Name	:	Shahid Ali
Id	:	13942
Subject	:	Software Verification and Validation
Degree	:	BS(SE)
Module	:	6th Semester
Submitted to	:	Mr. Zain Shoukat
Date	:	25/06/2020

Q.1: MCQS

1. When should company stop the testing of particular software?
Ans: b. It depends on the risks for the system being tested

2. White-Box Testing is also known as _____ .
Ans: d. All of the above

3. _____ refers to a different set of tasks ensures that the software that has been built is traceable to Customer Requirements.
Ans: c. Validation

4. _____ verifies that all elements mesh properly and overall system functions/performance is achieved.
Ans: d. System Testing

5. What do you verify in White Box Testing?
- *Published on 03 Aug 15*.
Ans: d. All of the above.

6. _____ refers to the set of tasks that ensures the software correctly implements a specific function.
- *Published on 03 Aug 15*
Ans: a. Verification

7. Who performs the Acceptance Testing?
- *Published on 03 Aug 15*
Ans: b. End users

8. Which of the following is not a part of Performance Testing?
- *Published on 30 Jul 15*
Ans: c. Measuring the LOC.

9. Which of the following can be found using Static Testing Techniques?
- *Published on 29 Jul 15*
Ans: a. Defect

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

- *Published on 29 Jul 15*
Ans: c. Unit testing.

Q2. Explain Black Box testing and White Box testing in detail.
Ans: Black Box Testing: BLACK BOX TESTING is defined as a testing technique in which functionality of the Application under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications. In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program

The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application.Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Black Box Techniques:

1) **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.
2) **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.
3) **Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.

White Box Testing: White Box Testing is testing a software solution's internal structure, design and coding. It is also known as clear box testing, open box testing, structural testing, transparent box testing, code-based testing, and Glass Box testing. It is usually performed by developers. In this type of testing, the code is visible to the tester. It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security.It is one of two parts of the box testing approach to software testing. Its counterpart, Black box testing, involves testing from an external or and-user type perspective. On the other hand, white box testing is based on the inner workings of an application and revolves around internal testing. The term " white box" was used because of the see-through box concept. The clear box or white box name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

White Box Testing Techniques:

1) **Statement Coverage:** this technique is aimed at exercising all programming statements with minimal tests.
2) **Branch Coverage:** this technique is running a series of tests to ensure that all branches are tested at least once.

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

Q3. 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");
}
```

Ans: Cyclomatic complexity will be equal to four (4)

Formula;

1. Cyclomatic complexity = No of predictes +1

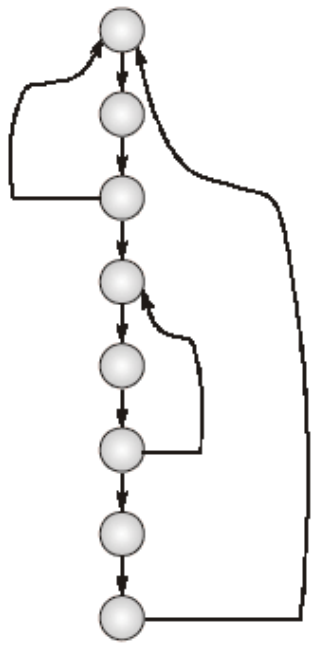
For the given program, predicates are if, while. Total 2 if and 1 while condition.

So answer will be 4.

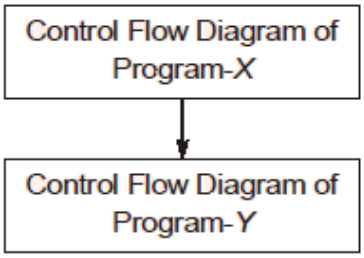
Program-*X*:

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

Control Flow Diagram of Program-*Y*:



Control Flow Diagram of Program-*Z*:



The values of McCabe's Cyclomatic complexity of Program-X, Program-Y and Program-Z respectively are

(A) 4, 4, 7
(B) 3, 4, 7
(C) 4, 4, 8
(D) 4, 3, 8

Answer: (A)

Explanation:

The cyclomatic complexity of a structured program[a] is defined with reference to the control flow graph of the program, a directed graph containing the basic blocks of the program, with an edge between two basic blocks if control may pass from the first to the second. The complexity M is then defined as.

M = E − N + 2P,

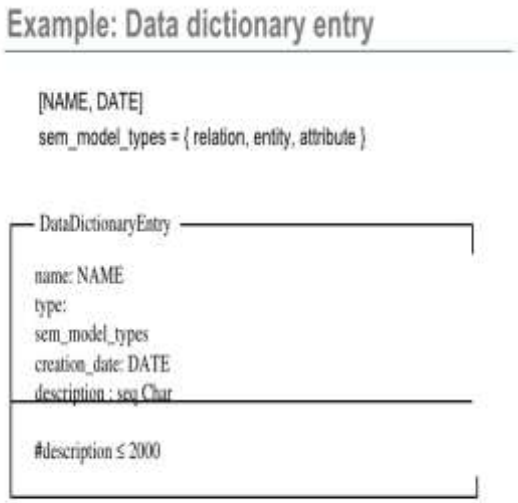
where

E = the number of edges of the graph.

N = the number of nodes of the graph.

P = the number of connected components

Q4. What is Z specification and why it is used for; also give some example this code written in



Z specification.

ANS: Z specification: The Z notation / zEd / is a formal specification language used for describing and modeling computing systems. It is targeted at the clear specification of

computer programs and computer-based systems. In general, Z specification cannot be interpreted or compiled into a running program (or prototype) Z is not a programming. Z is a model-based notation. In Z you usually model a system a system by representing its state- a collection of state variable and their values- and some operations that can change its state. A model that is characterized by the operation it describes is called an abstract date type (ADP). Z was designed for people, not machines. Z based on the standard mathematical notation used in axiomatic set theory, lambda calculus and first-order predicate logic. All expressions in notation are typed, thereby avoiding some of the paradoxes of naïve set theory. Z contains a standard catalogue (called the mathematical toolkit) of commonly used mathematical functions and predicates, defined using Z itself.

Why Z ?

- Expressive power.
- Precise formalism.
- Can be used to model a broad range of systems.
- Accuracy important for safety-critical systems.

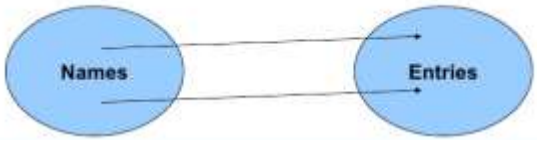
Data dictionary modeling

- A data dictionary may be thought of as a mapping from a name (the key) to a value (the description in the dictionary)
- Operations are
  - Add. Makes a new entry in the dictionary or replaces an existing entry
  - Lookup. Given a name, returns the description.
  - Delete. Deletes an entry from the dictionary
  - Replace. Replaces the information associated with an entry

Basic Data Representation

DataDictionary

$ddict : NAME \pfun DataDictionaryEntry$



Function Summary

Name	Symbol	dom f	One-to-one?	ran f
Total function	$\rightarrow$	= X		$\subseteq$ Y
Partial function	$\nrightarrow$	$\subseteq$ X		$\subseteq$ Y
Injection (total)	$\rightarrowtail$	= X	Yes	$\subseteq$ Y
Surjection (total)	$\twoheadrightarrow$	= X		= Y
Bijection	$\rightarrowtail\!\!\!\!\twoheadrightarrow$	= X	Yes	= Y

Data dictionary initialization

Init_DataDictionary

Δ DataDictionary

---

ddict' = ϕ

Add and lookup operations

Add_OK

Δ DataDictionary
entry?: DataDictionaryEntry

Accessing sub elements

---

entry?.name $\notin$ dom ddict
ddict' = ddict $\cup$ { entry?.name $\rightarrow$ entry? }

Lookup_OK

Ξ DataDictionary
name?: NAME
entry!: DataDictionaryEntry

---

name? $\in$ dom ddict
entry! = ddict(name?)

Add and lookup operations