

Lab 5

Part 1: Manual mutation testing

Selected values:

Input Values	Actual output	<code>i % 3 != 0 && i % 5 != 0</code>	<code>i % 3 == 0 i % 5 == 0</code>	<code>i % 3 != 0</code>	<code>i * 3 == 0</code>	<code>i - 5 == 0</code>
2	2	Fizz!Buzz!	2	Fizz	2	2
15	Fizz!Buzz!	Fizz	Fizz!Buzz!	Fizz!Buzz!	Fizz!Buzz!	Fizz!Buzz!
25	Buzz	Buzz	Fizz!Buzz!	Fizz	Buzz	25
27	Fizz	Fizz	Fizz!Buzz!	27	27	Fizz
28	28	Fizz!Buzz!	28	Fizz	28	28

What is the practical use case of introducing mutants? Say that the five pairs of input and output printed to the terminal was our tests?

Mutants use to test the quality of test cases that already written and if there is anything missing in tests then mutation is the way to find it out. Yes the above changes in the code like change `&&` with `||` is mutation testing.

Why would we in an industrial context want to automate the process of introducing mutants?

Because it's very hard to test all mutations manually or by random guessing. Also some deeper parts remains untested with this. So automation testing change values accordingly with every expected pair and validate current tests and generate new.

Part 2: Automation mutation testing

Below images showing the code coverage and mutant score generated by the test suit.

Code coverage

LCOV - code coverage report

Current view: top level		Hit		Total	Coverage
Test: coverage.info		Lines:	1429	2126	67.2 %
Date: 2020-10-14 16:43:39		Functions:	1166	1530	76.2 %
Directory	Line Coverage	Functions			
/home/muhis51/Documents/TDD04/Lab5/dextool/vendor/fused_gmock/gtest	41.0 % 25 / 61	47.1 % 24 / 51			
/usr/include/c++/7	97.0 % 97 / 100	90.8 % 99 / 109			
/usr/include/c++/7/bits	74.9 % 829 / 1107	77.7 % 795 / 1023			
/usr/include/c++/7/ext	77.4 % 24 / 31	86.3 % 107 / 124			
deps	0.0 % 0 / 35	0.0 % 0 / 34			
src	56.4 % 438 / 776	73.1 % 128 / 175			
test	100.0 % 16 / 16	92.9 % 13 / 14			

Generated by: LCOV version 1.13

Mutation Score

[Statistics](#)

[Long Term View](#)

[NoMut Details](#)

[Minimal Test Set](#)

[Test Case Similarity](#)

Path >	Score >	Alive >	NoMut >	Total >
src/game.cpp	0.4	12	0	20
src/mobsystem.cpp	0	10	0	10
src/renderssystem.cpp	0.6	8	0	20
src/util.h	0.267	44	0	60

NoMut is the number of alive mutants in the file that are ignored. This increases the score.

What do the results of running dextool tell us about test quality? Relate the findings from doing this experiment to the first lab. Would we make similar observations for tests for highly coupled code as in the Colony example?

Results showing us about mutation score, which files got mutate values and how much it changes. If we change or mutate it and test get pass that means mutant is alive and if it get fails its called mutant killed. If alive mutants are high that means it is not a good quality tests.

We can test colony example with this method of mutation and observe the behavior same like the above example. But high coupling can cause a problem in mutation because modules are too much depending on each other's.

Part 3: Extend the test suit

```

TEST_F(Basic, StepRight) {
    // act
    window->inject(WindowEvent::ArrowRight);
    game->update();
    game->render();
    window->render();
    EXPECT_EQ(window->layoutEvents.size(), 35163);
}

TEST_F(Basic, ArrowDown) {
    // act
    window->inject(WindowEvent::ArrowDown);
    game->update();
    game->render();
    window->render();
    EXPECT_EQ(window->layoutEvents.size(), 35163);
}

TEST_F(Basic, ArrowUp) {
    window->inject(WindowEvent::ArrowUp);
    game->update();
    game->render();
    window->render();
    EXPECT_EQ(window->layoutEvents.size(), 35159);
}

```

After generating above new test cases mutation score increased.

File-> src/util.h

Old score-> 263

New score-> 283

[Statistics](#)

[Long Term View](#)

[NoMut Details](#)

[Minimal Test Set](#)








[Test Case Similarity](#)

Path >	Score >	Alive >	NoMut >	Total >
src/game.cpp	0.4	12	0	20
src/mobsystem.cpp	0	10	0	10
src/renderssystem.cpp	0.6	8	0	20
src/util.h	0.283	43	0	60

NoMut is the number of alive mutants in the file that are ignored. This increases the score.

LCOV - code coverage report

Current view: top level		Hit	Total	Coverage
Test: coverage.info	Lines:	1492	2153	69.3 %
Date: 2020-10-15 20:26:03	Functions:	1227	1565	78.4 %

Directory	Line Coverage ↕	Functions ↕
/home/muhis511/Documents/TDD004/Lab5/dextool/vendor/fused_gmock/gtest	 39.1 % 25 / 64	52.1 % 38 / 73
/usr/include/c++/7	 97.0 % 97 / 100	90.8 % 99 / 109
/usr/include/c++/7/bits	 74.9 % 829 / 1107	79.7 % 815 / 1023
/usr/include/c++/7/ext	 77.4 % 24 / 31	89.5 % 111 / 124
deps	 71.4 % 26 / 35	28.6 % 10 / 35
src	 58.2 % 452 / 776	73.7 % 129 / 175
test	 100.0 % 40 / 40	96.2 % 25 / 26

Generated by: LCOV version 1.13

Part 4: Equivalent mutants

Find at least one equivalent mutant in the fizz_buzz example. You might have already found one!

`i % 3` and `i %-3` are generating same results and are equivalent mutants.

If we consider mutation score when doing automated mutation testing. Why is the existence of equivalent mutants a practical problem?

It generate same results and when we generate equivalent mutant is test nothing expect doing an extra test.