FINAL ASSIGNMENT

NAME MUHAMMAD SOHAIL

ID # 14071

DAGREE BS (SE)

COURSE NAME *SOFTWARE* VERIFICATION &VALIDATION

INSTUCTOR NAME ZAIN SHAUKAT

SEMESTER 6TH

DATE 25/06/2020

Q1. MCQS (10)

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

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

a. Structural testing
b. Code-Based Testing
c. *Clear box testing*
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.

a. Verification
b. Requirement engineering
c. *Validation*
d. None of the above

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

a. Integration testing
b. Validation testing
c. Unit testing
d. *System Testing*

5. What do you verify in White Box Testing?
- Published on 03 Aug 15

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.*

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

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

7. Who performs the Acceptance Testing?
- Published on 03 Aug 15

a. Software Developer
b. End users
c. Testing team
d. Systems engineers

8. Which of the following is not a part of Performance Testing?
- Published on 30 Jul 15

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

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

a. *Defect*
b. Failure
c. Both A & B

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

- Published on 29 Jul 15

a. Integration testing
b. Validation testing
c. Unit testing
d. None of the above.

What is 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 BlackBox 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.

How to do BlackBox Testing

Here are the generic steps followed to carry out any type of Black Box Testing.

- Initially, the requirements and specifications of the system are examined.
- Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
- Tester determines expected outputs for all those inputs.
- Software tester constructs test cases with the selected inputs.
- The test cases are executed.
- Software tester compares the actual outputs with the expected outputs.
- Defects if any are fixed and re-tested.

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones -

- **Functional testing** - This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** - Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Tools used for Black Box Testing:

Tools used for Black box testing largely depends on the type of black box testing you are doing.

- For Functional/ Regression Tests you can use - QTP, Selenium
- For Non-Functional Tests, you can use - LoadRunner, Jmeter

Black Box Testing Techniques

Following are the prominent Test Strategy amongst the many used in Black box Testing

- **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.
- **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.
- **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 (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.

Definition by ISTQB

- **white-box testing:** Testing based on an analysis of the internal structure of the component or system.
- **white-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

Example

A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.

White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

Levels Applicable To

White Box Testing method is applicable to the following levels of software testing:

- Unit Testing: For testing paths within a unit.
- Integration Testing: For testing paths between units.
- System Testing: For testing paths between subsystems. However, it is mainly applied to Unit Testing.

Advantages

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.

- Testing is more thorough, with the possibility of covering most paths.

Disadvantages

- Since tests can be very complex, highly skilled resources are required, with a thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing is closely tied to the application being tested, tools to cater to every kind of implementation/platform may not be readily available.
White Box Testing is contrasted with Black Box Testing. Read the Differences between Black Box Testing and White Box Testing.

Q4. What is Z specification and why its is used for, also give some example this code written in Z specification?

The **Z notation** is a formal specification language used for describing and modelling computing systems. It is targeted at the clear specification of computer programs 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

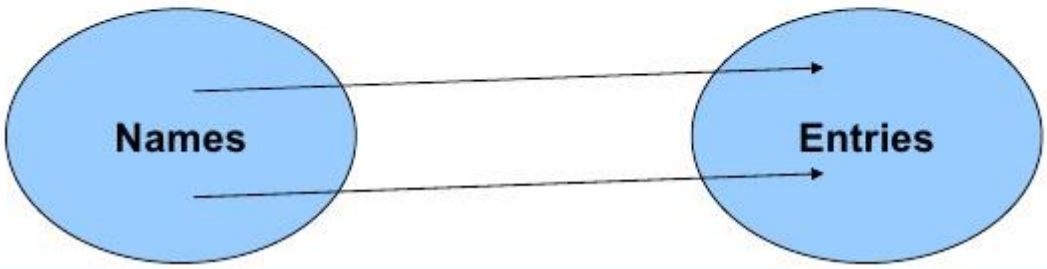
Data dictionary modeling

- 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 ⇸ 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_OK

Δ DataDictionary
entry?: DataDictionaryEntry

Accessing sub elements

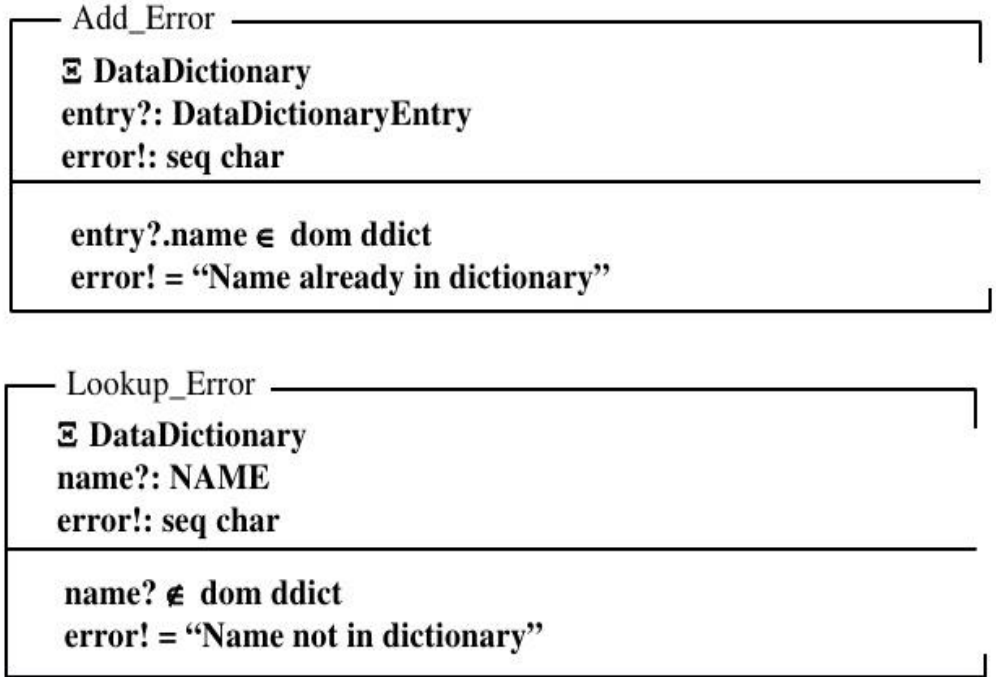
entry?.name $\notin$ dom ddict
ddict' = ddict $\cup$ { entry?.name $\rightarrow$ entry? }

Lookup_OK

Ξ DataDictionary
name?: NAME
entry!: DataDictionaryEntry

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

Add and lookup operations

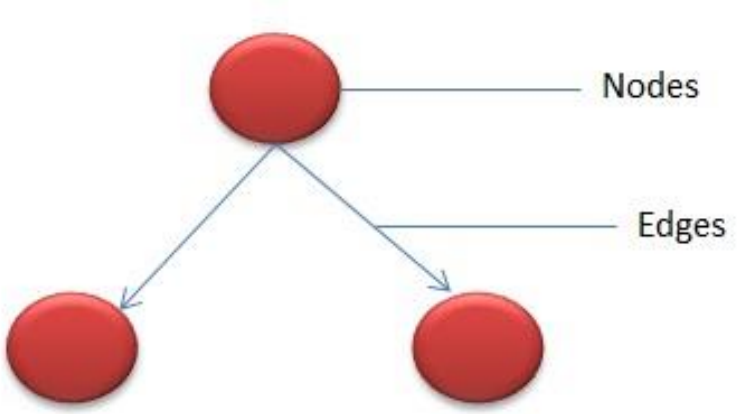


QUESTION 03

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.

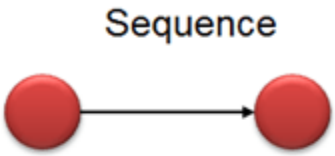
This metric was developed by Thomas J. McCabe in 1976 and it is based on a control flow representation of the program. Control flow depicts a program as a graph which consists of Nodes and Edges.

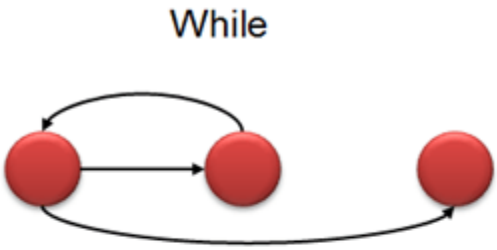
In the graph, Nodes represent processing tasks while edges represent control flow between the nodes.

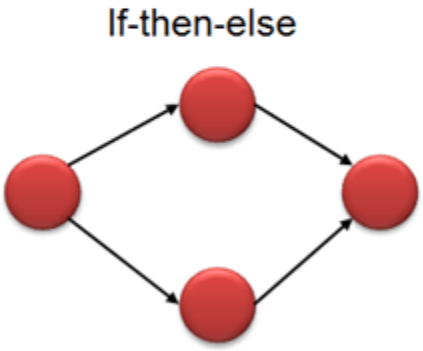


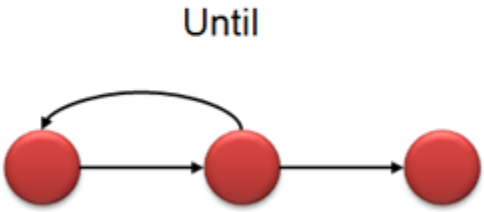
Flow graph notation for a program:

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.







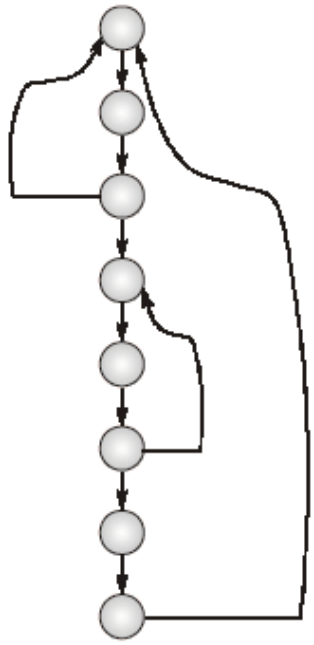


Consider three software items: Program-X, Control Flow Diagram of Program-Y and Control Flow Diagram of Program-Z as shown below

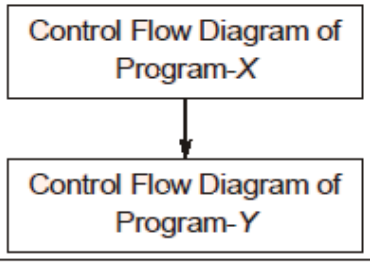
Program-*X*:

```
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*:



Cyclomatic complexity

Cyclomatic complexity is equal to 4 (4)

Formula

1 Cyclomatic complexity = no of predicates +1

For given in the program predicates are if while total 2 if and 1 while condition

so the answer is 4

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
```