



Muhammad Ali Khan

16550

Final Term

Software Verification and validation

Q1. MCQS

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

b. It depends on the risks for the system being tested

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

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.

c. Validation

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

d. System Testing

5. What do you verify in White Box Testing?

- Published on 03 Aug 15

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

7. Who performs the Acceptance Testing?

- Published on 03 Aug 15

b. End users

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

- Published on 30 Jul 15

c. Measuring the LOC.

9. Which of the following can be found using Static Testing Techniques?

- Published on 29 Jul 15

a. Defect

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

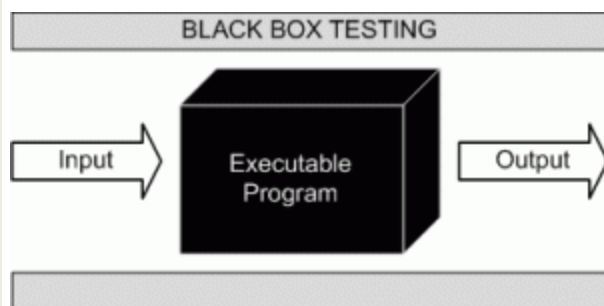
- Published on 29 Jul 15

c. Unit testing

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

Black Box Testing

BLACK BOX TESTING, also known as Behavioral Testing is a software testing method in which the internal structure/design/implementation of the item being tested is **not** known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Definition by ISTQB

- **Black box testing:** Testing, either functional or non-functional, without reference to the internal structure of the component or system.
- **Black box test design technique:** Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

Example

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

Levels Applicable To

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

- Integration Testing
- System Testing
- Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black-box testing method comes into use.

Techniques

Following are some techniques that can be used for designing black box tests.

- *Equivalence Partitioning*: It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- *Boundary Value Analysis*: It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
- *Cause-Effect Graphing*: It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

Advantages

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

Disadvantages

- Only a small number of possible inputs can be tested and many program paths will be left untested.

- Without clear a specification, which is the situation in many projects, test cases will be difficult to design.
- Tests can be redundant if the software designer/developer has already run a test case.
- Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

White Box Testing

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.

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

Ans: Cyclomatic Complexity of the above code is as follow:

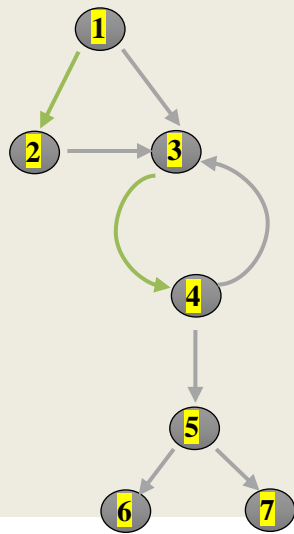
The cyclomatic complexity is the number of loops/conditions +1

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
```

Definition Z specification:

Z is a model oriented formal specification language based on Zermelo-Fränkel axiomatic set theory and first order predicate logic. It is a mathematical specification language, with the help of which natural language requirements can be converted into mathematical form.

Introduction:

With the ever-increasing complexity of computer systems, reliable and effective, design and development of high-quality systems that satisfy their requirements is extremely important. In

the mission and safety critical system failure can cause cost overrun, loss of lives or even severe economic consequences can arise. So, in such situations, it is necessary that errors are uncovered before software is put into operation. These challenges call for acceptance of proper engineering methods and tools and have motivated the use of formal methods in software engineering. There are varieties of formal specification languages available to fulfill this goal and one way to achieve this goal is by using Z formal specification language. Z is model oriented formal method based on set theory and first order predicate calculus.

Description of Z Formal Specification Language: The Z language is a model oriented, formal specification language that was proposed by Jean-Raymond Abrial, Steve Schuman and Bertrand Meyer in 1977 and it was later further developed at the programming research group at Oxford University. It is based on Zermelo Fränkel axiomatic set theory 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. It cannot be executed, interpreted or compiled into a running program. It allows specification to be decomposed into small pieces called schemas. The schema is the main feature that distinguishes Z from other formal notations. 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. Z specification is useful for those who find the requirements, those who implement programs to meet those requirements, those who test the consequences, and those who write instruction manuals for the system. Z also helps in refinement towards an implementation by mathematically relating the abstract and concrete states. Z is being used by a wide variety of companies for many different applications. In the Z notation there are two languages: **Mathematical Language** The mathematical language is used to describe various aspects of a design: objects and the relationships between them by using propositional logic, predicate logic, sets, relation and functions. **Schema Language** The schema language is used to structure and compose descriptions: collecting pieces of information, encapsulating them, and naming them for reuse.

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.

Data dictionary as a function

DataDictionary
DataDictionaryEntry
ddict: NAME \leftrightarrow {DataDictionaryEntry}

Data dictionary - initial state

Init-DataDictionary

DataDictionary'

ddict' = \emptyset

Add and lookup operations

Add_OK

Δ DataDictionary
name?: NAME
entry?: DataDictionaryEntry

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

Lookup_OK

\exists DataDictionary
name?: NAME
entry!: DataDictionaryEntry

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

Add and lookup operations

Add_Error

\exists DataDictionary
name?: NAME
error!: seq char

name? \in dom ddict
error! = "Name already in dictionary"

Lookup_Error

\exists DataDictionary
name?: NAME
error!: seq char

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

Function over-riding operator

- ⊗ ReplaceEntry uses the function overriding operator (written \oplus). This adds a new entry or replaces an existing entry.
 - $\text{phone} = \{\text{Ian} \rightarrow 3390, \text{Ray} \rightarrow 3392, \text{Steve} \rightarrow 3427\}$
 - The domain of phone is $\{\text{Ian}, \text{Ray}, \text{Steve}\}$ and the range is $\{3390, 3392, 3427\}$.
 - $\text{newphone} = \{\text{Steve} \rightarrow 3386, \text{Ron} \rightarrow 3427\}$
 - $\text{phone} \oplus \text{newphone} = \{\text{Ian} \rightarrow 3390, \text{Ray} \rightarrow 3392, \text{Steve} \rightarrow 3386, \text{Ron} \rightarrow 3427\}$

Replace operation

Replace_OK

Δ DataDictionary
name?: NAME
entry?: DataDictionaryEntry

$\text{name?} \in \text{dom ddict}$
 $\text{ddict} \oplus \{\text{name?} \mapsto \text{entry?}\}$

Delete entry

Delete_OK _____

Δ DataDictionary
name?: NAME

name? \in dom ddict
ddict' = {name?} \triangleleft ddict

Data dictionary extract operation

- ⊗ The Extract operation extracts from the data dictionary all those entries whose type is the same as the type input to the operation
- ⊗ The extracted list is presented in alphabetical order
- ⊗ A sequence is used to specify the ordered output of Extract

The Extract operation

Extract

DataDictionary
rep!: seq {DataDictionaryEntry}
in_type?: Sem_model_types

$\forall n : \text{dom } \text{ddict} \bullet \text{ddict}(n). \text{type} = \text{in_type?} \Rightarrow \text{ddict}(n) \in \text{rng } \text{rep!}$
 $\forall i : 1 \leq i \leq \#\text{rep!} \bullet \text{rep!}(i). \text{type} = \text{in_type?}$
 $\forall i : 1 \leq i \leq \#\text{rep!} \bullet \text{rep!}(i) \in \text{rng } \text{ddict}$
 $\forall i, j : \text{dom } \text{rep!} \bullet (i < j) \Rightarrow \text{rep.name}(i) <_{\text{NAME}} \text{rep.name}(j)$

Data dictionary specification

The_Data_Dictionary

DataDictionary
Init-DataDictionary
Add
Lookup
Delete
Replace
Extract
