Software Testing: Tutorial 5

Mutation Testing

Consider the following program:

```
public int Segment(int t[], int l, int u){

   // Assumes t is in ascending order, and l<u,
   // counts the length of the segment
   // of t with each element l<t[i]<u

   int k = 0;

   for(int i=0; i<t.length && t[i]<u; i++){

      if(t[i]>l){k++;}

   }
   return(k);
}
```

This is not a particularly good example of programming but it is useful for the purposes of this tutorial.

- **Prerequisites:** Review the lecture on Mutation Testing and the papers by Offutt et al included in the required readings.
- **Preparation:** Review the code above; please try to ensure you understand the method and the implementation. The program assumes the array t is in ascending order and given two integers l and u it finds the length k of the sequence $t[j]$, $t[j+1]$,... $t[j+k-1]$ with:

  $((j = 0 or t[j-1] <= l)$ and $l < t[j])$ and

  $((t[j+k-1] < u)$ and $(j + k = t.length$ or $u <= t[j+k])$

  so k is the length of the longest subsequence of t with all elements greater than l and smaller than u.

Activities

Depending on the size of the tutorial group split into two, three or four groups.

1. (10 Minutes) In your groups, construct a test suite for this method. Each test case in the suite needs to specify the size of t, the values of the elements of t, the values l and u, and the expected result of the test. Write your test suite down on a separate sheet of paper. Use a simple coverage criterion to judge the adequacy of you test suite. You might just want to use statement coverage. The choice is yours.

2. (10 Minutes) In your groups, construct three or four mutants of the above program. Each mutant should be derived from the original program using one application of one of the following rules (i.e. each mutant contains only one mutation):

   (a) **Changing constants:** a constant c in the program may be replaced by $c + 1$ or $c - 1$.

   (b) **One-off in relations:** $<$ may be replaced by $<=$ and vice versa; $>$ may be replaced by $>=$ and vice versa.

   (c) **Inverted relations:** $<$ or $<=$ may be replaced by $>$ or $>=$ and vice versa.

   (d) **Duplicate statements:** any statement s; can be replaced by s;s;

   (e) **Deleted statements:** you may delete any statement.

3. (15 Minutes) Pass your test suite to a neighbouring group. With the test suite you have just received: for each of your mutants work out whether or not it is killed by the test suite.

4. (10 Minutes) If there is at least one mutant which is not killed in at least one of the groups do the following:

   (a) Get the tutor to write the unkilled mutants on the board.

   (b) In your groups check if any of the mutants on the board is killed by the test set you used in part 3.

   (c) Remove any mutant that is killed by at least one test suite.

   (d) If the board is empty of mutants go to stage 5, otherwise go to stage 6.

5. (10 Minutes) If there are no mutants which are unkilled by all the test suites so the following:

   (a) Get the tutor to write the test suites on the board.

   (b) In your groups, attempt to create a mutant that is not killed by any of the test suites on the board.

   (c) If you can create such a mutant write it on the board. If you cannot create such a mutant try to invent other mutation rules that let you create such a mutant.

6. (10 Minutes) To get here you must have a mutant on the board that has evaded death by test suite. In groups, try to strengthen your test suite until it kills the mutant on the board. Write the strengthened test suites on the board.

7. If there is time discuss how effective you think Mutation testing is in strengthening test suites.