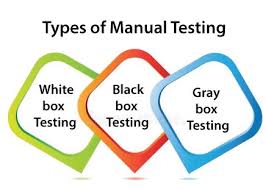
**COMMON MANUAL TESTING TECHNIQUES**

Manual testing is process of manually inspecting and validating the functionality of the software by human without using any automation code.

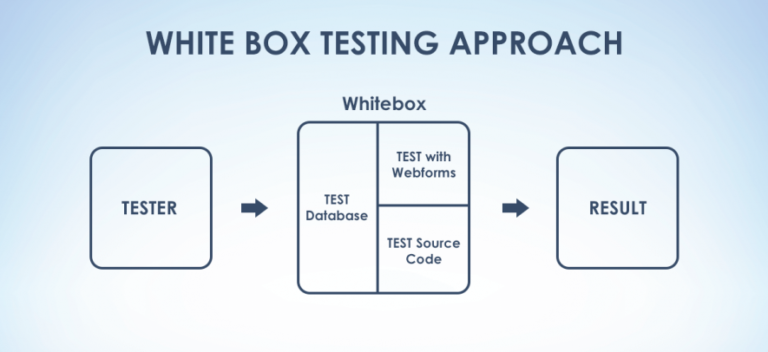
**Types of Manual Testing**

Manual testing can be divided into several types based on different approaches and the most common types includes White box testing, Blackbox testing and Greybox testing.



**White Box Testing**

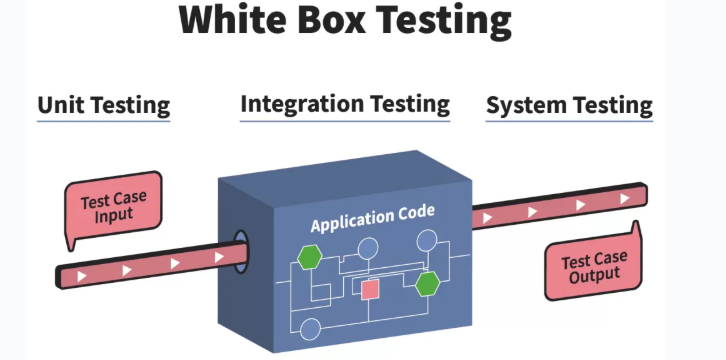
White Box testing is a procedure that includes verification of internal structure and logic of the software. Testers have full knowledge on software architecture and source code, allowing them to analyse internal structures and design test cases based on the system's design and code-paths. This approach aims to find hidden errors, verify control and data flow, and ensure internal operations are performed correctly.



White box testing is also called as clear box testing, transparent testing or open box testing, structural testing.

Techniques involved in white box testing includes statement coverage, path coverage, branch coverage, condition coverage. It also helps in identifying the internal bugs, detects inefficiency in code and check control and data flow of the program.

Various levels of white box testing are listed as follows



**Unit Testing**: It involves testing individual units or components of the software works correctly or not in isolation. It basically checks the application meets design requirements during development.

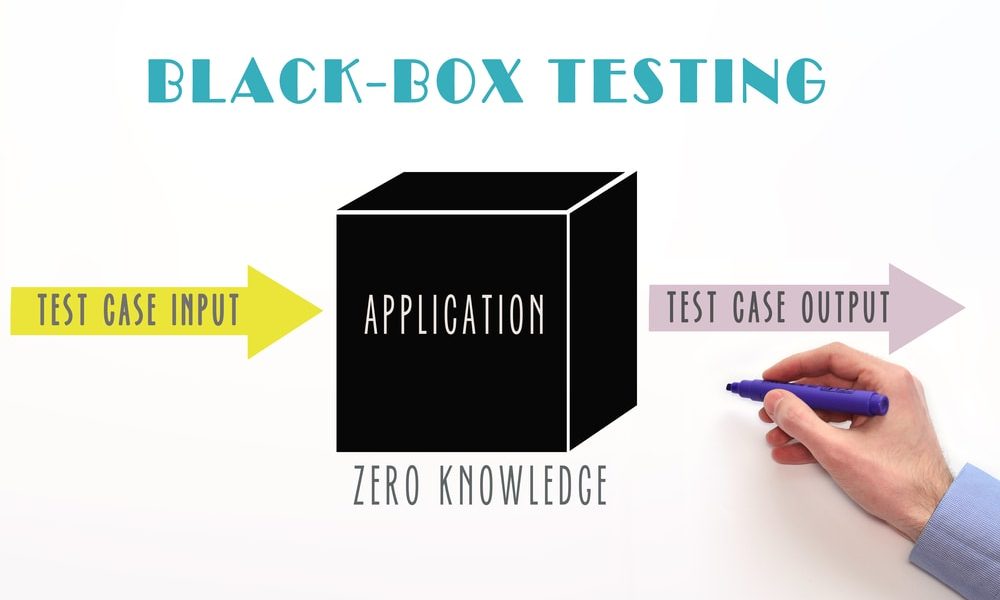
**Integration Testing**: This testing examines how different parts of the application works together. It basically verifies the interactions between integrated units or components of the software. This is performed after unit testing to make sure that not only each unit works well in isolation but also, they can work together effectively.

**Regression Testing**: It ensures that recent code changes have not adversely affected existing functionality. Test cases are re-executed to validate that previously developed and tested software performs correctly after modifications. It helps maintain software quality by preventing regression issues.

**System Testing**: It involves evaluating the entire integrated software system with full knowledge of its internal structure, code, and design.

**Black Box Testing**

The black box testing is the testing type in which the tester is unaware of internal working, code or architecture of the software and focuses mainly on verifying if its features and functionalities works as per the user requirements. It only deals with the inputs fed to the software and the outputs generated from them to ensure the system performs as required from an end-user perspective. Here the testers do not require any programming technical skills to work on black box testing.



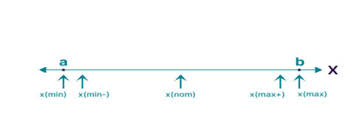
**Techniques used in Black box testing**

Following are the some of the black box testing techniques that are used to maximize the testing coverage with minimal number of test cases.

**Boundary Value Analysis**

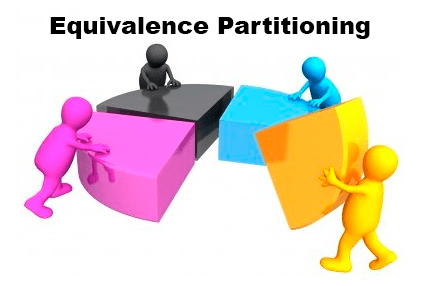
It mainly focuses on testing the boundaries or edges of input ranges. Its applied on the edges of input ranges rather than middle value. It helps identify potential vulnerabilities and ensures the system behaves correctly at its limits.

Boundary value analysis targets at just below, at and just above boundaries. If we have n – boundary values then its calculated as (n-1, n, n+1)



**Equivalence class partitioning**

It mainly groups the input data into similar values in the range and fetch any one value from the group or fetch any similar value between the range. It should not use the values near the boundaries.



Discover the inputs into groups based on the requirement and divide them into valid and invalid partitions. From each partition select one representative input value that acts as a substitute for all the other values in that partition. At last create and execute the test cases based on the selected values to verify that the actual result satisfies the expected result.

Example to validate Boundary value analysis and equivalence class partitioning

Calculate the age category for the following data with respect to days

* 1 Day to 6 months 🡪 Infant
* 6 Months to 2 Years 🡪 Toddler
* 2 Years to 4 Years 🡪 Child
* 4 Years to 12 Years 🡪 Kid
* 12 Years to 19 Years 🡪 Teen
* 19 years to 59 Years 🡪 Adult
* Above 59 Years 🡪 Old (age >=60)

**Answer:**

* 1month = 30 days, 1 Year = 365 days
* 6 months = 6 \* 30 = 180 days 🡪 Infant
* 2 years = 2\*365 = 730 days 🡪 Toddler
* 4 Years = 4\*365 = 1460 days 🡪 Child
* 12 Years = 12\* 365 = 4380 days 🡪 Kid
* 19 Years = 19 \* 365 = 6935 days 🡪 Teen
* 59 Years = 59 \* 365 = 21535 🡪 Adult
* 60 years = 60 \* 365 = 21900 days 🡪 Old

**Boundary Value Analysis**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Range** | **Below Lower Boundary** | **Lower Boundary** | **Middle /Above Lower Boundary** | **Below upper Boundary** | **Upper Boundary** | **Above Upper Boundary** | **Category** |
| 0 – 180 | -1 | 0 | 1 | 179 | 180 | 181 | Infant |
| 181 - 730 | 180 | 181 | 182 | 729 | 730 | 731 | Toddler |
| 731 - 1460 | 729 | 730 | 731 | 1459 | 1460 | 1461 | Child |
| 1461 - 4380 | 1460 | 1461 | 1462 | 4379 | 4380 | 4381 | Kid |
| 4381 - 6935 | 4380 | 4381 | 4382 | 6934 | 6935 | 6936 | Teen |
| 6936 - 21535 | 6935 | 6936 | 6937 | 21534 | 21535 | 21536 | Adult |

>21535 – Old 🡪 [ 21536, 22777, 28765]

**Equivalence Partitioning for age category calculation in days**

|  |  |  |  |
| --- | --- | --- | --- |
| **Range** | **Category** | **Valid Input** | **Invalid** |
| 0 – 180 | Infant | 85 | -1, 181 |
| 181 - 730 | Toddler | 250 | 180, 731 |
| 731 - 1460 | Child | 1000 | 730, 1461 |
| 1461 - 4380 | Kid | 3200 | 1460, 4381 |
| 4381 - 6935 | Teen | 5343 | 4380, 6936 |
| 6936 - 21535 | Adult | 11876 | 6935, 21536 |
| >21535 | Old | 30987 | 21535 |

**Decision Table Testing**

Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form. That is why it is also called as a Cause-Effect table where Cause and effects are captured for better test coverage.

Here the input and output are given as Boolean values. It helps to check all possible combinations of conditions for testing.

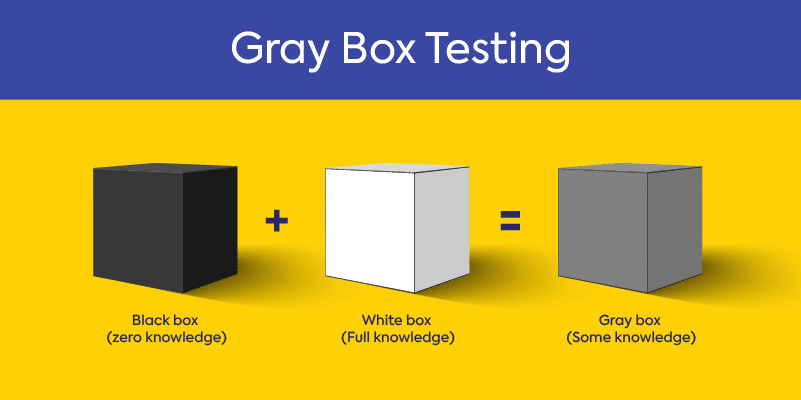
It provides clear representation of logic, thorough test coverage which ensures all scenarios are tested. Decision table organizes all possible input combinations and reduces complexity that improves comprehensive testing.

For Instance, in case of a login scenario it has two inputs username and password.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Username | Password | Output |
| Case 1 | Correct | Correct | True |
| Case 2 | Correct | Incorrect | False |
| Case 3 | Incorrect | Correct | False |
| Case 4 | Incorrect | Incorrect | False |

**Grey Box Testing**

Grey box testing is the blend of both white and black box testing. Here the tester partially knows the internal structure of the application being tested. It is also known as Translucent testing. It identifies all key paths to traverse through during the testing process.



Grey box testing techniques includes Matrix testing, Regression testing, Pattern testing and Orthogonal array testing.

**The Future of Manual Testing in the age of AI**

**Introduction**

Software testing is an essential step in building reliable applications. For years, manual testing has been the backbone of quality assurance. But with Artificial Intelligence (AI) and advanced automation tools rapidly gaining ground, many wonder if manual testing will still be relevant in the future. The reality is not about one replacing the other, but about how both can work together.

**Where Manual Testing Stands Today**

Manual testing is still valuable because it allows human judgment and creativity to come into play. Exploratory testing, usability checks, and spotting issues that affect real user experience are areas where people outperform machines. At the same time, manual testing can be slow, repetitive, and harder to scale when projects grow large.

**AI in Testing**

AI has changed the way testing can be done. With machine learning, test scripts can adapt on their own, analyse huge volumes of data, and even predict problem areas before they happen. This brings speed and efficiency to tasks like regression testing and performance checks. Still, AI comes with its own challenges like it requires good data, upfront investment, and it cannot fully understand human emotions or usability aspects.

**A Combined Approach**

The future of testing is not about choosing between manual and AI it is about combining them. AI can handle repetitive, time-consuming work, while manual testers focus on complex scenarios and the human side of software. For example, while AI can quickly check thousands of functions, only a human tester can judge whether the application feels intuitive for users.

****

**Looking Ahead**

Manual testers will continue to be important, but their role will evolve. Instead of spending most of their time on routine test cases, testers will focus on strategy, design, and validating AI-driven results. Skills like problem-solving, domain knowledge, and exploratory testing will be in higher demand. Testers will essentially become “AI-augmented professionals,” working alongside technology rather than competing with it.

**Conclusion**

Manual testing is not disappearing. It is transforming. AI brings speed and efficiency, but human insight ensures software is practical, user-friendly, and trustworthy. The future lies in a balance where AI handles the bulk work and humans provide the intelligence, empathy, and critical thinking. Together they create a stronger and more reliable testing ecosystem.

