8. Software testing strategies 1

- Definitions and objectives
- Software testing strategies
- Software test classifications
- White box testing
  - Data processing and calculation correctness tests
  - Correctness tests and path coverage
  - Correctness tests and line coverage
  - McCabe’s cyclomatic complexity metrics
  - Software qualification and reusability testing
  - Advantages and disadvantages of white box testing

Software tests - definition

Software Testing is a formal process carried out by a specialized testing team in which a software unit, several integrated units or an entire software package are examined by running the programs on a computer. All the associated tests are performed according to approved test procedures on approved test cases.

- Testing is the single biggest SQA task.
  - on average, 24% of the development budget is testing
- Code Testing ≠ Code Walkthrough

Software testing objectives

- Direct objectives
  - To identify and reveal as many errors as possible in the tested software
  - To bring the tested software, after correction of the identified errors and retesting, to an acceptable level of quality.
  - To perform the required tests efficiently and effectively, within the limits budgetary and scheduling limitation.
- Indirect objectives
  - To compile a record of software errors for use in error prevention (by corrective and preventive actions)

Laws of Testing

- The best person to test your code is someone else.
- A good test is one that finds an error.
- Testing can not prove the absence of errors.
- Complete test coverage is impossible, so concentrate on problem areas.
- It cost a lot less to remove bugs early.
Testing Stages

- Unit Testing
  - modules of code
- Integration Testing
  - design
- Validation Testing
  - requirements
- System Testing
  - system engineering

Reality Check...

- Why not just run the whole thing and see if it gives us the right answer or if it crashes?

Software testing strategies

- Incremental testing strategies:
  - Bottom-up testing
  - Top-down testing
- Big bang testing

Bottom-up testing
**Top-down testing**

Diagram showing stages 1 to 6 with integration and module testing.

**Use of stubs and drivers for Incremental testing**

Diagram showing top-down and bottom-up testing of module M8.

**Black box and white box - IEEE definitions**

**Black box testing** (functional/behavioral)
1. Testing that ignores the internal mechanism of the system or component and focuses solely on the outputs in response to selected inputs and execution conditions
2. Testing conducted to evaluate the compliance of a system or component with specified functional requirements

**White box testing** (glass box)
Testing that takes into account the internal mechanism of a system or component

**White Box Testing Fundamentals**

- White Box testing is much more expensive than Black Box testing.
- White Box is most appropriate when we must assure that the calculations are correct.
- Covering every possible path through a module is usually not practical.
  - 10 if-then statements might require 1024 test cases
  - instead, base the number of tests on the complexity of the module
Types of Code Coverage

- **Line Coverage**
  - Has every possible line of code been executed?

- **Function coverage**
  - Has each function in the program been executed?

- **Statement coverage**
  - Has each line of the source code been executed?

- **Condition coverage**
  - Has each evaluation point (such as a true/false decision) been executed?

- **Path coverage**
  - Has every possible route through a given part of the code been executed?

- **Entry/exit coverage**
  - Has every possible call and return of the function been executed?

Example One

```java
int example1 (int value, boolean cond1, boolean cond2)
{
    if ( cond1 )
        value ++;
    if ( cond2 )
        value --;
    return value;
}
```

- Total **Statement Coverage** with one case - True True.
- Total **Path Coverage** with four paths - TT TF FT FF.
- But, total path coverage is usually impractical, so Basis Path Testing is usually better.

Basis Path Testing

Objective is to test each conditional statement as both true and false

1. Draw a Flow Graph
2. Determine the Cyclomatic Complexity
   - CC = number of regions
   - CC = E - N + 2
3. Max Number of tests = CC
4. Derive a basis set of independent paths
5. Generate data to drive each path

Flow Graphs

- simple sequence
- if then else
- while
- do until
- case statement
Example One - using basis path

```java
int example1 (int value, boolean cond1, boolean cond2)
{
1    if ( cond1 )
2       value ++;
3    if ( cond2 )
4       value --;
5    return value;
}
Complexity = 3
```

Basis Paths | Test Data
---|---
1 3 5 | False False
1 2 3 5 | True False
1 2 3 4 5 | True True

Example One - sample driver

```
Test Data
false false
true false
true true
```

```java
int example1 (int value, boolean cond1, boolean cond2)
{
    ...
    }
print ("test one ", example1 (5, false, false));
print ("test two ", example1 (5, true, false));
print ("test three ", example1 (5, true, true));
```

Example Two

```java
float avg_negative_balance (int arraysize, float balances[])
{
    float total = 0.0;
    int count = 0;
    for I = 1 to arraysize
        if ( balances[I] < 0 )
            total += balances[I];
            count ++;
        end if;
    end for;
    return total / count;
}
```

Example Two - using basis path

```
float avg_negative_balance (int arraysize, float balances[])
{
    float total = 0.0;
    int count = 0;
    for I = 1 to arraysize
        if ( balances[I] < 0 )
            total += balances[I];
            count ++;
        end if;
    end for;
    return total / count;
}
```

Basis Paths | Test Data
---|---
1 2 5 | arraysize = 0
1 2 3 2 5 | arraysize = 1, balance[1] = 25
1 2 3 4 2 5 | arraysize = 1, balance[1] = -25
Example Two - Test Report

<table>
<thead>
<tr>
<th>Test #</th>
<th>Test Data</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>array size = 0</td>
<td>failed</td>
</tr>
<tr>
<td>2</td>
<td>size = 1</td>
<td>passed</td>
</tr>
<tr>
<td></td>
<td>array = [ 25 ]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>size = 1</td>
<td>passed</td>
</tr>
<tr>
<td></td>
<td>array = [ -25 ]</td>
<td></td>
</tr>
</tbody>
</table>

Errors not detected:
- precision errors when "total" gets very small

---

Loop Testing

Errors often occur near the beginnings and ends of loops.

- For each loop that iterates max N times, test
  - N = 0
  - N = max - 1
  - N = max
  - N = max + 1

- For nested loops
  - repeat above for the innermost loop, outer loop iterates once
  - then repeat all 5 possibilities for outer loop, while inner loop iterates only once

---

Example Two - using loop testing

```c
float avg_negative_balance (int arraysize, float balances[])
{
    float total = 0.0;
    int count = 0;
    for I = 1 to arraysize
        if ( balances[I] < 0 )
            total += balances[I];
            count ++;
        end if;
    end for;
    return total / count;
}
```

Test Case      Test Data
N = 0          size = 0
N = max-1      size = 999 (if SRS says max=1000)
N = max         size = 1000
N = max+1      size = 1001, but array has only 1000 elements

---

The Imperial Taxi Services (ITS) taximeter

Minimal fare: $2. This fare covers the distance traveled up to 1000 yards and waiting time (stopping for traffic lights or traffic jams, etc.) of up to 3 minutes. For every additional 250 yards or part of it: 25 cents. For every additional 2 minutes of stopping or waiting or part thereof: 20 cents. One suitcase: 0 change; each additional suitcase: $1. Night supplement: 25%, effective for journeys between 21.00 and 06.00. Regular clients are entitled to a 10% discount and are not charged the night supplement.
**ITS - Flow chart**

1. Charge the minimal fare
   - Charge the minimal fare
2. Distance
   - Distance
3. Distance $> 1000$
4. Distance $\leq 1000$
5. Waiting time
   - Waiting time
6. Waiting time $> 3$
7. Waiting time $\leq 3$
8. No. of suitcases
   - No. of suitcases
9. No. of suitcases $> 1$
10. No. of suitcases $\leq 1$
11. Regular client?
   - Regular client?
12. Yes
13. No
14. Night journey?
   - Night journey?
15. Yes
16. No
17. Print receipt

---

**ITS - Program flow graph**

1. R1
2. R2
3. R3
4. R4
5. R5
6. R6
7. R7
8. R8
9. R9
10. R10
11. R11
12. R12
13. R13
14. R14
15. R15
16. R16

---

**ITS - The minimum number of paths for full line coverage**

1. R1
2. R2
3. R3
4. R4
5. R5
6. R6

---

**ITS - The maximum set of independent paths**

1. R1
2. R2
3. R3
4. R4
5. R5
6. R6

---

McCabe's cyclomatic complexity metrics
Advantages and disadvantages of white box testing

- **Advantages:**
  - Direct determination of software correctness as expressed in the processing paths, including algorithms.
  - Allows performance of line coverage follow up.
  - Ascertains quality of coding work and its adherence to coding standards.
- **Disadvantages:**
  - The vast resources utilized, much above those required for black box testing of the same software package.
  - The inability to test software performance in terms of availability (response time), reliability, load durability, etc.