

## **SDLC :- Software Development Life Cycle**

It is a procedure to develop the software.

It is a process of creating or altering systems and the models and methodologies that people use to develop these systems.

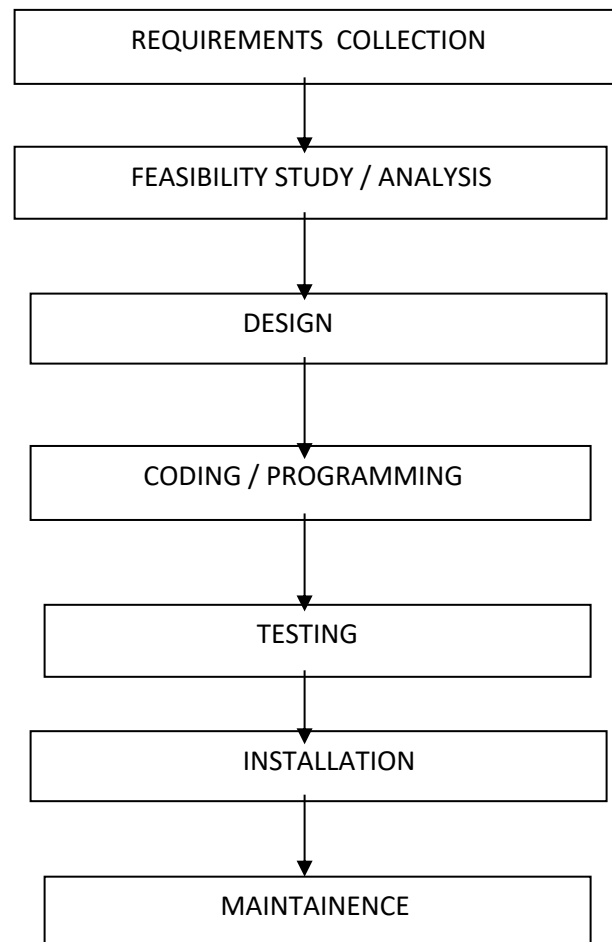
Any SDLC should result in a high quality system that meets or exceeds customer expectations, reaches completion within time and cost estimates, works effectively and efficiently and is inexpensive to maintain and cost effective to enhance.

Different procedures / models are available to develop a software namely,

### **1) Waterfall model**

It is a traditional model

It is a sequential design process, often used in SDLC, in which the progress is seen as flowing steadily downwards ( like a waterfall ), through the different phases as shown in the figure,



### **Requirements Collection :-**

- done by Business Analysts and Product Analysts
- gathering requirements
- translates business language into software language

**For ex,** let us consider the example of a banking software.

### **Feasibility Study :-**

- done by software team consisting of project managers, business analysts, architects, finance, HR, developers but not testers
- architect – is the person who tells whether the product can be developed and if yes, then which technology is best suited to develop it.
- here we check for,
  - technical feasibility
  - financial feasibility
  - resource feasibility

### **Design :-**

There are 2 stages in design,

HLD – High Level Design

LLD – Low Level Design

HLD – gives the architecture of the software product to be developed and is done by architects and senior developers

LLD – done by senior developers. It describes how each and every feature in the product should work and how every component should work. Here, only the design will be there and not the code.

**For ex,** let us consider the example of building a house.

### **Coding / Programming :-**

- done by all developers – seniors, juniors, freshers
- this is the process where we start building the software and start writing the code for the product.

### **Testing :-**

- done by test engineers
- it is the process of checking for all defects and rectifying it.

**Installation :-**

- done by installation engineers
- to install the product at a client's place for using after the software has been developed and tested.

**For ex,** consider the example of a software to be developed and installed at Reliance petrol bunk.

**Maintenance :-**

- here as the customer uses the product, he finds certain bugs and defects and sends the product back for error correction and bug fixing.
- bug fixing takes place
- minor changes like adding, deleting or modifying any small feature in the software product

100 % testing is not possible – because, the way testers test the product is different from the way customers use the product.

**Service – based companies and Product – based companies****Service – based companies :-**

They provide service and develop software for other companies

They provide software which is specified as per the client company's requirement and never keep the code of the developed product and does not provide the software to any other company other than the client company.

Ex – Wipro, Infosys, TCS, Accenture

**Product – based companies :-**

They develop software products and sell it to many companies which may need the software and make profits for themselves

They are the sole owners of the product they develop and the code used and sell it to other companies which may need the software.

Ex – Oracle, Microsoft

**Drawbacks of Waterfall Model :-**

In waterfall model, backtracking is not possible i.e, we cannot back and change requirements once the design stage is reached. Thus the requirements are frozen once the design of the software product is started. Change in requirements – leads to change in design – thus bugs enter the design – which leads to change in code which results in more bugs. Thus the requirements are frozen once the design of the product is started.

Advantage of requirements freezing is we get a stable product because there is no change in design and code.

Drawback of requirements freezing – the customer may not be satisfied if the changes he requires is not incorporated in the product. The end result of waterfall model is not a flexible product.

Major drawback of waterfall model – testing is a small phase which is done after coding. Requirement is not tested, design is not tested, if there is a bug in the requirement, it goes on till the end and leads to lot of re-work.

**Advantages of waterfall model** – requirements do not change nor does design and code, so we get a stable product.

**Applications of waterfall model :-**

Used in – developing a simple application

- for short term projects
- whenever we are sure that the requirements will not change

**For ex,** waterfall model can be used in developing a simple calculator as the functions of addition, subtraction etc and the numbers will not change for a long time.

## **2 ) SPIRAL MODEL**

The spiral model is shown in the figure in the next page.

Ra- requirements analysis of module A. Similarly with Rb, Rc, Rd.

Da – design of module A. Similarly with Db, Dc, Dd

Ca – coding of module A. Similarly with Cb, Cc, Cd

Ta – testing of module A. Similarly with Tb, Tc, Td

In Spiral model, the software product is developed in small modules. Let us consider the figure shown below in developing a s/w product X. X is built by integrating A,B,C and D.

The module A – requirements of the module is collected first and then the module is designed. The coding of module A is done after which it is tested for defects and bugs.

The module B – once module A has been built, we start the same process for module B. But while testing module B, we test for 3 conditions – a)test module B b)test integration of module B with A c)test module A.

The module C – after building module A,B, we start the same process for module C. Here we test for the following conditions – 1) test module c, b, a 2) test for integration of C and B, C and A, A and B.

And thus the cycle continues for different modules. Thus in the above example, module B can be built only after module A has been built correctly and similarly for module C.

**DESIGN**

**REQUIREMENTS COLLECTION**

	Ra	Rb	Rc			
Dc	Db		Da			Rd
Cc	Cb	Ca	Ta		Tb	Tc

## CODING

## TESTING

For spiral model, the best example that we can consider is the MS-Excel application.

The MS-Excel sheet consists of a number of cells that are the components of Excel sheet.

Here we have to create the cells first (module A). Then we can do operations on the cells like merge cells into two , split cell into half (module B ). Then we can draw graphs on the excel sheet (module C).

### Advantages of Spiral Model :-

- 1) Requirement changes are allowed.
- 2) After we develop one feature / module of the product, then and only then we can go on to develop the next module of the product.

Whenever the customer request for major changes in requirements in a particular module, then we change only that module and do testing of both unit and integration of units. This change in requirements comes up in a separate cycle just to do the changes.

Whenever the customer request minor changes in the product, then the s/w team makes the minor changes along with the new module to be developed simultaneously in a single cycle. We don't consider making the minor change in a separate cycle of the spiral model due to time and resource constraints.

The documents collected by Business analysts during requirement collection stage is known as **CRS ( Customer Requirement Specification )** or **BRS ( Business Requirement Specification )** or **BS ( Business Specification )**. In this document , the client explains how their business works or the requirement of the s/w he needs. The BA gathers CRS from the client and translates it into **SRS ( Software Requirement Specification )**. The SRS contains how the software should be developed and is given by the BA to developers. For more detailed explanation of how to go about developing the s/w, the BA/developer builds another document – **FS ( Functional Specification )**. FS explains how each and every component should work.

**Drawbacks of Spiral Model** – Traditional model and thus developers only did testing job as well.

### Applications of Spiral Model

- whenever there is dependency in building the different modules of the software, then we use Spiral Model.
  - whenever the customer gives the requirements in stages, we develop the product in stages.
- 

### **3) V – MODEL / V & V MODEL (Verification and Validation Model )**

This model came up in order to overcome the drawback of waterfall model – here testing starts from the requirement stage itself.

The V & V model is shown in the figure in the next page.

**1) In the first stage**, the client send the CRS both to developers and testers. The developers translate the CRS to the SRS.

The testers do the following tests on CRS,

1. Review CRS
  - a. conflicts in the requirements
  - b. missing requirements
  - c. wrong requirements

2. Write Acceptance Test plan

3. Write Acceptance Test cases

The testing team reviews the CRS and identifies mistakes and defects and send it to the development team for correcting the bugs. The development updates the CRS and continues developing SRS simultaneously.

**2 ) In the next stage**, the SRS is sent to the testing team for review and the developers start building the HLD of the product. The testers do the following tests on SRS,

1. Review SRS against CRS

a. every CRS is converted to SRS

b. CRS not converted properly to SRS

2. Write System Test plan

3. Write System Test case

The testing team reviews every detail of the SRS if the CRS has been converted properly to SRS.

**3 ) In the next stage**, the developers start building the LLD of the product. The testers do the following tests on HLD,

1. Review HLD

2. Write Integration test plan

3. Write Integration test case

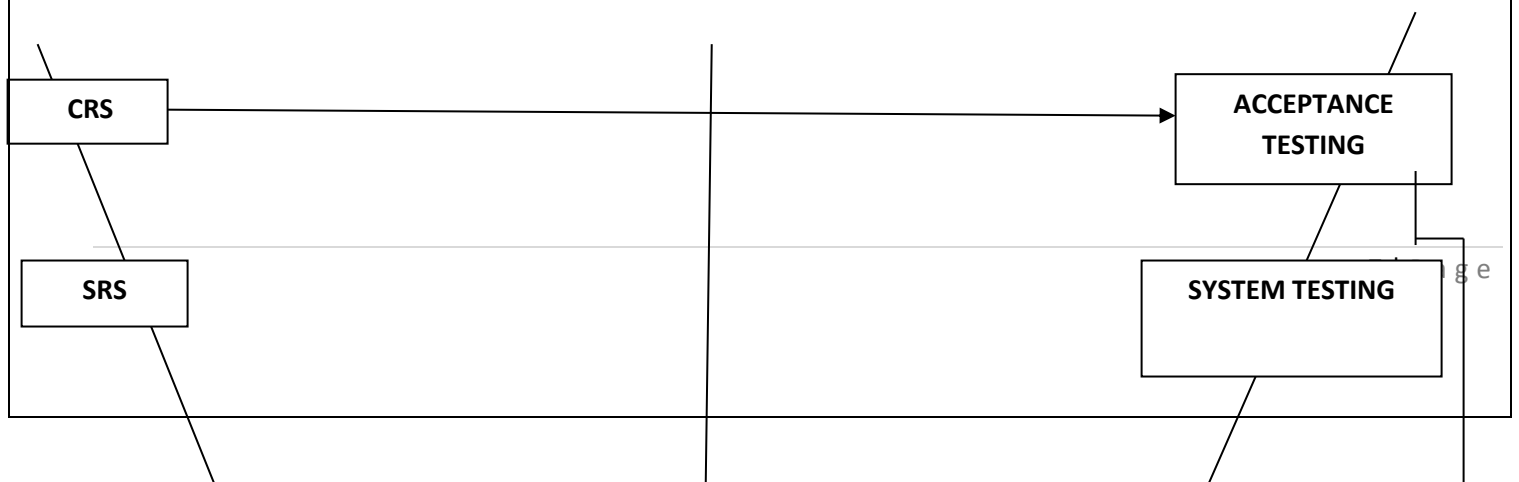
**4 ) In the next stage**, the developers start with the coding of the product. The testing team carries out the following tasks,

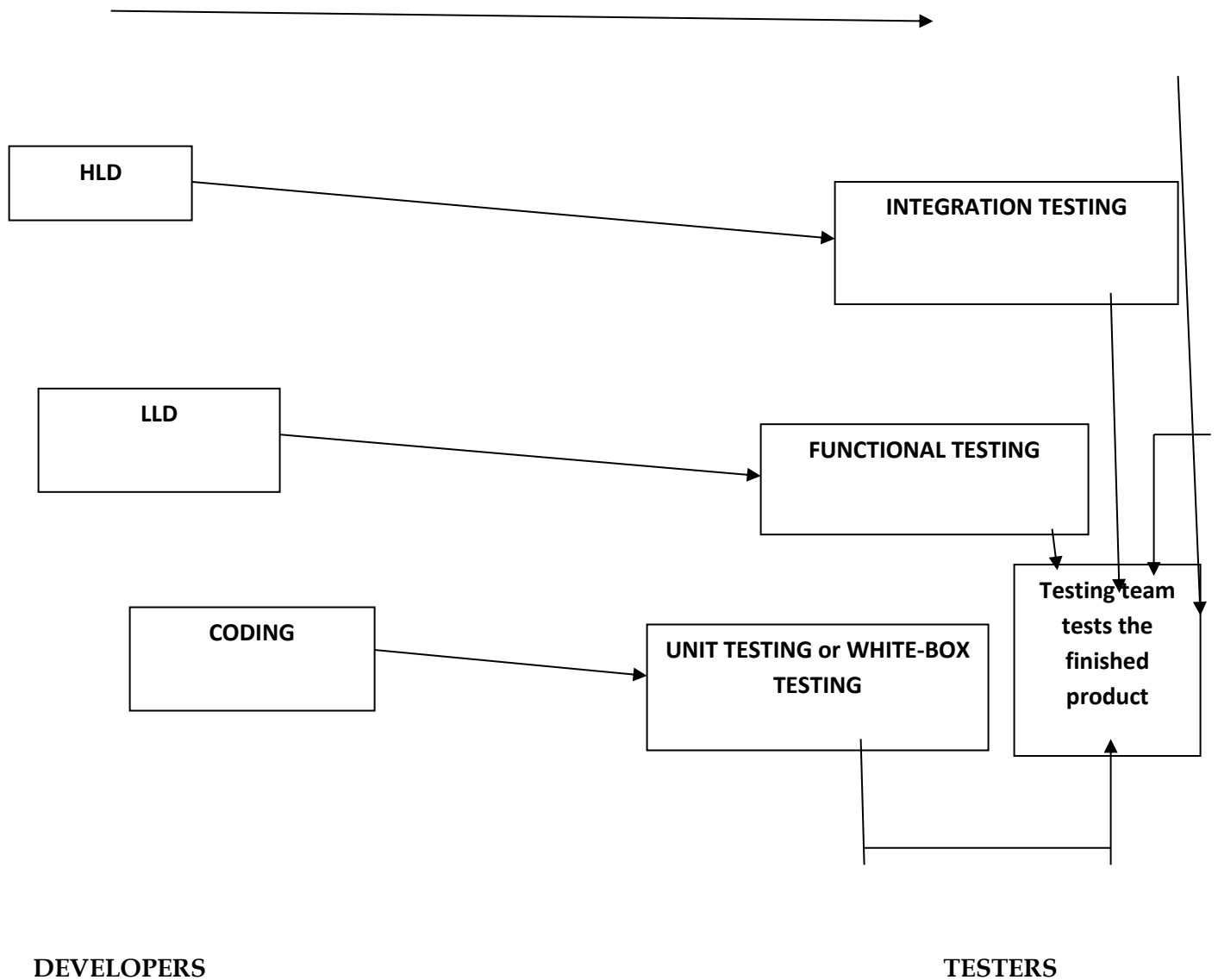
1. Review LLD

2. Write Functional test plan

3. Write Functional Test case

After coding, the developers themselves carry out **unit testing** or **also known as white box testing**. Here the developers check each and every line of code and if the code is correct. After white-box testing, the s/w product is sent to the testing team which tests the s/w product and carries out functional testing, integration testing, system testing and acceptance testing and finally deliver the product to the client.





### How to handle requirement changes in V&V :-

Whenever there is change in requirement, the same procedure continues and the documents will be updated.

### Advantages of V&V model



- 1) Testing starts in very early stages of product development which avoids downward flow of defects which in turn reduces lot of rework
- 2) Testing is involved in every stage of product development
- 3) Deliverables are parallel/simultaneous – as developers are building SRS, testers are testing CRS and also writing ATP and ATC and so on. Thus as the developers give the finished product to testing team, the testing team is ready with all the test plans and test cases and thus the project is completed fast.
- 4) Total investment is less – as there is no downward flow of defects. Thus there is less or no re-work

#### **Drawbacks of V&V model**

- 1) Initial investment is more – because right from the beginning testing team is needed
- 2) More documentation work – because of the test plans and test cases and all other documents

#### **Applications of V&V model**

We go for V&V model in the following cases,

- 1) for long term projects
- 2) for complex applications
- 3) when customer is expecting a very high quality product within stipulated time frame because every stage is tested and developers & testing team are working in parallel

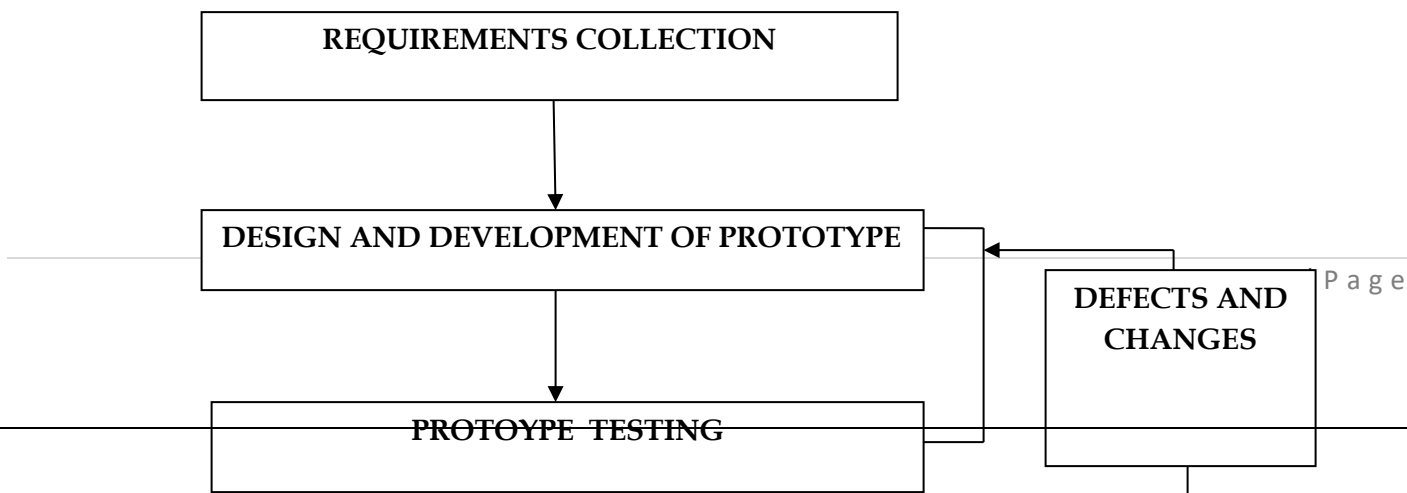
---

#### **4) PROTOTYPE DEVELOPMENT MODEL**

The requirements are collected from the client in a textual format. The prototype of the s/w product is developed. The prototype is just an image / picture of the required s/w product. The customer can look at the prototype and if he is not satisfied, then he can request more changes in the requirements.

Prototype testing means developers/ testers are checking if all the components mentioned are existing.

The difference b/w prototype testing and actual testing – in PTT, we are checking if all the components are existing, whereas, in ATT we check if all components are working.



## APPROVAL

From “REQUIREMENT COLLECTION” to “CUSTOMER REVIEW”, textual format has been converted to image format. It is simply extended requirement collection stage. Actual design starts from “DESIGN” stage.

Prototype development was earlier done by developers. But, now it is done by web designers/content developers. They develop prototype of the product using simple ready-made tools. Prototype is simply an image of the actual product to be developed.

### Advantages of Prototype model

- 1) In the beginning itself, we set the expectation of the client.
- 2) There is clear communication b/w development team and client as to the requirements and the final outcome of the project
- 3) Major advantage is – customer gets the opportunity in the beginning itself to ask for changes in requirements as it is easy to do requirement changes in prototype rather than real applications. Thus costs are less and expectations are met.

### Drawbacks of Prototype model

- 1) There is delay in starting the real project
- 2) To improve the communication, there is an investment needed in building the prototype.

### Applications

We use this model when,

- 1) Customer is new to the s/w
- 2) When developers are new to the domain
- 3) When customer is not clear about his own requirement

There are 2 types of prototype,

**Static Prototype** – entire prototype of the requirement is stored in a word document with explanation and snapshots and instructions on how to go about building the s/w, how the finished product will look like and its working etc.

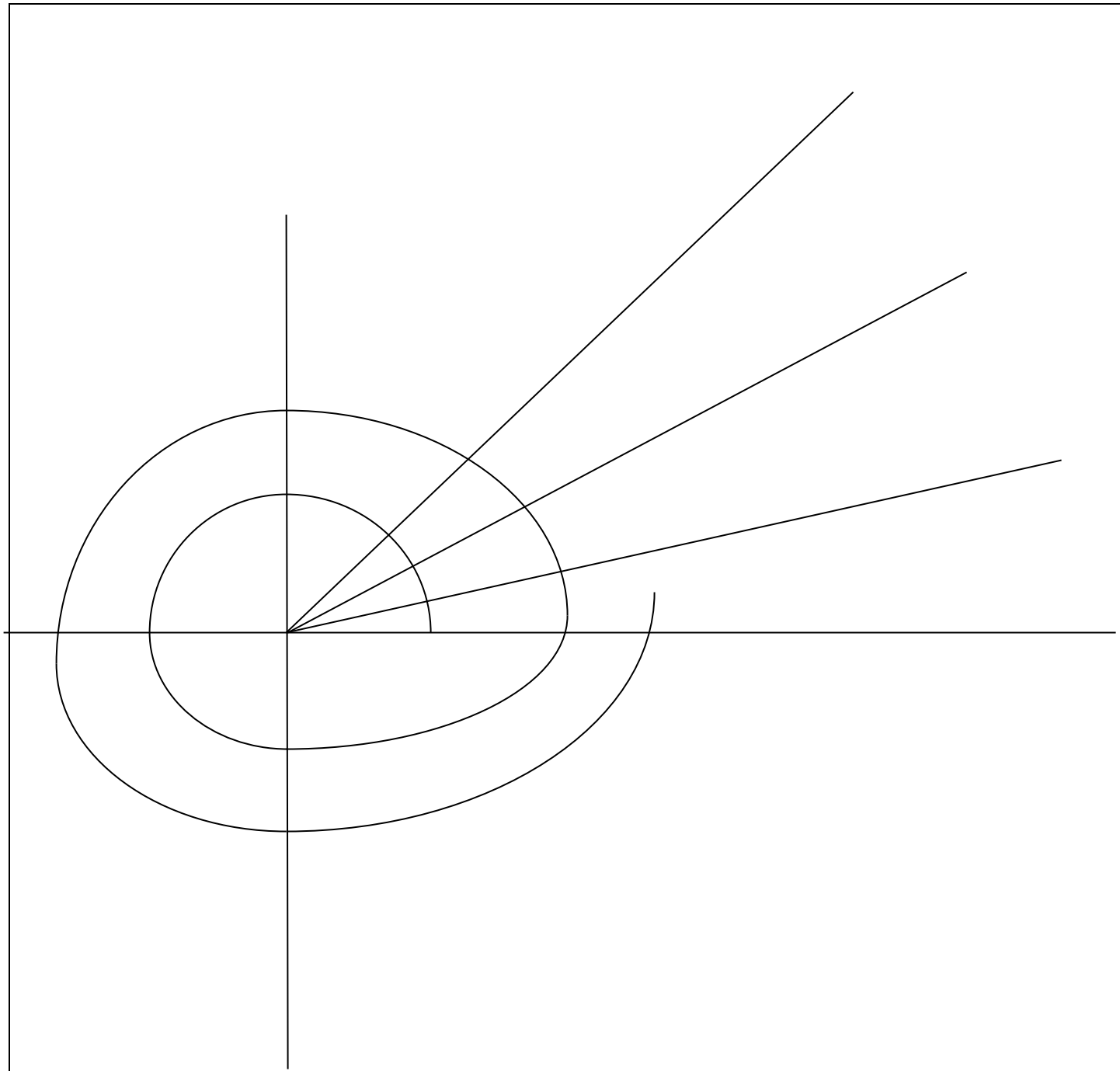
**Dynamic Prototype** – similar to a browser, but we can't enter any information. Only the features are available without entering data. It's like a dummy page, made out of HTML with tags and links to different pages representing features of the project

5) **Derived model or Customized model** – we can take any of the above 4 models and change it as per business needs and requirements

### 6) HYBRID MODEL

It combines 2 or more models and modify them as per business requirements.

#### A) Hybrid model of Spiral and Prototype development models



**We go for this model when,**

- 1) Whenever there is dependency, we go for this hybrid model
- 2) When the customer gives requirement in stages, we develop the product in stages using this hybrid model.
- 3) When the customer is new to the s/w domain
- 4) When developers are new to the domain

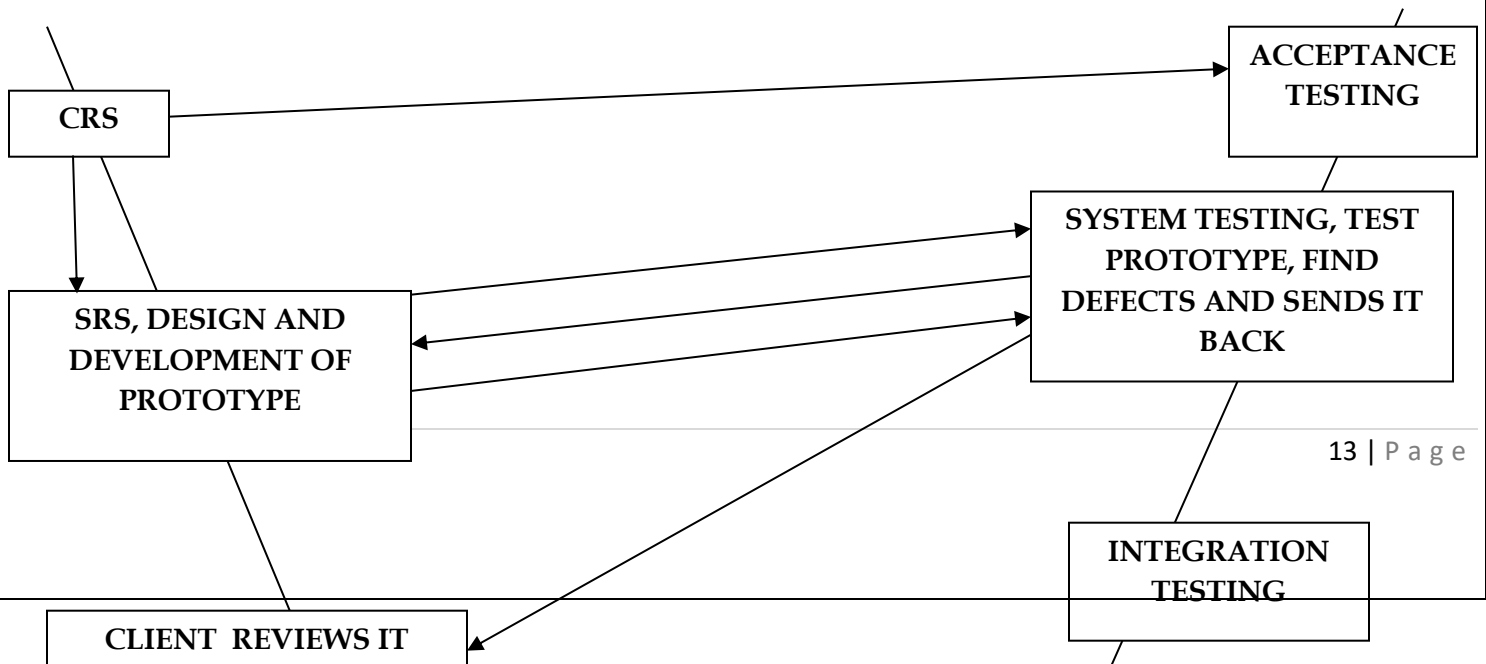
5) When customer is not clear about his own requirements

### Hybrid model of V&V and Prototype model

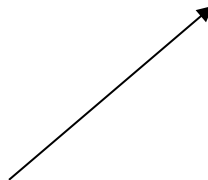
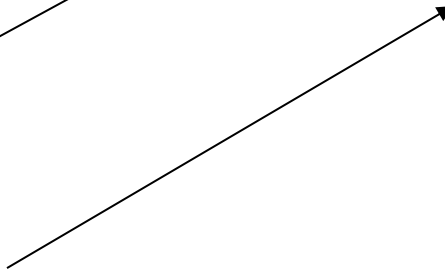
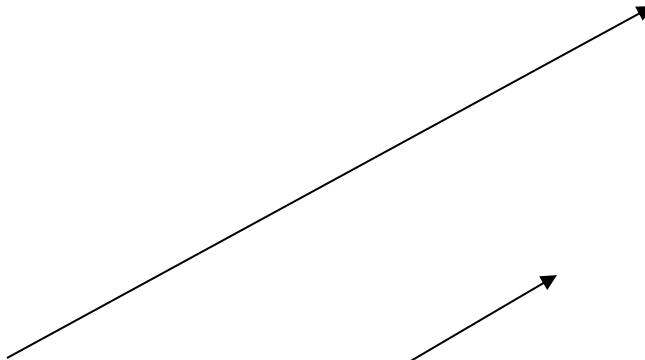
**We go for this model when,**

- 1) Testing starts from early stages of product development which avoids downward flow of defects, thus reducing re-work.
- 2) When customer is expecting a very high quality product within stipulated time frame because every stage is tested and developers and testing team work in parallel.
- 3) When client and developers are both new to the domain
- 4) When customer is not clear about his own requirements

In this hybrid model, the testing team is involved in testing the prototype.



Changes  
And  
defects



DEVELOPERS

TESTERS

## INTERVIEW QUESTIONS

1) *What is SDLC ?*

2) *What are the different models available ?*

ANS) *Tell the 1<sup>st</sup> 6 models and 7) RUP – Rational Unified Process Model 8) Agile Model*

9) *RAD – Rapid Application Development*

3) *Advantages, Disadvantages and Applications of each model*

4) *A model for every project in our resume -> for 3 projects we do, be prepared to tell which model we used for each project and why we used that particular model only. The most common answer we can tell – we used a hybrid of “so and so” model for the reasons such as – client was not sure of his requirements, etc .*

### **Definition of Software Testing**

It is a process of finding or identifying defects in s/w is called s/w testing. It is verifying the functionality(behavior) of the application(s/w) against requirements specification. It is the execution of the s/w with the intention of finding defects. It is checking whether the s/w works according to the requirements.

**There are 3 types of s/w testing, namely,**

**1) White box testing** - also called **unit testing** or **structural testing** or **glass box testing** or **transparent testing** or **open-box testing**

**2) Grey box testing**

### 3) Black box testing – also called as **functional testing** or **behavioral testing**

## **WHITE BOX TESTING (WBT)**

Entire WBT is done by developers. It is the testing of each and every line of code in the program. Developers do WBT, sends the s/w to testing team. The testing team does black box testing and checks the s/w against requirements and finds any defects and sends it to the developer. The developers fixes the defect and does WBT and sends it to the testing team. Fixing defect means the defect is removed and the feature is working fine.

**Test engineers should not be involved in fixing the bug because,**

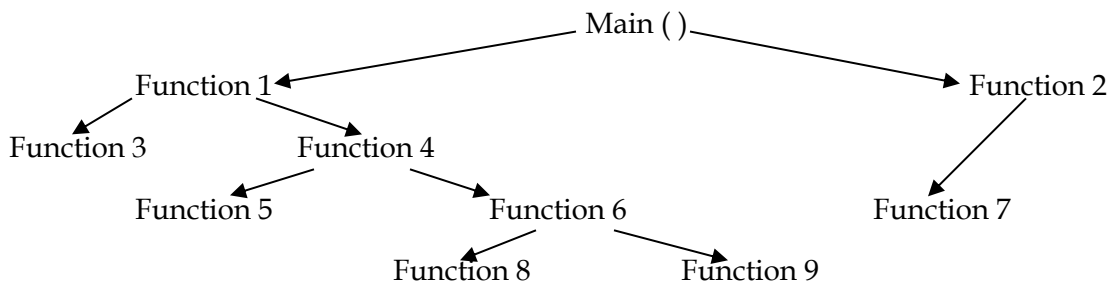
- 1) if they spend time in fixing the bug, they lose time to catch some more other defects in the s/w
- 2) fixing a defect might break a lot of other features. Thus, testers should always identify defects and developers should always be involved in fixing defects.

**WBT consists of the following tests :**

#### **a) Path testing**

Write flow graphs and test all the independent paths.

*Writing flow graphs means* – flow graphs means representing the flow of the program, how each program is interlinked with one another.



***Test all independent paths*** – Consider a path from main( ) to function 7. Set the parameters and test if the program is correctly in that path. Similarly test all other paths and fix defects.

#### **b) Condition testing**

Test all the logical conditions for both true and false values i.e, we check for both “if” and “else” condition.

**If( condition) - true**

```
{  
    .....  
    .....
```



```
}
```

Else - false

```
{
```

```
.....
```

```
.....
```

```
}
```

The program should work correctly for both conditions i.e, if condition is true, then else should be false and vice-versa

### **c) Loop testing**

Test the loops(for, while, do-while, etc) for all the cycles and also check for terminating condition if working properly and if the size of the condition is sufficient enough.

For ex, let us consider a program where in the developer has given about 1lakh loops.

```
{
```

```
While ( 1,00,000 )
```

```
.....
```

```
.....
```

```
}
```

We cannot test this manually for all 1lakh cycles. So we write a small program,

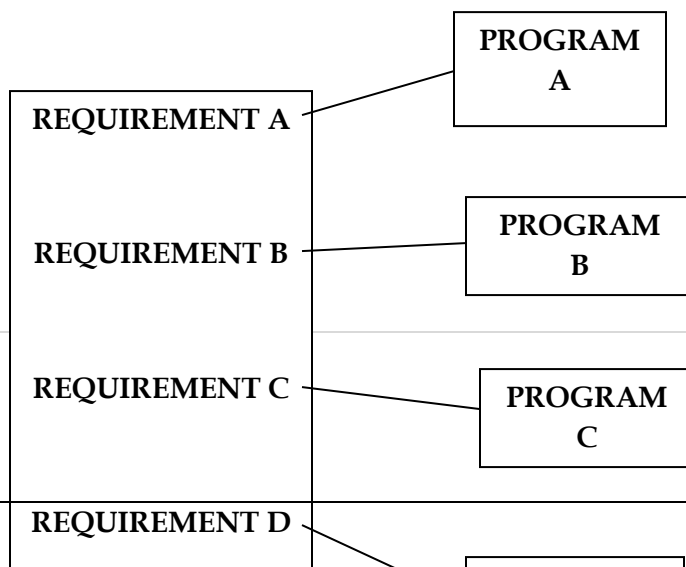
**Test A**

```
{
```

```
.....
```

```
..... }
```

Which checks for all 1lakh loops. This Test A is known as unit test. The test program is written in the same language as the source code program. Developers only write the test program.



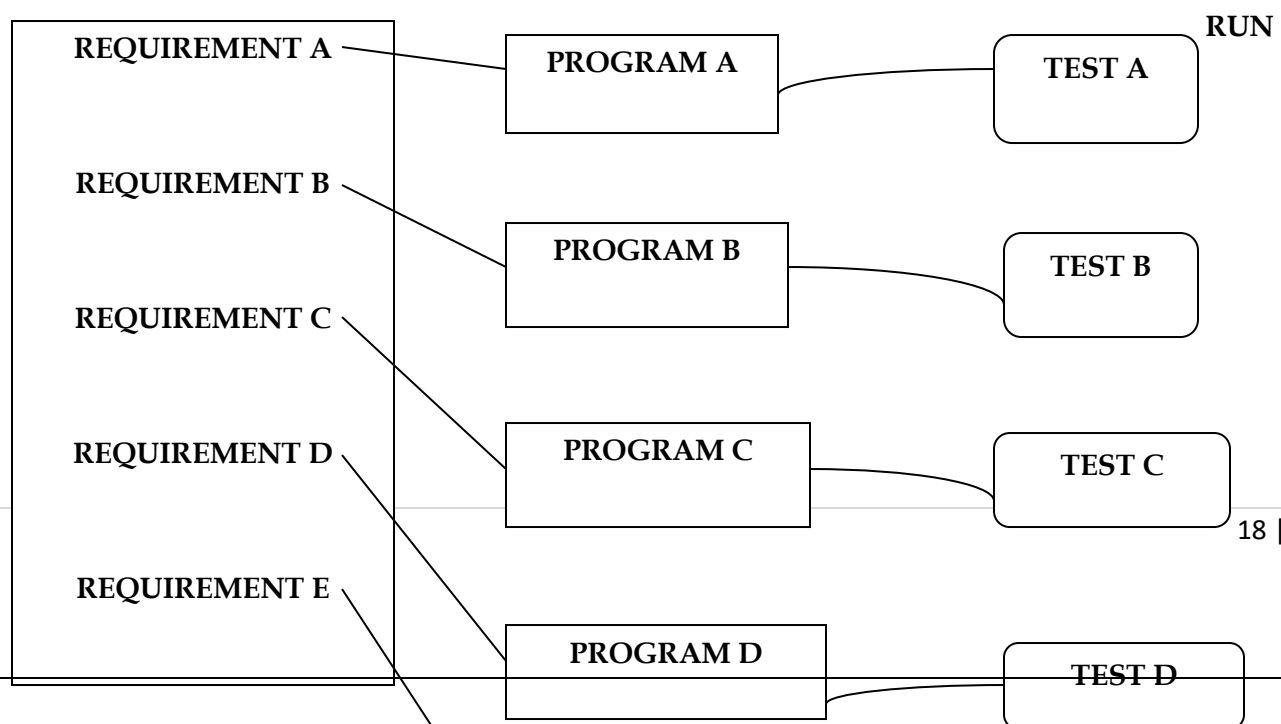
Let us consider the above case as shown in the figure. Suppose the project consists of Requirement A,B,C,D,E. Then the developer writes the Codes A,B,C,D,E for the corresponding requirements. The program consists of 100s of lines of code.

As developers do WBT, they test the 5 programs line by line of code for defects. If in any the program, there is a defect, the developers identify the defect and rectifies it, and they again have to test the program and all the programs again. This involves a lot of time and effort and slows down the project completion time.

Let us consider another case. Suppose the client asks for requirement changes, then the developers have to do the necessary changes and test the entire 5 programs again. This again involves a lot of time and effort.

**This drawback can be corrected in the following manner.**

We write a test program for the corresponding program. The developers write these test programs in the same language as the source code. The developers then run these test programs also called as unit test programs. These test programs connect to the main programs and run as the programs. Thus if there is any requirement change or defects in the program, then the developers simply make the changes either in the test program / main program and run the test program as a whole.



The collection of test programs is known as **test suite**.

As early as we catch the bug, cost of fixing the bug will be less. If we delay in catching the bug, cost of fixing the bug increases exponentially.

Let us consider a program for the calculator. Let us consider the addition function.

CALCULATOR

```
-----  
-----  
-----  
Addition  
{  
    .....  
    .....  
}
```

add.java

```
myAdd ( A, B )  
{  
    C = A + B  
    Print C  
}
```

Test program

```
X = callAdd.myAdd ( 10, 5 )  
If ( X == 15 )  
{  
    Print("test is pass")  
}  
Else  
{  
    Printf("test is fail")  
}
```

The other **test programs** are,

```
Y = callAdd.myAdd (10.5, 5.2)  
If ( Y == 15.7 )  
{  
    Print ("test is pass")  
}  
Else  
{  
    Print("test is fail")  
}
```

Similarly we do for,

Z = 9,00,000 and 1,00,000

P = -10, -5

In all the above test programs, the variables of X, Y, Z, P are calling the function add.java and getting the add. We do the addition manually and check for the remaining results. We can see that the test programs are more tedious and lengthy and time consuming.

Thus, we have readymade programs.

Like, **for example**, we have a ready-made program named **Junit** which has a feature named **Assert** which runs the above and compares the 2 values and automatically prints pass or fail.

Thus we can replace the test programs with these simple programs,

Import Junit

X = callAdd.myAdd(10, 5)

Call Assert (X, 15)

It automatically prints pass or fail. Similarly we can do for the other tests for Y, Z and P.

#### **d) From memory point of view of testing**

The size of code of a program increases because,

**i) The logic** used by the programmer may vary. If one developer writes a code of 300kb file size, then another developer may write the same program using different logic and write of code of 200kb file size.

**ii ) Reuse of code is not there** : for example, if for 5 programs of the same s/w, the 1<sup>st</sup> ten lines of the program is the same. Then we can write these ten lines as a separate function and write a single function call using which the 5programs can access this function. Also, if any defect is there we can simply change the line of code in the function rather than all the programs.

**iii )** Developers declare so many variables, functions which may never be used in any part of the program. Thus, the size of the program increases.

**For ex,**

Int i = 10 ;

Int j = 15 ;

String S = "hello" ;

.....

.....

.....

.....

.....

Int a = j ;

Create user

{

.....

.....

100 lines of code

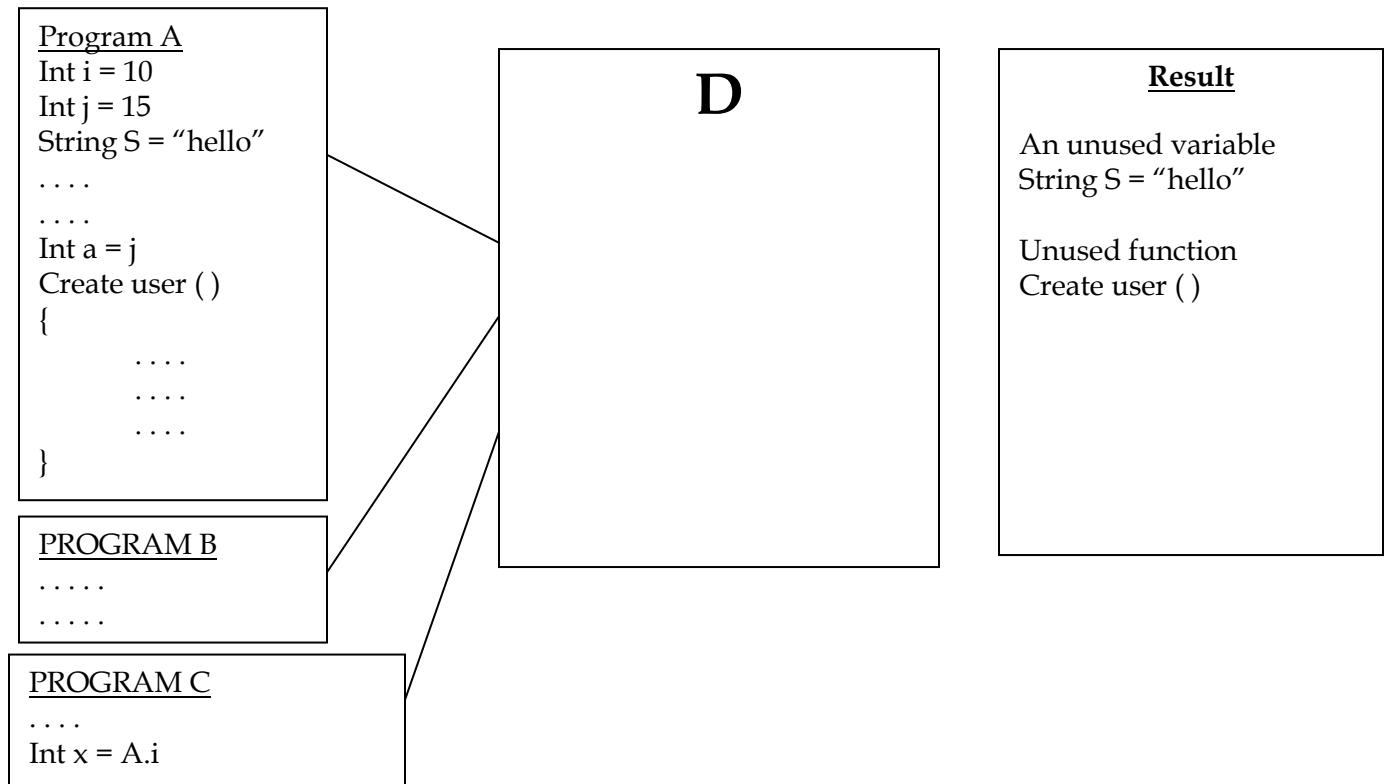
.....

}

In the above program, we can see that the integer i has never been called anywhere in the program and also the function create user has never been called anywhere in the program. Thus this leads to memory wastage.

We cannot recognize this error manually by checking the program because of the large program. Thus we have ready-made tools to check for unnecessary variables and functions.

We have a tool by name **Rational Purify**



Programs A, B, and C is given as input to D. D goes into the programs and checks for unused variables. It then gives the result. The developer can then click on the various results and call or delete the unused variables and functions.

This tool is specific only for C, C++ languages. For other languages, we have other similar tools.

**iv )** The developer doesn't use already existing inbuilt functions and sits and writes the entire function using his logic. Thus leads to waste of time and also delays.

Let us consider there is an already inbuilt function **sort ( )**. Instead the developer sits and writes his own program **mysort ( )**. This leads to waste of time and effort. Instead he could have used a single function call **sort ( )**.

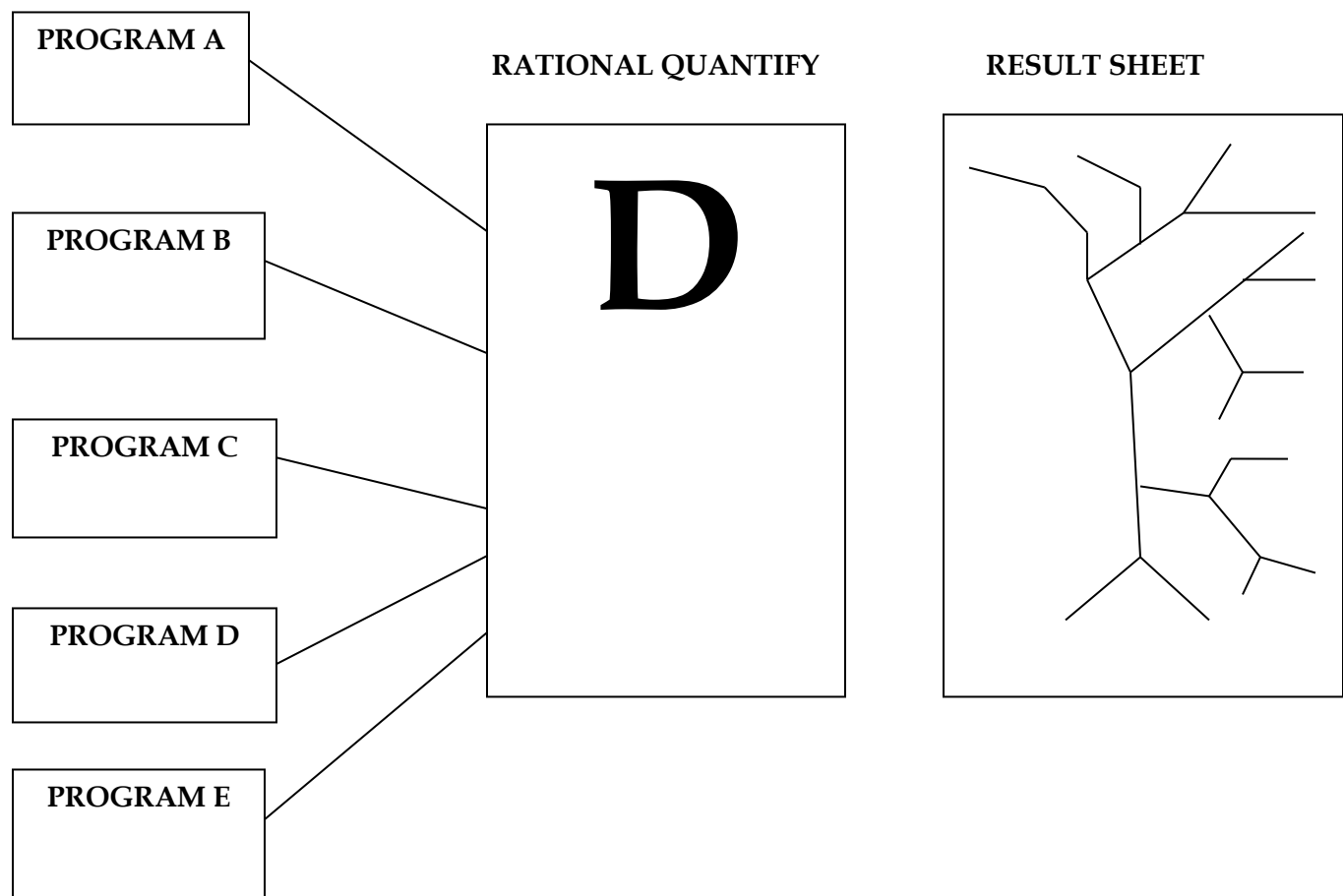
#### **e ) Test for response time/ speed/ performance of the program**

The reasons for slow performance could be,

i ) Logic used

ii ) Switch case – Do not use **nested if**, instead use **switch** case

iii ) Also there must be appropriate use of “**or**” and “**and**” for conditional cases.



As the developer is doing WBT, he sees that the program is running slow or the performance of the program is slow. The developer cannot go manually through the code and check which line of the code is slowing the program.

We have a tool by name **Rational Quantify** to do this job automatically. As soon as all the programs are ready. This tool will go into the program and runs all the programs. The result is shown in the result sheet in the form of thick and thin lines. Thick lines indicate that the piece of code is taking longer time to run. When we double-click on the thick line, then the tool automatically takes us to the line/piece of code which is also colored differently. We can modify that code and again use rational quantify. When the sequence of lines are all thin, we know that the performance of the program has improved.

Developers do a lot of WBT automatically rather than manually because it saves time.

### Difference between White Box Testing and Black Box testing

#### 1) White Box Testing

#### 2) Black Box Testing

a) 1) Done by developers

2) Done by test engineers

- b) 1) Look into the source code and test the logic of the code  
2) Verifying the functionality of the application against requirement specifications
  - c) 1) Should have knowledge of internal design of the code  
2) No need to have knowledge of internal design of the code
  - d) 1) Should have knowledge of programming  
2) No need to have knowledge of programming
- 

## **BLACK BOX TESTING**

It is verifying the functionality ( behavior ) against requirement specifications.

### **Types of Black Box Testing**

#### **1) FUNCTIONAL TESTING**

Also called component testing. Testing each and every component thoroughly (rigorously) against requirement specifications is known as functional testing.

**For ex,** let us consider that Citibank wants a s/w for banking purpose and it asks the company Iflex to develop this s/w. The s/w is something as shown below. When the user clicks his valid user name and enters his password, then he is taken into the homepage. Once inside the homepage, he clicks on amount transfer and the below page is displayed. He enters his valid account number and then the account number to which the money is to be transferred. He then enters the necessary amount and clicks on transfer. The amount must be transferred to the other account number.

Now in black box testing, the test engineer tests the s/w against requirements and checks if the s/w is working correctly as per requirements.

<b>AMOUNT TRANSFER</b>		
Account Balance Amount Transfer Loans Insurance Transactions Logout	From Account Number	<input style="width: 90%;" type="text"/>
	To Account Number	<input style="width: 90%;" type="text"/>
	Amount	<input style="width: 90%;" type="text"/>
	<div style="display: flex; justify-content: space-around; margin-top: 10px;"><div style="border: 1px solid black; padding: 5px 20px; cursor: pointer;"><b>TRANSFER</b></div><div style="border: 1px solid black; padding: 5px 20px; cursor: pointer;"><b>CANCEL</b></div></div>	

**This is how the requirements given by the client looks like** (figure below). It is usually a word document file. Let us consider that Citibank gives a 80pg SRS in MS-WORD format. The test engineer then looks at the requirements and correspondingly checks the s/w.

Now the test engineer does all possible tests on the 2 account numbers. Now, he proceeds with the testing of Amount transfer. These are the following tests he conducts for testing the amount field,

He enters the following data in the amount field,

- |                        |   |                      |   |          |   |
|------------------------|---|----------------------|---|----------|---|
| a) - 100               | X | b) 100\$             | X | c)100.50 | X |
| d) Hundred rupees only | X | e) 100 blank space 0 |   | X        |   |
| f) 100                 |   | g)0.001              | X |          |   |

For all the above cases except for **f)** , it should throw an error message. If it doesn't throw, then there is a bug in the s/w and the s/w must be sent to the development team to repair the defect.

## CITIBANK ONLINE - SRS

### 1. LOGIN

**1.1 Username** : should accept only 8 – 22 characters

**1.2 Password** : should accept only 8 – 36 characters. Special characters are allowed.

**1.3 Forgot Password** :.....

1.3.1 .....

1.3.2 .....

**1.4 Registration** :.....

1.4.1 .....

1.4.2 .....

### 2. LOANS

**2.1 Personal Loan** :.....

2.1.1 .....

2.1.2 .....

**2.2 Home Loan** :.....

2.2.1 .....

2.2.2 .....

### 3. INSURANCE

3.1 .....

3.2 .....

SCROLL



**Thus, during testing, we must remember the following points,**

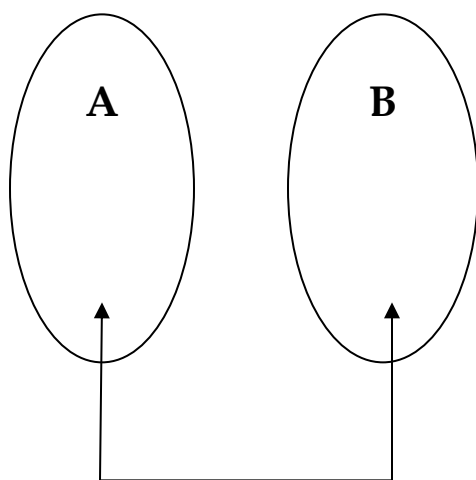
- a) We must always start testing the application with the valid data. In the above example for amount transfer, we see that we have entered the valid data 100 only in the 6<sup>th</sup> test. This should not be done, because if the valid data itself is not taken correctly, then we need not have to waste our time checking for the invalid data
- b) If the application works for valid data, only then we must start testing for invalid data
- c) If the application is not working for 1 of the invalid values, then we can continue testing for all the other invalid values and then submit the test report of all the defects for invalid values.
- d) In testing, we should not assume or propose requirement. If we have any queries, talk to the one who knows the requirements very well and clarify the queries.
- e) We must not do **over-testing** ( testing for all possible junk values ) or **under-testing** ( testing for only 1 or 2 values ). We must only try and do **optimize testing** (testing for only the necessary values- both invalid and valid data).
- f) We must do both **positive testing** (testing for valid data) and **negative testing** (testing for invalid data).

**The characteristics of a good requirement are,**

- 1) **Unitary ( cohesive )** - the requirement addresses 1 and only 1 thing
- 2) **Complete** - the requirement is fully stated in 1 place with no missing information
- 3) **Consistent** - the requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation
- 4) **Non-Conjugated ( Atomic )** - the requirement is atomic i.e, it does not contain certain conjunctions. Ex - "the postal code field must validate American and Canadian postal codes" should be written as two separate requirements : 1) "The postal code field must validate American Postal codes" and 2) "The postal code field must validate Canadian Postal codes".
- 5) **Traceable** - the requirement meets all or part of a business need as stated by stakeholders and authoritatively documented
- 6) **Current** - the requirement has not been made obsolete by the passage of time
- 7) **Unambiguous** - the requirement is concisely stated without recourse to technical jargon, acronyms etc. it expresses objective facts, not subjective opinions. It is subjective to one and only one interpretation.
- 8) **Mandatory** - the requirement represents a stakeholder defined characteristic the absence of which will result in a deficiency that cannot be ameliorated
- 9) **Verifiable** - the implementation of the requirement can be determined through one of 4 possible methods - inspection, demonstration, test or analysis.

## 2) INTEGRATION TESTING

Testing the data flow or interface between two features is known as integration testing.



Take 2 features A & B. Send some data from A to B. Check if A is sending data and also check if B is receiving data.

Now let us consider the example of banking s/w as shown in the figure above ( amount transfer ).

**Scenario 1** - Login as A to amount transfer - send 100rs amount - message should be displayed saying 'amount transfer successful' - now logout as A and login as B - go to amount balance and check balance - balance is increased by 100rs - thus integration test is successful.

**Scenario 2** - also we check if amount balance has decreased by 100rs in A

**Scenario 3** - click on transactions - in A and B, message should be displayed regarding the data and time of amount transfer

**Thus in Integration Testing, we must remember the following points,**

- 1) Understand the application thoroughly i.e, understand how each and every feature works. Also understand how each and every feature are related or linked to each other.
- 2) Identify all possible scenarios
- 3) Prioritize all the scenarios for execution
- 4) Test all the scenarios
- 5) If you find defects, communicate defect report to developers
- 6) Do positive and negative integration testing. **Positive** - if there is total balance of 10,000 - send 1000rs and see if amount transfer works fine - if it does, then test is pass. **Negative** - if there is total balance of 10,000 - send 15000rs and see if amount transfer happens - if it doesn't happen, test is pass - if it happens, then there is a bug in the program and send it to development team for repairing defects.

INBOX

COMPOSE MAIL

SENT ITEMS

TRASH

SPAM

CONTACTS

FOLDERS

LOGOUT

COMPOSE MAIL

TO

FROM

SUBJECT

TEXT FIELD

☐ SAVE TO DRAFTS

☐ ADD TO CONTACTS

SEND

CANCEL

Let us consider gmail software as shown below. We first do **functional testing** for username and password and submit and cancel button. Then we do **integration testing** for the above. The following scenarios can be considered,

**Scenario 1** - Login as A and click on **compose mail**. We then do **functional testing** for the individual fields. Now we click on **send** and also check for **save drafts**. After we send mail to B, we should check in the **sent items** folder of A to see if the sent mail is there. Now we logout as A and login as B. Go to **inbox** and check if the mail has arrived.

**Scenario 2** - we also do **integration testing** for **spam** folders. If the particular contact has been marked as spam, then any mail sent by that user should go to **spam** folder and not to the **inbox**.

We also do **functional testing** for each and every feature like - **inbox,sent items etc** .

ADD USER

DELETE USER

LIST USERS

EDIT USERS

PRODUCT SALES

PRODUCT PURCHASES

SEARCH USERS

HELP

ADD USERS

USERNAME

PASSWORD

DESIGNATION

EMAIL

TELEPHONE

ADDRESS

Team lead

Manager

.....

.....

SUBMIT

CANCEL

e

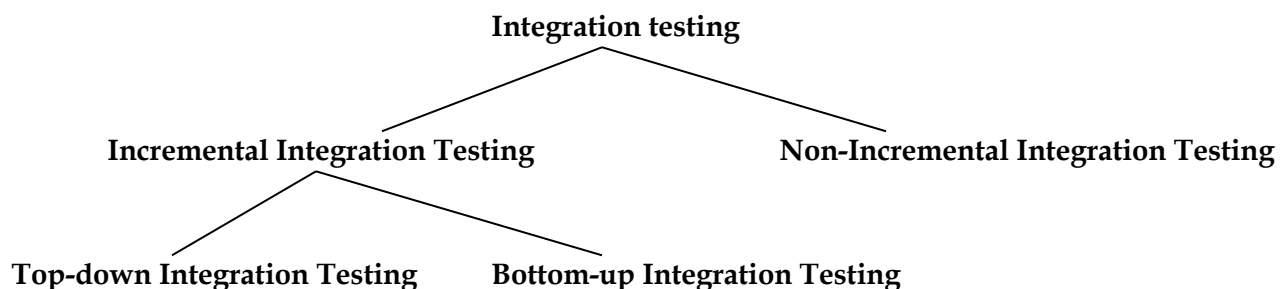
Let us consider the figure shown above.

We first do **functional testing** for all the text fields and each and every feature. Then we do **integration testing** for the related features. We first test for **add user and list user and delete user and then edit user and also search user**.

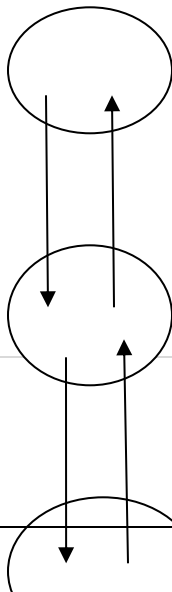
**Points to remember,**

- 1) There are features we might be doing only doing functional testing and there are features we might be doing both integration and functional testing. It depends on features.
- 2) Prioritizing is very important and we should do it at all the stages which means – open the application and decide which feature to be tested first. Go to that feature and decide which component must be tested first. Go to that component and decide what value to be entered first. Don't apply same rule everywhere!! Testing logic changes from feature to feature.
- 3) Focus is important i.e, completely test 1 feature and then only move onto another feature.
- 4) Between 2 features, we might be doing only positive integration testing or we might be doing both positive and negative integration testing. It depends on the feature.

There are **two types** of **integration testing**,



**Incremental Integration Testing :**



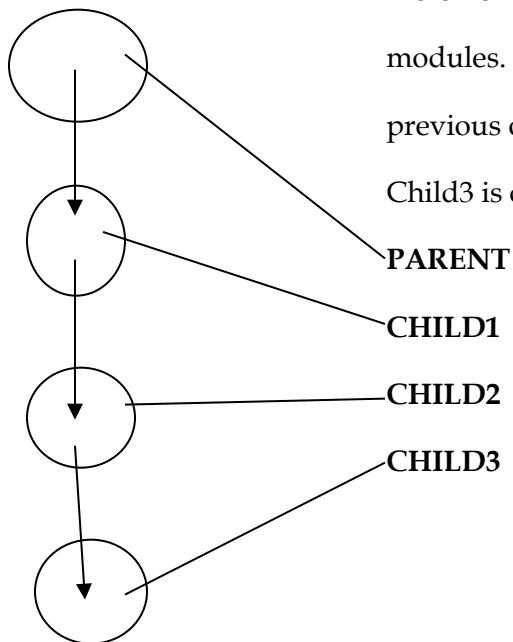
Take two modules. Check if data flow between the two is working fine. If it is, then add one more module and test again. Continue like this. Incrementally add the modules and test the data flow between the modules.

There are **two** ways,

a) Top-down Incremental Integration Testing

b) Bottom - up Incremental Integration Testing

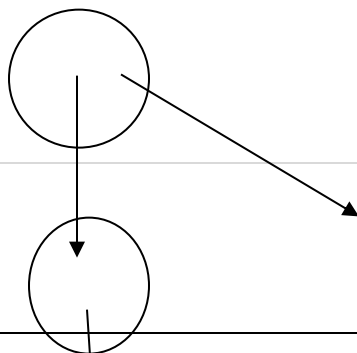
### Top-down Integration Testing :



Incrementally add the modules and test the data flow between the modules. Make sure that the module that we are adding is child of previous one.

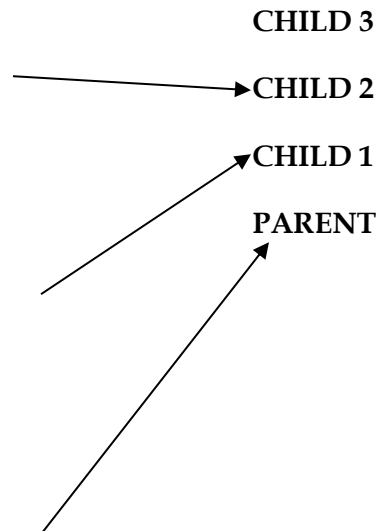
Child3 is child of child2 and so on.

### Bottom-up Integration Testing :



Testing starts from last child upto parent. Incrementally add the modules and test the data flow between modules. Make sure that

the module you are adding is the parent of the previous one.

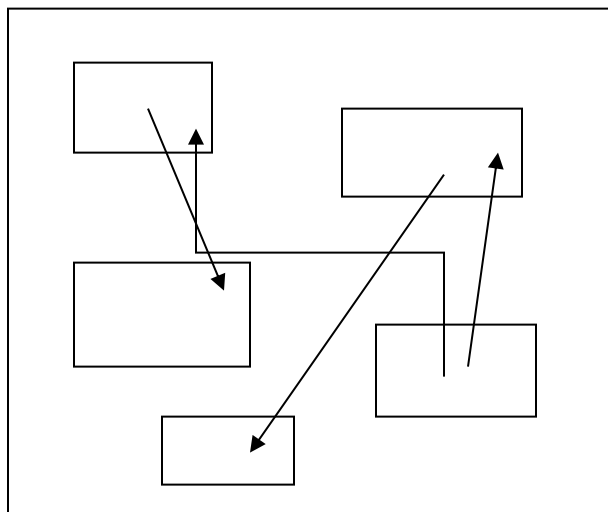


### Non - incremental Integration Testing

We use this method when,

- a) When data flow is very complex
- b) When it is difficult to identify who is parent and who is child.

It is also called **Big - Bang method**.



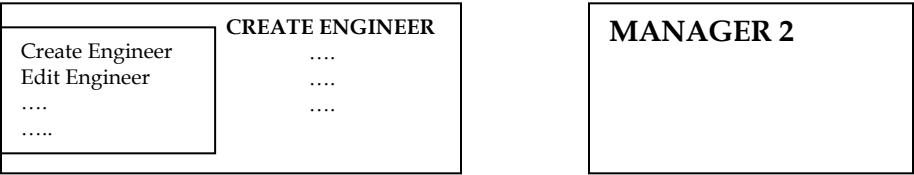
Combine all the modules at a shot and start testing the data flow between the modules. The disadvantage of this is that, **a)** We may miss to test some of the interfaces **b)** Root cause analysis of the defect is difficult - identifying the bug where it came from is a problem. We don't know the origin of the bug.

### **Example for Incremental Integration Testing :**

CEO

CREATE MANAGER	
Create manager	User name <input type="text"/>
Edit manager	Password <input type="text"/>
List manager	Email <input type="text"/>
Delete manager	
.....	
.....	
<input type="button" value="SUBMIT"/> <input type="button" value="CANCEL"/>	

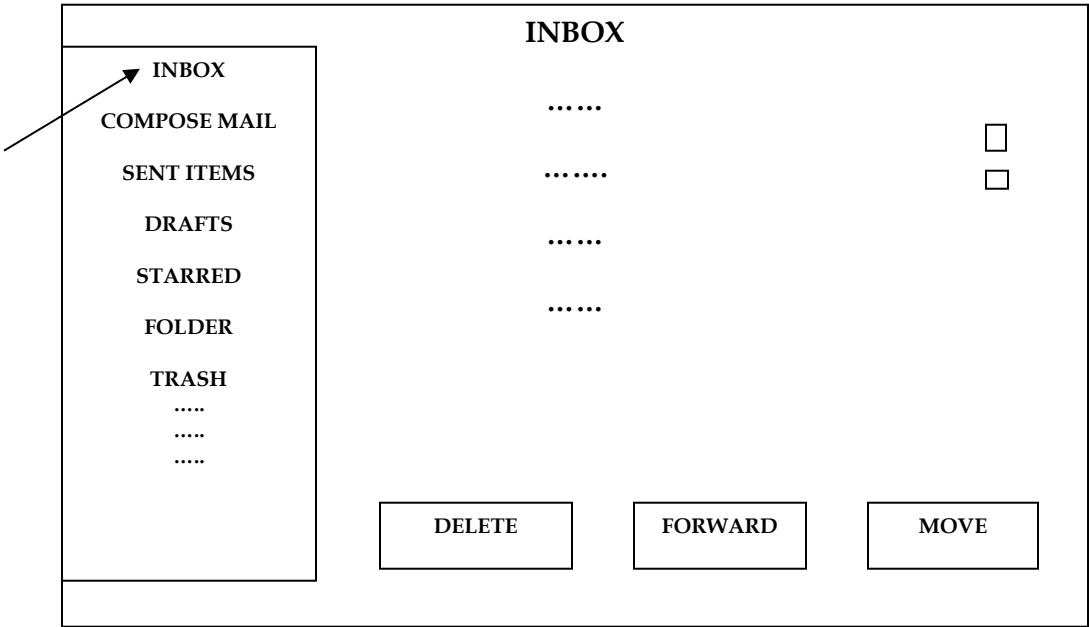
MANAGER ←



In the above example. The development team develops the s/w and send it to the CEO of the testing team. The CEO then logs onto the s/w and creates the username and password and send a mail to a manager and tells him to start testing the s/w. The manager then edits the username and password and creates an username and password and send it to the engineer for testing. This hierarchy from CEO to Testing Engineer is **top-down incremental integration testing**.

Similarly, the testing engineer once he finishes testing sends a report to the manager, who then sends a report to the CEO. This is known as **bottom-up incremental integration testing**.

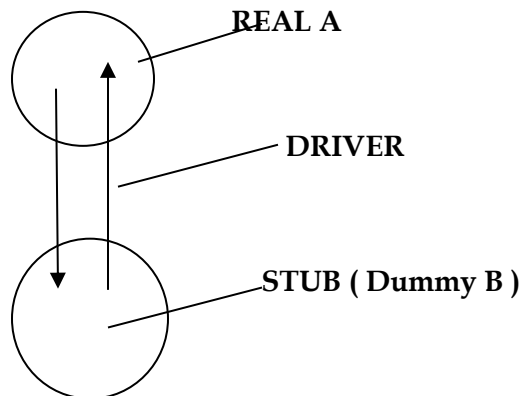
**Example for non-incremental Integration testing**





The above example displays a homepage of a gmail inbox. When we click on inbox link, we are transferred to the inbox page. Here we have to do **non-incremental integration testing** because there is no parent and child process here.

### Stub and Driver



Stub is a dummy module which just receives data and generates a whole lot of expected data, but it behaves like a real module. When a data is sent from real module A to stub B, then B just accepts the data without validating and verifying the data and it generates expected results for the given data.

The function of a **driver** is it checks the data from A and sends it to stub and also checks the expected data from stub and sends it to A. **Driver** is one which sets up the test environment and takes care of communications, analyses results and sends the report. We never use stubs and drivers in testing.

In **WBT**, **bottom-up integration testing** is preferred because writing drivers is easy. In **black-box testing**, no preference and depends on the application.

### *INTERVIEW TIPS and QUESTIONS*

*1) In interview, they'll ask – Which is the most preferred method of testing and give 4 options –*

- a) Functional Testing   b) White-box testing   c) Top-down integration testing  
d) Bottom – up Integration testing*

*Ans) Always write Bottom-up testing – unless asked specifically for anything else*

*2) In interview, they'll ask – When does testing start ?*

*Ans) always tell, testing starts as soon as the requirements are handed over – coz the interviewers always have V&V model in mind – if any specified model is asked, then answer according to that model.*

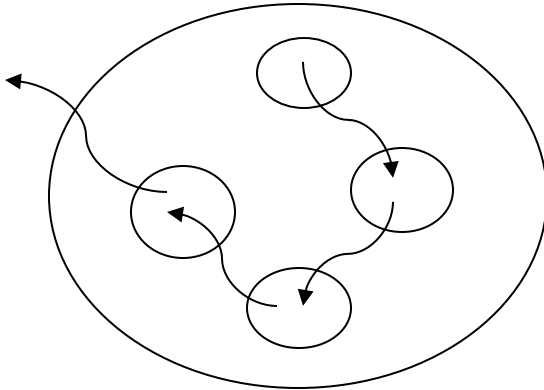
*3) In interview, they'll ask – I have 2 modules A and B – A has been built, B is yet to be built – but i need to test B, how can I do Integration Testing for A and B ?*

*Ans) we create a dummy module B also known as Stub, and then tell about stubs. If they ask have you ever written a stub – then tell – stubs are very rarely used and that you just know about the concept through the internet.*

### 3) SYSTEM TESTING

It is **end-to-end testing** wherein **testing environment is similar to the production environment**.

#### End - to - end testing



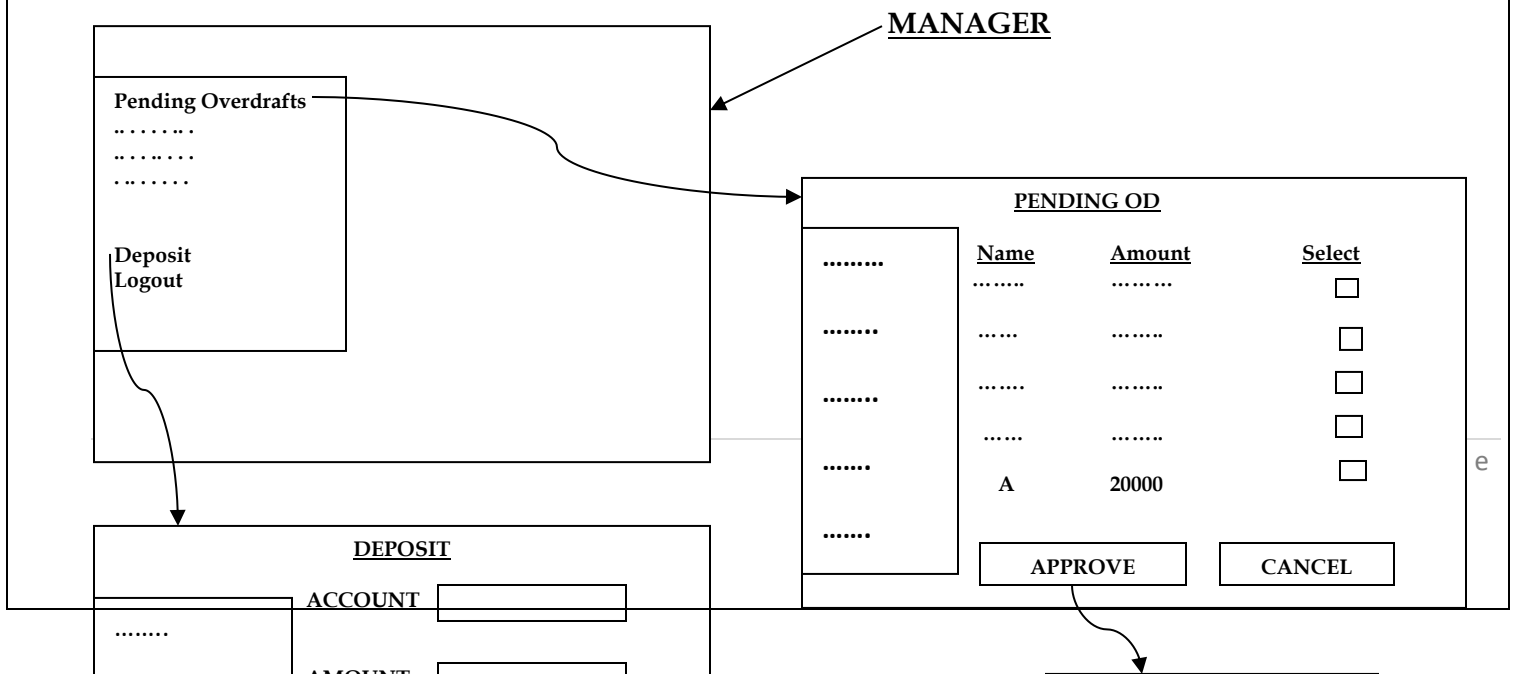
Here, we navigate through all the features of the software and test if the end business / end feature works. We just test the end feature and don't check for data flow or do functional testing and all.

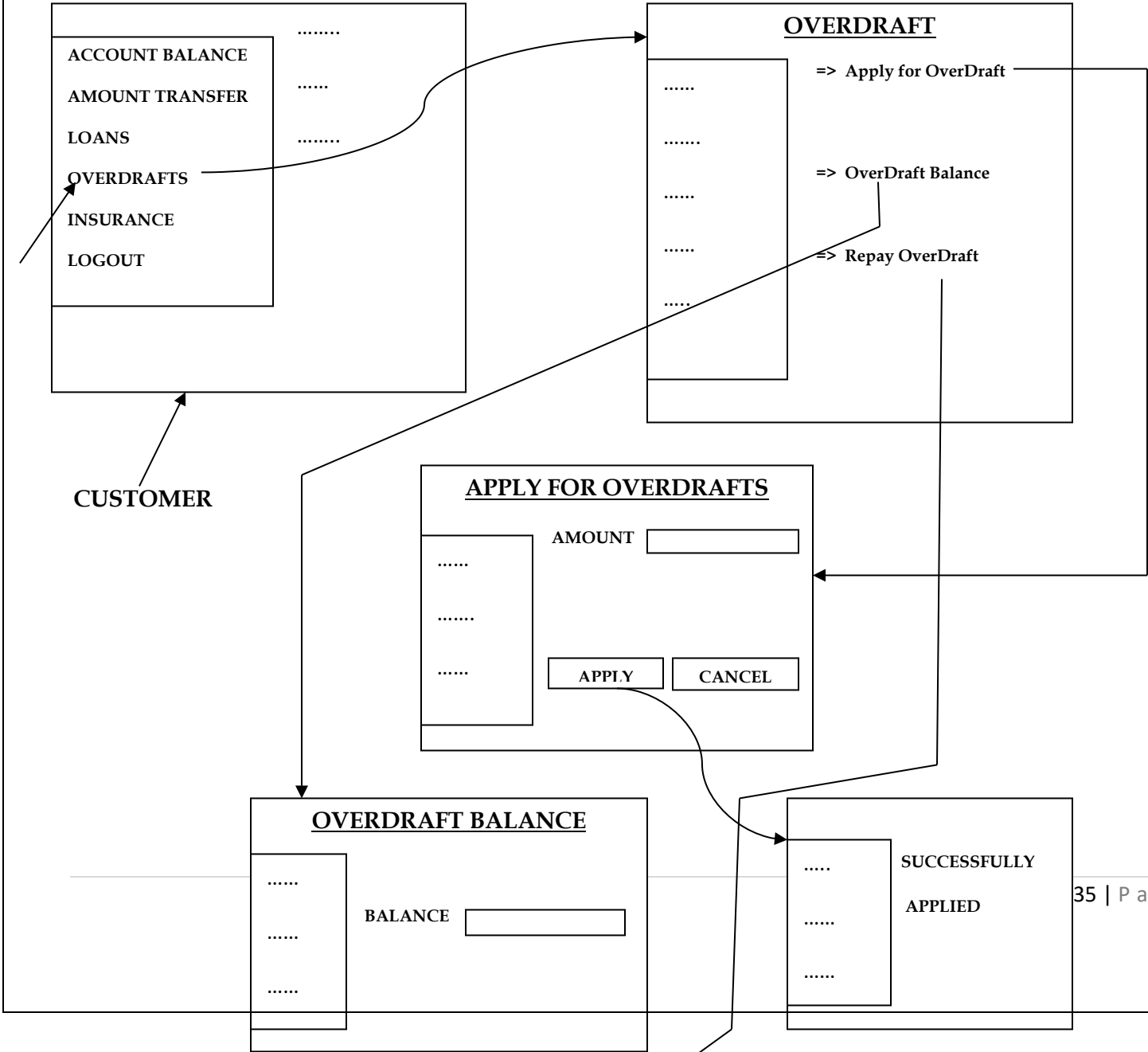
**Let us consider an example to explain System Testing.**

Let us consider Citibank wants a software for overdraft feature. It asks IFlex company to develop the software and it provides CRS to develop the feature. The CRS contains how the overdraft feature works,

The difference between personal loan and overdrafts is – personal loans, loans can be provided upto 20 times more than the monthly income and also takes a long time to approve by the manager for personal loan. Whereas, in Overdraft, the loan amount is twice the monthly income and takes hardly a day to be approved by the manager. For ex, if a customer of Citibank wants a overdraft loan of Rs 20,000 over his monthly income of Rs 10,000. The manager approves the loan. Let us say that the interest rate is 2% and the activation fee for the first time is Rs 250. When the customer repays the loan at the end of the month, then the total amount he pays is –  $20,000 + (2\% \text{ of } 20,000) + \text{Activation fee } (250) = 20,000 + 400 + 250 = 20,650\text{Rs}$ . Now, for the 2<sup>nd</sup> time if the same customer wants another overdraft loan – then no activation fee is taken. Now the customer applies for another overdraft loan of Rs20,000. This time the amount he has to repay is –  $20,000 + (2\% \text{ of } 20,000) = 20,400\text{Rs}$ .

The development team develops the software which looks something like this, (Shown in the next page).





The development team develops the required software as shown above. The first figure represents the software that can be accessed by the manager only. The 2<sup>nd</sup> figure represents the software that can be accessed by the bank's customers.

Let us consider system testing now. We test for interest calculation when the customer takes overdrafts for the 1<sup>st</sup> time and when he takes overdrafts for the 2<sup>nd</sup> time.

#### **Scenario 1**

- 1) Login as A – Apply for OD Rs 20000 – Click on Apply – Logout
- 2) Login as manager – Approve OD of A – Logout
- 3) Login as A – Check OD Balance – Rs 20000 should be deposited – Logout
- 4) Change the server date to next 30days
- 5) Login as A – Check OD Balance –  $20000 + 400 + 250 = 20650$  – Logout
- 6) Login as manager – click on Deposit – Deposit Rs 650 – Logout
- 7) Login as A – Repay OD amount – Check OD balance – Rs 0
- 8) Login as manager – Click on Deposit – Deposit Rs 20000 to A's account - logout
- 9) Login as A – Apply for OD Rs 20000 – Click on Apply – Logout
- 10) Login as manager – Approve OD of A – Logout
- 11) Login as A – Check OD Balance – Rs 20000 should be deposited – Logout
- 12) Change the server date to next 30days
- 13) Login as A – Check OD Balance –  $20000 + 400 = 20400$  – Logout

14) Login as manager – Deposit 400 – logout

15) Login as A – repay OD amount – Check OD balance – Rs 0

**Scenario 2** – now we test another scenario where in – let us consider that the bank gives an offer that states that – a customer who takes Rs 50000 as OD for the first time will not be charged activation fee and activation fee will not be refunded when he takes another OD for the 3<sup>rd</sup> time – we have to test for 3 test scenarios – wherein we have to take OD of Rs 50000 for the first time and check for OD Repay Balance after applying for another OD for 3<sup>rd</sup> time.

**Scenario 3** – now we take in other scenario – let us consider that the software is being used normally by all customers – suddenly Citibank decides to lower the Activation fee to Rs 125 for new customers – we have to test OD for new customers and see if its accepting only Rs 125.

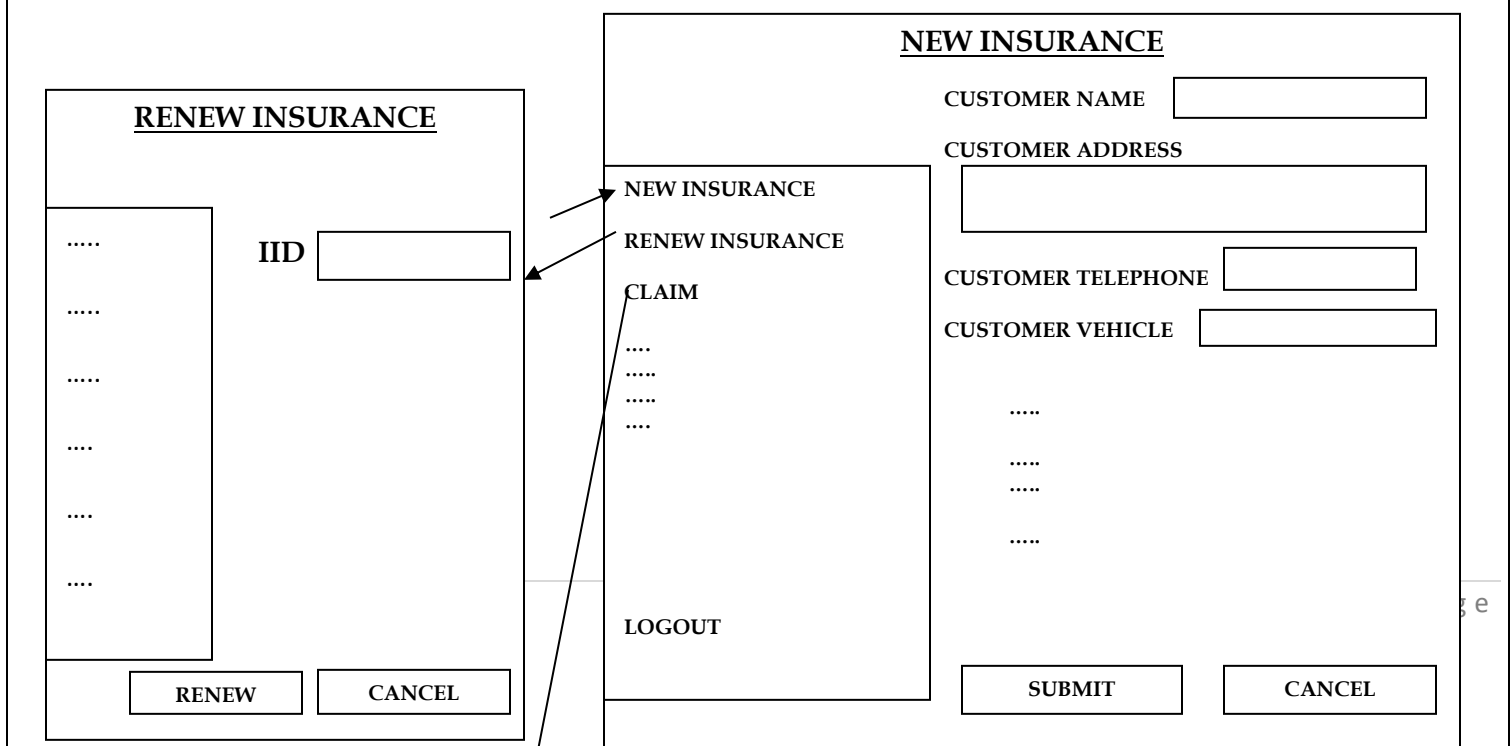
But, then we get a requirement conflict – Suppose the customer has applied for Rs 20000 as OD with the existing Activation Fee for Rs 250. Before the manager is yet to approve it, the bank lowers the activation fee to Rs 125. Now we have to test what Activation Fee is charged for the OD of the Pending customer – In this case, the testing team cannot assume anything – they have to contact the Business Analyst or the Client and find out what they want in such a case.

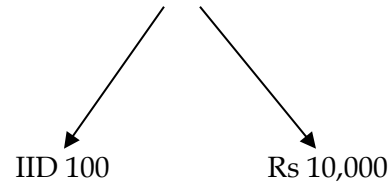
Thus, if they ( Client ) give 1 set of requirement, we must come up with maximum possible scenarios.

Let us consider another **example – insurance domain**

When we buy a car, we have to obtain an insurance on that. Let us consider that Bajaj Allianz Insurance company is the one which provides this car insurance. The insurance policy works in this way – for the 1<sup>st</sup> year when the car is bought, the insurance to be paid is Rs10000. For the 2<sup>nd</sup> year, the insurance must be renewed at Rs10000 again. For the 3<sup>rd</sup> year, if no claim has been made, then that customer is offered a discount of Rs1500 and the he must have to renew the insurance at Rs8500 only. If an insurance claim has been made, then the insurance must be renewed at Rs10000 only. Bajaj Allianz Insurance wants the s/w to be developed which works as above. Thus it gives Wipro the CRS of above, to develop the s/w.

The development team develops the s/w as shown below,





The above s/w works in this way. When the insurance agent logs in to the home page, he clicks on New Insurance and creates a new insurance policy for the new customer and fills up all the details and the new customer is assigned a IID (Insurance ID) 100. He then pays the insurance amount Rs10,000. After 1year, when the time has come for renewal, then the agent logs in and clicks on Renew Insurance and enters the IID and renews insurance for Rs 10,000. The different test scenarios for the above are,

**Scenario 1 :**

- 1) Login as Agent – click on New Insurance – Create IID 100 and Amount Rs10000
- 2) Change server date by 1year
- 3) Login as agent – Click on Renew Insurance – and pay amount Rs 10,000
- 4) Change server date to 1year
- 5) Login as Agent – Renew Insurance – IID 100 – Insurance Amount must be Rs 8500 since no claim has been made – thus the test is pass

**Scenario 2 :**

**Same as 1) , 2) , 3)**

- 4) Before you renew for the 3<sup>rd</sup> year, Claim Insurance – Change server date to 1year
- 5) Login as Agent – Renew IID 100 – Rs 10000 – No discount should be made because of the claim made above.

**Scenario 3 :**

**Same as 1), 2), 3) and 4) of 1<sup>st</sup> scenario**

- 5) Before you renew 3<sup>rd</sup> time, - Click on Claim IID 100 , Rs 15,000 – Try to claim the amount – We shouldn't be able to claim this because the insurance has expired.

Let us consider another **example – Advertisements on the Internet**

When we open a website, say [www.yahoo.com](http://www.yahoo.com) , we see an ad posted on the above – top of the homepage – it remains there for a few seconds before it disappears – this management of ads is done by something known as AMS – Advertisement Management System. Now we do s/w testing for this domain.



The various **test scenarios** are,

**Scenario 1 :** the first case is the normal scenario as explained above. The test engineer does end-to-end testing for the normal scenario. Wherein the Nokia manager makes the request for the Ad and the Ad is deployed at the concerned date and time.

**Scenario 2 :** let us consider a scenario where-in the Nokia manager feels that the Ad space is too costly and cancels the request. At the same time, Coca-Cola makes a request for the Ad space on Feb 14<sup>th</sup> at 9AM. Since the request of Nokia has been cancelled, thus Coca-Cola's Ad must be deployed On Feb 14<sup>th</sup> at 9AM after all the request and the payment has been made. Now, if there is a change in heart from Nokia and they feel that they are ready to make the payment for Feb 14<sup>th</sup> at 9AM – that slot should not be given because Coca-Cola has already utilized that space. Thus , an alternative calendar must open up for Nokia to make their booking.

**Scenario 3 :** Login as AMS manager – Click on Set Rates – and set the rate for Ad space on logout page to 5\$ per second. Login as Nokia manager – and choose the date and time to put up an Ad on the logout page. The payment should be 50\$ for 10seconds for an Ad on Yahoo!mail's Logout page.

**Thus, the actual definition of End-to-End testing can be given as –** Take all possible end-to-end business flows and check whether in the software, all the scenarios are working or not. If it is working, then the product is ready to be launched.

### **Testing Environment and Why it should be similar to Production Environment ?**

After the requirements have been collected and the design of the s/w has been developed, the CRS is then given to the development team for coding and building of the modules and the s/w. the development team stores all the modules and the code it builds in a development server which they name it REX (any name can be given to the server).

The development team builds module A of the s/w – does WBT – installs the s/w at <http://qa.citibank.com> - zips the code of module A and stores it in REX – the team lead of the development



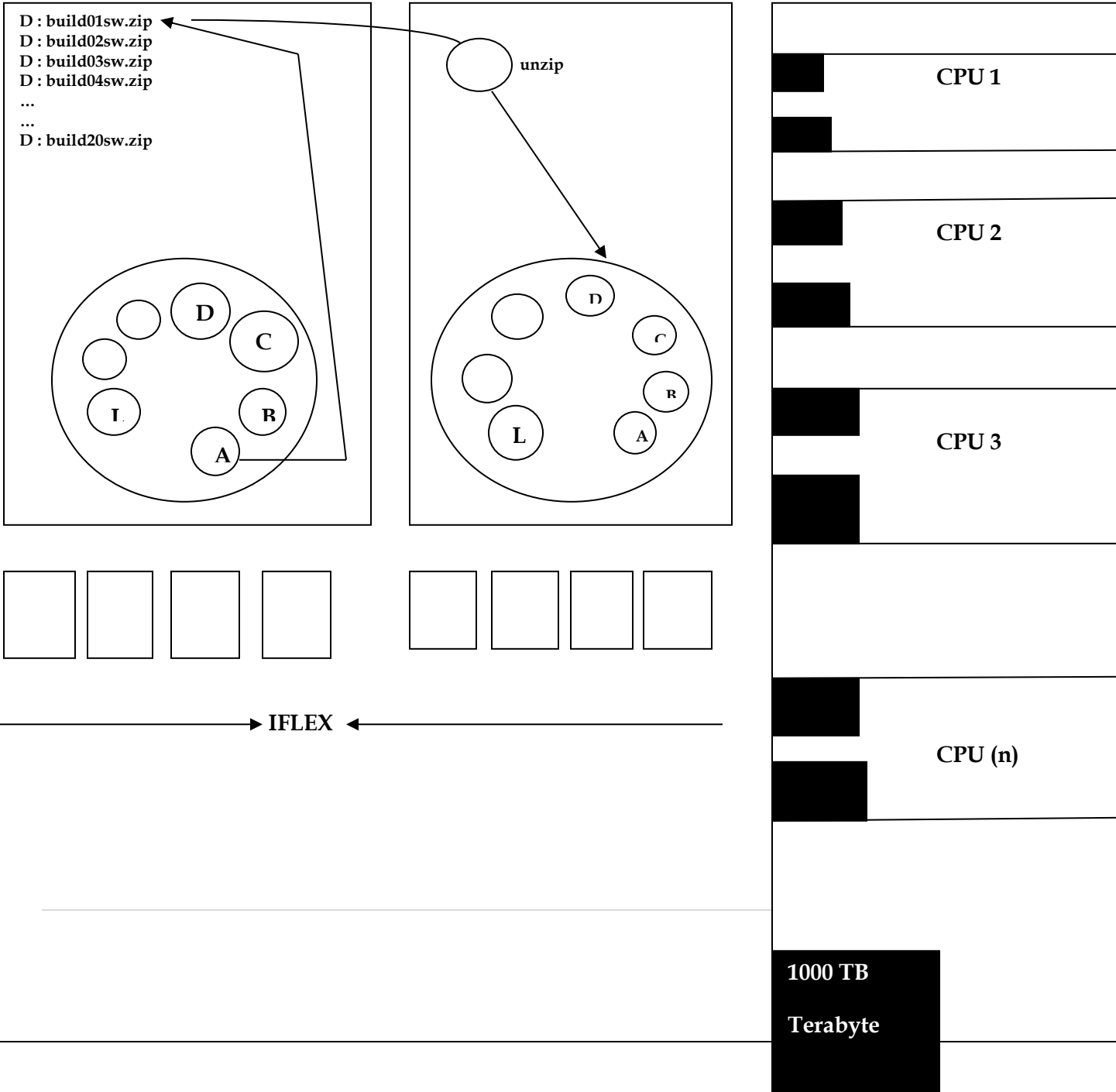
team then emails the zip file of module A to the test lead and tells him that the module A has been built and WBT has been performed and that they can start testing the module A – the test lead first unzips the module A and installs it in the testing team server named QA - the test lead then calls in the test engineers in his team and assigns them different parts of the module A for testing – this is the **first cycle** – the testing team do functional testing on A – let’s say the testing team finds 100bugs in module A – for each bug found, the testing team prepares a report on the bug in a Word document file and each bug is assigned a number – like this, the testing team finds 100bugs in the s/w – each test engineer when he finds a bug, he immediately emails bug report to the development team for defect repair – the testing team take 5days to test module A.

The developers are reading the defect reports, goes through the code, fixes the problem – when testing team is testing s/w, the developers are fixing defects and also preparing another module and also doing WBT for the repaired program – now the developers fix majority of the defects(say 70) and also build module B – now the team lead of the development team installs the s/w at the above website, zips the code of the module B and sends a mail to the test lead containing the code – the test lead first uninstalls the old s/w and installs the new one containing module B and also repaired module A – and sends a mail to the test engineers in his team containing the new module and repaired module –

DEVELOPMENT SERVER ( REX )

TESTING SERVER ( QA )

PRODUCTION SERVER



**Whenever a new build comes in, the testing team concentrates on testing the new feature first – because the probability of finding the bugs is more, we expect more number of bugs in the new feature – as soon as new build comes in,**

- a) test new features    b) Do integration testing
- c) retest all the fixed defects    d) test unchanged(old) feature to make sure that it is not broken
- e) in the new build, we retest only fixed defects    f) each test engineer retests only his bugs which are fixed, he is not responsible for other bugs found by other test engineers.

**We find new bugs in old feature because –** a) fixing the bugs may lead to other bugs

- b) adding new features (modules)    c) might have missed it in the earlier test cycle

In the **second cycle** – we do both functional and integration testing for A and B – we find 80 bugs – each bug is sent in a report of Word format – the developers repair about 40bugs and also repair 5bugs of the remaining 30bugs in the first test cycle.

Like this we carry on, and do about 20cycles and reach a stage wherein the developers are developing the 20<sup>th</sup> build, say module L – now the testing team gets a server which is similar to the production server (real –time server on which the s/w will run at the client’s place) – and install the s/w there – and they start off with system testing.

**We start System Testing –** a) when the minimum number of features are ready    b) basic functionality of all the modules must be working    c) testing environment should be similar to production environment

**We say that the product is ready for release when,**

- a) all the features requested by customer are ready
- b) when all the functionality, integration and end-to-end scenarios are working fine
- c) when there are no critical bugs
- d) bugs are there, but all are minor and less number of bugs
- e) by this time, we would have met the deadline or release date is very near.

**The entire period right from collecting requirements to delivering the s/w to the client is known as release. Each time a new module is built and old module is tested is known as Build – testing. Each build takes about 5days or more or less.**

**In interview, when they ask how many builds have u tested – optimum answer would be 26 – 30 builds.**

**The testing environment should be similar to production environment means,**

**1) The hardware should be similar to production - a)** The make ( manufactured by ) should be similar to production server ( **for ex,** if the production server is HP, then test server should also be HP server ) **b)** configuration and make must be similar, but different capacities i.e, number of CPUs ).

**2) The software should be similar to production - a)** The OS should be similar **b)** Application server should be similar **c)** Web server should be similar **d)** Database server should be similar

**3) Data should be similar to production - a)** We should create data similar to production **b)** We should create a script to create a dummy data which is similar to production environment .

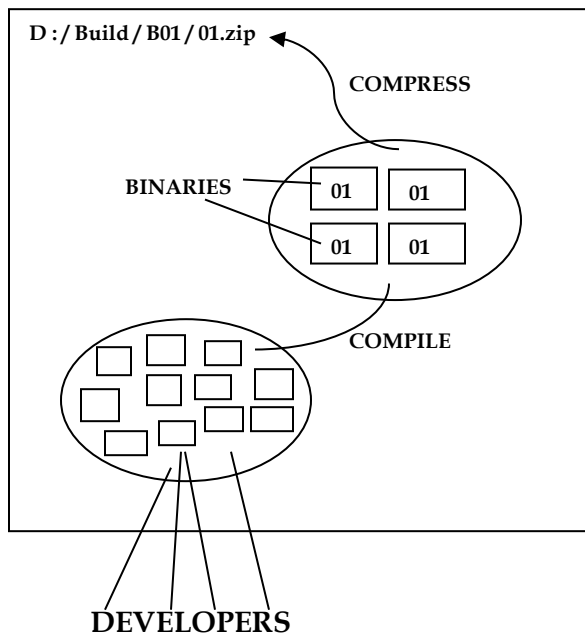
ex -     while 20000  
          create Username  
          create password into table customer  
          Run

In real time environment, we may make lakhs of entries into database. But, while testing we can't enter manually lakhs of entries, so we write a test script program which generates thousands of entire and thus can be used for testing.

**In Testing environment, who is involved in installing the software ?**

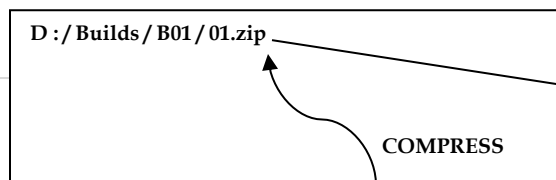
- Test engineer ( anybody from testing team )
- Anybody from development team
- Release engineer / Build engineer

**Build -** Build is a piece of software which is copied, unzipped and installed at the testing server.

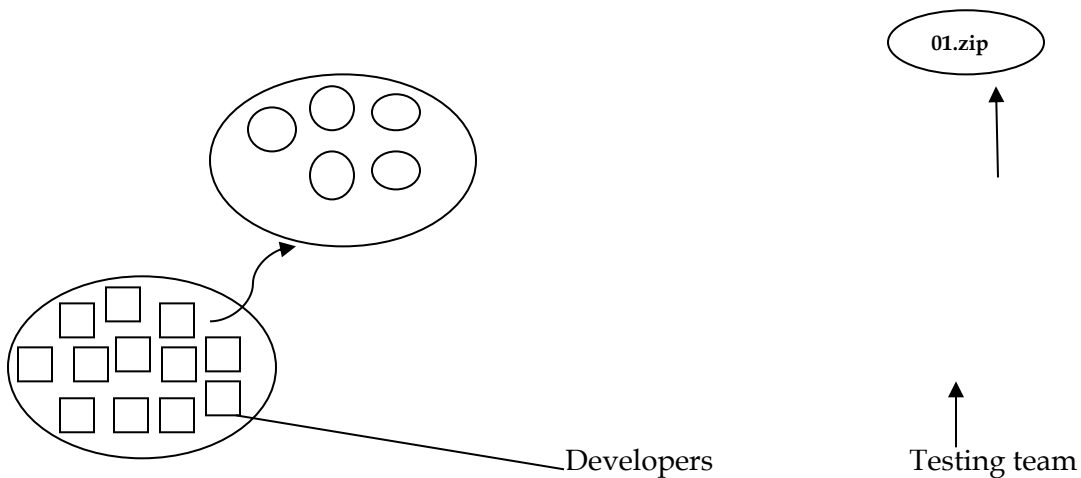


All the programs will be compiled and then compressed ( compressed file should be in zip, rar, war, gunzip, tar, jar ) format and compressed file is called build which is copied and pasted in the test environment, installed and we start testing the software.

**CASE 1 : Test Engineer is installing the s/w**  
**REX**



Unzip it and installation is



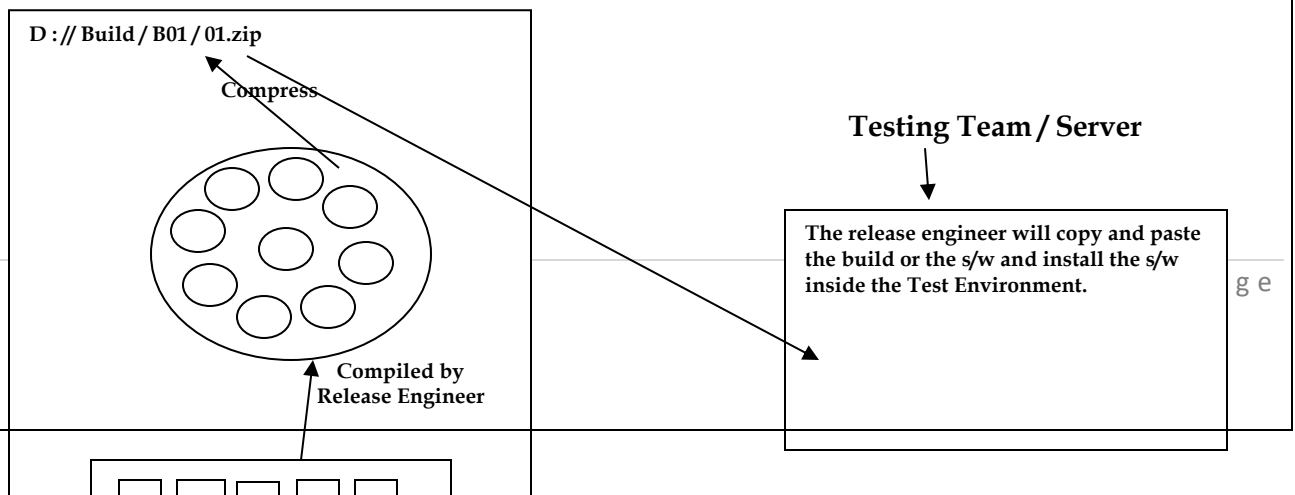
Here, the developers once they get the requirements, they start developing the software. As each feature is built, the code is compiled, compressed and stored in a compressed format. In the above example, the first build is shown. As soon as the first build is ready, the development lead sends a mail to the test lead saying that the first build is ready and they can start testing it. He also gives the name of the server and the file in which the first build is stored. The test engineer then goes to the development server named Rex, and copies the file to the testing server. He then unzips the file, installs the s/w in the testing team server and allots the various features of the first build to be tested to the testing team.

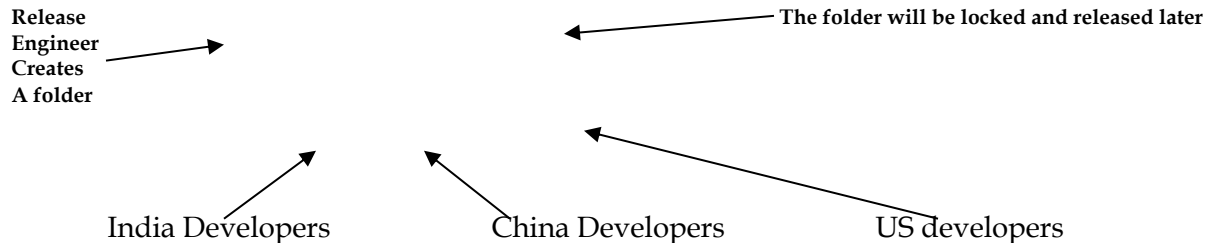
### **CASE 2 : Develop is installing the software**

When the developer is installing the s/w in the testing environment, and test engineer team should open the browser, copy the URL that is sent by the developer and paste it in the browser. If the developer installs, they'll give username and password. The test lead then logs in using the given the username and password, and creates his own username and password through which the testing team will log in and start testing the software.

### **CASE 3 : Release engineer / Build Engineer**

**Release engineer** is the one who manages the source code. If the developers are working at different location, the release engineer first installs a VCT ( Version Control Tool ) like CVS, VSS, Clear Case and creates a folder in the tool. The developers should copy and paste the programs into that folder. Once the folder is locked, the developers can't send their programs when the folder is locked. Now, release engineer will compile, compress and then build a s/w. Now he will only install the s/w inside the test environment (testing server) and sends a mail to the testing team. The testing team then start testing the s/w. If they find bugs, they report it to the developers. Once the developers fix these bugs and creates a new module simultaneously, the folder will be unlocked and once again the developers send the programs into that folder which is locked after a new build has been developed.





The release engineer performs 2 functions,

- Manages source code
- Creates a build and they install in the test environment

Whenever the Release Engineer compiles the program, if the program does not compile then he will send the program back to the developers and ask them to check (or) not to send that program at that particular build.

#### When do we find Release Engineer (ing) ?

- When the product is complex and big
- When more number of developers are there

Once the product is ready to release to the customer, then release engineer will only release the product to the customer because he will be knowing which piece of software is working fine.

The Automatic release of the Build can also be done by the release engineer. He should write a script while compiling the program. If the release engineer finds any bugs in the program, he will go and see who has written that program and sends that program to that developer ( to fix the bug and send or not insert that program into that folder ). **All the activities done by Release Engineer is called Release Management or Release Engineering.** For the entire project to manage resources, we use VCT. Whenever the developers want to take the program out from the VCT, we use **check out** option. Whenever the developer wants to insert a program into the VCT, we use **check in** option.

**Test cycle** – Time spent on testing a build or software completely / time taken by the test engineer to completely test one software.

5 days – 1 <sup>st</sup> test cycle	5 days – 2 <sup>nd</sup> test cycle	5 days – 3 <sup>rd</sup> test cycle
-------------------------------------	-------------------------------------	-------------------------------------

The developer writes a program and creates a module A and sends the module to the test engineer for testing. The test engineer tests the module and finds bugs and reports them to the developer. The developer fixes the bug and also creates a new module B. The developers only integrate module A & B and send it to test engineer. The test engineer will uninstall the old s/w and installs the new s/w and then does functional and integration testing for A & B. Whenever the new module is given for testing, if the test engineer catches

bugs and reports it, the developers fix the bugs. But, the developer does not send that particular module for testing, instead he sends the fixed module along with the new module for testing.

Whenever the new build comes, we should always uninstall the old build and install the new build (latest build). Whenever we uninstall, all the accounts and data created will be deleted. So, whenever we install new build – we must always login with manager Username and Password and create our own Username and Password.

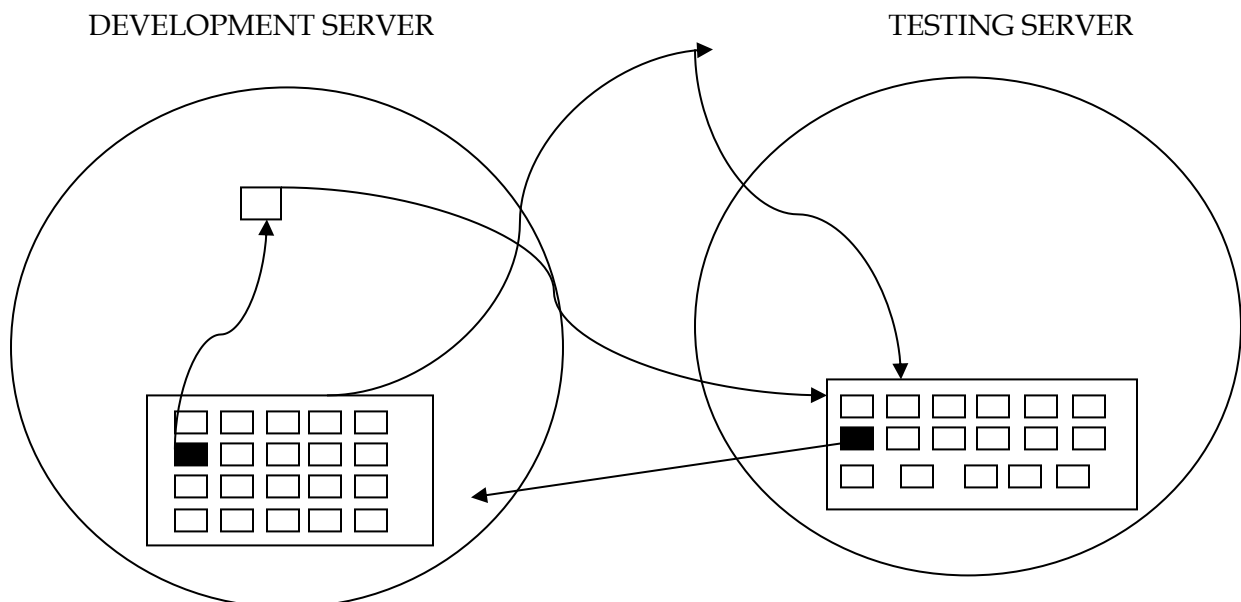
When the first build comes in, immediately we find a bug and send it immediately to the development team. Thus, immediately we find a bug within a cycle. If another build comes within a cycle where the bug is fixed, then we call it **respin**.

**We find respin** – when the test engineer finds blocker defects / critical defects. **For ex**, if the login feature itself is not working in gmail, then the test engineer cannot continue with his testing and it has to be fixed immediately – thus respin comes into picture. Inbox, Compose mail, Sent items are not blocker defects.

**More number of respin** in a cycle means the developer has not built the product properly.

Whenever the developer writes so many programs and sends the module for testing. The test engineer finds bugs and reports it to the developers. Once the developer comes to know the bug, he will look into the source code, if the problem is with only 1 program and that too only a few lines – he will fix the bug. He (the developer) will take the modified program, compile it and compress it into installable (format) and sends a mail to the test engineer that a **patch file** has been sent. Once the test engineer starts installing this **patch file**, the program which had defect will be replaced by the corrected program.

**Patch** – is a piece of software which has only modified programs.



The above figure shows how a patch file is installed. Let us consider that the testing team have installed the build and started testing the s/w. they find bugs and report it to the development team. The block (shaded) is the defect program. The developer looks at the defect program and sees that it just needs a few minor changes. He makes the necessary changes, compiles it and compress it and creates an installable (patch) and goes to the testing server and just installs the patch file which has the modified program. The testing team need not have to uninstall and install the build again.

### *INTERVIEW Tips & Questions*

**1) What is Build ?**

**Ans ) Answer is in the notes**

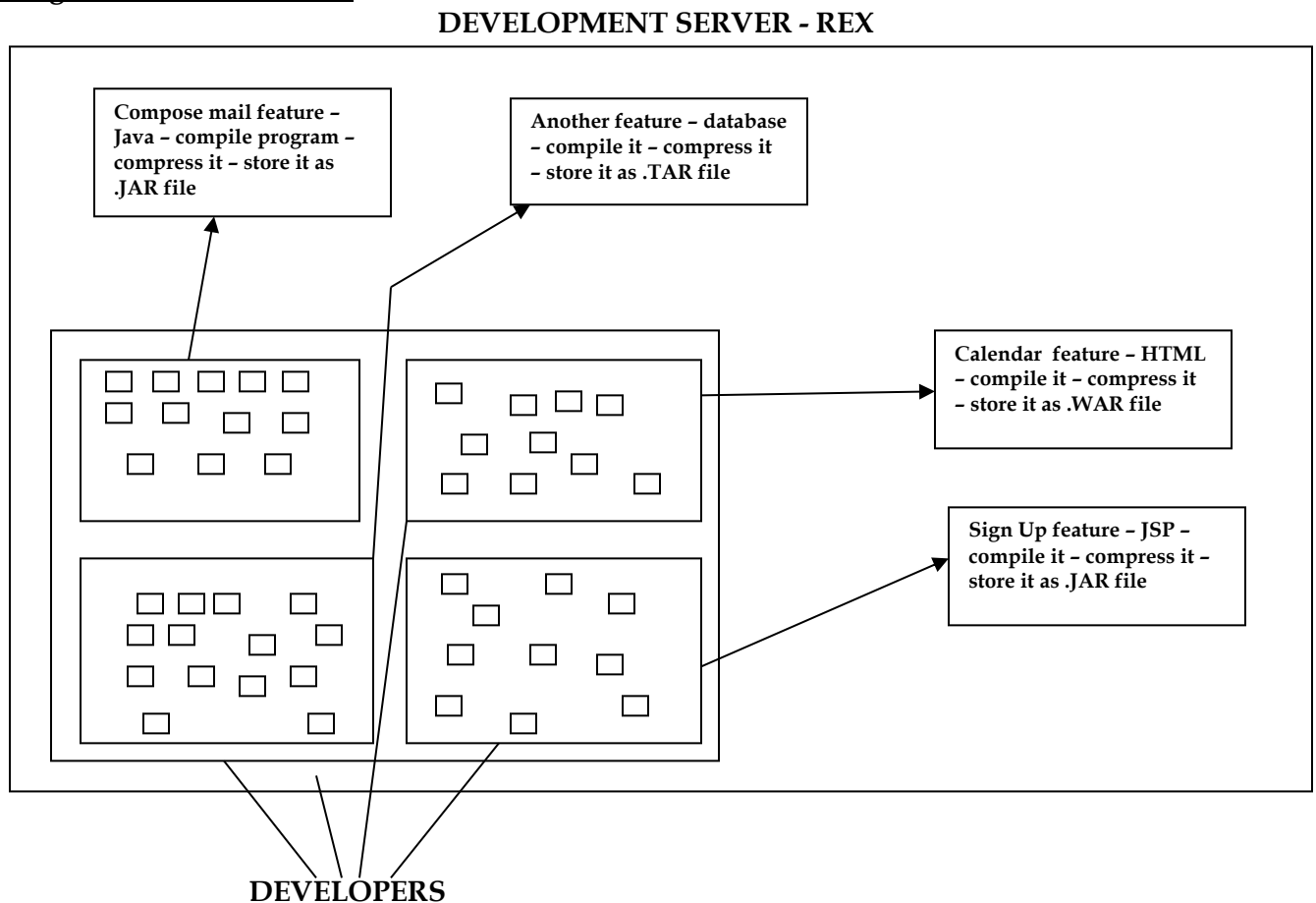
**2) In what format will you get builds ?**

**Ans ) in compressed format like a zip file, tar file, rar file, jar file etc.**

When we get a job, we will be working on 3 types of projects,

- **Stand-alone application** – software installed in one computer and used by only one person. **For ex -** Installing s/w of a Calculator, Adobe Photoshop, MS Office, AutoCad
- **Web Application** – any application software accessed through browser is called web application. **For ex -** yahoo.com, gmail.com
- **Client - Server application** – here, we are installing both client and server software to access the application.

### Testing WEB APPLICATION



Compose mail, Inbox, Sent Items, Trash, Spam, Contacts, SignUp, Login, LogOut, Forgot password, Settings, Documents, Tasks, Calendar, Help, Chat. .... Etc

## REQUIREMENTS

The customer sends the requirements to both development team and testing team. Now, the test lead will understand the requirements and logically assign the work to other team members.

The developers write the programs to build the features. They have to build the various features given by the requirements of the customers. The developer uses *different languages* and compiles and compresses the programs.

The developer compiles the Java program and then compresses it to **.Jar** file.

The developer compiles the JSP program and then compresses it to **.Jar** file.

The developer will compile the database and then compress it to **.Tar** file.

The developers will not compile the HTML directly and compress it to **.War** file.

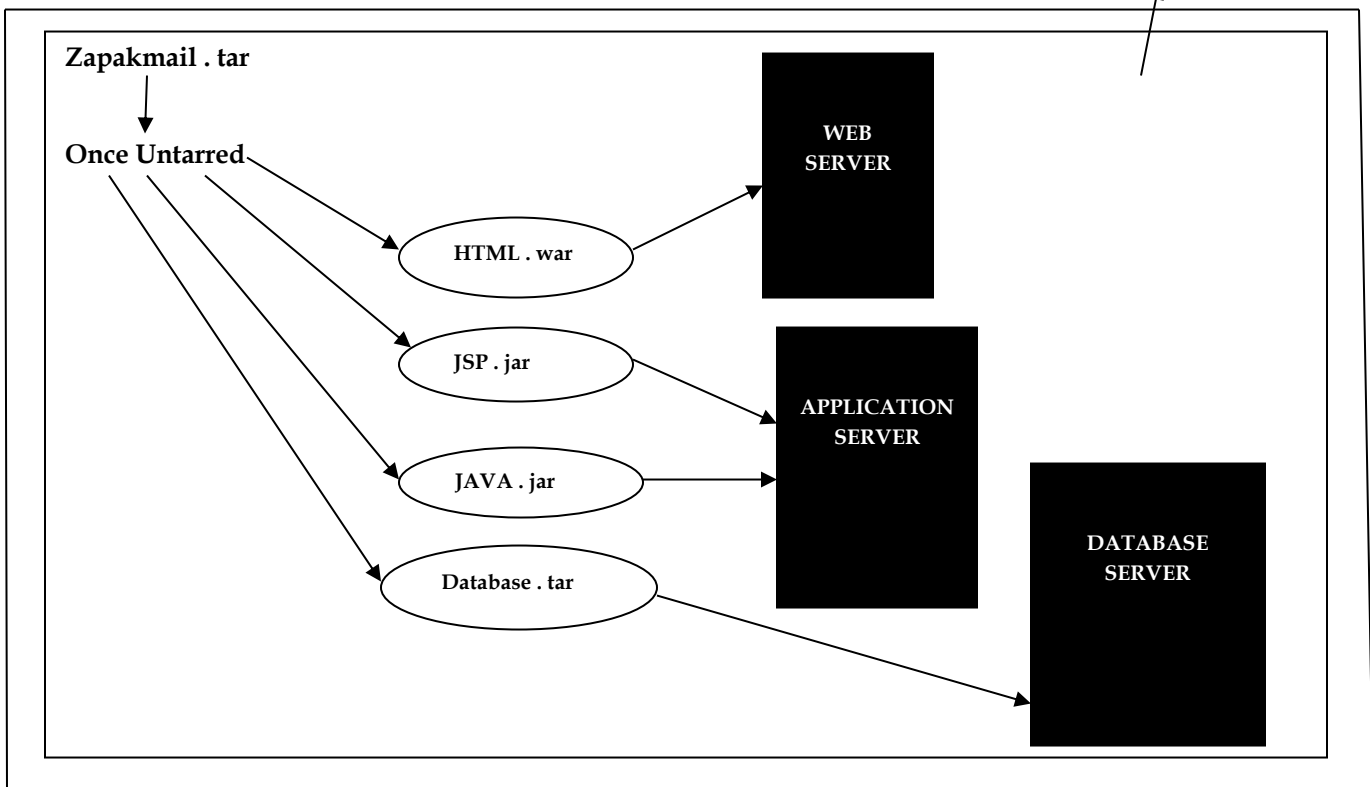
Now again he will compress all the compressed file into **.tar** and puts it into 1 file and stores it in **D:** drive as, **D :// Builds / B01 zapakmail . tar** and sends a mail to the testing team.

In the test environment 1<sup>st</sup> he will install the OS, Web Server, Application Server, Database Server. This is done before installing the product software. After installing, he will copy from **D :** drive and paste the **Tar** build. He will then **untar**, then we get **.jar, .tar, .war, .jar** files.

All JSPs and Web HTML programs should be moved into WebServer. All Java Programs should be moved into ApplicationServer and all DataBase programs goes to DataBaseServer. **All this is illustrated in the figure below,**

## TESTING SERVER

## Operating System





For Web Application, the WebServer, ApplicationServer, DatabaseServer should be installed. Once the application is ready for testing, the test lead will send a mail to his team with the following **URL**,

**<http://QA.zapakmail.com>**

with all the above required features like – compose mail, inbox, sent items etc.

The test engineer will open the browser and copies the **url** sent to him along with the username and password.

\*\*\*

Whenever new build comes,

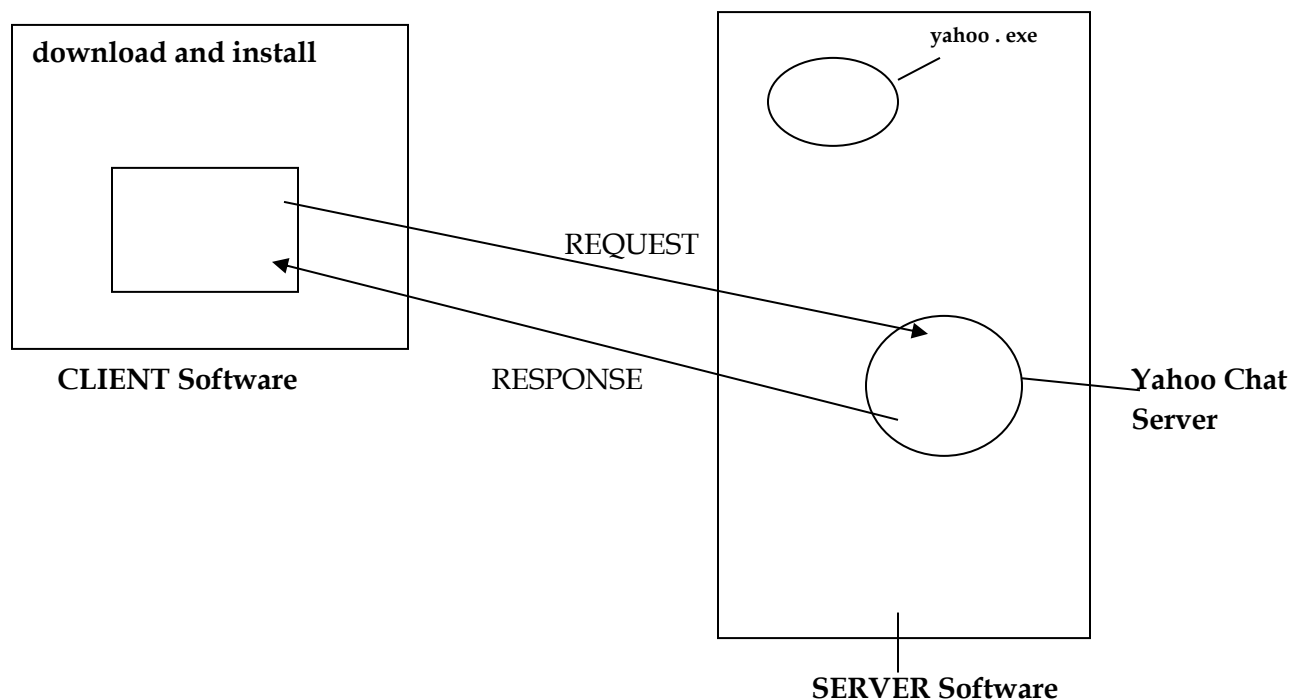
- Test Lead removes all programs in Web, Application and Database Servers
- Once again copy, untar and move new files to their specific server
- Once again start the server

**Testing CLIENT – SERVER application :-**

Ex – Yahoo messenger, Gtalk, ATM

Here, we **consider the example of YahooMessenger :-**

It requires 2 software – client software and server software.



Here, Yahoo messenger s/w is the client s/w and Yahoo server is the server s/w.

The main use of server s/w is,

- Server is used to communicate the information
- To store the information

Here, we 1<sup>st</sup> download yahoo messenger into the client software, the s/w is installed from the server. Yahoo messenger is installed in the client server. It then displays the username and password page (Login page). Once the client enters the username and password and submits, the request will be sent to the server, it will

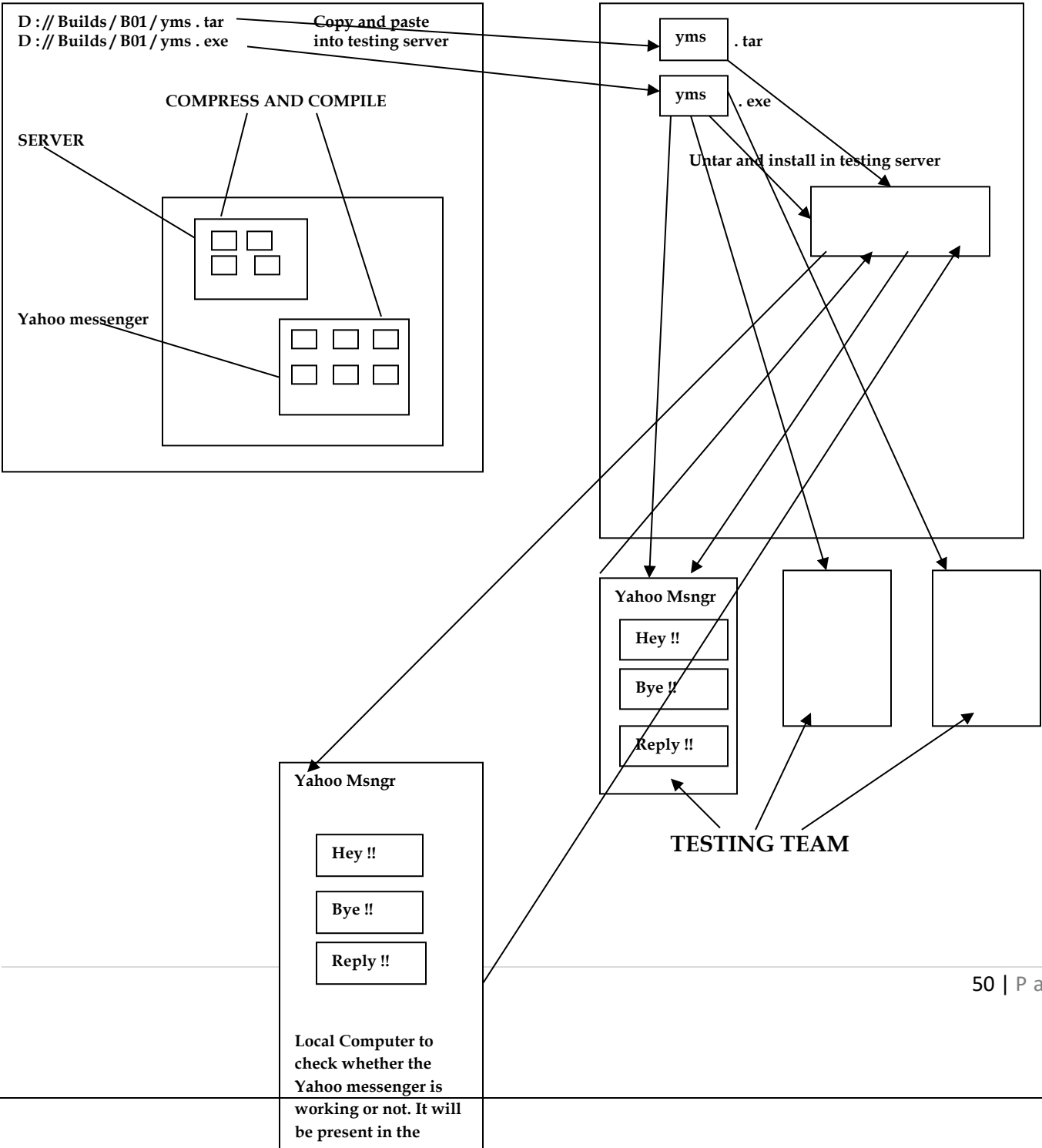
send a response to display the Homepage. The response is coming from the client server itself. The client displays the next window.

When we enter the information and submit, it goes to the Yahoo Server and stores information and sends a response. The client server displays the next window because of the executable Yahoo Messenger. Exe file on the client's PC.

The calendar, contacts etc and all other features of the YahooMessenger will open from the client s/w only because he has already downloaded and installed the s/w before only.

Whenever we double-click on Contacts, the page/window will be displayed once we enter the details and submit, the request goes to the server and stores the information and response is sent back.

DEVELOPMENT SERVER - Rex



Once the developer writes the program for YahooMessenger and server, he compiles and compresses,

- Server s/w will be named as yms.tar
- Client s/w will be named as ym.exe

This is saved in the folder as,

D : // Builds / B01 / yms.tar

D : // Builds / B01 / ym.exe

This is copied to QA ( test environment ) from Rex (development environment ) and pastes both the build. Yms.tar is untarred and install it in QA server

Then test engineer downloads and installs ym.exe in their computer as well as the local computer. Now start testing all the features.

When a bug is detected, it can wither be in client s/w or server s/w. changes can be done in server s/w or client s/w.

There are cases where we can predict or may not predict where the defect is.

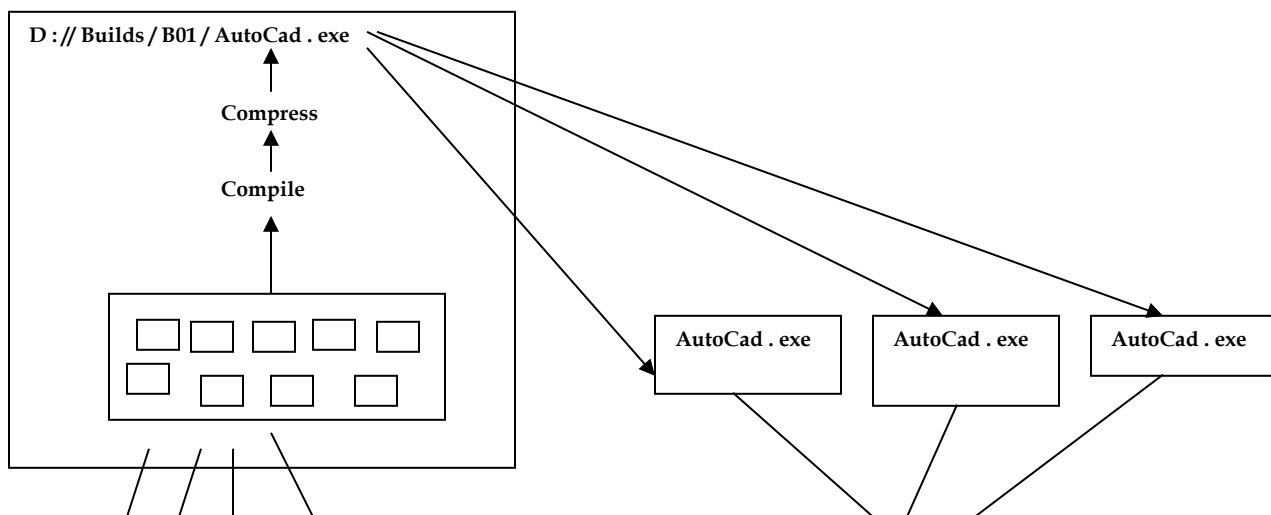
When the Test engineer is sending bugs to the developers, the developers will fix the bugs and add new features in either server s/w or client s/w or both.

When the new s/w build is sent to the Testing team. They will uninstall the server and installs the new server s/w. They will also uninstall the YM in their computer and install new YM because by adding new features, the previous YM may not work properly.

Whatever OS is used by end-users, the s/w has to be tested in all Os(s). Ex - XP, Vista, Windows 7 etc

Testing the s/w in different OS(es) is called **compatibility test**.

### Testing STAND - ALONE application :



## DEVELOPMENT TEAM

## TESTING TEAM

**Examples** of stand-alone applications are – Photoshop editor, Nero, MS-word, etc  
Here, we are installing only 1 s/w which is client s/w.

The developers write the programs, they compile and compress and save it in,

**D :// Builds / B01 / AutoCad.exe**

Development lead sends a mail to Test lead saying that the product is ready.

Now, the Testing Team should copy, paste and install the s/w in his own computer or local computer and test the allotted features.

Whenever the developer sends a new Build, the Test engineer will uninstall the old build and install the new build. Before releasing the product, we should also test for compatibility. Always we should work on latest build.

## RELIABILITY TESTING

**Testing the functionality of an application continuously for a particular period of time.**

**For ex** – let us consider our cellphones / mobiles. The s/w may not work continuously. In mobile, after a week (or) ten days, the phone may hang up because whatever feature is present in the phone, whenever it is used it continuously creates an object in the RAM of the phone. Once the RAM is completely filled with objects, if we get any call or message – we will be unable to press the call button and the phone hangs up. Thus we make use of a **clean-up software**. When we switch off the phone and switch it on again, all the objects in the RAM gets deleted.

For doing Reliability testing, we write an automated program or script and click on Run. We do Reliability testing using ready-made tools and not manually. The test engineer will run the programs using Automated tools.

## RECOVERY TESTING

**Testing the application to check how well it recovers from crashes or disasters.**

*The steps involved in Recovery Testing are,*

1. Introduce defect and crash the application – Somebody will guide us as to how and when will the software crash. OR. By experience after few months of experience on working the project, we can get to know how and when the s/w can and will crash.
2. Whenever s/w crashes, it should not disappear but should write error log message (or) crash log message where in reason for crashing should be specified. **Ex - C :// Program Files /QTP /crash.log**
3. It should kill its own process before it disappears. **For ex** – In Windows, we have TaskManager to show which process will be running.
4. Re-open the application. The application must be reopened with previous settings.

**For ex** – when using Mozilla FireFox, if the power goes off. When we switch on the PC and re-open Mozilla FireFox, we get a message asking whether we want to **start a new session** or **restore previous session**.

For any product developed, the developers write a recovery mechanism which explains – why the s/w is crashing, whether crashlog messages are written or not, etc.

## ACCEPTANCE TESTING

**Acceptance testing is done by end users. Here, they use the s/w for the business for a particular period of time and check whether the s/w can handle all kinds of real-time business scenarios / situations.**

For Acceptance testing, let us consider the example shown below.



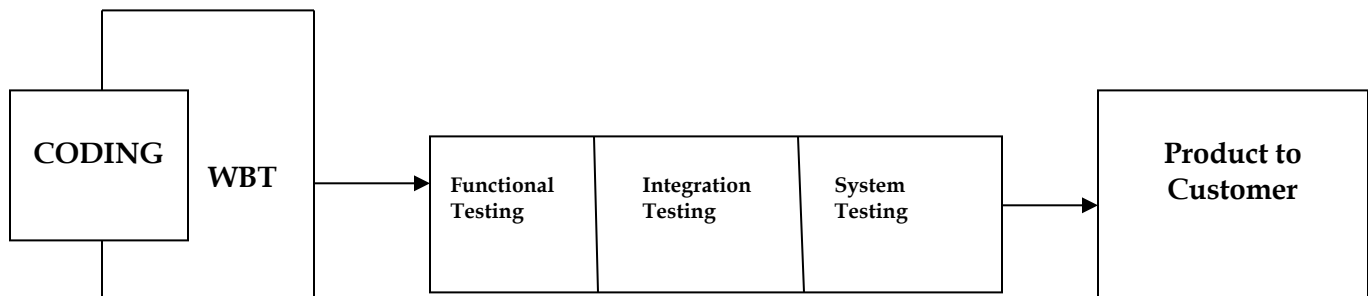
Fed-ex with its requirements asks Wipro to develop the s/w and Wipro agrees to give the s/w in 2 releases like below,



On September 8<sup>th</sup>, test manager tells the project manager that there is a critical bug in the application which will take another 5days to fix it.

But the project manager says you just deliver the application and by the time they implement in Fed-ex, it takes another 25days so we can fix the bugs or otherwise we will have to pay the penalty for each day after the said release day. **Is this the real scenario ? – No.** Then what happens, we will see now in 3 cases which really and who really does the acceptance testing.

**CASE 1 :-** here, we will discuss how the acceptance testing is done or how the test engineer testing becomes the acceptance testing here.



Usually, the actual flow of testing will be like above. But, here a small difference we see where the system testing or end-to-end testing becomes the acceptance testing. To understand this, follow the sequence below,

Fed-ex gives the requirements and Wipro develops the s/w and do all testing and gives it to Fed-ex  
Are the Fed-ex going to use the s/w as soon as they get from Wipro ? – NO, certainly not. Then what do they do ? – Observe,

Fed-ex, they have some group of Test Engineers and after they get the s/w, this team starts testing it. So, now we can understand that though the test engineer do the testing but it is done at customer level. This end-to-end testing is called ACCEPTANCE TESTING.

**The difference between Wipro test engineers and Fed-ex test engineers are,**

- The Wipro testing do Functional Testing, Integration Testing and System testing. But at Fed-ex, the testing team do only end-to-end testing / system testing.

The difference between end-to-end testing of Wipro and Fed-ex is,

- Fed-ex engineer is a domain expert
- Fed-ex engineer understands the business well
- Fed-ex engineer tests for real time data
- Fed-ex engineer is the one who gave the requirements.

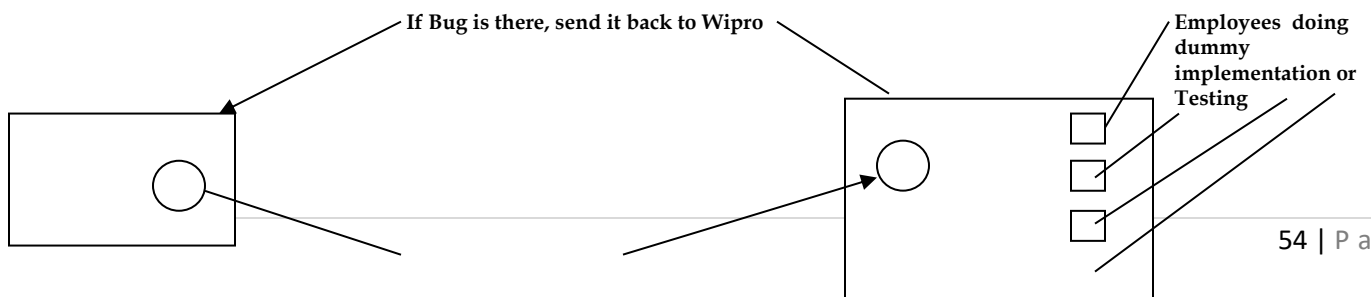
To understand this, we see the example below. If the application format is like below,

In the above example, after the product is given to Fed-Ex Test Engineers, they do testing and they know after the application has been filled above, it should produce an message saying “Parcel 1 Docket ID Produced”. If this is not happening, they give back the application for fixing bugs. Now, the Fed-Ex checks whether this feature is there or not in the requirement. If it is there and Wipro have not done fix it, then Penalty Counts for Wipro from that day, whereas the TE at Wipro will not be knowing this and thus arises the difference in testing at Wipro and Fed-Ex.

Thus, the TE become END-USERS here and this testing is known as Acceptance Testing.

### CASE 2 :-

In this case, we see how the employees are becoming end-users and do acceptance testing.





The s/w is developed and tested at Wipro's place and then sent to Fed-ex. At Fed-Ex, they have less TEs and so it is not possible for them to do Acceptance testing. So, out of 400 employees of Fed-ex, Fed-ex gives the s/w to 40 employees and installs the product at their systems and asks them to start using the s/w and come up with bugs or issues.

Now, the 40 employees, they do dummy implementation (i.e, they implement the data into the application and also have the data written manually). Now, the employee here becomes the end-users and come up with bugs and issues when using the s/w.

These issues are verified against requirements and now penalty is charged for Wipro ( sometimes, penalty is charged on an hourly basis ).

If the bug found is not as per requirement, then Fed-Ex can go for CR or RFE.

CR - Change Request - i.e, if the requirement has not been specified properly, then Fed-Ex gives the correct requirement and requests for change.

RFE - Request For Enhancement - if Fed-Ex feels that a particular module can be enhanced and developed in a better way, then they can send the CRS as RFE and Wipro goes on to make the necessary changes.

**Thus, Acceptance Testing can also be defined as - end-to-end testing done by engineers sitting in customer's place. Here, they take real time scenarios and check whether the s/w works or not. Here also, we are able to take real time business scenarios because the end-users know how the business flow works.**

**We are getting more and more builds for Acceptance Testing means,**

- The product quality which is delivered to customers is not good. Development and testing both are not good.
- After receiving the s/w, customer is getting more and more ideas, so he is asking for more and more changes
- The requirement which was given in the beginning is not clear.

### **CASE 3 :-**

Here, the Fed-ex customers become the end users.

Here, the s/w is developed and tested and implemented at Fed-ex production servers and thousands of users start using the s/w. This comprises the **1<sup>st</sup> release**. When using the s/w, Fed-ex comes up with more number of features and enhancements and sends the CRS to Wipro who make the additional changes and modules and give it to Fed-ex.

Thus, what is happening here is - the requirements are collected by Fed-ex from customers and end-users and then the s/w is developed.

**The number of cycles depends on,**

- Number of features
- Complexity of features
- How new features affect old features

**Hot fix** - in production environment, whenever the client finds critical bugs - developers fix the bugs - small team of TEs test it - reinstall the s/w - client starts using the new s/w. This entire process is known as **Hot fix**. It takes few hours to 1day.

**For ex**, if the login feature itself is not working at the production environment, then the client immediately sends it for fixing which is done asap.

## SLA - Service Level Agreement

### Interim Release - ( short release ).

Between 2 major releases - there is a short release of enhancements - this comes up when the client requires a small bunch of features very urgently. Out of 70 developers, around 10 come out and out of 30 TEs, around 3 come out - they develop and test the s/w - client does 1 short round of Acceptance testing - before adding it to the production environment - this interim could take just around 15 days to 1 month.

**\*\*\* SMOKE TESTING or SANITY TESTING or DRY RUN or SKIM TESTING or BUILD VERIFICATION TESTING \*\*\*** (Very very important interview question)

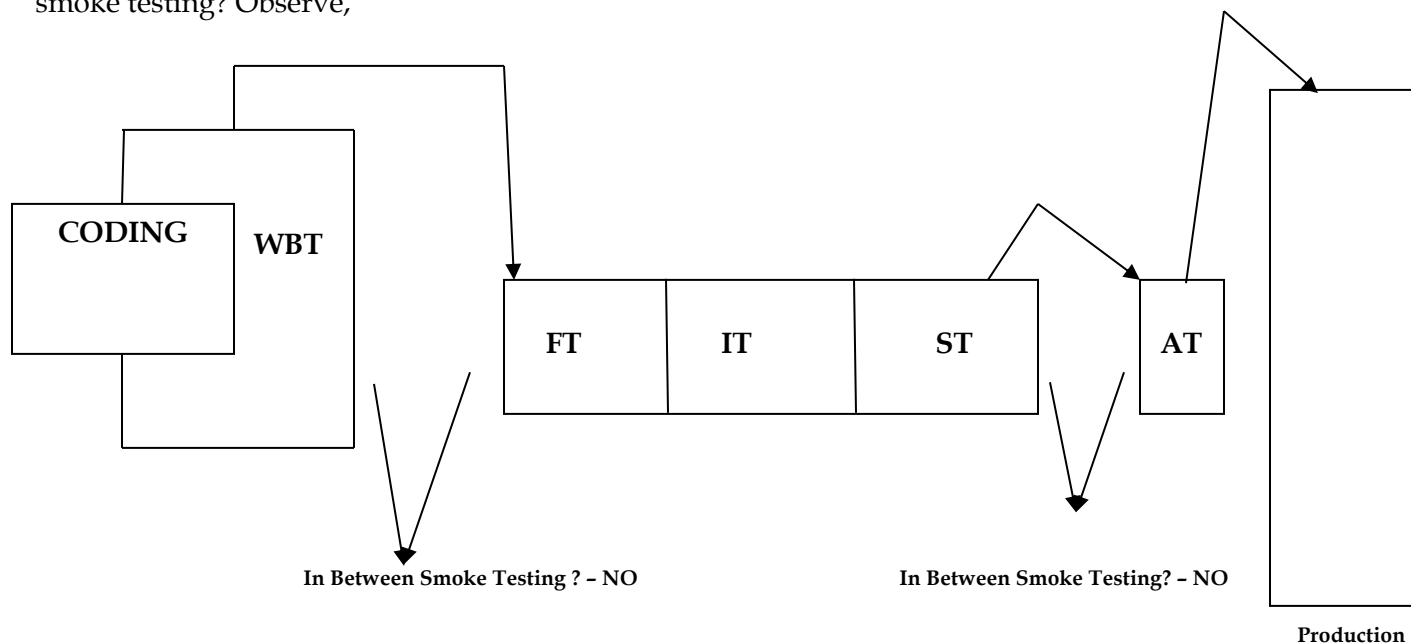
**Testing the basic or critical features of an application before doing thorough testing or rigorous testing is called as smoke testing.**

It is also called Build Verification Testing - because we check whether the build is broken or not.

Whenever a new build comes in, we always start with smoke testing, because for every new build - there might be some changes which might have broken a major feature ( fixing the bug or adding a new feature could have affected a major portion of the original software).

In smoke testing, we do only positive testing - i.e, we enter only valid data and not invalid data.

Do we have separate testing (or) do we have to do it in between FT, IT, ST ? Then, where actually do we do smoke testing? Observe,



From the above diagram, it may be confusing when we actually do smoke testing

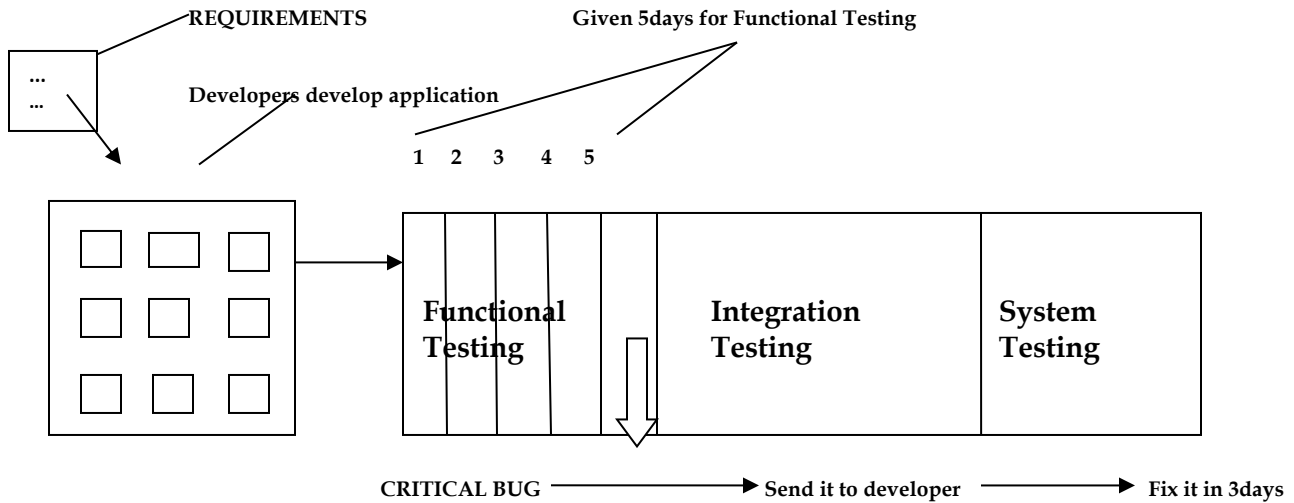
Now, we have to understand that smoke testing is done in all testing before proceeding deep into the testing we do.

The below example will make us understand better when to do smoke testing,

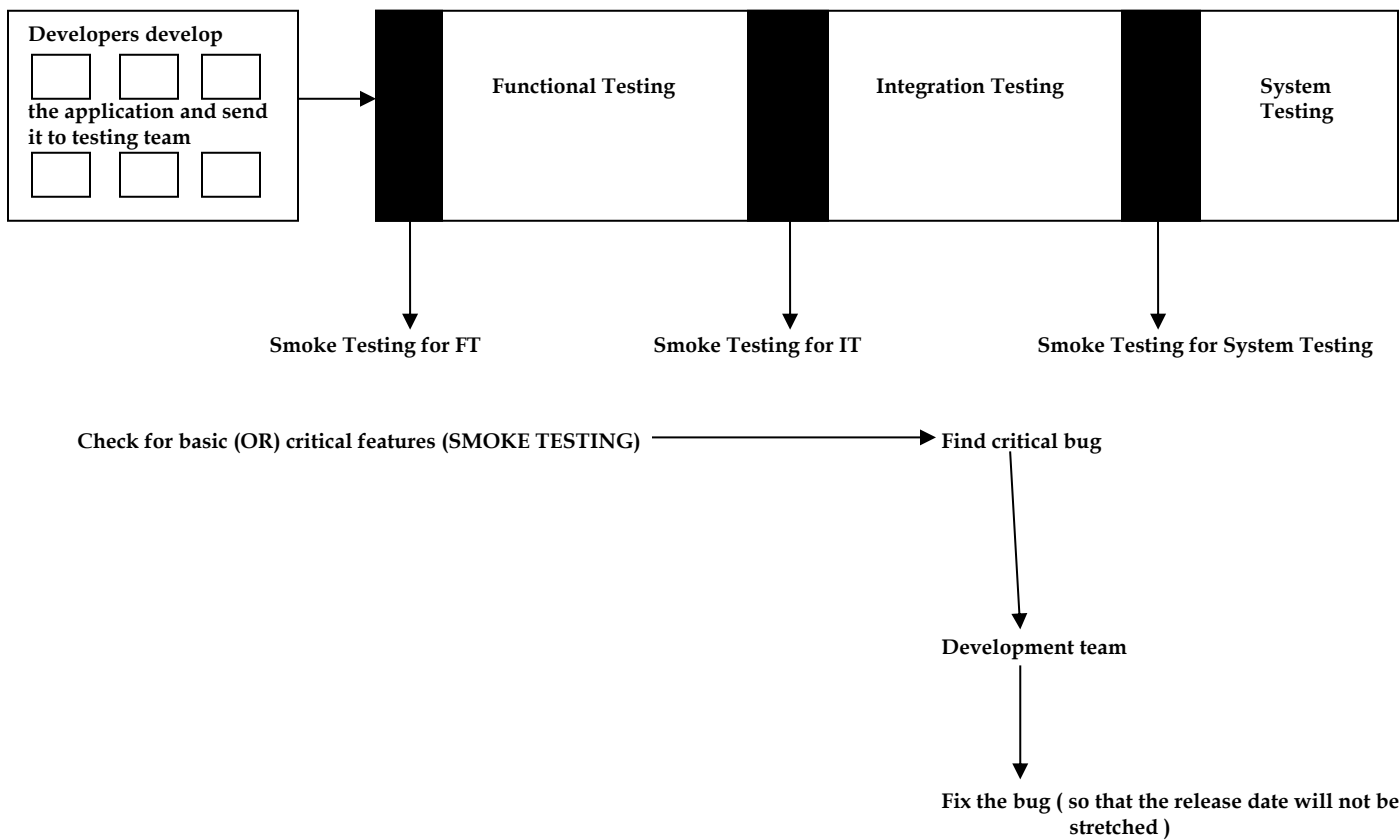
Developers develop application and gives it for testing. The testing team will start with FT. suppose we assume that 5 days we are given for FT. on the 1<sup>st</sup> day, we check one module and later 2<sup>nd</sup> day we go for another module. On the 5<sup>th</sup> day, we find a critical bug, when it is given to the developer - he says it will take another 3 days to fix it. Then we have to stretch the release date to extra 3 days.

Then how do we overcome this ? - Observe how smoke testing works here. In the above scenario, instead of testing module by module deeply and come up with critical bug at the end, it is better to do smoke testing before we go for deep testing i.e, in each module - we have to test for basic (or) critical feature and



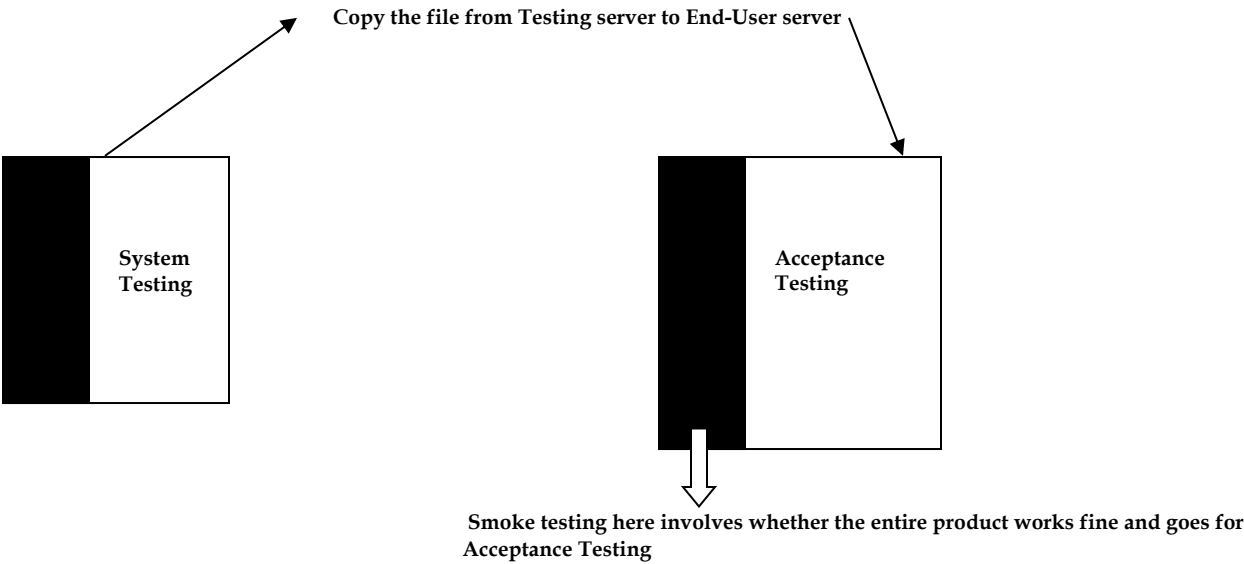


then proceed for deep testing. The scenario will be like this as shown in the figure below,



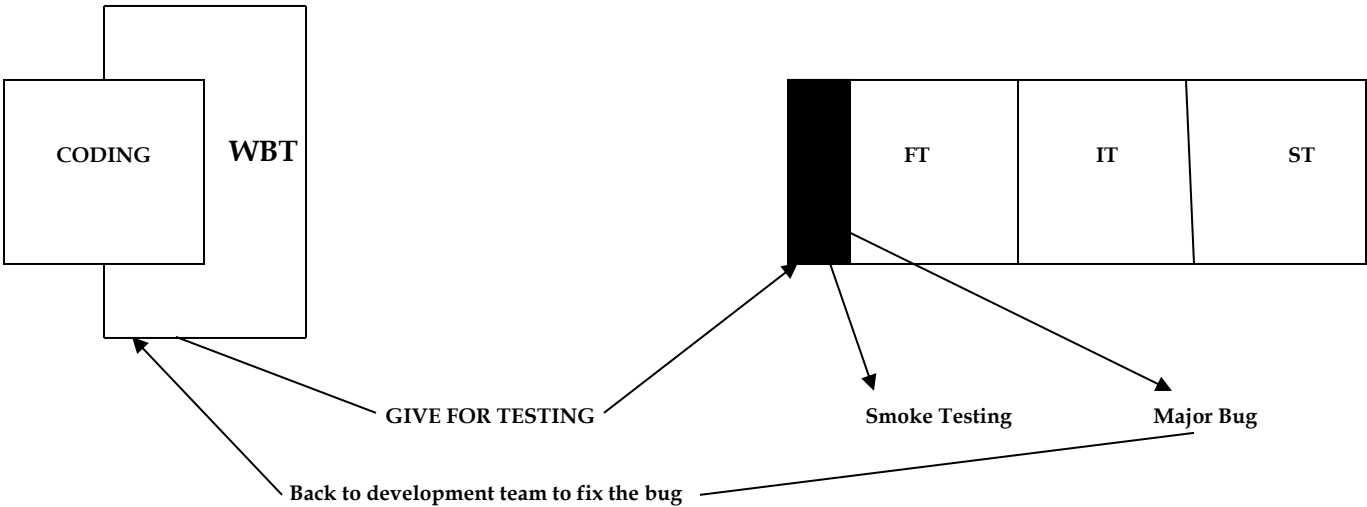
Question arises – how do we know which is the critical feature? – we will come to know which is the critical feature or basic feature when we proceed with the testing.

Smoke Testing in Acceptance Testing

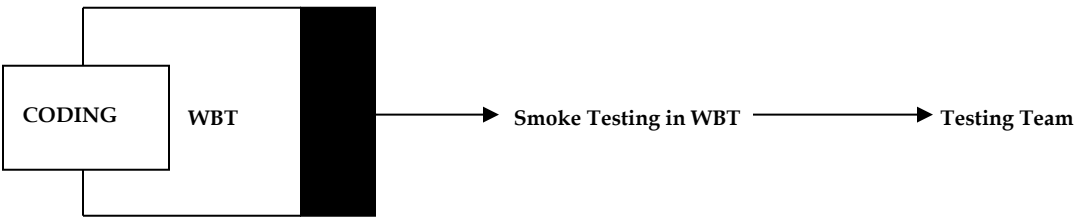


Let us now consider 3cases where in which we see where we do smoke testing in different kinds of testing.

CASE 1

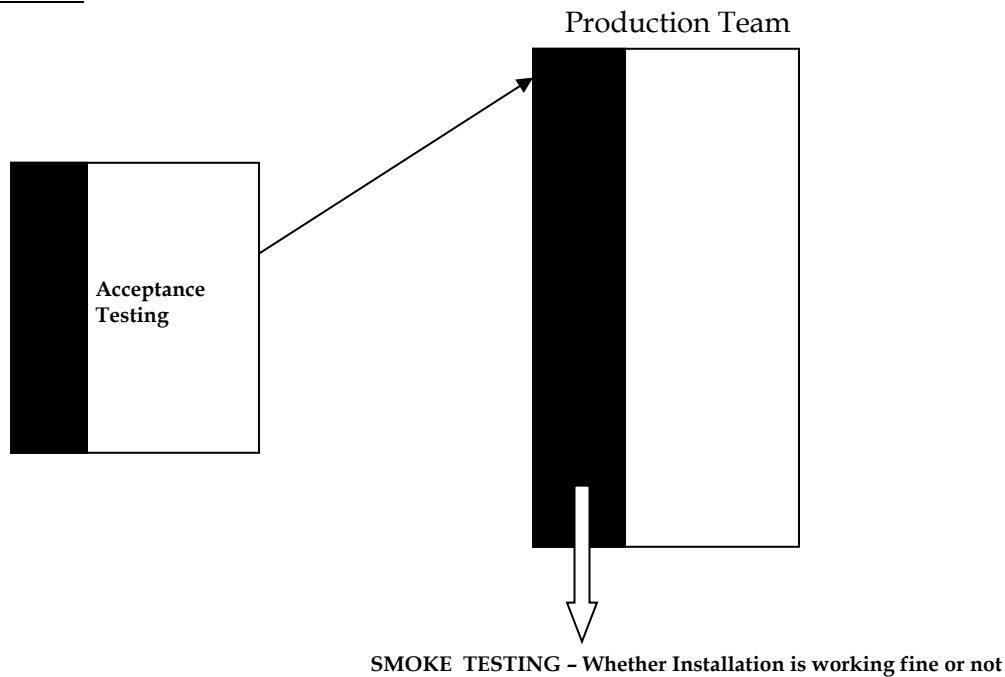


Major bug in the initial stages is an insult to the developer. Sometimes, the below procedure is also followed,



If this is done, then the testing team need not do smoke testing as the bugs are already fixed in WBT. But depending upon the project or the organization, normally this is not followed.

### CASE 2



In the above scenario, smoke testing in production team means after we do Acceptance Testing, Smoke Testing involves whether the s/w developed is installed fine or not.

### Important Points to Remember

- When we are doing smoke testing, we do only positive testing (only valid data is entered)
- Here, we test only basic or critical features
- Here, we take basic features and test for important scenarios
- Whenever the build comes to the customer, before the customer / client does Acceptance Testing, he also does Smoke Testing before doing Acceptance Testing
- When the product is installed in production, we do quick smoke testing to ensure product is installed properly.

### Why we do Smoke testing ?

- Just to ensure that product is testable
- Do smoke testing in the beginning – catch bugs in basic features – send it to development team so that development team will have sufficient time to fix it.
- Just to ensure that product is installed properly.

In early stages of product development, doing smoke testing fetches more number of bugs. But, in later stages of product development, if you do smoke testing – the number of bugs that you are going to catch in smoke testing will be very less. Thus, gradually the effort spent on smoke testing is less.

**Note :-** only for information purpose (NOT FOR STUDYING)

Smoke testing is classified into two types,

- *Formal Smoke Testing* – the development team sends the s/w to the test lead. The test lead then instructs the testing team to do smoke testing and send report after smoke testing. Once, the testing team is done with smoke testing, they send the smoke testing report to the Test lead.
- *Informal Smoke Testing* – here, the test lead says the product is ready and to start testing. He does not specify to do smoke testing. But, still the testing team start testing the product by doing smoke testing.

## INTERVIEW QUESTIONS

### *1) Difference between Smoke Testing and Sanity Testing and Dry Run*

#### *Ans) Sanity Testing*

- *Narrow and deep testing. Unscripted*
- *Take some very very important features and do deep testing*
- *It is manually done*

#### *Smoke Testing*

- *Scripted. Shallow and wide testing*
- *Take all important features and do high-level testing*
- *Build comes – write automation scripts and run the script. Thus test done automatically.*

*Dry Run - A dry run is a testing process where the effects of a possible failure are intentionally mitigated. For example, an aerospace company may conduct a "dry run" of a takeoff using a new aircraft on a runway before the first test flight.*

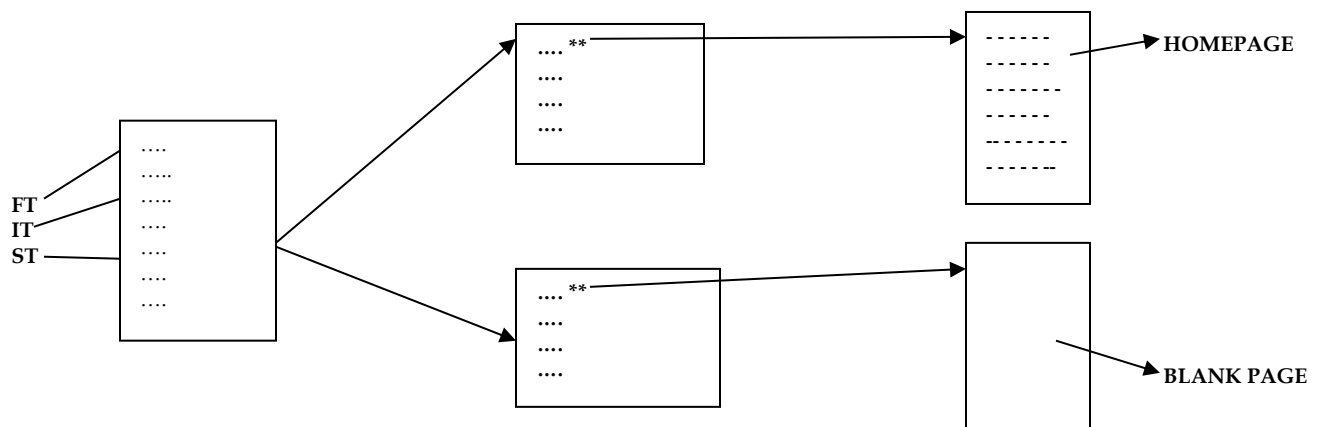
*If he still expects more differences – then just tell – “ for study purposes, i had visited wikipedia website and also [www.allinterview.com](http://www.allinterview.com) . So, i know this answer. Can u please tell me which website i should visit to get to know the exact answer. That’s how i could answer this question “.*

AD – HOC Testing ( also called Monkey Testing/ Gorilla Testing )  
Testing the application randomly is called Ad-hoc testing.

### Why we do Ad-hoc testing ?

1) End-users use the application randomly and he may see a defect, but professional TE uses the application systematically so he may not find the same defect. In order to avoid this scenario, TE should go and then test the application randomly (i.e, behave like an end-user and test).

For ex,

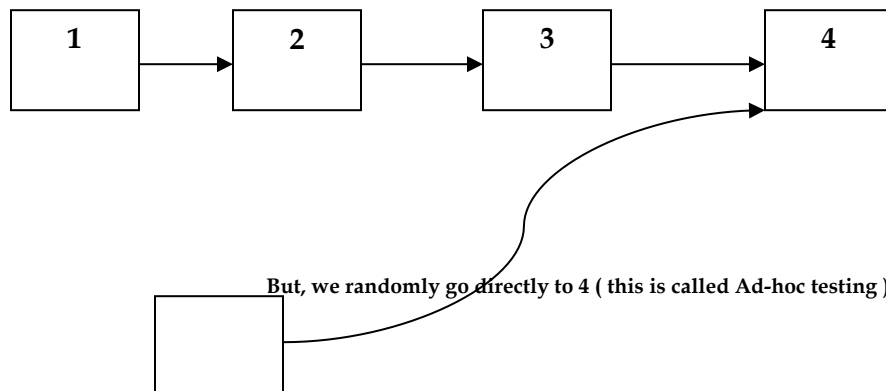


In the above figure, after we have tested the application for FT, IT and ST – if we click on some feature instead of going to homepage (or) sometimes datapage, if it goes to blank page then it will be a bug. In order to avoid these kind of scenarios, we do Ad-hoc testing.

2) Development team looks at the requirements and build the product. Testing Team also look at the requirements and do the testing. By this method, Testing Team may not catch many bugs. They think everything works fine. In order to avoid this, we do random testing behaving like end-users.

3) Ad-hoc is a testing where we don't follow the requirements (we just randomly check the application). Since we don't follow requirements, we don't write test cases.

Requirement says to test like below ( 1 -> 2 -> 3 -> 4 )



Examples of Ad-Hoc testing for Gmail :

- 1) Login to Gmail using valid username and password. Logout from Gmail. Click on Back button. It should not go back to Inbox page. If it does, then it is a javascript error and it is a bug. It should go back to Login page and say session expired.
- 2) Login to Gmail homepage using valid username and password. Once we are in Inbox page, copy the URL of the inbox which is in the address bar of the homepage and paste it in Notepad file. Logout from Gmail. Now, open browser page and paste the URL of the inbox in the address bar. It should not go to the inbox, instead it must go to the welcome page of Gmail.
- 3) Login into Gmail. Go to Settings and Change Password. Set the old password only as the new password and see what happens.

Let us consider **an example of Fraud Management System of Online Banking Application.**

**BLOCK ACCOUNT**

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
BLOCK ACCOUNT  
.....  
.....  
.....

ACCOUNT NUMBER

**BLOCK** **CANCEL**

Confirmation Message should be displayed

**ACCOUNT IS BLOCKED**

When we click on the **Block Account** link, we are transferred to the Block Account page where we find several features in that. Enter the data and click on Block Account, then that account has to be blocked.

Now, we will see how we can do **Ad-hoc testing on this application.**

- 1) Login as Bank Manager and enter the Account Number and click Block and see whether it is blocked or not.
- 2) Before blocking the Account, Go and Delete the person whose account is to be blocked and again Login and check whether it is blocked or not. As we click the Block it should throw a message saying customer not available as an error message. Here, we randomly check the application and nothing is mentioned in the requirements. Thus here we do Ad-hoc testing.
- 3) Suppose some User B transfers money to A whose account is blocked. In this case also, we should get a message saying Account(of A) is blocked [ i.e, by the time B transfers money, manager blocks A's account]. Actually the requirement does not say check for money transfer from other account and do testing. But this testing is done by TE not against the requirement. Even sometimes without throwing message that Account is blocked the money gets transferred. In this case also, the TE checks for it and thus it becomes Ad-hoc Testing.

**NOTE :-**

- Ad-hoc testing is basically negative testing because we are testing against requirements ( out of requirements ).
- Here, the objective is to somehow break the product.

**When to do Ad-Hoc testing ?**

- Whenever we are free, we do Ad-hoc testing. i.e, developers develop the application and give it to testing team. Testing team is given 15days for doing FT. In that he spends 12 days doing FT and another 3days he does Ad-hoc testing. We must always do Ad-hoc testing in the last because we always 1<sup>st</sup> concentrate on customer satisfaction
- After testing as per requirements, then we start with ad-hoc testing
- When a good scenario comes, we can stop FT, IT, ST and try that scenario for Ad-hoc testing. But we should not spend more time doing Ad-hoc testing and immediately resume with formal testing.
- If there are more such scenarios, then we record it and do it at the last when we have time.

**GLOBALIZATION TESTING**

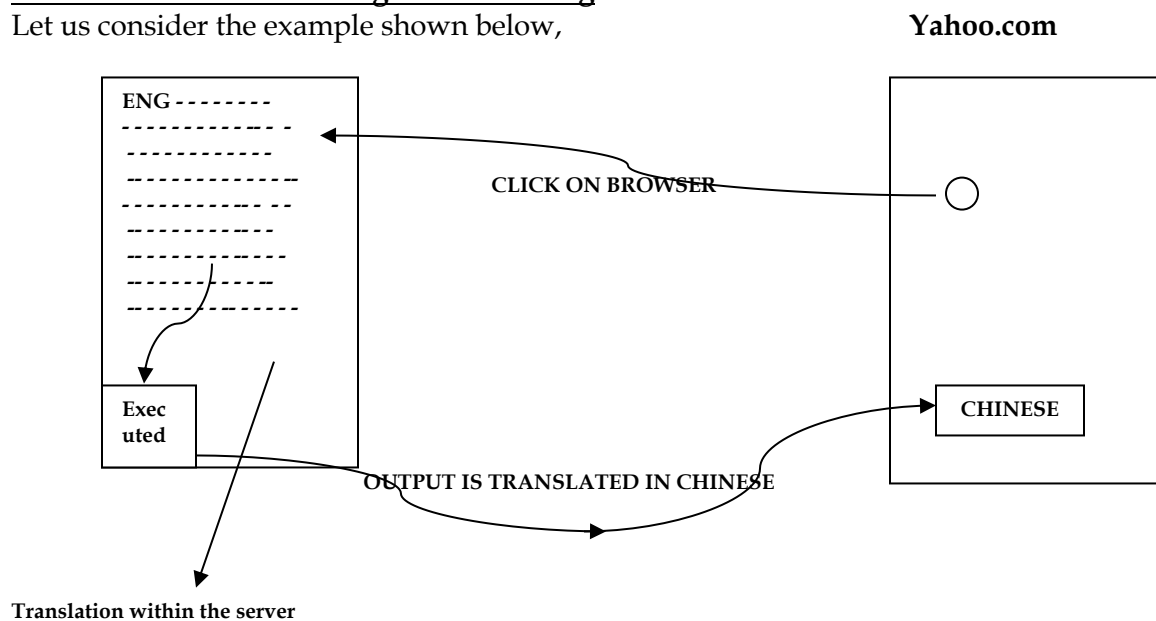
Developing the application for multiple languages is called **globalization** and testing the application which is developed for multiple languages is called **globalization testing**.

There are **2 types** of globalization testing,

- Internationalization Testing ( I18N testing )
- Localization Testing ( L10N testing )

**Internationalization Testing or I18N testing**

Let us consider the example shown below,



Suppose if we want the application in Chinese, then we click on the browser it will take to the server where the program is in English, from there it is executed and the output is translated into Chinese and displayed in Chinese language.

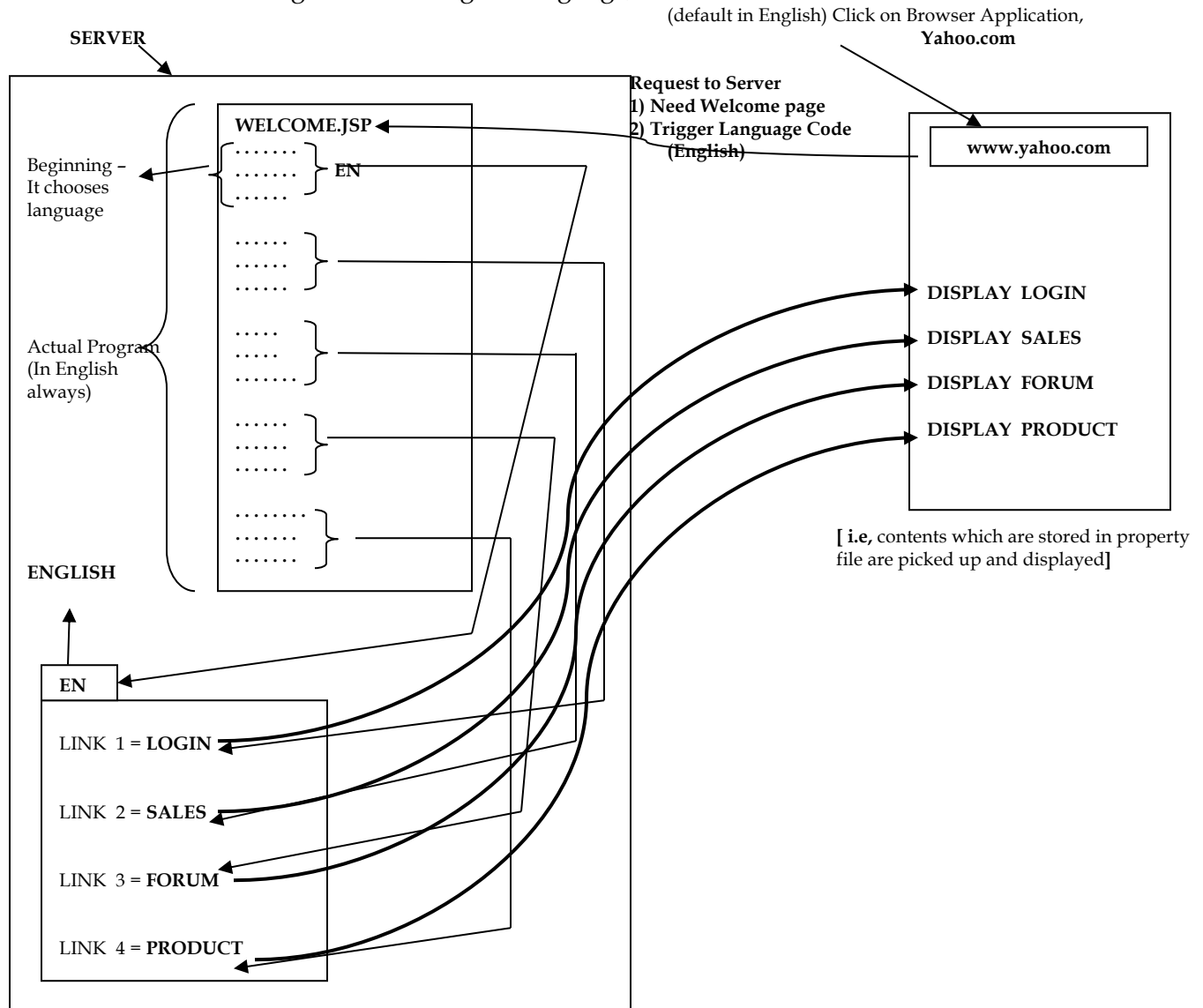
How do we do this translation of language ? – By using a translator ? – NO. Then how do we do it ? Before thinking, let us see the **drawbacks in using a translator**.

When the programs are translated from English to different languages, using translator the following occurs,

- Meanings are changed
- Not conveying the feeling. Thus usage of translators are ruled out. For ex - in English, the word "welcome" when translated to Kannada using translators will mean "baavige baa"!!!

The above problem is handled by using **property files**. *Property Files are nothing but files containing source data (like, a Notepad). In Java, it is called Resource Bundle.*

Let us see how I18N testing works for English language,



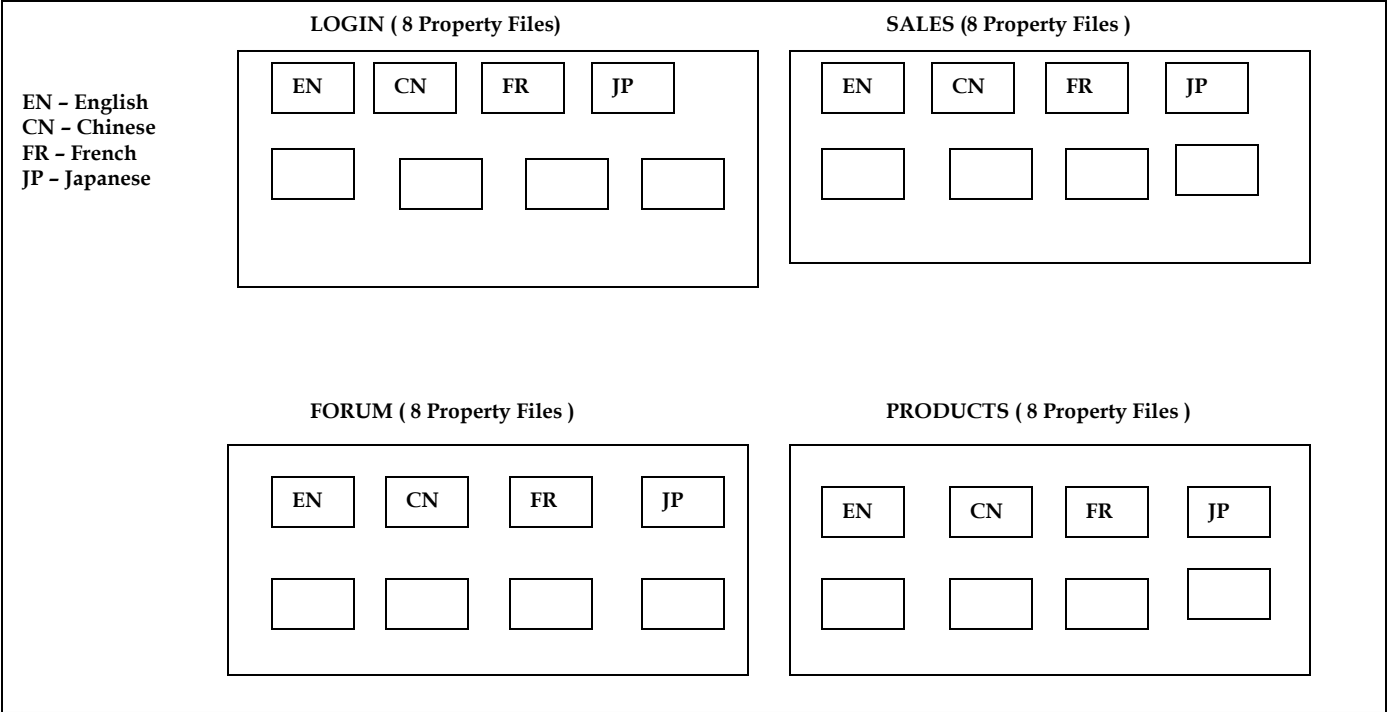
Write a common program in English for all languages and have different property files for different languages. Suppose, if we have 10 different languages, then we should have 10 different property files. **We see how the above request works,**

Click on the browser, it takes a request to the server saying the following - need Welcome page, trigger the language code EN



It takes to the program where the language is executed and takes the property file of ENGLISH.  
Now, the English property file is selected and the next coding in the program connects to its respective links. **For example**, if the next set of coding is for Login – it connects to Link 1, picks up the data stored there and displays it in English. Same procedure followed for other links also ( like sales, forum and products ).  
Again, the same procedure followed for other languages also.

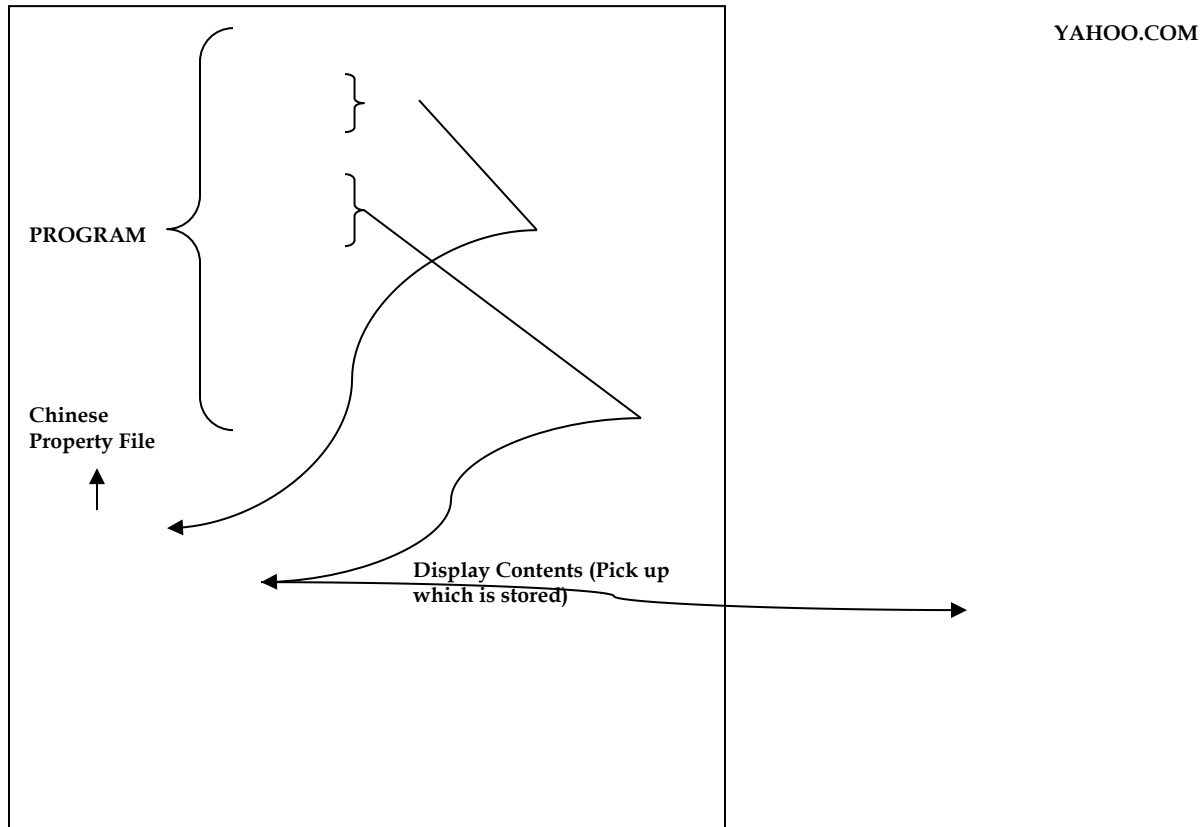
*Who writes these property files ?*  
1<sup>st</sup>, the developers writes a property file in English. Language experts takes this and manually write the property file and gives it to developers.  
Now, the developers place this property file as shown below in the server.



Let us see how we will translate into Chinese language and test for it.  
Same procedure like above  
Click on browser. Select for language Chinese before you click.  
After clicking, it takes it to Welcome page with a trigger language code CN  
So, it takes to the CN property file and display the contents (Chinese) stored in the file according to the link it gets connected ( **for ex**, LINK 1 to Login display the contents of LOGIN. Similarly for other like Sales, Forum, Product also ).

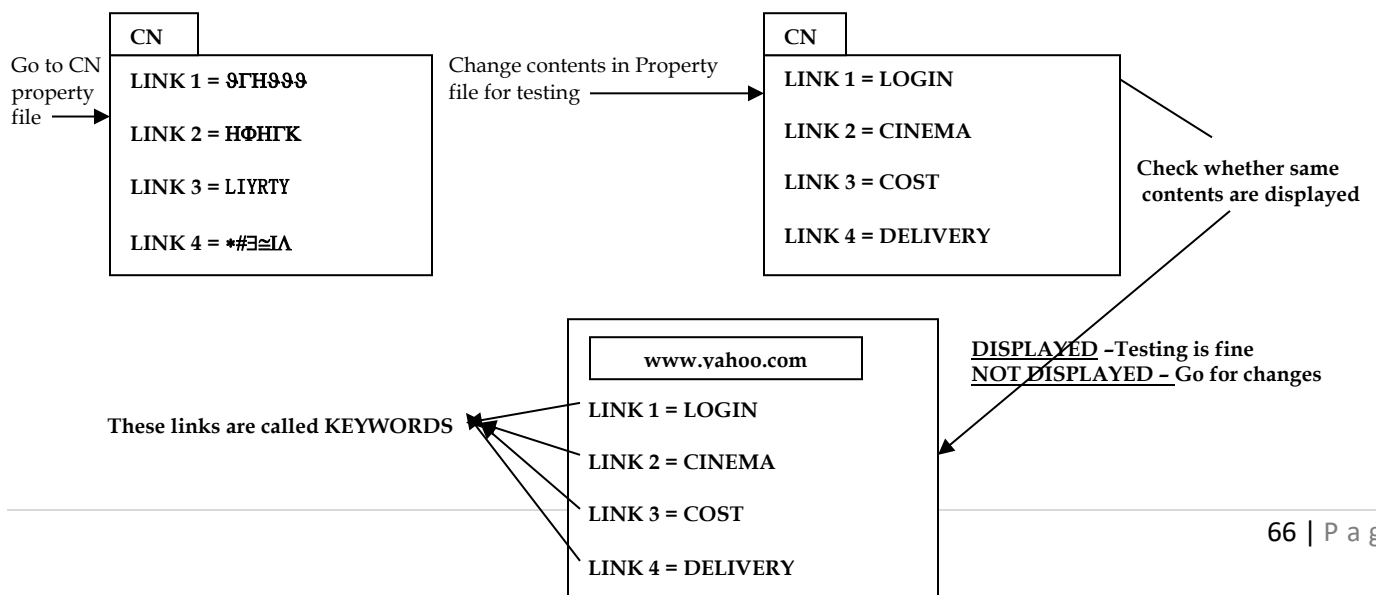


## SERVER



Now, we have changed the language from English to Chinese. Now, *how do we test whether it is in Chinese displayed or some other language ?*

Go to Chinese property file and change the contents in the file. **For ex**, see below figure,



After we do changes in property file, we once again select the Chinese language and see whether the change we have made in the content is displayed as same. If it is displayed, then testing is fine.

As a TE, we must do **changes in property file, and not in the program**(actual program).

If it is not displayed, catch the bugs and the bugs are to be fixed only in the program.

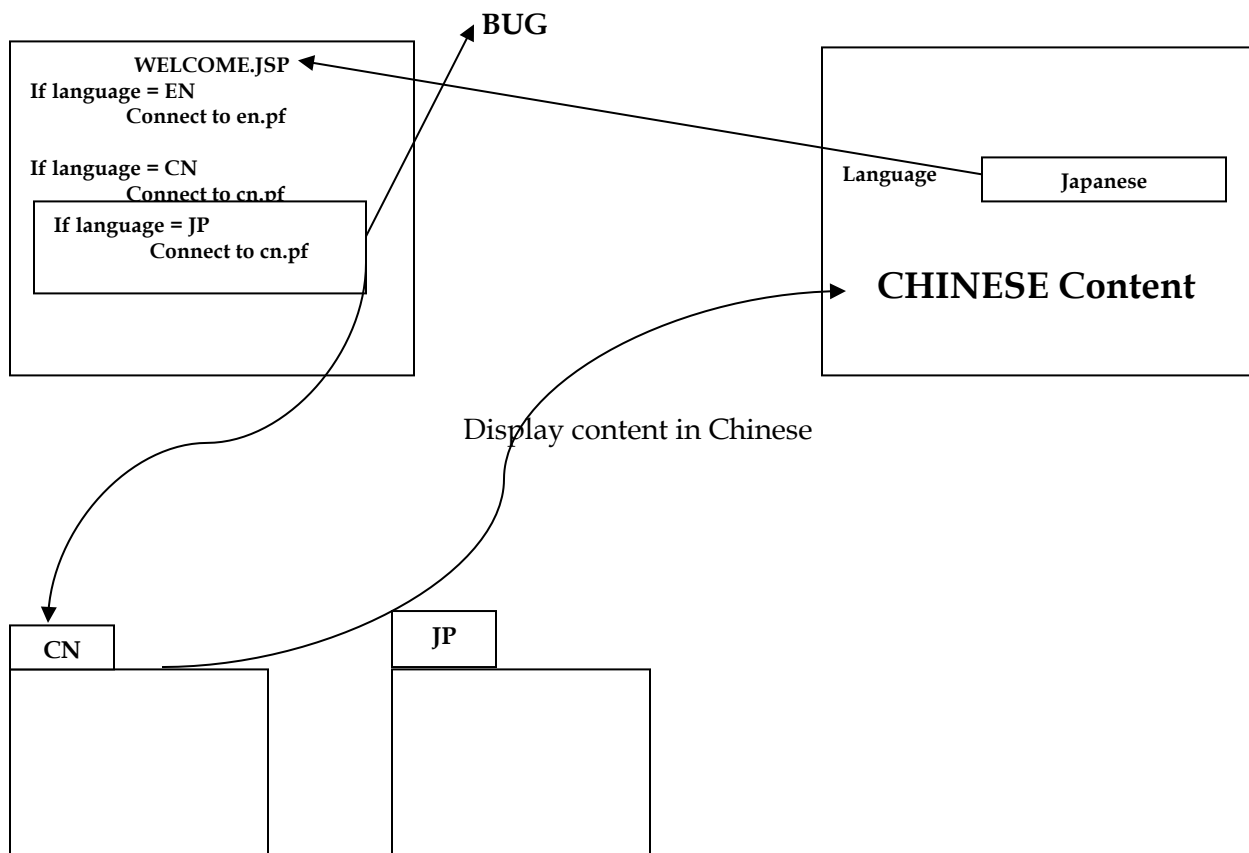
Thus, we do I18N testing for the following,

- Check whether content is in right language
- Check whether right content is in right place

Now, let us see the possible bugs in I18N testing :-

- 1) If I select English – displayed in English  
If I select Chinese – displayed in Chinese  
If I select Japanese – displayed in Chinese

Why does this happen ? – **observe,**



Why if I select Japanese, it is displaying in Chinese ?

Developers they do copy and paste the coding and forget to change the respective property file like for Japanese – instead of *connect to jp.pf* , they forget to change it and it still remains in the *connect to cn.pf* and thus displays the contents in Chinese language

- 2) Check for reverse direction language i.e, how the break works. There are 2 types of languages – Uni-directional language and Bi-directional language. Uni –directional language starts either from the left or right. Bi-directional language can start either from right or left. So we have to check if the text is displayed properly as per the language
- 3) Alignment Problem :- We have to check whether the alignment specification for different languages is followed properly. Right alignment or left alignment.

Now, let us do *globalization testing for online shopping for books* as shown below :

<http://www.onlineshopping.com/buybooks>

Language

ENGLISH

CHINESE

UKRAINE

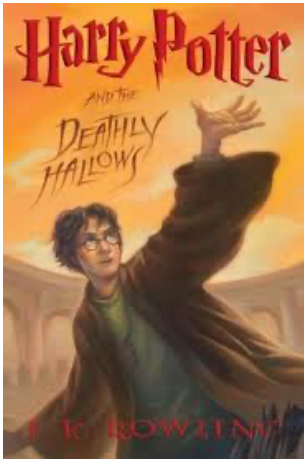
JAPANESE

TYPE : Novel

NAME : Harry Potter and The Deathly Hallows

AUTHOR : J.K.Rowling

PRICE : \$100



NAME :

CREDIT CARD NUMBER :

EXPIRY DATE :

ADDRESS :

PINCODE :  TELEPHONE :

BUY NOW

CANCEL

How do we connect it to other languages and test it ? – go to respective property files – change the contents and test it. If all the content is in right language, then I18N testing is successful.

### **Localization Testing ( L10N testing )**

Format testing is nothing but Localization testing (OR) Testing done for format specification according to region/country is called L10N testing.

*Let us see the different format testing we do in L10N testing,*

**a) Currency Format testing**

Here, we do not worry about the functionality of the format ( like \$ is converted to Rs or not ). We only test whether the \$ should be in the first or the last.

Ex : 100\$, \$100, Rs100. The standard should be as per country standards.

**b) Date Format testing**

Here, check whether the date format is according to its country format. This is also L10N testing.

Ex : in US, date format is : MM – DD – YY  
in India, date format is : DD – MM – YYYY

**c) PinCode Format testing**

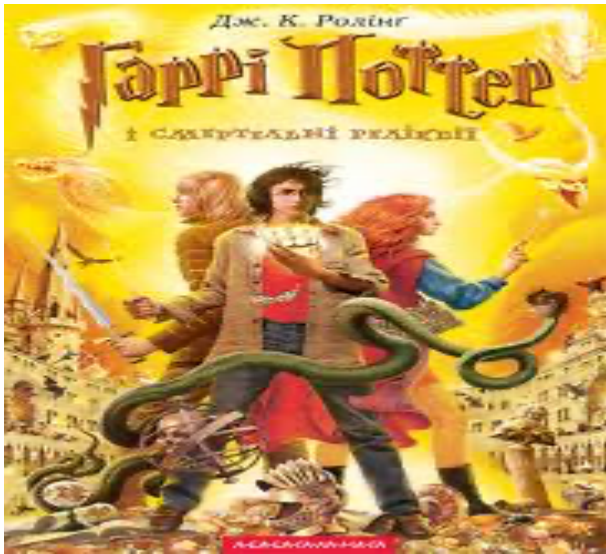
There are countries having Pincode with characters like AB100. Here, Checking for the format in pincode is L10N testing.

Checking whether AB is translated to Chinese is I18N testing.

Date format	}	<b>L10N testing</b>
Currency format		
Pincode format		

**d) Image Format testing**

In image – only name on the image can be changed – the image cannot be changed. Thus we must have multiple images depending on the country. When we click on Ukraine, it goes to Ukraine server, takes the contents and also selects Ukrainian image and thus contents are displayed in Ukraine. This is shown below,



For I18N testing – no images with text is allowed. In case, we want to have images – then we must have different images for different languages.

If we do any color change, then it is called as L10N testing (like national flag where the color is specific to its country).

**Tool tip** – ALT + TAB – move the mouse on the image – keep for 1second – we get a small box explaining the image – the tool tip must be changed to the corresponding language – if it does not, then it is a bug.

Now, we see a person from India sitting in China and wants to browse in English.

In this case, we have to go and change the locale which is available like below,

**http :// hp.cpm/sales.html locale = en\_US**

Language code -> en\_US ->country code

fr\_FR

de\_DE

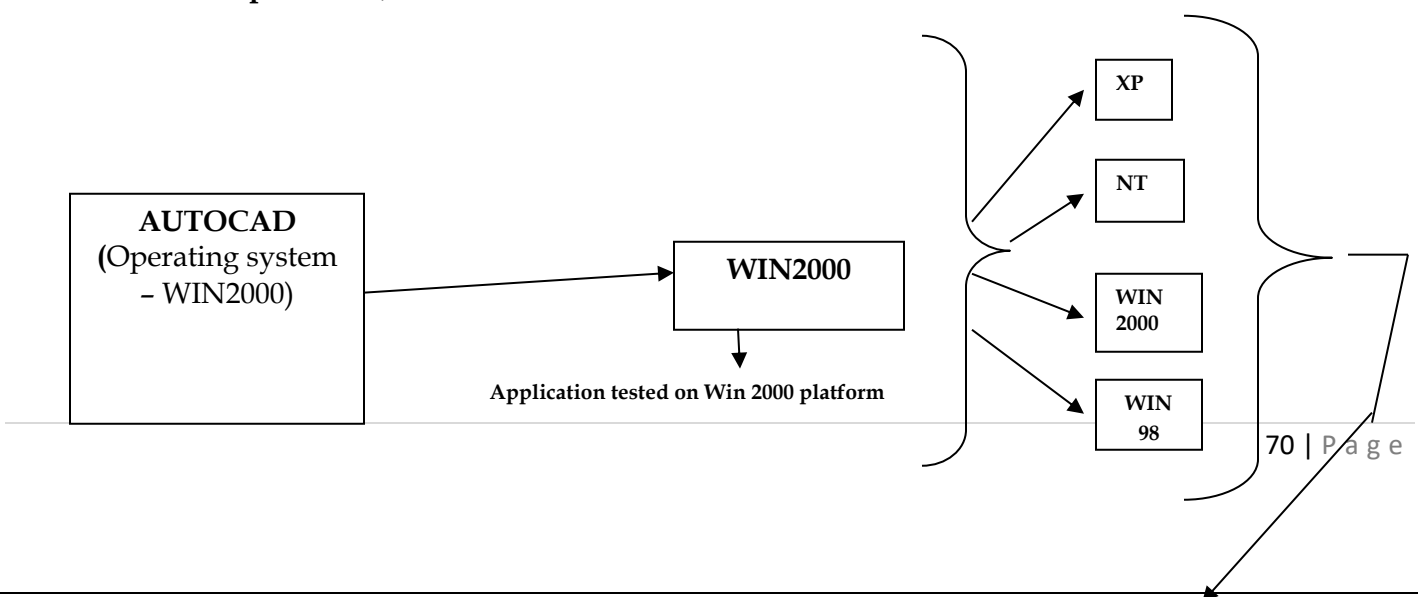
en\_DE

To be there in selected language – Go to Control Panel – change to language – come to browser – ctrl+shift (or) alt + Shift – change the text to that language.

### COMPATIBILITY TESTING (CT)

Testing the functionality of an application in different software and hardware environment is called Compatibility testing.

Consider an example below,



AutoCad with Win2000 was tested on Win2000 platform. Once it is tested, the product is sold but the customer who buys it uses in different platforms like XP, NT, Win2000, Win98 etc.

Customer therefore (or) End-users therefore blame the s/w, they don't think about the OS they are using because the application was tested on Win2000 and the user may be using it in XP.

Thus in order to overcome the above scenario, we do test the application in all platforms and then sell the product.

In the above example, we run the application in different platforms and if we find a bug in any of the platforms we have tested (say, Windows 98), then the s/w is given back to the developer for fixing the bug. Once, the bugs are fixed, then we have to continue testing not only on the platform which has bug, but also once again for all the platforms because fine tuning may cause a change in other platforms. Thus, maybe we may use one of the following scenarios after fixing the bug,

- a) if we are sure of the area of the bug, then test only that area.
- b) if we are not sure of the bug – then go for testing all the platforms.

Now, the question arises do compatibility testing is done for all platforms ? – NO. Observe how it is tested, Suppose only 8 TEs are present for testing, in this case –we don't have sufficient people for testing on all platforms. So do we stop testing ? – NO. Then how will we test – see below,

Now the TE will see for maximum use of platform by customers and do test for only those platforms.

PLATFORM
XP – 70 %
Win2000 – 20 %
Windows NT – 7 %
Windows 98 – 1 %
Windows ME – 1 %

In this example, TEs they look for most widely used platform i.e, in this case XP, Win 2000, NT, etc and do testing for all the above said platforms and do not test for the minimum used platform. If we are not testing other platforms doesn't mean it will not work. It will work on other platforms also, but if any bugs or issues arises – we are not fixing and not responsible for it. For this scenario to be avoided, sometimes in the product they mention for which OS it supports.

Now, we can think where we can do Compatibility testing ? – when we cannot force the users to use an application, then we go for CT.

Now, let us consider an example below to see where and how we do CT. (Figure in the next page)

If we take HP, IBM, SUN – these are h/w companies – they manufacture servers, when they want to sell the product – they can't sell the product only – therefore these companies have their own servers.

SOLARIS and UNIX can be tested by using OpenSource Internet Explorer ( IE )

By default, IE comes with OS. IE is always tightly embedded with OS, so we cannot remove OS from IE [if we try remove, may be the application may not work].

IE generates lots of revenue to the company as it is robust and has got loops of security in it.

Opera claims it is the fastest browser. IE is not Microsoft original product [it is actually got and combined with other applications]

Developer builds [www.shaadi.com](http://www.shaadi.com) and installs it in the testing server.

For the above application, (shaadi.com) - 4 Test Engineers are given 5days for testing. The testing is as follows by 1 engineer which is the same for the rest of the testing team,

1 TE opens OS - WIN 98 - Test for IE 5.0

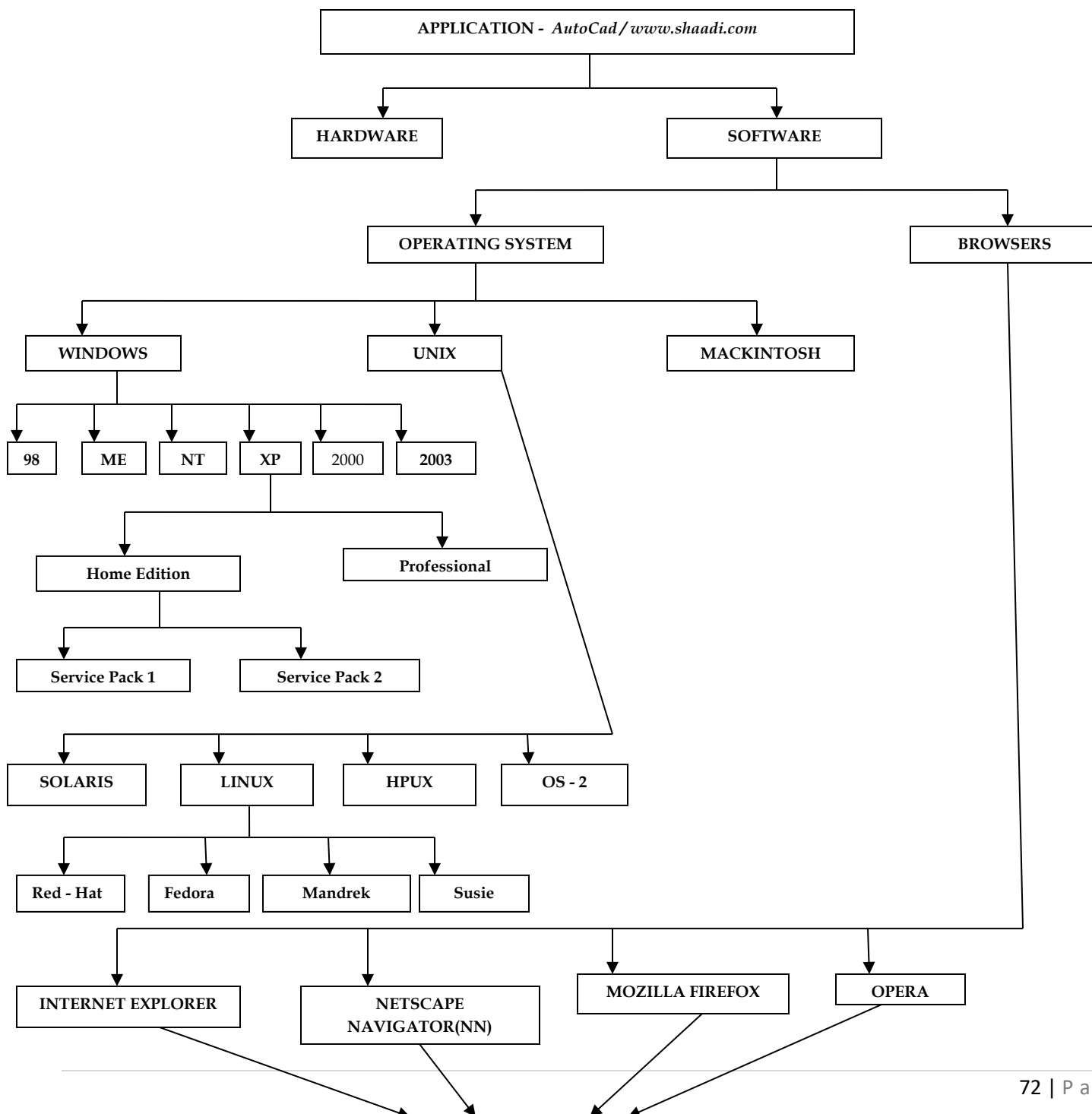
Opens OS - Win 98 - Test for IE 5.5

Opens OS - WIN 98 - Test for IE 6.0

Thus for 4 TE to test 1 OS and 8browsers within 5days, it takes 40days to test in 1 OS.

So far all OS to test  $40 \times 6 = 240$ , it takes 240days for the TE to test the application.

Like this list of all flavours of platform are available, but we have to test for the OS as browser which is in maximum use. Thus for Windows, we do test for WIN 2000 and Windows XP which is widely used





## DIFFERENT VERSIONS OF ALL THESE

Similarly Mackintosh is used by Apple computers and they sell in Europe and US.

Similarly for browsers we do test for the following cases which are widely used.

IE – for 5.5 and 6.0 – we do CT

NN – for 6.0 and 7.0 – we do CT and also for Mozilla Firefox we do CT

So, we always think about ROI (Return on Investment) and do the testing and also follow the criteria below,

- Number of users are less – we don't prefer CT
- Number of users are more – we prefer CT

WINDOWS 2000 Professional – normally corporate companies

Now, **let us see the set-up of the environment for AutoCad testing with OS**

Install Win 2000 in your machine and start testing for CT (1 of the platforms)

Similarly test for the next platform – IE (IE 5.5 and 6.0). open the IE 5.0 and test for it.

In this window, allow only 1 instance of IE i.e, 1<sup>st</sup> we should test for IE 5.5, remove and then allow many instances (i.e, to have both 5.5 and 6.0 at the same time)

After we finish testing for the 2 available most widely used platform, we move on to the next platform NN 6.0 and 7.0 – open NN 6.0. here, the window allows 2 instances i.e, to have both 6.0 and 7.0 at the same time.

### **How to handle bugs if we are testing for different platforms**

Testing functionality, integration and end-to-end testing on various platforms is what we do in Compatibility testing.

Compatibility issue – a feature not working in 1 OS, but working fine in all other OS. It happens because program written by developer is not working in only 1 platform, but working on all other platforms.

The compatibility issue is sent to developer for fixing the bugs and sent for testing. Here, to retest the application, we have to uninstall the OS and not the browser and re-install the OS and test.

Functionality issue – also called *Functionality defect issue* – a feature not working in all platforms/OS.

It is also possible to have different platforms in the same machine. For this, each platform(OS) should be in different drive like below. We can run 3 OS in same m/c in different drives – C: Windows 98, D: XP, E : Windows 2000

### **Hardware Compatibility Test**

- Test on different processors,
- Test on different RAM
- Test on different Motherboard
- Test on different VGA cards

In Test on different processors – we test for **make** – Intel, AMD processor and also test for **speed** – 3.1GHz, 2.7GHz

In Test on different RAM –we test for **make** – Samsung, Transient and also test for **size** – 1GB, 2GB

In Test on different Motherboard – we test for **make**

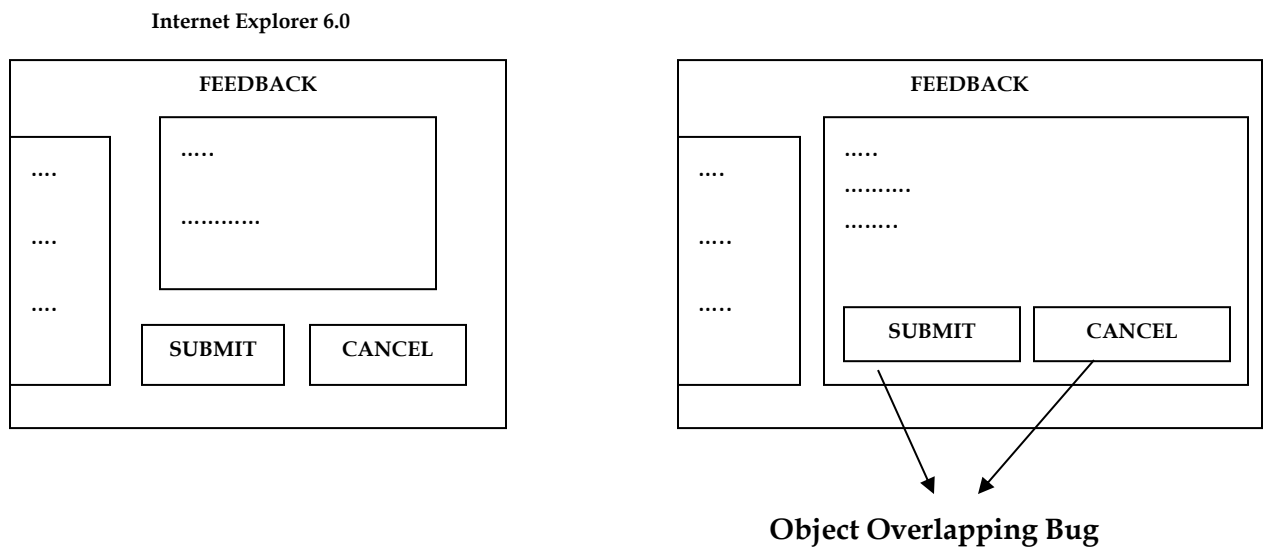
In Test on different VGA cards – we test for **make** – Nvidia, ATI

H/w compatibility test is not done for Web applications – because all programs run on the server and not on the local computers. We do h/w CT only for stand-alone applications.

*The various Compatibility bugs are,*

- Scattered content
- Alignment issues
- Broken frames
- Change in look and feel of the application
- Object overlapping
- Change in font size, style and color
- Object overlapping

**Object Overlapping**

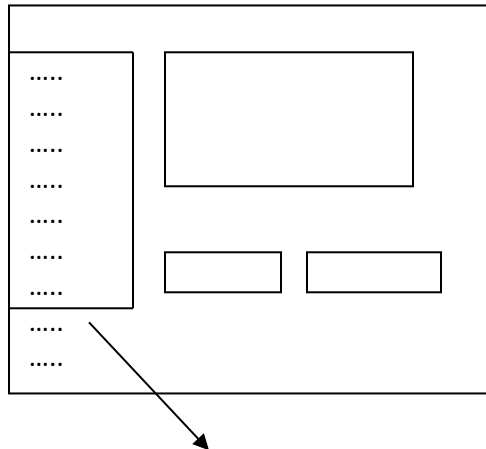


**Scattered Content**

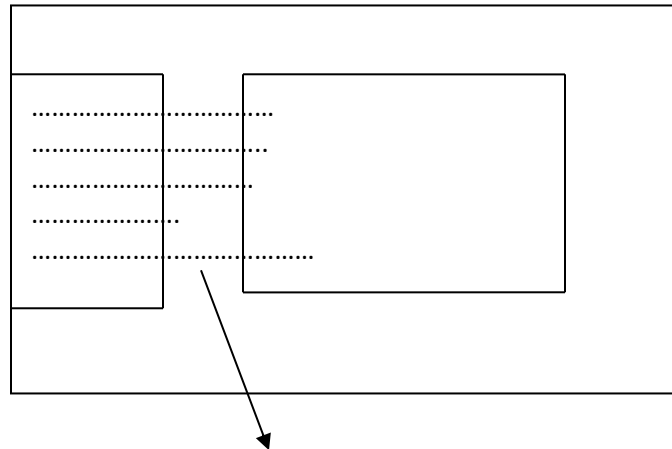
INBOX		
NAME	TIME	SUBJECT
A .... .... .... ....	....	.....
ATTACHED	SELECT	
YES .... .... ....	<input type="checkbox"/>	

All the fields (Name, Time, Subject, Attached, Select) must be in the same row. Instead, they are on different lines. This is called scattered content.

### Broken Frames



**Vertical Broken Frames**



**Horizontal Broken Frames**

Thus, we should develop the application in such a way that it works on many platforms.

### **INTERVIEW QUESTIONS**

**1) How do you test a Web Application? What are the types of test do you do on Web Application?**

**Ans ) Web Application means any dotcom – gmail, yahoo, etc**

**The types of test are,**

- **Functional Testing**
- **Integration Testing**
- **System Testing**
- **Compatibility Testing – Test in different OS, different browsers, different versions**
- **Usability Testing – check whether it is user friendly**
- **Accessibility Testing**
- **Ad- hoc Testing**
- **Smoke Testing**
- **Regression Testing**
- **Security Testing**
- **Performance Testing**
- **Globalization Testing – only if it is developed for multiple languages**

**2) Why we do Compatibility testing?**

**Ans) we might have developed the s/w in 1platform – chances are there that users might use it in different platforms – thus it could lead to defects and bugs – people may stop using the s/w – thus business will be affected and hence we do Compatibility Testing.**

## TEST CASES

<u>AMOUNT TRANSFER</u>	
....	From Account Number
	(FAN) <input type="text"/>
.....	To Account Number
.....	(TAN) <input type="text"/>
.....	Amount <input type="text"/>
.....	
	<input type="button" value="TRANSFER"/> <input type="button" value="CANCEL"/>

## SRS

### CBO – Online (CitiBank Online)

#### 1) Welcome Page

##### 1.1) Log in

1.1.1) Username text field should accept only 6-8 characters

1.1.2) Password text field should accept only 4-7 characters

#### 2) Loans

.....  
.....  
.....

.....  
.....  
.....  
.....  
.....

#### 70) Amount Transfer

##### 70.1) FAN Textfield

70.1.1) Should accept only 10 – digit integer

70.1.2) Should accept only those account numbers generated by Manager

##### 70.2) TAN Textfield

70.2.1) Should accept only 10 –digit integer

70.2.2) Should accept only those account numbers generated by Manager

##### 70.3) Amount Textfield

70.3.1) Should accept only the integers between 100 – 5000

70.3.2) Should not accept more than balance

.....  
.....  
.....  
.....

The testing quality depends on,

- Mood of the TE
- Testing is not consistent
- Varies from person to person

So, we write test cases.

*Test case is a document which covers all possible scenarios to test all the feature(s).*

It is a set of input parameters for which the s/w will be tested. The SRS are numbered so that developers and testing team will not miss out on any feature.

#### *When do we write test cases?*

Customer gives requirements – developer start developing and they say they need about 4 months to develop this product – during this time, testing team start writing test cases – once it is done, they send it to test lead who reviews it and adds some more scenarios – developers finish developing the product and the product is given for testing – the test engineer then looks at the test cases and starts testing the product – the TE never looks at the requirements while testing the product – thus testing is consistent and does not depend on the mood and quality of the test engineer.

*The list of values derived to test the Amount text field is – Blank, -100, hundred, 100, 6000, Rs100, \$100, \$+?, 0100, Blankspace100, 100.50, 0, 90*

When writing test cases, actual result should never be written as the product is still being developed. Only after execution of test cases should the actual result be written.

#### *Why we write test cases?*

- **To have better test coverage** – cover all possible scenarios and document it, so that we need not remember all the scenarios
- **To have consistency in test case execution** – seeing the test case and testing the product
- **To avoid training every new engineer on the product** – when an engineer leaves, he leaves with lot of knowledge and scenarios. Those scenarios should be documented, so that new engineer can test with the given scenarios and also write new scenarios.
- **To depend on process rather than on a person**

Let's say a test engineer has tested a product during 1<sup>st</sup> release, 2<sup>nd</sup> release and has left the company for the 3<sup>rd</sup> release. As this TE has mastered a module and has tested the application rigorously by deriving many values. If that person is not there for 3<sup>rd</sup> release, it becomes difficult for the new person – hence all the derived values are documented, so that it can be used in feature.

When developers are developing the 1<sup>st</sup> product (1<sup>st</sup> release), TE writes test cases. In the 2<sup>nd</sup> release, when new features are added, TE writes test cases. In the next release, when features are modified – TE modifies test cases or writes new test cases.

#### Test Case Design Techniques are,

- Error Guessing
- Equivalence Partitioning
- Boundary Value Analysis (BVA)

#### Error Guessing :

Guessing the error. If the Amount text field asks for only integers, we enter all other values, like – decimal, special character, negative etc. Check for all the values mentioned above.

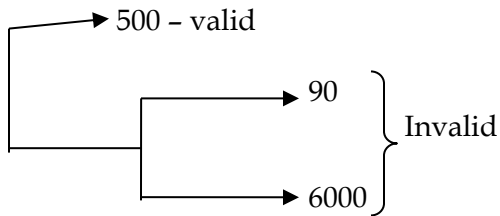
#### Equivalence Partitioning



According to Pressman,

1) If the input is a range of values, then design the test cases for 1 valid and 2 invalid values.

For ex, Amount text field accepts range of values

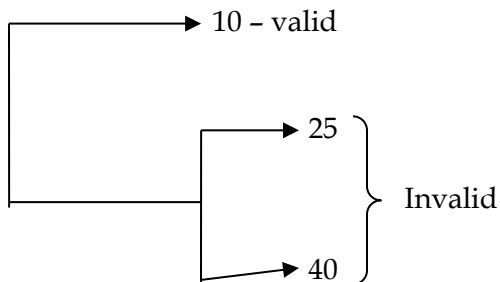


2) If the input is a set of values, then design the test cases for 1 valid and 2 invalid values.

**ONLINE SHOPPING**

Product ID

Printer = 10 ; Scanner = 20 ; Mouse = 30 ;



3) If the input is Boolean, then design the test cases for both true and false values. Ex - checkboxes, radiobuttons etc.

☐ Male

☐ Female

*In PRACTICE, we do the following,*

Testing the application by deriving the below values,

90      100      1000      2000      3000      4000      5000      6000

Lets see a program. Understand the logic and analyse why we use Practice method,

```
If (amount <100 or >5000)
{
    Error message
}
If (amount between 100 & 2000)
{
    Deduct 2%
}
If (amount > 2000)
{
    Deduct 3%
}
```

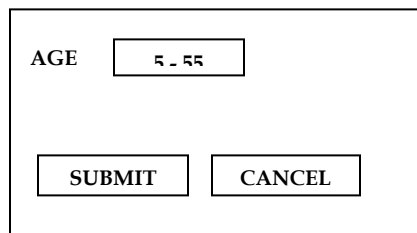
When Pressman techniques are used, the first 2 programs are tested, but if Practice method is used, all these are covered.

It is not necessary that for all applications, practice methodology needs to be used. Sometimes, Pressman is also fine.

But, if the application has any deviation, splits or precision – then we go for Practice method.

If Practice methodology has to be used, it should be – **a) Case specific      b) Product specific**

**c) Number of divisions depends on the precision (2% or 3% deduction)**



Here, Pressman technique is enough to test for Age text field (1 valid and 2invalid)

But, if the Age text field is for insurance (10years and above compulsory and different policies for different age groups) – then we need to use Practice method. Depending on this, divisions of values are done.

### **BVA - Boundary Value Analysis**

If input is a range of values between A – B, then design test case for A, A+1, A-1 and B, B+1, B – 1.

Thus, a number of bugs can be found when applying BVA because developer tends to commit mistakes in this area when writing code.

```
If ( Amount <= 100 )  
{  
    Throw error  
}  
If ( Amount >= 5000 )  
{  
    .....  
}
```

If 'equals' is there, then even 100 value is expected.

When comparing Equivalence Partitioning and BVA, testing values are repeated – if that is the case, we can neglect Equivalence Partitioning and perform only BVA as it covers all the values.

### **INTERVIEW QUESTIONS**

**1) What are test case design techniques ?**

**Ans) Explain about Error Guessing, Equivalence Partitioning ( Pressman only ) and BVA.**

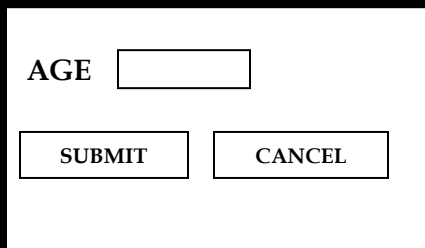
***If they ask more, only then talk about Practice method***

**2) What are Testing techniques / Black Box techniques / Testing methodologies / Testing types ?**

**Ans) First talk about WBT and BBT.**

***If the interviewer is not satisfied with the answer, then immediately start answering about test case design techniques.***

**3)**



AGE

***How do you test the above field ?***

**Ans) Use test case design techniques and derive the values.**

***I'll test the text field by using these test case design techniques – Error Guessing, Equivalence Partitioning and Boundary Value Analysis***



## *Interview Questions (continued)*

4)

A login form with the following elements:

- A label "Username" followed by a text input field.
- A label "Password" followed by a text input field.
- A checkbox labeled "Remember Password".
- Two buttons at the bottom: "LOGIN" and "CANCEL".

*How do you test the above fields ?*

*Ans) For any application like this, we must always start testing with end-to-end scenarios.*

- *Enter valid username and password – goes to inbox? – logout*
- *Enter valid username and password – remember password – goes to inbox – logout*
- *Open gmail in the browser – username and password should always be there – click on login button – goes to homepage*
- *Change password – now logout and open gmail in the browser – old password should be there – should not login – remove remember password – type in new password – login*
- *Open gmail – type in new password – remember it*
- *Try and open in both Internet Explorer and Mozilla FireFox simultaneously – what happens depends on the requirements*

*And then start testing for CANCEL button*

*After all this, if he asks for more scenarios – then go for functional testing for each text field*

5) *How to test a pen ?*

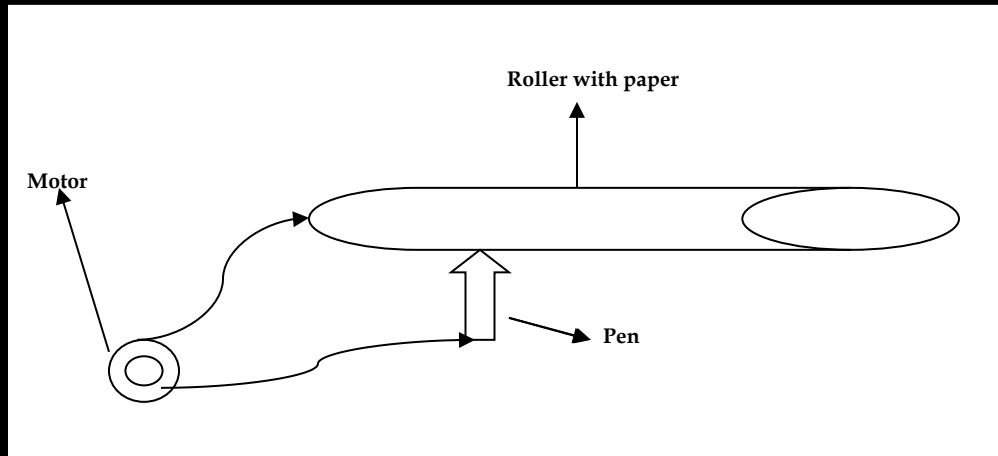
*Ans) we can do both Manual and Automation testing. 1<sup>st</sup> let us see how to do Manual Testing,*

- *Functional Testing – each part as per requirement – refill, pen body, pen cap, pen size*
- *Integration Testing – Combine pen and cap, and integrate other different parts and see whether they work fine*
- *Smoke Testing – basic functionality – writes or not*
- *Ad-hoc Testing – throw the pen down and start writing, keep it vertically up and write, write on the wall*
- *Usability Testing – whether user friendly or not – whether we can write it for longer periods of time comfortably*
- *Compatibility Testing – different environments, different surfaces, weather conditions – keep it in oven and then write, keep it in freezer and write, try and write on water*
- *Performance Testing – writing speed*
- *Recovery Testing – throw it down and write*
- *Globalization Testing – I18N testing – whether the print on the pen is as per country language and culture – L10N testing – price standard, expiry date format*
- *Reliability Testing – drop it down and write, continuously write and see whether it leaks or not*
- *Accessibility Testing – usable by handicapped people*

### *Interview Questions (continued...)*

Now, we will see how we can do Automation Testing on this pen :-

Take a roller. Now put some sheets of paper on the roller. Connect the roller onto a motor. Connect the pen to the motor and switch on the motor. The pen starts scribbling on the paper. Once the pen has stopped writing, now see how many pages it has written , number of lines on each page, length of each line and multiplying all this – we can get for how many kilometers the pen can write.



### FUNCTIONAL Test Cases

We have already shown the template for writing test cases.

The main intention of a TE is to write test cases effectively and also efficiently.

Here, the actual result is written after execution of the test cases and most of the time it would be the same as the expected result. It will be different if only the test step is failed. So, actual result field can be skipped, and we can elaborate about the bug in comments field.

Also, the input field can be removed and these details can be added in description field.

Its not that the above mentioned template is a standard one, the template can vary in each company and also with each application depends on TE and Test Lead. But, for testing 1 application, all TEs should follow a standard template which is formulated.

Test cases should be written in such a way that even if a new TE should/can understand and execute the same.

When entering URL hostname, hostname is mentioned as development team throws the s/w to testing team after say 4months – so we are not sure of the hostname.

Username and password are not mentioned as they keep changing. The user might need to change it every 15days depending upon the application.

When writing functional test cases - we 1<sup>st</sup> check for which field we write test cases and then elaborate accordingly. If say, amount transfer is the field we are writing FT, then elaborate this is not login feature. Input is required in FT, if the application is data driven, then i/p column is required else it is time consuming.

### ***Lessons for writing Functional Test Cases***

- Before we start writing test case, come up with options and select the best option and then only start writing test case
- In Expected result, use 'should be' or 'must be'
- Elaborate only those steps on which we have focus. Do not elaborate on all the steps.
- Highlight object names
- Do not hard code the test case. Write a generic test case
- Organize steps properly so that it reduces lot of execution time.

### **INTEGRATION Test Cases**

Something which is covered in FT case should not be written in IT case and something written in IT case should not be written in ST case.

Always start any test case with navigation steps to help new user understand it.

Strategy to write Integration test cases,

- Understand the product
- Identify the possible scenarios
- Prioritize
- Write the test cases according to priority

When TE is writing test cases, the factors he needs to consider are,

- If the test cases are in detail
- Coverage, all test case values or scenarios are depicted
- Think in execution point of view
- Template used to write test cases must be unique

Best test case is when less number of steps are involved, coverage is more, when these test cases are thrown to anyone, they need to understand.

### **SYSTEM Test Cases**

Consider the scenario given in **Pg 35 and Pg 36**. Let us write system test cases for the end-to-end business flow. The basic scenario is shown in the above pages.

## **Header of Test Case :**

We always fill the body of the test case first before filling up the header of the test case.

### **1) Test case name :**

*CBO\_AT\_more than balance*

Project name                      Module name                      Scenario to be tested

### **2) Requirement number :**

*32.3 Amount Transfer*

### **3) Module name :**

*Amount Transfer*

### **4) Pre-condition :**

*Test Engineer should know the balance of user A and user B*

Pre-condition is a set of actions or settings that you should have done before executing step number 1.

**For ex,** If in an application, we are writing test cases for add users, edit users and delete users – the precondition would be – see if user A is added before editing it and deleting it.

### **5) Test data :**

The data we should have before executing step number 1. **Ex** – username, password, account number of users.

The test lead may give the test data like username or password to test the application. Or the TE may himself generate the username and password.

### **6) Severity :**

It can be *major, minor* or *critical*.

To analyse the severity of the test case, it depends on the header.

Choose the severity according to module. If in the module, there are many features. In that, even if 1 feature is critical, we claim that test case to be critical. It depends on the feature for which we are writing the test case.

In Gmail,

Compose mail -> Critical

Sent items -> Minor

Feedback -> Major

In a banking application,

Amount transfer -> Critical

Feedback -> Minor

We write severity because we can prioritize our execution based on the severity of the feature.

### **7) Test case type :**

It can be functional test cases or integration test cases or system test case or positive or negative or positive and negative test cases.

## 8) Brief description :

*Following test case ensures that Amount Transfer feature is not allowing more than balance.*

Test engineer has written test case for a particular feature. If he comes and reads the test cases for a moment, he will not know for what feature has written it. So, this gives a brief description of for what feature test cases are written.

**Step number** is important. If say, step number 10 is failing – we can document defect report and hence prioritize working and also decide if it's a critical bug.

### *Header format of a Test Case*

<b>Test Name :</b> CBO_AT_more than balance
<b>Requirement Number :</b> 70.3 (in SRS document, amount transfer is numbered)
<b>Project Name :</b> Citibank Online
<b>Module Name :</b> Amount Transfer
<b>Severity :</b> critical (depends on feature we are testing)
<b>Pre-condition :</b> 1)Test engineer should know the balance amount of user A & B. 2) execution of xyz test case
<b>Test Case Type :</b> Functional testing, Integration testing, System testing
<b>Test Data :</b> Username and password of user A & B, account number
<b>Brief Description :</b> briefly explains the scenario

### Footer of a Test Case :

- 1) **Author :** Who wrote this test case
- 2) **Reviewed by :**
- 3) **Approved by :**
- 4) **approval date :**

### Test Case Review :

#### *Test Case Review process / Peer review process :*

Customer gives requirements – development team start developing the product looking at the requirements – testing team start writing test cases looking at the requirements. Test engineer (you) are writing test cases for a particular module based on the requirements. Once all the possible test cases have been written for that particular module, you send a mail to the Test lead saying that you have finished writing test cases for that module. Now, what the test lead does is – he tells someone in the same testing team to review your test cases. The reviewer reviews all your test cases looking at your module's requirements and in case of any mistakes sends it to you and also to your test lead. You correct all the mistakes and send a copy of the corrected test cases both to the test lead and to the reviewer. It need not be that all mistakes pointed out by the reviewer be correct, if you feel they are wrong, then you need to give proper justification as to why your test cases are correct. Once the reviewer says all the test cases are fine, he sends a mail to the test lead saying

all the test cases are fine. The test lead then approves your test cases and sends an approval mail to you saying that all the test cases are fine and to start executing the test cases.

While reviewing, the reviewer checks the following,

1) **Template** - he checks whether the template is as per decided for the project

2) **Header** :

- a) Checks whether all the attributes are captured or not
- b) Checks whether all the attributes in the header are filled or not
- c) Checks whether all the attributes in the header are relevant or not

3) **Body** :

- a) Check whether all possible scenarios are covered or not
- b) Check whether the flow of test case is good or not
- c) Check whether the test case design techniques are applied or not
- d) The test cases should be organized in such a way that it should less time to execute
- e) Check whether the test case is simple to understand and execute
- f) Check whether proper navigation steps is written or not

Once test cases are reviewed, the review comments should not be sent via email or in notepad. The **test case review template** is shown below,

**TEST CASE REVIEW TEMPLATE**

Test Case Name	Step No.	Reviewer		Author	Comments
		Comments	Severity		
CBO_AT_more than balance	Pre-condition	Pre-condition is missing	Major		not fixed. Give justification saying pre-condition is not needed for this
	8	Click on "Back button" step is missing	minor		fixed

CBO_Insurance_Age field validation (5 - 55)	10	Insufficient test coverage. Apply test case design techniques and derive the following values - 56, 4, 10years, 10.5	critical		fixed
---	----	--	----------	--	-------

Reviewer will use the above template and send the comments. If the author fixes the test case, he would report as fixed. If he feels that the test case he has written is correct, he will not fix them – but needs to give proper justification for it.

### Interview Tips

*In interview, when the interviewer asks “how do you review a test case and what do you review in a test case?”*

*Always answer should start with – Body of test case, then header and finally template.*

**VERY VERY IMPORTANT !!**

### INTERVIEW QUESTIONS

*1) What is the duration of your current project ?*

*Ans) 8months – 1.5years. Whatever projects you put, be prepared to answer about any project. Always tell – “by the time i joined, 2major releases were over. I joined the project during the 3<sup>rd</sup> release and I have spent around 8months here”.*

*2) Totally, in your current project, how many screens (features) are there ?*

*Ans) an average complex application will have about 60 – 70 screens. A simple application like ActiTime has around 20 – 30 screens. So tell about 60-70 screens.*

*3) Totally, how many test engineers are there in your current project ?*

*Ans) For 70 screens, 10 – 15screens / engineer.  $70/15 = 5$ engineers. So, you can tell anywhere between 3 – 8 test engineers.*

*4) Totally in your current project, how many test cases are there in your current project ?*

*Ans) For 1 screen – you can write 10 – 15test cases (includes FT, IT, ST, positive and negative scenarios)  $70 * 15 = 1050$ . You can tell anywhere from 800 – 1050 – 1200 test cases.*

*Test case means 1 entire document with header, footer and many scenarios.*

### Interview questions (continued..)

g e

*5) Totally, how many test cases you have written in your current project ?*

*Ans) This includes for all 3releases. You have joined in 3<sup>rd</sup> release. In 2releases, they would have written around 700test cases(worst case – 650test cases). You wrote in 3<sup>rd</sup> release test cases, so  $550/5 = 110$ . You can tell anywhere between 80 -110 test cases (maximum of 240 also)*

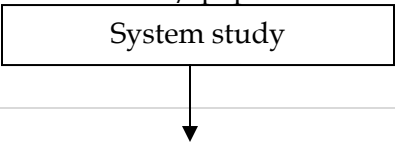
**Procedure to write the test cases :-**

*System study* – understand the application by looking at the requirements or SRS given by the customer.

*Identify all scenarios :*

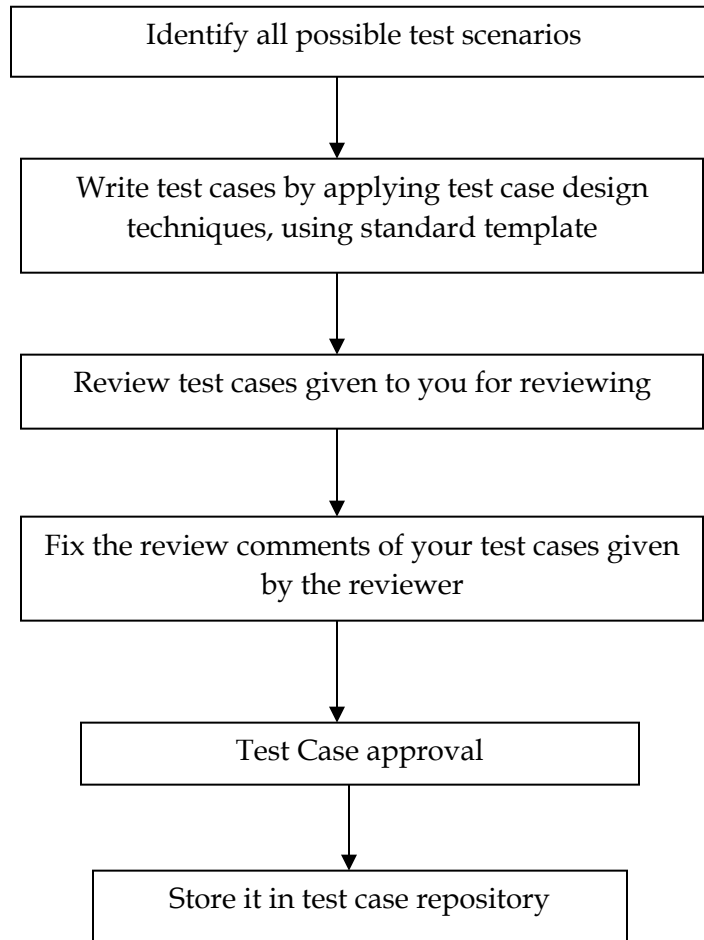
- 1) When the product is launched – what are the possible ways the end user may use the s/w. Identify all possible ways
- 2) What and all possible business flows are there
- 3) Document all possible scenarios in a document / paper – it is called test design/high level design.

System study



```
graph TD; A[System study] --> B[ ]
```





Test design is a record having all the possible scenarios.

4) Brainstorming session

5) Measure the efficiency of brainstorming session

*Write test cases :*

1) Convert all the identified scenarios to test cases

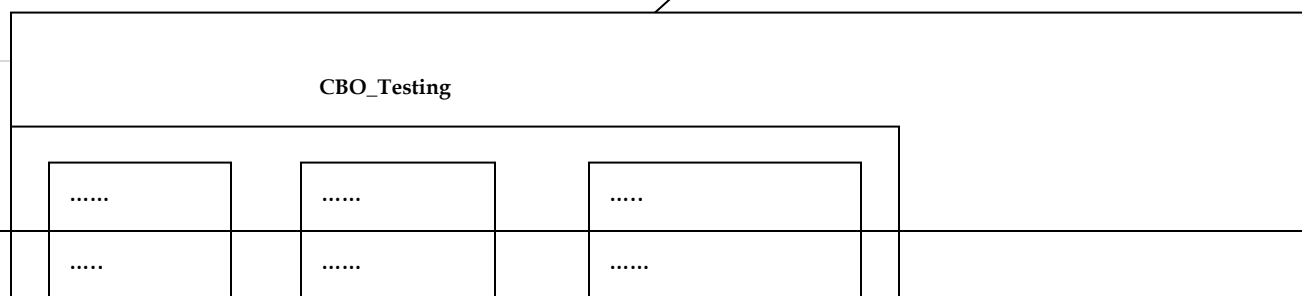
- Group scenarios related to 1feature
- Prioritize
- Write test cases

2) When converting, apply test case design techniques

3) Use standard test case template - standard means the one decided for the project.

**Test Case Repository :** (TCR)

QA



TCR is a shared folder. For security, they keep the entire folder in version control tool(VCT).

Customer gives requirements, developers are developing features and test engineers are writing test cases looking at the requirements. The test cases which are approved by the test lead are stored in a test case repository. When test cases are needed for execution, then test engineers will *check in* and retrieve their respective test cases. After execution, the test engineers then *check out* of the TCR. When any test cases are not needed, it is *dropped* from the TCR.

Always, the testing team keep taking back-up of the entire TCR folder to prevent any crashes from affecting the project.

QC(quality center) is used to store all test cases or the test cases might be stored in Test Link which is a test management tool.

#### **Procedure to execute test cases :**

Customer gives requirements – developers are developing the features looking at the requirements. The test lead gives a list of features for each TE to write test cases and execute them. The TEs first understand the product by looking at the requirements and then start writing test cases.

Now, let us consider that TE1 has been given loans feature, TE2 has been given insurance feature and TE3 has been given Amount Transfer feature. Now, all these TEs start writing test cases for their respective features. After the test cases have been written, reviewed and approved – they are stored in the test case repository.

Now, by this time – developers have given the 1<sup>st</sup> build – in 1<sup>st</sup> build, features which have been developed are – Personal loans(half), life insurance(half) and Amount balance. Whenever developer gives builds – create another folder(Tiger\_Test Execution Result) – create in that folder B01(1<sup>st</sup> build) – copy,paste all test cases from QA – Build1 comes – remove all test cases for the features which are not yet developed – keep only those test cases for which the features have been developed i.e, the relevant test cases – then do smoke testing and all other testing and then fill in the Status columns of the Test Cases – bugs are caught and sent to the development team.

After this, developers give the 2<sup>nd</sup> build B02 – here, the features developed are – complete personal loan, complete amount transfer and also complete life insurance – create a folder named B02 – copy, paste the entire test cases in that folder – and the same method as in B01 follows.

Same procedure follows all Builds. Thus we have all the results and can check the results for all the builds.

Build comes in – execute all relevant test cases – by the end of the cycle, summary report must be filled. This summary report is known as *Test Execution Report / Test Summary Report*.

**TEST EXECUTION REPORT (Build1)**

Module Name	Total Test Cases	Total Executed	Total Pass	Total Fail	%Pass	%Fail
Loans	420	120	96	24	80%	20%
Insurance	500	200	180	20	90%	10%
Amount Transfer	400	100	95	5	95%	5%

<b>Total</b>	<b>1320</b>	<b>420</b>	<b>371</b>	<b>49</b>	<b>88%</b>	<b>12%</b>
--------------	-------------	------------	------------	-----------	------------	------------

This report is called ,

***Tiger - B01 - Test Execution Report / Test Summary Report***

Test Lead prepares this report. The TE sends the individual features which he has tested and tells how many he has executed and all that statistics.

The Test Lead then sends the report to,

- Test Manager
- Development Team
- Management
- Customer (depends on whether the project is a '**fixed bid**' or '**time & material bid**').

If it is a 'time and material' bid, then the test execution report must be sent to the customer as well.

The development team needs,

- A list of test cases that are failed
- Each developer needs a list of test cases which are failed for his features.

In each sheet is a list of test case names and everything as shown below,

***LOANS feature***

Step No.	Test Case Name	Status	Comments
1	....	Pass	....
2	....	Pass	....
3	....	Fail	....
4	.....	Fail	....
5	.....	Fail	....
6	....	Not Executed	....
7	....	Not Executed	....
....	....	....	....
....	....	....	....
...	...	...	....
420	...	....	....

The developer knows only the test case name. then he directly goes to the test case through a link or directly and then sees which test case is failed.

The Test Execution Report is stored in the B01 (respective builds) outside all folders.

For compatibility testing, the Test Execution Report looks something like this,

Step No.	Test Case Name	Windows XP		Windows 7		Windows Vista	
		Status	Comments	Status	Comments	Status	Comments
1	....	Pass	...	Pass	...	Pass	....
2	....	Pass	....	Pass	....	Pass	...
3	...	Pass	...	Pass	...	Pass	...
...	...	...	...	...	...	...	...
...	....	...	...	...	...	...	...
..	....	...	...	...	....	...	...
7	...	Fail	...	...	...	Pass	...
...	....	Fail	...	...	...	Pass	...
237	...	Fail	...	...	...	Fail	...

Again, we can do compatibility testing for browsers on various platforms,

WINDOWS XP					
Internet Explorer		Mozilla FireFox		Opera	
Status	Comments	Status	Comments	Status	Comments

### **REGRESSION TESTING :**

*1<sup>st</sup> build* – Customer gives requirements – development team start developing features – testing team start writing test cases – testing team write about 1000 test cases for the 1<sup>st</sup> release of the product and after execution of the test cases – the product is released – customer does acceptance testing – and the product is moved to production.

*2<sup>nd</sup> build* – now, customer asks for 2extra features to be added and gives the requirements for the extra features – development team start building the extra features – testing team start writing test cases for the extra features – about 200extra test cases are written – thus a total of 1200 test cases are written for both the releases – now testing team – start testing the new features using the 200 new test cases – once that's done, then start testing the old features using the old 1000 test cases to check if adding new features has broken the old features. Testing old features is called regression testing. Once everything has been tested, now the product is given to the customer who does acceptance testing and then moves the product to production.

*3<sup>rd</sup> build* – after the 2<sup>nd</sup> release, the customer wants to remove one of the features (say Loans) – he removes all the *Loans* related test cases (about 100) – and then tests all the other features to check if all the other features are working fine. This is called regression testing.

### ***Talk 1***

Testing the unchanged features to make sure that it is not broken because of the changes (changes means – addition, modification, deletion or defect fixing)

## Talk 2

Re-execution of same test cases in different builds or releases to make sure that changes (addition, modification, deletion or defect fixing) are not introducing defects in unchanged features.

*When the development team gives a build, chances are there they would have done some changes. That change might affect unchanged features. So, Testing the unchanged features to make sure that it is not broken because of the changes is called Regression Testing.*

Majority of time spent in testing is on regression testing.

Based on changes, we should do **different types of regression testing**,

- Unit Regression Testing
- Regional Regression Testing
- Full Regression Testing

### a) Unit Regression Testing (URT)

Here, we are going to test only the changes.

In Build B01, a bug is found and a report is sent to the developer. The developer fixes the bug and also sends along some new features developed in the 2<sup>nd</sup> build B02. The TE tests only if the bug is fixed.

For ex,

**CREATE USER**

Name

Address

Telephone Number

Email Id

....

....

....

....

When developer gives the above application for testing in the 1<sup>st</sup> build – the TE finds that clicking on the **submit** button goes to a blank page – this is a bug and is sent to the developer for defect fixing – when the new build comes in along with the defect fixes – the TE tests only the **submit** button. Here we are not going to test the other features of the 1<sup>st</sup> build and move to test the new features sent in the 2<sup>nd</sup> build. We are sure that fixing the **submit** button is not going to affect other features – so we test only the fixed defect.

*Testing only the modified features is called Unit Regression Testing.*

Let us consider another **example**,

(Search field) 1 - 20 characters

SEARCH

CANCEL

**Build 1 - B01**

(Search field) 1 - 40 characters

SEARCH

CANCEL

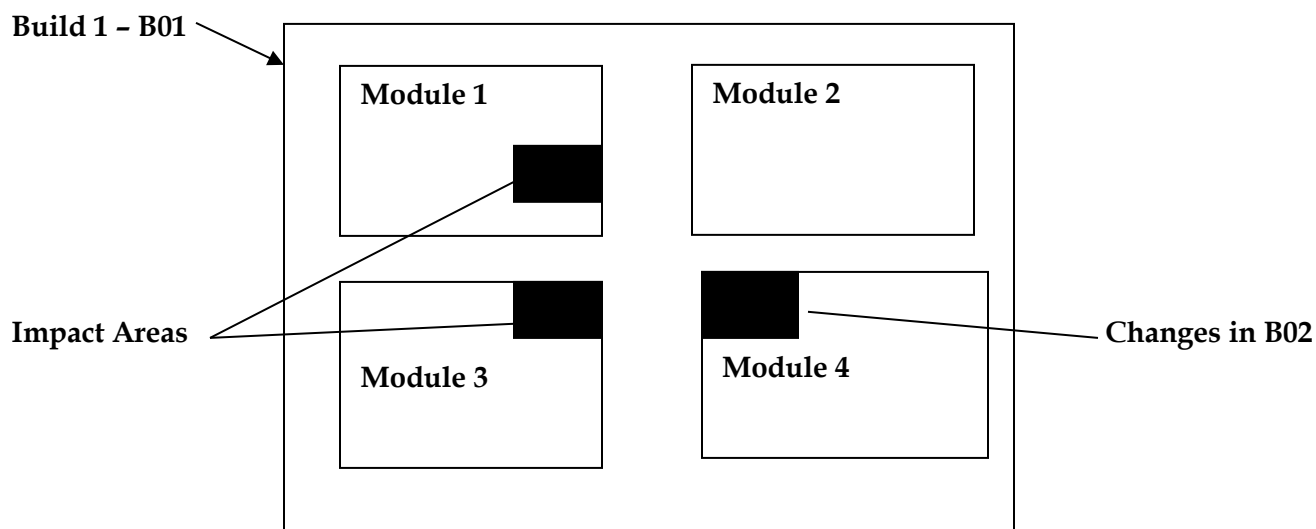
**Build 2 - B02**

For the above application, in the 1<sup>st</sup> build - the developers develop a “**search**” field which accepts 1-20 characters. The TE test the search field using test case design techniques.

Now, the customer makes some changes in the requirements and requests that the “**search**” field be able to accept 1-40 characters. The TE tests only the search field to see if it accepts 1-40 characters and doesn’t test for any other feature of the 1<sup>st</sup> build.

#### b) Regional Regression Testing (RRT)

Testing the changes and impact regions is called Regional Regression Testing.



The module 1,2,3,4 is given by developers for testing during the 1<sup>st</sup> build. The TE finds a defect in Module 4. The defect report is sent to the developers and the development team fixes the bug and sends the 2<sup>nd</sup> build in which the bug is fixed. Now, the TE realizes that defect fixing in module 4 has impacted some features in module 1 and 3. So, the TE first tests module 4 where the bug has been fixed and then tests the impact areas i.e, module 1 and module 3. This is known as *regional regression testing*.

#### **Story 1**

After the 1<sup>st</sup> build, the customer sends some changes in requirement and also to add new features to be added to the product. The requirements are sent to both development team and testing team.

The development team starts making the changes and also building the new features as per the requirements.

Now, the test lead sends a mail to the customer asking – which and all are the **impact areas** that will be affected after the necessary changes are made – so that he will get an idea as to which and all features needed to be tested again. He also sends a mail to the development team to know which and all areas in the application will be affected as a result of the modifications and additions of features. And similarly he sends a mail to his testing team for a list of impact areas. Thus he gathers **impact list** from the customer, development team and also the testing team.

This **impact list** is sent to the all testing engineers who look at the list and check if their features are modified and if yes they then they do regional regression testing. The **impact areas** and changed areas are all tested by the respective engineers for whom the features are allotted. Each TE tests only his features which could have been affected as a result of the changes and modifications.

The problem with the above method is that the test lead may not get the full idea of the impact areas because the customer and development team may not have so much time to respond to his emails.

### Story 2

To solve the above problem (story 1), we do the following.

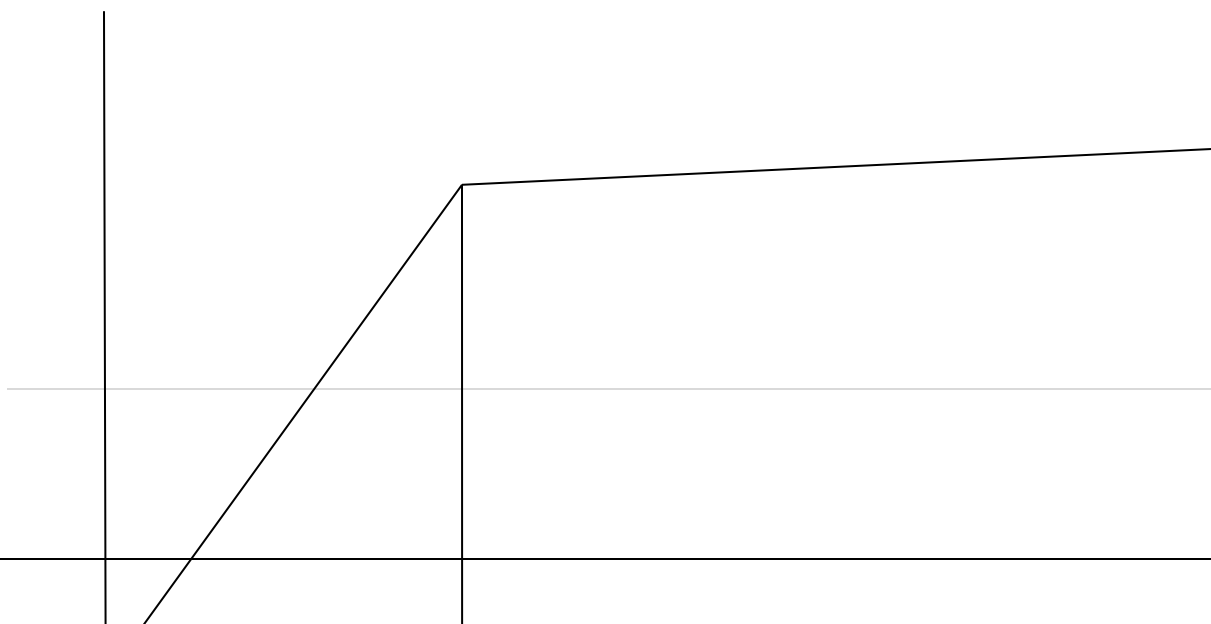
Whenever a new build comes in along with new features and defect fixes. The testing team will have a standing meeting – they discuss if their features are affected by the above changes and thus they themselves do impact analysis and come up with the impact list where maximum possible impact areas are covered and chances of bugs creeping up is less.

Whenever the new build comes, the testing team does the following,

- Smoke testing (check basic functionality)
- Test new features
- Test the modified features
- Retesting the bugs
- Regional regression testing (checking the impact areas)

The below graph shows that *increase in testing effort will not lead to catching more bugs*,

Thus, we can see that the initial effort spent on regional regression testing will lead to catching more number of bugs. But with the effort spent on full regression testing will diminish the number of bugs we catch. Thus, we can conclude increase in testing effort will not lead to catching more bugs.



### Full Regression Testing

After 2 releases of the product, during the 3<sup>rd</sup> release - customer asks for adding 2 new features, deleting 1 feature and modifying 1 feature. Also some bugs needed to be fixed. The testing team after doing impact analysis find out that making all the above changes will lead to testing the entire product.

Thus, *Testing the changes and all the remaining features is called Full Regression Testing.*

*When do we do Full Regression Testing ?*

- When changes are more
- Whenever the changes are done in the root of the product. **For ex**, JVM is the root of Java application. Whenever any changes are made in JVM, the entire Java application is tested.

Regional Regression Testing is the most preferred method of regression testing. But the problem is, we may miss a lot of bugs doing Regional Regression Testing.

We can solve this problem by the following method - when a product is given for testing, for the 1<sup>st</sup> ten cycles, we do regional regression testing, then for the 11<sup>th</sup> cycle, we do FRT. Again, for the next 10 cycles, we do RRT and for the 21<sup>st</sup> cycle we do FRT. Thus we continue like this, for the last ten cycles of the release - we do **only** FRT. Thus, following the above method - we can catch a lot of bugs.

### Interview Questions

**1) What is Regression Testing ?**

**Ans)** 1<sup>st</sup> tell - definition of Regression Testing

*Then continue with,*

*Based on the changes, we test only the changes OR the changes and impact areas OR the changes and the entire product.*

*Thus, we have different types of Regression Testing, namely,*

- *Unit Regression Testing - test only the changes*
- *Regional Regression Testing - test only the changes and impact areas*
- *Full Regression Testing - test all the changes and the entire product*

**2) Difference between Re-testing and Regression Testing.**

**Ans)** Re-Testing - developer fixes the bug(or makes some changes) and gives the product for testing. We are testing only the fixed bug(or changed areas) i.e, we are testing only the defect fixes. We are re-validating the defect



**Disadvantages** of doing regression testing manually again and again,

- Monotonous job
- Efficiency drops down
- Test execution time is more
- No consistency in test execution

Thus, we go for Automation to solve this problem. When we have more cycles of Regression testing – we go for Automation.

**Automation :**

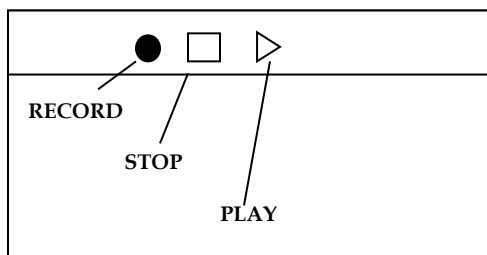
Customer gives requirements – we start writing test cases – about 1000 test cases are written for the entire product – development team gives the 1<sup>st</sup> build – we convert about 600 test cases to QTP scripts and the remaining 400 test cases are not converted – we executed the converted QTP test cases using QTP tool. Remaining 400, we test manually – thus 60% of time is saved by automating our testing.

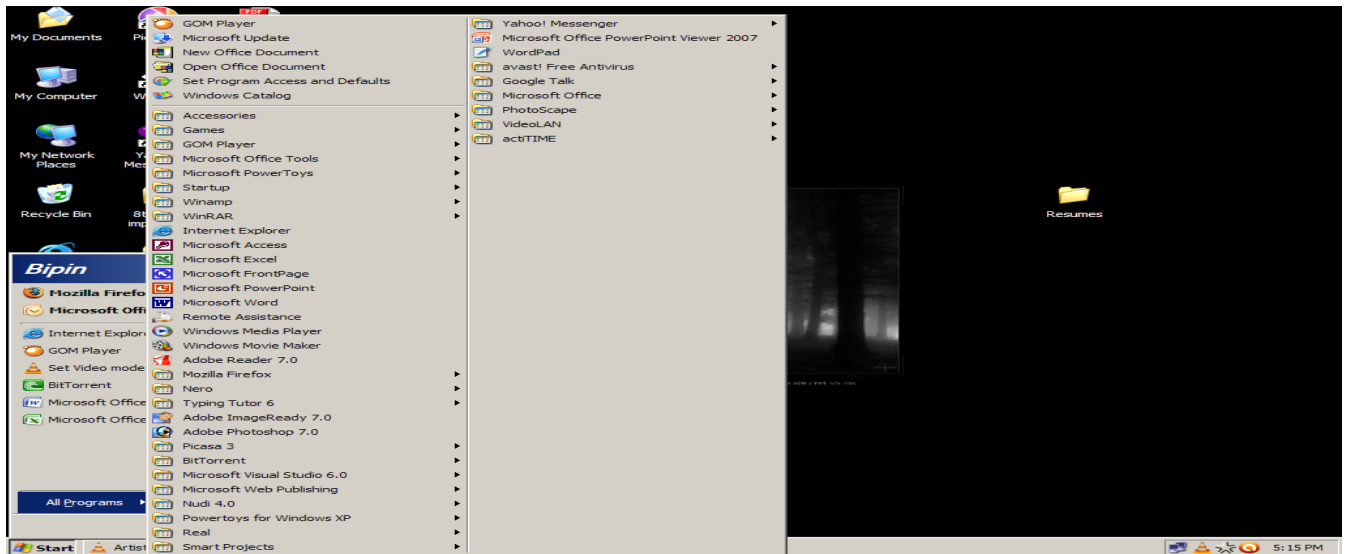
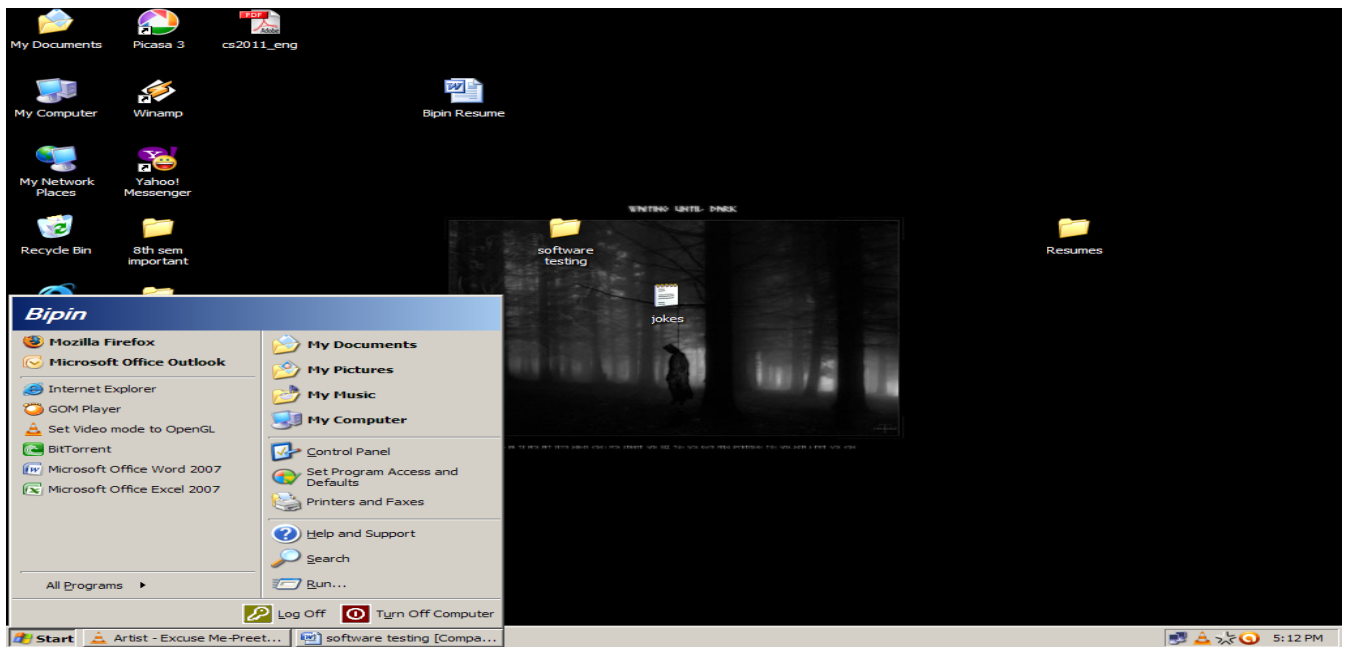
**QTP** stands for **Quick Test Professional**

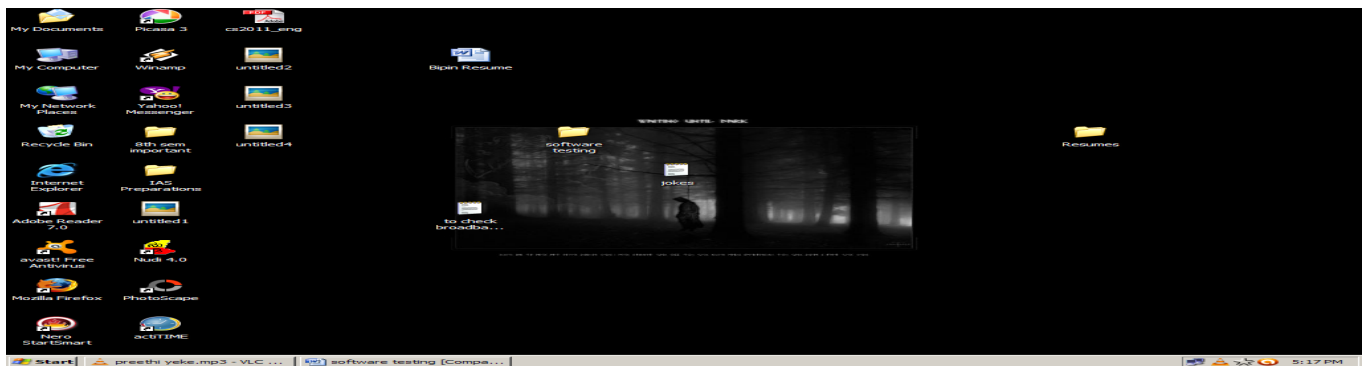
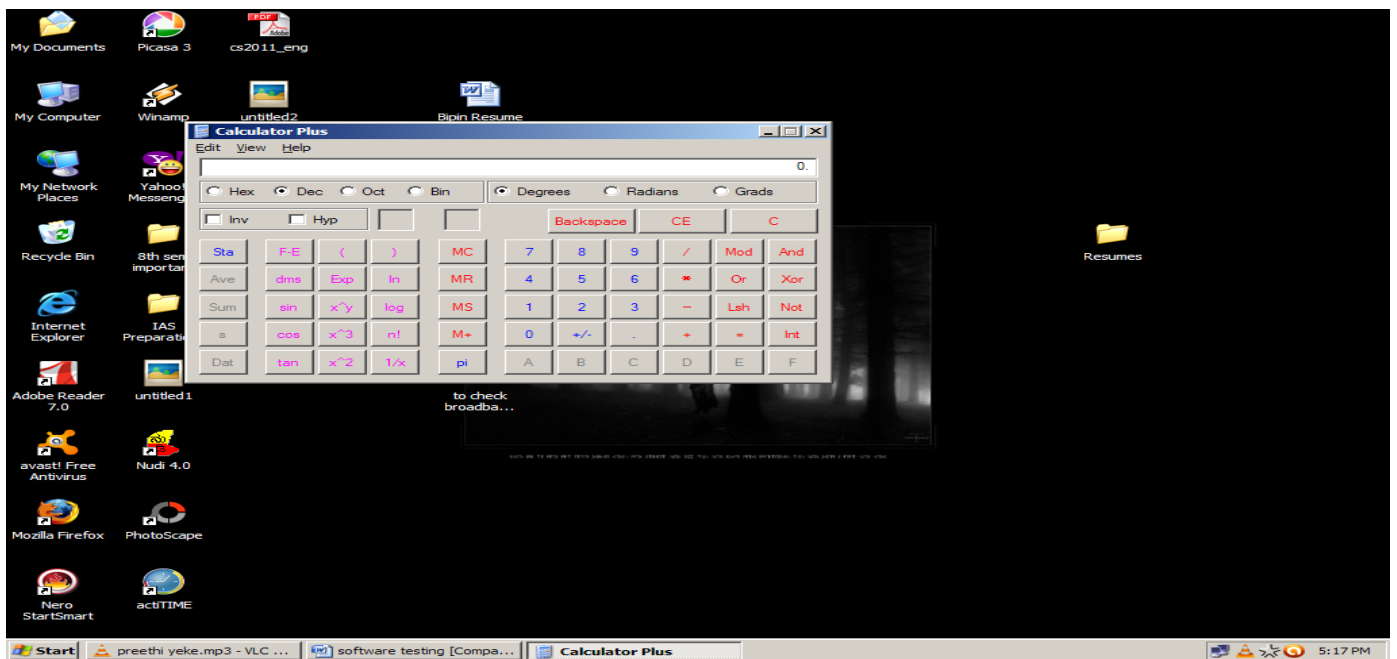
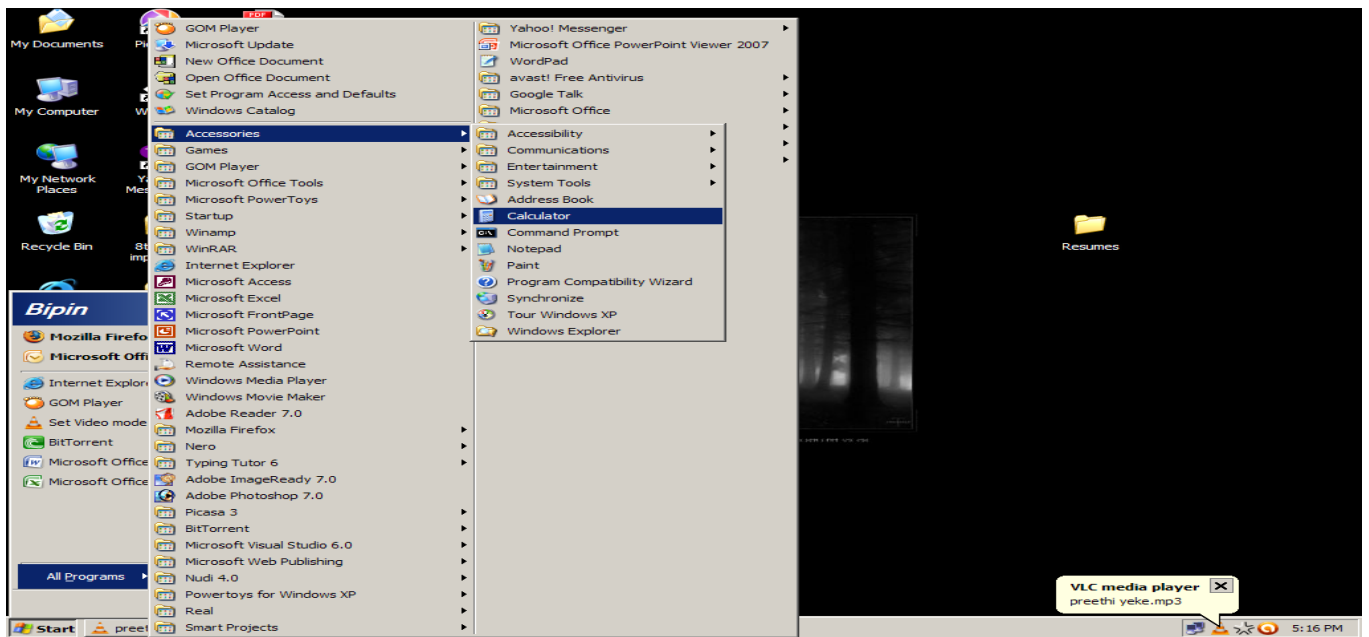
Now we see, **how to convert the manual test cases to QTP scripts (Automation scripts).**

If we open QTP, we see 3 buttons – record, play and stop.

This records every click and action on the desktop. It records the action and plays it back.



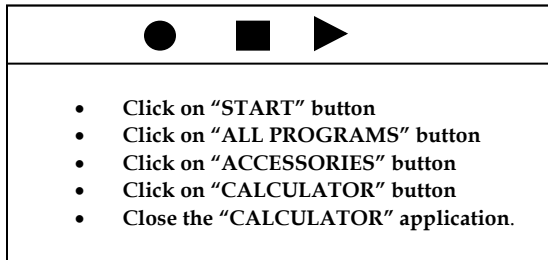




We open QTP on the desktop and perform the following functions,

- Click on "START" button
- Click on "ALL PROGRAMS" button
- Click on "ACCESSORIES" button
- Click on "CALCULATOR" button
- Close the "CALCULATOR" application.

All the above actions are recorded in the QTP tool as shown below,



When we click on "play" button, the QTP button automatically performs all the actions it has recorded. This is the basic functionality of QTP.

Now, the TE has written about 1000 test cases – of which he wishes to convert 600 to QTP (or Automation) scripts.

Let us consider a sample test case as shown below,

#### ***Header of the Test Case***

<i>Click on browser. Enter URL - xyz.com</i>
<i>Enter valid Username(abc123) and password(ijk123). Click on Login button</i>
<i>Click on Sales link.</i>
....
....
....
....
<b><i>Footer of the Test Case</i></b>

Now, whatever action is performed as above, QTP records the entire actions and thus the manual test case is converted to QTP scripts.

To go for QTP (automation), we need 3 things,

- Application
- Test Case
- QTP

Purpose of QTP is only to do regression testing – to check if the old features are broken and not to catch new bugs.

If the developer gives new build, we should first manually test the new features and also test the existing features using QTP.

Thus any new features we test manually. Whatever is not automated, we test manually and the remaining test cases are automatically tested.

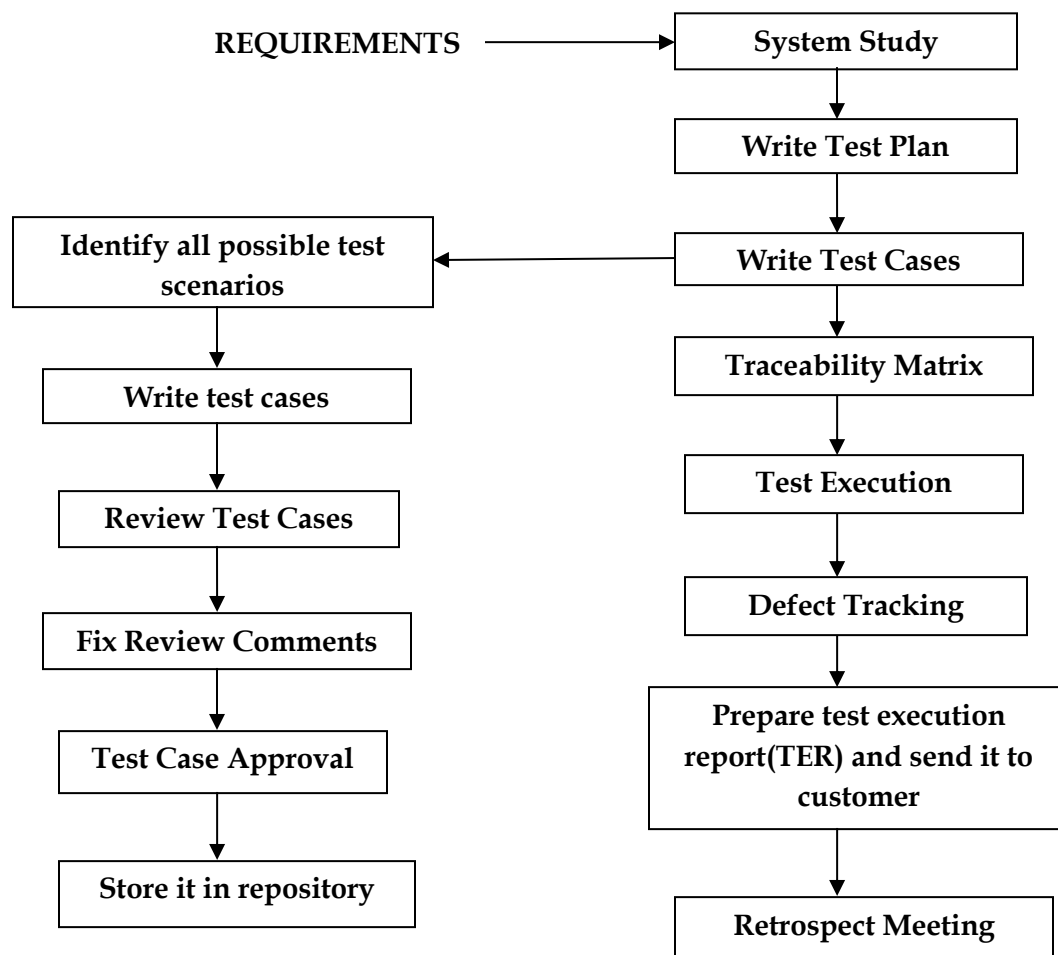
In a new release, when the customer gives new features – we first write test cases and manually test the new features. For the 1<sup>st</sup> release of the product, we will not automate the test case. For the 2<sup>nd</sup> release onwards –we should automate the test cases. For the 3<sup>rd</sup> release – the TE will automate the test cases for the 2<sup>nd</sup> and 1<sup>st</sup> release and manually test the new features of the 3<sup>rd</sup> release.

### **SOFTWARE TEST LIFE CYCLE (STLC)**

Testing itself has many phases i.e is called as STLC.

STLC is part of SDLC

Defect Life Cycle is a part of STLC



**Requirement** is the input for testing.

**Test Plan** – is a document which derives all future activities of the project. All future testing activities is planned and put into a document and this document is known as Test Plan. It contains – number of engineers needed for the project, who should test which feature, how the defects must be communicated to the development team, when we should start and finish writing test cases, executing test cases, what are the types of testing we use to test for the application etc.

**Write test case** – we write test cases for each feature. These test cases are reviewed, and after all mistakes are corrected and once the test cases are approved – then they are stored in the test case repository.

**Traceability Matrix** – it is a document which ensures that every requirement has a test case .

Test cases are written by looking at the requirements and test cases are executed by looking at the test cases. If any requirement is missed i.e, test cases are not written for a particular requirement, then that particular feature is not tested which may have some bugs. Just to ensure that all the requirements are converted, traceability matrix is written. This is shown below,

**TRACEABILITY MATRIX**

Requirement Number	Test Case Name
1	...
2	...
3	
4	...
5	...
6	
7	...

For the requirements (3 and 6) for which test cases are not written, the cells are marked in thick border so that they are distinct and then test cases are written for them.

The Traceability Matrix is also known as RTM(Requirement Traceability Matrix) or CRM(Cross Reference Matrix).

**Defect Tracking** – any bug found by the testing team is sent to the development team. This bug has to be checked by the testing team if it has been fixed by the developers.

**Test Execution Report** :- Send it to customer – contains a list of bugs(major, minor and critical), summary of test pass, fail etc and when this is sent, according to the customer – the project is over.

TER is prepared after every test cycle and sent to development team, testing team, management and customer(depends if it is a *fixed bid project* or *time & material bid project*).

The last TER of the last test cycle is always sent to the customer. And this means that the project is over-according to the customer.

**Retrospect meeting** – (also called Post Mortem Meeting / Project Closure Meeting)

The Test Manager calls everyone in the testing team for a meeting and asks them for a list of **mistakes** and **achievements** in the project.

<b>MISTAKES</b> (Mistakes in the Process)	<b>ACHIEVEMENTS</b> (good process/procedure followed)
1) Review process is not good	1)Last day of each cycle generally swap modules and do ad-hoc testing
....	.....
.....	.....
.....	....

This is done by test lead or test manager. Here, the manager documents this retrospect meeting and stores it in QMS (Quality Management System). It is a folder, where inside this folder, there is another folder called Retrospect folder and here this excel sheet document is stored. When we get new project, while we write

the test plan – we will open this retrospect file and will try and implement the good practices and correct the mistakes.

### Interview Questions

*Q) What is test life cycle (OR) What is STLC (OR) I'll give you a product. What is the process you'll follow to test it (OR) In your current project, what is the process you are following to test the product.*

*Ans) everything has only one answer – STLC*

*Start from system study upto retrospect meeting. Briefly explain every stage and link every stage by saying “and then we move on”.*

*After retrospect meeting, we store it in QMS(Quality Management System). When a new requirements or project comes – during test plan stage – the test lead takes retrospect meeting excel sheet from the Test case repository and implements the good process followed and avoids the mistakes of the previous release. This procedure continues in the next release. Again the 2<sup>nd</sup> release retrospect meeting excel sheet is seen during test plan of a new project or during the 3<sup>rd</sup> release.*

*THUS WE FINE TUNE THE TEST LIFE CYCLE. So mistakes are reduced and also good procedures are followed, thus increasing the quality of the product.*

Now we will look in detail each step of the STLC.

## REQUIREMENTS COLLECTION / SYSTEM STUDY

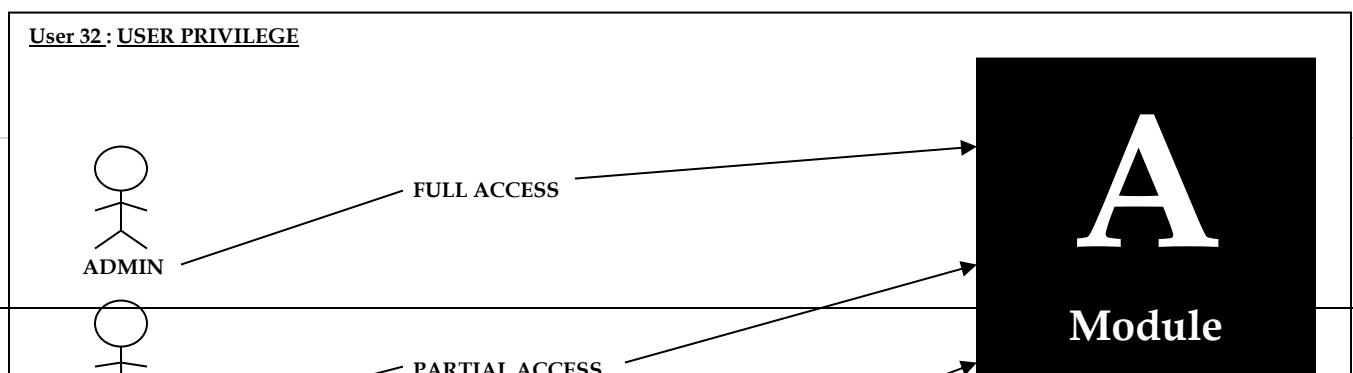
The requirements can be in any of the following forms,

- CRS (Customer Requirement Specification)
- SRS (System Requirement Specification)
- FS (Functional Specification)
- If we don't have requirements and if we are given only the application, then we do *exploratory testing*.
- Use case

### Use Case

**Use case** is a pictorial representation of requirements. It explains how the end user interacts with the application. It gives all possible ways of how the end user uses the application.

Below is shown an example of how a **use case** looks like,



The above figure shows a sample **use case** of one of the requirements in the CRS.  
For the module A of the application, there are 7 features.  
Admin has access to all the 7 features.  
For a paid user – access to 4 features  
For a free user – no access to any of the features.

**Ex – for admin**  
*Precondition* – admin must be created  
*Action* – login as paid user  
*Post condition* – 4 features must be there



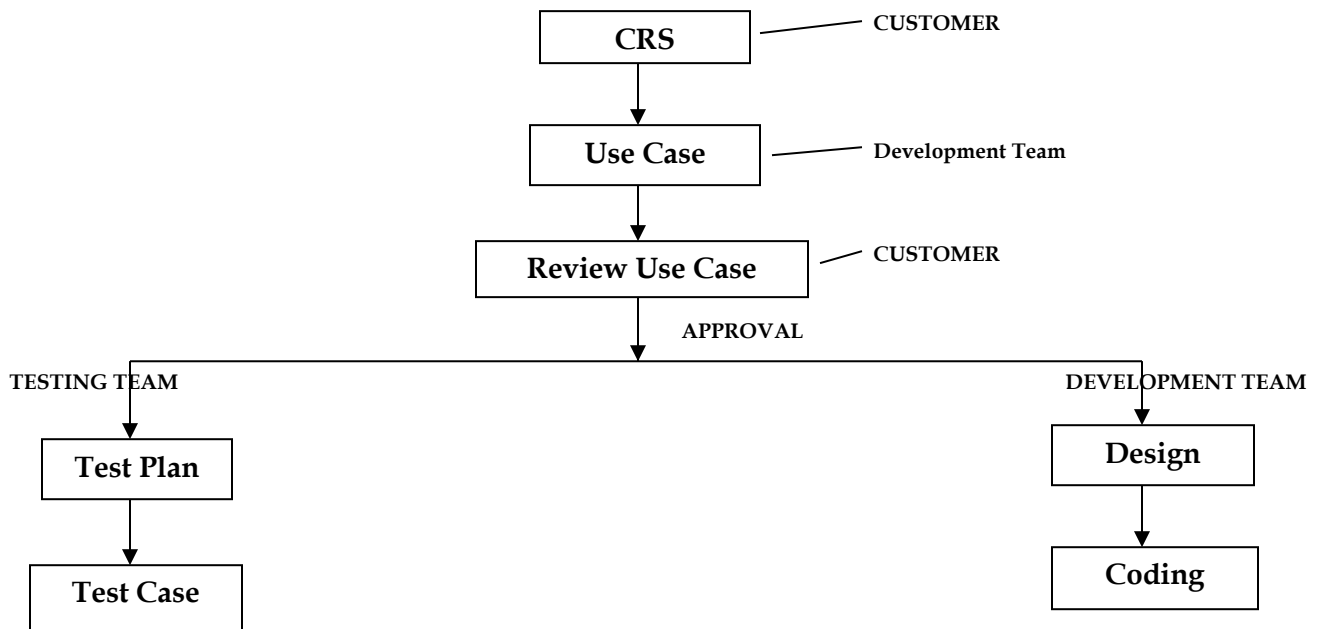
**Ex** – for free user

*Precondition* – free user must be created

*Action* – login as free user

*Post condition* – no features

### Who writes use cases



Customer gives the CRS for the application to be developed. The development team write the **use case** for the CRS and the **use case** is sent to the customer for review. If the customer approves it, then the approved **use case** is sent to the development team for design and coding. The approved **use case** is also sent to the testing team who start writing test plan and later on start writing test cases for the features of the application.

### Difference between use case and prototype

**Use case** – talks about how the product should work. It is a pictorial representation of the application and its various features and also how they should work.

**Prototype** – here, we will not see how the end user interacts with the application. It's just a screenshot of the application (exact image of the application)

### How developers develop use cases

Developers use standard symbols to write **use cases** for universal understanding. He uses UML – Unified Modelling Language to develop **use cases**.

There are readymade tools to write use cases – like **Rational Rose**. It has readymade UML symbols – we can just drag and drop them to write **use cases** – developers use these symbols and write **use cases**.

When you join a company, always ask for –

- Requirements of the project
- Test plan of the project
- Test cases(existing) of the project
- Application which is to be developed.

## TEST PLAN

Test plan is a document which drives all future testing activities.

Test plan is prepared by **Test manager(20%)**, **Test Engineer(20%)** and by **Test Lead(60%)**.

There are **15 sections** in a test plan. We will look at each one of them below,

**1) OBJECTIVE** :- It gives the aim of preparing test plan i.e, why are we preparing this test plan.

### **2) SCOPE :-**

#### **2.1 Features to be tested**

For ex, Compose mail

Inbox

Sent Items

Drafts

#### **2.2 Features not to be tested**

For ex, Help

...

...

...

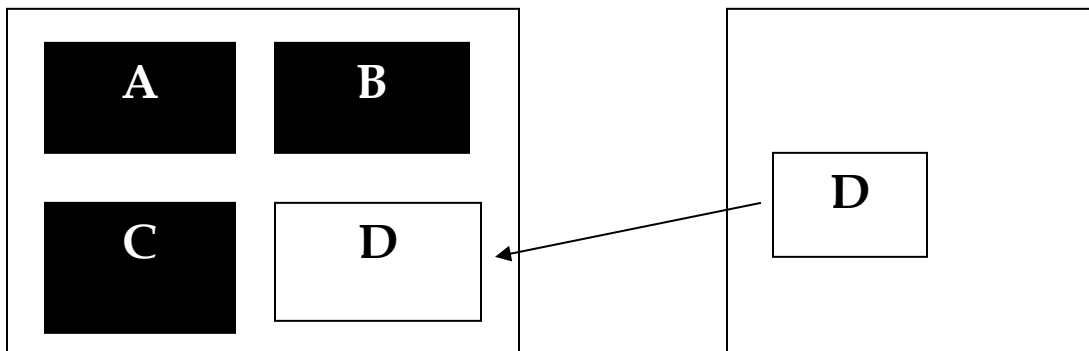
...

i.e, In the planning stage, we decide which feature to test and which not to test due to the limited time available for the project.

**How do we decide this (which features not to be tested) ?**

**a)** "HELP" is a feature developed and written by a technical writer and reviewed by another technical writer. So, we'll not test this feature.

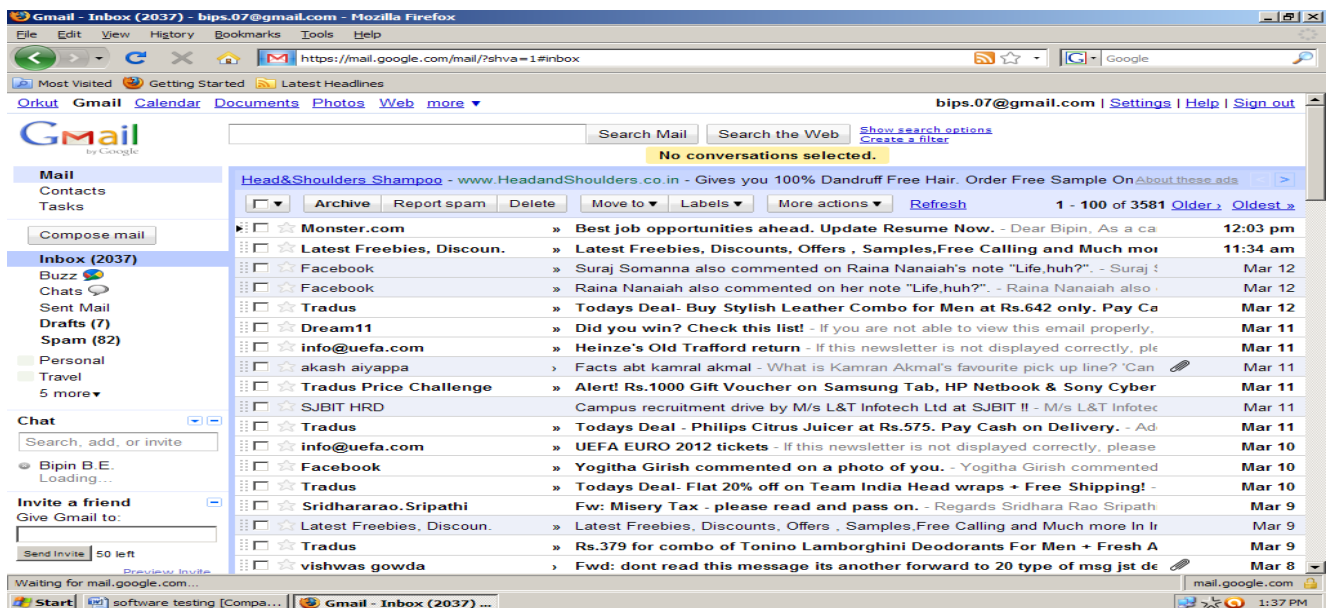
**b)**



Let us consider that an application with features A, B, C and D are to be developed as per requirements. But then, D has already been developed and is in use by another company. So, the development team will purchase D from that company and integrate with the other features A, B and C.

Now, we will not do functional testing on D because D is already in use in the market. But we will do integration testing and system testing between A, B, C and D because the new features may not work with D properly.

**c)**



The application might be having link to some other application. Here, our scope of testing is limited to,

- Whether link exists
- If it goes to homepage of the corresponding application when we click on the link.

Let us consider the **example** of Gmail. When we log into gmail, we see many links to other applications like - orkut, picassa, youtube etc. when we logged into Gmail and when we click on the orkut link - it must take us to Orkut's homepage.

Such features are called *Single sign-on feature* - it is a feature wherein 1 login allows access to multiple applications.

d) In the 1<sup>st</sup> release of the product - features that have been developed are - a, b, c, d, e, f, g, h, ... m, n, o. Now, the customer gives requirements of new features to be built for enhancement of the product during the 2<sup>nd</sup> release. The features to be developed are - p, q, r, s, t.

During test plan, we write scope,

### **Scope**

#### **Features to be tested**

P, Q, R, S, T (new features)

A, B, C, D, E, F

#### **Features not to be tested**

G, H, I, J, ... N, O

Thus we first test new features and then test old features which might be affected by building the new features i.e, impact areas. We do regression testing for A, B, C, ... F.

### **3) TESTING METHODOLOGIES (Types of Testing)**

Depending upon the application, we decide what type of testing we do for the various features of the application. We should also define and describe each type of testing we mention in the testing methodologies so that everybody (dev team, management, testing team) can understand, because testing terminologies are not universal.

**For example**, we have to test [www.shaadi.com](http://www.shaadi.com), we do the following types of testing,

Smoke testing	Functional testing	Integration testing
System testing	Adhoc testing	Compatibility testing
Regression testing	Globalization testing	Accessibility testing
Usability testing	Performance testing	

For standalone applications, like AutoCad, we do the following types of testing,

Smoke testing	Functional testing	Integration testing
System testing	Adhoc testing	Compatibility testing
Regression testing	Globalization testing	Accessibility testing
Usability testing	Reliability testing	Recovery testing
Installation / Uninstallation testing		

### **4) APPROACH**

The way we go about testing the product in future,

- a) By writing high level scenarios
- b) By writing flow graphs

#### ***a) By writing high level scenarios***

for ex, we are testing [www.yahoo.com](http://www.yahoo.com)

i) Login to Yahoo – send a mail and check whether it is in Sent Items page

ii) Login to .....

iii) .....

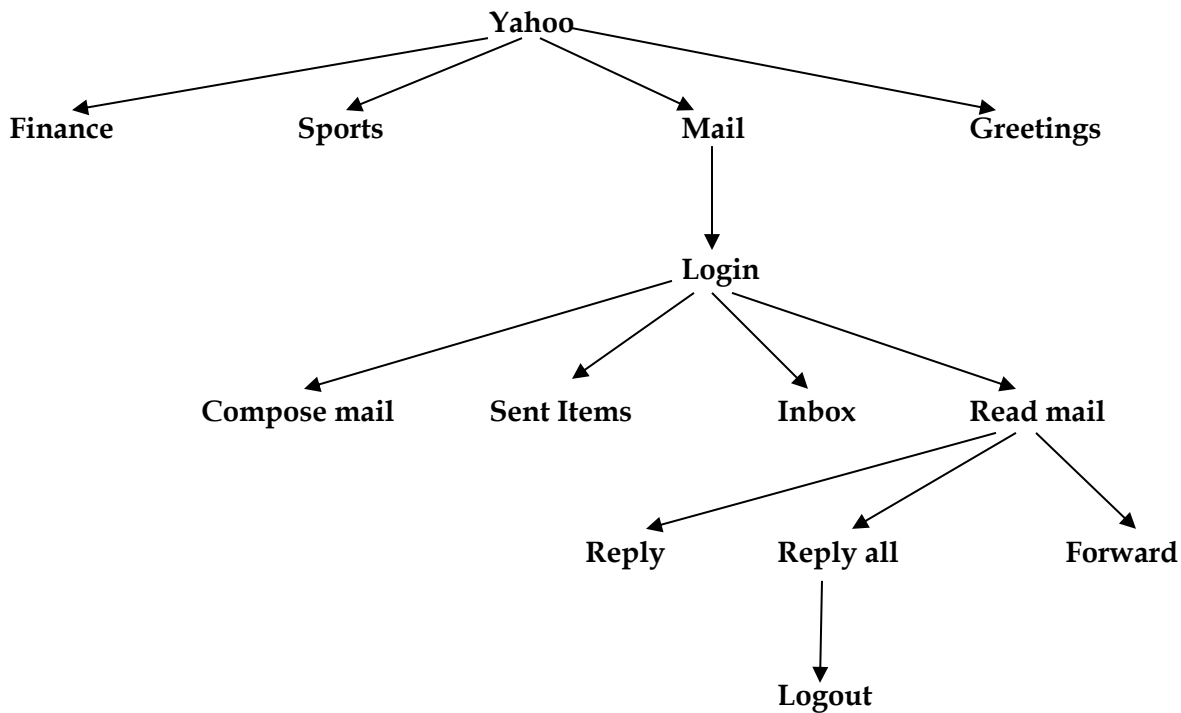
.....

.....

.....

This is written only to explain the approach to be taken to test the product. Only for the critical features, we will write a few very high level scenarios. We don't cover all scenarios here. That is the job of the respective Test Engineers for whom the features have been allocated.

***b) By writing flow graphs***



We write flow graphs because of the following advantages,

- i. Merging is easy
- ii. Coverage is easy

Flow graphs are written because writing high level scenarios is time consuming.

**5) ASSUMPTIONS**

When writing test plans, certain assumptions would be made like technology, resources etc.

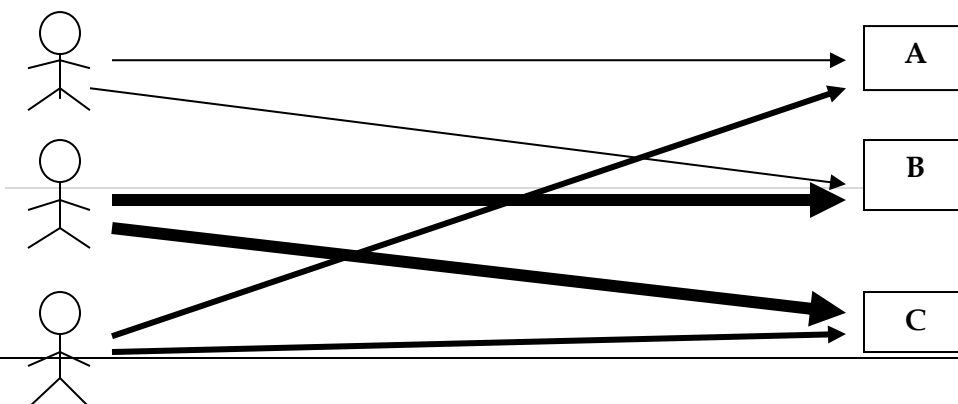
**6) RISKS**

If the assumptions fail, risks are involved

**7) CONTINGENCY PLAN OR MITIGATION PLAN OR BACK-UP PLAN**

To overcome the risks, a contingency plan has to be made. Atleast to reduce the percentage from 100% to 20%

Let us consider an **example for 5, 6, 7**



In the project, the **assumption** we have made is that all the 3 test engineers will be there till the completion of the project and each are assigned modules A, B, C respectively. The **risk** is one of the engineers may leave the project mid-way.

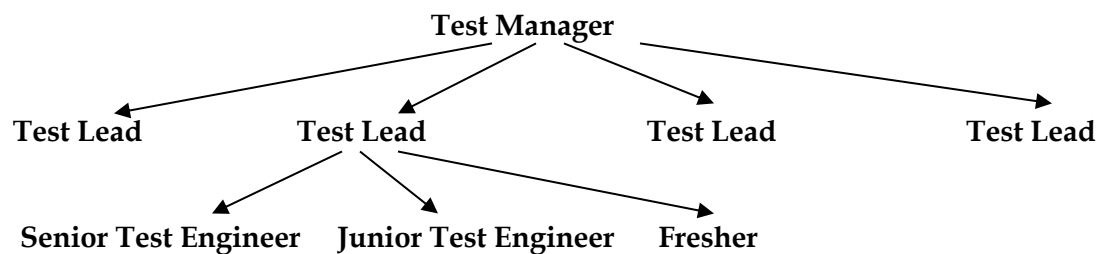
Thus, the **mitigation plan** would be to allocate a primary and secondary owner to each feature. Thus, one engineer quits - the secondary owner takes over that particular feature and helps the new engineer to understand their respective modules.

Always **assumptions, risks, mitigation plan** are specific to the project.

The different types of risks involved are,

- Resource point of view
- Technical point of view
- Customer point of view

## **8) ROLES AND RESPONSIBILITIES**



When a Big project comes, it's the Test Manager who writes the test plan.

If there are 3 small projects, then Test Manager allocates each project to each Test lead. The Test lead writes the test plan for the project which he is allocated.

### **8.1 Test Manager**

- Writes or reviews test plan
- Interacts with customer, development team and management
- Sign off release note
- Handle issues and escalations
- ....
- ....
- ....

### **8.2 Test Lead**

- Writes or reviews test plan
- Interacts with development team and customers

- Allocates work to test engineers and ensure that they are completing the work within the schedule
- Consolidate reports sent by Test Engineers and communicate it to development team, customers(if it is a time&material project) and management
- ...
- ...
- ...

### 8.3 Test Engineer 1

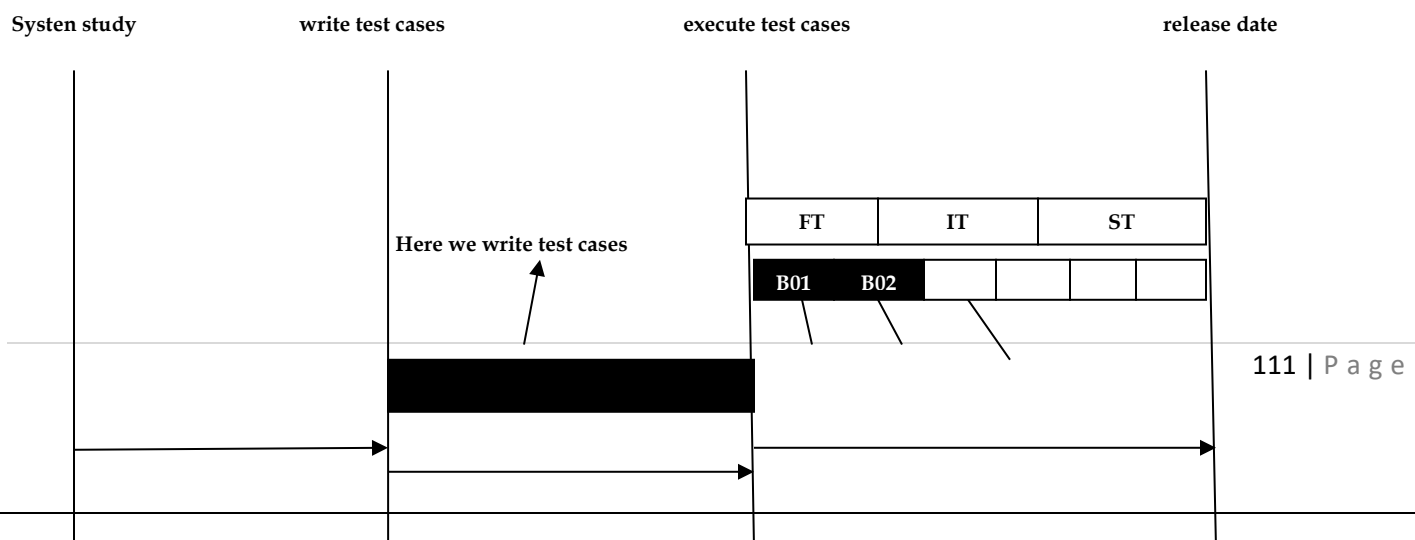
- Review test plan
- Write test cases for trend analysis
- Asset survey
- Write traceability matrix
- Review test cases written for sales and purchase modules
- Execute test cases written for trend analysis, asset survey, registration (old module developed in previous release. Adding trend analysis and asset survey has affected. Old module has been affected. So do regression testing)
- Perform compatibility testing using Internet Explorer, Mozilla Firefox and Google Chrome in Windows XP and Windows Vista
- Prepare test execution report and communicate it to Test lead.
- ....
- ....
- ...

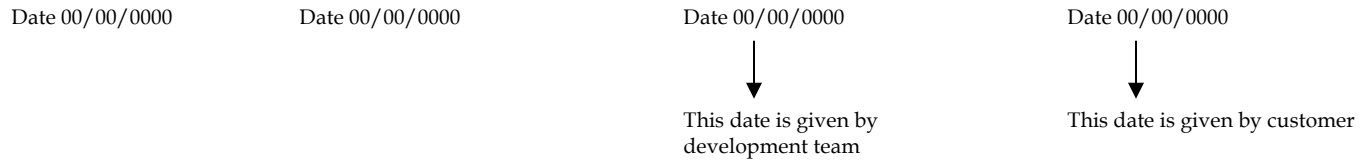
### 8.4 Test Engineer 2

- Set up and install the product
- Identify test cases to be automated
- Automate identified test cases using QTP
- Execute and maintain automation scripts
- ...
- ...

## 9) SCHEDULES :-

This section contains – when exactly each activity should start and end? Exact date should be mentioned and for every activity, date will be specified.





Thus, as we can see from the above figure – for every specified activity, there will be a starting date and closing date. For every build, there will be a specified date. For every type of testing for each build, there will be a specified date.

## **10) DEFECT TRACKING**

In this section, we mention – how to communicate the defects found during testing to the development team and also how development team should respond to it. We should also mention the priority of the defect – high, medium, low.

### **10.1 Procedure to track the defect**

....  
....  
....  
....

### **10.2 Defect tracking tool**

We mention the name of the tool we will be using to track the defects

### **10.3 Severity**

#### *10.3.1 Blocker(or Showstopper)*

....  
.... (define it with an example in the test plan)  
For ex, there will be bug in the module. We cannot go and test the other modules because this blocker has blocked the other modules.

#### *10.3.2 Critical*

...  
... (define it with an example)  
Bugs which affects the business is considered critical

#### *10.3.3 Major*

...  
... (define it with an example)  
Bugs which affects look and feel of the application is considered as major

#### *10.3.4 Minor*

...  
... (define it with an example)

### **10.4 Priority**

#### *10.4.1 High – P1*

...

#### *10.4.2 Medium – P2*

...

#### *10.4.3 Low – P3*



...  
...  
... P4

So, depending on the priority of the defect(high, medium or low), we classify it as P1, P2, P3, P4.

## **11) Test Environment**

### **11.1 Hardware**

11.1.1 Server :- Sun Starcat 1500

(this is the name of the server from which testing team take the application for testing)

11.1.2 Client :-

3 machines with following configurations,

Processor : Intel 2GHz

RAM : 2GB

...

...

...

(this gives the configurations of the computers of the Test Engineers i.e, the testing team)

### **11.2 Software**

11.2.1 Server

OS : Linux

Web Server : TomCat

Application Server : Websphere

Database Server : Oracle (or) MS – SQL Server

(the above servers are the servers which the testing team will be using to test the product)

11.2.2 Client

OS : Windows XP, Vista, 7

Browsers : Internet Explorer, Internet Explorer 7, Internet Explorer 8, Mozilla FireFox,

Google Chrome

(the above gives the various platforms and browsers in which the testing team will test the product)

### **11.3 Procedure to install the software**

...

...

...

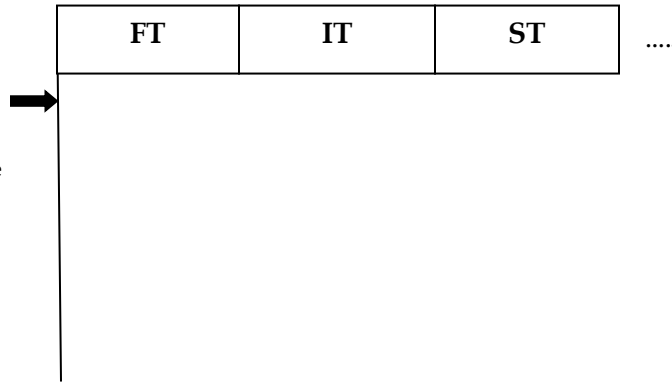
(Development team gives how to install the software. If they have not yet given the procedure, then in the test plan, we just write it as TBD – to be decided)

## **12) Entry and Exit Criteria**

EXIT →  
1)Based on %age test execution  
2)Based on %age test pass  
3) Based on severity

#### ENTRY

- a) WBT should be over
- b) Test cases should be ready
- c) Product should be installed with proper test environment
- d) Test data should be ready
- e) Resources should be available



Before we start with Functional Testing, all the above entry criteria should be met.

After we are done with FT, before we start with Integration Testing, then the exit criteria of FT should be met. The percentage of exit criteria is decided by meeting with both development and test manager. They compromise and conclude the percentage. If the exit criteria of FT is not met, then we cannot move onto IT.

Based on severity of defects means,

The testing team would have decided that in order to move onto the next stage, the following criteria should be met,

- There should not be more than 20critical bugs
- There should not be more than 60major bugs
- There should not be more than 100minor bugs.

If all the above are met, then they move onto the next testing stage.

But the problem with the above method was,

21 critical, 50major, 99minor – cant exit because there are more than 20 critical bugs.

10critical, 90major, 200 minor – can exit. But the 10 critical bugs can affect the product.

Thus, they came up with the concept of “*weight of defects*”. i.e, 3major = 1 critical, 5minor – 1critical and total critical should not be more than 60.

So, for,

21 critical – 21

50major – 16critical

99minor – 19critical

Totally there are 56critical bugs, so we can move onto the next stage.

But for the 2<sup>nd</sup> example, we cannot move on.

#### Entry criteria for IT :

- should have met exit criteria of FT

...

...

...

(remaining all are same as entry criteria of FT)

#### Exit criteria for IT :

...

...

...

All points are same as exit criteria for FT.

But if the %age pass for FT is 85%, then the %age pass for IT should be 90% - because as we reach the later stages of testing, we expect the number of defects to be less.

**Entry criteria for ST :**

- exit criteria of IT should be met
- minimum set of features must be developed
- test environment should be similar to production environment

...

...

(remaining all are same as of IT)

**Exit criteria for ST :**

- everything remains same as of above, but the pass %age is now 99% - there should be 0 critical bugs. There could be some 30major and 50minor bugs. If all this is met, then product can be released.

*Note : All the numbers given above are just for example sake. They are not international standard numbers!!!.*

**INTERVIEW QUESTIONS**

**Q) Customer gets 100% defect free product means,**

- |                             |                         |
|-----------------------------|-------------------------|
| a) Testing team is not good | b) Developers are super |
| c) Product is old           | d) All of the above     |

**Ans) a) is correct. Testing team is not good – because – fundamentals of software testing says there is no product which has zero defects.**

**13) TEST AUTOMATION**

**13.1 Features to be automated**

...

...

...

**13.2 Features not to be automated**

...

...

...

**13.3 Which is the automation tool you are planning to use**

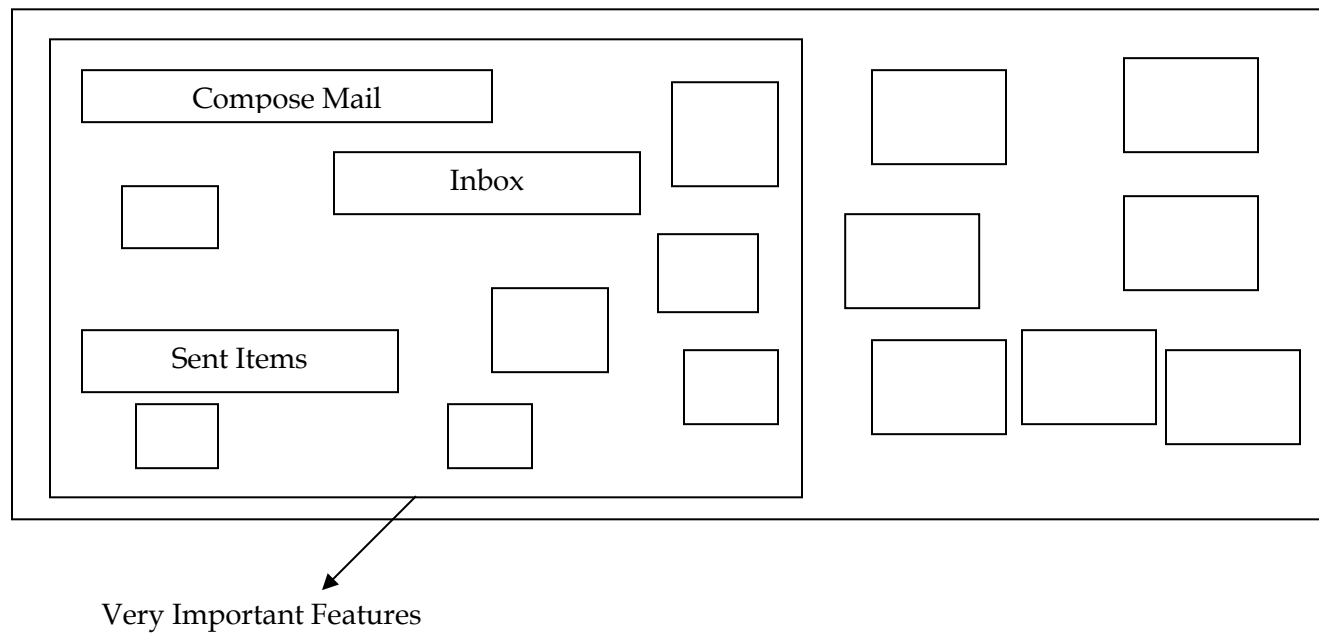
...

**13.4 What is the automation framework you are planning to use**

...

We automate the test cases only after the 1<sup>st</sup> release (*we have studied this earlier*).

### 13.1 On what basis do we decide which feature to be automated ?



If the features are very important and need to be repeatedly tested, then we automate that feature. Because manually testing the feature takes longer time and also becomes tedious job.

### 13.2 How to decide which features are not to be automated ?

- For ex, "HELP" is a feature that is not repeatedly tested – so we don't have to automate it.
- If the feature is unstable and has lot of defects – we will not automate because it has to be tested repeatedly manually.
- If there is a feature that has to be repeatedly tested, but we are predicting a requirement change for that feature – so we don't automate it as changing the manual test case is easier than changing the automation script.

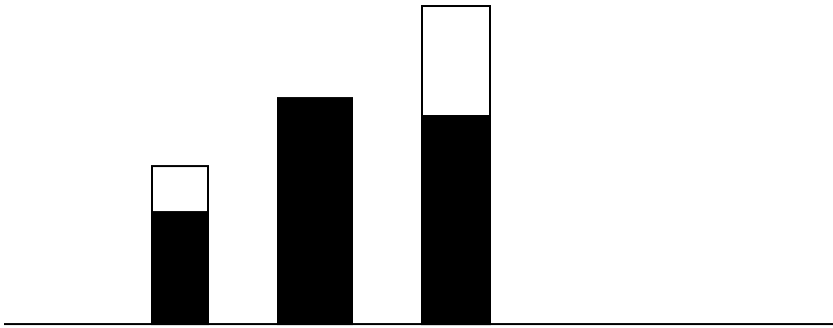
## 14) DELIVERABLES

It is the output from the testing team. It contains what we will deliver to the customer at the end of the project. It has the following sections,

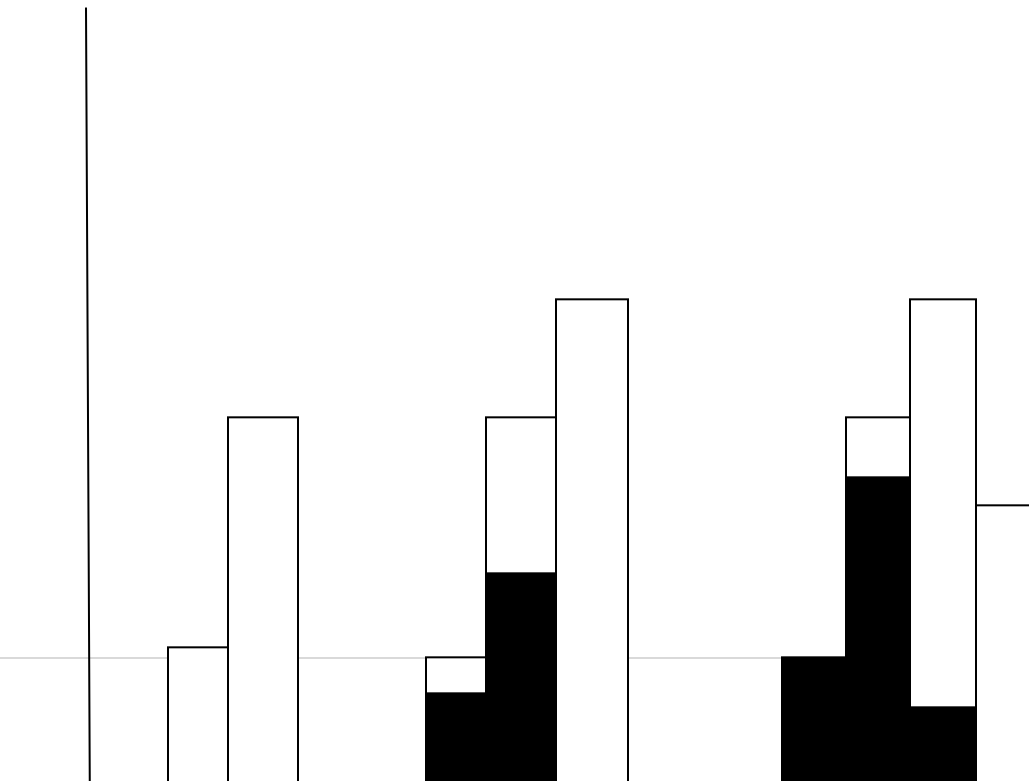
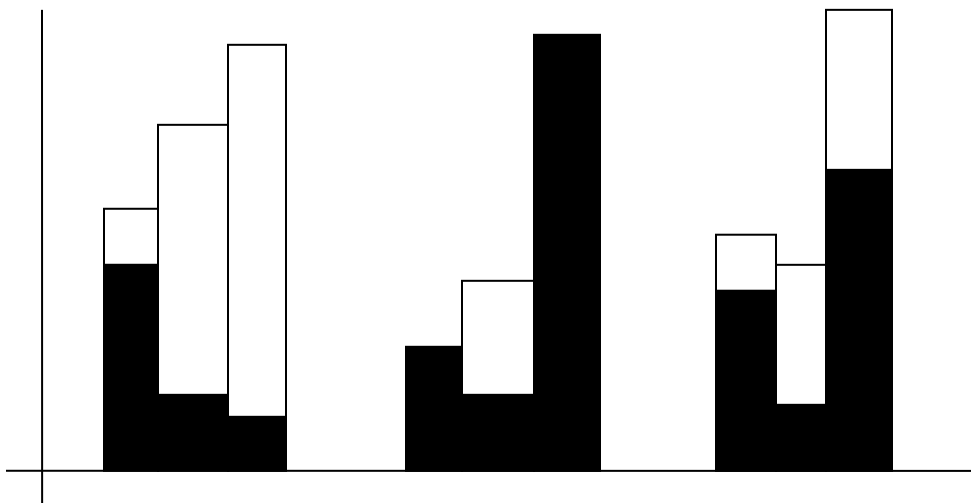
- ❖ **14.1** Test Plan
- ❖ **14.2** Test Cases
- ❖ **14.3** Test Scripts
- ❖ **14.4** Traceability Matrix
- ❖ **14.5** Defect Report
- ❖ **14.6** Test Execution Report
- ❖ **14.7** Graphs and Metrics
- ❖ **14.8** Release Note

### 14.6 Graphs and Metrics

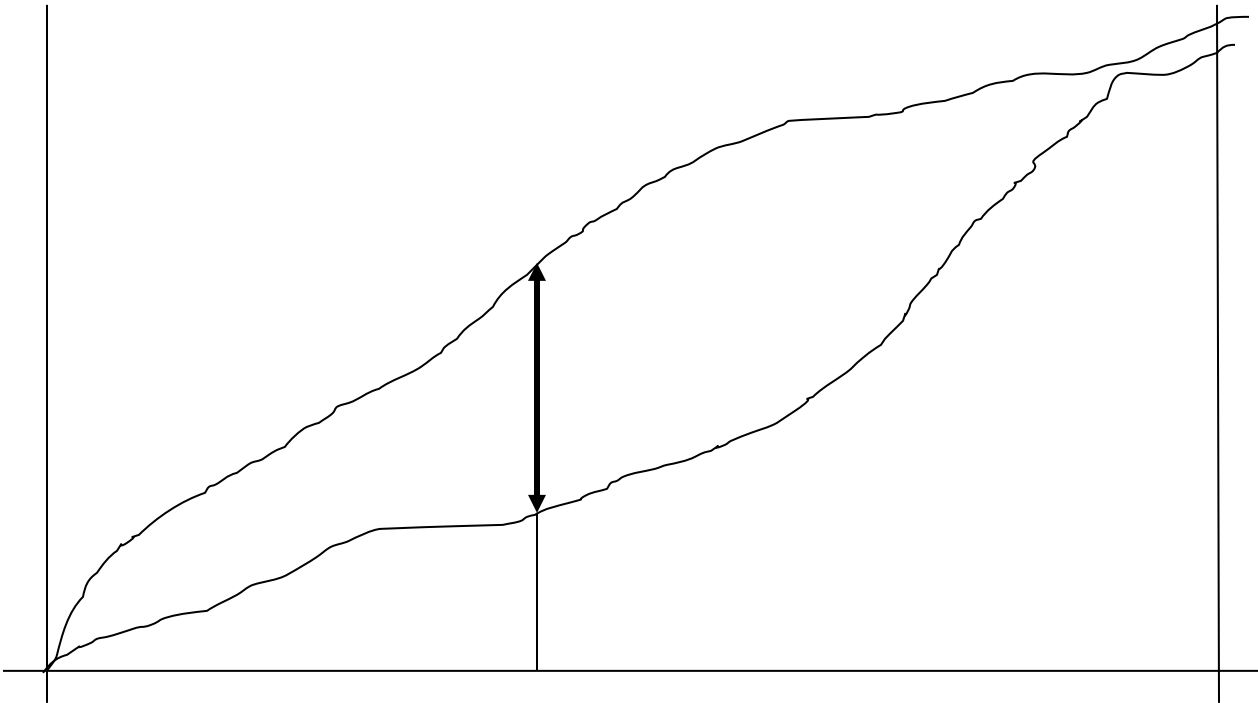
Here, we will just mention what are the types of graphs we will deliver and also give a sample of each graph we will be delivering.



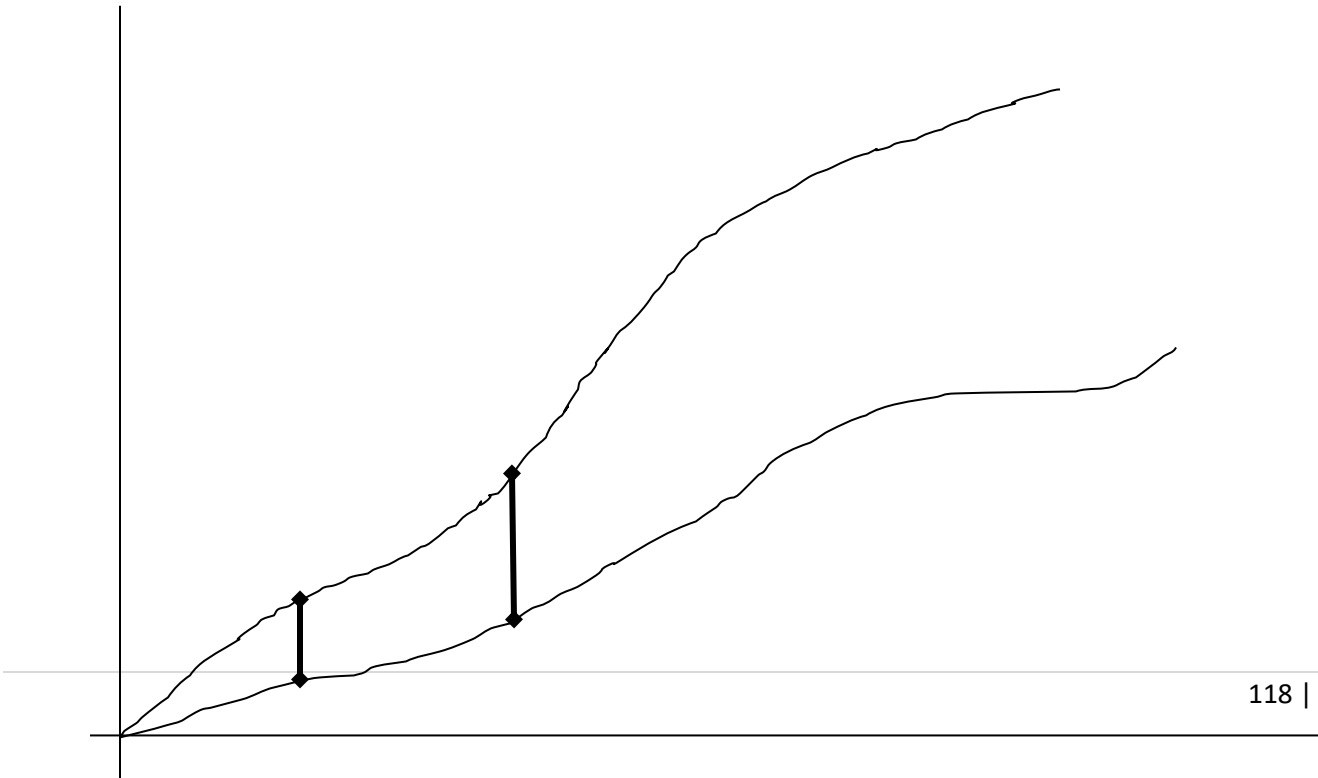
**(Defect Distribution Graph)**



(Build-wise Graph)



(Defect Trend Analysis Graph)



*Graph 1 :-* in this graph we depict – how many bugs have been found and how many bugs have been fixed in each module.

*Graph 2 :-* in this graph, we depict – how many critical, major and minor bugs have been found for each module and how many have been fixed for each module.

*Graph 3 :-* in this graph, we depict – build wise graph i.e, in each build how many bugs have been found and fixed for each module. According to the module, we have found defects. Adding C has introduced a lot of bugs in A and B. Adding D has introduced a lot of bugs in A, B and C.

*Graph 4 :-* Defect Trend Analysis graph depicts – this graph is prepared every month and we must send it to management. It's a kind of forecast. By the end of the project, “rate of fixing defects” curve must have an upward trend. Test Lead prepares this graph.

*Graph 5 :-* Test manager prepares this graph. This graph is prepared to understand the gap in estimation of defects and the actual defects that have occurred. This graph helps in better estimation of defects in the future.

## Metrics

Module Name	Critical		Major		Minor	
	Found	Fixed	Found	Fixed	Found	Fixed
Sales	40	36	80	30	90	15
Purchase	..	...	...	...	...	...
Asset Survey	...	...	...	...	...	...

## Defect Distribution Metrics

We generate the *defect distribution graph*(graph 1) by looking at the above data. Similarly we can generate many such metrics.

**For ex,**

Test Engineer Name	Critical		Major		Minor	
	Found	Fixed	Found	Fixed	Found	Fixed
Bipin	40	36	80	30	90	15
Rajath	..	...	...	...	...	...

Amith	..	...	...	...	...	...
-------	----	-----	-----	-----	-----	-----

In the above graph, we are maintain a record of all the test engineers in the project and how many bugs have been caught and fixed etc. We can use this data for future analysis. When a new requirement comes, we can decide who to give the complex feature for testing based on the number of bugs they have found. We will be in a better position to know who can handle the complex features very well and find maximum number of bugs.

### *Interview Questions and Tips*

**1) What is Metrics ?**

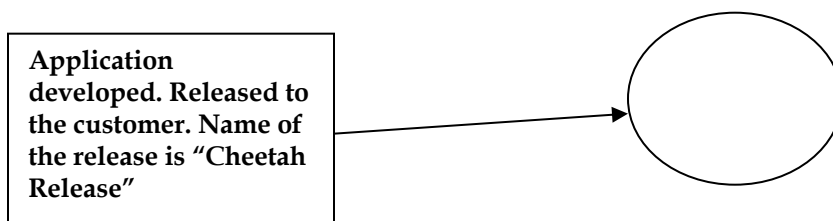
**Ans) We can tell any of the above.**

**2) On the last day of the project i.e on release date, we find a critical bug. Then what will you do? Will you release the product or fix the critical bug?**

**Ans) I<sup>st</sup> say – Testing team prepares a report and sometimes also provides a suggestion on what can be done. But, it's the Management team which takes a decision on whether to release the product or not.**

**But, now the Interviewer asks you – “I am asking what will you do, not what the management will do?” Then answer like this – “I will not release the product with critical bug because I want to deliver a high quality product”.**

### 14.7 Release Note



The product is developed and tested and released to the customer. The name of the release is “Cheetah Release”.

The release note contains,

1) List of pending/open bugs

...

...

2) List of features added, modified or deleted

...

...

3) Platforms(OS, Browsers, Hardware) in which the product is tested

..



...



Let us consider that Cheetah release is the 2<sup>nd</sup> release of the product after the 1<sup>st</sup> release Tiger release. Some of the bugs found in the 1<sup>st</sup> release has been fixed in the 2<sup>nd</sup> release. Also a list of features which have been added, modified and deleted from the 1<sup>st</sup> release to the 2<sup>nd</sup> release will be mentioned here.

..

**Release Note** is a document prepared during release of the project and signed by test manager

## 15) TEMPLATES

This section contains all the templates for the documents which will be used in the project. Only these templates will be used by all the test engineers in the project so as to provide uniformity to the entire project. The various documents which will be covered in the Template section are,

- Test Case
- Traceability Matrix
- Test Execution Report
- Defect Report
- Test Case Review Template
- ...
- ...
- ...

**This is how a Test Plan document looks like,**

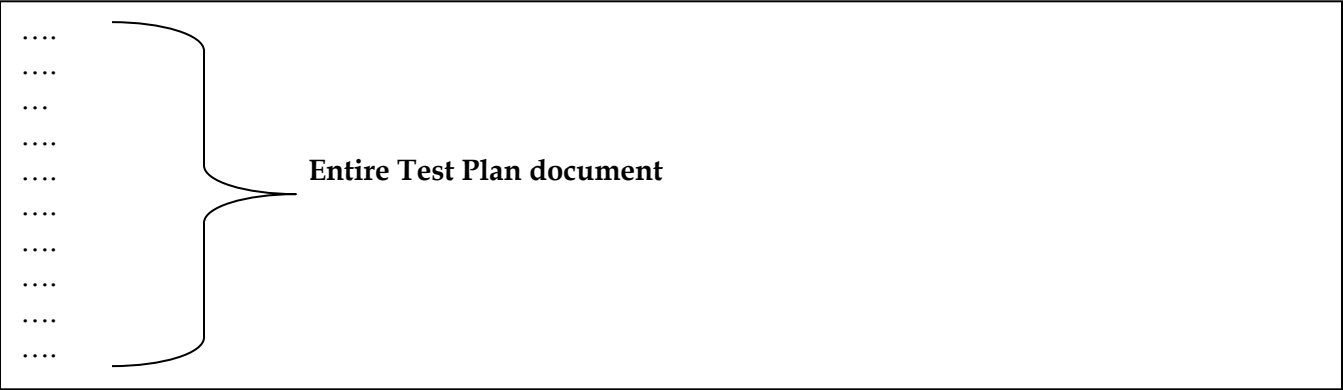
Pg 1

CBO_Testplan					
Revision History					
Version	Author	Reviewed By	Approved By	Comments	Approval Date
1	..	...	Name of manager	Version 1.0 is developed	dd/mm/yyyy

Pg 2

<u>TABLE OF CONTENTS</u>		
1	Objective	pg 1
2	Scope	pg 2
3	Approach	pg 3
..		
...		
...		
...		

Pg 3 - Pg 19



Pg 20

<u>REFERENCES</u>
1) CRS
2) SRS
3) FS
4) Design document
...
...
...

In the 1<sup>st</sup> page – we initially fill in only *version(write as Draft 1.0), author, comments and reviewed by*. Later on when the manager approves it, we fill in the *Approved by and approval date* and also remove (Draft) written in version column.

Generally Test Engineers only review it and the test plan is approved by Test Manager.

When new features come, we modify the Test Plan. Change the version and whichever features need to be changed. Again it is reviewed, updated and approved by manager. Test Plan must be updated whenever changes happen.

References (Pg 20) contains all the documents that are used to write the test plan document.

#### ***Who writes Test Plan ?***

- Test Lead – 60%
- Test Manager – 20%
- Test Engineer – 20%

Thus, we can see from above – in 60% of projects, Test plan is written by Test Lead and so on as shown above.

#### ***Who reviews Test Plan ?***

- Test Engineer
- Test Lead
- Test Manager
- Customer
- Development team

Test Engineer looks at the Test plan from his module point of view.

Test Manager looks at the Test plan from the customer point of view.

#### ***Who approves Test Plan ?***

- Test Manager
- Customer

#### ***Who writes Test Cases ?***

- Test Engineer
- Test Lead

#### ***Who reviews Test Cases ?***

- Test Lead
- Test Engineer
- Development Team
- Customer

#### ***Who approves Test Cases ?***

- Test Lead
- Test Manager
- Customer

## TRACEABILITY MATRIX

We have learnt about TM earlier. Once the missing requirements are identified – we write the test cases for the requirements which we have missed – review it and get it approved – and then store the test cases in the repository and then fill in the name of the test case for which the requirements have been missed.

*Traceability Matrix is a document which has got the mapping between requirements and test cases. We write TM to make sure that every requirement has got atleast 1 test case.*

<u>REQUIREMENTS</u>
...
...
....
....
....
....
...

Serial No.	Module Name	High Level Requirements	Detailed Requirement	Test Case Name	Automation Script Name
1	LOANS	1.1 Personal Loans	1.1.1 ...	CBO_PL_Approval	
			1.1.2 ...	CBO_PL_Eligibility check	
		1.2 Home Loans	1.2.1 ...		
			1.2.2 ...		
2	AMOUNT TRANSFER	2.1 FAN text field	2.1.1 Should accept only 10 digit integer	...	
			2.1.2 Should accept only those numbers which are created by manager	..	
		2.2 TAN text field	2.2.1 Should accept only 10 digit integer	...	
			2.2.2 Should accept only those numbers created by manager		

		2.3 Amount Text field	2.3.1 Should accept only 100 - 5000, positive integer	...	
			2.3.2 Should not accept more than balance		
...	...	...	...	...	

In the above TM – we have written the above TM looking at the requirements. For every requirement, we write the test case name(given in the header of the test case) and thus we know that we have written a test case for that particular requirement. Now, we block the cells for which we have not written test cases. Thus we have missed out on these requirements.

We block it in red. For whichever requirement we have not written test case – we read the requirements – write test cases – review it and get it approved – store it in repository – then go fill in the test case name for whichever requirements we have missed. Thus, in this way for every requirements we have atleast 1 test case.

Look at the figure in the next page,

Thus now all the blocked cells have been removed and test cases have been written for the missing requirements.

Serial No.	Module Name	High Level Requirements	Detailed Requirement	Test Case Name	Automation Script Name
1	LOANS	1.1 Personal Loans	1.1.1 ...	CBO_PL_Approval	
			1.1.2 ...	CBO_PL_Eligibility check	
		1.2 Home Loans	1.2.1 ...	...	
			1.2.2 ...	...	
2	AMOUNT TRANSFER	2.1 FAN text field	2.1.1 Should accept only 10 digit integer	...	
			2.1.2 Should accept only those numbers which are created by manager	...	
		2.2 TAN text field	2.2.1 Should accept only 10 digit integer	...	
			2.2.2 Should accept only those numbers created by manager	...	

		2.3 Amount Text field	2.3.1 Should accept only 100 - 5000, positive integer	...	
			2.3.2 Should not accept more than balance	...	
...	...	...	...	...	

But 100% coverage is not assured in Traceability matrix. **For ex**, a requirement may have about 10 scenarios, but we have covered only 3 scenarios. Thus, TM will not promise you that we have 100% coverage. Instead it ensures that every requirement has atleast 1 test case which in turn gives the confidence that we are touching all the features atleast once.

In the above table, we have a column named **automation script name**. If we are automating any test case, we fill in the automation script name (which is same as manual test case name) and which we are not converting, we write it as manual. Thus, we can know which test case is automated and manual. We write this because we cant check every test case to see if it is manual or automated. This is shown in the table given below,

Serial No.	Module Name	High Level Requirements	Detailed Requirement	Test Case Name	Automation Script Name
1	LOANS	1.1 Personal Loans	1.1.1 ...	CBO_PL_Approval	CBO_PL_Approval
			1.1.2 ...	CBO_PL_Eligibility check	CBO_PL_Eligibility check
		1.2 Home Loans	1.2.1 ...	...	..
			1.2.2 ...	...	...
2	AMOUNT TRANSFER	2.1 FAN text field	2.1.1 Should accept only 10 digit integer	...	Manual
			2.1.2 Should accept only those numbers which are created by manager	...	Manual
		2.2 TAN text field	2.2.1 Should accept only 10 digit integer	...	...

			2.2.2 Should accept only those numbers created by manager	...	...
		2.3 Amount Text field	2.3.1 Should accept only 100 - 5000, positive integer	...	...
			2.3.2 Should not accept more than balance	...	Manual
...	...	...	...	...	...

TM gives traceability from high level requirements to automation script name.

### Advantages of Traceability Matrix

- Ensures that every requirement has atleast 1 test case
- If suddenly requirement is changed – we will be knowing which is the exact test case or automation script to be modified
- We will come to know which test case should be executed manually and which are to be done automatically.

### Who writes Traceability Matrix?

Test Lead gives an empty template and Test Engineer fills it up for his respective modules. Each Test Engineer fills it up for their modules. For ex, TE concerned with Loans feature will fill in the details(test case names) for Loans feature. Similarly other TE(s) do for their features.

Test Lead consolidates the report after everything has been filled up.

### *Interview Tips and Questions*

#### *1) What is Traceability Matrix ?*

*Ans) Tell – it gives traceability from high level requirements to automation script name*

*If the interviewer tells its wrong then answer like below,*

*a) it gives traceability like : CRS – SRS – FS – Test Case name – Automation Script Name*

*b) CRS – usecase – Test Case name – automation script name*

#### *2) What is the difference between Test Case Review and Traceability Matrix ?*

*Ans) Test Case Review – here, we verify whether all the scenarios are covered for specific requirements.*

*Traceability Matrix – here, we ensure that every requirement has got atleast one test case.*

### What is Test Strategy?

It is a document which captures the approach on how we go about testing the product.

The approach can be from ,

a) resource point of view

b) automation or not

..

..

Like this from many points of view, we capture the approach.  
After Test Strategy, only then we start writing the detailed Test Plan and continue further.  
The Test Strategy is reviewed by development team and after changes are made, it is approved.

## **TEST EXECUTION**

Here, we test the product.

We test repeatedly for 40 – 60 cycles. We do all types of testing on the application. Test Execution is the phase where we spend 80% of our time on the project. Only 20% is spent on the remaining stages.

## ★ ★ ★ **DEFECT TRACKING** ★ ★ ★ (30% of Interview Questions comes from this topic)

**Defect** : If a feature is not working according to the requirement, it is called a defect.  
Deviation from requirement specification is called as defect.

### **INTERVIEW QUESTIONS**

*1) What is the difference b/w defect, bug, error and failure?*

*Ans)*

***BUG** is informal name given to the defect.*

***ERROR** – it is a mistake done in the program because of which we are not able to compile or run the program*

***FAILURE** – defect leads to failure or defect causes failure. Chances are there 1 defect might lead to 1 failure or multiple failures.*

*Take the example of Amount Transfer feature not working for end users when end user tries to transfer money – submit button is not working. Thus, this is a failure.*

***DEFECT** – The variation between the actual results and expected results is known as defect.*

A bug occurs **only** because of the following reasons,



- **Wrong implementation** :- Here, wrong implementations means coding. *For ex*, in an application – when you click on “SALES” link – it goes to “PURCHASE” page – this occurs because of wrong coding. Thus, this is a bug.
- **Missing implementation** :- We may not have developed the code only for that feature. *For ex*, open the application – “SALES” link is not there only – that means feature has not been developed only – this is a bug.
- **Extra implementation** :- The developer develops extra feature which is not needed and not there in the requirements also. *For ex*, consider the below application

In the above application – requirements say only to develop “submit” and “cancel” button – but the developer also develops “help” button which is not there in the requirements.

- If we develop extra features not there in the requirements, it leads to unnecessary extra effort
- Chances are there that adding extra features might affect other features.

Now, let’s understand **“WHAT IS DEFECT TRACKING”**

Developer develops the product – test engineer starts testing the product – he finds a defect – now the TE must send the defect to the development team.

He prepares a defect report – and sends a mail to the Development lead saying “bug open”.

Development lead looks at the mail and at the bug – and by looking at the bug – he comes to know to which development engineer developed that feature which had a bug – and sends the defect report to that particular developer and says “bug assigned”.

The development engineer fixes the bug – and sends a mail to the test engineer saying “bug fixed” – he also “cc mail” to the development lead.

Now the TE takes the new build in which the bug is fixed – and if the bug is really fixed – then sends a mail to the developer saying “bug closed” and also “cc mail” to the development lead.

Every bug will have an unique number.

If the defect is still there – it will be sent back as “bug reopen”.

We should also send a copy of the defect report to the TL. Why do we do this ? Because,

- He should be aware of all the issues that are there in the project
- To get visibility (i.e, he should know that we are working)

90% of projects – we don’t take permission from Test Lead to send bugs to development team.

Around 10% of projects, we take permission because,

- **Customer is new** – for ex, Reliq has a testing team which is testing a product developed by Vodafone developers. We cant send all sorts of major, minor and critical bugs to their development team. So test lead first approves the defect and then sends it to the development team saying it's a valid bug.
- **When we are new to the project**

When we should send defects to development team? – as soon as we catch the defect, we send it to development team.

Why do we send it immediately?

- Otherwise someone else will send the defect (common features)
- Development team will have sufficient time to fix the bug if we send the bug asap.

**NOTE**

For which and all types of testing do we write test cases ?

**Smoke Testing** – here, we are testing basic features only. So we can pull out some test cases which has all the basic features – so we don't have to write test cases for this.

**Functional Testing** – yes, we write test cases

**Integration Testing** – yes, we write test cases

**System Testing** – yes, we write test cases

**Acceptance Testing** – yes, customer may write test cases

**Compatibility Testing** – we don't write because the same test cases as above are used for testing on various platforms

**Adhoc Testing** – we dnt write because they are “on the fly” or “random” ideas/scenarios. However, if we find critical bug, then we convert that scenario into test case.

**Performance Testing** – we may not write test cases because we do this using tool.

**Usability Testing** – we use standard checklist. We don't write test cases because we are just testing the look and feel of the application here.

**Accessibility Testing** – here also, we use checklist

**Reliability Testing** – we don't write manual test cases and we use automation tool

**Regression Testing** – Yes – we have FT, IT, ST test cases. No – because we use the same test cases.

**Recovery Testing** – Yes, we write test cases. We check how the product recovers from crashes

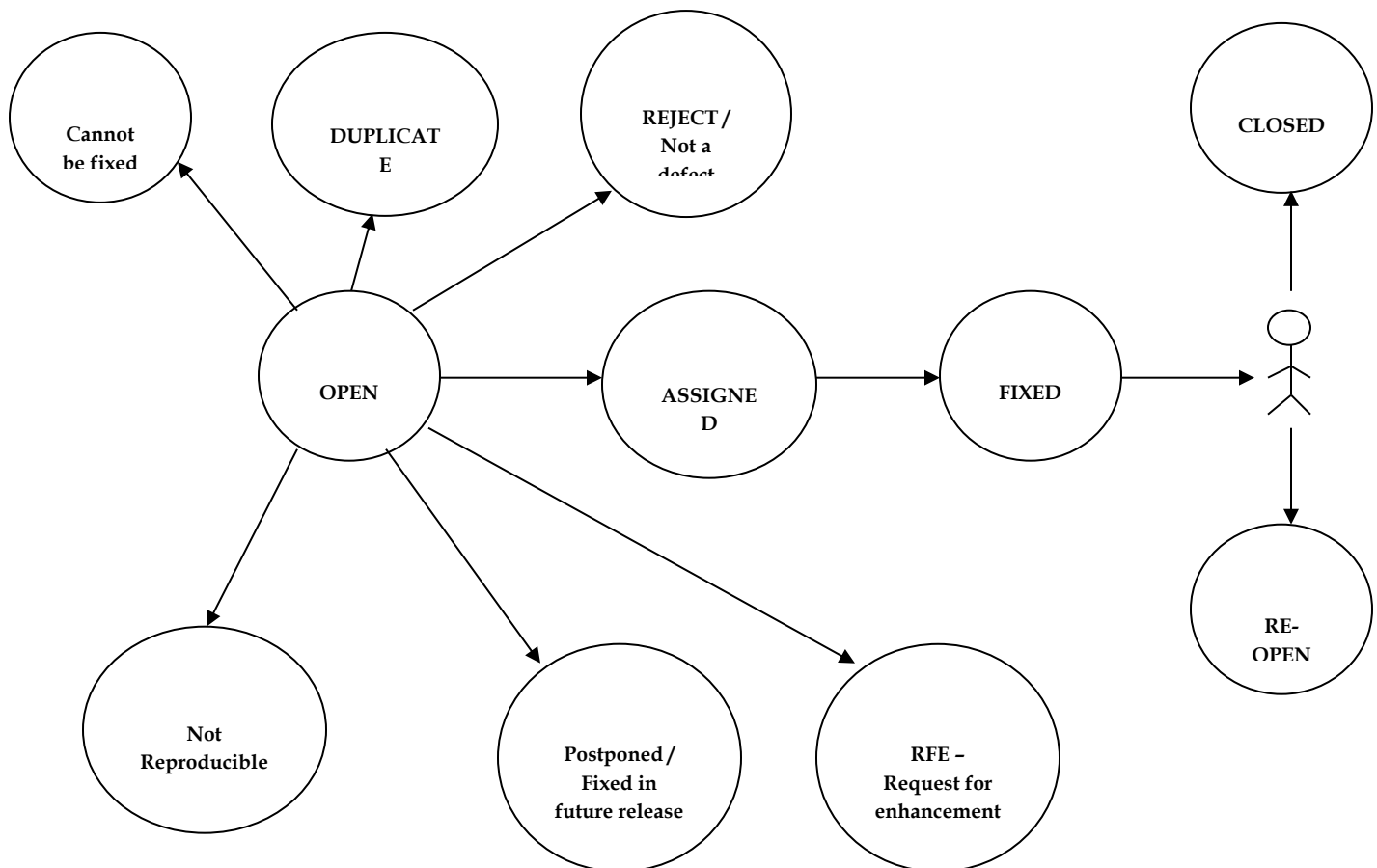
**Security Testing** – yes, we write

**Globalization Testing,**

L10N testing – yes, we write

I18N testing – yes, we write

## ★★ DEFECT LIFE CYCLE ★★



Customer gives requirements – developers are developing the s/w – testing team is writing test cases looking at the requirements

Developer develops the product – test engineer starts testing the product – he finds a defect – now the TE must send the defect to the development team.

He prepares a defect report – and sends a mail to the Development lead saying “bug open”.

Development lead looks at the mail and at the bug – and by looking at the bug – he comes to know to which development engineer developed that feature which had a bug – and sends the defect report to that particular developer and says “bug assigned”.

The development engineer fixes the bug – and sends a mail to the test engineer saying “bug fixed” – he also “cc mail” to the development lead.

Now the TE takes the new build in which the bug is fixed and re-tests it again – and if the bug is really fixed – then sends a mail to the developer saying “bug closed” and also “cc mail” to the development lead.

Every bug will have an unique number.

If the defect is still there – it will be sent back as “bug reopen”.

### **“REJECT” BUG**

Now, when the TE sends a defect report – the Development Lead will look at it and **reject the bug**.

**Bug is rejected because,**

#### **1) Misunderstanding of requirements**

**2) While installing or configuring the product** – wrongly configured or installed the product and we found a bug in the product – send it to development team – developer says “reject” because he looks at the defect report and comes to know that the Testing team has not installed the product correctly.

#### **3) Referring to old requirements**

Chances are there that Testing team may be referring to old requirements while testing the product.

*For ex,* - in the 1<sup>st</sup> build – developers have given the product for testing in which a **sales link button** is there, test engineer does testing and reports any defects – in the 2<sup>nd</sup> build, customer has given a requirement change to the development team where-in he has asked them to remove the **sales link button** – but the testing team is not aware of this change – so when the development team gives the new build – obviously they would have removed the **sales button** – test engineer when he looks at the feature – he is referring to old requirements which has **sales link button** – but in the new requirements it has been removed which the test engineer is not aware of – so he reports a bug – development team then reply back saying “refer to new requirements in which sales link button has been removed”.

#### **4) Because of extra features**

Consider the example and figure shown in Pg 130.

Here, “**help**” button is a feature which has been developed by the developer as an extra feature not part of he requirements – obviously when the TE looks at the application and requirements, since Help feature is not there – he reports it as a bug – the developer rejects it saying “it is an extra feature and that he wont test it” – Test Engineer replies back by re-opening the bug saying “update the requirements” – updating the requirements requires lot of running around for the development team, talking to the customer and all that – so he either removes it or talks to the customer.

### **“DUPLICATE” BUG**

The TE finds a bug and sends a defect report and also assigns the bug report a number – let’s say he sends a bug report numbered as Bug 25 – now the developer replies back by saying “Bug 25 is duplicate of Bug 10” i.e, another Test engineer has already sent that bug earlier and it is being fixed.

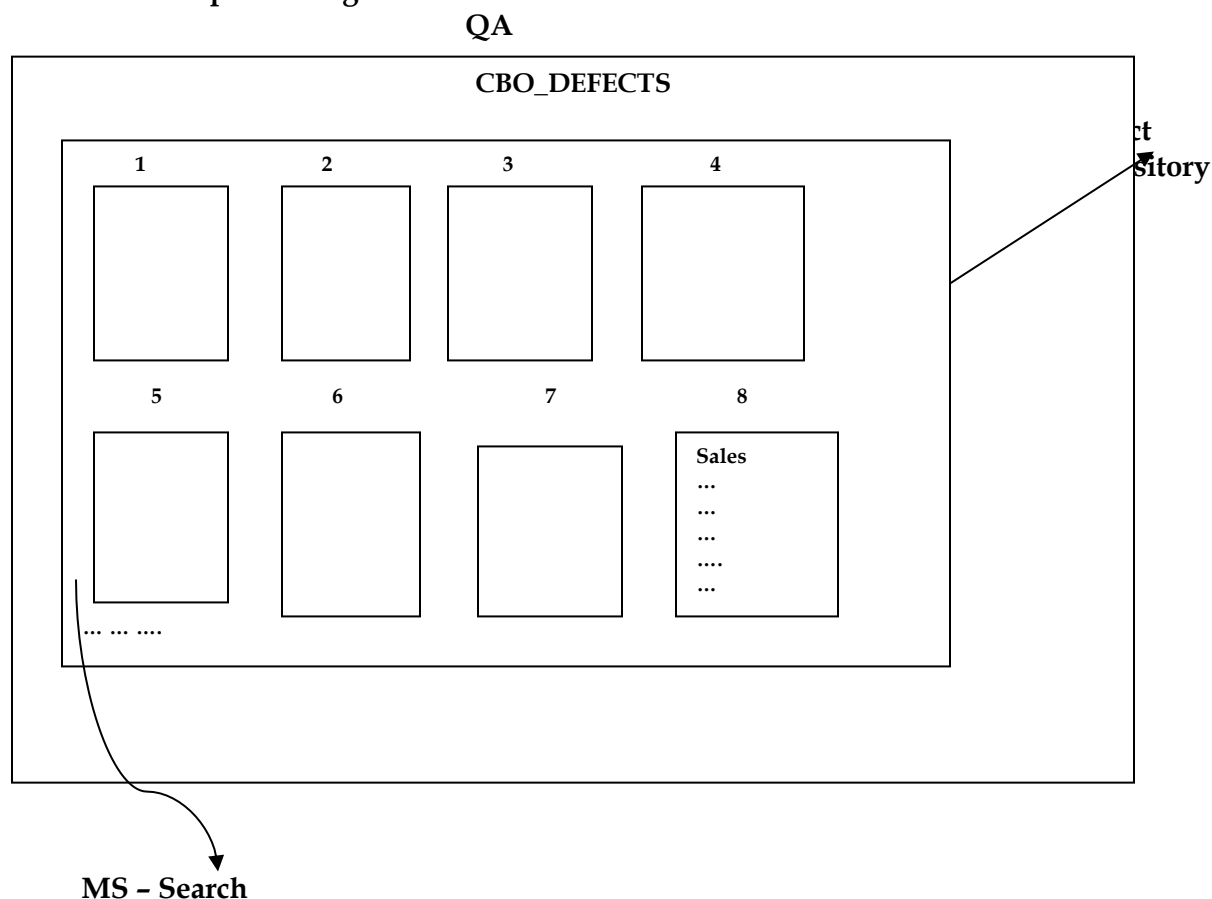
#### **Why do we get duplicate bugs?**

- *Because of common features* – Lets say Test Engineer A and B are testing an application – Test Engineer A and B to test their features must Login to the application using valid username and password before they can go deep into the application and start testing their features – A enters valid username and password and clicks on Login button – it goes to a blank page – this is a bug – so A prepares a defect report for this bug and sends it to developer – After sometime, B also tries to login

and finds the same bug – he also prepares a defect report and sends it to developer – but developer sends back defect report of B saying “it is a duplicate”.

- *B finds bug in A's module* – let's say A and B are testing their respective features – A is doing functional testing and integration testing on all his features – Similarly B is also testing all his features – In one scenario, B needs to go to A's feature(say m12) and send data to B's feature(say m23) – when B clicks on m12, he finds a bug – although its feature of A, he can still prepare a defect report and send it to developers – and B sends the defect report to development team – now, A after testing all his features comes to his feature m12 – he also finds the same defect and sends a defect report – but the developers send it back saying “it is duplicate”.

## How to avoid duplicate bugs?



Let us consider Test Engineer A is testing Sales module and finds a defect where-in – he enters all the information and then when he clicks on Submit button, it is going to Blank page – he prepares a defect report in which he explains the defect.

Now, before he sends it to developer – A logs into a server named QA – there, he goes into a folder called “Defect Repository” which has all the defect reports prepared by TE(s) and sent to developers and clicks on MS-Search and gives Sales as the search keyword and hits on search button – now, it starts looking inside the folder if there are any defect report which has Sales in it – if the search results in No files – then A will first copy-paste his defect report in the defect repository and then send the defect report to the development team who then assign it to the development engineer.

Now, after a few days – another TE B finds the same defect and he also prepares a report – but before he sends it to development team. He first logs onto QA and Defect Repository and searches for Sales keyword – he finds the report which was earlier sent by TE A – so he doesn't send the defect to development lead.

But, always – the TE if he finds a defect similar to his if the Search operation yields some results – before he concludes that his defect has already been sent – he must first read the defect report and ascertain whether it is the same bug which he has found – chances are there that it might have the keyword Sales, but it is about some other defect in Sales feature.

**For ex,** - TE A is testing password text field – he finds that it is accepting 12 characters also (when requirement says “it must accept only 10characters”) – that's a bug. So he prepares a defect report which has the keyword password and stores it in the defect repository and also sends it to Dev team. Now, another TE B is also testing password field – he finds there is a bug where-in password text field is also accepting blank space character – before he prepares a report, he goes to the defect repository and searches for Password keyword – there he finds the earlier report sent by A – but that's a different defect – he reads it and realizes that it's not the same defect which he has found – so, he prepares a defect report regarding the blank space acceptance in the password text field – and stores it in the defect repository and also sends it to the development team.

### **CANNOT BE FIXED**

Chances are there – Test Engineer finds a bug and sends it to Development Lead – development lead looks at the bug and sends it back saying “cannot be fixed”.

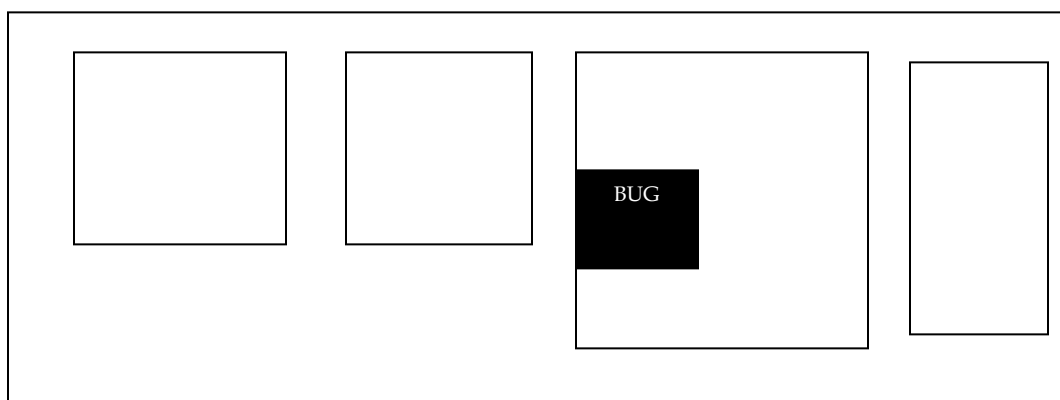
Why does this happen? – Because,

- Technology itself is not supporting i.e, programming language we are using itself is not having capability to solve the problem
- Whenever there is a bug in the root of the product. If it's a minor bug, then development lead says “cannot be fixed”. But, if it's a critical bug, then development lead cannot reject the bug
- If the cost of fixing the bug is more than the cost of the bug itself – cost of the bug means loss incurred because of the bug.

### **POSTPONED or Fixed in the next release**

**a)** We find a bug during the end of the release (could be major or minor but cannot be critical) – developers won't have time to fix the bug – such a bug will be postponed and fixed in the next release and the bug status will be “open”

**b)**



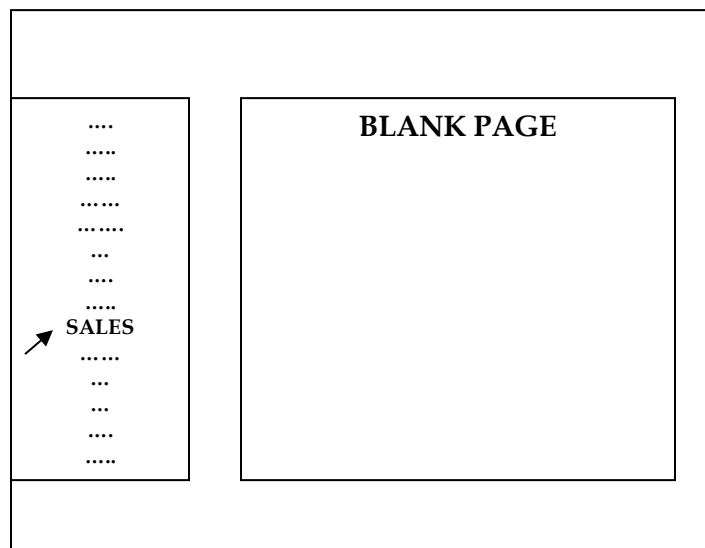
Developers have built the above application – Test Engineers are testing the application and they find a bug – now, the bug is sent to the development team.

Development team looks at the report – and replies back to the testing team saying “they will not fix the bug for now, as the customer is thinking of making some changes to that module or he might as well ask for a requirement change where-in the module itself might have to be removed” – so they won’t fix the bug for now – but if no requirement change happens, then they will fix the bug.

c) If the bug is minor and it is in feature exposed to internal users.

**For ex,** consider Citibank employee using a s/w where-in he is making keeping track of accounts and customer details – now, for example if the “Sort by name” feature is not working – it’s ok because it’s a minor bug because the development team can always fix the bug during the next release.

### NOT REPRODUCIBLE

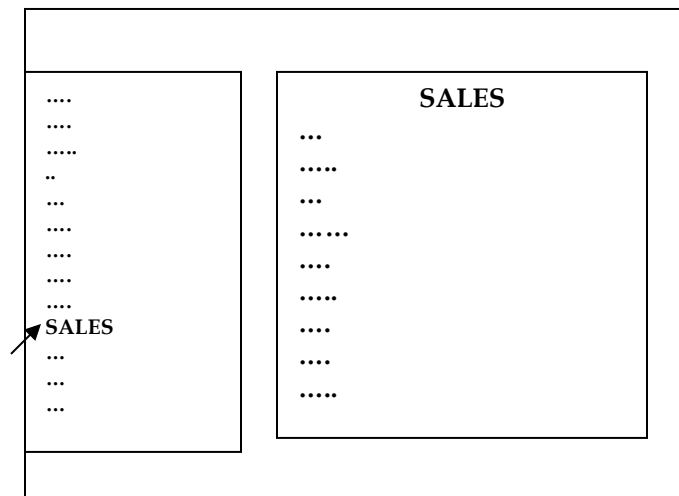


**Open the application in Mozilla FireFox**

- 1) Open browser and enter [www.citibank.com](http://www.citibank.com) and click on Go button
- 2) Login using valid username and password
- 3) In the homepage, click on “SALES link button”
- 4) It goes to blank page

### **DEFECT REPORT**

Let us consider the above application – the TE is testing the above application in Mozilla Firefox– when he clicks on SALES link button – it goes to a blank page – this is a bug – the TE prepares a defect report and sends to the development team.



**Open the application in Internet Explorer**

The development lead looks at the defect report and opens the application in Internet Explorer – when he clicks on Sales link button, Sales page opens and the application works perfectly.

Then, what was the problem? – the answer is, the TE didn't specify in the defect report in which browser to open the application – so, the application didn't work for the TE in Mozilla FireFox – but, when the developer opened it in Internet Explorer, the application worked totally fine.

Since the developer is not able to see any defect – he replies back by saying “I am not able to find the defect” or “I am not able to reproduce the defect”.

### Why we get “not reproducible” defects?

1. Because of platform mismatch
2. Because of improper defect report
3. Because of data mismatch
4. Because of build mismatch
5. Because of inconsistent defects

#### 1) Because of platform mismatch

- Because of OS mismatch
- Because of browser mismatch
- Because of ‘version of browser’ mismatch
- Because of ‘settings of version’ mismatch

#### 2) Because of improper defect report

Example to explain this - take figure and explanation in Pg 136 and 137. Here, the TE must have specified to open the application in Mozilla Firefox

Let us consider another example shown below to explain “improper defect report”

The screenshot shows a web form titled "USER DETAILS". It contains several input fields, some of which are obscured by "....". Below the input fields is a checkbox labeled "REMEMBER PASSWORD" which is checked. At the bottom of the form are two buttons: "SUBMIT" and "CANCEL".

- 1) Open browser and enter [www.citibank.com](http://www.citibank.com) and click on Go button
- 2) Enter user details and click on Submit button
- 3) It goes to blank page

#### DEFECT REPORT

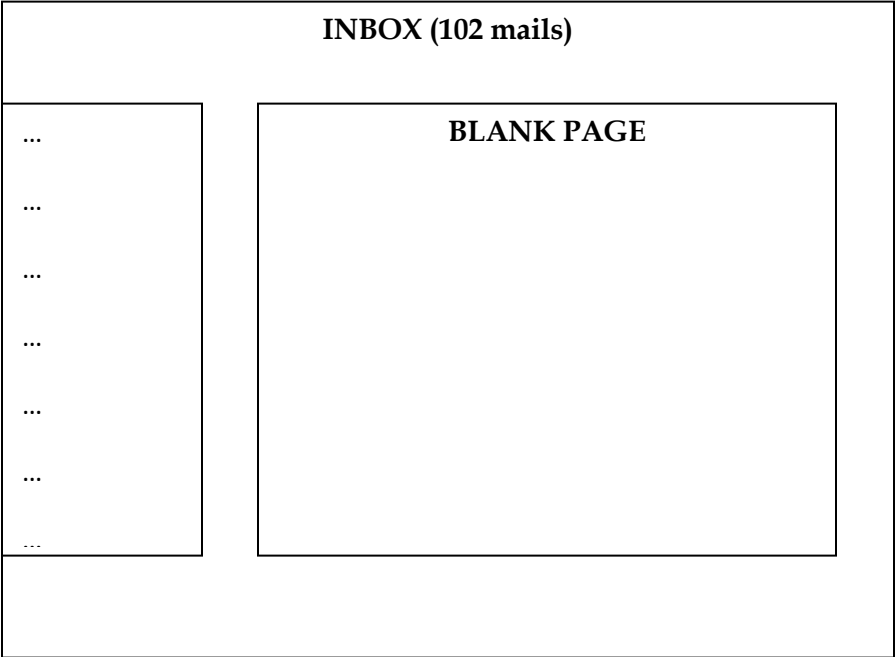
In the above application – TE is testing the above application – when he enters all the user details and selects checkbox and clicks on Submit button – it goes to blank page – but, when the TE does not select the checkbox and clicks on Submit button, a new user account is created, i.e, it works fine. The TE prepares a defect report and sends it to development lead.



The Dev team looks at the report(as shown in box) – and tests the application – it is working absolutely fine. Then what happened? – Observe the defect report sent by the testing team – nowhere as he mentioned to “select checkbox and click on submit button” – thus because of the improper defect report – the development team is not able to reproduce the defect.

**3) Because of data mismatch**

Consider the example shown below,



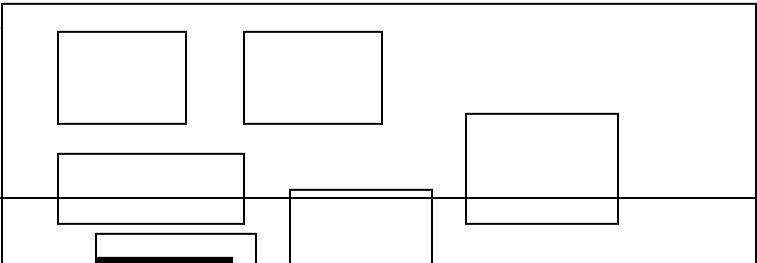
**Login as ABC and click on Inbox in homepage of ABC**

The above email application has been developed – the TE starts testing the above application – after testing and testing the above application using username ABC and valid password – there are almost 101 mails in ABC’s inbox – now, again when the TE opens the application and goes to test the application – no mails are being displayed when he clicks in Inbox and instead he gets a blank page. So – the defect he has found is that – when the number of mails in Inbox exceeds 100 mails , he gets a blank page – so he prepares a defect report as shown in the box below,

- 1) Open the browser and enter the following URL [www.email.com](http://www.email.com)
  - 2) Login using valid username and password
  - 3) Click on Inbox
  - 4) Blank page is displayed

The development team looks at the defect report – he logs in to the application using username XYZ and clicks on Inbox – no blank page is displayed and the Inbox is working fine. What happened here? – The TE should have mentioned in the defect report as “Login as ABC” or “Login to any mailbox which has more than 100mails”. Thus, the developer does not find any defects and thus he mentions it as “not reproducible defect”.

**4) Because of build mismatch**



Developer develops build 1 and gives it for testing – in Build 1, there are 2 defects – bug 1 and bug 2 – TE while testing finds bug1 and reports it to development team – development team when they fix bug1, bug2 automatically gets fixed – development team is developing build 2 and gives it to testing team for testing – testing team just when it finishes testing build1 – he finds Bug 2 and reports it to development team – the development team however do not find any defect as they are testing build 2 – so they report back saying “bug is not reproducible”.

**5) Because of inconsistent defects**

To explain this, let us consider an example : When TE is testing an email application – he composes mail in User A and sends it to User B – he then logs out from user A and logs into User B and checks B’s inbox – but no mail is there in the Inbox – thus he finds a defect – now, just to confirm the defect before sending a defect report – the TE again logs into User A and sends a mail to User B – he then logs out from User A and logs into User B and checks B’s inbox – but, this time the mail is there in B’s inbox!!  
Thus, we don’t know when the defect comes and when the feature works fine. This is called inconsistent defect.

**REQUEST FOR ENHANCEMENT (RFE)**

Test engineer finds a bug and sends it to development team – when development team look at the report sent by the TE – They know it’s a bug, but they say its not a bug because its not part of the requirements. Let us consider the **example** shown below,

CREATE USER

CREATE USER

...

...

...

...

...

...

...

...

...

....

SUBMIT

CANCEL

In the above **example**, the TE is testing the above fields. After he enters all the data into the required fields, he realizes that he needs to clear all the data – but there is no **CLEAR button** – and he has to manually clear all the fields. He reports this as a bug – but the development team will look at the defect report and see that

**clear button** is not mentioned in the requirements – so they don’t reject or close the bug – instead they reply back to TE saying that it is RFE.

Developers agree that it is a bug – but will say that the customer have not specified these details and hence will give it a name as RFE.

If a defect is known as RFE, then we can bill the customer. Otherwise, development team needs to fix the bug

Development team always have a tendency to call a defect as RFE, so a TE needs to check the justification given by development team – if it is valid, then he will accept it as a RFE – but if it is not, then he will respond to the development team with proper justification.

### *INTERVIEW QUESTIONS & TIPS*

#### *1) Explain Defect Life Cycle ( \*\*\*\* VERY VERY IMPORTANT QUESTION \*\*\*\*)*

*When TE is testing the product – he finds a defect – he prepares a defect report and creates a status as “OPEN” – and sends it to development lead – development lead looks through the defect report – and if the defect is valid – he then finds out who developed that module in which the defect has been found and assigns the defect to that development engineer – the development lead also changes the status of the defect report to “ASSIGNED” – the development engineer fixes the defect and changes the status to “FIXED” – the build is then sent to the TE for re-testing – the TE tests the fix – and if the defect has been really fixed – then he changes the status of the defect report to “CLOSED” – if the defect is still there, he changes the status to “RE-OPEN”.*

*Sometimes it so happens that TE finds a defect and sends it to development lead – development lead looks at the report and rejects the report and he explains why it has been rejected,*

*REJECT (explain)*

*DUPLICATE ( explain Duplicate)*

*CANNOT BE FIXED (explain)*

*POSTPONED (explain)*

*NOT REPRODUCIBLE (explain)*

*RFE (explain)*

### DEFECT REPORT

**Defect ID** – it is an unique number given to the defect

**Test Case Name** – whenever we find a defect, we send the defect report and not the test case to the developer. For defect tracking, we only track the defect report and not the test case. Test case is only for reference for the TE. We always only send the defect report whenever we catch a bug.

When we are doing ad-hoc testing – no test case is written for ad-hoc testing because they are “out of the box” testing scenarios – if we find a critical bug – then we convert that ad-hoc scenario into a test case and send it to development team.

Given below is – how a defect report looks like – it is a MS – WORD file.

**DEFECT ID :** BugID \_ 1578

**RELEASE NAME :** Tiger

**BUILD ID :** B03

**MODULE NAME :** Sent Items

**STATUS :** Open

Assigned

Fixed

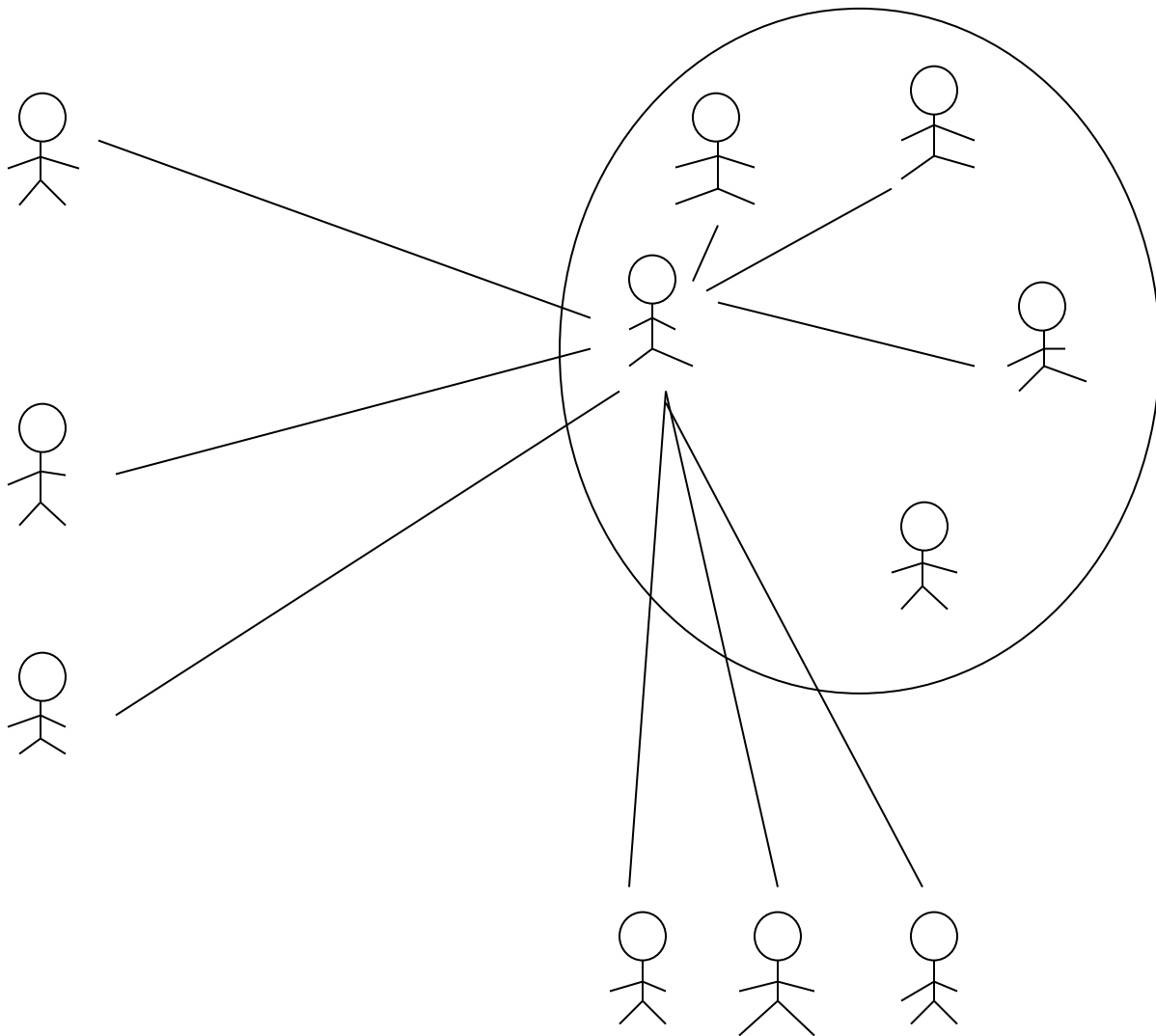
Closed

Duplicate

**INTERVIEW TIPS**

*Q) What are the attributes of a defect report ? (OR) What is the mode of communicating a defect ?  
Ans) the answer for the above questions are – just explain the defect report.*

## Defect Evaluation Team / Bug Counseling Team



Let us consider the above figure to explain about **Bug review meeting / Bug Counseling**

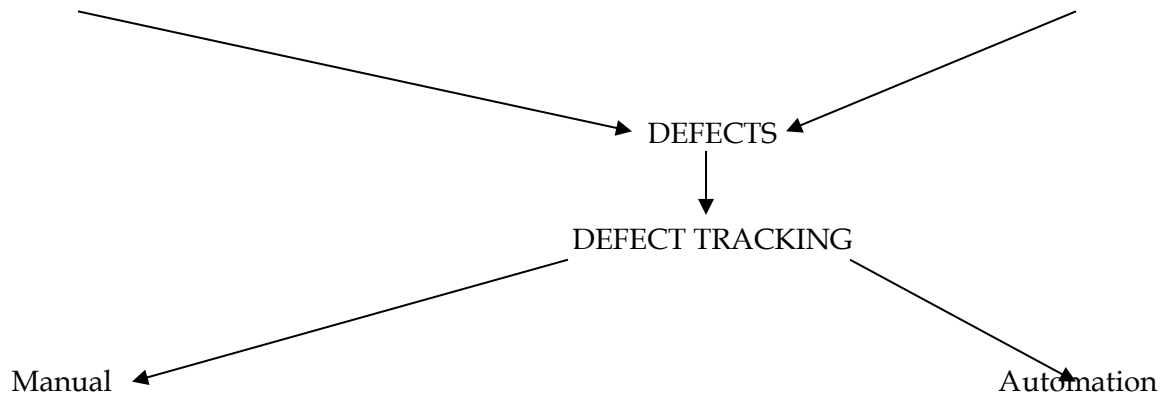
The Defect Evaluation Team (DET) consists of Development Lead, Development Manager, Senior Development Engineers and Test Lead.

They will have a group mail id – say – [xyz\\_det@abc.com](mailto:xyz_det@abc.com) – Now, whenever TE finds a bug, he sends it to DET – In DET, only the Development Lead (DL) looks into the bug report and assigns the bug – although the mail goes to everybody in the DET, only DL will check it and assign the bug. This process continues from Monday morning upto Friday afternoon – by this time about 100 bugs have been reported by TE(s) – out of these 100 bugs, 60 have been assigned and the remaining 40 are pending.

Now, on Friday evening – DET team will have a meeting where they discuss about the pending bugs and whether to assign the bugs or reject the bugs – but for everything, they will have to give proper justification. This meeting is called Bug Review Meeting / Bug Counseling. By the end of the meeting (i.e, 1<sup>st</sup> cycle) – the status of all the pending bugs will changed from pending to other status(es).

Manual

Automation



### **How to track a defect manually?**

- 1) Find a defect
- 2) Ensure that it is not duplicate (i.e, verify it in Defect Repository)
- 3) Prepare defect report
- 4) Store it in defect repository
- 5) Send it to development team
- 6) Manage defect life cycle (i.e, keep updating the status)

### **Tracking of defects using Automation (i.e, Defect Tracking Tool)**

The various tools available are,

- Bugzilla
- Mantis
- Rational Clear Quest
- TeleLogic
- Bug\_track
- QC - Quality Center - it is a test management tool - a part of it is used to track the defects.

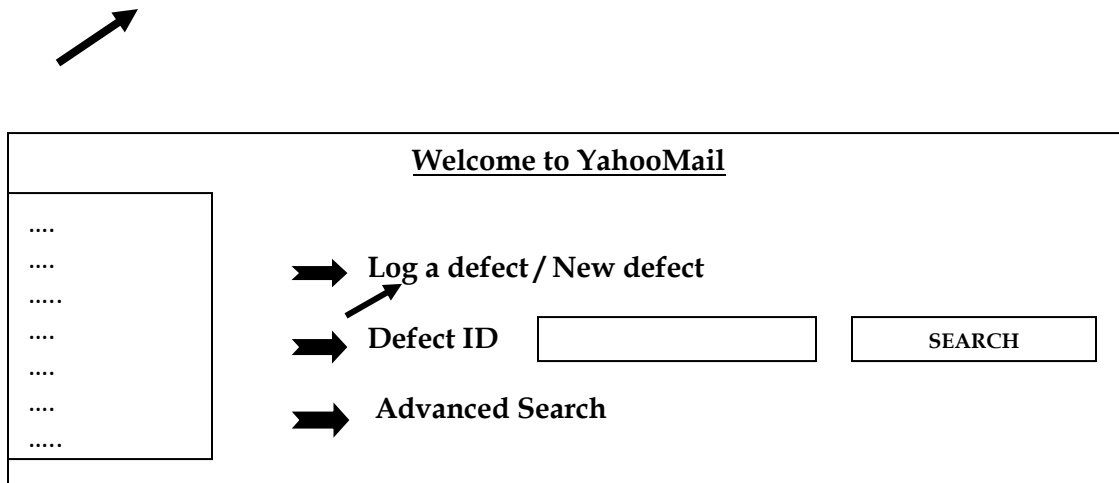
There are **2 categories of tools**,

- 1) Service based - In service based companies - they have many projects of different clients - each project will have a different defect tracking tool
- 2) Product based - in product based companies - they use only one defect tracking tool.

Now, let us see **how the defect tracking tool works**,

The image shows a web login interface. At the top, there is a text box containing the URL "http :// www.hp.com". Below this, there are two rows of labels and input fields: "USERNAME" followed by an empty text box, and "PASSWORD" followed by an empty text box. At the bottom, there are two buttons: "LOGIN" and "CANCEL".

Figure 1



The screenshot shows the YahooMail login interface. At the top, it says "Welcome to YahooMail". On the left, there is a vertical list of links, with an arrow pointing to the top one. In the center, there are three main options, each preceded by a right-pointing arrow: "Log a defect / New defect", "Defect ID" (which is followed by a text input field and a "SEARCH" button), and "Advanced Search".

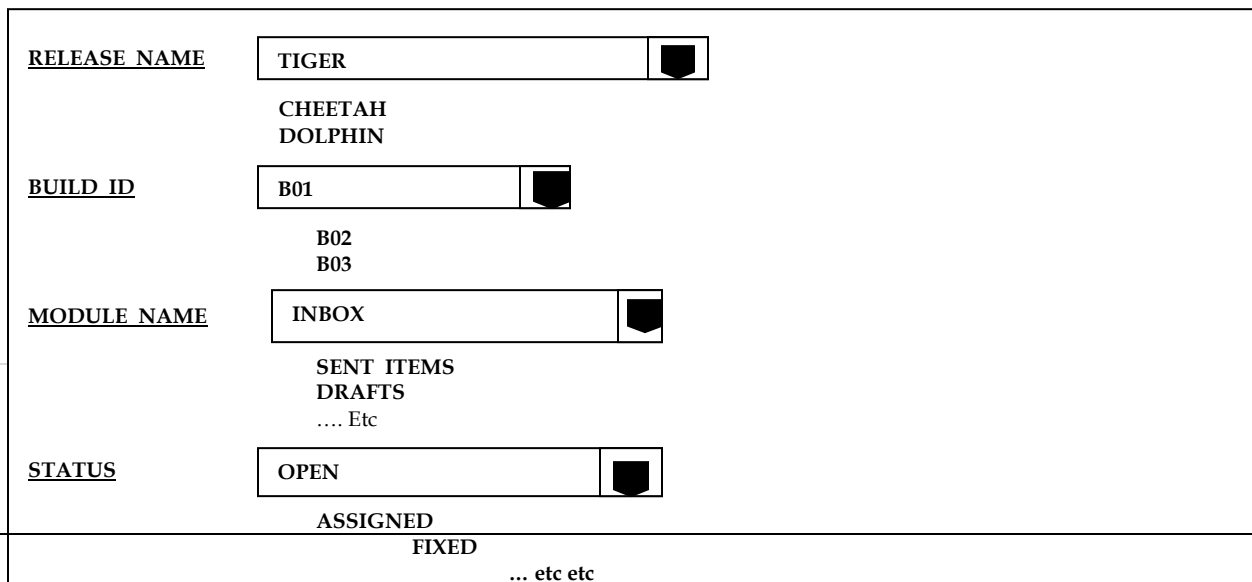
Figure 2

Assume that you are working as a TE in HP and that you are working on YahooMail project. We enter [http : // www.hp.com](http://www.hp.com) as the URL in the browser and give valid username and password and click on **login** button.

The username of your's can be obtained by the following two ways,

- Use **employee id** as username along with the standard password to access the system
- Or send a request to the admin of tool and ask for username and password.

When we login, we go to the page shown in figure 2. We then click on **log a defect**. When we click on this link, we get the following page shown below,



The screenshot shows a form for logging a defect. It has four main sections, each with a label on the left and a dropdown menu on the right. The first section is labeled "RELEASE NAME" and has a dropdown menu with "TIGER" selected, and "CHEETAH" and "DOLPHIN" as other options. The second section is labeled "BUILD ID" and has a dropdown menu with "B01" selected, and "B02" and "B03" as other options. The third section is labeled "MODULE NAME" and has a dropdown menu with "INBOX" selected, and "SENT ITEMS", "DRAFTS", and "... Etc" as other options. The fourth section is labeled "STATUS" and has a dropdown menu with "OPEN" selected, and "ASSIGNED" and "FIXED" as other options. At the bottom, there is a text input field with "... etc etc" as a placeholder.

When we click on **log a defect** link, we get the following page as shown above.

We enter all the details and then when we click on **submit button**, the following happens,

- It generates a bug ID automatically, **ex** - bugID\_1578
- It stores the data in the database
- It automatically sends the mail to the Development Lead and also the copy of the mail goes to the TE himself.

Development Lead opens the mail - he logs in to the tool using his username and password - then he gets the **welcome** page - he then enters the BugID into the **defect id** text box and clicks on **search** button

When he clicks on **search** button - he gets the particular bug report. He looks at the report and after checking it's a valid bug, he then **changes the status in the report to 'assigned'** and when he does that - he gets a **TO text field** as shown in the figure above - there he enters the email id of the concerned developer



who has to fix that bug – and then when he clicks on **submit button** – whatever changes he has made is stored in the database – and the mail goes to the concerned TE, Developer and DL himself.

When the particular developer gets the the BugID, then he searches for the bug using the bugID and after seeing what's the bug – he goes to his module and fixes the bug.

When developer fixes the bug – he goes and changes the status to **fixed** and again he submits it. This is also saved in database. Then mail goes to TE, DL and to the developer himself.

TE now opens the same bugID and checks the status whether it is fixed or not – if it is changed to **fixed** – then he re-tests the bug and if it is really fixed, then he goes to status and changes it to **closed** – if the bug is not fixed, then he changes the status to **reopen**.

This comprises the entire defect life cycle.

As soon as TE submits the bug report, then he or anybody cannot change two fields in the entire report – they are 'brief description' and 'detailed description'.

If any changes are to be made in these fields – then they must be mentioned in the **comments** field.

If a new defect is found within a old defect within a short period (say 10days), then instead of opening a new defect – he can open the old defect and login the new defect in the **comments** page.

### *How to check for duplicate bugs ?*

When developer changes the status to **duplicate**, then the TE should check whether the previous bug & the sent bug is same or not.

To check whether its duplicate or not, click on **advance search** & get it confirmed if the bug is duplicate or not. If it's not duplicate, then TE should give proper justification.

Click on **Advanced Search** (see figure next page)

To avoid duplicate bugs, go to Advanced Search.

When ever we catch a bug – Before logging the bug to developer for fixing – first go & check whether it is logged before or not. To do so, click on **Advanced Search** & enter the data in the **text** field & click on **Search**. You will get the bug ID(S). if you enter '**password**' & search, then it will give you different bug ID having **password** text. We must go & check it & log it for fixing if its not logged before.

Defect found by Testing team should never be closed just like that by the Development team. TE looks the product from customer point of view – so if a developer says it's a minor bug, testing team always considers it as major bug.

#### SEARCH BY

Release Name

Module Name

Status

Priority

Severity

Found By

### RESULT SHEET

<u>Release Name</u>	<u>BugID</u>	<u>STATUS</u>	<u>Severity</u>	<u>Priority</u>
TIGER	BugID 1578	OPEN	...	...
TIGER	BugID 1890	CLOSED	...	...
TIGER	BugID 1235	ASSIGNED	...	...

### SEVERITY of a Bug

**Severity** is impact of the bug on customer's business.

- **Critical** – A major issue where a large piece of functionality or major system component is completely broken. There is no work around & testing cannot continue.
- **Major** – A major issue where a large piece of functionality or major system component is not working properly. There is a work around, however & testing can continue.
- **Minor** – A minor issue that imposes some loss of functionality, but for which there is an acceptable & easily reproducible workaround. Testing can proceed without interruption.

### Blocker Defect

There are 2 types in **blocker** defect,

- ✓ **Major flow is not working** – Login or signup itself is not working in CitiBank application
- ✓ **Major feature is not working** – Login to CitiBank. Amount Transfer is not working

### PRIORITY of a Bug

It is the importance to fix the bug (OR) how soon the defect should be fixed (OR) which are the defects to be fixed first.

- **High** – This has a major impact on the customer. This must be fixed immediately.
- **Medium** – This has a major impact on the customer. The problem should be fixed before release of the current version in development
- **Low** – This has a minor impact on the customer. The flow should be fixed if there is time, but it can be deferred with the next release.

Development team will fix the **high priority** defects first rather than of **high severity**.

Generally, severity is assigned by Tester / Test Lead & priority is assigned by Developer/Team Lead/Project Lead.

---

### EXTRA(s) Information

#### Why does a software have bugs ?

- ✓ Mis-communication OR No communication -> as to specific of what an application should or shouldn't do(the application's requirements)
  - ✓ Software complexity
  - ✓ Programming errors
  - ✓ Changing requirements
  - ✓ Time pressure
- 

#### When do we stop the testing ?

- ≈ When the time span is less, then we test the important features & we stop it
- ≈ Budget
- ≈ When the functionality of the application is stable
- ≈ When the basic feature itself is not working correctly

In the last 2 days of release, the code will be freezed. When the code is freezed, the defect found will be fixed in later stages.

---

### **Why do we do testing ?**

- ✓ To ensure the quality
  - ✓ To verify all the requirements are implemented correctly
- 

We get the build at every 15 days - when the release date is near, then we get the build for every 2 or 3 days.

Identifying the scenario & writing test cases means - we write only system & integration test scenarios

---

**Defect tracking** - the defect which is reported, we track whether it is fixed by the developer or not.

In Test plan / testing methodologies - instead of **functional testing**, we must say **component testing**.

**Test Bed** is the other name for **test data**

---

The **various requirements specifications** given by the customer are,

- ≈ CRS / SRS / FS / Business Specification
  - ≈ Design document
  - ≈ Use cases
  - ≈ Application itself
- 

The **defect report** varies from company to company. But the following are the mandatory attributes of a defect report in all companies,

- ✓ Defect ID
  - ✓ Severity
  - ✓ Priority
- 

**Defect Density** = (Total no. of defects found - total defects fixed) / (Total no. of TC(s) executed)

---

**Verification** - Am I building the product right?

**Validation** - Am I building the right product?

---

### **Bugs found in Prototype testing**

- **Missing Requirements** - company logo is missing in reports
  - Instead of using list property, we used dropdown button or checkbox
  - Links are not placed at right position
  - There may be some orphan files i.e - when we click on particular link, no page is displayed
  - When we click on the link, instead of sales page, report page is opening
- 

## **USABILITY Testing**

Testing the user friendliness of an application is called Usability testing

---

Let us start with an example such that we have two applications A & B which are different but doing the same job. In this, we see which one is user friendly

Given below are some of the parameters we look into for testing. In this most of them are not measurable,

- ✓ Speed
- ✓ Help
- ✓ Navigation should be simple
- ✓ Compatibility
- ✓ Look & feel
- ✓ Features
- ✓ Location of components

One important parameter other than the above said parameters is “**Effort needed to learn the application**”. Some of them define usability testing like above written in green

Suppose in the example, we understand the software A in 2 hrs, but we take 4hrs to understand B. Let us see different cases here,

- since we understand A in 2hrs, it becomes user friendly compared to B
- suppose **look & feel** is not good for A. In this case though, we understand A in 2hrs, we cannot say that A is user friendly.
- Therefore, we look into many parameters before we say user friendliness of a software

### What is LOOK & FEEL

The application should be such that it should be pleasant looking [ suppose if we have red color text in yellow background, definitely we don't feel like using it ] and make a feel to the end-user to use it.

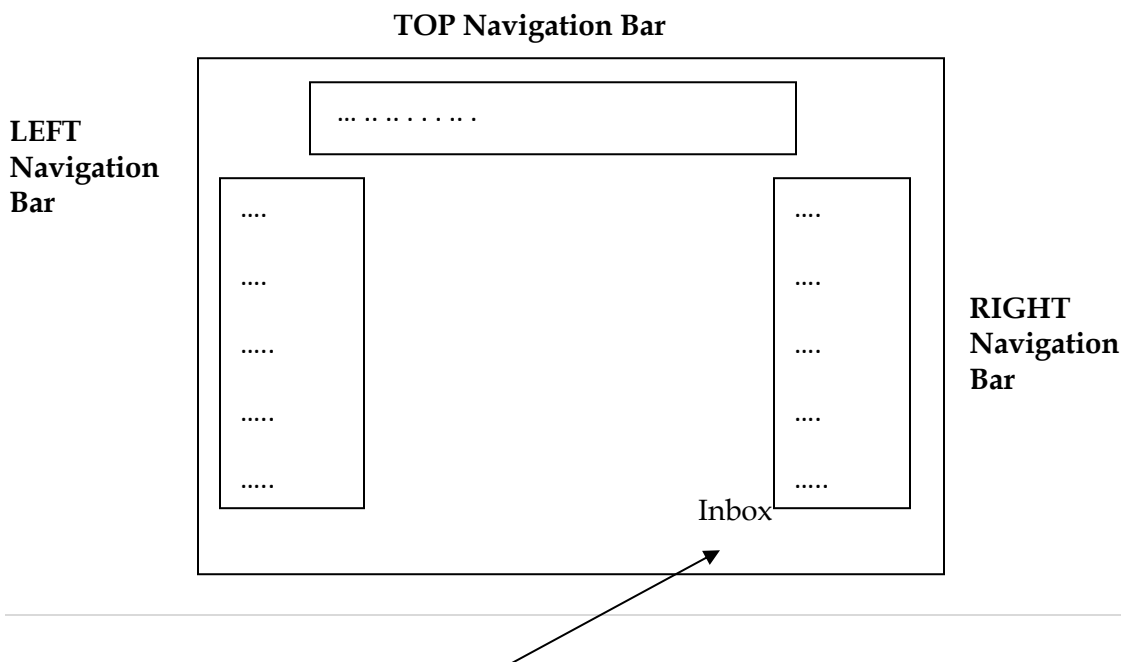
### Let us now see some of the examples of Usability testing

1) Consider some of the examples below,

In this example, we see the Inbox – the most frequently used feature is not kept in the right place.

End – users therefore search for the Inbox & after sometime, they find the Inbox lying somewhere down.

This type of application may lose the customers because they are less user – friendly.



## Search for INBOX – after sometime we find it here

How to overcome this,

See to it that the important features are placed in the **left navigation** bar and **top navigation** bar (This is normally followed)

Then we should see whether frequently used features are easily accessible (like Inbox, Compose Mail etc)

While testing, we therefore have to list all the important features before we start testing an application

2) Consider the sketch shown below,

```

    graph TD
      TOOLS --> DRAW
      DRAW --> LINE
      LINE --> Triangle
  
```

The end – user to draw the above sketch every time, he clicks **Tools -> Draw -> Line** for each line and finally he clicks **Tools -> Draw -> Line** to complete the sketch.

In this , we see the end-user wants to draw a simple sketch, but he wastes his time in clicking each time the **Tools -> Draw -> Line** for the sketch. This becomes a waste of time for the end-user.

Therefore, while testing , we must make user friendly of the application by creating **Tool bar** which makes the job simple by clicking on the necessary tools instead of going & clicking each time **Tool -> Draw -> Line**

3) Suppose we have a shopping website which has 10 pages of product like shown below,

The question here is – after going to the 10<sup>th</sup> page – how do we come back to the 5<sup>th</sup> page ? – Do we click

**back – after – back – back -> NO ->** then how do we do – Observe,

To make user friendly application, we always have **Bread Crumbs**

What are **Bread Crumbs** ?

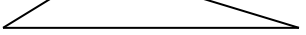
In each page, we will have a link to go to its respective page. Suppose we have seen all the 7 pages & we are in 8<sup>th</sup> page – that 8<sup>th</sup> page will have the link of all the 7 pages – so whatever page (Suppose 5<sup>th</sup> page) you like to visit, you click on the link in the 8<sup>th</sup> page & go to the 5<sup>th</sup> page.

150 | Page

See to it that the important features are placed in the **left navigation** bar and **top navigation** bar (This is normally followed)

While testing, we therefore have to list all the important features before we start testing an application

TOOLS  
↓  
DRAW → LINE

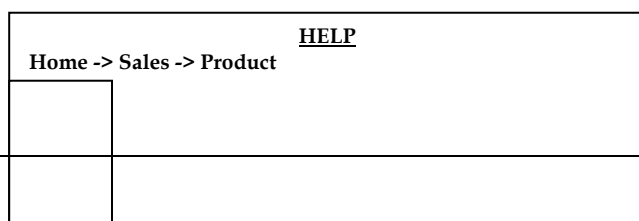
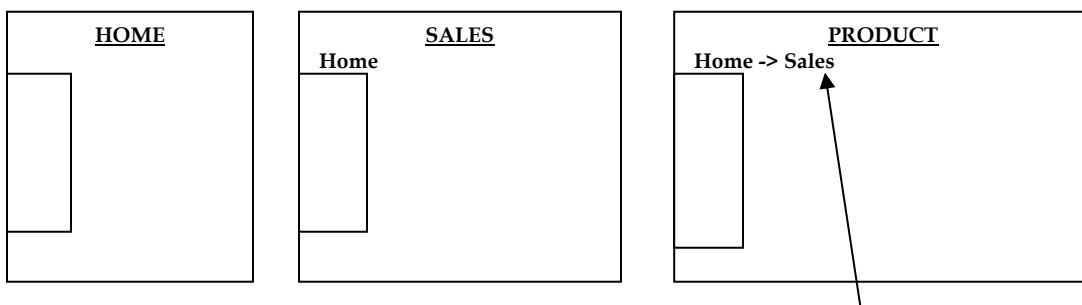
A simple line drawing of a triangle, representing the output of the 'DRAW' command. The triangle is isosceles with a horizontal base and two equal-length sides meeting at a top vertex. It is drawn with thin black lines on a white background.

Therefore, while testing , we must make user friendly of the application by creating **Tool bar** which makes the job simple by clicking on the necessary tools instead of going & clicking each time **Tool -> Draw -> Line**

**HOME**      **SALES**      **PRODUCT**      **HELP**      .....

## What are Bread Crumbs ?

In each page, we will have a link to go to its respective page. Suppose we have seen all the 7 pages & we are in 8<sup>th</sup> page – that 8<sup>th</sup> page will have the link of all the 7 pages – so whatever page (Suppose 5<sup>th</sup> page) you like to visit, you click on the link in the 8<sup>th</sup> page & go to the 5<sup>th</sup> page.



If a developer develops a **pop - up** window, always he should keep the **Yes** button in the beginning. If he swaps this, then there should be some valid reason behind it.

Generally, professional Test Engineer(s) should not do usability testing – the reason is they know where exactly the feature lies & how it works – therefore, for a TE, everything becomes user friendly in that application.

Therefore, only end user should do usability testing.

**For ex,**

The employee / CEO goes & collects the s/w (let us take a **game software**) & distributes to various end users like kids, friends etc. Now these end-user(s) use the games software & gives the feedback to the CEO / employee.

Now this employee / CEO looks into the feedback & sees the major feedback & consolidates all the feedback report collected from the people.

Ex – if a feature for all 50 people has been reported, then that it is considered. If a feature has been reported only by 1 or 2 people, then it becomes minor.

After consolidating, he goes for fixing the bugs.

One thing we have to keep in mind is that all the applications cannot be given to end-users & thus it depends on the application.

**For ex,**

In case of a **banking application** – if we develop an application for manager (call manager as end-user). For others call this respective person as end-users.

In the example above – end-users (MANAGER) starts using the application – 2 TE(s) sits at the back of him & takes report of the defect whether the end-user goes in a right way as the ydevelop as in a wrong way into the application.

Now the manager checks each link before he clicks on it because he knows the TE is watching him.

So the drawback here is – TE came to know they are not getting correct feedback.

Then how to overcome this? – do we fix camera & take feedback? -> in this , the end-user becomes more serious & thus it becomes a drawback.

80% of time, we end up doing usability testing for the reasons above mentioned.

Sometimes TE has to do usability testing for the following reasons,

- Do not want to outsource to other company
- No money to spend on Usability testing

But there are scenarios we do Usability testing where we swap the features among TE and do usability testing.

### **How to conduct Usability testing? (OR) What process should we follow to do Usability testing?**

Prepare OR derive checklist (i.e, what are the points to be checked). If we don't prepare a checklist, we may miss some features in the application.

For Usability testing, we should prepare a genuine checklist specific to the product we develop.

Example of a checklist for an application,

- ✓ For this application, one of the checklists includes color of already checked link should be changed to red
- ✓ All the images should have **alt** tag(Tooltip)
- ✓ All the pages should have link to Homepage
- ✓ Login feature should have **Forgot Password** link

Like the above checklist, we can derive as many checklists as possible based on the application (or) product.

While deriving checklist, we should derive a common checklist which can be executed for all pages.

There is another case where the customer gives the checklist for the application.

---

### **ACCESSIBILITY Testing/ ADA (American Disability Act)/ 508 Compliance testing**

Testing the application from the physically challenged person point of view.

**For ex,** Suppose if a blind person is accessing the Internet – the application should be in such a way that even the physically challenged person should be able to access it without any problems.

When a blind person clicks on anything – the response connected into voice & the person hears & uses it – Response sent should be easily read by the browser & commented in voice.

In the above example, whatever the response is sent to the browser should be easily read – the application should be designed like that – the response sent should be immediately connected into voice. Thus, the blind person can use it without facing any problem

Red & Green color should not be used

The 508 testing has got some rules that should be followed while developing the application. Some of the rules are,

- ✓ All comments should have **Alt** tags
- ✓ Red & Green color objects should not be displayed
- ✓ Should be able to access all components just by using keyboard.

Like this, there are many rules.