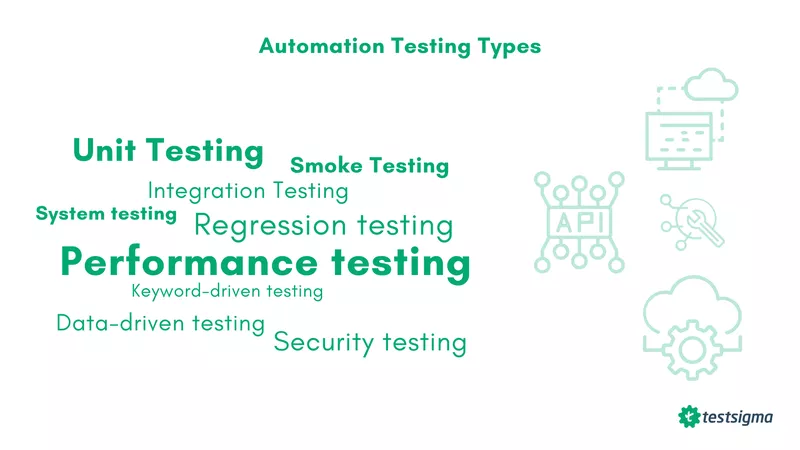
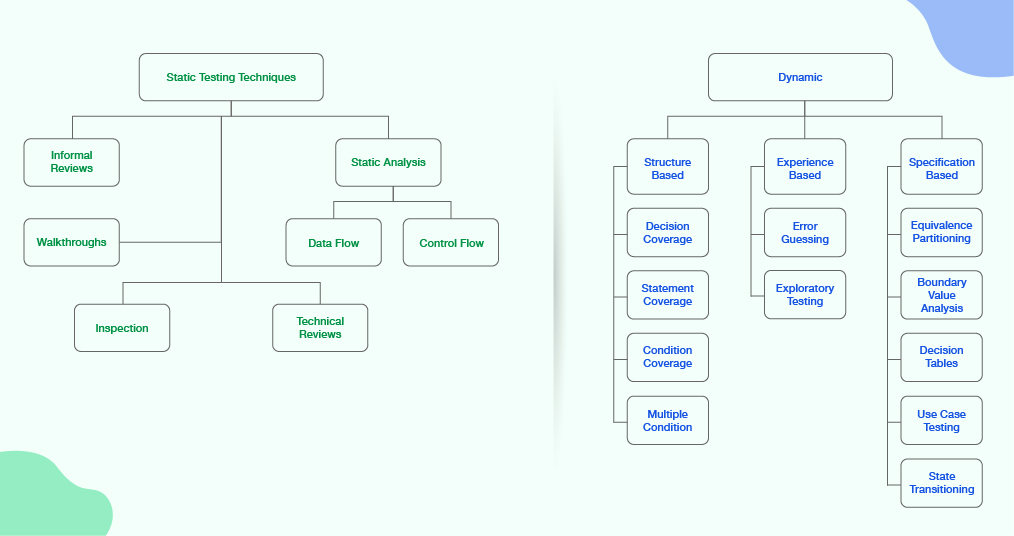
**Why we use Test Case Design Techniques?**

The intention of the ‘test design’ phase is to produce test scenarios and executable test cases that provide testing coverage and creates maximum chances to find defects when executed in the ‘test execution’ phase. [**https://testsigma.com/blog/manual-testing-vs-automation-testing-which-one-should-you-choose/**](https://testsigma.com/blog/manual-testing-vs-automation-testing-which-one-should-you-choose/)

****

****

**Software testing techniques broadly two types:**

1. **Static**

a)reviews

b)walk throws

c)tech reviews

d)inspection

e)static code analysis

f)data flow/control flow

1. **dynamic**
2. White box→ statement coverage, condition, decision, multiple decision,
3. Experienced base→ error discussion, explotetray testing
4. Black box→ Equivalence partitioning(EP),boundary value analysis(BVA), decision table(DT) , state transition(sT), use case testing

**Equivalence partitioning(EP)(e-commerce ar discount % module mone rakhbo): (invalid | valid | invalid)**

dividing the input or output data of a system into groups that behave in the same way, and then selecting one representative value from each group to test.

<https://www.linkedin.com/advice/0/what-benefits-challenges-using-equivalence-partitioning#:~:text=Equivalence%20partitioning%20is%20a%20test,from%20each%20group%20to%20test.>

**Advantage:** reduce the testing effort, cost, and duration, while still achieving a high level of coverage and quality. Equivalence partitioning can also help avoid redundant or unnecessary test cases. **equivalence partitioning helps identify and isolate defects(? Jante hobe) more easily**

## **Challenges of equivalence partitioning:** it requires a good understanding of the system requirements, specifications, and functionality. not be suitable for all types of testing, such as performance, security, or usability testing

# **Boundary Value Analysis**

***Maximum of 4n+1 test cases(Single Fault Assumption ? jante hobe)***

* Minimum value.
* Just above the minimum.
* Nominal Value.
* Just below Max value.
* Max value.

**Why Combine Equivalence Partitioning and Boundary Analysis Testing:**

* In this test cases are reduced into manageable chunks.
* The effectiveness of the testing is not compromised on test cases.
* Works well with a large number of variables

[Boundary Value Analysis](https://www.geeksforgeeks.org/boundary-value-analysis-triangle-problem/)

# **Decision Table Testing(developer ra ki ki condition(if/else) bosabe , ai table er maddome test kora jai)**

Example : user registraiton in a web page(name,email,password)

**The Decision table is a highly effective tool utilized for both requirements management and complex software testing. Through this table, we can check and verify all possible combinations of testing. The testers can quickly identify any skipped needs by reviewing the True(T) and False(F) values assigned for these conditions.**

[**https://www.guru99.com/decision-table-testing.html**](https://www.guru99.com/decision-table-testing.html)

# **State Transition Testing(**login in a bank account like bkash. attempt to login but give log in info wrong 3 time lock account)

**4 things are important**

**(state(circle),**

**transition(arrow sign),**

**event(click for an action example click in login button)**

**action(display error or success msg)**

**User** [**Acceptance testing**](https://testsigma.com/blog/acceptance-test-vs-system-test/) **e mainly manual testing er use beshi.**

manual testing performed when human intervention is necessary to test the ready-to-use product before releasing it in the market.

**Automation Testing** is a technique that uses tools to write scripts and execute test cases. It is the best way to enhance the execution speed, effectiveness, and test coverage in software testing.

## **Workflow of Unit Testing**

This testing generally follows four steps:

* The first step is creating test cases or figuring out what you want to test and how.
* The second step is writing the code for the tests.
* The third step is running the tests, If the tests pass, the code does what it’s supposed to do, and you can move on. If the tests fail, there’s a bug in the code, and you’ll need to debug it before moving on.
* The fourth step is maintaining the tests. As your code changes, you’ll need to update your tests accordingly. That might mean adding new test cases or modifying existing ones.

## **Unit Testing Techniques**

Now that you know the basics of this testing let’s look at some techniques you can use.

### **Mocking:**

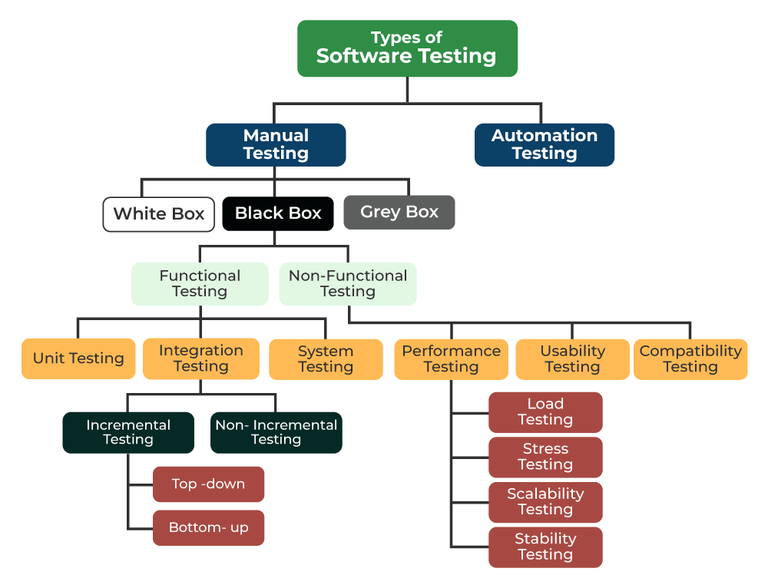
Mocking is when you create dummy objects for real objects in your code. This allows you to test how your code interacts with other objects without using those other objects.

### **Stubs:**

Stubs are similar to mocks, but they’re typically used to test external API calls. For example, say you’re unit testing some code that makes an HTTP request to an external API. You don’t want to make that HTTP request during your unit test because it’s slow, and it might introduce errors that have nothing to do with your code. Instead, you can create a stub that imitates the API call and returns the data you expect from the real API call.

### [**Data-Driven Testing**](https://testsigma.com/data-driven-testing)**:**

Data-driven testing is where you write your unit tests such that they can be run with different sets of data. This is especially useful if you have a lot of edge cases to test, or if the data you’re working with is too large to hard-code into your unit tests.

****