Agenda.

→ Why Testing ?

→ TDD (Test Driven Development)

→ Flaky Test cases.

→ Types of Testing
  - Unit Testing
  - Integration Testing
  - Functional Testing

⇒

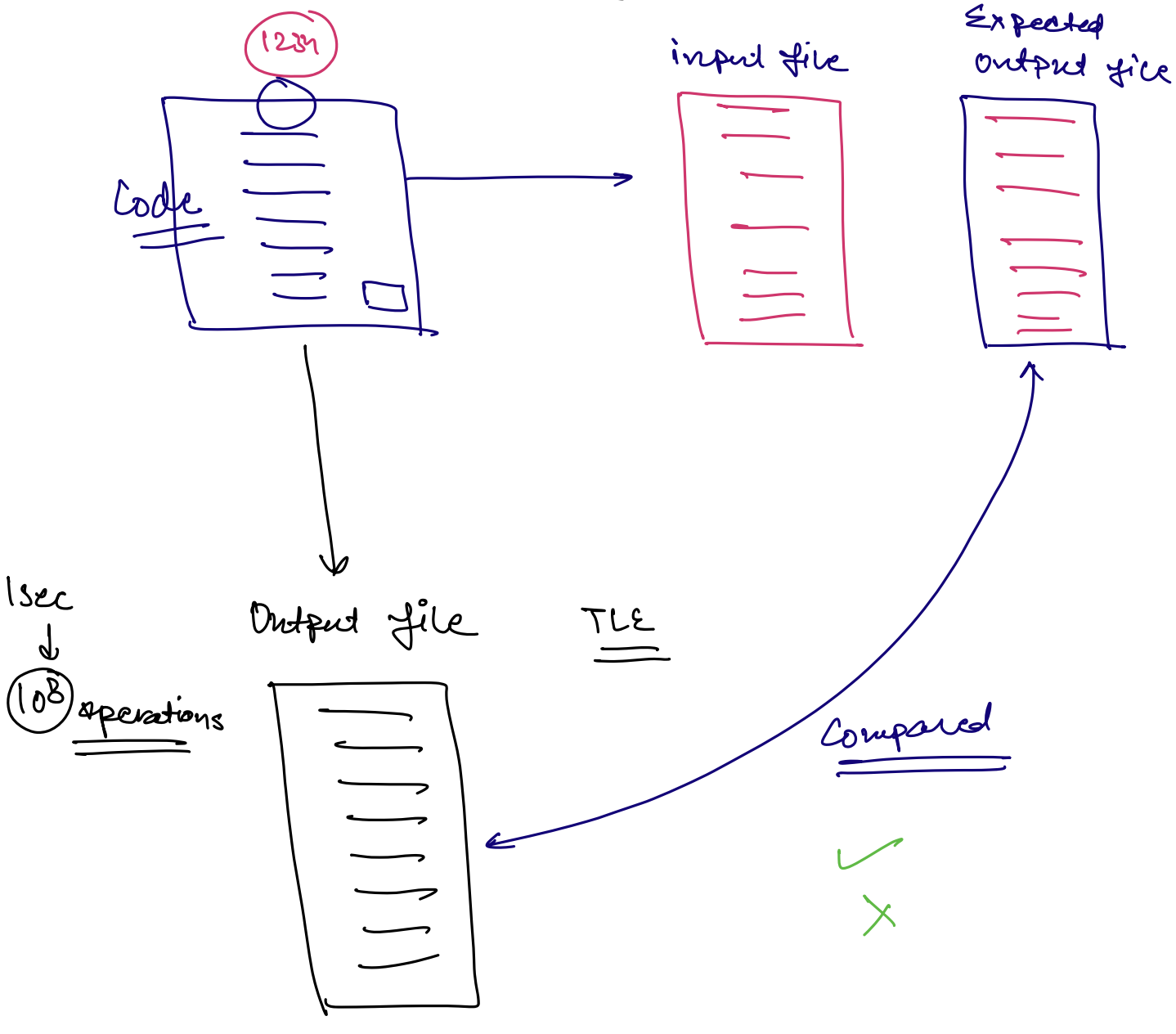
```
int numberOfLanes(int lanesOnOneSide){
    return multiplyUtil(lanesOnOneSide);
}

int multiplyUtil(int x){
    return x*2;
}                    3

int noOfMarriedPeople(int noOfUniquePairs){
    return multiplyUtil(noOfUniquePairs);
}

void testMultiplyUtil(){
    int x = multiplyUtil(2)
    if(x != 2) throw an Exception;
}
```

⇒ We should write test cases, that should get executed automatically before anyone is trying to submit the code and if any of the test case fail, code submission must be allowed.

Github actions.

Edge Cases.

Developers should also write the test cases for the features that they have implemented.



⇒ Before we submit any piece of code all the test cases should be executed.

↓

Comprehensive

Project.

⇒ TDD

↓

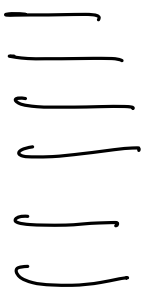
Test Driven Development.

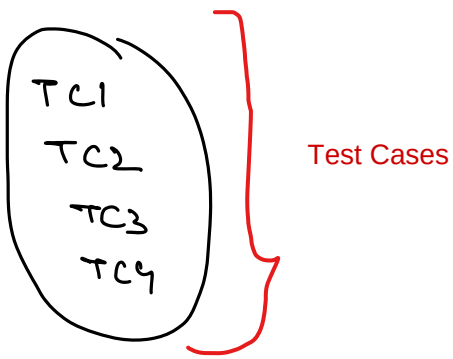
⇒ First write all the Test cases and then implement the feature.

DSA problems on any online judge

⇓

TDD.





⇒ We need to figure out the different conds where our code may break in future.

Conditions

code

Flaky Tests. ⇒ Sometimes they pass & sometimes they fail.

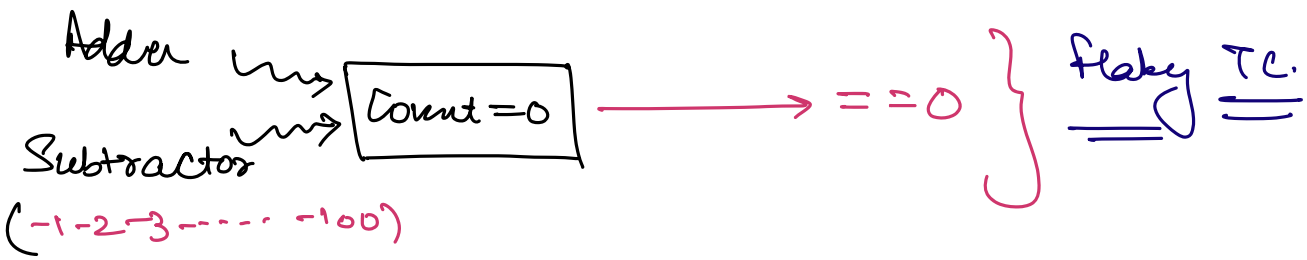
Inconsistent / Unreliable.

→ N/w Connection — if network fails - >can be flaky

→ Randomization — if you are generating some random values and testing against that -> it may pass or it may fail as well

→ Concurrency — synchronization -> when multiple threads are trying to access the same resource then typically flakiness comes into picture

(+1+2+3+---- . . . +100)



⇒ Try not to have any flaky TC in your code.

⇒ Critical Section — when multiple threads are trying to access at the same time

⇒ Race condition — when multiple threads are trying to enter inside the critical section

Asynchronous ≡ Parallel.

We can decide how many request can a server run in parallel, if we decide 10 asynchronous request then request coming after should be in queue.

⇒ Types of Testing

- Unit Testing
- Integration
- functional

```
a() {
    ...
    b();
    ...
}
```

If a() is a function and b() is written inside a(). If b() fails due a bug, a() also fails

Test case of a() can fail :

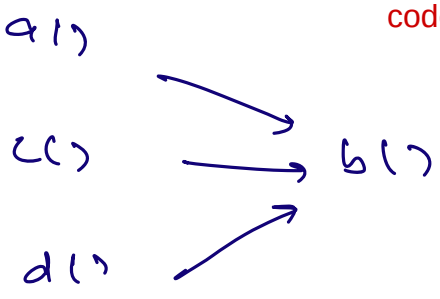
1. a() has some bug.
2. b() has some bug

Ideally, the test case of fun A shouldn't fail because of a bug in fun B.

⇒ Testing in Isolation.

Design test case for a() and b() separately. Test case of a() should not depend on b()

A() should fail coz of B() as it is changing the output of A() and owner of A() should know if someone else code broke my code.



⇒ Test case of fun^ a() should only fail if there's an issue in a(). if & only if

if we donot want our function to fail because b() is failing then we should use the concept of mocking

⇒ MOCKING : Hard coding the output from dependencies.

Here we can use mocking -> assuming that you'll get expected data from b() and b() is passing in that case will you test case for a() pass or not

⇒ We should test our code in isolation.

Mock, Stub & fake are 3 types of mocking doubles

UNIT TESTING.

Every individual piece of code should get tested by a testcase.

Every unit test case are fast coz you are hard coding the values of b() or dependent function

⇒ Every test case will be short & fast.

⇒ No dependency on any other fun^.
(Dependencies will be mocked).

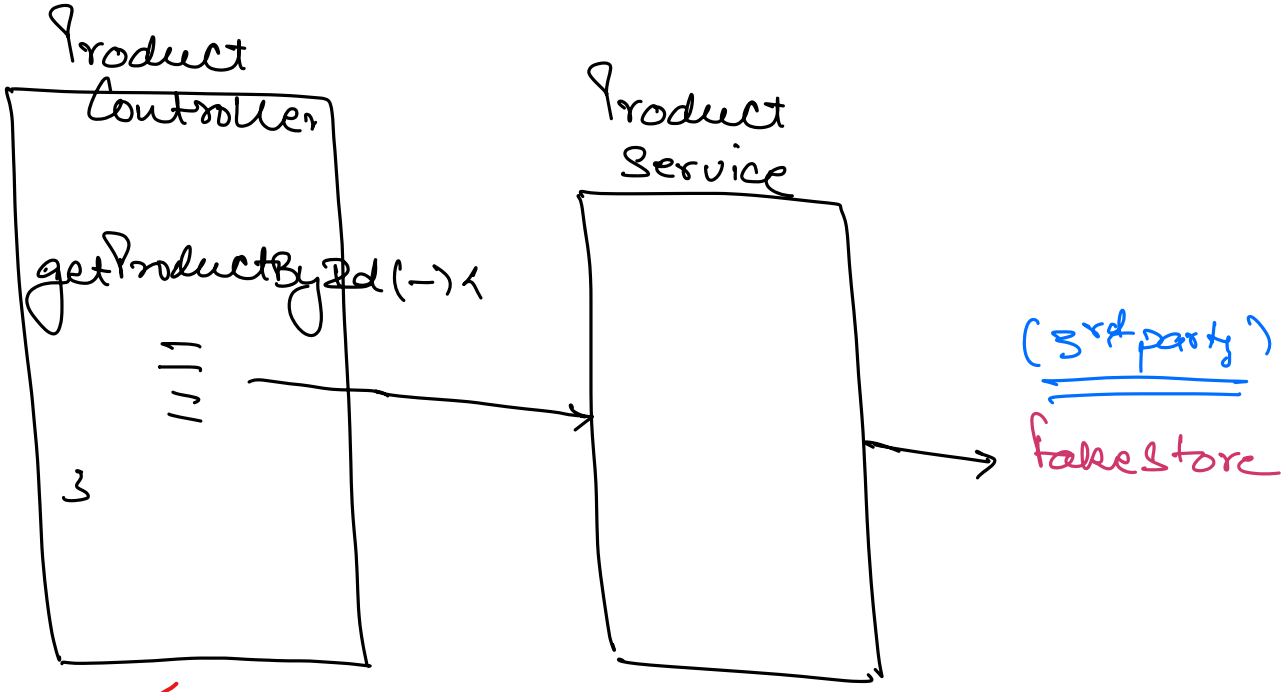
Test Coverage. ⇒ % of code covered by the test cases.

$\frac{980}{1000} \approx 98\%$ ✗ $\approx 80\%$

Integration Testing

⇒ Testing each functionality of our software system where all the dependencies will also be triggered as it is they are present in real codebase

⇒



⇒ Actual dependencies will be triggered, still some 3rd party depencies may be mocked/hard coded

Here we will test all dependent functions except dependent function coming from 3rd party.

We will write integration test cases similar to unit test cases.

In unit cases, you will test a(), b(), c() if they function individually or not but in integration test cases you will actually call b() from a(), c() from b(). And if c() is to call 3rdparty services like payment gateway -> you have to see if payment gateway is giving correct output are your functions working correctly or not

When you combine multiple unit test you get one integration test cases.

Functional = User Testing

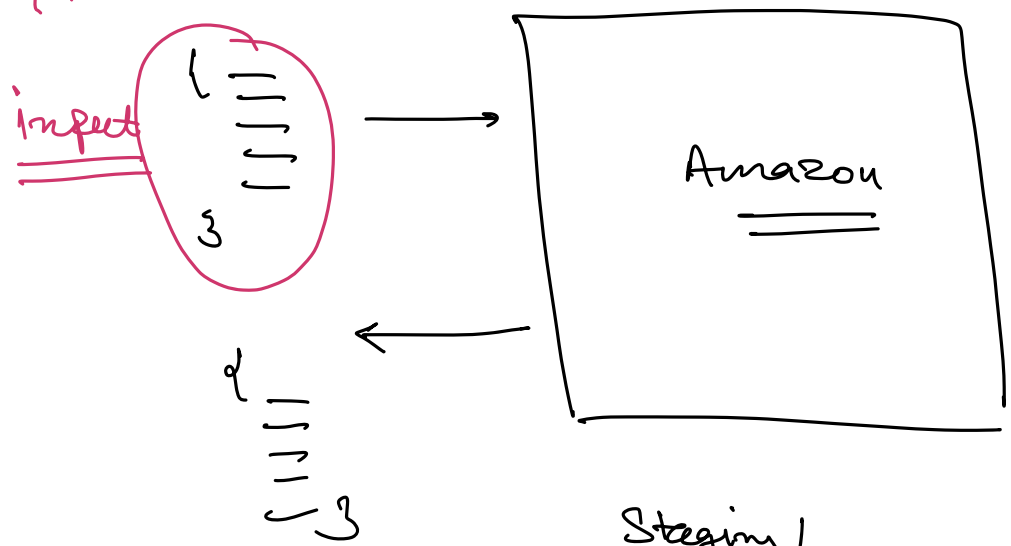
Functional testing is when user gives an input data, is it getting the desired output or not. -> where it is throwing an error, user is not bothered about that

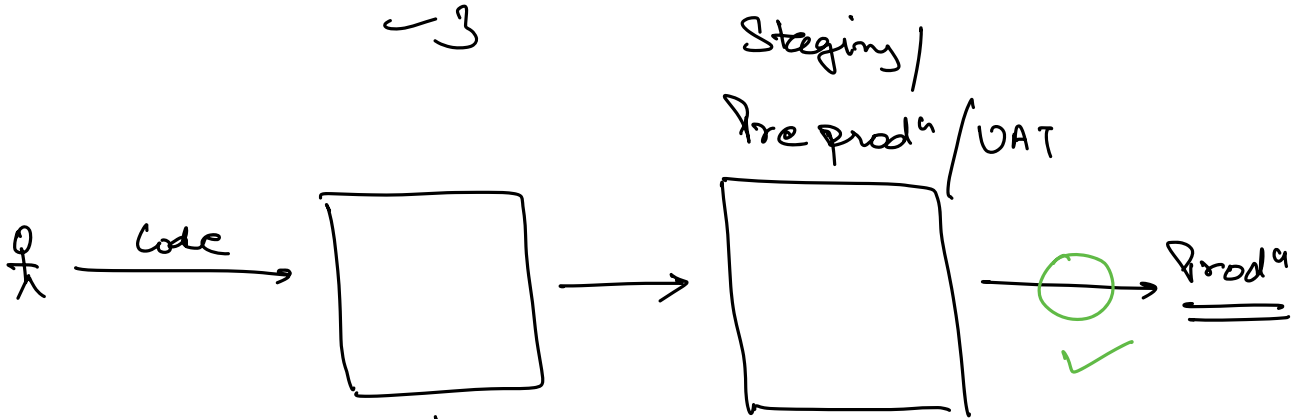
Pre-Prod

UAT

(User Acceptance Testing)

POST / Products



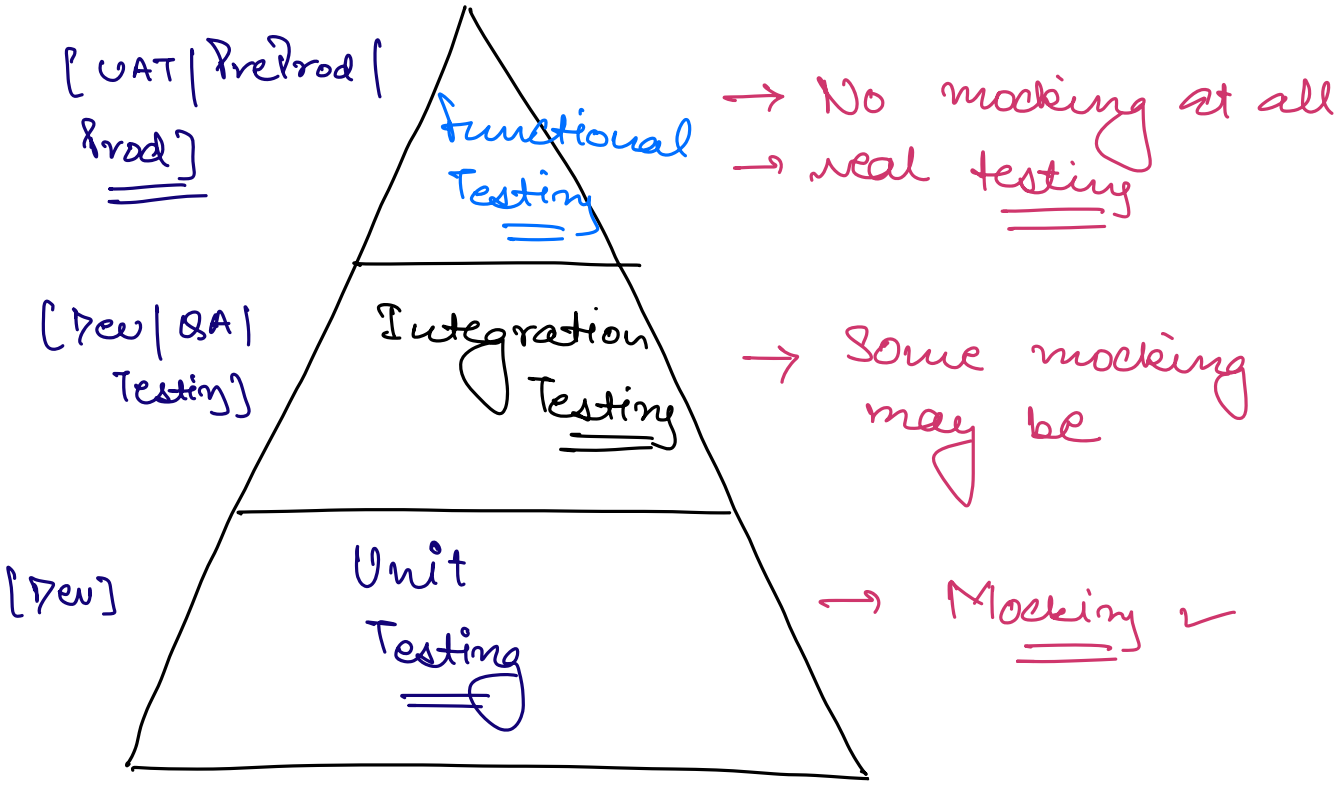


Dev Testing
+
Unit TC
+
Integration Testing
+
QA.

Stages of development. First you write the code in your laptop as a developer
So you do Dev (developement Testing(if this is working or not)
Then Unit Testing
Then Integration testing &
In some companies you donot have QA team

In Some companies, you have Staging/Pre-Production/UAT environment. This UAT environment almost similar to Production environment. You upload the code/ deploy the code to pre-production and ask client to test in Pre-production environment coz you might have the complete access to Pre-production.
You will client time 1 to 7 days lets say and tell them that you have deploy the code to Pre-Prod and client is free to test. Once they accept the changes then you promote it to production environment. This is when you consider your code is pushed to production environment. This is what full Developmental Cycle looks like

Functional Testing is 99% UAT, User Acceptance Testing. In some companies, UAT is also called Pre-Prod/ Pre-Production/ UAT/Stagging environment


[UAT | PreProd | Prod]
Functional Testing
→ No mocking at all
→ real testing
[Dev | QA | Testing]
Integration Testing
→ Some mocking may be
[Dev]
Unit Testing
→ Mocking

—— * ——