Department of Computing

Curtin University

Software Engineering Testing (SET)

Week 9 Laboratory/Tutorial

The following exercises are intended to be done in a laboratory/tutorial session with a teaching assistant or instructor present. The exercises have been designed to reinforce concepts taught in SET.

1. Answer the following questions:

   (a) Just as with graphs, it is possible to generate an infinite number of tests from a grammar. How and what makes this possible?

   (b) Why does it make sense to remove ineffective test cases?

   (c) For which purposes is mutation testing used?

   (d) To check the quality of your test suite, you insert 2000 mutations into your code (one after the other). 1000 of these single mutations will be found by your test suite. What do you conclude?

   (e) Due to the results of (d), you revise your test suite. Now, your test suite detects 1950 of the 2000 mutations. Additionally you find 8 new defects in your program. What does this tell you about the number of remaining defects in your program?

2. Consider the stream BNF in Figure below and the ground string “B 10 06.27.94."
Give three valid and three invalid mutants of the string.

stream ::= action*
action ::= actG | actB
actG ::= "G" s n
actB ::= "B" t n
s ::= digit$^{1-3}$
t ::= digit$^{1-3}$
n ::= digit^{2} "." digit^{2} "." digit^{2}
digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

3. Answer questions (a) through (d) for the mutant in the method sum ( ) [in line 6’].

```
//Effects: If x null throw NullPointerException
//   else return the sum of the values in x

1. public static int sum(int[] x)
2. {
3.     int s = 0;
4.     for (int i=0; i < x.length; i++) }
5.     {
6.        s = s + x[i];
6'.    // s = s - x[i]; //AOR
7.     }
8.     return s;
9.  }
```

(a) If possible, find a test input that does **not** reach the mutant.
(b) If possible, find a test input that satisfies reachability but **not infection** for the mutant.
(c) If possible, find a test input that satisfies infection, but **not propagation** for the mutant.
(d) If possible, find a test input that kills mutant m.

4. Provide reachability conditions, infection conditions, propagation conditions, and test case values to kill mutants 1, 2, 5, and 6 in Figure below.

Original Method	With Embedded Mutants
int Min (int A, int B)	int Min (int A, int B)
{	{
int minVal;	int minVal;
minVal = A;	minVal = A;
if (B < A)	Δ1 minVal = **B**;
{	if (B < A)
minVal = B;	Δ2 if (B > A)
}	Δ3 if (B < **minVal**)
return (minVal);	{
} // end Min	minVal = B;
	Δ4 **Bomb()**;
	Δ5 minVal = **A**;
	Δ6 minVal = **failOnZero (B)**;
	}
	return (minVal);
	} // end Min