Black Box Testing

Introduction

- Testing Technique specifies a strategy that is used in testing select input test cases and analyze test results
- When the features and operational behavior of the product needs to be tested, **Functional Testing or Black Box Testing** can be approached.
- Advantage is the internal working of the system is ignored.
- Total Cost of Ownership (TCO) and Return on Investment (ROI) are two criteria that favor the Black Box testing technique.
- if the application has to be checked for stability or needs to be ascertained for thoroughness, it would have to undergo white box testing.

Structural and Functional Testing

- Structural and functional testing is also known as white box and black box testing respectively.

Black Box Testing

- Black box testing, also termed as behavioral testing, checks if the software works as per the desired requirements or specifications. It is called black box testing because the tester performs the tests without
- knowing the internal logic of how exactly the software works. He/she focuses on the outputs generated in response to selected inputs and execution conditions.

Black Box Testing

- In neural networking or Artificial Intelligence simulation, a black box is used to describe the constantly changing section of the program environment which a programmer cannot test easily.

Black Box Testing

- Developing efficient test cases is very essential during black box testing.
- the tester has no knowledge of the internal working of the software, they need to rely completely on the analysis of the transformation of the inputs to the outputs based on which they find bugs in the software.
- It enables the tester to know whether or not the software does what it is supposed to do.

Example

- Testing search engine is a good example for black box testing. You are not aware of the processes that work behind the search engine to provide the desired information.
- While testing a search engine you provide input in the form of words or characters, and check for output parameters such as relevance of the search result, time taken to perform the search or the order of listing the search result.

Advantages of Black Box Testing

- Black box testing has many advantages, which include the following:

1. Testers do not have to understand the internal working of the software, and it is easy to create test cases with the perspective of an end user.

2. The testers mainly deal with the Graphic User Interfaces (GUI) for output, and they do not spend time analyzing the internal interfaces. Therefore, test cases can be developed quickly and easily.

3. As soon as the specification of the product is complete, the test cases can be designed.

4. Black box testing helps to expose any ambiguities or inconsistencies in the specifications, and tests are carried out from a user's perspective.

Black Box Testing example

- Testing the functions of an ATM is a good example of black box testing. The tester acts as a customer who is using the ATM and checks the functions of the machine. He/She does not know the internal working of the logic. The test cases are developed to check the functions through the GUI of the ATM such as change in display of the GUI when card is detected, masking the password or navigating from main menu to a specific function.

Disadvantages of Black Box Testing

1. A tester can test only a small number of possible inputs and it is highly impossible to test every possible input stream.

2. It is very difficult to design test cases if specifications are not clear and concise.

3. Situations, such as unnecessary repetition of test inputs, can occur if the tester is not informed of test cases that the programmer has already tested.

4. This type of testing cannot be focused on specific segments of function that may be very complex, therefore bugs can go undetected.

Black Box Testing

- When there is a complex system to be tested like the online Indian railways booking system, it is difficult to identify tricky inputs and write test cases to cover all possible scenarios.

Black Box Testing

- Performing a black box test, the tester attempts to find the following errors based on the behavior of the software:

- Missing or incorrect functionality.
- 1. Errors in interface.
- 2. Data structure errors.
- 3. Performance errors.
- 4. Initialization and terminal errors.

White Box Testing

- White box testing is also called as glass box testing.
- The tester focuses on the structure of the software code. The tester develops test cases to check the logical working of the software code.
- Black box testing helps to answer the validation question "are we building the right software?",
- white box testing helps to answer the verification question "are we building the software right?"

White Box Testing

- Each software module is tested independently
- The tester has to develop test cases not only to test the individual module of the software, but also to test how exactly the modules interact with each other when software is executed the tests are carried out at the source code level
- tester checks all the parameters of the code such as efficiency of the code written, branching statements, internal logic of the module, interfaces between external hardware and internal module, memory organization, code clarity, and so on.
- the test cases must be carefully designed in order to cover the internal working of the application

White Box Testing

- The tester who writes the test cases to perform white box testing has to be aware of the language and logic used to develop the test software. He/she needs to know programming concepts as well.

Advantages of White Box Testing

1. As the tester has the knowledge of internal coding, it is very easy to develop test cases to test the software effectively.

2. Testing is carried out at the code level; hence it helps in optimizing the code.

3. Unnecessary or extra lines of code which can generate hidden bugs can be removed.

Example

- A test case to check for bugs in the loops that are used in a software application, should include the following situations:

1. If the loop iterates zero times.
2. If the loop iterates once.
3. If the loop iterates twice.
4. If the loop iterates several times.
5. If the loop iterates n - 1 times.
6. If the loop iterates n times.
7. If the loop iterates n + 1 times.
8. If the loop iterates infinite times.

Disadvantage of White Box Testing

1. It is highly impossible to check every code to find out the hidden errors or bugs, which may cause problems that lead to failure of the software.

2. Skilled testers are required to carry out this test, which increases the cost.

3. The time required to carry out this test for complex software is very high.

Black or White ?

- Selecting the right testing method for testing the software is very important, since both black and white box testing methodologies have their merits.
- Following are a few questions which can help you in taking the right approach:

1. Who will be the users of the application?
2. Prior to release, which parts of the application must be tested and why?
3. When do we make significant changes to the User Interfaces and will this affect the actual code of the application?
4. Where is the application likely to be installed?
5. How will end users be using the application?
6. Which platforms does the application need to support after installation?

Static Black Box Testing and Dynamic Black Box Testing Techniques

Static Black Box Testing Techniques

- Used to test the software without compilation.
- When the test is performed to check the specifications, it is called static black box testing.
- the specification is a document that provides information of the software functionalities. This document is created during the initial stages of the Software Development Life Cycle (SDLC) based on the input from the customer and designer.
  - *The focus of static black box testing is to check for completeness or appropriateness of the product or application developed*

Static Black Box Testing

- specification document not only lists the functionalities of the software, but it also provides vital information of the software to the user or customer.
- This test is more often a research, since the tester should make sure that no vital information is missed or incorrect information is provided in the document.
- to find confusing and misinterpreted information in the software application. Any such information is considered as a bug.
  - *The tester must understand the customer's or end user's expectations and make sure that the document meets these requirements.*

Example

- The mobile user manual is an example of specification document.
- The user needs this manual to operate the mobile applications and to know all the features that are available in the mobile.
- As a tester when you test the mobile as per manual, you need to test it with an end user perspective and make sure that the manual meets all the quality requirements.
- The specification should be correct, clear, and complete with all the information of features available in the mobile model.

Static black box testing - High Level Specification Test Technique

- Testing the specification of a document is considered static, since we do not execute a program.
- requires a methodical approach to view the specification from a high level.
- reviewed for the fundamental explanation.
- should be checked if it is complete and if there has been any omissions
- research oriented and the research helps to understand how the specification is organized and the reason behind the organization of the specification.
- view the specification from the perspective of the customer who would be using the software. It is important to understand and meet customer expectations.

Static black box testing

- Standards have to be mandatorily followed whereas following guidelines are subjective to the requirements of the product.
- Tests should make sure that the document strictly follows these standards and guidelines.
- Any violation of these will be treated as a bug and has to be corrected.
- This includes the page size, color patterns, style, font size, and so on.

Static black box testing

- As a tester, you have to research on what should appear on the software. Standards and guidelines are formed based on certain rules followed by software developed.
- ***Conventions Followed by Corporates:*** The software should adhere to the terms and conventions used by the company.
- ***Industry Requirements:*** Certain Industry segments like medical, pharmaceutical, and financial industries follow their own conventions while developing software.
- ***Government Standards:*** Government agencies follow rules stipulated by them. For example, Military standards are unique.
- ***Graphical User Interface (GUI):*** Software that works under Microsoft Windows or Macintosh has separate published standards and guidelines that dictate the look and feel of the user.
- ***Security Standards:*** Any software developed has to meet certain security standards or levels. In some cases, they need to be certified that they meet the necessary criteria.

Example

- Most of the technical documents are based on Microsoft Manual of Style for Technical Publication (MSTP) which defines standards and guidelines for publishing technical documents such as user manuals, installation guide and operating manual.

Static Black Box Testing - Low level Specification Test Technique

Completeness: The specification should bring complete information of the product

Accuracy: The defined goals of the proposed specification should be addressed without any errors.

Precision, Unambiguity and Clarity: The description should be easy to comprehend. The content has to be exact and not vague.

Relevance: Explanations are necessary and should be traceable to the requirements of the customer.

Feasibility: The feature must be implemented with the available personnel, tools and resources, with no additional cost.

Code-free Explanation: The specification should define the product and should avoid unnecessary explanations of the technology, design and architecture.

Testable: Since Low-level specification looks out for clarity of the explanation, a tester has to keenly test

Specification Test Technique

- Low-level specification looks out for clarity of the explanation, a tester has to keenly test whenever he/she encounters the following words:

1. ***Always, None, All, Never:*** The above mentioned cases denote a certain case. A tester has to foresee where the conditions could be violated
2. ***Certainly, Clearly, Obviously, Evidently:*** These words are persuasive words which should be tested.
3. ***Etc., And So on:*** Specifications need to be absolute or should be explained with no confusion. It is best that such words are avoided.
4. ***Some, Sometimes, Often, Usually, Mostly:*** Vague words like these need to be avoided.
5. ***Good, Fast, Cheap, Efficient, Small:*** Unquantifiable terms such as these need to be avoided.

Dynamic Black Box Testing

- dynamic black box testing is carried out with data.
- “dynamic” is used since the tester is able to observe the changes exhibited by the system. The test is carried out by providing pre-defined inputs and the outputs are recorded.
- outputs are compared with the correct output, and the variation that exists between the actual output and desired output are segregated as bugs.
- tests are carried out using test cases. Test cases have to be defined effectively in order to find the bugs.

Dynamic Black Box Testing

- A software application undergoes a dynamic black box test.
- The test case defines that the application has to produce an output D when the inputs A and B are given. If the application gives an output C, then the application fails the test case and this is a bug.

Techniques used to perform dynamic black box testing

- test-to-pass and test-to-fail,
- equivalence partitioning,
- data testing
- state testing

Test to Pass and Test to Fail

- **test-to-pass** approach checks for the standard functions of the program.
- The test cases focus mainly to check the normal operation of the software.
- Test cases do not push the software to its limit or will not try to break the software.
- Try to find out the bugs by operating the software under normal conditions.

Test to Pass and Test to Fail

- **Test-to-fail** approach test cases push the software to its extreme
- main focus is to push the software to its limit and check for bugs when the software is operated under extreme conditions.
- tester tries to provide extreme and erroneous values as inputs to check how the software can be broken.

Example

- A calculator application is developed to calculate the average of more than 1000 integer values. The calculator first has to be set to "Find average" mode and all the values whose average has to be found will be entered. Once all the values have been entered the “equal” button can be clicked to display the average of all the entered numbers. As a tester when you perform test-to-pass, you calculate the average for few hundred values, but will not cross the thousand values mark. This is testing the application by providing the normal input.
- When you perform test-to-fail, you provide more than thousand values and check the output of the software. You even take it further by providing more than two thousand values and check the output produced by the application. Such inputs might overload the application. This helps the tester to check the
- maximum number of values for which the application can find the average.

Test to Pass and Test to Fail

- the tester should first begin the test with test-to- pass, and check whether the software works fine without any bugs. Once it clears test-to-pass, the tester can perform test-to-fail.

Equivalence Partitioning

- Equivalence partitioning, or equivalence classing is a process of classifying the test cases and grouping them into different categories or classes.
- It helps to reduce the number of test cases to a finite number of test cases without compromising on the quality of the test being carried out.
- objective of partitioning is to identify the test cases that perform same kind of testing and generate similar output.
- Since the test cases within one class are considered to be equivalent,
- picking up one test case from each class would be sufficient to detect the bugs in the software.
- This technique helps to reduce the volume of test cases considerably.

Equivalence Partitioning

- The class can be divided into two types, valid class and invalid class. All the classes which fall under valid type are values that satisfy the condition. Classes that do not satisfy the condition fall under the invalid class.

Equivalence Partitioning - Example

- A software module checks the age of the individual who has applied for a driving license. Any individual who is above 18 years and less than or equal to 60 years is eligible to apply for the driving license.

- ***Condition:*** Eligibility criteria for driving license
- ***Valid class:*** Age between 18 to 60 years
- ***Invalid class:*** 1. Age below 18 years
  2. Age above 60 years

Guidelines that are used to define Equivalence classes

1. If the test case input condition is a range, then you can define a minimum of one valid and two invalid equivalence classes.
2. If the test case input condition is a specific value, then you can define one valid and one invalid equivalence class.
3. If the test case input condition is a member of a set, then you can define one valid and one invalid equivalence class.
4. If the input condition is a Boolean value, then you can define one valid and one invalid equivalence class.

Data Testing

- software can be divided into two parts, *Data* which consists of inputs from keyboard, mouse or disk files, and *Program* which consists of code of the program which specifies the logic or syntax.
- Data testing refers to testing the data, which includes the data that the user inputs to the software and the output he/she receives.
- intermediate values that are generated, but are not displayed on the output device are also vital data when it comes to debugging the software.

Data Testing

- Testing can be carried out on the following four levels.
    1. Boundary conditions
    2. Sub-boundary conditions
    3. Nulls
    4. Bad data

Boundary Conditions

- Software works as it should under normal conditions, but it is important to test whether the software can operate properly under extreme conditions. Every application has a limit or maximum and minimum values it can process, and the tester has to identify these extreme limits to prepare test cases. These extreme values are provided as inputs, and tested for occurrence of bugs.

Boundary Conditions

- Testing the boundary is done by adding one, or a bit more, to the maximum value. It could follow the rule like the following:

1. First value -1 and/or Last value +1
2. Start conditions -1 and/or Finish condition +1
3. Less than Empty condition and/ or More than Full condition
4. Minimum value -1 and/ or Maximum value +1

Example

- A text field on a web page allows the user to enter up to 255 characters. The boundary values can be between 1 and 255 characters. The tester can enter only one character or 255 characters in the text field. 254 or 256 characters can be considered as boundary values to test the text field of the web page.
- CD writer software can write 256k bytes of data on to a CD. The boundary value to test this software can be 1k bytes or 256 k bytes. The tester can
- provide the software 1 k bytes or 256k bytes of data to write it on the CD.

Sub-Boundary Conditions

- Normal boundary conditions are based on the values that the user enters, but sub-boundary conditions are system-specific. These are the values that are related to the system hardware on which the software runs. Even though these values are not relevant to the end user, the tester has to test them, since it will result in unidentified bugs.

Sub-Boundary Conditions - Example

- The decimal range of a byte is 0 to 255, which can be represented in binary as 0000 to 1111. The tester can write test cases to make the software handle these values. Pass the sub-boundary value 0, 1, 255 or 256 and check how the software responds.

Sub-Boundary Conditions - Example

- **Nulls**
- Null means no value. Testing is carried out without providing any input. The software should be able to cope with this kind of situation where no input is provided to the software.

Example

- Login page has two empty fields, Login ID and password. Imagine a test case where the user tries to log in by providing the login ID and leaves the password field blank or vice versa. Here, only one field is filled by the user and the other field is left blank or null.

Sub-Boundary Conditions

- **Bad Data**
- The tester enters erratic or irrelevant data and checks the response of the software. The software might work correctly for all the correct inputs, but it is very important to check its response when incorrect or irrelevant data is provided as input.
- Example
- A login password field of a Webpage accepts only numerical values. It does not accept alphabets or combination of alphabets or numbers as password. If the tester enters these data, i.e., alphabets or combination of alphabets and numbers as password to test the Webpage, it is called as a bad data test.

State Testing

- State testing refers to testing the software state.
- State is a mode or condition of the software at any given time when the software is running.
- State diagrams are used to indicate the state of the software. This indicates the actual working and the logical flow of the software.
- State based testing is used for high level black box testing of programming languages like object oriented programming. Object oriented programming languages that have features like encapsulation can be easily depicted using state diagrams.

State Testing

- Every program has two important states, the start state which indicates the beginning of the program and the end state which indicates the end of the program.
- In between these two states, a number of intermediate states represent the logical flow of the software.
- During the executions, the program flows from one state to another. This change of state is called state transition.

State Testing - example

- The Windows paint program expresses itself in different states. When the application is opened, the cursor takes the pencil tool as the default state. When a different tool is selected, say an air-brush, the cursor changes its state to bring out the properties of the air-brush. The internal working of the software toggles every time the user selects a different tool. This change can be termed as state transition.

- *All software that is event driven makes use of state based techniques to analyze the program flow. For example, the GUI programs make use of state based techniques both at the design stage and at the testing stage.*

State Testing

- State diagram is a graph that depicts all the states and transitions taking place during the flow. The tester checks the states and transitions occurring at each level of the state diagram. The tester has to check each state and the transition that takes place, since every state is unique and each transition is a new operation that is performed on the current state to generate an output that is moving to the next state.

State Testing - example

- Let us consider a program where a transition has to happen from state A to B and from B to C. The tester checks if the transition happens from state A to C and from C to B. This incorrect transition will be considered as a bug in the software.

State Testing

- State based technique helps to obtain a model that depicts the behavior of a program very clearly. However, drawing a state diagram is a challenging task. Any software has many states and a transition associated with it. Depicting the same can be a complex process.

Random Testing and Mutation Testing

- **Random Testing**
- The tester provides random inputs to the test software and checks the output of the software against the expected output. Any mismatch in the actual output and expected output will be treated as a bug.
- popular testing techniques which is also called as Gorilla testing or Adhoc testing.

Random Testing Example

- In a mathematical expression, C = 2 * B;
- The variable B can take values from -500 to 500. The test can be carried out for five random values- 24, -348, 75, 499, and -1. The expected result would be 48, -
- 696, 150, 998, and -2 respectively. The expected result is compared with the actual result when this expression is executed.

Mutation Testing

- the tester makes minor modifications in the program's source code and performs the test. If the program clears the test even after it is modified, the program is considered to be defective. Any such modification is called mutation.
- the behavior of the code when some modification is made to the original code.
- bugs can be found in the software, after the modifications are made to the original software.

Mutation Testing

- Mutation testing is carried after the test software has cleared the preliminary test and is found bug free. The mutation can vary from a simple change of operator name or variable name to replacing the logical operator with its inverse. A complex mutation involves changing the order of execution of the code or removing some lines of codes.

Mutation Testing - Example

- The expression, a !=b;
- can be mutated as,
- a ==b;

Mutation Testing

- The test cases used are those which have been created during preliminary testing. If the test cases are well written, the mutated program should fail. However, if it clears the test case, either the test case is weak which has to be re-checked or the program has bugs.

- Questions