Introduction To Systematic Testing!

Blackbox vs. Whitebox Testing

Black Box Testing	COMPARISON PARAMETER	White Box Testing
Black box testing is based on external expectations as the internal behavior of the application is unknown.	Testing base	White box testing is based on the internal behavior of an application.
It is used for system testing and acceptance testing among others.	Use	It is used for unit testing, and integration testing, among others.
Black box testing does not require programming knowledge.	Programming knowledge requirement	White box testing requires programming knowledge.
It is difficult to automate as the tester and programmer are dependent on each other.	Ease of automation	It is easy to automate as a tester and programmer can be the same person.
The tester can be a developer, programmer, or customer who acts as the end user.	Tester	The tester can be a programmer or a developer.
Granularity is low.	Granularity	Granularity is high.
Code access is not required.	Code Access	Code access is required.
The main objective of black box testing is to check the functionality of the system under test.	Main objective	The main objective of white box testing is to check code quality of the application under test.
It is also known as data-driven testing, functional testing, and box testing.	Alias	It is also known as structural testing, clear box testing, glass box testing, and code-based testing.
Black box testing does not support algorithm testing.	Algorithm testing	Algorithm testing works for white box testing.
It is less time-consuming.	Time	It is a more time-consuming and extensive method of testing.

Code Coverage Standards:

- Condition/Decision

- Statement
- Condition
- Decision
- Branch

- Modified Condition
- Observable Modified Condition / Decision

Challenges in Structural Coverage:

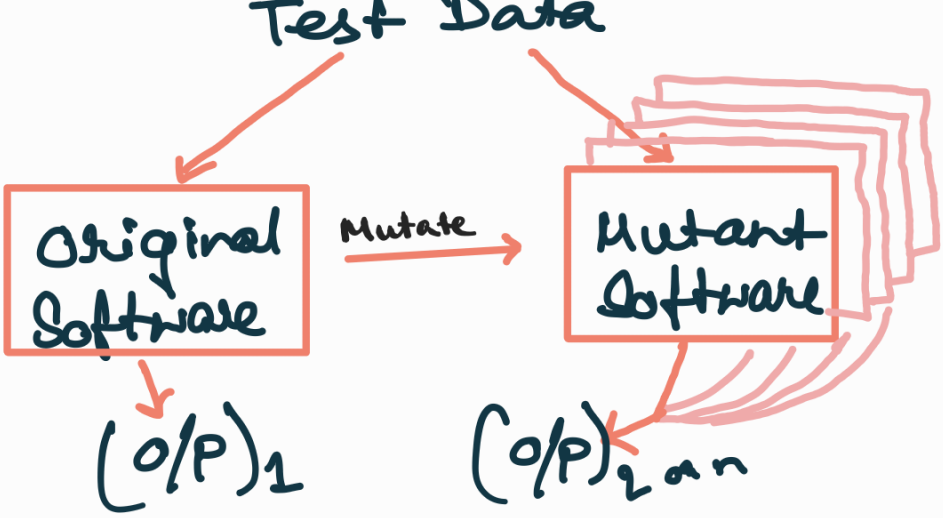
- Interprocedural & gross-level coverage of large O-O programs.
- Regression testing
- Late binding

MUTATION TESTING:

Suppose you have a collection of test cases that all pass... HOORAY!

"Program testing can be used to show the presence of bugs, but never their absence". — Dijkstra

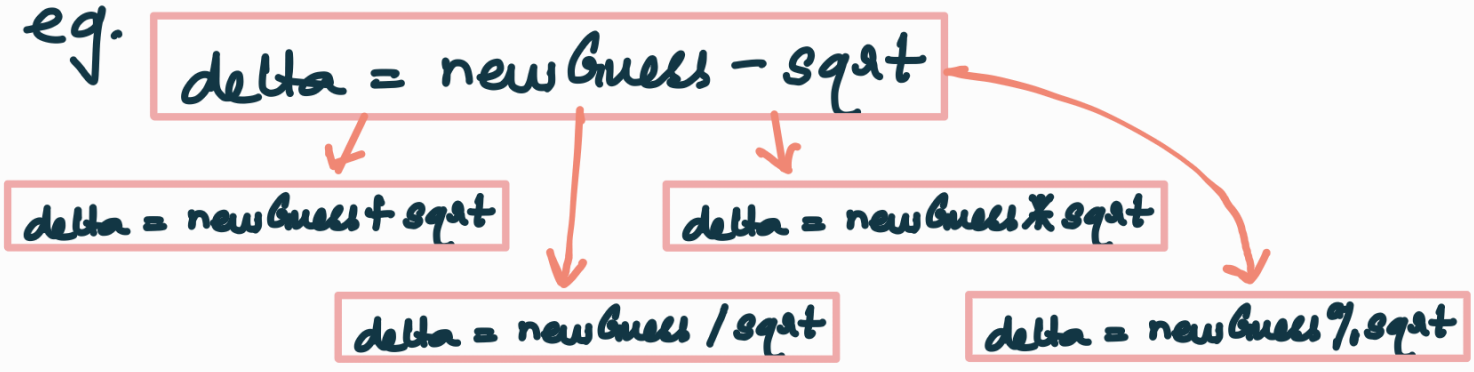
General Idea —



≠ kill mutant ! [pass]
= Problem !

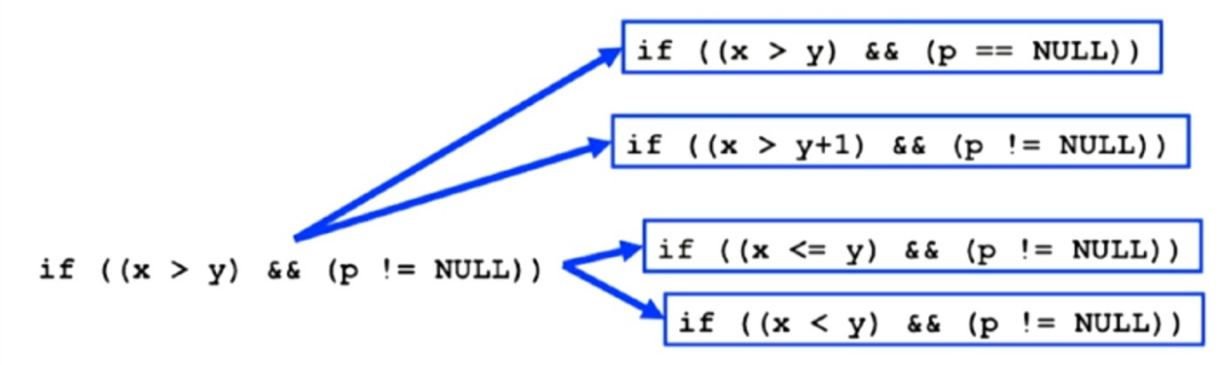
→ How do we create a mutant ?

eg.



→ Mutation operator :

> Operators
> off by one
> Switch variable name of same type
> etc.



→ Calculating Mutation Adequacy Score (MAS).

How well did you do?

$$\text{Mutation Score (MS)}\ \% = \frac{\#\text{Dead}}{\#\text{Mutants} - \#\text{Equivalent}}$$

Summary

How do we know if our tests are good?

→ Take your existing tests for some software under test.

→ Generate a new version of the software with a single error injected.

→ Run your tests.

→ If your tests all pass... YOU HAVE A PROBLEM!