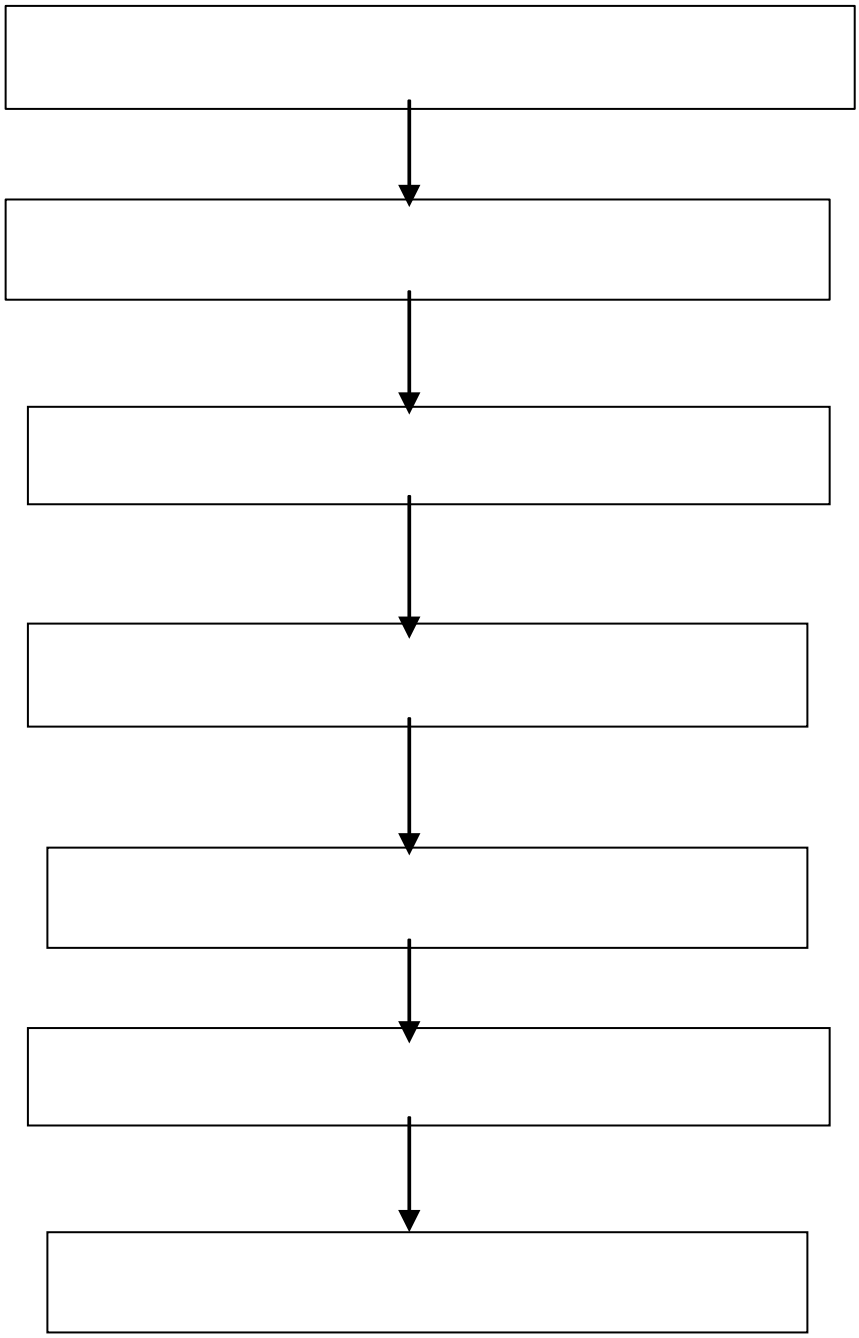
SDLC :- Software Development Life Cycle

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.

Why developers should not test the s/w?

- He is always in over confidence that whatever they build will always works
- They will never see the product from –ve point of view
- Developer somehow they want to build the product they don't want to see the product breaking
- Even though they know that bugs are there they will hide it
- Chances are there to use the time allocated to testing to build the product and they say time is not there

Why requirements keep changing in many of the projects?

- Every software is developed to support the business and the business keps changing so customer requirements keep changing
- Because of competition requirements keeps changing
- To adopt the new technology

Why we will not allow requirement changes in this model?

- As the requirement changes design changes and when this design changes lot of mistakes may happen in the design
- A lot code will changed when the code changes lot of mistakes will happen in the code, to avoid that we frees the requirement.

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 and 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 :-

The 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 freezed 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 freezed 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.

Advantages of the water fall model

- It is simple to adopt
- Since, requirements are freezed at the end we get the stable product
- Initial investment is less

Draw back

- It is not flexible model
- Testing is small phase which is done only after coding
- Design is not tested and the requirement is not tested if there is a defect either in the requirement or in the design it will flow till the end and it leads to lots of re-work
- Total investment will be more because of rework

Application

- Whenever developing the small s/w or application
- Whenever we go for short term projects
- Whenever we are sure that the requirements are not going to change

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

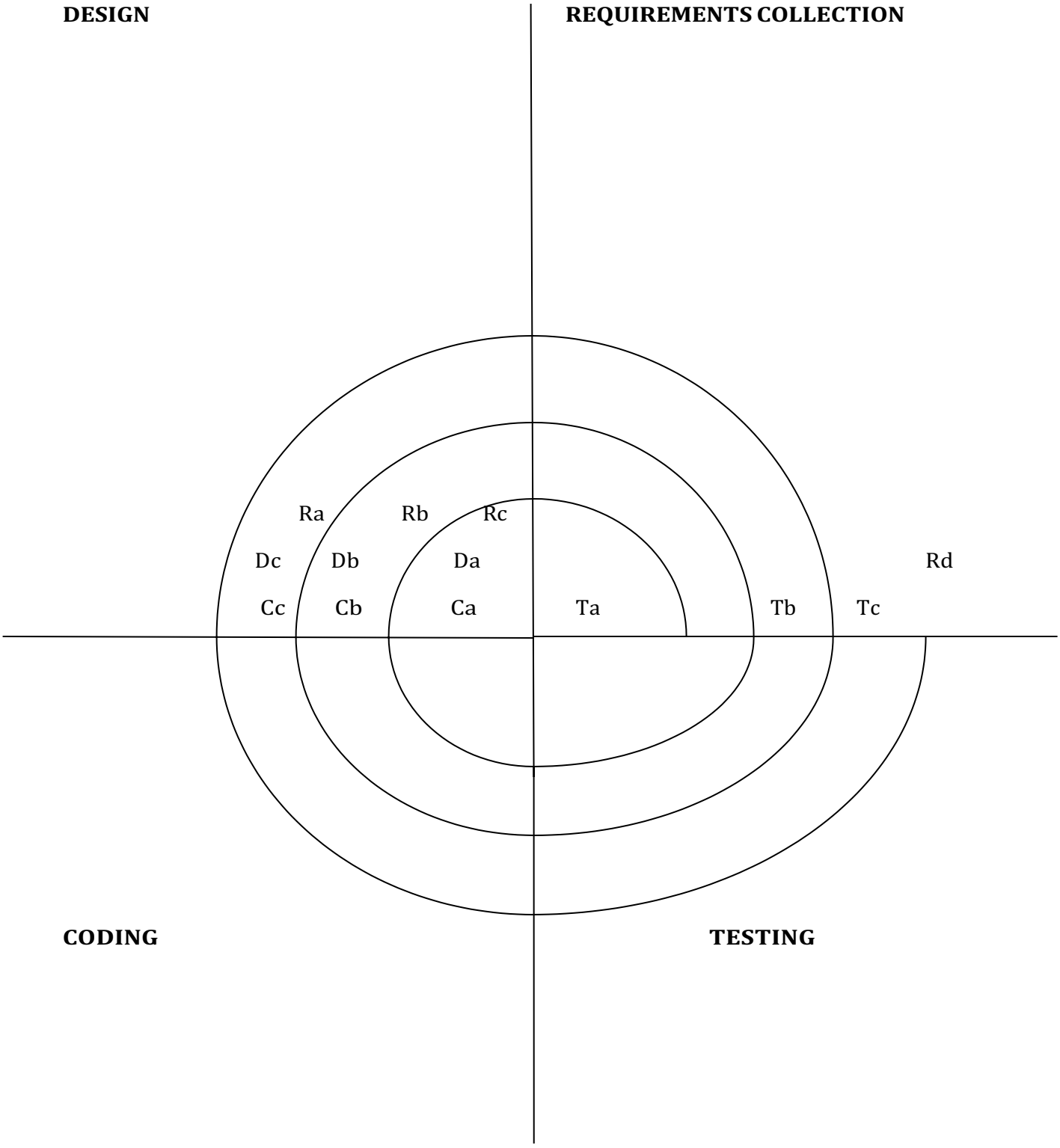
In Spiral model, the software product is developed in small modules. Let us consider the figure shown below in developing a s/w product X. X is built by integrating A,B,C and D.

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.



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

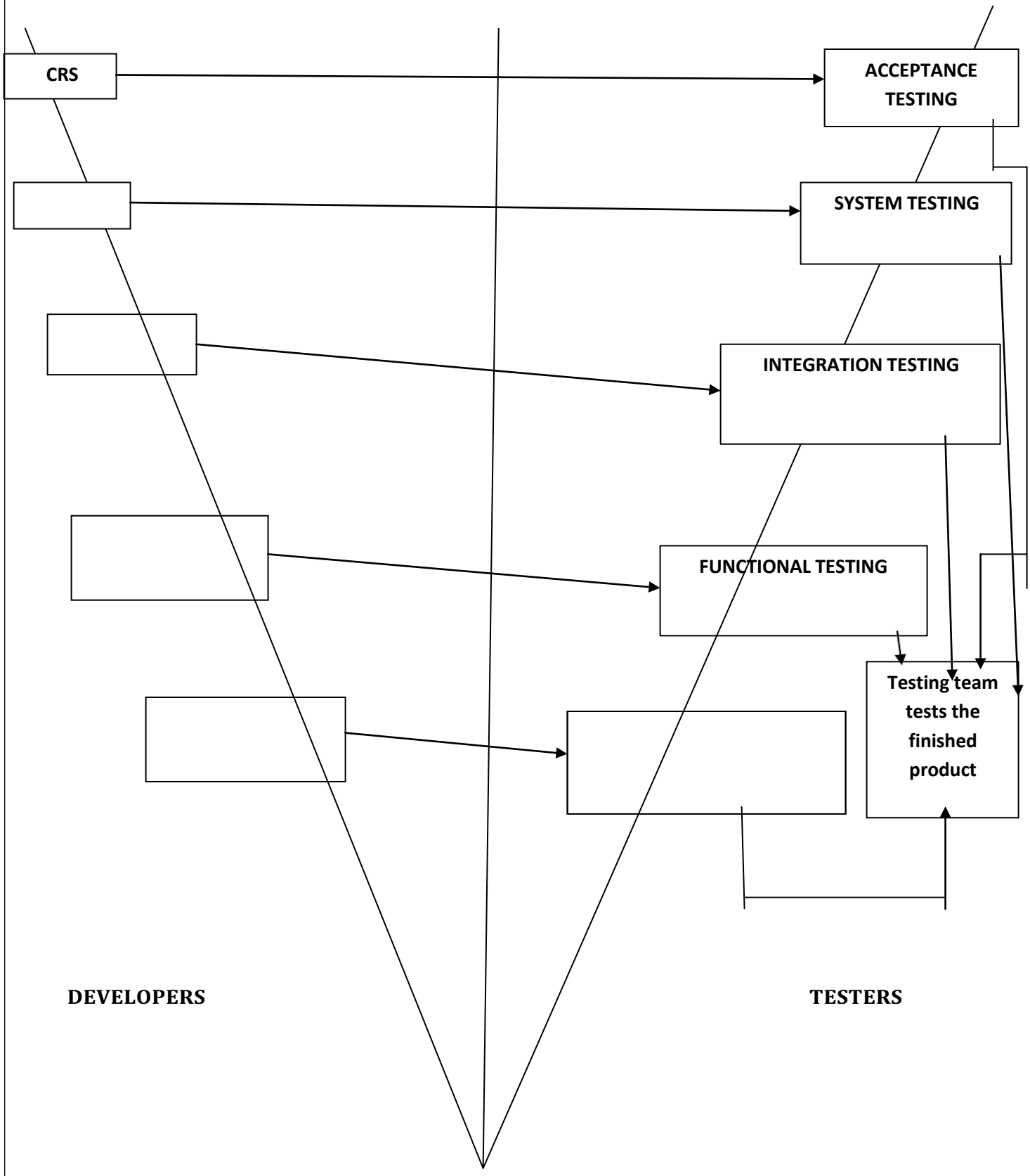
4) In the next stage, the developers start with the coding of the product. The testing team carries out the following tasks,

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.


CRS
ACCEPTANCE TESTING
SYSTEM TESTING
INTEGRATION TESTING
FUNCTIONAL TESTING
Testing team tests the finished product
DEVELOPERS
TESTERS

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

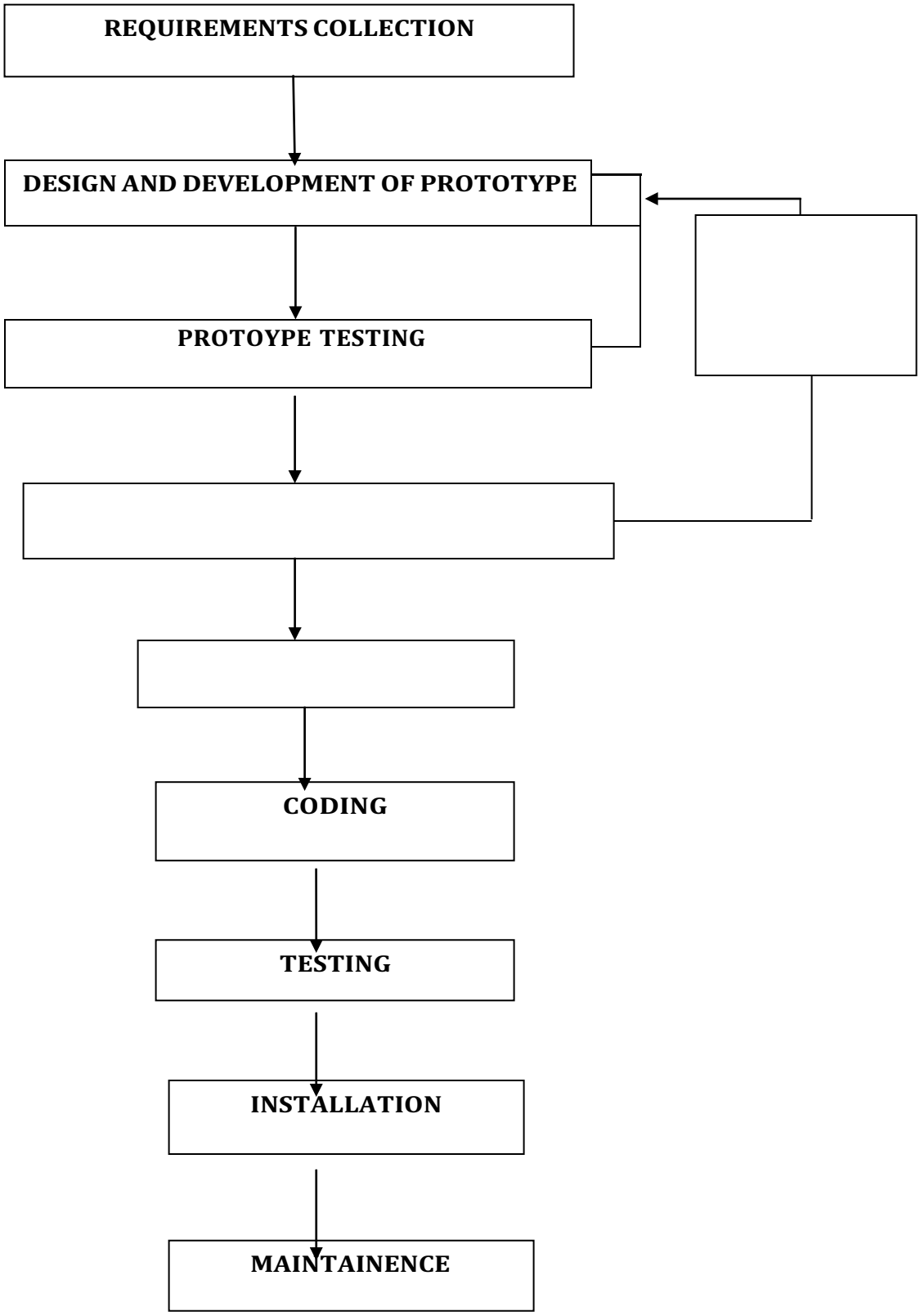
3) when customer is expecting a very high quality product within stipulated time frame because every stage is tested and developers & testing team are working in parallel

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.


REQUIREMENTS COLLECTION
DESIGN AND DEVELOPMENT OF PROTOTYPE
PROTOYPE TESTING
CODING
TESTING
INSTALLATION
MAINTAINENCE

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.

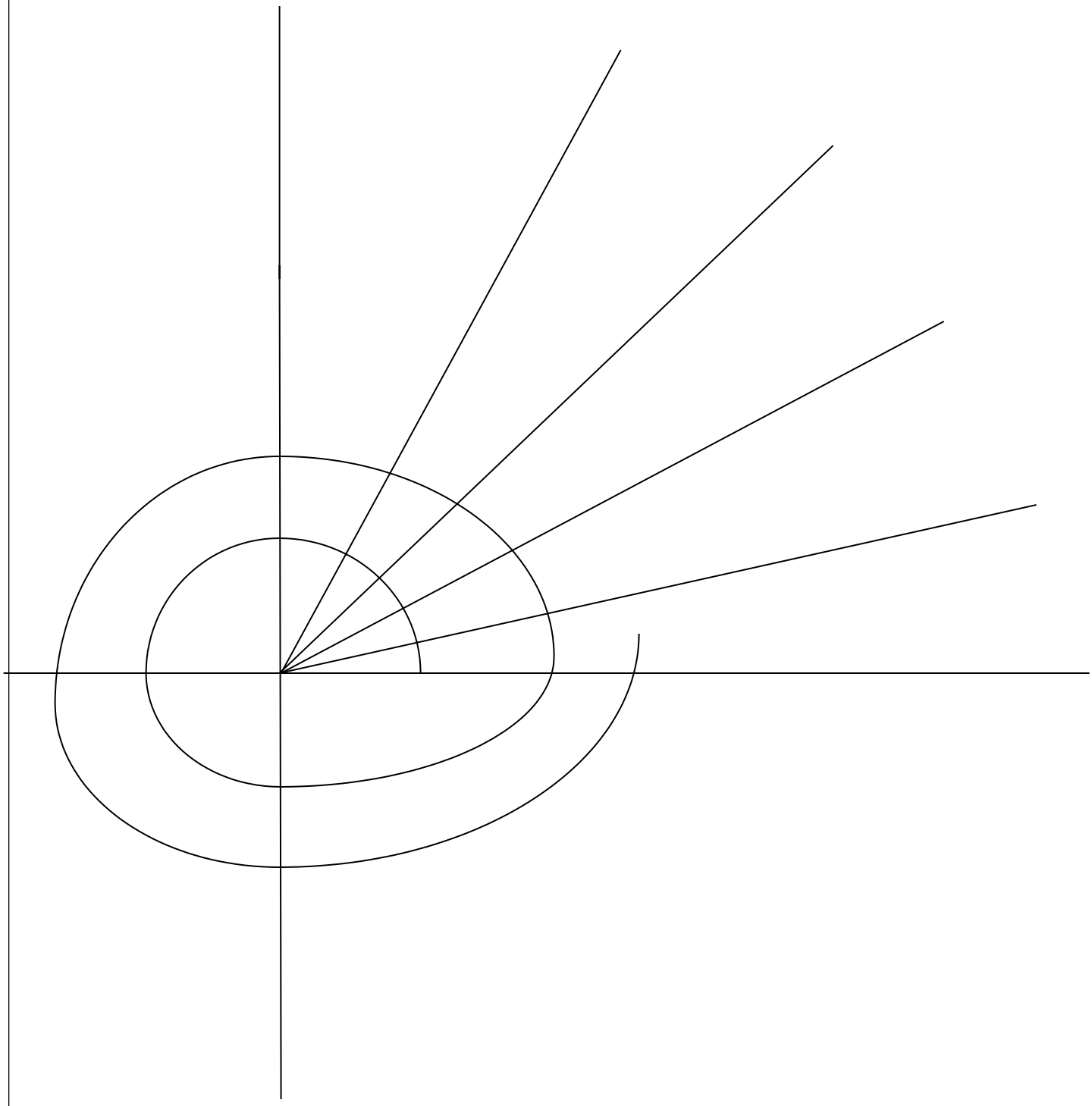
Dynamic Prototype – similar to a browser, but we can‟t enter any information. Only the features are available without entering data. It‟s like a dummy page, made out of HTML with tags and links to different pages representing features of the project

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, Advantages

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,

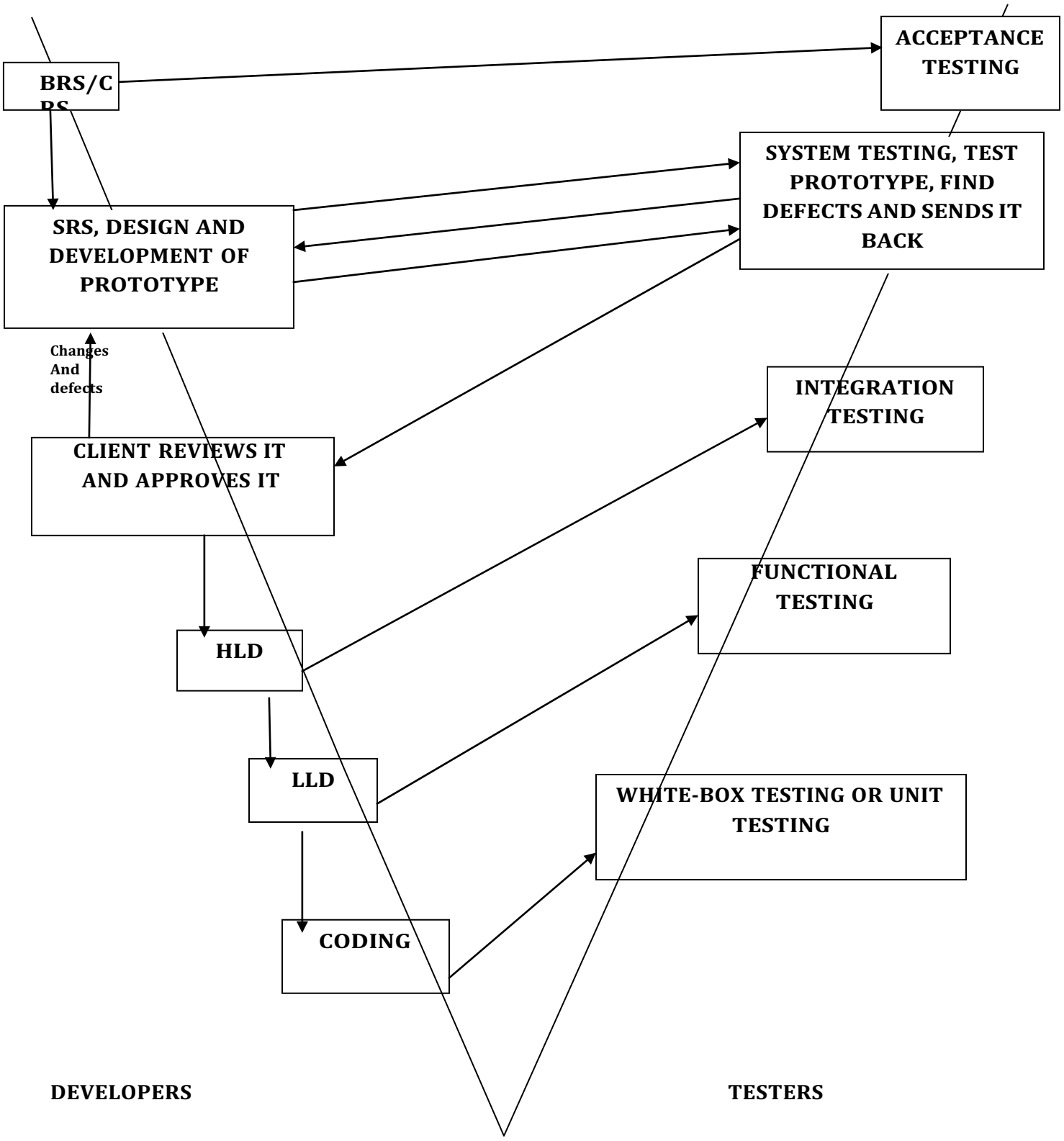
1) Testing starts from early stages of product development which avoids downward flow of defects, thus reducing re-work.

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.


BRS/C
RS
ACCEPTANCE TESTING
SYSTEM TESTING, TEST PROTOTYPE, FIND DEFECTS AND SENDS IT BACK
SRS, DESIGN AND DEVELOPMENT OF PROTOTYPE
Changes And defects
INTEGRATION TESTING
CLIENT REVIEWS IT AND APPROVES IT
FUNCTIONAL TESTING
HLD
LLD
WHITE-BOX TESTING OR UNIT TESTING
CODING
DEVELOPERS
TESTERS

INTERVIEW QUESTIONS

1) What is SDLC ?

2) What are the different models available ?

ANS) Tell the 1st 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 .

Interview questions

What is SDLC?
Tell what
Tell different stages and explain each stages
Tell about different types of model
Explain each model
What are the different models available
Water fall model
Spiral
V
Prototype
Hybrid
Azile
What are the advantages, disadvantages and applications of each model
Can you explain waterfall model in detail
Can you explain v model
Can you explain Azilemodel

Project specific

In the current project which model you are following, can you explain
Tell Azile model, and explain it
Tell me your roles and responsibilities
How do you rate yourself in Manual testing/java/SQL/Selenium?
out of 5

7.5 out of 10

HR question

- Please tell me about yourself / introduce yourself/ brief me about yourself
- Why should we hire you?
- Why S/W testing?
- Why you want to join our company?
- Where you want to find yourself down the line 5 years
- Strengths and weakness
- How much salary
- Ready to relocate
- Are you ready to work in shift

Definition of Software Testing

Software testing

Procedure of finding or identifying defects in the s/w is called s/w testing

It is verifying the functionality of the application against requirement specification.

It is the execution of the s/w with the intention of finding defects.

It is checking whether the s/w works according 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

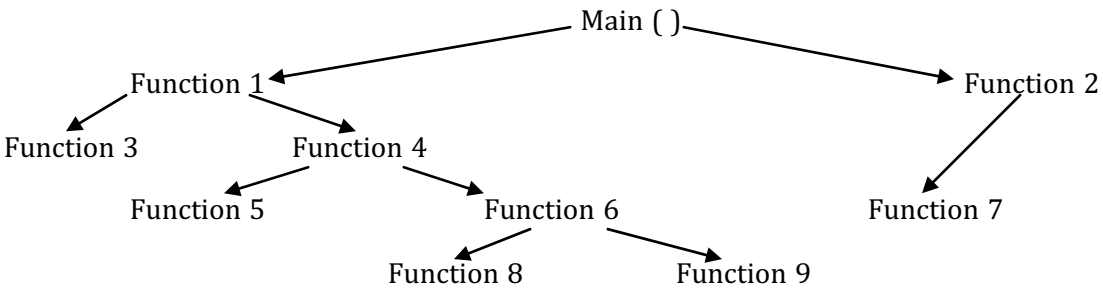
2) fixing a defect might break a lot of other features. Thus, testers should always identify defects and developers should always be involved in fixing defects.

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,

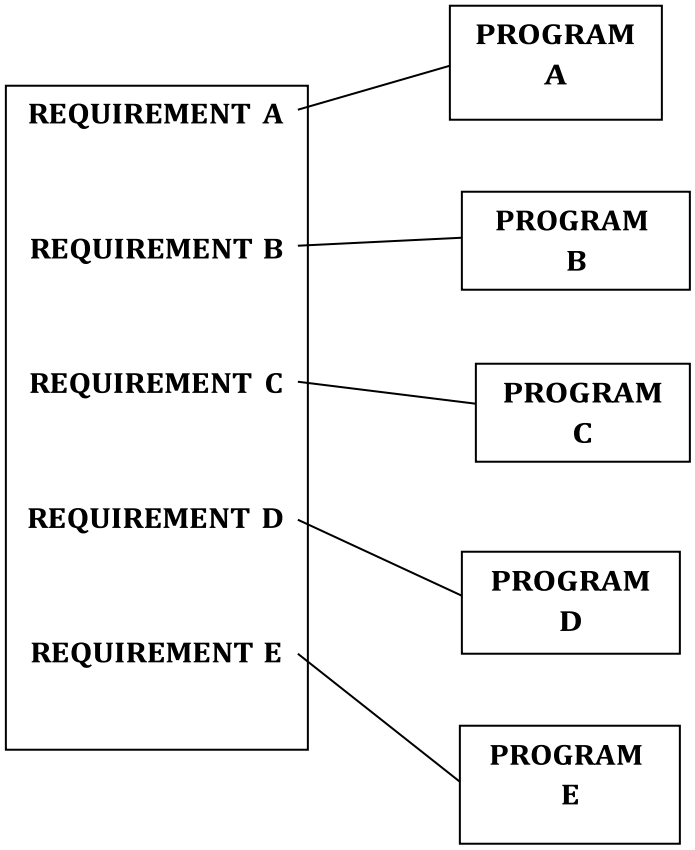
Test A

{

……

…… **}**

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.



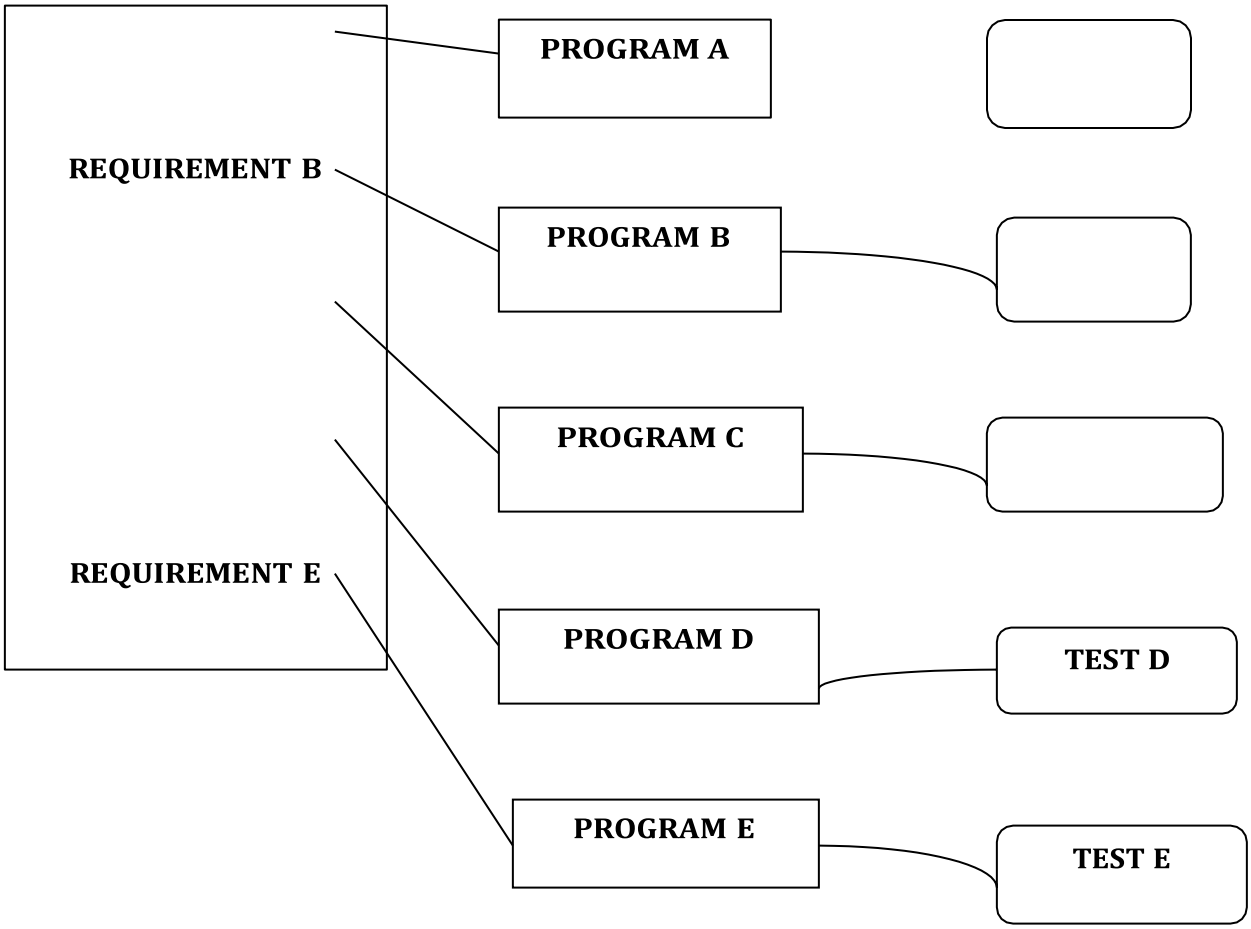
Let us consider the above case as shown in the figure. Suppose the project consists of Requirement A,B,C,D,E. Then the developer writes the Codes A,B,C,D,E for the corresponding requirements. The program consists of 100s of lines of code.

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.

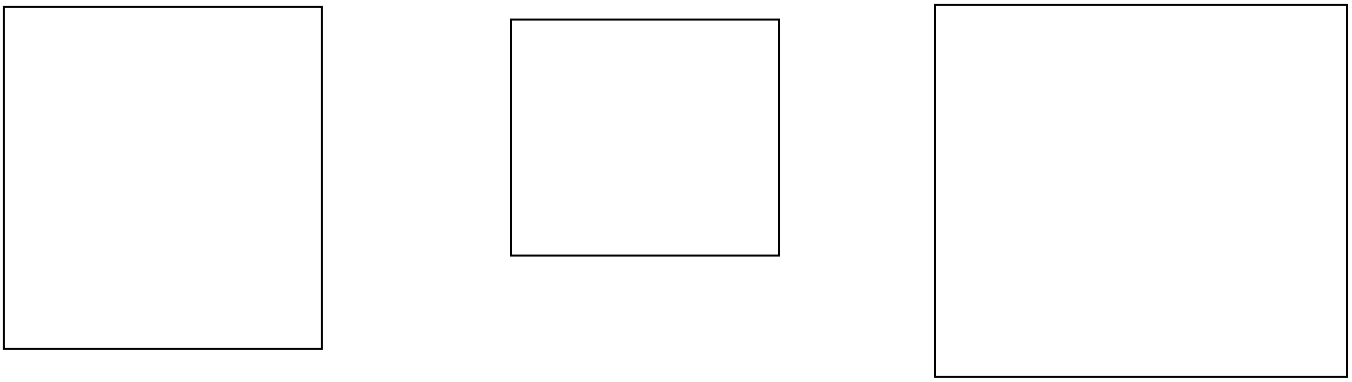


The collection of test programs is known as **test suite.**

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 add.java Test program



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 1st 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.

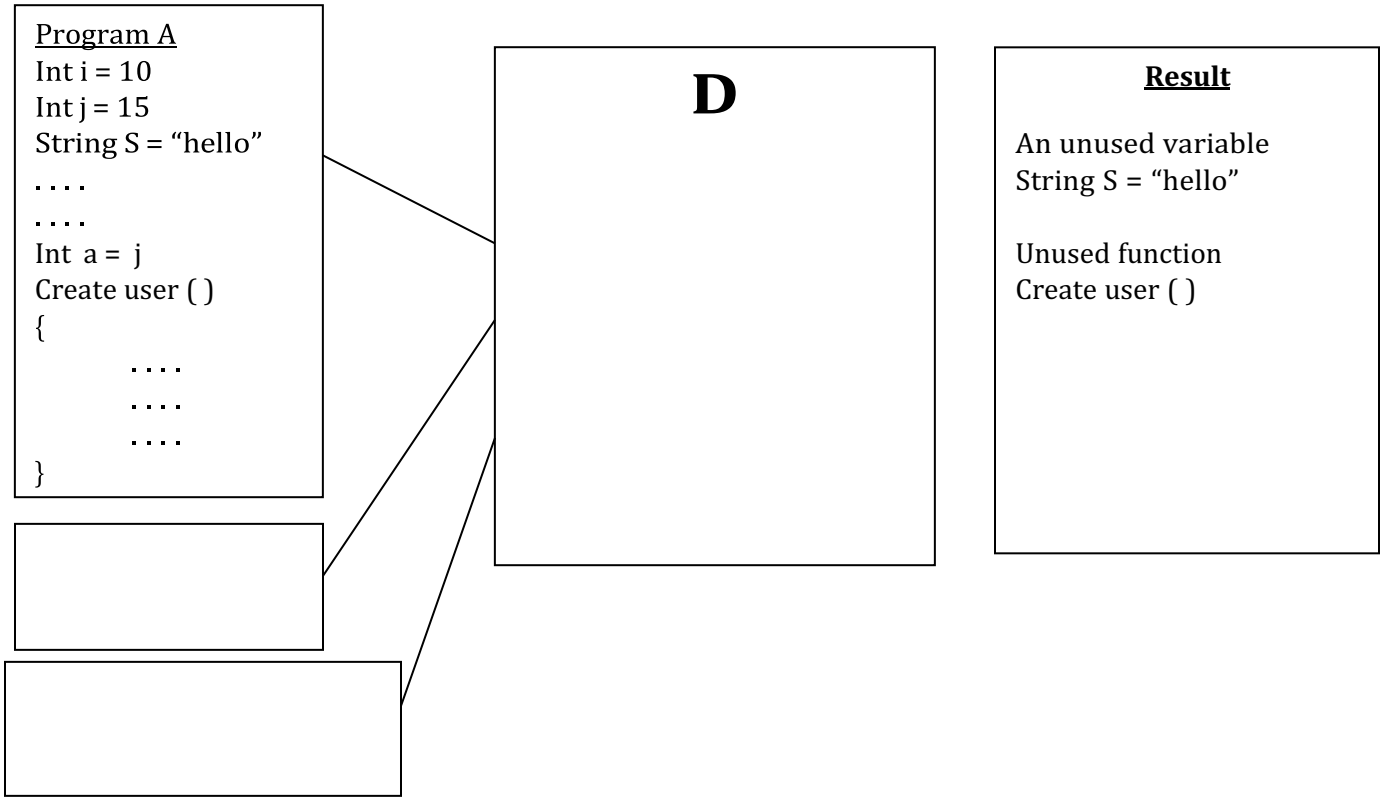
For ex,

```
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 ( ).**

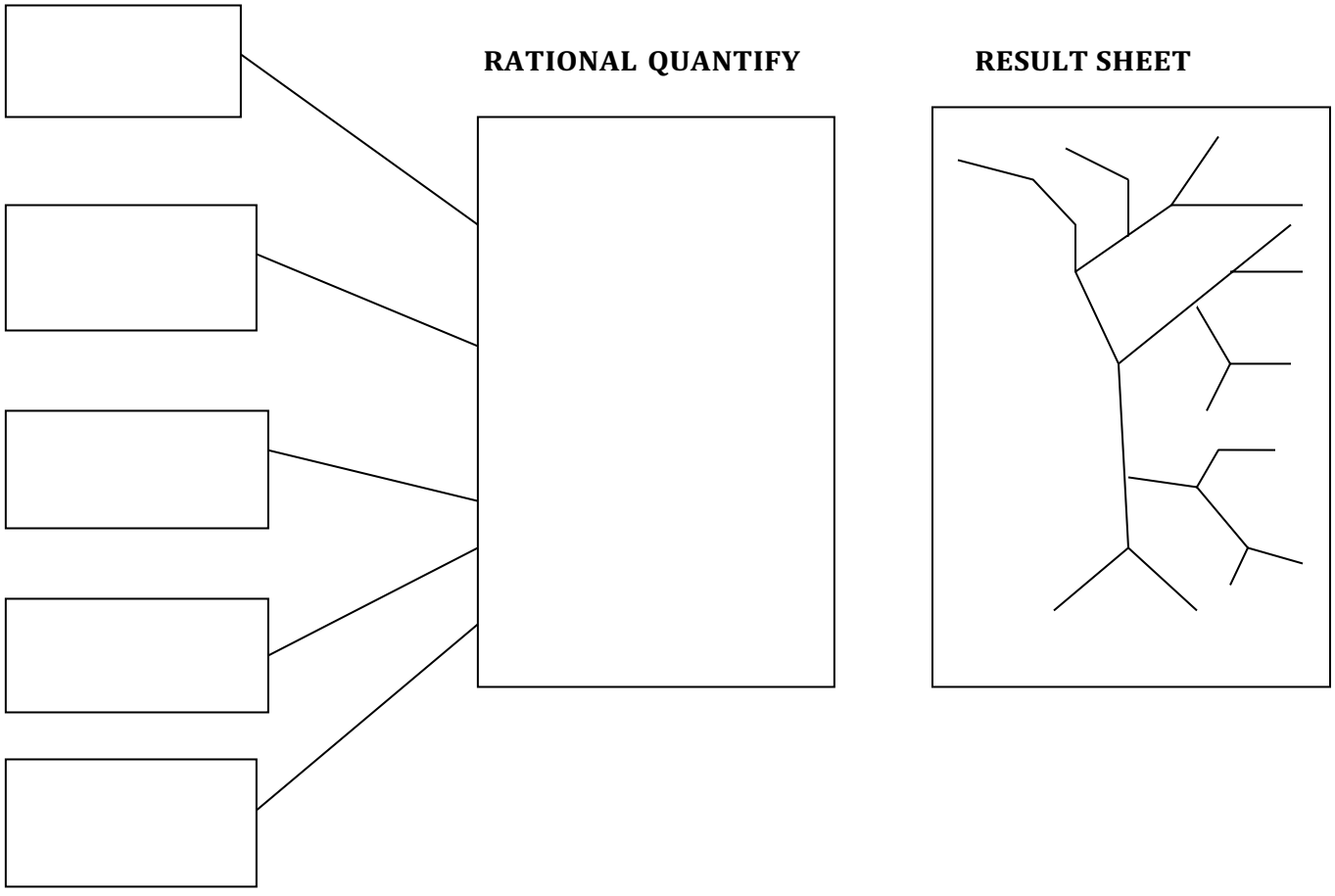
e) Test for response time/ speed/ performance of the program

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.

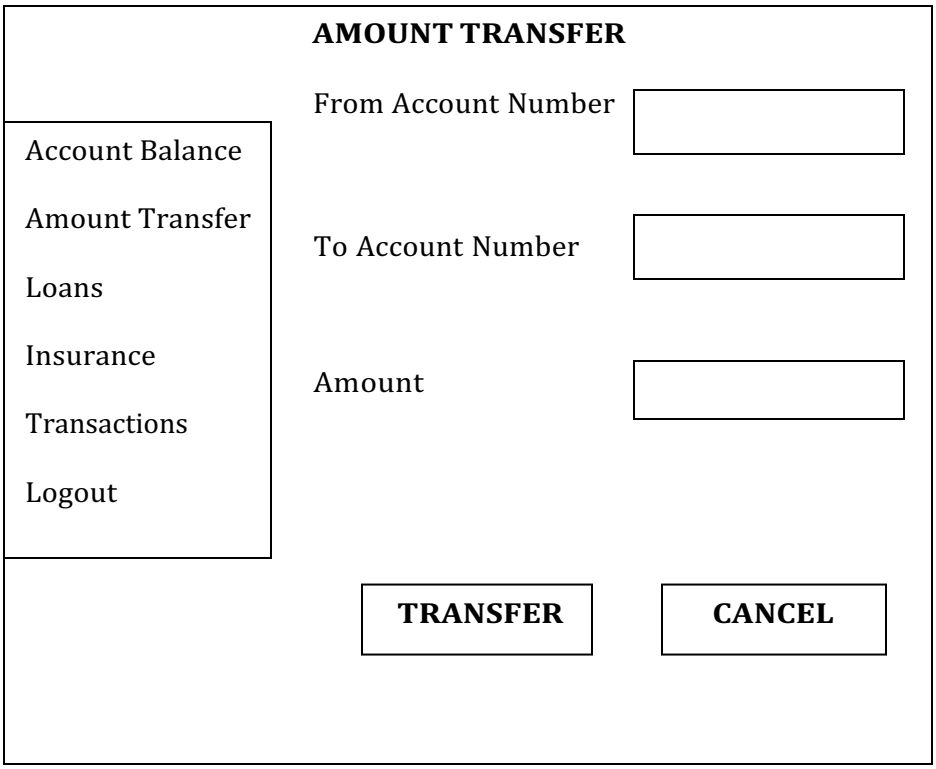
Types of Black Box Testing

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.



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

Username : should accept only 8 – 22 characters

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

Forgot Password : . . . . . .

1.3.1 . . . . . .

1.3.2 . . . . . .

1.4 Registration : . . . . . .

1.4.1 . . . . . .

1.4.2 . . . . . .

2. LOANS

Personal Loan : . . . . . .

2.1.1 . . . . . .

2.1.2 . . . . . .

2.2 Home Loan : . . . . . .

2.2.1 . . . . . .

2.2.2 . . . . . .

3. INSURANCE

3.1 . . . . . .

3.2 . . . . . .

. . . .

. . . .

.. . .

Now, we scroll to page 30, and see the 60th requirement.

We see the requirements specification for AMOUNT TRANSFER

60. AMOUNT TRANSFER

From account number text field

Should accept 10-digit integer

Should accept only those accounts which are created byManager

To account number text field

Should accept 10-digit integer

Should accept only those accounts created by manager

Amount Textfield

Should accept only positive integers

Should not accept more than balance

. . . .

. . . .

. . . . . .

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 6th 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 characterstics 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

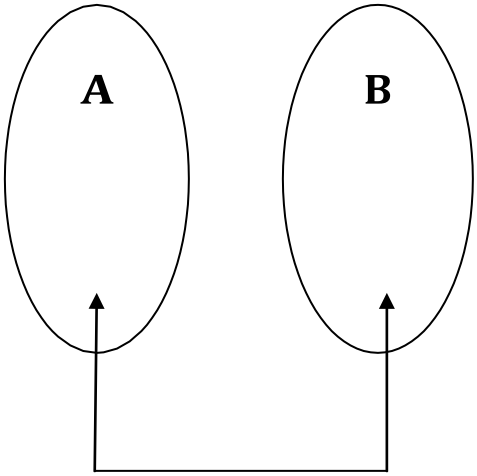
7) Unambiguous – the requirement is concisely stated without recourse to technical jargon, acronyms etc. it expresses objective facts, not subjective opinions. It is subjective to one and only one interpretation.

8) Mandatory – the requirement represents a stakeholder defined characterstic 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.

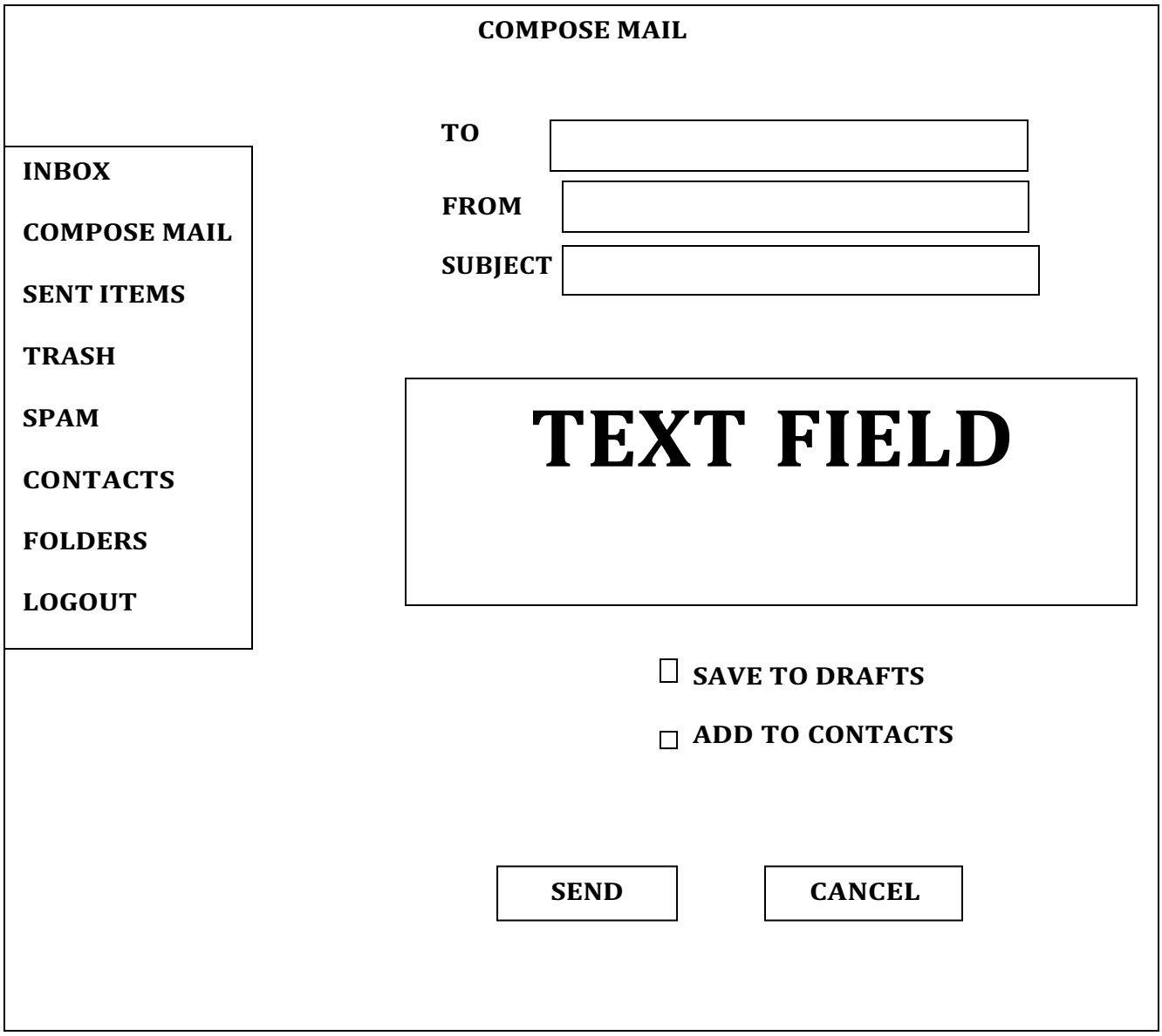
2) Identify all possible scenarios

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.

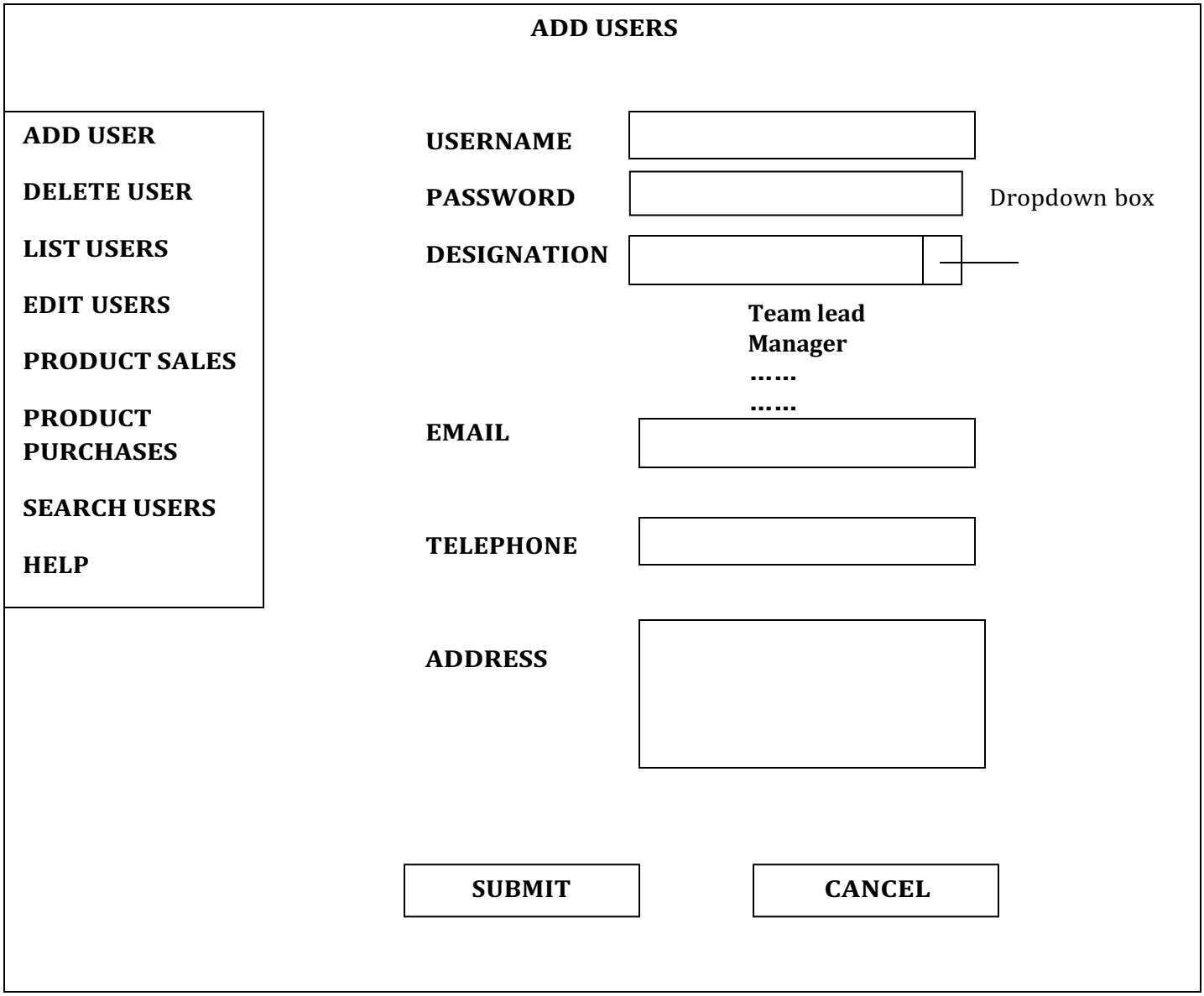


Let us consider gmail software as shown below. We first do **functional testing** for username and password and submit and cancel button. Then we do **integration** testing for the above. The following scenarios can be considered,

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 .**



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,

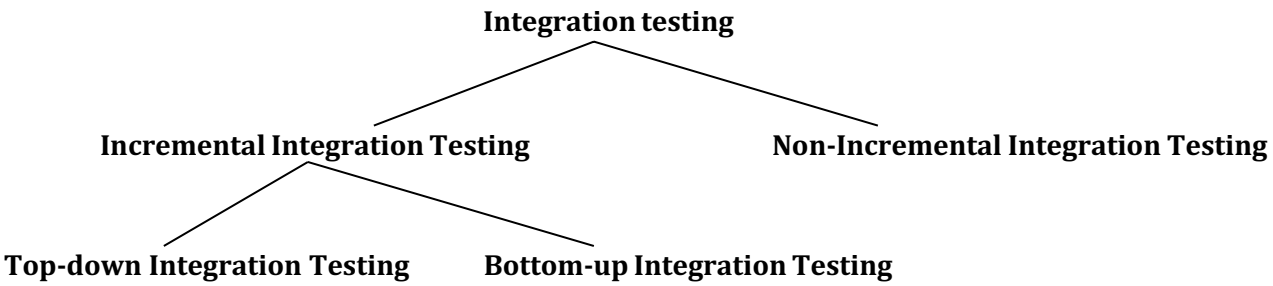
1) There are features we might be doing only functional testing and there are features we might be doing both integration and functional testing. It depends on features.

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.

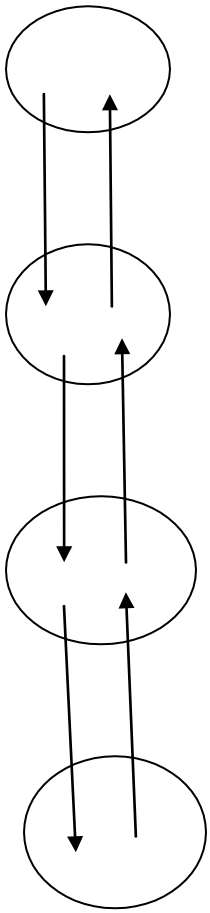
3) Focus is important i.e, completely test 1 feature and then only move onto another feature.

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,**



<u>Incremental Integration Testing :</u>



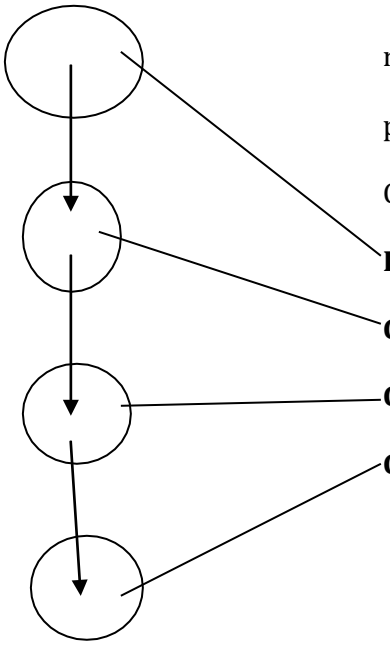
Take two modules. Check if data flow between the two is working fine. If it is, then add one more module and test again. Continue like this. Incrementally add the modules and test the data flow between the modules.

There are **two** ways,

a) Top-down Incremental Integration Testing

b) Bottom – up Incremental Integration Testing

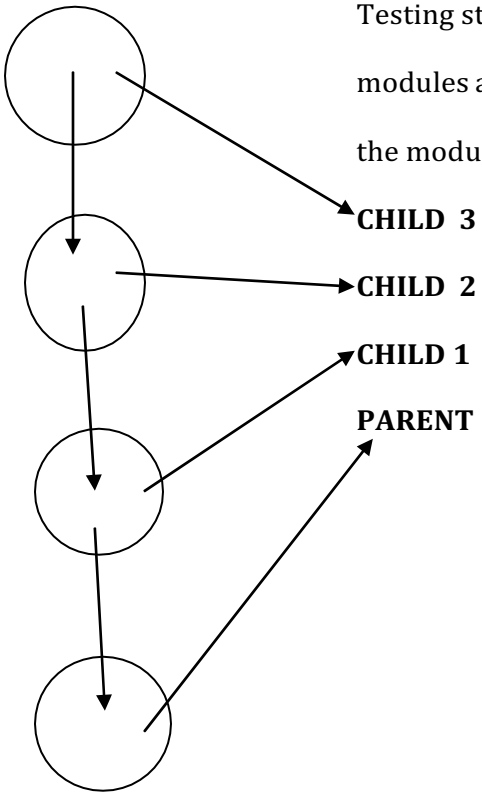
Top-down Integration Testing :



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.

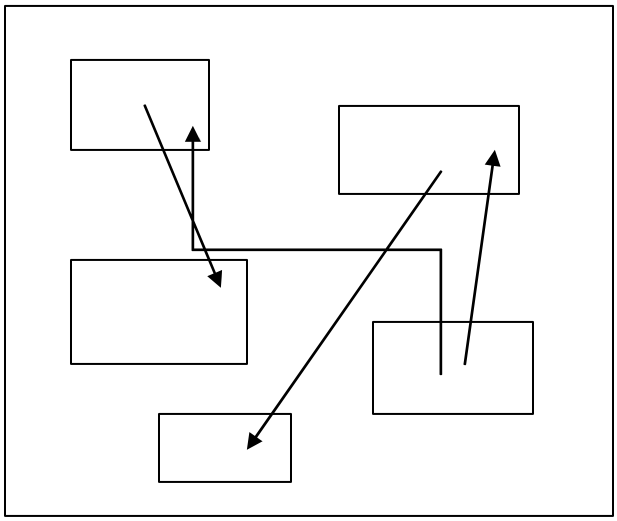
Non – incremental Integration Testing

We use this method when,

a) When data flow is very complex

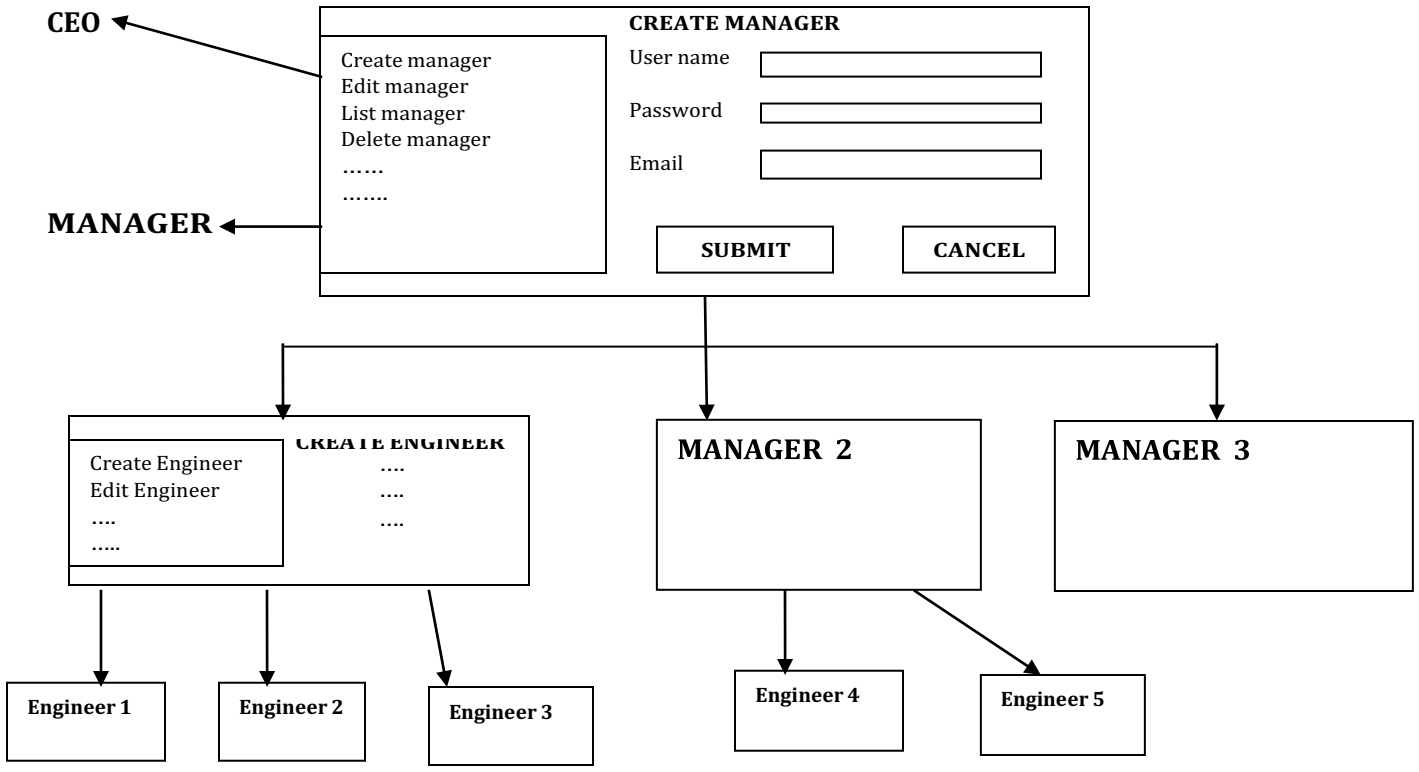
b) When it is difficult to identify who is parent and who is child.

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.

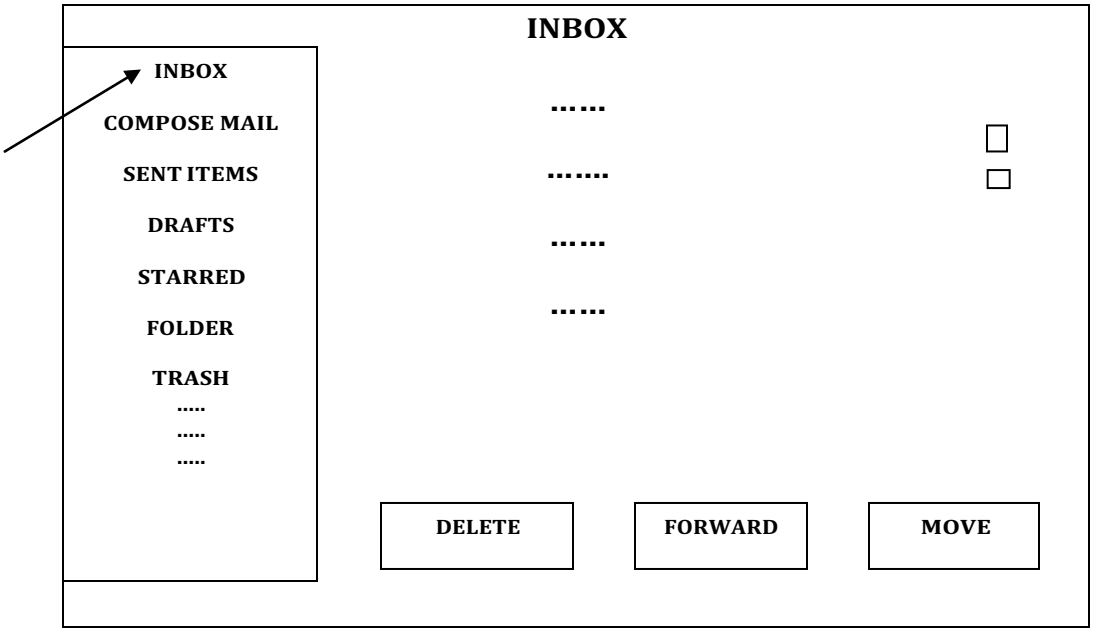
Example for Incremental Integration Testing :



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.**

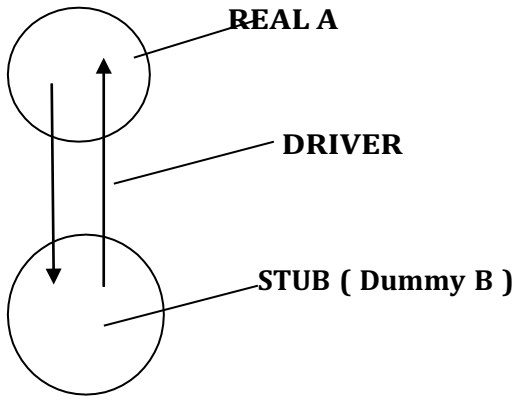
Similarly, the testing engineer once he finishes testing sends a report to the manager, who then sends a report to the CEO. This is known as **bottom-up incremental integration testing.**

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 4options –
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 2modules 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.