Final-Term Assignment

Software Verification and validation

Submitted By: Mudassir Ahmad Khan
ID# 14086
BSSE *(6th Semester)*

Submitted To: Mr. Zain Shaukat
(Lecturer)

Date: 25/06/2020

Question No: 01 MCQS

1. When should company stop the testing of a particular software?

a) After system testing done
b) It depends on the risks for the system being tested
c) After smoke testing done
d) None of the above **Ans: b**

2. White-Box Testing is also known as ________ .

a) Structural testing
b) Code-Based Testing
c) Clear box testing
d) All of the above **Ans: d**

3. ___________ refers to a different set of tasks ensures that the software that has been built is traceable to Customer Requirements.

a) Verification
b) Requirement engineering
c) Validation
d) None of the above **Ans: c**

4. ________ verifies that all elements mesh properly and overall system functions/performance is achieved.

a) pIntegration testing
b) Validation testing
c) Unit testing
d) System Testing **Ans: d**

5. What do you verify in White Box Testing?

a) Testing of each statement, object and function on an individual basis.
b) Expected output.
c) The flow of specific inputs through the code.
d) All of the above. **Ans: d**

6. __________ refers to the set of tasks that ensures the software correctly implements a specific function.

a) Verification
b) Validation
c) Modularity
d) None of the above. **Ans: a**

7. Who performs the Acceptance Testing?

a) Software Developer
b) End users
c) Testing team
d) Systems engineers **Ans: b**

8. Which of the following is not a part of Performance Testing?

a) Measuring Transaction Rate.
b) Measuring Response Time.
c) Measuring the LOC.
d) None of the above. **Ans: c**

9. Which of the following can be found using Static Testing Techniques?

a) Defect
b) Failure
c) Both A & B **Ans: a**

10. Testing of individual components by the developers are comes under which type of testing?

a) Integration testing
b) Validation testing
c) Unit testing
d) None of the above. **Ans: c**

Question No: 02

Explain Black Box testing and White Box testing in detail.

Ans:

Black Box testing:

Definition: Black-box testing is a testing strategy that ignores the internal mechanism of a system or component and focuses solely on outputs generated in response to selected inputs and execution conditions.

Description: In black box testing, the structure of the program is not taken into consideration. It takes into account functionality of the application only. It is also called functional testing.

Tester is mainly concerned with the validation of the output rather than how the output is produced. Knowledge of programming or implementation logic (of internal structure and working) is not required for testers. It is applicable mainly at higher levels of testing - Acceptance Testing and System Testing.

The software into which known inputs are fed and where known outputs are expected is termed a black box. The transformation of the known inputs to the known outputs is done via the system and is not checked in this kind of testing. This transformation process system is called the black box.

In this kind of testing, the testers concentrate on functional testing, that is, on providing a known input and check if the known output is obtained. This method is generally followed while carrying out acceptance testing, when the end user is not a software developer but only a user.

It is different from white box testing in the sense that in white box testing, the tester ought to have the programming knowledge and understanding of code to test the application whereas it may not be the case in black box testing.

Techniques that are used in black box testing are:

1. Identification of equivalence class: Partition any input domain into minimum two sets: valid values and invalid values. For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.

2. Generating test cases:
(i) To each valid and invalid class of input assign unique identification number.
(ii) Write test case covering all valid and invalid test case considering that no two invalid inputs mask each other.

3. Boundary value analysis: Boundaries are very good places for errors to occur. Hence if test cases are designed for boundary values of input domain then the efficiency of testing improves and probability of finding errors also increase. For example – If valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.

4. Cause effect Graphing: This technique establishes relationship between logical input called causes with corresponding actions called effect. The causes and effects are represented using Boolean graphs.

5. Requirement based testing: It includes validating the requirements given in SRS of software system.

6. Compatibility testing: The test case result not only depend on product but also infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly.

White Box testing

WHITE BOX TESTING (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.

White Box Testing Techniques:

Statement Coverage: This technique is aimed at exercising all programming statements with minimal tests.

Branch Coverage: This technique is running a series of tests to ensure that all branches are tested at least once.

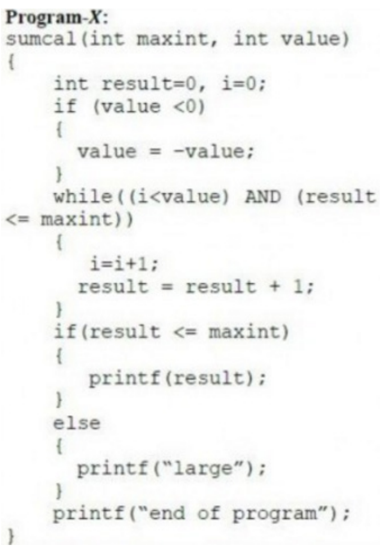
Path Coverage: This technique corresponds to testing all possible paths which means that each statement and branch is covered.

White Box Testing is coverage of the specification in the code:

1. Code coverage
2. Segment coverage: Ensure that each code statement is executed once.
3. Branch Coverage or Node Testing: Coverage of each code branch in from all possible was.
4. Compound Condition Coverage: For multiple conditions test each condition with multiple paths and combination of the different path to reach that condition.
5. Basis Path Testing: Each independent path in the code is taken for testing.
6. Data Flow Testing (DFT): In this approach you track the specific variables through each possible calculation, thus defining the set of intermediate paths through the code.DFT tends to reflect dependencies but it is mainly through sequences of data manipulation. In short, each data variable is tracked and its use is verified. This approach tends to uncover bugs like variables used but not initialize, or declared but not used, and so on.
7. Path Testing: Path testing is where all possible paths through the code are defined and covered. It’s a time-consuming task.
8. Loop Testing: These strategies relate to testing single loops, concatenated loops, and nested loops. Independent and dependent code loops and values are tested by this approach.

Question No: 03

Find the cyclomatic Complexity and draw the Graph of this code.



Program-*X*:

```
sumcal(int maxint, int value)
{
    int result=0, i=0;
    if (value <0)
    {
      value = -value;
    }
    while((i<value) AND (result
<= maxint))
    {
       i=i+1;
      result = result + 1;
    }
    if(result <= maxint)
    {
       printf(result);
    }
    else
    {
      printf("large");
    }
    printf("end of program");
}
```

Solution:

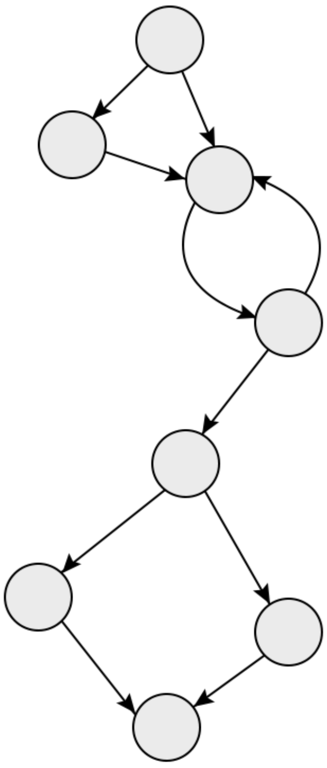
<u>Complexity:</u>

cyclomatic Complexity of program is the number of conditions + 1.

There are 2 'if' condition and 1 'while' condition

Program 'X' = 4

<u>Control flow diagram for Program:</u>



Question No: 04

What is Z specification and why its is used for, also give some example this code written in Z specification.

Ans:

The Z language is a model oriented, formal specification language that was proposed by Jean-Raymond Abrail, Steve Schuman and Betrand Meyer in 1977 and it was later further developed at the programming research group at Oxford University. It is based on Zermelo Fränkel axiomatic set theory and first order predicate logic. The Z notation, is a strongly typed, mathematical, specification language. It has robust commercially available tool support for checking Z texts for syntax and type errors in much the same way that a compiler checks code in an executable programming language. It cannot be executed, interpreted or compiled into a running program. It allows specification to be decomposed into small pieces called schemas. The schema is the main feature that distinguishes Z from other formal notations. In Z, both static and dynamic aspects of a system can be described using schemas. The Z specification describes the data model, system state and operations of the system. Z specification is useful for those who find the requirements, those who implement programs to meet those requirements, those who test the consequences, and those who write instruction manuals for the system.

Z is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. All expressions in Z notation are typed, thereby avoiding some of the paradoxes of naive set theory. Z contains a standardized catalogue (called the mathematical toolkit) of commonly used mathematical functions and predicates, defined using Z itself.

Although Z notation (just like the APL language, long before it) uses many non-ASCII symbols, the specification includes suggestions for rendering the Z notation symbols in ASCII and in LaTeX. There are also Unicode encodings for all standard Z symbols.

Example No. 01

Library

```
Library
  books: $\mathbb{F}$ Copy
  records: Copy $\rightarrowtail$ Data
  users, staff: REAL_PERSON
  ---------------------------------
  $\forall b_1, b_2 : books \bullet b_1.id = b_2.id \Leftrightarrow b_1 = b_2$
  dom records = books
  $\forall$ user: users $\bullet$
      # {b: books | (records b).lastuser = user $\wedge$
          (records b).status = out} $\leq$ MAX
```

Example No. 02

Books and Copies

[AUTHOR, TITLE, SUBJECT, BOOKID]

Book
author: AUTHOR
title: TITLE
subjects: $\mathbb{P}$ SUBJECT

Copy
Book
id: BOOKID

Example No. 03

Transient Book Information

[PERSON]
STATUS ::= out | in

Data
lastuser: PERSON
status: STATUS

MAX: $\mathbb{N}$
nobody: PERSON
REAL_PERSON == PERSON \ {nobody}

Thank You!