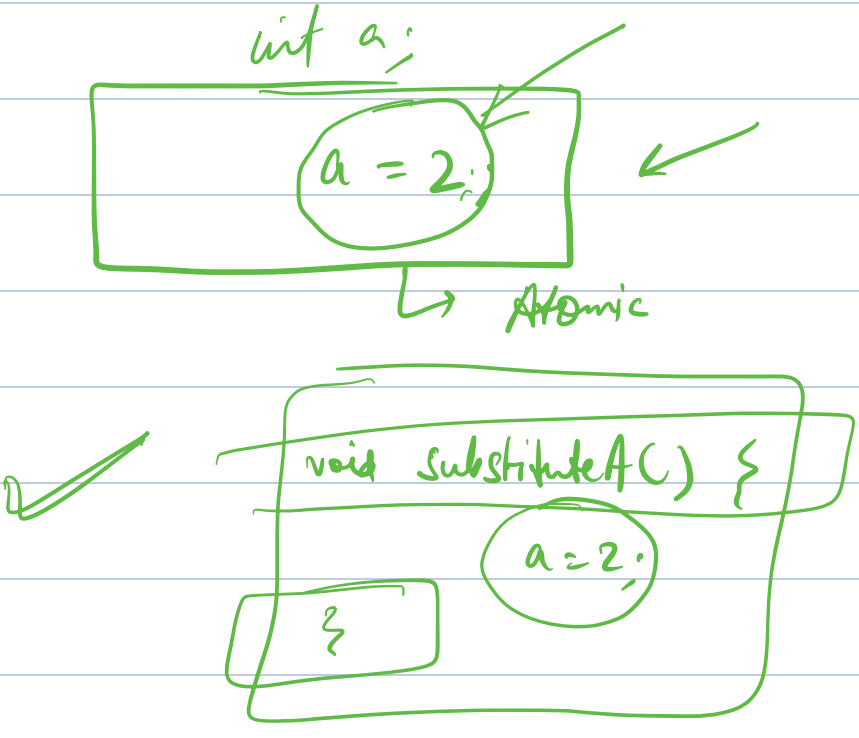
→ Different types of Testing
- ↳ Unit Testing
- ↳ Integration Testing
- ↳ Functional Testing

→ Best Practices of Writing UTs

Unit
- ↳ something very small
- ↳ which cannot be broken further
- ↳ Atomic



1 or More Unit Test for a public Methods

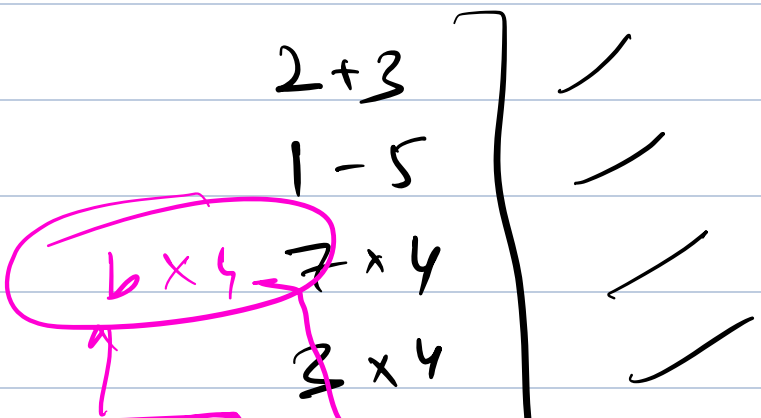
what are we testing here?

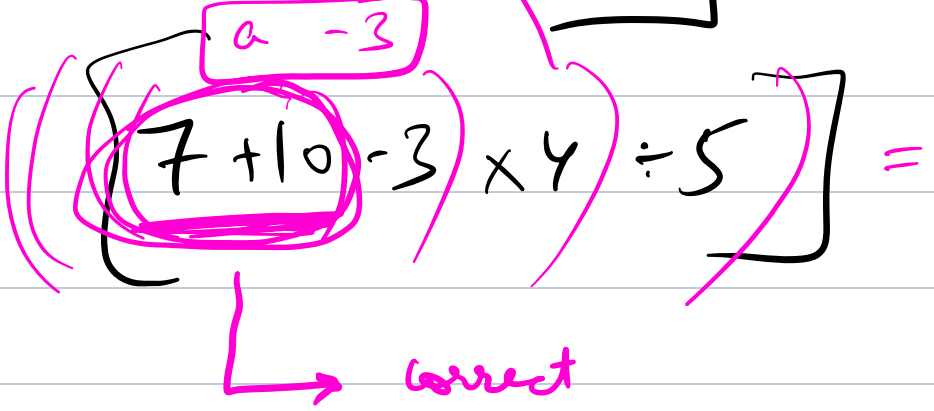
→ whether our code handles all scenarios correctly or not

```
class Calculator {
    int add(int a, int b){
        return a+b;
    }

    int multiply(int a, int b)
    {   return a * b;
    }

}
```





int getElementByIndex(int a[], int index)

{

}

Test Scenarios →

Sad	Arr = NULL,	index = any Number
Sad	Arr = [],	index = any Number
Sad	Arr = [1, 2, 3],	idx = 3 (Size)
Sad	Arr = [1, 2, 3],	index = -1
Happy Edge	Arr = [1, 2, 3],	index = 0
Happy Edge	Arr = [1, 2, 3],	index = 2
Happy	Arr = [1, 2, 3],	index = 1 {mid}

Code Coverage →

% age of your source code which is covered by one or more unit test

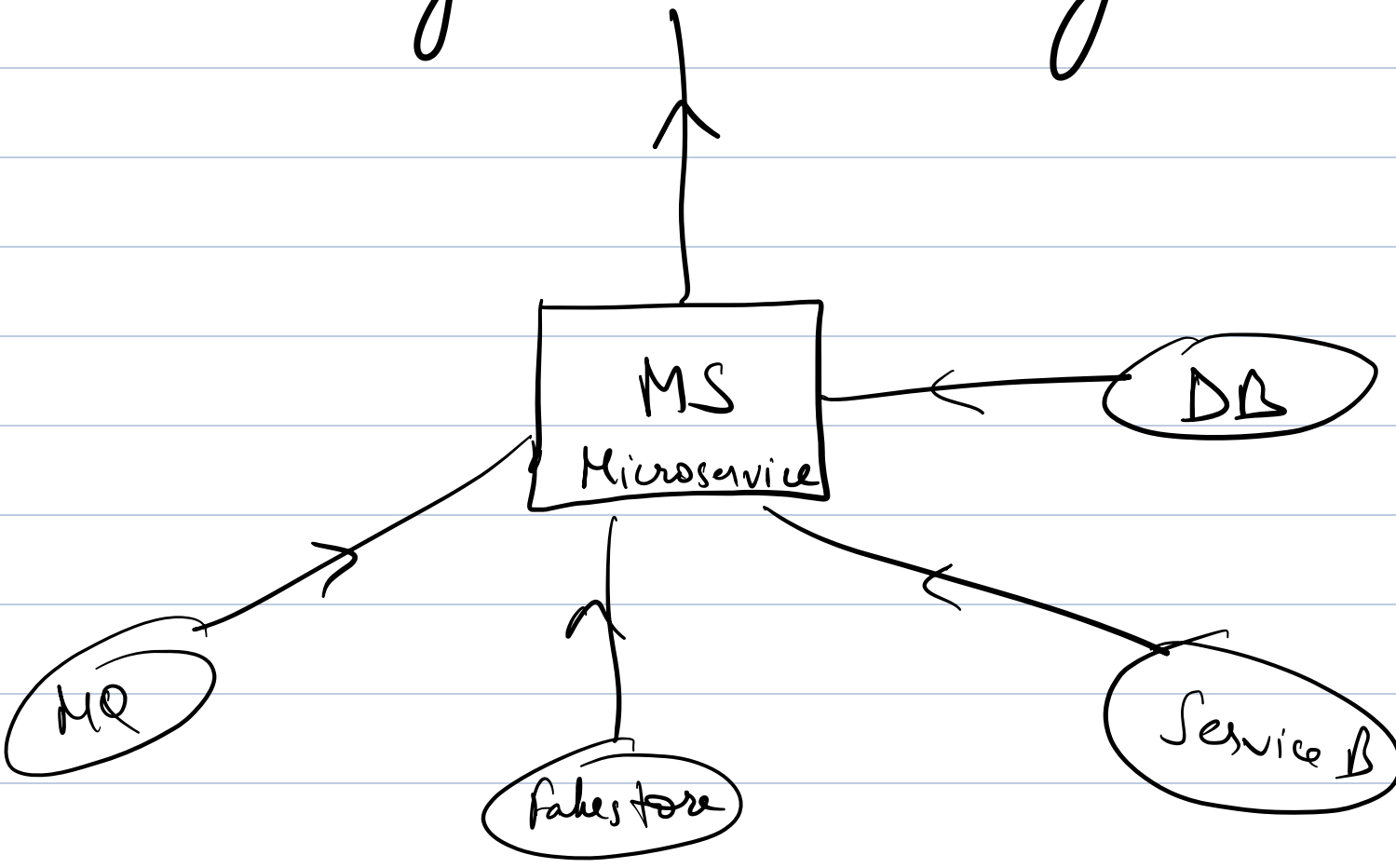
```
int getElementByIndex(int a[], int index) {
    if (index < 0 || index >= a.size())
        return -1
    return a[index];
}
```

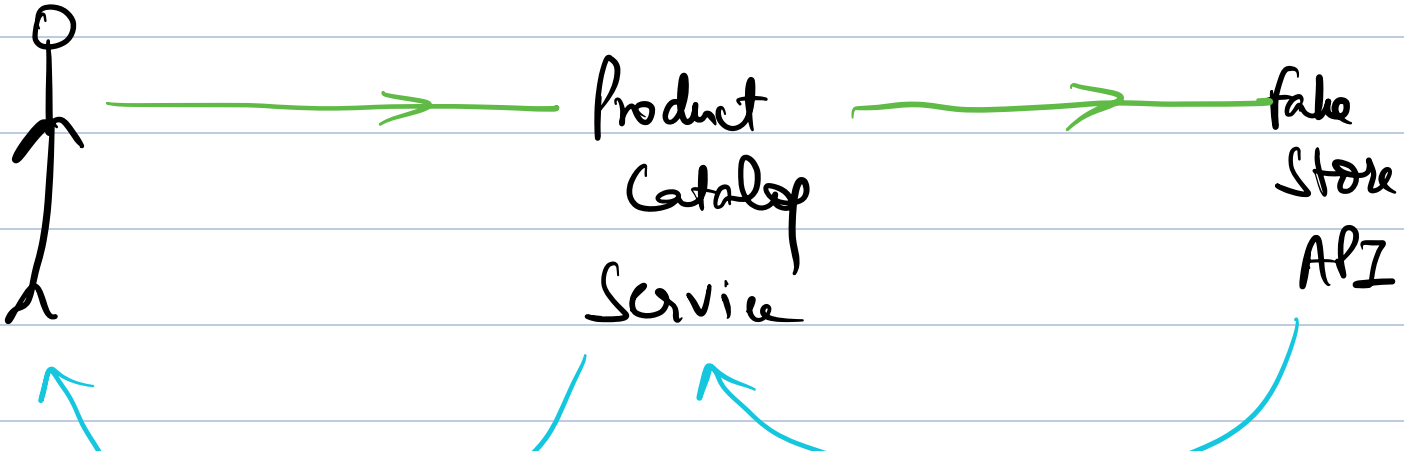
$\frac{2}{3}$ = 66 % {1, 2, 3}, index = 1 ✓

33 % {1, 2, 3}, index = -1 ✓

100 %

Integration Testing

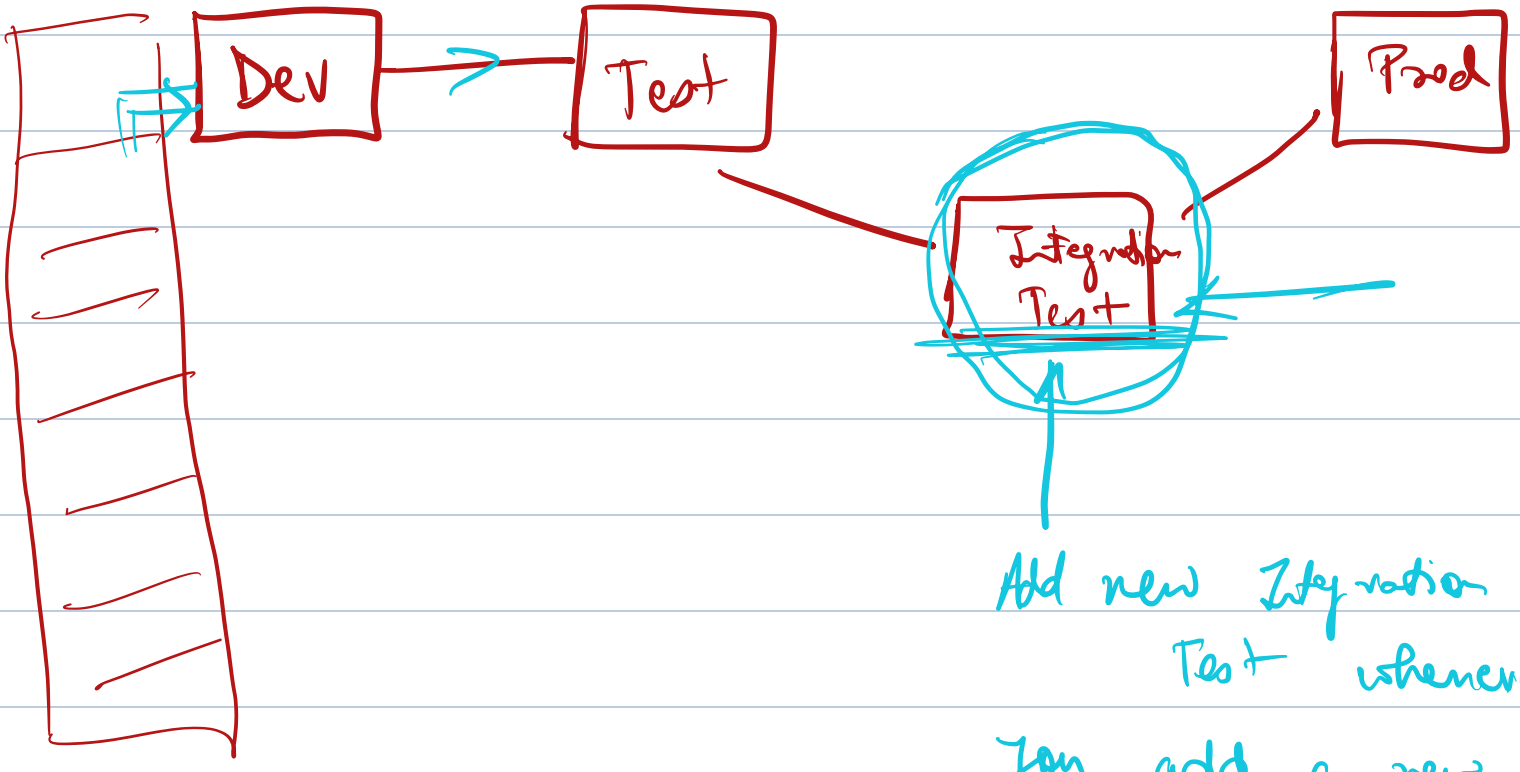




→ Once a day ✓ {Time Trigger}

→ Integrate in CI CD Pipeline & Run after each PR completes {Event Trigger}

Amazon



Add new Integration Test whenever you add a new functionality

$

{

"title" : "——"

"description" : "——"

"Category" : "——"

"price" : "——"

"type" : "——"

}

{

"state" : "ACTIVE",

"Body" : {

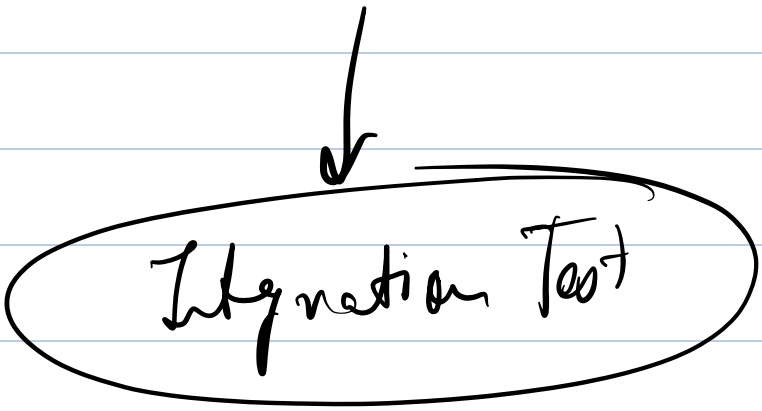
"id" :

"nee"

"description

"price"

}



Functional Testing

↳ Behavior of a System

Facebook / Insta

→ creating a profile
→ deleting a profile
→ friend / unfriend
→ like / follow / share / comment
→ create a page
→ create a post

→ block / Report

Creating a profile

↳ DB ✓

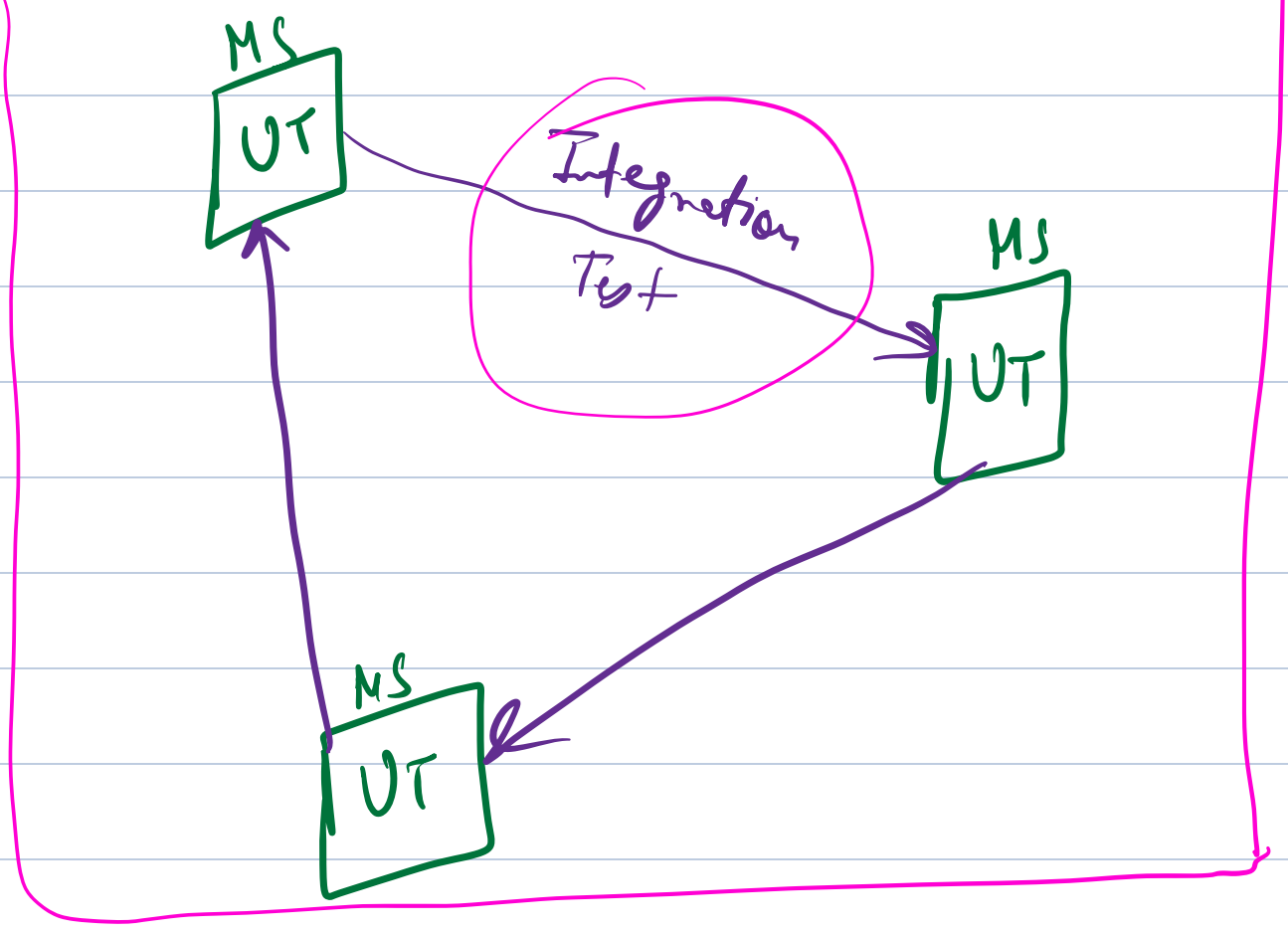
↳ Email / SMS ✓

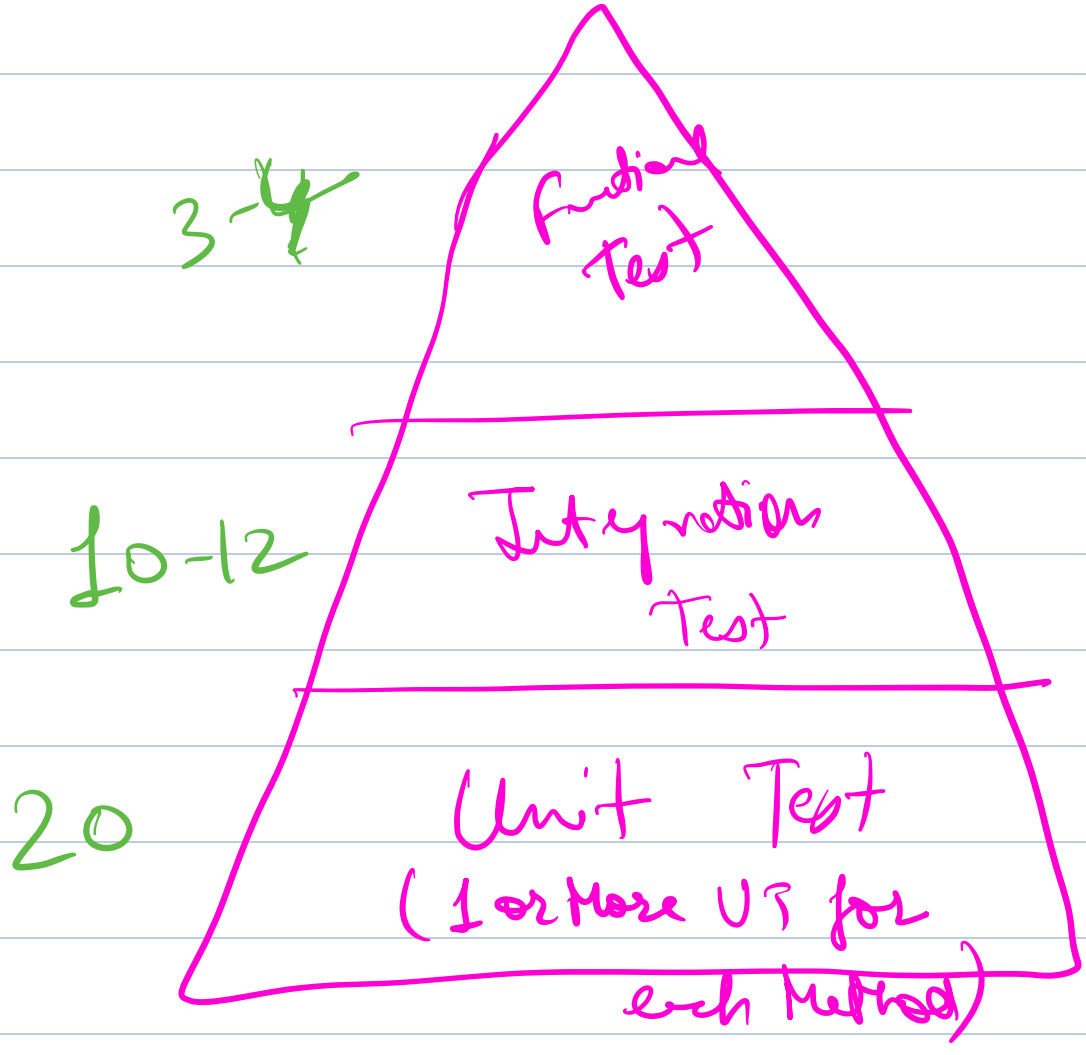
↳ Cache ✓

↳ ⋮

Functional Testing Means testing a system like a real customer

Functional Testing


MS
UT
Integration
Test
MS
UT
MS
UT


3-4
Functional
Test
10-12
Integration
Test
20
Unit Test
(1 or More UT for
each Method)

Best Practices for Writing UTs

1. we should write our UT quickly

Arrange → create

Act → call

Assert → check

Arrange means create object of a class whose method we are going to test

Create Mock of External Dependencies & other things whatever required

---

Act means calling the function which we want to test & store result in some variable

---

Assert means checking if received output matches expected result or not.

```
class Calculator {

    int add(int a, int b){
        return a+b;
```

}

}

```
void testAdd() {
    // Arrange
    Calculator c = new Calculator();
    // Act
    int res = c.add(5, 6);
    // Assert
    assert(res == 11)
}
```

---

2. UTs should be repeatable

In contrary, your Integration Test can fail sometimes

also

---

3. UTs should be Automatic

↓

Less Manual Effort

Inputs are always Hard coded

---

4. UTs should always test (i/p & o/p) behavior & not implementation.

```
int multiply (int a, int b){
    return a * b;
}
```

```
int multiply (int a, int b){
    for (int i=1; i<=b; i++){
        a += a;
    }
    return a;
}
```