why each of them is equivalent. **(Only when late, -50 to 0 points)**

- **Exercise 9.** What are the upsides and downsides of mutation testing? Explain your argument. **(Required, 5 points)**