Test Coverage and Mutation testing

Bingquan Wang

1 Terminology

In this paper we use the following symbol:

- T - master test suite
- t - sub test suite
- P - program under test

1.1 Standard terminologies

Formal definition for Mutation Score:

$$Mutation\ Score = \frac{\#killedMutants(T,P)}{\#non\text{-}equivalentMutants(P)}$$

Equivalent mutants by formal definition are mutants can never be killed. But in practice, it is a undecidable problem. So equivalent mutants and non-equivalent mutants are often estimated by, assumption that the test suite is in good quality:

$$equivalentMutants = \#totalMutants - \#killedMutants$$

$$non\text{-}equivalentMutants = \#killedMutants$$

And Mutation Score is estimated using the following equation.

$$Mutation\ Score = \frac{\#killedMutants(T,P)}{\#totalMutants(P)}$$

1.2 Terminologies used in the paper

In Holmes's paper in addition to Mutation Score, they also introduced effectiveness. However, there is no clear mathematics definition given. To understand the effectiveness better, we conclude following definition from the original paper.

In the paper, the defination for raw effectiveness is "raw kill score is the number of mutants a test suite detected divided by the total number of non-equivalent mutants that were generated for the subject program under test." (section 3.6, paragraph 1, sentence 2).

"Number of mutants a test suite detected" we use $\#killedMutants(t,P)$ to represent. And "total number of non-equivalent mutants that were generated" is represented by $\#killedMutants(T,P)$ (assuming the standard estimation for non-equivalent mutants, as stated in section 1.1).

$$rawEffectiveness = \frac{\#killedMutants(t,P)}{\#killedMutants(T,P)}$$

Holmes paper also give a definition of normalised effectiveness: "The normalized effectiveness measurement is the number of mutants a test suite detected divided by the number of non-equivalent mutants it covers"(section 3.6, paragraph 1, sentence 3).

Here, the number of non-equivalent mutants it covers. A mutant is covered means it is executed. (section 3.6, paragraph 1, sentence 4).

$$normalisedEffectiveness = \frac{\#killedMutants(t,P)}{\#non\text{-}euquivalentExecutedMutants(t,P)}$$

PIT does not tell executed mutants directly but gives information about mutants not executed, so number of non-equivalent executed mutants can be calculated:

$$\#non\text{-}euquivalentExecutedMutants = \#totalMutants(P) - \#no\_coverageMutant(t,P) - \#equivalentMutants(t,P)$$

I think the above is the same as *killedMutants(t,P)*

I think normalised effectiveness they meant is: he normalized effectiveness measurement is the number of mutants a test suite detected divided by the number of mutants it covers. Which gives:

$$normalisedEffectiveness = \frac{\#killedMutants(t,P)}{\#executedMutants(t,P)}$$

2 Critics of research

- Estimation of non-equivalent mutant as killed mutants holds only true when the test suite is in good quality.
- When they randomly generate test suite, they tried to control the size of test suite. However, it appears that they assume each test case test for one method, which in fact is not true. Consider this situation: inside a method A, it calls method B, and inside method B it calls method C. Instead of testing one method A, it actually is testing A, B and C. And calling other methods is common in programming. For a given test case which only test A, the test coverage may not change but the mutation testing may be affected as method B or C can not pass mutation test.
- Another problem with calling other methods within a method is multiple counting of effectiveness. There are probably multiple test cases actually testing the same functionality. Taking the previous example, test case for A, B and C may finally used for testing A. So there is duplicate of effectiveness.
- This paper introduced its own concept of effectiveness called normalised effectiveness. For a test suite t and Program P:

  $$normalised\ Effectiveness = \frac{\#killedMutants(t,P)}{\#non\text{-}euquivalentExecutedMutants(tP)}$$

  But to get the executed mutants, the first assumption is the test suite needs to potentially detect the mutant; and the second assumption is test cases must be in the same quality throughout program, which can not be true. There might be cases where it detects mutants but the test suite is poorly constructed (because randomness) and can not kill mutant, which leads to low effectiveness. Or there are cases that small number of test cases can not execute a lot of mutants but hight in killing mutant result in a high effectiveness. I think this normalised effectiveness has too may variants, and needs to be improved.