Final Term

Software Verification and validation

Marks: **50**

ID: 14646 **NAME:** Abdul Musawer

Q1. MCQS (10)

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

b. It depends on the risks for the system being tested **(Correct Answer)**

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

d. All of the above **(Correct Answer)**

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

c. Validation **(Correct Answer)**

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

d. System Testing **(Correct Answer)**

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

d. All of the above. **(Correct Answer)**

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

a. Verification **(Correct Answer)**

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

b. End users **(Correct Answer)**

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

c. Measuring the LOC. **(Correct Answer)**

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

a. Defect **(Correct Answer)**

10. Testing of individual components by the developers comes under which type of testing?
- Published on 29 Jul 15

c. Unit testing **(Correct Answer)**

Q2. Explain Black Box testing and White Box testing in detail. (10)

ANSWER:

BLACK BOX TESTING:

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Behavioural Testing Techniques:

There are different techniques involved in Black Box testing.

- Equivalence Class
- Boundary Value Analysis
- Domain Tests
- Orthogonal Arrays
- Decision Tables
- State Models
- Exploratory Testing
- All-pairs testing

Pros of Black Box Testing:

- Well suited and efficient for large code segments.
- Code access is not required.
- Clearly separates the user's perspective from the developer's perspective through visibly defined roles.
- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.

Cons of Black Box Testing:

- Limited coverage, since only a selected number of test scenarios is actually performed.
- Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
- Blind coverage, since the tester cannot target specific code segments or error prone areas.
- The test cases are difficult to design.

WHITE BOX TESTING:

White box testing is a testing technique that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

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.

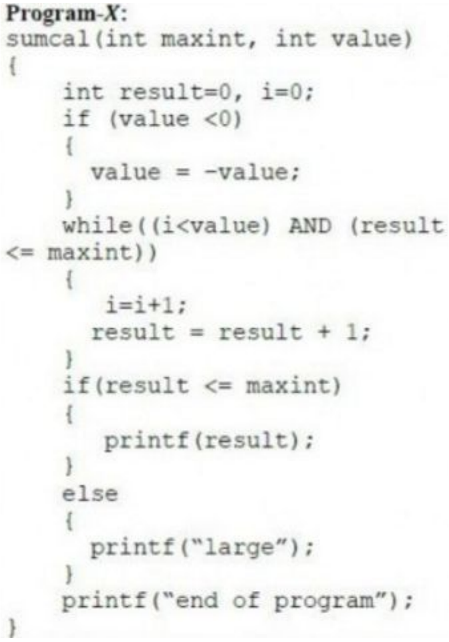
Pros of White Box Testing:

- Forces test developers to reason carefully about implementation.
- Reveals errors in "hidden" code.
- Spots the Dead Code or other issues with respect to best programming practices.

Cons of White Box Testing:

- Expensive as one has to spend both time and money to perform white box testing.
- Every possibility that few lines of code are missed accidentally.
- In-depth knowledge about the programming language is necessary to perform white box testing.

Q3. Find the cyclomatic Complexity and draw the Graph of this code. (15)

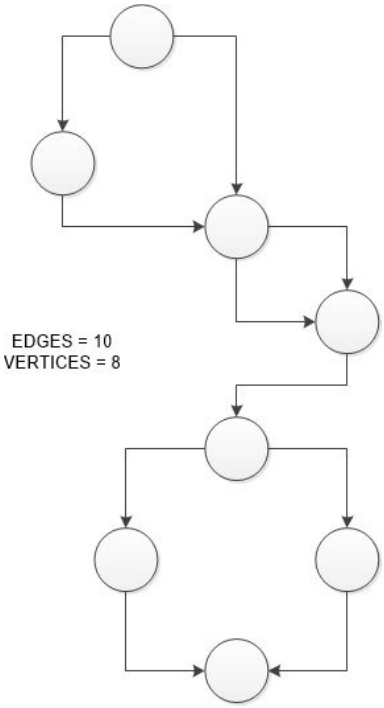


Program-*X*:

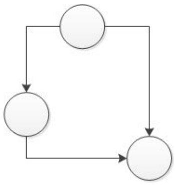
```
sumcal(int maxint, int value)
{
    int result=0, i=0;
    if (value <0)
    {
      value = -value;
    }
    while((i<value) AND (result
<= maxint))
    {
       i=i+1;
      result = result + 1;
    }
    if(result <= maxint)
    {
       printf(result);
    }
    else
    {
      printf(“large”);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

ANSWER:

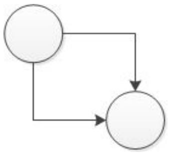
<u>Cfg of the code:</u>



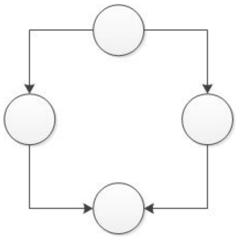
If Condition:



While Condition:



If-else Condition:



CYCLOMATIC COMPLEXITY:

- Cyclomatic complexity of Program X is the number of conditions +1.
- There are two ‘if’ conditions and 1 'while’ condition.
- Therefore Program ‘X’ = 4.

Cyclomatic complexity will be equal to four(4).

formula: M = E − N + 2P,

where, E is the number of edges of the graph.

N is the number of nodes of the graph.

P is the number of connected components.

Program X, E = 11, N = 9, P = 1, So M = 11-9+2*1 = 4

OR

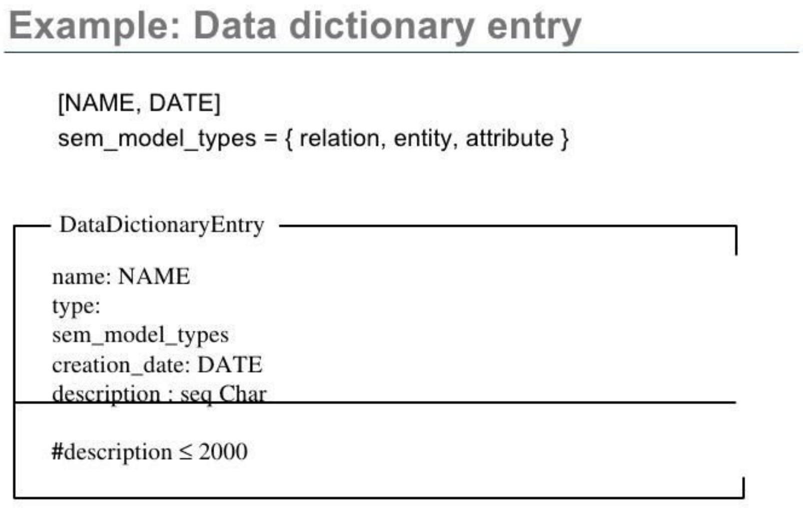
Logically as my per understanding:

Cyclomatic complexity = No of predicates +1

for the Given program, predicates are if, while. total 2 if and 1 while condition.

so the answer will be 4.

Q4. What is Z specification and why its is used for, also give some example this code written in Z specification. (15)



Example: Data dictionary entry

[NAME, DATE]
sem_model_types = { relation, entity, attribute }

```
DataDictionaryEntry
name: NAME
type:
sem_model_types
creation_date: DATE
description : seq Char
-----
#description ≤ 2000
```

ANSWER:

<u>Ans: Z SPECIFICATION:</u>

It is a model based sequential approach used for describing and modelling computing systems.

Z is not a programming language and does not compile into executable code, and also does not run interpreter.

<u>USES:</u>

- The purpose of Z is to describe the behaviour of a system such as software application in the language of modern mathematics.
- It is targeted at the clear specification of computer programs and computer-based systems in general.

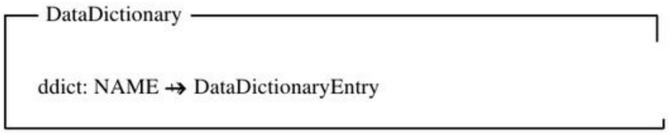
WHY “Z”:

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

Although UML and data-flow diagrams are useful in modeling system behaviour, there are limitations in the amount of detail that they can describe given the expressive power of modern mathematics. It is natural to adapt mathematics to the description of computer systems and the use of mathematics to describe computer systems also lends itself to precise formalism this allows a clear unambiguous specification of the requirements of software useful in large software development teams the expressive power of Z also allow one to model not just computer system but system of almost any kind in particular the accuracy and expressive power of Z make it useful for the description of safety critical systems such as banking systems and medical equipment.

EXPLANATION OF EXAMPLE:

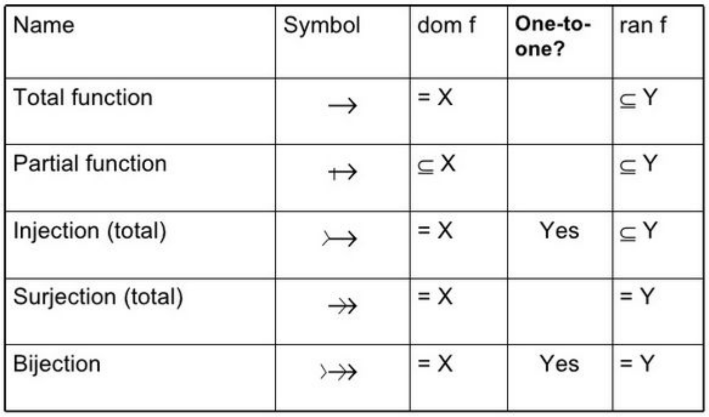
- A Data dictionary may be thought of as a mapping from a name (the key) to a value (the description in the dictionary).
- Operations are:
  - **Add.** Makes a new entry in the dictionary or replaces an existing entry.
  - **Lookup.** Given a name, returns the description.
  - **Delete.** Deletes an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



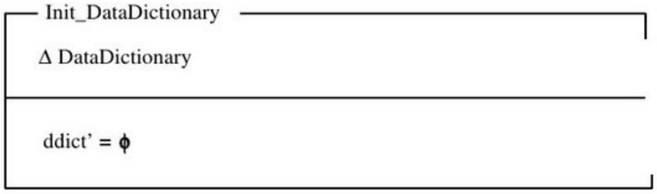
DataDictionary

ddict: NAME ⇸ DataDictionaryEntry

Names
Entities



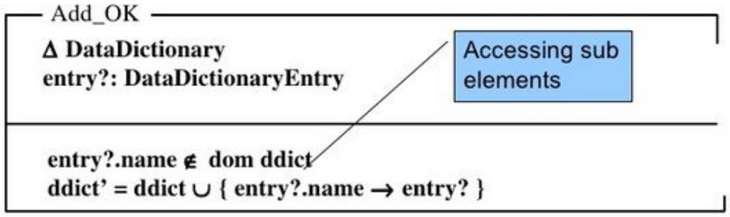
Name	Symbol	dom f	**One-to-one?**	ran f
Total function	→	= X		⊆ Y
Partial function	⇸	⊆ X		⊆ Y
Injection (total)	↣	= X	Yes	⊆ Y
Surjection (total)	↠	= X		= Y
Bijection	⤖	= X	Yes	= Y



Init_DataDictionary

Δ DataDictionary

ddict’ = ϕ

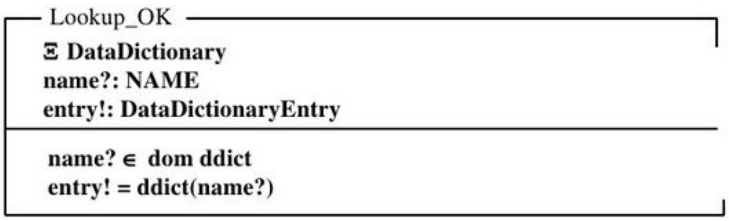


Add_OK

Δ DataDictionary
entry?: DataDictionaryEntry

entry?.name ∉ dom ddict
ddict’ = ddict ∪ { entry?.name → entry? }

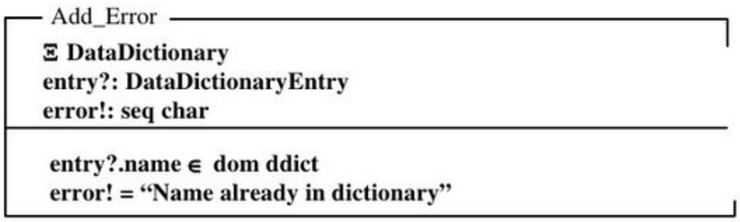
Accessing sub elements



Lookup_OK

Ξ DataDictionary
name?: NAME
entry!: DataDictionaryEntry

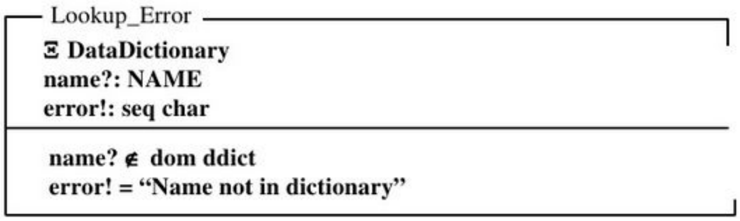
name? ∈ dom ddict
entry! = ddict(name?)



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