**Unit test**

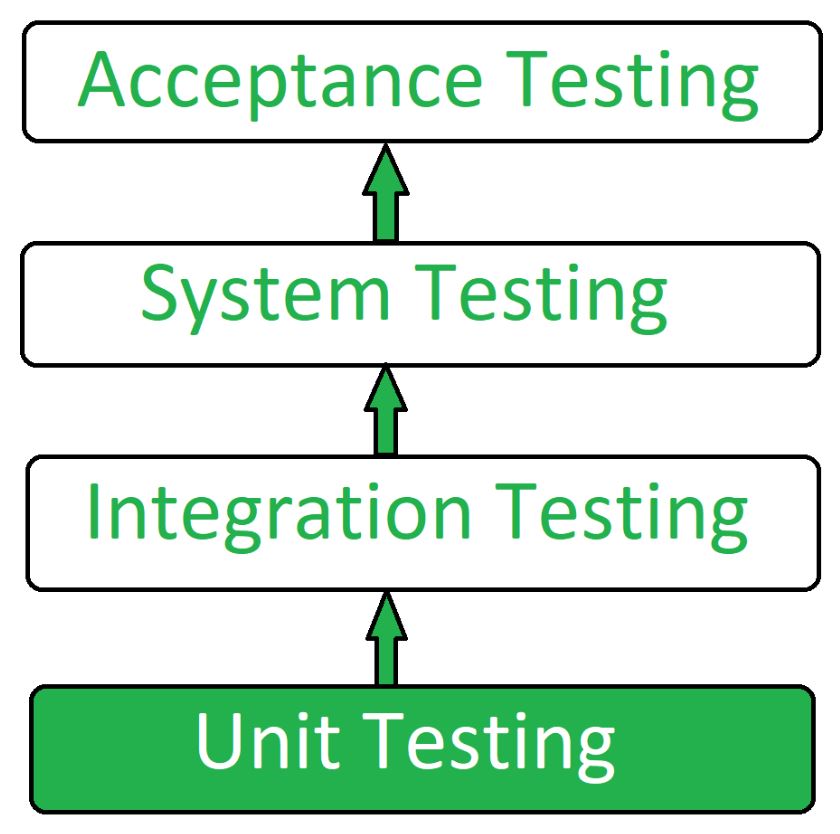
Introduction

**Why?**

* Make the code more reliable, can be reused
* Cost less when we recognize bug early
* Easier to fix whenever changes are made

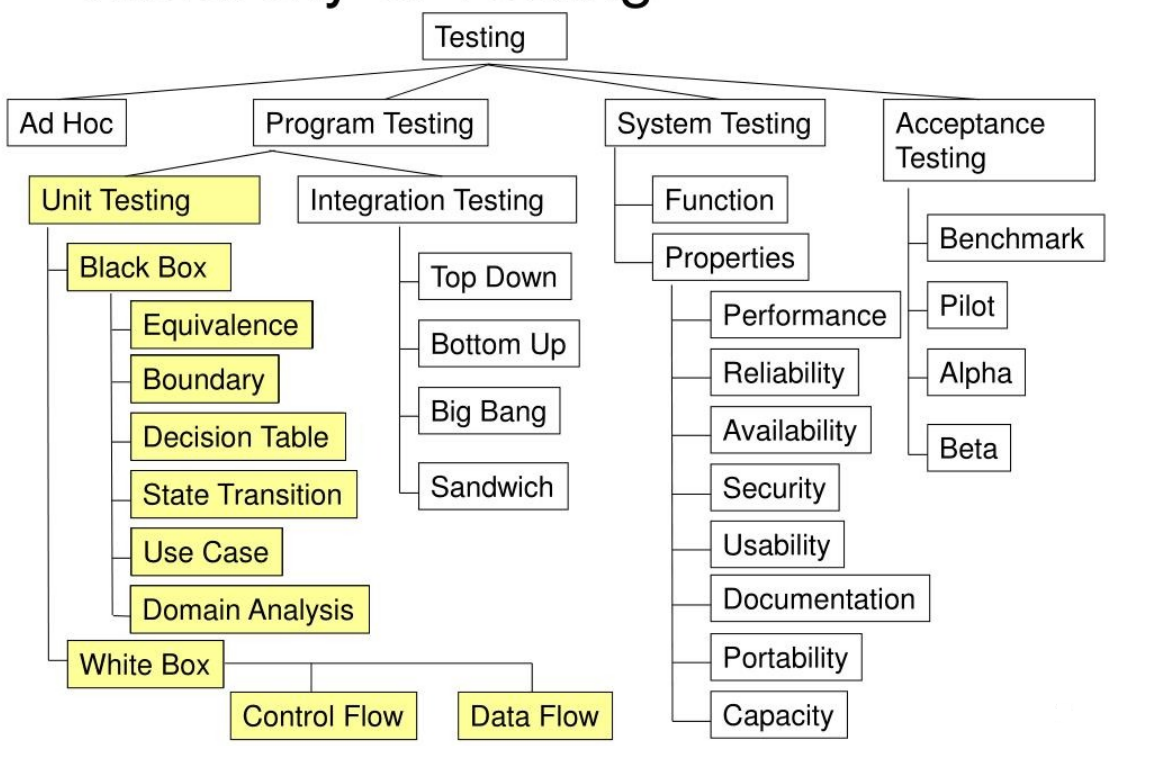
**Note!**

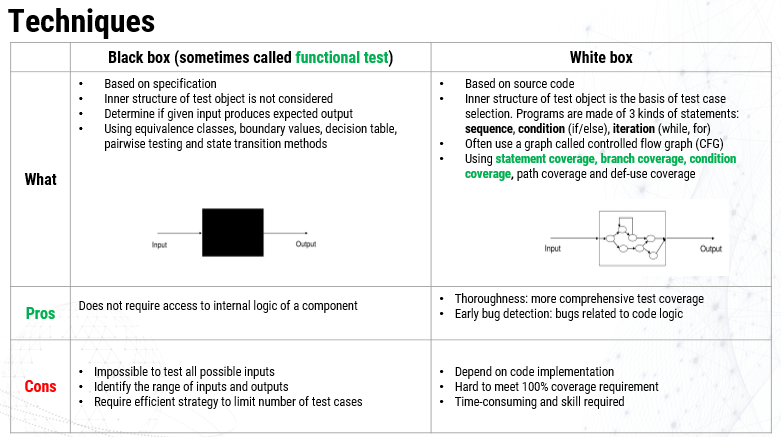
* Test case is independent. Do not call another test case when in a case
* Always checking every single module independently
* Make sure test case names are understandable, unique
* Whenever changes are made, make sure all above test case are tested again
* Bug need to be handle before going to next part
* Do not make ALL test case, only focus on things that can impact the system
* Also need test case for system performance



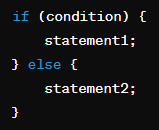
Unit Testing:

* Test focusing on the **smallest units of code**, such as (functions, procedures, subroutines, subprograms, methods, classes)
* Test component in **isolation from the rest** of the system and in a controlled environment (appropriately chosen input data, take guidance from component design)





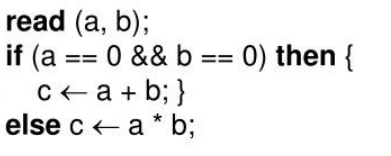
In this section, we focus on **Statement coverage (C0)** and **Branch coverage (C1)** in White box testing



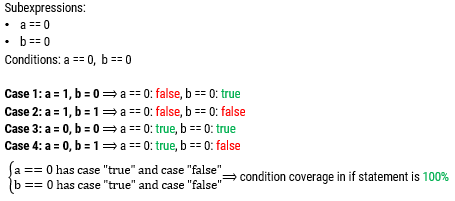
For example we have this function, **Statement coverage** means we make sure that statement1 and statement2 run at least 1 time (do not relevant to condition)

With **Branch coverage**, condition here have 2 branch is true and false, we need to make sure that 2 branchs are executed (do no relevant to statement)

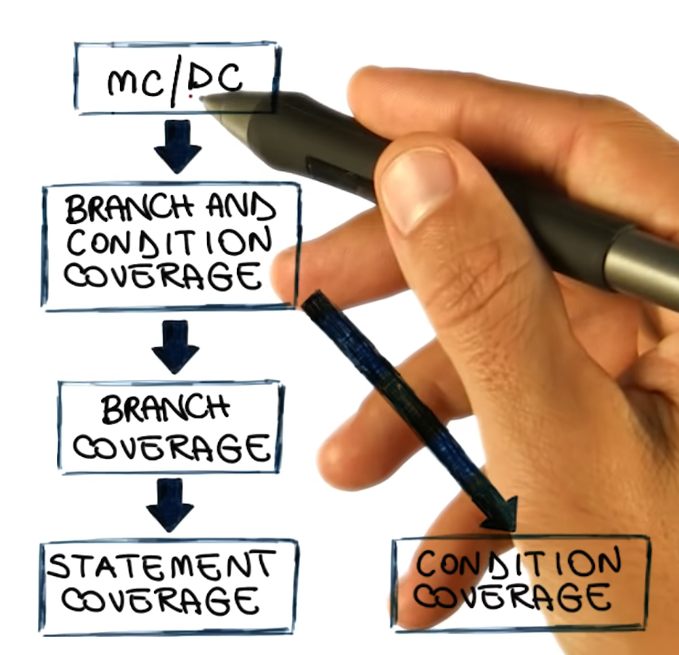
But what happen if ‘condition’ on above example have more than 1 condition:



* **Condition coverage (C2)**



**Modified Condition Decision Coverage (MCDC)** [**https://www.youtube.com/watch?v=DivaWCNohdw**](https://www.youtube.com/watch?v=DivaWCNohdw)



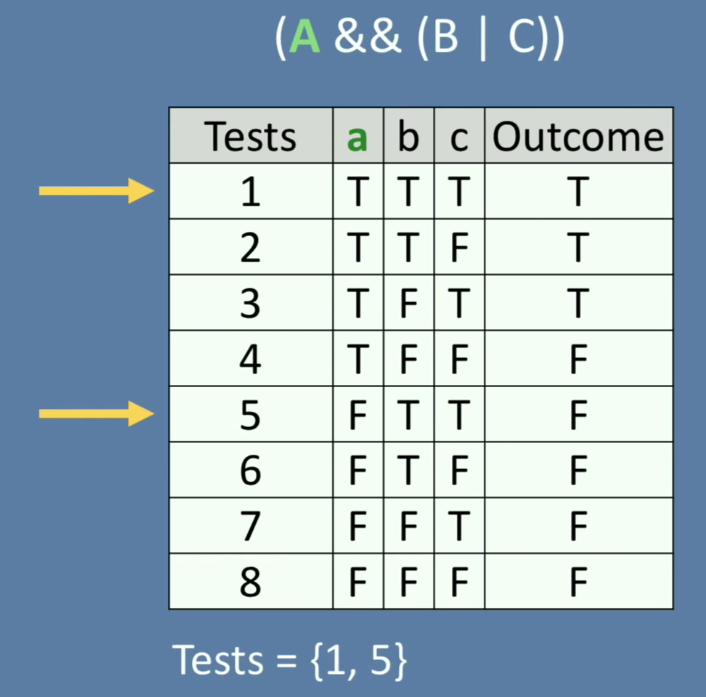
Why?

This includes:

* Branch coverage
* Statement coverage
* Conditions coverage
* Every condition independently affect the outcome (other changes doesn’t affect)

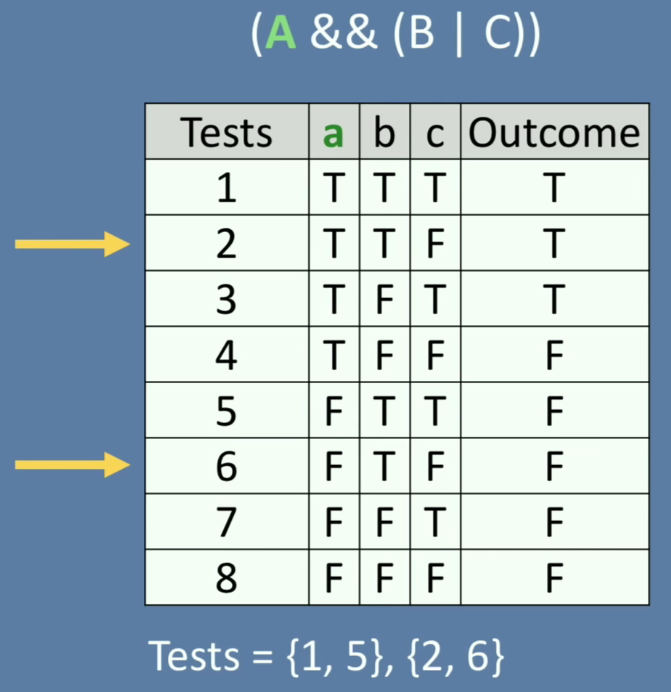
But can decrease the amount of test from 2n to n+1 with n is the number of conditions

**For example:**



In this example we need to check the independency of A by searching on truth table. Let start with test 1, and we change A from T to F to see if the outcome changes also. And luckily, test 5 show that if we change A -> Outcome change to F. This means B and C changes is not neccesary (these changes are reduntdant -> make our test becomes more complex)

Doing the same way we get test 2 and test 6; test 3 and test 7



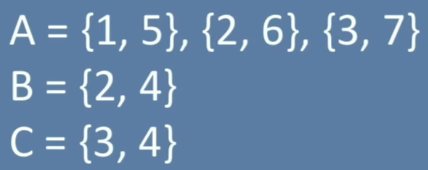
But test 4 and test 8 do not match the rule (outcome is the same)

* With A we got test {1,5,2,6,3,7} that can change the system behavior

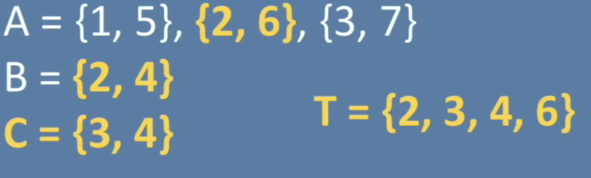
Coming to B, we start with fair test 1 and test 3 -> not match

+ Fair 2 and 4 -> match. The rest cases do not match

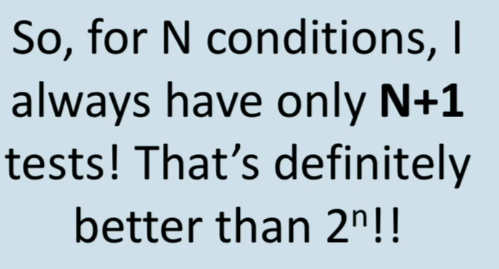
Coming to C, we only get fair 3 and 4



We just need AT LEAST 1 test for each condition -> we got :



(Why don’t choose {1,5} or {3,7}, here we can pick {3,7} because 3 is in C test case already -> less case. Choose {1,5} is still ok but it increases 1 more case in total)

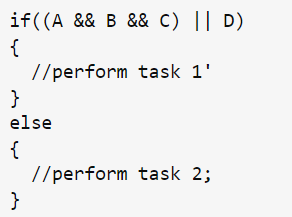


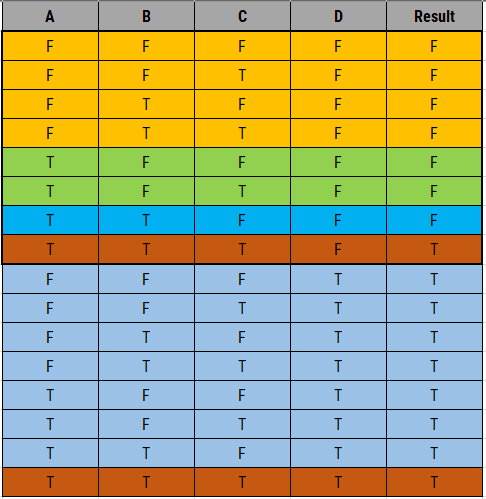
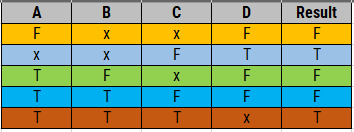
<https://www.youtube.com/watch?v=bwtALQVx86w>

<https://www.youtube.com/watch?v=DivaWCNohdw&list=PLAwxTw4SYaPkoQFThzsc9e7Fe3QV_KJCs&index=69>

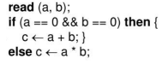
If there is no independent condition -> We need to comeback to C0,C1 and C2

Use to remove redundancy in Condition coverage (C2) -> only test the important combination condition

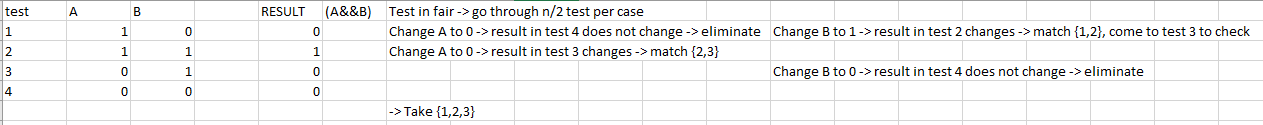


 -> 

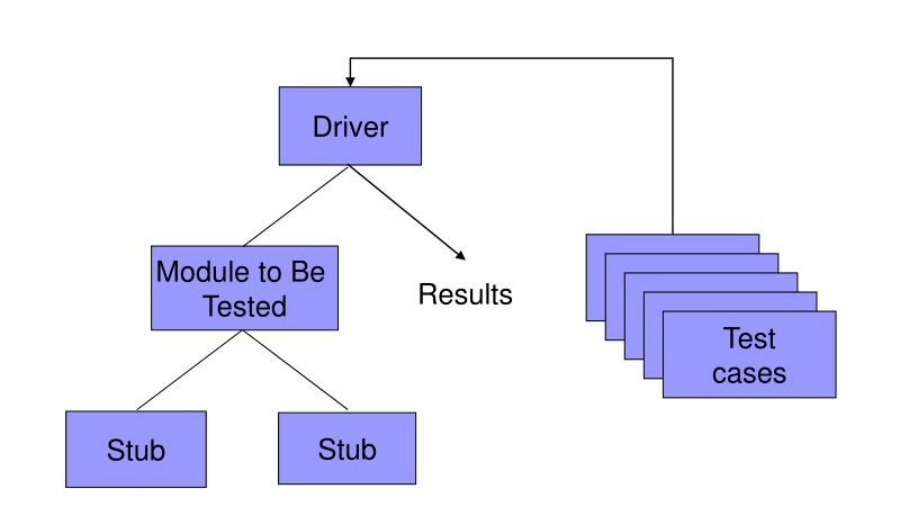
**Example:**



This is how we brainstorm



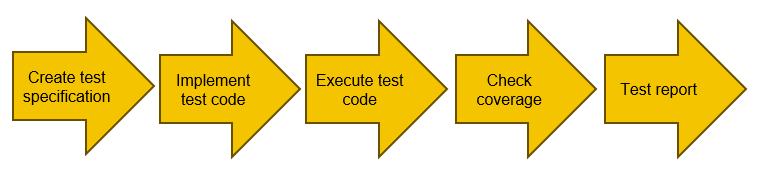
**Unit test procedures**



Start with test cases above, after filter necessary test cases(is the core of Unit Test, written by tester), we take it to **Driver**: that can link module to be tested, stub and test cases

* What is stub (stub function/classes,…): Imagine our program need to be tested but some part of it does not complete/already tested, so we need to stimulate the behavior of that part (return an output with specific input). For example the module multiple\_two\_number(int a, int b) does not complete, we can use stub to take input with a = 2, b=3 and output is 6 without going through calculation inside that function.

**How to perform UT?**



Test hello world