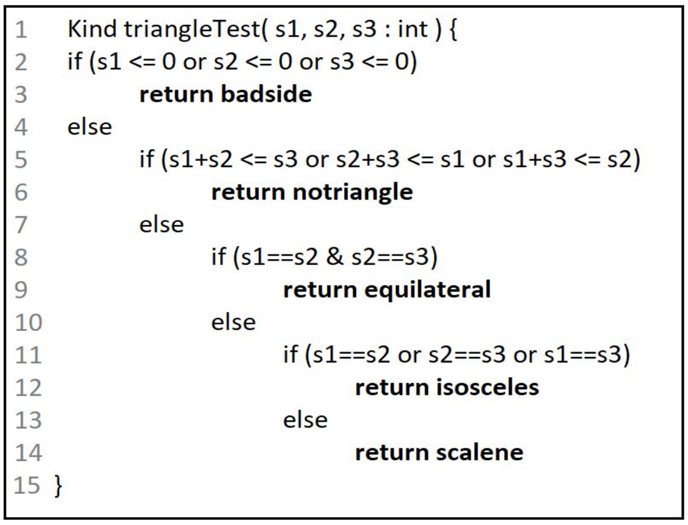
# DD2459: Software Reliability

## Lab 1: White-box Testing

Jonathan Öderyd
19950214-8159
oderyd@kth.se
DD2459 - VT19
TCSCM -17

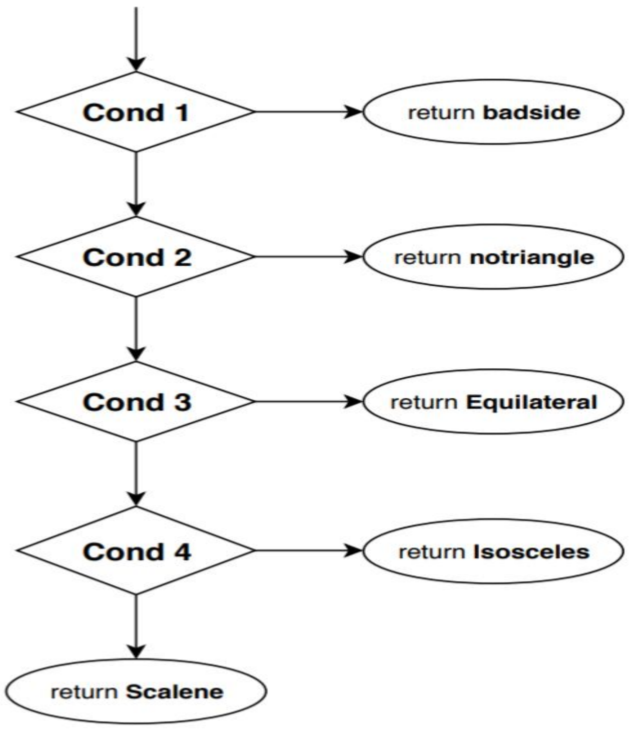
### Triangle Test algorithm



```
Kind triangleTest( s1, s2, s3 : int ) {
if (s1 <= 0 or s2 <= 0 or s3 <= 0)
        return badside
else
        if (s1+s2 <= s3 or s2+s3 <= s1 or s1+s3 <= s2)
                return notriangle
        else
                if (s1==s2 & s2==s3)
                        return equilateral
                else
                        if (s1==s2 or s2==s3 or s1==s3)
                                return isosceles
                        else
                                return scalene
}
```

# Question1 - Control Flow Coverage

**Condensation graph:**



*Cond1, Cond2, Cond3, Cond4 represents the four if-statements. Arrows to the right indicates paths when conditions are evaluated as true, and down arrows indicates false.*

## 1.1 Node Coverage

### 1.1(a) - NC Test Requirements

**NC TR1:** Cond1, badside
**NC TR2:** Cond1, Cond2, notriangle
**NC TR3:** Cond1, Cond2, Cond3, equilateral
**NC TR4:** Cond1, Cond2, Cond3, Cond4, isosceles
**NC TR5:** Cond1, Cond2, Cond3, Cond4, scalene

### 1.1(b) - NC Test Requirements

triangleTest(-1, 1, 1) - corresponds to TR1
triangleTest(1, 1, 3) - corresponds to TR2
triangleTest(3, 3, 3) - corresponds to TR3
triangleTest(2, 3, 2) - corresponds to TR4
triangleTest(4, 5, 6) - corresponds to TR5

## 1.2 Edge Coverage

### 1.2(a) - EC Test Requirements

*(same as 1.1(a))*
**EC TR1:** Cond1, badside
**EC TR2:** Cond1, Cond2, notriangle
**EC TR3:** Cond1, Cond2, Cond3, equilateral
**EC TR4:** Cond1, Cond2, Cond3, Cond4, isosceles
**EC TR5:** Cond1, Cond2, Cond3, Cond4, scalene

### 1.2(b) - EC Test Cases

*(same as 1.1(b))*
triangleTest(-1, 1, 1) - corresponds to TR1
triangleTest(1, 1, 3) - corresponds to TR2
triangleTest(3, 3, 3) - corresponds to TR3
triangleTest(2, 3, 2) - corresponds to TR4
triangleTest(4, 5, 6) - corresponds to TR5

### 1.2(c)

Because every edge has to be visited in order to reach all nodes, an vice versa. Each of the 5 test cases uses at least one unique node (and edge).

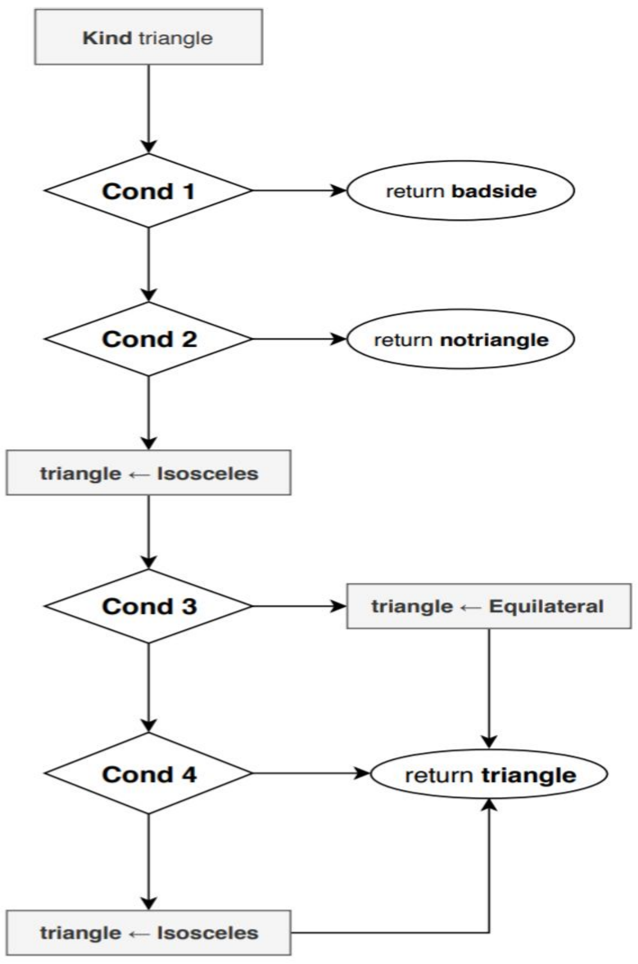
## Question 2 - Logic Coverage

### 2.1 Predicate Coverage

2.1(a) - PC Test Requirements

**PC TR1**: s1<= 0 || s2<= 0 || s3<= 0
**PC TR2**: s1>0 && s2>0 && s3>0
**PC TR3**: (s1+s2<= s3) || (s2+s3 <= s1) || (s1+s3 <= s2)
**PC TR4**: (s1+s2 >s3) && (s2+s3 >s1) && (s1+s3 >s2)
**PC TR5**: s1==s2 & s2==s3
**PC TR6**: s1!=s2 || s2!=s3
**PC TR7**: s1==s2 || s2==s3 || s1==s3
**PC TR8**: s1!=s2 && s2!=s3 && s1!=s3

## 2.1(b) - Modified Condensation graph



## 2.1(c) - PC Test Cases

**PC TC1:** s1=0, s2=0, s3=0 (Badside)
**PC TC2:** s1=1, s2 =1, s3=2 (Notriangle)
**PC TC3:** s1=2, s2 =2, s3=2 (Equilateral)
**PC TC4:** s1=1, s2=2, s3=2 (Isosceles)
**PC TC5:** s1=5, s2 =6, s3=7 (Scalene)

{TC0, TC1, TC2, TC3} achieves nodes coverage.
{TC0, TC1, TC2, TC3, TC4} achieves predicate coverage.

## 2.2 Clause Coverage

2.2(a) - Test Requirements

| | | |
|---|---|---|
| **CC TR1.1:** s1<=0 | **CC TR1.2:** s2<=0 | **CC TR1.3:** s3<=0 |
| **CC TR1.4:** s1 > 0 | **CC TR1.5:** s2 > 0 | **CC TR1.6:** s3 > 0 |
| **CC TR2.1:** s1+s2<= s3 | **CC TR2.2:** s2+s3<= s1 | **CC TR2.3:** s1+s3<= s2 |
| **CC TR2.4:** s1+s2 > s3 | **CC TR2.5:** s2+s3 > s1 | **CC TR2.6:** s1+s3 > s2 |
| **CC TR3.1:** s1 == s2 | **CC TR3.2:** s2 == s3 | **CC TR3.3:** s1 != s2 |
| **CC TR3.4:** s2 != s3 | **CC TR3.5:** s1 == s3 | **CC TR3.6:** s1 != s3 |

2.2(b) Test Cases

| | |
|---|---|
| **CC TC1:** s1=0, s2=0, s3=0 | (TR1.1, TR1.2, TR1.3) |
| **CC TC2:** s1=1, s2=1, s3=2 | (TR1.4, TR1.5, TR1.6, TR2.1, TR2.5, TR2.6) |
| **CC TC3:** s1=2, s2=1, s3=1 | (TR2.2, TR2.4) |
| **CC TC4:** s1=1, s2=2, s3=1 | (TR2.3) |
| **CC TC5:** s1=2, s2=2, s3=1 | (TR3.1, TR3.4, TR3.6) |
| **CC TC6:** s1=1, s2=2, s3=2 | (TR3.2, TR3.3) |
| **CC TC7:** s1=2, s2=1, s3=2 | (TR3.5) |

## 2.3 MCDC Coverage

2.3(a) - MCDC Test Requirements

**TR1**: s1> 0 && s2> 0 && s3> 0 (Cond1 = false)
**TR2**: s1<= 0 && s2> 0 && s3> 0
**TR3**: s1> 0 && s2<= 0 && s3> 0
**TR4**: s1> 0 && s2> 0 && s3<= 0

**TR5:** s1+s2> s3 && s2+s3> s1 && s1+s3> s2 (Cond2 = false)
**TR6:** s1+s2<= s3 && s2+s3> s1 && s1+s3> s2
**TR7:** s2+s3<= s1 && s1+s2> s3 && s1+s3> s2
**TR8:** s1+s3<= s2 && s1+s2> s3 && s2+s3> s1

**TR9:** s1==s2 & s2==s3 (Cond3 = true)
**TR10:** s1!=s2 & s2==s3
**TR11:** s1==s2 & s2!=s3

**TR12:** s1!=s2 && s2!=s3 && s1!=s3 (Cond4 = false)
**TR13:** s1==s2 && s2!=s3 && s1!=s3

**TR14:** s1!=s2 && s2==s3 && s1!=s3
**TR15:** s1!=s2 && s2!=s3 && s1==s3

### 2.3(b) - MCDC Test Cases

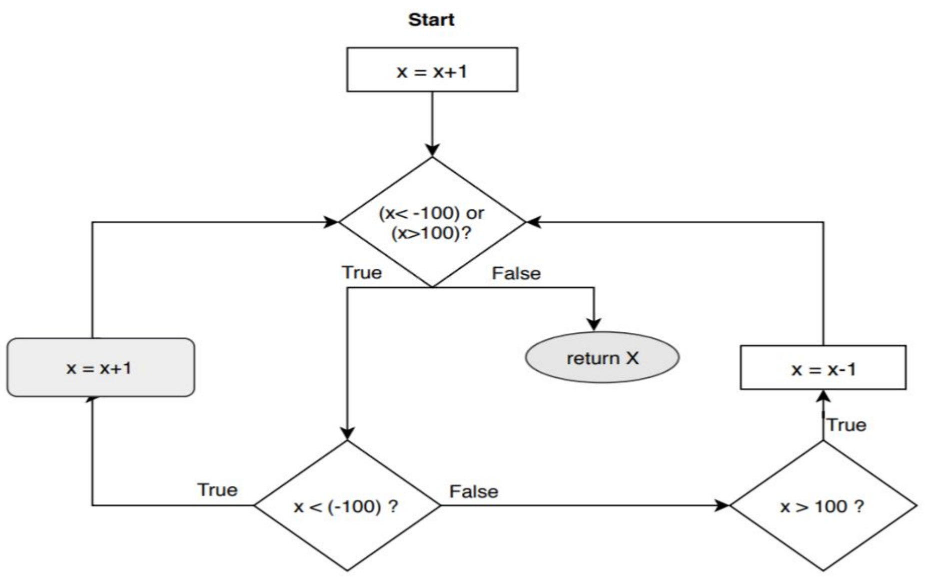
**TC1:** s1=0, s2=1, s3=1
**TC2:** s1=1, s2=0, s3=1
**TC3:** s1=1, s2=1, s3=0

| | |
|---|---|
| **TC4:** s1=3, s2=3, s3=3 | (TR5, TR1 and TR9) |
| **TC5:** s1=3, s2=3, s3=6 | (TR6) |
| **TC6:** s1=6, s2=3, s3=3 | (TR7) |
| **TC7:** s1=3, s2=6, s3=3 | (TR18) |

| | |
|---|---|
| **TC8:** s1=5, s2=6, s3=7 | (TR12) |
| **TC9:** s1=3, s2=3, s3=2 | (TR11, TR13, ) |
| **TC10:** s1=2, s2=3, s3=3 | (TR14) |
| **TC11:** s1=3, s2=2, s3=3 | (TR15) |

# Question 3.

### 3(a)



## 3(b)

*Test requirements:*
**TR1:** x >= 100
**TR2:** x < -101

*Motivation:*
This algorithm will essentially output the input value x+1 if -101<x<100. If x is not within this limit, the output is -100 for negative values and 100 for positive values.
If the input x >=100, the algorithm will first increase the value of x by 1. This would evaluate the first predicate as true and decrement x until x<=100. A test case for input x=100 would reach all nodes except for the one furthest to the left in the graph (indicated by rounded edges).
The graph can be modified slightly by moving this node between the first and second predicate. When doing this it’s important to change the assignment after the third predicate as well from “x=x-1” to “x=x-2”.
If the graph or code is not allowed to change, one additional test requirement is needed for x<-101 to achieve full node coverage.

## 3(c)

**TC1:** x =100
**TC2:** x =-102

## 3(d)

**TC1** and **TC2** achieves node coverage, but the third predicate must be reached and evaluated to false in order to achieve predicate coverage. This is impossible to reach however based on the logic of the code. If the first predicate is true and the second one is false, the last predicate has to be true.

*clauses:*
C1 = x< -100
C2 = x >100
*predicates:*
P1 = C1 || C2
P2 = C1
P3 = C2

P1(true) & P2(false) ⇒ P3=true
I believe that predicate coverage would not provide a better test suite since it could not reach all predicates as true and false. I think a better idea would be to remove the third predicate by removing “if (x > 100) then” from the code. Predicate coverage and node coverage would then be the same.

## Question 4. Self-Assessment

For each of the five sets of test cases you have produced in Questions 1 and 2 (i.e. for each of the five coverage models NC, EC, PC, CC, RACC) , answer the following 14 self assessment questions. For each coverage model, score 1 point for a requirement that is satisfied (maximum 14 points). Which coverage model achieves the highest score?

1. Do you have a test case that represents a valid scalene triangle?
-Yes

2. Do you have a test case that represents a valid equilateral triangle?
-Yes

3. Do you have a test case that represents a valid isosceles triangle?
-Yes

4. Do you have at least three test cases that represent valid isosceles triangles such that you have tried all three permutations of two equal sides?
-Yes, in 2.3(b) TC5, TC6 & TC7

5. Do you have a test case in which one side has a zero value?
-Yes

6. Do you have a test case in which one side has a negative value?
-Yes

7. Do you have a test case with three integers such that the sum of two is equal to the third?
-Yes

8. Do you have at least three test cases in category 7 such that you have tried all three permutations where the length of one side is equal to the sum of the lengths of the other two sides?
-Yes

9. Do you have a test case with three integers greater than zero such that the sum of two numbers is less than the third?
-No, only different cases for when it’s equal

10. Do you have at least three test cases in category 9 such that you have tried all three permutations
-Yes

11. Do you have a test case in which all sides are zero?
-Yes
12. Do you have at least one test case specifying non-integer values? *
-No

13. Do you have at least one test case specifying the wrong number of values (2 or less, four or more) **
-No

14. For each test case, did you specify the expected output from the program in addition to the input values?
-For the ones that are not so obvious, yes. I have not written it for easy cases.