NAME : MASOOD SAID

ID : 13723

SEMESTER : 6th **FINAL TERM**

PAPER : Software Verification and validation

INSTRUCTOR : ZAIN SHAUKAT

Q1. MCQS

ANS :

1) b

2) d

3) C

4) d

5) d

6) a

7) b

8) c

9) a

10) c

Q2. Explain Black Box testing and White Box testing in detail.

ANS : BLACK BOX TESTING :

Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester.

It is also called as Behavioral/Specification-Based/Input-Output Testing

This can be applied to every level of software testing such as Unit, Integration, System and Acceptance Testing.

Testers create test scenarios/cases based on software requirements and specifications. So it is AKA Specification Based Testing.

The tester passes input data to make sure whether the actual output matches the expected output. So it is AKA Input-Output Testing.

Types of Black Box Testing:

Functionality Testing: In simple words, what the system actually does is functional testing

Non-functionality Testing: In simple words, how well the system performs is non-functionality testing

Black Box Testing Techniques:

1)Equivalence Partitioning

2)Boundary Value Analysis

3)Decision Table

4)State Transition

Equivalence Partitioning: Equivalence Partitioning is also known as Equivalence Class Partitioning. In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be proposed in the same way.

Boundary Value Analysis: Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions. The Behavior at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.

Decision Table: Decision Table is aka Cause-Effect Table. This test technique is appropriate for functionalities which has logical relationships between inputs (if-else logic). In Decision table technique, we deal with combinations of inputs. To identify the test cases with decision table, we consider conditions and actions.

State Transition: Using state transition testing, we pick test cases from an application where we need to test different system transitions. We can apply this when an application gives a different output for the same input, depending on what has happened in the earlier state.

WHITE BOX TESTING :

White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.

It is also called as Glass Box, Clear Box, Structural Testing.

Types of White Box Testing

1)Unit Testing

2)Static Analysis

3)Dynamic Analysis

4)Statement Coverage

Unit Testing

Unit Testing is one of the basic steps, which is performed in the early stages. Most of the testers prefer performing to check if a specific unit of code is functional or not. Unit Testing is one of the common steps performed for every activity because it helps in removing basic and simple errors.

Static Analysis

As the term says, the step involves testing some of the static elements in the code. The step is conducted to figure out any of the possible defects or errors in the application code.

The static analysis is an important step because it helps in filtering simple errors in the initial stage of the process.

Dynamic Analysis

Dynamic Analysis is the further step of static analysis in general path testing. Most of the people prefer performing both static and dynamic at the same time. The dynamic analysis helps in analyzing and executing

the source code depending on the requirements. The final stage of the step helps in analyzing the output without affecting the process.

Statement Coverage

Statement coverage is one of the pivotal steps involved in the testing process. It offers a whole lot of advantages in terms of execution from time to time.

The process takes place to check whether all the functionalities are working or not. Most of the testers use the step because it is designed to execute all the functions atleast once. As the process starts, we will be able to figure out the possible errors in the web application.

Q3. Find the cyclomatic Complexity and draw the Graph of this code?

Program-*X*:

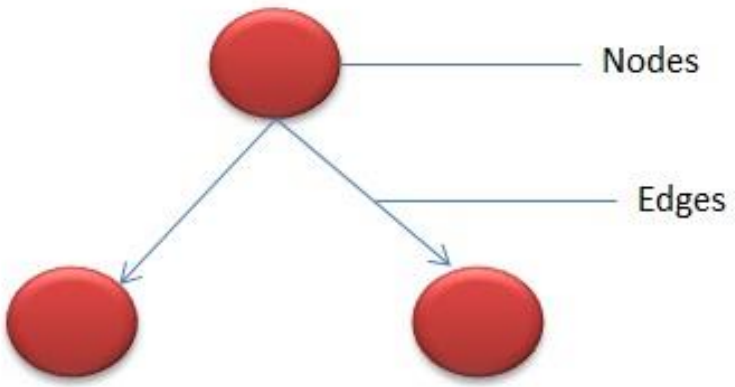
```
sumcal(int maxint, int value)
{
    int result=0, i=0;
    if (value <0)
    {
      value = -value;
    }
    while((i<value) AND (result
<= maxint))
    {
       i=i+1;
      result = result + 1;
    }
    if(result <= maxint)
    {
       printf(result);
    }
    else
    {
      printf("large");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

ANS : CYCLOMATIC COMPLEXITY:-

CYCLOMATIC COMPLEXITY is a software metric used to measure the complexity of a program. It is a quantitative measure of independent paths in the source code of the program. Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths. Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program.

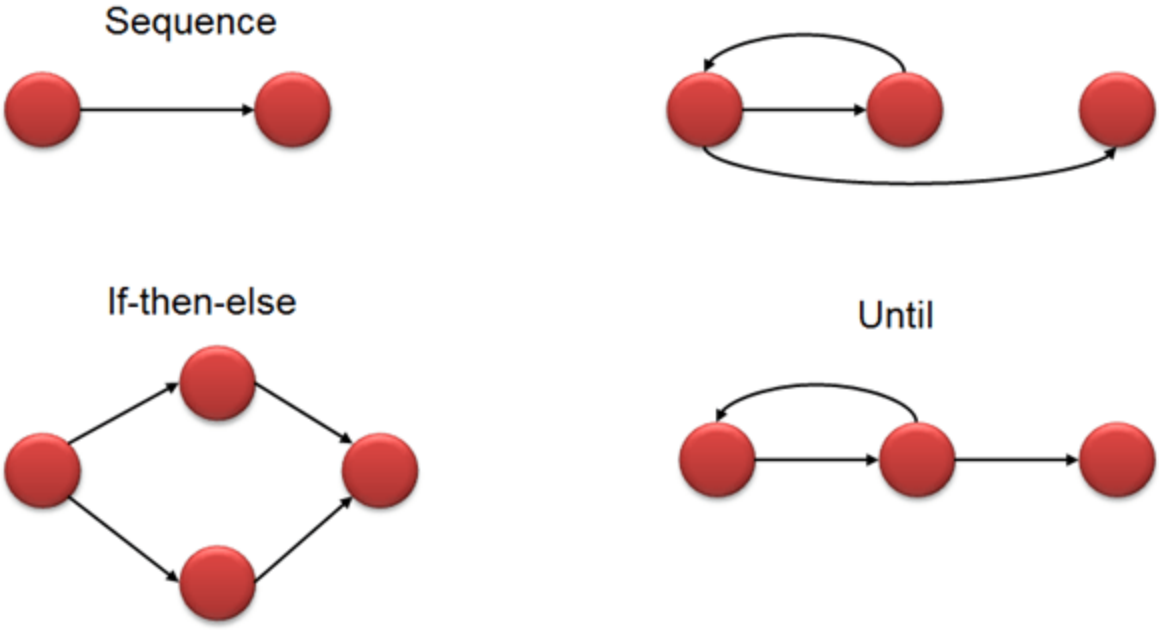
This metric was developed by Thomas J. McCabe in 1976 and it is based on a control flow representation of the program. Control flow depicts a program as a graph which consists of Nodes and Edges.

In the graph, Nodes represent processing tasks while edges represent control flow between the nodes.



Flow graph notation for a program:

Flow Graph notation for a program defines several nodes connected through the edges. Below are Flow diagrams for statements like if-else, While, until and normal sequence of flow.

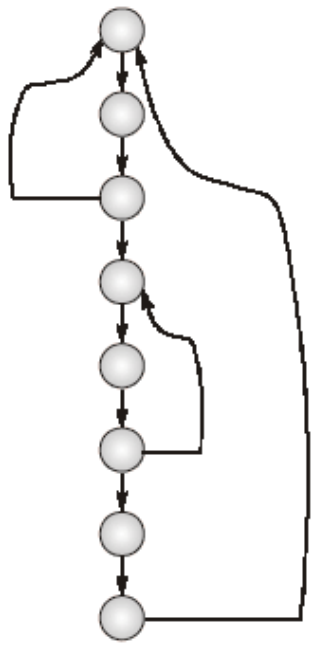


Consider three software items: Program-X, Control Flow Diagram of Program-Y and Control Flow Diagram of Program-Z as shown below

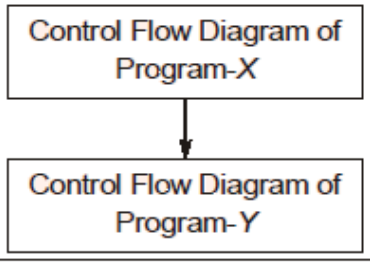
Program-*X*:

```
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

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.

$M = E - N + 2P,$

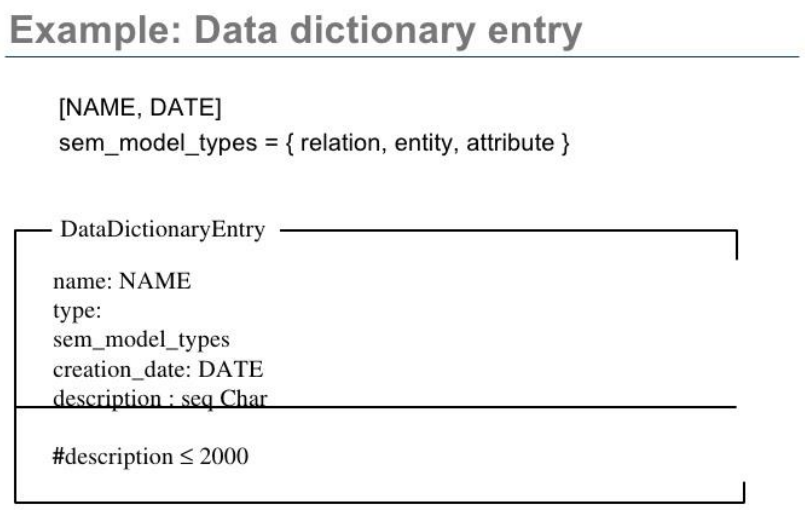
Where

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?



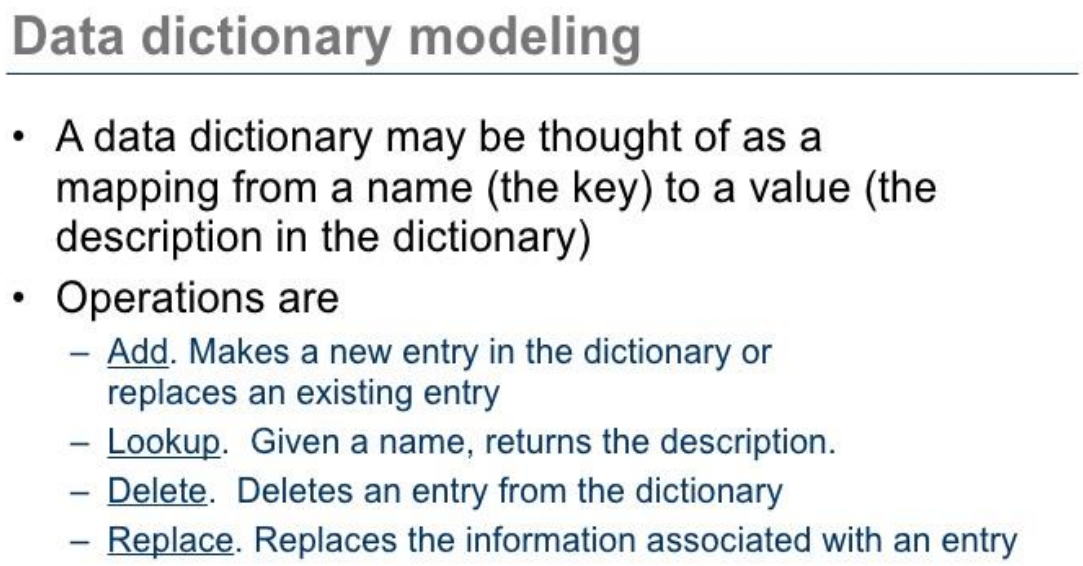
ANS :

The Z notation is a formal specification language used for describing and modelling computing systems. It is targeted at the clear specification of computer programe and computer-based systems in general

Usage and notation

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.

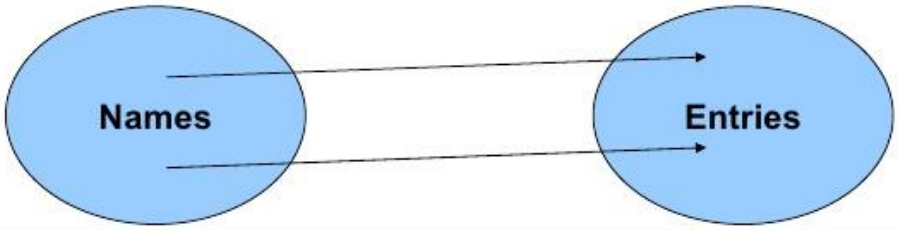
Although Z notation (just like the APL language, long before it) uses many non-ASCII symbols, the specification includes suggestions for rendering the Z notation symbols in ASCII and in LaTeX. There are also Unicode.



Basic Data Representation

DataDictionary

$ddict : NAME \pfun DataDictionaryEntry$

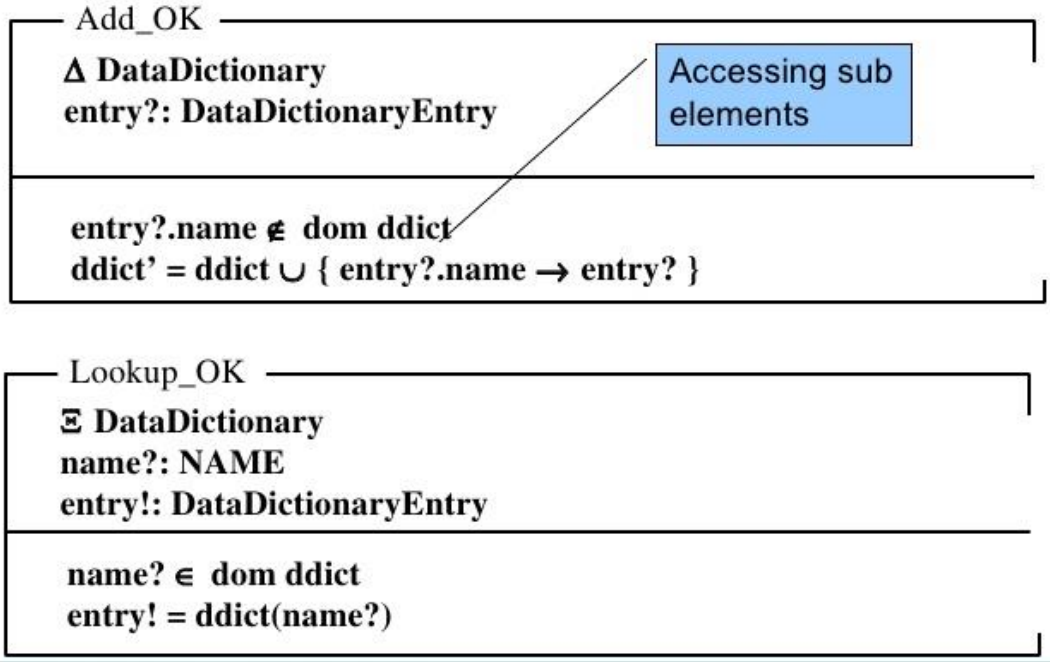


Function Summary

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

Add and lookup operations



Add and lookup operations

Add_Error

Ξ DataDictionary
entry?: DataDictionaryEntry
error!: seq char

---

entry?.name ∈ dom ddict
error! = "Name already in dictionary"

Lookup_Error

Ξ DataDictionary
name?: NAME
error!: seq char

---

name? ∉ dom ddict
error! = "Name not in dictionary"

THE END