Final Assignment

Software Verification & Validation

Submitted By: Muhammad Zain Ul Abideen
ID# 13740
BSSE
Submitted To: Respected Sir Zain Shukat *(Lecturure)*

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?

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.

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

7. Who performs the Acceptance Testing?

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

8. Which of the following is not a part of Performance Testing?

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?

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

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

a. Integration testing

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

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

Ans: **Black Box Testing:**

Definition:

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.

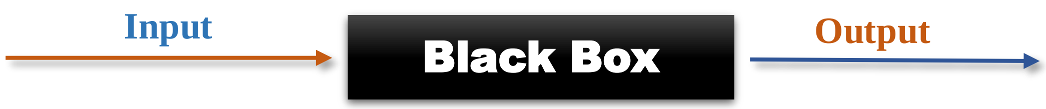


Fig: Black Box

Explanation:

In Black-Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program. In the above figure the black box can be any software system you want to test. It can be a game, a database, or any android application or can be a web application or website. In black box testing we can test applications or websites by only focusing on the inputs and outputs like can the software system giving me my desired result by entering the input. In black box testing don’t have to focus on the internal code implementation.

Steps To Do Black-Box Testing:

- Initially, we have to examine the requirements and specifications of the system.
- The 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.
- The tester determines the 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:

The main three types of black box testing are as follow:

- Functional testing
- Non-functional testing
- Regression testing
    - **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 for Black-Box Testing:

The tools used for Black-Box testing depends of what type of testing we are doing:

- For Functional/ Regression Tests we can use - QTP, Selenium tools.
- For Non-Functional Tests, we can use - LoadRunner, Jmeter tools.

White-Box Testing:

Definition:

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.

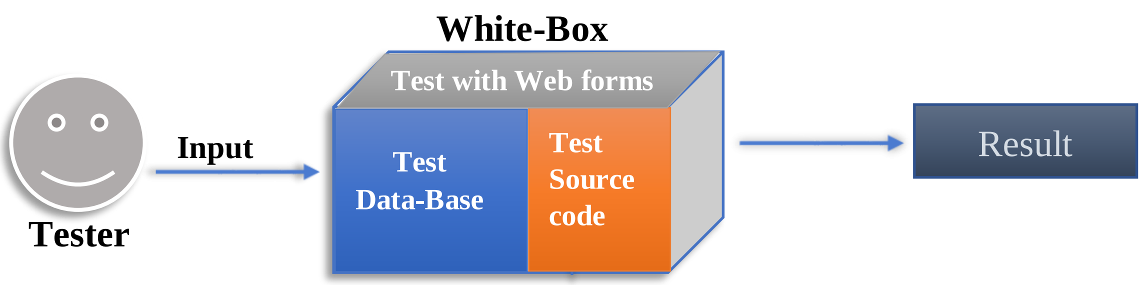


Fig: White-Box

Explanation:

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" 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. The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

What we verify in White-Box Testing:

The following has to be verified by White-Box testing:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis

How do you perform White Box Testing:

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. Other methods include Manual Testing, trial, and error testing and the use of testing tools as we will explain further on in this article.

Techniques used in White-Box testing:

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.

Following are important White-Box 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:

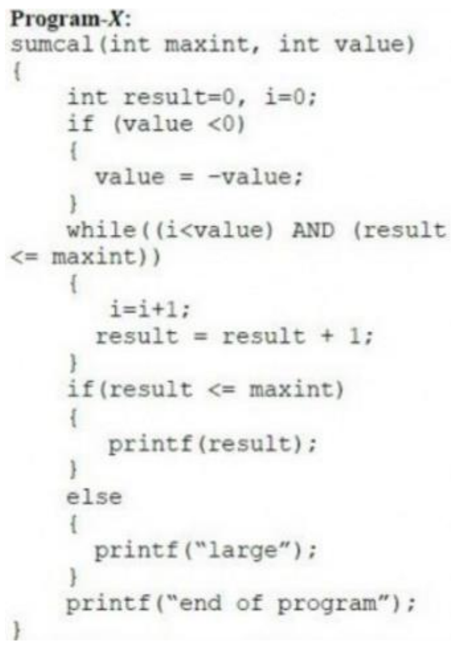
- Unit Testing
- Testing for Memory Leaks
- White-Box Penetration Testing
- White-Box Mutation Testing

White Box Testing Tools:

Below is a list of top white box testing tools.

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

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 of the above code is as follow:**

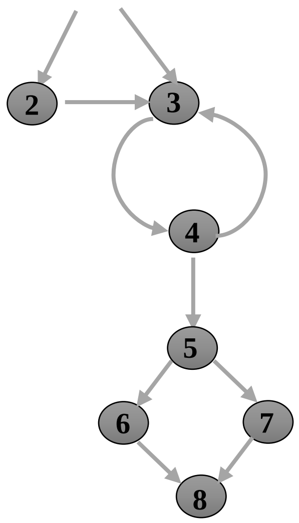
The cyclomatic complexity is the number of loops/conditions +1

So in the above code there are total 3 conditions in which 2 “if conditions” and “ 1 while condition”.

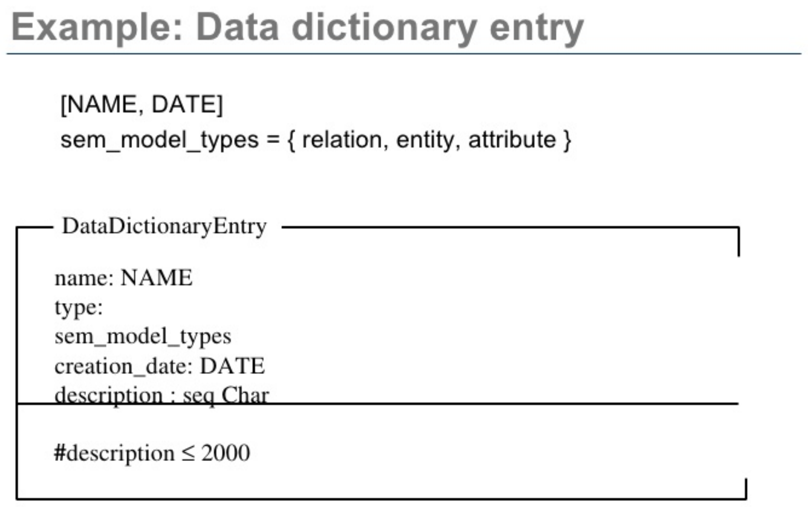
There for the V(G) = P+1 = 3+1 = 4

So the cyclomatic complexity for the above code is 4.

Graph: The control flow graph for the above code is bellow:



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



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
└────────────────────────────────────────────────
```

Ans: **Z Specification:**

Z specification is a formal specification language. that was proposed by Jean-Raymond Abrail, Steve Schuman and Betrand Meyer in 1977 and it was later further developed at the programming research group at Oxford University. Which is used for describing and modeling computing systems. It is based on the standard mathematical notation used in axiomatic set theory, lambda, calculus and first order predicate logic. The Z notation, is a strongly typed, mathematical, specification language. It has robust commercially available tool support for checking Z texts for syntax and type errors in much the same way that a compiler checks code in an executable programming language. In Z, both static and dynamic aspects of a system can be described using schemas. The Z specification describes the data model, system state and operations of the system. It cannot be executed, interpreted or compiled into a running program.

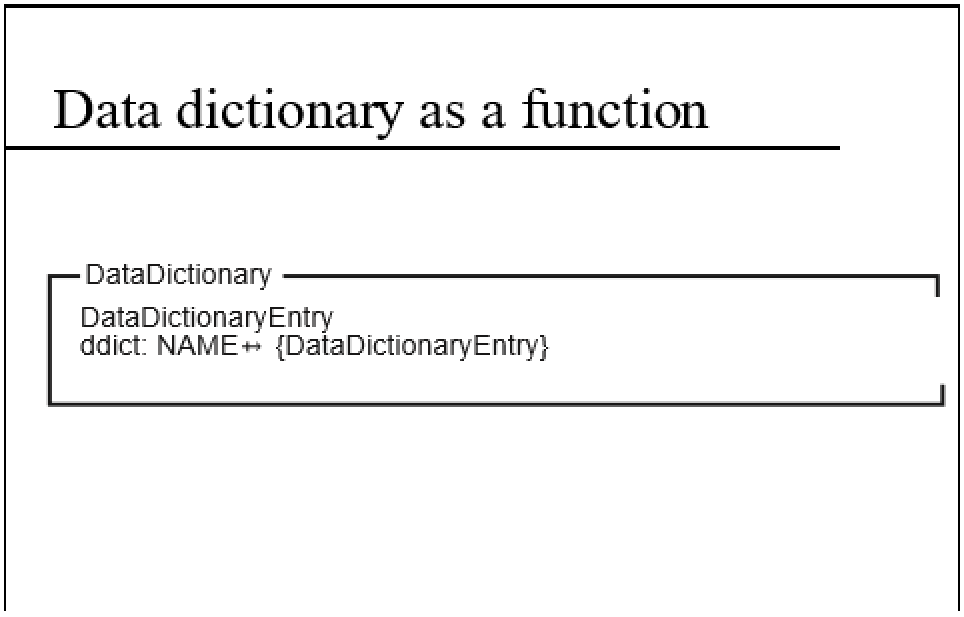
Z is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. All expressions in Z notation are typed, thereby avoiding some of the paradoxes of naive set theory. Z contains a standardized catalogue (called the mathematical toolkit) of 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 encodings for all standard Z symbols.

Why it uses for:

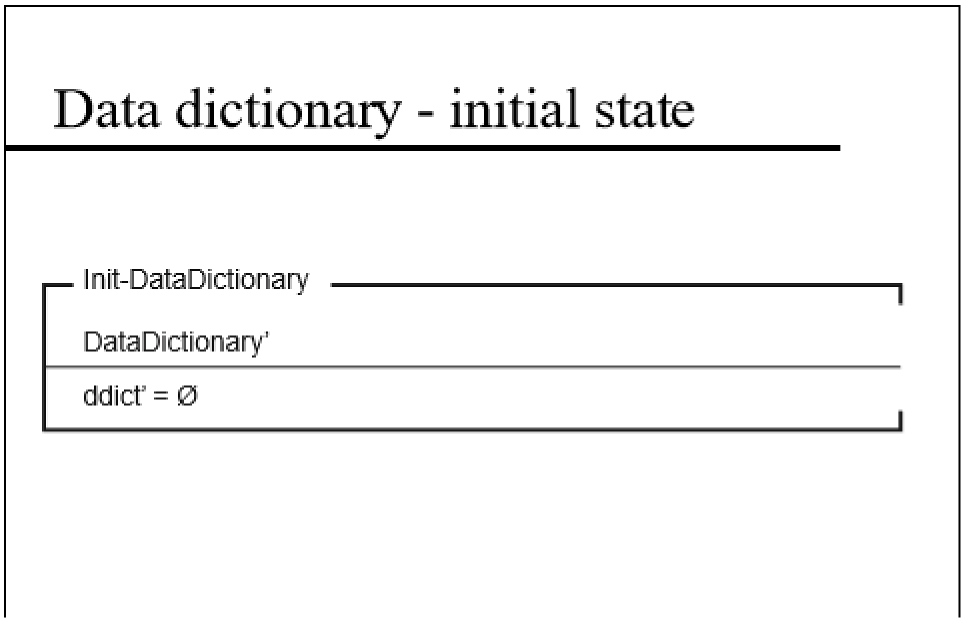
- A Z specification forces the software developer to completely analyze the problem domain. (e.g. identify the state space and pre and post conditions for all operations).
- A Z specification forces all major design decisions to be made prior to coding the implementation. Coding should not commence until you are certain about what you should be coding.
- A Z specification is a valuable tool for generating test data, and the conformance testing of completed systems.
- A Z specification allows formal exploration of properties of system.
- The flexibility to model a specification which can directly lead to the code.
- A large class of structural models can be described in Z without higher – order features, and can thus be analyzed efficiently.
- Independent Conditions can be easily added later.

Examples:



Data dictionary as a function

```
DataDictionary
DataDictionaryEntry
ddict: NAME ⇸ {DataDictionaryEntry}
```



Data dictionary - initial state

```
Init-DataDictionary
DataDictionary'
----------------------------
ddict' = Ø
```

Add and lookup operations

Add_OK

Δ DataDictionary
name?: NAME
entry?: DataDictionaryEntry

$name? \notin \mathrm{dom}\ ddict$
$ddict' = ddict \cup \{name? \mapsto entry?\}$

Lookup_OK

Ξ DataDictionary
name?: NAME
entry!: DataDictionaryEntry

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