Name Syed Muhammad Salman Khan

ID:13662

Subject:Software verification and validation

Submitted to:Sir Zain Shaukat

Final Examination

Final Term

Software Verification and validation

Marks: **50**

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.

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

Program x:

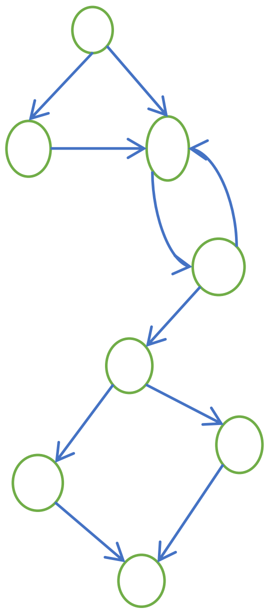
Cyclomatic complexity of program X is the number of condition +1.

(Cyclomatic complexity = condition + 1)

There are two (2) "if" conditions and 1 "while" condition.

Therefore program **'X' = 4**

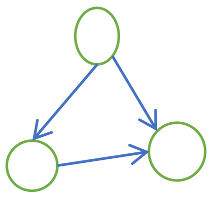
Control flow diagram program X:



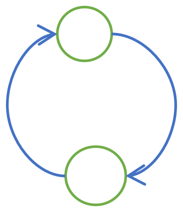
Edges=10

Vertices=8

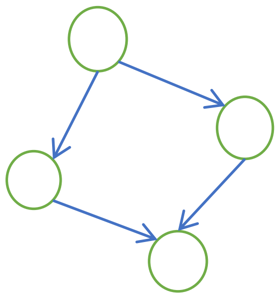
IF condition:



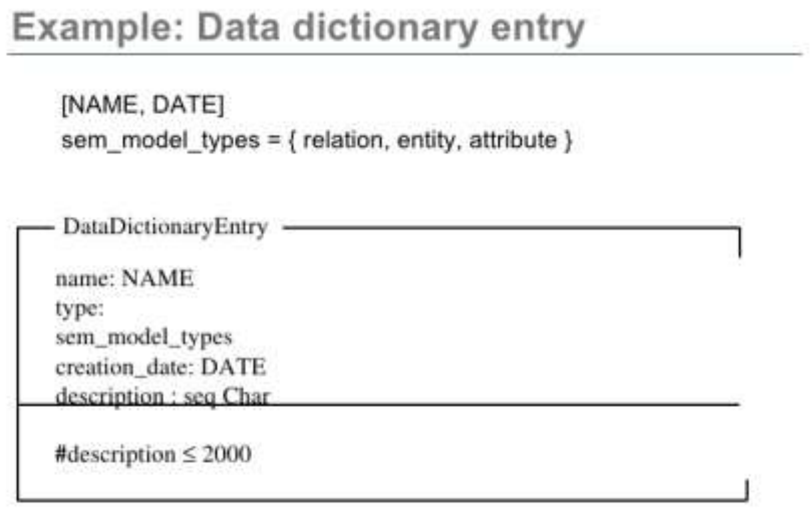
While condition:



IF Else condition:



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

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 entry

DataDictionaryEntry

entry: NAME

desc: seq char

type: Sem_model_types

creation_date: DATE

//Data dictionary as a function

DataDictionary

DataDictionaryEntry

ddict: NAME → {DataDictionaryEntry}

//Data dictionary - initial state

Init-DataDictionary

DataDictionary'

ddict'= ∅

//Add and lookup operations

Add_OK

Δ DataDictionary

name?: NAME

entry?: DataDictionaryEntry

name? ∉dom ddict

ddict'= ddict ∪{name? → entry?}

Lookup_OK

Ξ DataDictionary

name?: NAME

entry!: DataDictionaryEntry

name? ∈dom ddict

entry! = ddict (name?)

Add_Error

Ξ DataDictionary

name?: NAME

error!: seq char

name? ∈dom ddict

error! = "Name already in dictionary"

Replace_OK

Δ DataDictionary

name?: NAME

entry?: DataDictionaryEntry

name? ∈dom ddict

ddict' ⊕{name? → entry?}

//Delete entry

Delete_OK

Δ DataDictionary

name?: NAME

name? ∈dom ddict

ddict'= {name?} ddict

//The Extract operation

Extract

DataDictionary

rep!: seq {DataDictionaryEntry}

in_type?: Sem_model_types

∀n : dom ddict • ddict(n). type = in_type? ⇒ddict (n) ∈rng rep!

∀i : 1 ≤ i ≤ #rep! • rep! (i).type = in_type?

∀i : 1 ≤ i ≤ #rep! • rep! (i) ∈rng ddict

∀i , j: dom rep! • (i < j) ⇒rep. name(i) < NAME rep.name (j)

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

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 Black Box 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

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, **Blackbox testing**, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing.

The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "**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.

How do we perform White Box Testing

To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:

STEP 1) UNDERSTAND THE SOURCE CODE

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

Step 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer.

White Box Testing Techniques

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It identifies areas of a program that are not exercised by a set of test cases. Once gaps are identified, you create test cases to verify untested parts of the code, thereby increasing the quality of the software product

There are automated tools available to perform Code coverage analysis. Below are a few coverage analysis techniques a box tester can use:

Statement Coverage:- This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.

Branch Coverage - This technique checks every possible path (if-else and other conditional loops) of a software application.

Apart from above, there are numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc. Each technique has its own merits and attempts to test (cover) all parts of software code. Using Statement and Branch coverage you generally attain 80-90% code coverage which is sufficient.

Following are important WhiteBox Testing Techniques:

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Multiple Condition Coverage
- Finite State Machine Coverage
- Path Coverage

- Control flow testing
- Data flow testing

Types of White Box Testing

White box testing encompasses several testing types used to evaluate the usability of an application, block of code or specific software package. There are listed below --

> **Unit Testing:** It is often the first type of testing done on an application. Unit Testing is performed on each unit or block of code as it is developed. Unit Testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a single function or an object and test it to make sure it works before continuing Unit Testing helps identify a majority of bugs, early in the software development life cycle. Bugs identified in this stage are cheaper and easy to fix.
>
> **Testing for Memory Leaks:** Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.Apart from above, a few testing types are part of both black box and white box testing. They are listed as below

- White Box Penetration Testing: In this testing, the tester/developer has full information of the application's source code, detailed network information, IP addresses involved and all server information the application runs on. The aim is to attack the code from several angles to expose security threats
- White Box Mutation Testing: Mutation testing is often used to discover the best coding techniques to use for expanding a software solution.

White Box Testing Tools

Below is a list of top white box testing tools.

- Parasoft Jtest
- EclEmma
- NUnit
- PyUnit
- HTMLUnit
- CppUnit