JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self-Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

IT8076 – SOFTWARE TESTING

(Regulation 2017)

Year/Semester: III/VI CSE

2020 – 2021

Prepared by

Ms. R. Revathi

Assistant Professor/CSE

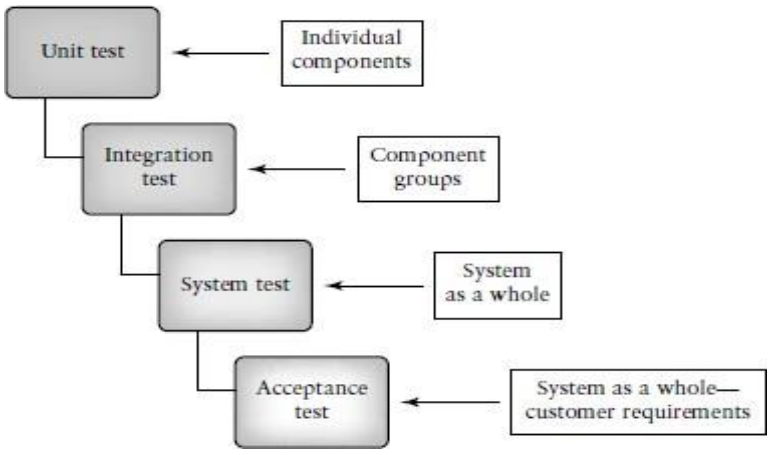
UNIT-III

The Need of Levels of testing, Unit test , Unit test planning, Designing the unit test. Test Harness, Running the unit tests and recording results. Integration Tests- Designing Integration test- Integration Test Planning- Scenario Testing –Defect Bash Elimination-System testing -Acceptance testing-Performance Testing-Regression Testing-Internationalization Testing- Adhoc Testing-Alpha-Beta Tests-Testing OO Systems-Usability and Accessibility Testing- configuration testing – compatibility testing –testing the documentation –website testing

Need For Levels Of Testing:-

- Execution based software testing, especially large systems, is usually carried out at different levels
- Major Phases of testing:
    - Unit Test
    - Integration Test
    - System Test
    - Acceptance Test

Principal goal is to detect functional and structural defects in the unit. At the integration level several components are tested as group, and tester investigates component interaction. At the system level the system as a whole is tested and a principle goal is to evaluate attribute such as ability, reliability and performance



Level of Testing and Software Development Paradigms

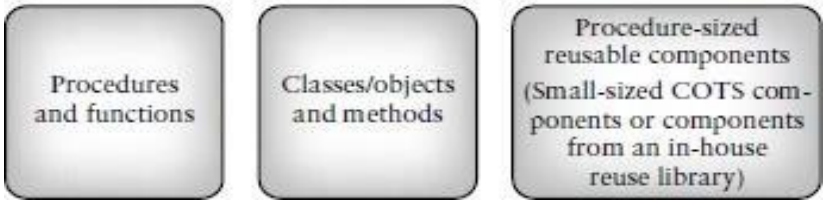
The approach used to design and develop a software system has an impact on how a testers plan and design suitable tests.

- The major approaches to system development- 1) Bottom up 2) Top down
- These approaches are supported by two major types of programming languages-
  1) procedure Oriented and 2) Object Oriented
- The different levels of systems developed with both approached using their traditional procedural programming languages or object oriented programming languages.

- Systems developed with procedural languages
    - are generally viewed as being composed of passive data and active procedures
    - When test cases are developed the focus is on generating input data to pass to the procedures (or functions) in order to reveal defects.
- Object Oriented systems
    - are viewed as being composed of active data along with allowed operations on that data, all encapsulated within a unit similar to abstract data type.

Unit test: Functions, Procedures, Classes, and Method as Unit

- **A workable definition for a software unit is as follows**
    - A Unit is the smallest possible testable software component
        - It can be characterized in several ways. For example a unit in a typical procedure oriented software system"
            - Perform a single cohesive function
            - Can be compiled separately
            - Is a task in a work breakdown structure (from the manager's point of view)
            - Contain code that can fit on a single page or screen.
- A unit is traditionally viewed as a function or procedure implemented in a procedural (imperative) programming language.
- In object oriented systems both the method and the class/object have been suggested by researchers
- A unit may also be a small sized COTS component purchased from an outside vendor that is undergoing evaluation by the purchaser, or simple module retrieved from an in-house reuse library



Unit Test- The Need for Preparation

- The principal goal □for unit testing is insure that each individual software is functioning according to its specification
- Good testing practice □ calls for unit tests that are planned and public.
- Planning includes
    - designing test to reveal defects such as functional description defects, algorithmic defects , data defects, and control logic and sequence defects.
    - Resources should be allocated and test cases should be developed, using both white and black box test design strategies.
- The unit should be tested by an independent tester (other than testers) and the test results and defects found

- Each unit should also be reviewed by a team of reviewers, preferably before the unit test.
- Unit test in many cases is performed informally by the unit developer soon after the module is completed, and it compiles cleanly.
- Some developers also perform an informal review of the unit .
- To prepare for unit test the developers/ testers must perform several tasks. These are:-

1. Plan the general approach to unit testing
2. Design the test cases, and test procedures
3. Define relationships between the test
4. Prepare the auxiliary code necessary for unit test.

Unit test Planning

A general unit test plan should be prepared.It may be prepared as a component of the master test plan or a stand alone plan.It should be developed in conjunction with the master plan and the project plan for each project

- **Phase 1 : Describe Unit test Approach and Risk**

In this phase of unit test planning the general approach to unit test planning is outlined: The test planner

1. Identifies test risks
2. Describes techniques to be used for designing the test cases for units
3. Describes techniques to be used for data validation and recording of test results
4. Describes the requirement for test harness and other software that interfaces with unit to be tested eg:- any special software needed for testing object oriented units

- During this phase the planner also identifies completeness requirements ie what will be covered by the unit test and to what degree (state , functionality, control, data flow patterns)
- Planner also identifies termination condition for unit test.
- They include coverage requirement and special cases
- Special cases may result in abnormal termination of unit test
- Planner estimate the resources needed for unit test such as hardware, software and staff and develop tentative schedule under constraints identified at that time

Phase 2:- Identify Unit Features to be Tested

- This phase requires information from the unit specification and detailed design description
- The planner determines which features of each unit will be tested, for example functions, performance requirement , state and state transition , control structures , messages and data flow patterns
- Some features will be covered by the tests, they should be mentioned and risks of not testing them be assessed.
- Input and output of each test unit should be identified.

Phase 3: Add levels of Detail to the Plan

- In this phase the planner refines the plan as produced in the previous two phases
- The planner adds new details to the approach, resource and scheduling portions of the unit test plan

- Eg:- Existing test cases that can be reused for this project can be identified in the
- Unit availability and integration scheduling information should be included in the revised version of the test plan
- Planner must be sure to include a description of how test results will be recorded.
- Test related documents that will be required for this task eg test logs, test incidents report should be described.

Designing the Unit test

- It is important to specify the following test design information with the unit test plan
  - The test cases (including I/O and expected output for each test cases)
  - The test procedures (steps required run the test)
  - As a part of the unit test design process, developers / tester should also describe the relationship between the tests.
  - Test suites can be defined that binds related tests together as a group.
- Test cases, test procedures and test suites may be reused from the past projects if the organization has been careful to store them so that they can be easily retrievable and reusable
- Test case design at unit level can be base on the use of black and white box design strategies
- Both of these approaches are useful for designing test cases for functions and procedures
- They are useful to designing test for individual methods in a class. This approach gives the tester the opportunity to exercise logic structure and /or data flow sequence or to use mutation analysis, all with the goal of evaluating the structural integrity of the unit

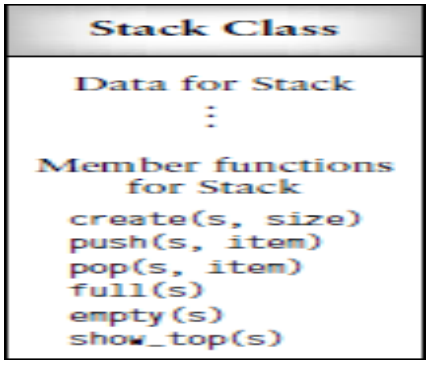
Class as a testable Unit

- If an organization is using the object oriented paradigm to develop software system it will need to select the component to be considered for unit test.
- Choice consist of 1) individual methods as a unit or 2) the class as a whole.
- Additional code in the form of tests harness, must be built to represent the called methods within the class. This is costly;
- Building such test harness for each individual method often require developing code equivalent to that already existing in the class itself.
- In spite of the potential advantages of testing each method individually, many developers/testers consider the class to be the component of choice for unit testing. The process of testing classes as units is sometimes called component test
- When testing on the class level we are able detect not only traditional types of defects, for example, those due to control or data flow errors, but also defects due to the nature of object oriented systems, for example, defects due to encapsulation, inheritance, and polymorphism errors.

Issue 1:- Adequately Testing Classes

- The potentially high costs for testing each individual method in a class

- These high cost will be particularly apparent when there are many methods in a high as 20 to 30.
- If the class is selected as the unit to test, it is possible to reduce these cost since many cases the methods in a single class server as drivers and stubs for one another.
- This has the effect of lowering the complexity of the test harness that needs to be developed.
- some cases driver classes that represent outside classes using the methods of the class under test will have to be developed.
- For example : create, pop, push empty, full and show_top methods associated with the stack class.
- When testers unit(or components) test this class what they will need to focus on is the operation of each of the methods in the class and the interaction between them
- For example, a test sequence for a stack that can hold three items might be:
  create(s,3), empty(s), push(s,item-1), push(s,item-2), push(s,item-3),
  full(s), show_top(s), pop(s,item), pop(s,item), pop(s,item), empty(s), . . .



Issue 2: Observation of Object States and State Changes

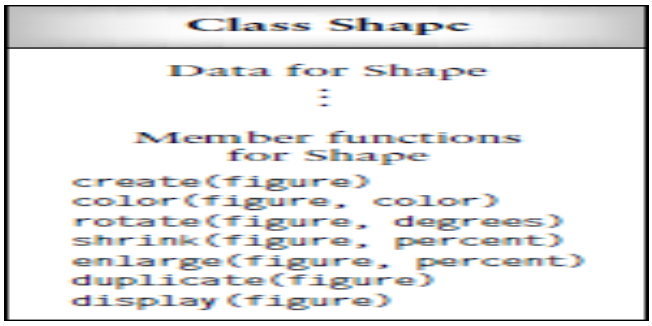
- Methods may not return a specific value to a caller
- They may instead change the state of an object .
- The state of an object is represented by a specific set of values for its attributes or state variables
- Methods often modify the state of an object, and the tester must ensure that each state transistor is proper
- The test designers can prepare a state table that specifies states , the object can assume, and then in the table indicate sequence of messages and parameters that will cause the object to ensure each state.
- When the test are run the tester can enter results in this table. The first call to the method *push* in the stack class, changes the state of the stack so that empty is no longer true. It also changes the value of the stack pointer variable, top.
- To determine if the method *push* is working properly the value of the variable *top* must be visible both before and after the invocation of this method. In this case *show_top* within the class may be called to perform this task.
- The methods full and empty also probe the state of the stack. A sample augmented sequence of calls to check the value of top and the *full/ empty* state of the three item stack is :

Issue 3:- The Retesting of classes-I

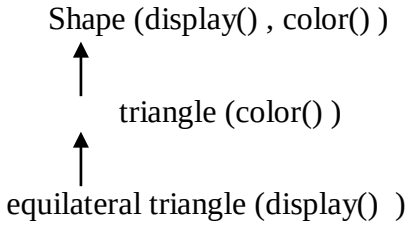
- One of the most beneficial features of object oriented development is encapsulation used to hide information
- A program unit , in this case a class, can be built with a well defined public interface that proclaims its services to client classes. The implementation of the services is private. Client who use the service s are unaware of implementation details. The interface is unchanged , making changes to the implementation should not affect the client classes. A tester object oriented code would therefore conclude that only the class with implementation changes to its methods needed to be retested.
- In an object-oriented system, if a developer changes a class implementation that class needs to be retested as well as all the classes that depend on it. If a superclass, for example, is changed, then it is necessary to retest all of its subclasses

Issue 4:- The Retesting of classes-II

- Classes are usually a part of a class hierarchy where there are existing inheritance relationships
- Subclasses inherit methods from their super classes
- Tester may assume that once a method in a super class has been tested , it does not need retested in a subclasses that inherit it.



- There may be overriding of methods where a subclass may replace an inherited methods with a locally define methods.
- Designing a new set of test cases may be necessary.
- This is because the two methods may be structurally different



- Suppose the shape superclass has a subclass, triangle, and triangle has a subclass, equilateral triangle. Also suppose that the method display in shape needs to call the method color for its operation.

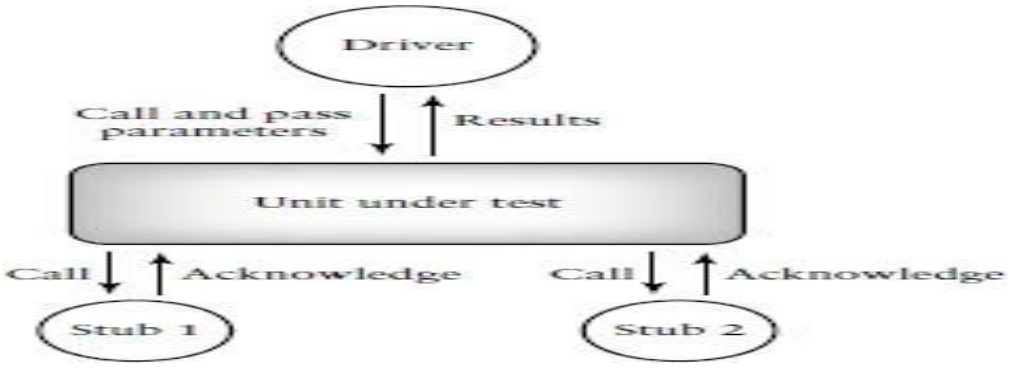
- Equilateral triangle could have a local definition for the method display. That definition for color which has been defined in triangle.
- This local definition of the color method in triangle has been tested to work with the inherited display method in shape, but not with the locally defined display in equilateral triangle.
- This is a new context that must be retested. A set of new test cases should be developed.
- The tester must carefully examine all the relationships between members of a class to detect such occurrences.

The Test Harness

- The auxiliary code developed to support testing of units and components is called as test harness
- The harness consist
  - **drivers call the target code**
  - **stubs represent modules it calls.**
- The development of drivers and stubs requires testing resources.
- The drivers and stubs must be tested themselves to insure they are working properly and that they are reusable for subsequent releases of the software
- Drivers and stubs can be developed at several levels of functionality
- Eg:- a driver could have the following options and combinations of options:
  - Call the target unit
  - Do 1, and pad pass input parameters from the table
  - Do 1,2, and display parameters
  - Do 1,2,3 and display result (output parameters)

The stub should also exhibit bit different levels of functionality

- For example a stub could
  - Display a message that it has been called the target unit
  - Do1, and display any input parameters passes from the target units
  - Do1,2, and pass back result from a table
  - Do1,2,3 and display result from table



Running a Unit tests and Recording Results

- Unit test can begin when
  - The unit become available from the developers
  - The test cases have been designed and reviewed
  - The test harness and any other supplemental to supporting tools

  - The status of the test efforts for a unit, and a summary of test results must be recorded in a unit test worksheet

Unit Test Worksheet

Unit Name: _____

Unit Identifier: _____

Tester: _____

Date: _____

Test case ID	Status (run/not run)	Summary of results	Pass/fail

- It is very important that the tester at any level of testing to carefully record, review and check test results.
- The tester must determine from the results whether the unit has passed or failed the test
- If the test is failed, the nature of the problem should be recorded in what is sometimes called the test incident report.
- Differences from expected behavior should be described. When a unit fails a test there may be several reasons for the failure.
  - fault in the unit implementation
  - A fault in the test case specification (the input or the output was not specified correctly)
  - A fault in test procedures execution(the test should be rerun)
  - A fault in the test environment (perhaps a database was not set up properly)
  - A fault in the unit design (the code correctly adheres to the design specification , but the latter is incorrect)
- When a unit has been completely tested and finally passes all of the required tests it is ready for integration

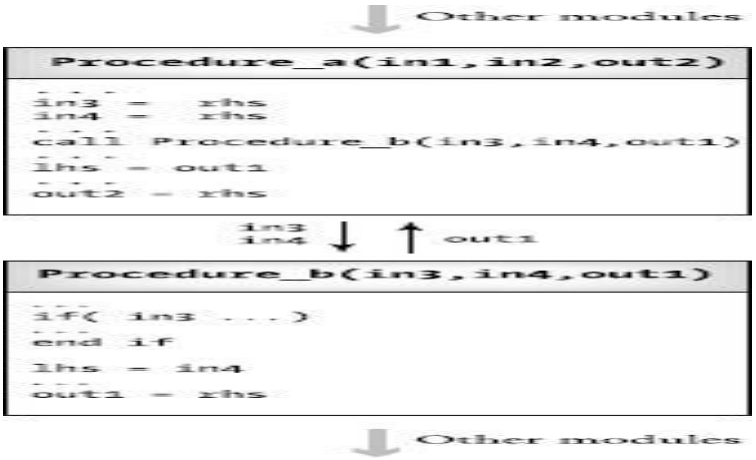
Integration Test-Goals

- Integration test for procedural code has two major goals
  - To detect defects that occur on the interfaces of units
  - To assemble the individual unit into working subsystem and finally a complete system that is ready for system test.
- In unit test the tester attempts to detect defects that are related to the functionality and structure of the unit.
- Some simple unit interfaces are more adequately tested during integration test when each unit is finally

- Few minor expectations, integration test should only be performed on unit successfully passed unit testing.
- A tester might believe erroneously that since a unit has already been tested during a unit test with a drivers and stubs, it does not need to be retested in combinations with other units during integration test.
- Integration testing works best as an iterative process procedural oriented system.
- One unit at a time integrated into a set of previously integrated modules which have passed a set of integration tests.
- The interface and functionality of the new unit is combination with the previously integrated units is tested
- When a subsystem is built from units integrated in the stepwise manner, then performance, security and stress test can be performed in this subsystem.
- Integrating one unit at a time helps tester in several ways.
- It keeps the number of new interfaces to be examined small, so that can focus on these interfaces only.
- Experienced tester know that many defects occur at module interface.
- Another advantage is that the massive failures that often occur multiple units are integrated at once is avoided.
- Approach also helps the developers, it allows defect search and repair confined to a small known number of components and interfaces
- Integration process is object oriented systems is driven by assembly of the classes into cooperating groups.
- The cooperating groups of classes are tested as a whole and then combined into higher level groups.

Designing Integration tests

- Integration test using a black or white box approach , Some unit test can be reused
- Since many error occur at module interfaces, test designers need to focus on exercising all input/output parameter pairs and all calling relationships
- The tester needs to insure the parameters are of the correct type and in the correct order.
- The author has had the personal experience of spending many hours trying to locate a fault that was due to an correct ordering of parameters in the calling routine



- Example : Procedure_b is being integrated with Procedure_a. Procedure_a parameters in3, in4. Procedure_b uses those parameters and then returns a value for the output parameter out1. Terms such as lhs and rhs could be any variable or expression.
- The parameter could be involved in a number of *def* and/or *use* data flow patterns
- The actual usage patterns of the parameters must be checked at integration time.
- Some black box test used for module integration may be reusable from unit testing.
- When units are integrated and subsystems are to be tested as a whole, new tests will a have to be designed to cover the functionality tests at the integration level are the requirements document and the user manual.
- Tester need to work with requirement analyst to insure that the requirement are testable, accurate and complete.
- Black Box tests should be developed to insure proper functionally and ability to handle subsystem stress.
- Integration Testing of clusters of classes also involves building test harness which in this case are special classes of objects built for testing
- Class testing we evaluated intra class method interaction , at the cluster level we test inter class method interaction as well
- We want to insure that message are being passed properly to interfacing objects, object state transition are correct when specific events occur , and that the cluster are performing their required functions.
- A group of cooperative classes is selected for a test as a cluster.(packages in java)
- If developers have used the Coad and Yourdon's approach , then a subject layer could be used to represent a cluster.

Integration test Planning

- Integration test must be planned
- Planning can begin when high level design is complete so that the system architecture is defined.
- Documents relevant to integration test planning are the requirement document, the use manual and usage scenario. These document contains structure charts, the state charts and data dictionary , cross reference table , module interface description
- The strategy for integration of the unit must be defined .
- For procedural-oriented system
  - the order of integration of the units of the units should be defined. This depends on the strategy selected. Consider the fact that the testing objectives are to assemble components into subsystems and to demonstrate that the subsystem functions properly with the integration test cases.
- For object-oriented
  - systems a working definition of a cluster or similar construct must be described, and relevant test cases must be specified. In addition, testing resources and schedules for integration should be included in the test plan. The plan includes the following items:
    - Cluster this cluster is dependent on

- A natural language description of the functionality of the cluster to be tested.
- List a classes in the cluster
- A set of cluster test cases

Integration Testing Types:-

Integration testing can be viewed as

1) type of testing
2) phase of testing.

Integration is defined to be a set of interactions, all defined interaction among the components need to be tested. The architecture and design can give the details of interactions within the systems, however testing the interactions between one system and another system required detailed understanding of how they work together.

Integration Testing As a Type of Testing :-

Integration testing means testing of interfaces. They are

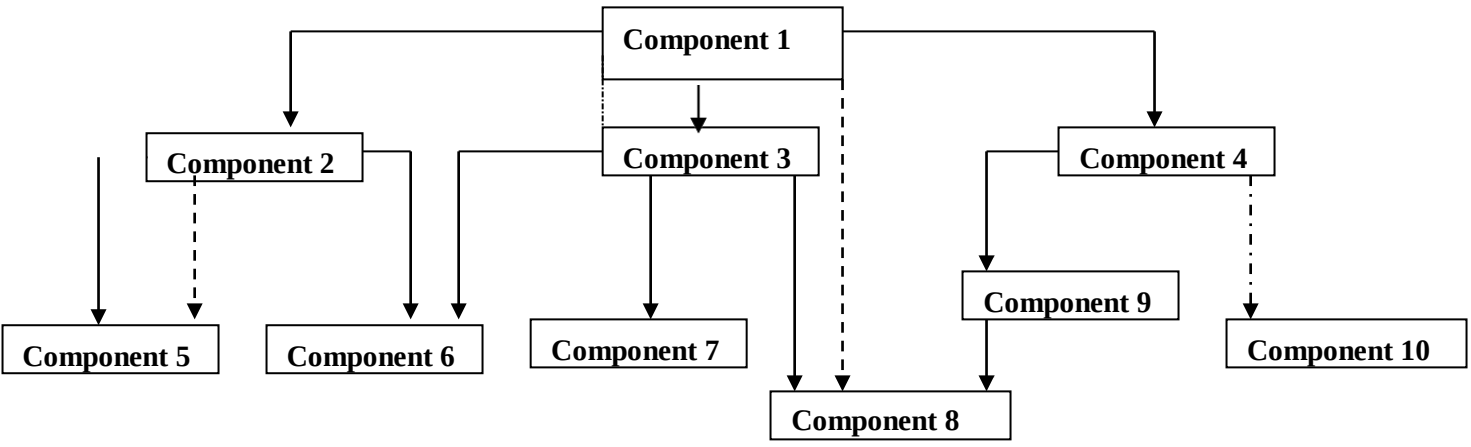
Internal Interfaces - provide communication across two modules within a projects or product, internal to the product, and not exposed to the customer or external developers

Exported or ***External Interfaces.***. - Exported interfaces are those that are visible outside the product to third party developers and solution providers.

"Intergration Testing Type Focuses on testing interfaces that are "Implicit and Explicit" and "Internal and External"

Implicit interface -> Documentation given

Explicit interface-> No documentation given



"A set of Modules and Interfaces"

In the above diagram, it is clear that there are at least 12 interfaces between the modules to be tested (9 explicit and 3 explicit). Now what will be the order of interface to be tested. There are several methodologies available , to in decide the order for integration testing. These are as follows:-

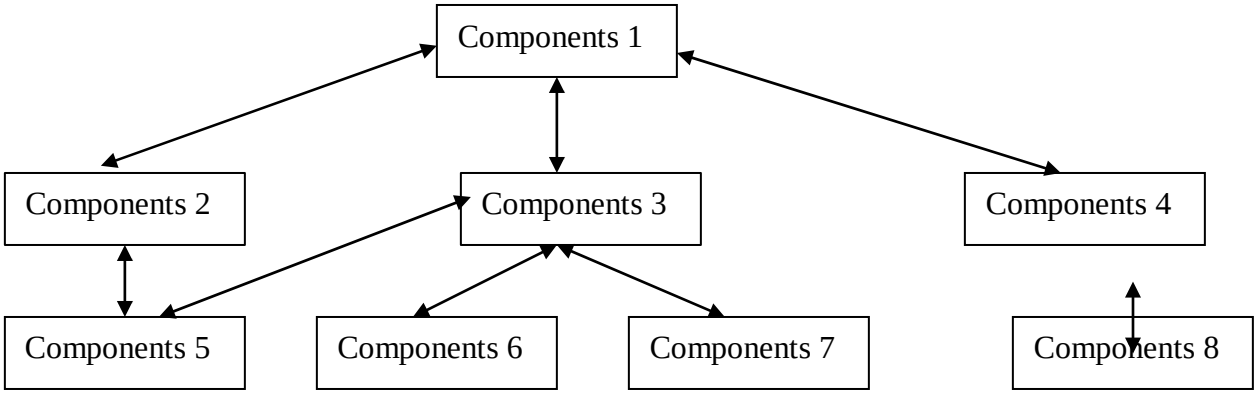
1. Top Down Integration

3. Bi-Directional Integration
4. System (Big Bang) Integration

Top-Down Integration :-

Integration Testing involves testing the topmost component interface with other components in same order as you navigate from top to bottom, till we cover all the components.

To understand this methodology, we will assume that a new product/ software development where components become available one after another in the order of component number specified .The integration starts with testing the interface between Component 1 and Component 2 .To complete the integration testing all interfaces mentioned covering all the arrows, have to be tested together. The order in which the interfaces are to be tested is depicted in the table below. In an incremental product development, where one or two components gets added to the product in each increment, the integration testing methodology pertains to only those new interfaces that are added .



Order of testing Interfaces

Steps	Interfaces Tested
1	**1-2**
2	**1-3**
3	**1-4**
4	**1-2-5**
5	**1-3-6**
6	**1-3-6-(3-7)**
7	**(1-2-5)-(1-3-6-(3-7))**
8	**1-4-8**
9	**(1-2-5)-(1-3-6-(3-7))-(1-4-8)**

For example , assume one component (component 8) is added for the current release , then the integration testing for current release need to include steps 4,7,8 and 9.

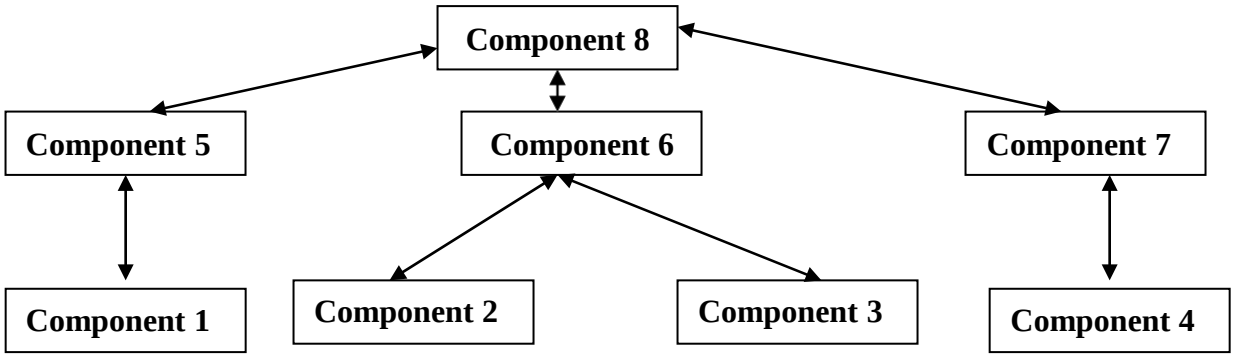
To optimize no of steps in integration(optimization of elapsed time) , following steps can be combined ,

- step 6,step 7 □ executed as single step,

Subsystem : set of components and their related interfaces can deliver functionality components is called as sub system . Ex: components in steps 4, 6 and 8 can be considered as subsystem.

Bottom-Up Integration:-

Bottom-up integration is just the opposite of top-down integration, where the components for a new product development becomes available in reverse order, starting from the bottom. Testing takes place from the bottom of the control flow upwards. Components or systems are substituted by drivers. Logic Flow is from top to bottom and integration path is from bottom to top. Navigation in bottom-up integration starts from component 1 converting all sub systems , till components 8 is reached. The order is listed below. The number of steps in the bottom up can be optimized into four steps. By combining step2 and step3 and by combining step 5-8 in the previous table.



Order of Interface tested using Bottom Up Integration

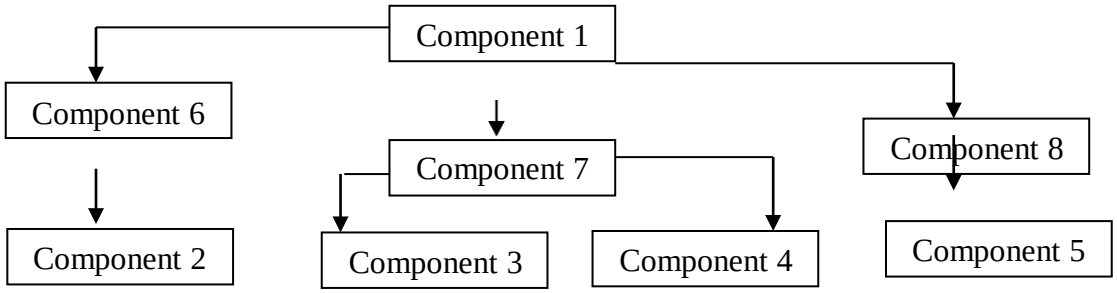
Steps	Interfaces Tested
1	1-5
2	2-6,3-6
3	2-6-(3-6)
4	4-7
5	1-5-8
6	2-6-(3-6)-8
7	4-7-8
8	(1-5-8)-(2-6-(3-6)-8)-(4-7-8)

Bidirectional Integration:- (***Sandwich Integration)*** Bi directional integration is a combination of the top-down and bottom –up integration approaches used together to derive integration steps. Let us assume software components become available in the order mentioned by the component numbers.

The Individual component 1, 2, 3, 4, and 5 are tested separately and bi-directional integration is performed initially with the use of studs and drivers. Drivers are used to provide upstream connectivity while stubs are provided for downstream connectivity.

A driver is a function which redirects the request to some other components and stubs simulate the behavior of a missing components. After the functionality of these integrated components are tested, the drivers and stubs are

discarded .Once component 6,7 and 8 becomes available, the integration methodology focus on only those components , as these are the components which need focus and are new.



Steps for Integration Using Sandwich Testing :-

Steps	Integration Tested
1	**6-2**
2	**7-3-4**
3	**8-5**
4	**(1-6-2)-(1-7-3-4)-(1-8-5)**

System (Big Bang) Integration:-

System Integration means that all the components of the system are integrated and tested as a single unit. Integration testing, which is testing of interface, can be divided into two types:-

- Components or Sub-System Integration
- Final Integration testing or system Integration

Big bang Integration is deal for a product where the interfaces are stable with less number of defects.

There are some major important disadvantages that can have a bearing on the release dates and quality of a product are as follows :-

1. When a Failure or defects is encountered during system integration, it is very difficult to locate the problem, to find out in which interface the defects exists. The debug cycle may involve focusing on specific interfaces and testing them again.
2. The ownership for correcting the root cause of the defects may be a difficult issue to pin point.
3. When integration testing happens in the end , the pressure from the approaching release date is very high. This pressure on the engineers may cause them to compromise on the quality of the product .
4. A certain components may take an excessive amount of time to be ready. This precludes testing other interfaces and wastes time till the end.

Choosing Integration Methods:-

Sno	Factors	Suggested Integration Methods
1	**Clear Requirement and Design**	**Top Down**

2	**Dynamically, Changing Requirements, Design, Architecture**	**Bottom-Up**

3	**Changing Architecture, Stable Design**	**Bi-Directional**
4	**Limited Changes to existing Architecture with less Impact**	**Big Bang**
5	**Combination of all the above**	**Select one of the above after careful analysis**

Integration Testing As a Phase of testing :-

"All testing activities that are conducted from the point where two components are integrated to the point where all system components work together , are considered a part of the integration testing phase."

The Integration testing phases focuses on finding defects which predominantly arise because of combining various components for testing, and should not be focused on for component or few components .Integration testing as a type focuses on testing the interfaces. This is a subnet of the integration testing phase.

Scenario Testing:-

Scenario testing is defined as a "set of realistic user activities that are used for evaluating the products" .It is also defined as testing involving customer scenarios. There are two methods to evolve scenario

1. System Scenario
2. Use case Scenario/ Role Based Scenarios.

System Scenario:-

System Scenario is a method where by the set of activities used for scenario testing covers several components in the system. The following approaches can be used to develop system scenarios.

Story-line : Develop a story-line that combines various activities of the product that may be executed by an end user.

Life-cycle / state transitions: Consider an object, derive the different transitions / modification that happen to the object and derive scenarios to cover them . Ex: Savings Bank Account(opening , deposit , withdraw , interest calculation etc) applied to money object

Deployment / implementation details from customer: develop a scenario from a known customer deployment / implementation details and create set of activities by various users in the implementation

Business verticals: Visualizing how a product / software will be applied to different business verticals and create a set of activities as scenarios (e.g., purchasing function is done differently in pharmaceuticals, software houses , government organization so make the product multi purpose)

Battle-ground scenarios: Create some scenarios to justify "the product works" and some scenarios to "try and break the system" to justify "the product doesn't work."

The set of scenarios developed will be more effective if the majority of the approaches mentioned above are used in combination, not in isolation. Scenario should not be set of disjointed activities which have no relation to each other. Any activity in a scenario is always a continuation of the previous activities. Effective Scenarios will have combination of current customers implementation foreseeing future use of product, and developing

adhoc test cases. Coverage is always a big question with respect to functionality in scenario testing. This is not meant to cover different permutations and combinations of features and usage in a product .

Coverage of Activities by Scenario Testing

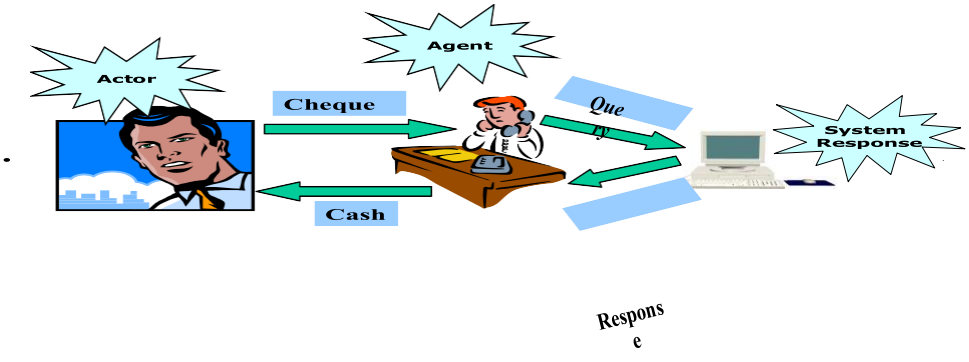
End User Activity	Frequency	Priority	Applicable Environments	No of times Covered
1.Login to Application	High	High	W2000,W2003,XP	10
2. Create an Object	High	Medium	W2000,XP	7
3.Modify Parameters	Medium	Medium	W2000,XP	5
4.List Object Parameters	Low	Medium	W2000,XP,W2003	3
5.Compose Mail	Medium	Medium	W2000,XP	6
6.Attach Files	Low	Low	W2000,XP	2
7.Send Composed Mail	High	High	W2000,XP	10

Use Case Scenarios:-

A use case Scenario is a stepwise procedure on how a user intends to use a system, with different user roles and associated parameters. A use case scenario can include stories, pictures and deployment details. Use cases are useful for explaining customer problems and how the software can solve those problems without any ambiguity

Example:-

The scenario above is explaining a example of withdrawing a cash from a bank. A customer fills up a cheque and gives it to an official in the bank. The official verifies the balance in the account from the computer and gives the required cash to the customer .The customer in this example is a actor, the clerk the agent , and the response given by the computer which gives the balance in the account , is called the system response



Actor	System Response
User would like to withdraw cash and inserts the card in ATM machine	Request for Password or PIN (Personal identification number)
User Fill-in the Password or PIN	
	Validate the password or PIN
	Give a list containing types of accounts
User selects an account type	
	Ask the user for amount
User Fill-in the amount of cash required	
	Check availability of funds Update account balance Prepare receipt Dispense cash
Retrieve cash from ATM	
	Print receipt

Actor and System Response in Use Case for ATM cash withdrawal

DEFECT BASH:-

1. Defect bash is an *ad hoc* testing, done by people performing different roles to bring out all types of defects. It is very popular among applications development companies, where the products can be used by people who perform different roles. The testing by all the participants during the defect bash is not based on written test cases. Defect bash brings together plenty of good practices that are popular in testing industry. They are as Follows :-Enabling people to "***cross boundaries and test beyond assigned area***"
2. Bringing different people performing different roles together in the organization for testing - "***Testing isn't for testers alone***"
3. Let everyone in organization use the product before delivery - "***Eat your own dog food***"
4. Bringing fresh pairs of eyes to uncover new defects – "***Fresh eyes have less bias***"
5. Bringing in people who have different levels of product understanding, to test the product together randomly – "***Users of software are not the same***"
6. Testing doesn't wait for the time taken for documentation – "***Does testing wait till all documentation is done?***"
7. Enabling people to say the "system works" as well as enabling them to "break the system" – "***Testing isn't to conclude that the system works or doesn't work***"

Even though it is said that defect bash is an ad hoc testing, not all activities of defects bash are un planned. All the activities in the defect bash are planned activities, except for what to be tested .It involves several

steps:-

1. **Choosing the frequency and duration of defect bash.**
2. **Selecting the right product build.**
3. **Communicating the objectives of each defect bash to everyone**
4. **Setting up and monitoring the lab for defect bash.**
5. **Taking action and fixing issues.**
6. **Optimizing the effort involved in defect bash.**

1. Choosing frequency and duration

- Too frequent or too few rounds may not meet objective
- Optimize duration involved

2. Selecting right product build

- Good-quality product
- Regression tested build
- Too many defects spoil confidence

3. Communication objective of defect bash

- Purpose & objective has to be clear

- Areas of focus to be communicated
- Defects that can be found easily by test team shouldn't be objective

4. Setting up and monitoring lab

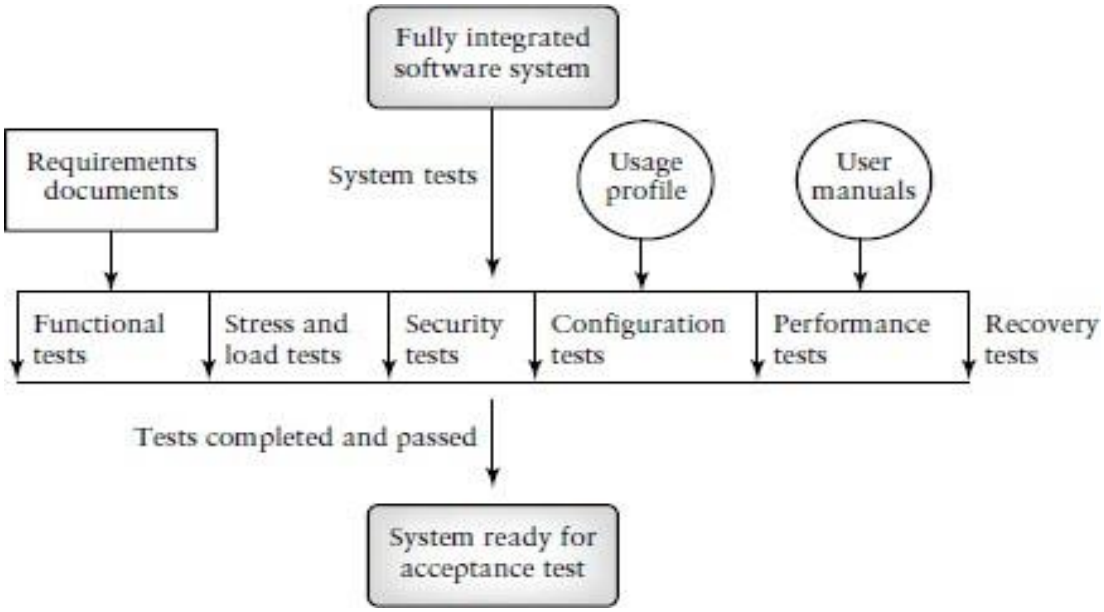
- Right configuration and resources
- Easy install & set-up help
- Optimized for both functional & non-functional defects
- Monitor all resources (RAM, disk, CPU, network)

5. Taking actions and fixing issues

- Duplicate defects
- Not possible to look at each defect alone due to volume
- Code reviews and inspections
- Communication to all users on defects and their resolution

System Testing

- When integration tests are completed, a software system has been assembled and its major subsystem have been tested
- System test planning should begin at the requirement based (black box) test
- System test planning is a complicated task. There are many components of the plan that need to be prepared such as test approaches, costs, schedules , test cases and test procedures
- System testing itself requires large amount of resources
- The goal to ensure that the system performs according to its requirements.
- System test evaluates both functional behavior and quality requirement such as reliability, usability, performance and security.
- The phase of testing is especially useful for detecting external hardware and software interface defects. Eg:- those causing race conditions, deadlocks , problems with interrupts and exception handling.
- The organization will want to sure that the quality of the software has been measured and evaluated before users/client are invited to use the system.
- In fact system test serves as a good rehearsal scenario for acceptance test.
- System test often requires many resources, special lab equipment.
- The best scenario is for the team to be part of an independent testing group.
- There are several types of system tests
    - Functional testing , Performance Testing
    - Stress testing , Configuration testing
    - Security Testing ,Recovery testing



- A load is a serious of input that stimulated a group of transaction

  A transaction is a unit of work seen from the system user's view

  A transaction consist of set of operations that may be performed by a person , software system or a device that is outside the system.

  A use case can be used to describe a transaction

- Ex : a telecomm system load that simulated a series of phone calls (transactions) of particular types and lengths arriving from different locations
- A load can be a real load, that is we can put the system under test to real usage by having actual telephone users connected to it.
- Loads can also produced by tools called load generators , they will generate test input data from system test. Load generators can be simple tools that outputs a fixed set of predetermined transaction

Functional testing

- Functional test at system level are used to ensure that the behavior of the system adheres to the requirements specification. All functional requirements for the system must be achievable by the system.
- Example personal finance system is required to allow users to set up account, add, modify and delete entries in the accounts, and print reports, the function based system and acceptance test must ensure that the system can perform these tasks
- Functional test are black box in nature ,The focus is on the inputs and proper output for each function
- Improper and illegal inputs must also be handled by the system
- Goals

- All classes of illegal inputs must be rejected (however the system should remain available.
- Apossible classes of system output must exercised and examined
- All effective system states and state transition must be exercised and examined
- All functions must be exercised

- If a failure is observed, a formal test incident report should be completed and returned with the test log to the developer for code repair. Managers keep track of these forms and reports for quality assurance purposes, and to track the progress of the testing process.

Performance Testing

There are two major requirements:

- **Functional Requirement**:-

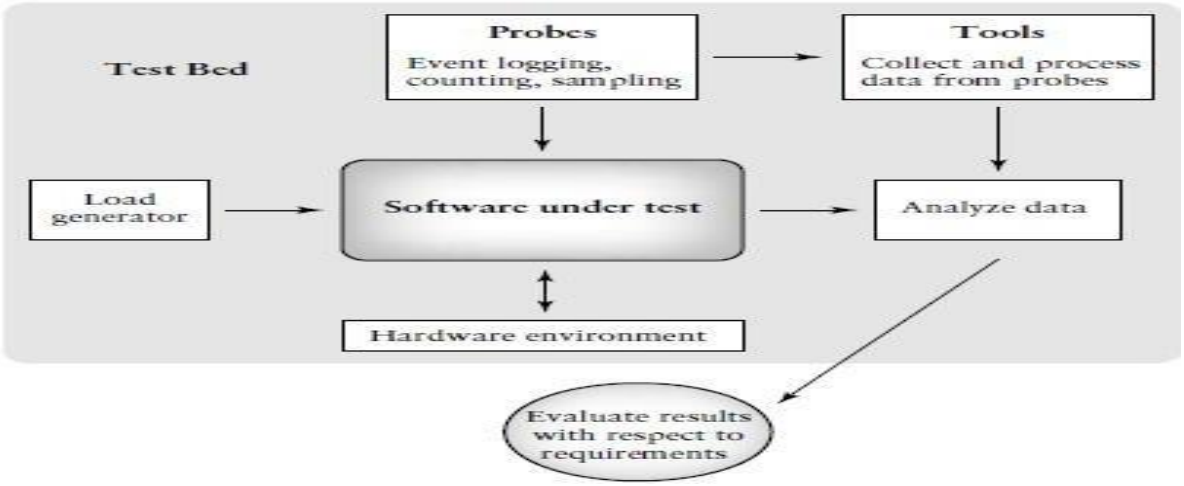
Users describes what function the software should perform. We test for compliance of these requirements at the system level with the functional based system test.

- **Quality Requirement** :-

There are nonfunctional in nature but describes quality levels expected for the software. One example of a quality requirement is performance level, the users may have objectives for the software system in terms of memory use , response time , throughput and delay

Goal : to see if the software meets performance requirements

- Testers also learn from performance test whether there are any hardware or software factors that impact on the systems requirement .
- Performance testing allows the testers to tune the system, ie to optimize the allocation of system resource
- Performance objectives stated clearly requirement documents , system test plans
- Results of Performance system test is quantifiable ex:no of CPU cycles , response time , no of transactions per second.
- Resources for the performance testing must be allocated in the system test plan, Example of resources are given below in a diagram



- A source of transaction to drive the experiment ,For example if you were performance testing on operating system you need a stream of data that represent typical user interactions .Typically the source of transaction

  for many system is load generator .
- An experimental test bed that includes hardware and software the system under test interacts with. The test bed requirement sometimes includes special laboratory equipment and space that must be reserved for the tests.
- Instruments or probes that help to collect the performance data, probes may be hardware or software in nature. Some probe tasks are event counting and event duration measurement. Eg:- if you are investigating memory requirements for your software you could use a hardware probe that collected information on memory usage as the system executes. The tester must keep in mind that the probes themselves may have an impact on system performance
- A set of tools to collect, store, process and interpret the data. Very often , large volume of data are collected, and without tools the testers may have difficulty in processing and analyzing the data in order to evaluate true performance levels.

Stress Testing

- When a system is tested with a load that causes it to allocate its resources in maximum amounts, this is called stress testing
- Eg:-if an OS is required to handle a 10 interrupts / second and the load cause 20 interrupt/ second, the system is being stressed
- Goal try to break the system; find the circumstance under which it will crash, this is sometimes called "*breaking the system"*
- Stress testing is important because it can reveal defects in real time and other types of systems, as well as weak areas where poor design could cause unavailability of services.
- Stress testing often uncovers race conditions, deadlocks , depletion of resource in unusual or un planned patterns , and upset in normal operation of the software system.
- System limits and threshold values are exercised , Hardware and software interactions are stretched to the limit
- Stress testing is supported by many of the resource used for performance test as shown in previous diagram , This includes the load generator , The tester set the load generator parameter so that load levels cause stress to the system
- Stress testing is important from the user/client point of view
- When system operate correctly under conditions of stress then client have confidence that the software can perform as required.

Configuration Testing

- It allows developer/tester to evaluate system performance and availability when hardware exchanges and reconfigurations occurs.

- Software Systems interact with hardware devices such as disc drivers, tape drivers and printers
    - Many Software system also interact with multiple CPU some of which are redundant
    - Eg:- a printer of type X should be substitutable for a printer of type Y, CPU A should be removable from a system composed of several other CPUs
    - Sensor A should be replaced with Sensor B
- Software will have a set of commands or menus that allow users to make these configuration changes
- If a system does not have specific requirements for device configuration changes then large-scale configuration testing is not essential.

- According to Beizer configuration testing has the following objectives
    1. Show that all configuration changing commands and menus work properly
    2. Show that all interchangeable and that they each enter the proper states for the specified conditions
    3. Shows that the system performance level is maintained when devices are interchanged, or when they fail
- Several types of operations should be performed during configuration test, some sample operations for tester are:-
    - Rotate and Per mutate the position of devices to ensure physiological/logical device permutations work for each device
    - Induce malfunctions in each devices , to see if the system properly handles the malfunction
    - Induce multiple device malfunctions to see how the system reacts
    - These operation will help to reveal problems (defects) relating to hardware and software when hardware exchange, and the reconfiguration occur.

Security Testing:-

- Designing and testing software system insure that they are safe and secure is a big issue facing software developers and test specialist
- Safety and Security is a big issue because of the Internet
- Users/Client should be encouraged to make sure their security needs are clearly known at requirement time, so that security issues can be addressed by designers and Testers.
- Computer Software and data can be compromised by
    - Criminals, intent on doing damages, stealing data and information, causing denial of service , invading privacy
    - Errors on the part of honest developers/ maintainers who modify, destroy or compromise data because of misinformation , misunderstanding , and/or lack of knowledge

Attacks can be random or systematic. Damage can be done through various means such as:-

- Viruses
- Trojan Horses
- Trap Doors

The effect of security breaches could be extensive and can cause

- Loss of information
- Corruption of information
- Privacy violations
- Denial of service

- Physical, psychological and economic harm to process or property can result from security breaches
- Developers try to ensure the security of their systems through use of protection mechanism such as passwords, encryption , virus checkers and the detection and elimination of trap doors
- Password checking and example of other areas to focus on during security testing are described below
  - **Password Checking**:- Test the password checker to insure that users will select a password that meets the condition descried in the password checker specification. Equivalence class partitioning and boundary value analysis based on the rules and conditions that specify a valid password can be used to design the tests .
  - **Legal and Illegal Entry with password**:- Test for legal and illegal system/data access via legal and illegal passwords.
  - **Password Expiration:-** If it is decided that password will expire after certain time period, tests should be designed to insure the expiration period is properly supported and that users can enter a new and appropriate password.
  - **Encryption:-**Design test cases to evaluate the correctness of both encryption and decryption algorithm for systems where data/message are encoded
  - **Browsing:-** Evaluate browsing privileges to insure that unauthorized browsing doesn't occur. Tester should attempt to browse illegally and observe system responses. They should determine what types of private information can be inferred by both legal and illegal browsing
  - **Trap Doors:-** Identify any unprotected entries into the system that may allow access through unexpected channel (trap doors) .Design test cases that attempt to gain illegal entry and observe results. tester will need to support of designer and developers for this task
  - **Viruses:-** Design test to insure that system virus checkers prevent or curtail entry of viruses into the system. Tester may attempt to infect the system with various viruses and observer the system response.

Recovery Testing:-

- Recovery testing subjects a system to losses of resources in order to determine if it can recover properly from these losses. This type of testing is important for transaction system.
- Eg:- on line banking software
- A test scenario might be to emulate loss of device during a transaction, Test would determine if the system could return to a well known state ,and that no transaction have been compromised.

- They usually have multiple CPU and /or multiple instance of devices , and mechanical to detect the failure of the device.They are also called as "*CHECK POINTS"*
- Beizer advises that tester focus on the following areas during recovery testing
- ***Restart:-*** The current system state and transaction state are discarded The most recent checkpoint record retrieved and the system initialized to the state in the checkpoint record. Tester must insure that all transaction have been reconstructed correctly and that all devices are in proper state. The system should then be able to begin to process new transaction
- ***Switchover:-*** The ability of the system to switch to a new processor must be tested .Switch over is the result of a command or detection of faulty processor by a monitor
- All transaction and processes must be carefully examined to detect:-
    - Loss of transaction
    - Merging of transaction
    - Incorrect Transactions
    - An unnecessary duplication of transaction

Difference between functional and non functional Testing

System test contains both functional and non functional Testing

Testing aspect	**Functional Testing**	**Non Functional Testing**
Involves	Product Features and functionality	Quality Factor
Tests	Product behavior	Behavior & Experience
Result Conclusion	Simple steps written to check expected results	Huge data collected and analyzed
Results Varies Due to	Product Implementation	Product Implementation , resources and configuration
Testing Focus	Defect detection	Qualification of product
Knowledge required	Product and domain	Product ,domain, design ,architecture ,statistical skills
Failures normally due to	Code	architecture , design ,code
Testing Phase	Unit, component, integration , system	System
Test case Repeatability	Repeated Many Times	Repeated only in case of failures and for different configuration
Configuration	One time setup for a set of test cases	Configuration changes for each test case
Example	1. Design / architecture verification 2. Business vertical testing 3. Deployment Testing 4. Beta Testing 5. Certification standards and Testing for compliance	1. Scalability Test 2. Performance Test 3. Reliability Test 4. Stress Test

ACCEPTANCE TESTING

- It is done by the customer or by the rep of the customer to check whether the product is ready for use in the real life environment.
- Customer defines a set of test cases that will be executed to qualify and accept the product
- Small in numbers, black box type of test cases
- Written to execute real life scenarios , verifying both functional & non functional aspects of the system
- Done prior to product delivery , sometimes jointly developed by the customer and product organization

1. **Acceptance Criteria**
   a. **Acceptance Criteria(AC) – Product acceptance**
   Acceptance criteria is not meant for executing test cases that have not been executed before. hence existing testcases are looked at and certain categories of test cases can be grouped as AC.
   Ex: all performance TC should pass to meet response time requirements
   b. **Acceptance Criteria – Procedure acceptance**
   It can be defined based on the procedures followed for delivery. It could be documentation and release media. Example
      - User , admin and troubleshooting doc should be part of the release
      - Along with binary code , source code of the product build scripts to be delivered in CD
      - A minimum of 20 employees are trained on the product usage prior to deployment

   c. **Acceptance Criteria – service level agreements(SLA)**
   service level agreements are part of contract signed by the customer and product organization. Important contract items are taken and verified.
   For Ex: time limits to resolve defects mentioned in SLA
      i. All major defects that come up during first 3 months of deployment need to be fixed free of cost
      ii. Down time of the implemented system should be less than 0.1%
      iii. All major defects are to be fixed within 48 hours of reporting

2. **Selecting test cases for Acceptance testing**
   a. End to End functionality verification
   b. Domain Test
   c. User Scenario test
   d. Basic Sanity Test
   e. New Functionality Test
   f. A few Non functional Tests
   g. Test Pertaining to legal obligations and service level agreements
   h. Acceptance test data
3. **Executing Acceptance Tests**

- If it is done by product organization ,forming a team is an important activity.
- It contains Product management , support, consulting team
    - 90 % --people with business process knowledge , 10% -- tech testing team
- Testing Team may or may not aware of testing , so appropriate training on the product and the process must be given . This could be in-house training material.
- The testing team members constantly interact with acceptance team members & help them
    - to get required test data,
    - select and identify test cases
    - analyze the acceptance test result
- During test execution , the acceptance test team reports its progress regularly

REGRESSION TESTING

- Regression testing is not a level of testing, but it is the retesting of software that occur when changes are made to ensure that new version of the software has retained the capability of the old version and no new defects has been introduced due to the changes. Regression Testing can occur at any level of test
- Ex:- when unit test are run the unit may pass a number of these tests until one of the test does reveal a defect. The unit is repaired and then retested with all the old test cases to ensure that the changes have not affected its functionality
- Regression testing is **selective re-testing** of the system with an objective to ensure that the bug fixes work and those bug fixes have not caused any un-intended effects in the system
- This testing is done to ensure that:
    - The bug-fixes work
    - The bug-fixes do not create any side-effects

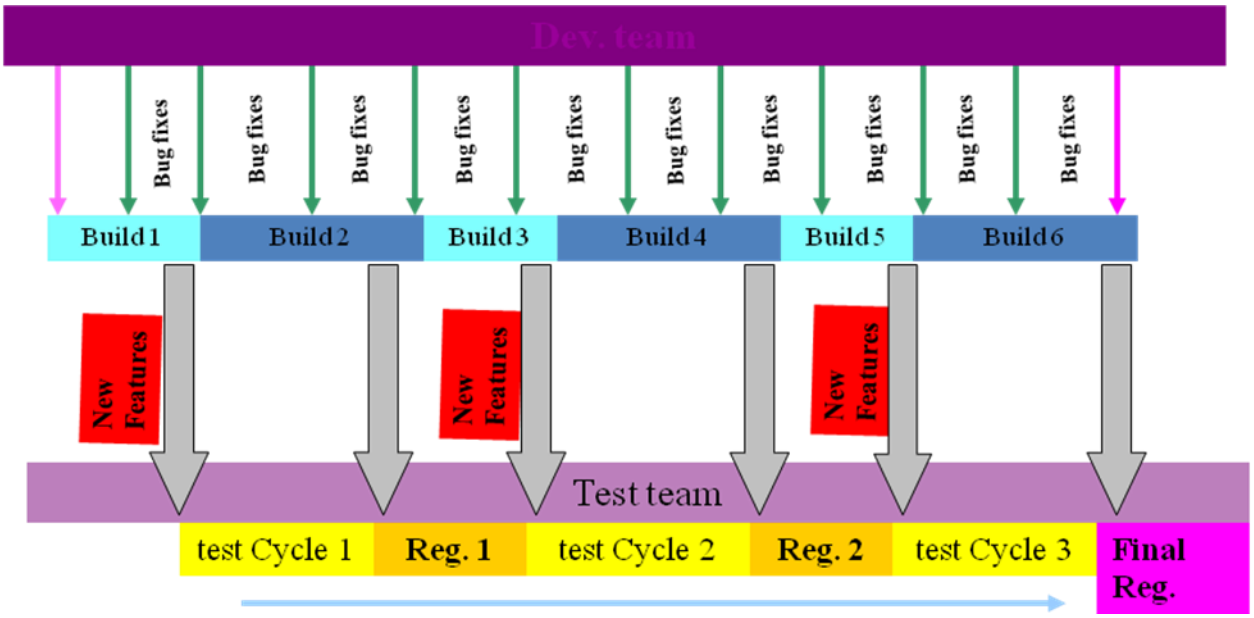
Regression Testing – Types

I. Final regression testing

- Unchanged build exercised for the minimum period of "cook time" (gold master build)
- To ensure that "**the same build of the product that was tested reaches the customer**"
- More critical than any other type of testing
- Used to get a comfort feeling on the product prior to release

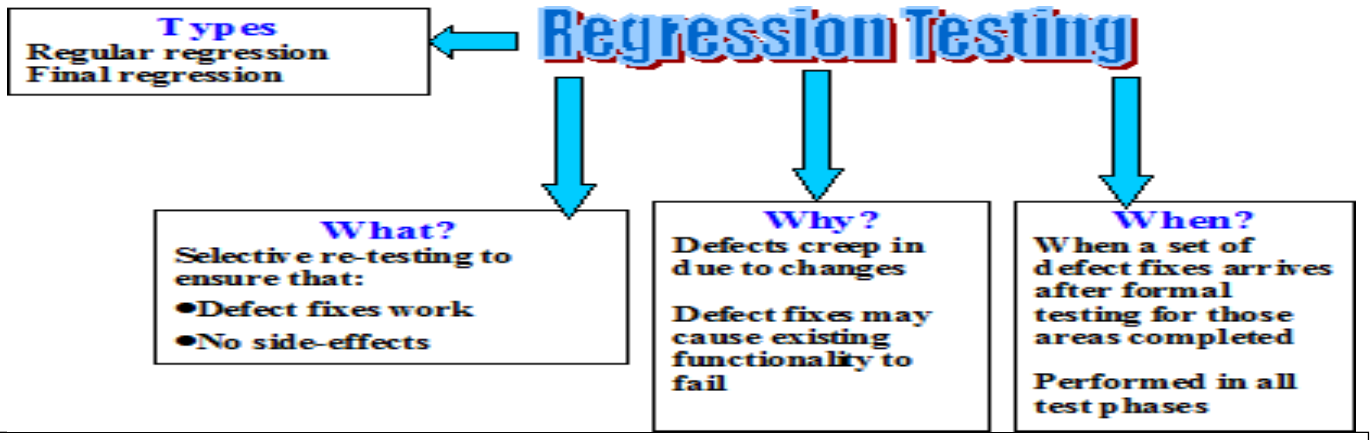
II. Regression testing

- To validate the product builds between test cycles
- Unchanged build is recommended but not mandatory
- Used to get a comfort feeling on the bug fixes, and to carry on with next cycle of testing
- Also used for making intermediate releases (Beta,Alpha)



When to do regression testing?

1. A reasonable amount of initial testing is already carried out
2. A good no of defects have been fixed
3. Defect fixes that can produce side-effects are taken care of

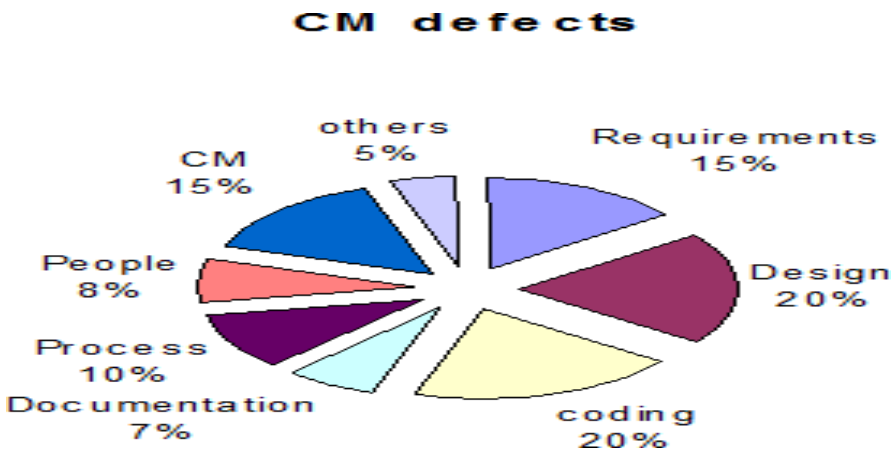


STEPS IN REGRESSION TESTING

1. Performing initial smoke tests
2. Understand the criteria for Selecting test cases
3. Classifying test cases
4. Methodology for selecting the TC
5. Resetting test cases for execution
6. How to conclude results

1. Performing initial smoke tests

- Identify the basic functionality that product must satisfy
- Designing the test cases to ensure that these functionality works , package them into smoke test suite
- Ensuring that every time the product is build this suite is run successfully before anything else is run

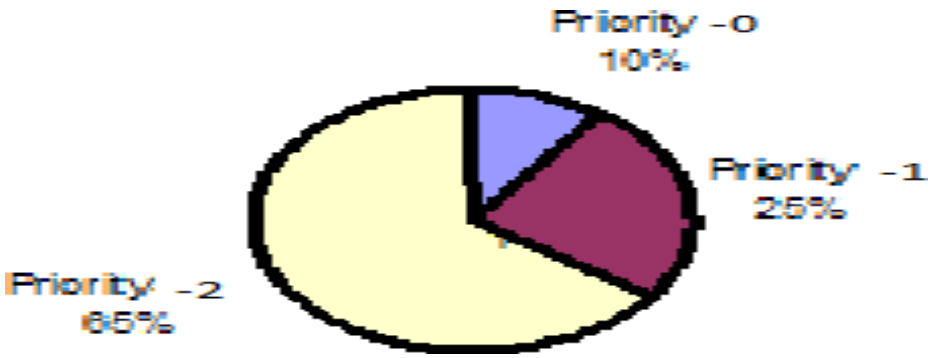


- If this suite fails , escalating to the developer to identify the changes or roll back to the state where smoke test suite succeeds

2. Understand the criteria for Selecting test cases

1. Include TC that has max defects.
2. Include TC where changes are made.
3. Include TC that test the basic functionality.
4. Include TC in which problems are reported.
5. Include TC that test end-to-end behavior.
6. Include TC for positive conditions.
7. Include the TC that are visible to the user.

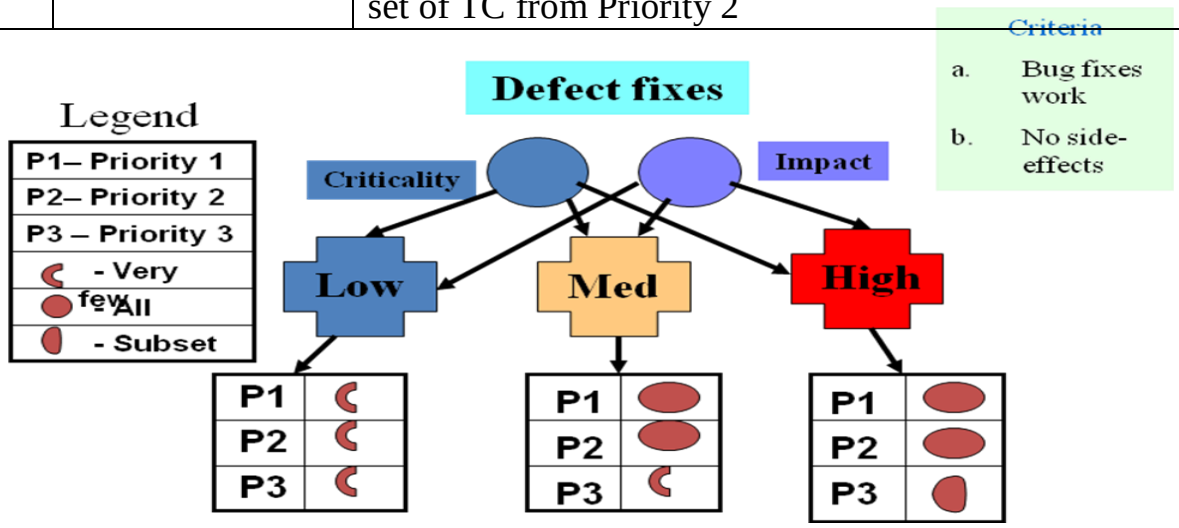
3. Classifying test cases



Priority -0	Called sanity Test case Check basic functionality & are run for accepting the build for further testing Done when a project goes through major change
Priority -1	Uses the basic and normal setup and these test cases deliver high project value to both development team and to customer
Priority -2	It deliver moderate project value Executed as a part of testing cycle

4. Methodology for selecting the TC

	Criticality & Impact of defect fixes	Action
Case 1	Low	Select few Test cases from the test case database(TCDB), execute them , they fall under priority 0,1,2
Case 2	Medium	execute test cases from priority 0,1 , few from Priority 2
Case 3	high	Execute all test case from priority 0,1 & carefully select sub set of TC from Priority 2



Alternative methodology :

1. Regress all
2. Priority based (priority 0,1, 2)
3. Regress Changes
4. Random Regression
5. Context based dynamic regression

5. Resetting test cases for execution

- When there is a major change in the product
- When there is a change in the build procedure that affects the product
- In a large release cycle where some test cases have not been executed for a long time
- When you are in the final regression test cycle with a few selected test cases
- In a situation in which the expected results could be quite different from history

6. How to conclude results

Current Result from regression	Previous Results	Conclusion	Remarks
FAIL	PASS	FAIL	• Regression failed and • Apply RESET guidelines and proceed after getting new build

PASS	FAIL	PASS	• Bug fixes are working and • Continue your regression to find side effects

FAIL	FAIL	FAIL	• Bug fixes not working • Wrong selection
PASS(with work around)	FAIL	Analyze the work round and if satisfied mark as PASS	• Work round needs good review as they create side effects
PASS	PASS	PASS	• This test case could have been included for finding side-effects or Wrong selection

INTERNATIONALIZATION TESTING:-

- **Introduction**
- **Primer**
- **Terminology**
- **Test phases for I18n**
- **Enabling testing**
- **Locale testing**
- **I18n validation**
- **Fake language testing**
- **Language testing**
- **Localization testing**
- **Tools**

Introduction:-

Market of software is becoming truly global. The advent of Internet has removed some of the technology barriers on widespread usage of software products and has simplified the distribution of Software Products

Building Software for the International market, supporting multiple languages, in a cost effective and timely manner is a matter of using internationalization standards throughout the software development life cycle- from requirements capture through design, development, testing and maintenance.

If some Guidelines are not followed in the SDLC for internationalization, the effort and additional cost to support every new language will increase significantly overtime. Testing for Internationalization is done to ensure that the software does not assume any specific language or conventions associated with a specific language. Testing for language or conventions associated with a specific language. Testing for Internationalization has to be done in various phases of SDLC.

Terminology Used in Internationalization

Definition of Language :-

- Language – Language is a tool used for communication. Language has Semantics or the meanings associated with the sentences. For the same language , the spoken usage , word usage and grammar could vary from country to another, however the character /alphabets may remain the same in most cases.

Character Set

- ASCII – American Standard Code for Information Interchange:
  - Uses 8 bit for representing characters
  - Basic ASCII uses 7 bits (128 chars) and extended ASCII uses 8 bits (256 chars)
  - European characters and punctuation symbols are easily represented in extended ASCII
  - It also includes accented chars (ñ, á, é, í, ó, ú)
- Double Byte Character Set (DBCS) :
  - Many of the languages (Chinese & japanese) can be represented in 8 bits.
  - DBCS uses 16 bits to represent characters.
  - In DBCS, 65536 different characters can be represented
- Unicode:
  - ASCII & DBCS represents characters of a single language
  - Unicode represents all characters of all languages
  - Unicode assigns a unique code to each character no matter what language or program or platform
  - Uses 16 bit encoding
  - Unicode transformation format : Specifies algorithmic mapping of character into Unicode
  - Each language has a unique number in Unicode

Microsoft operating System	Path to Internationalization
Windows 3.1	ANSI
Windows 95	ANSI and limited Unicode
Windows NT 3.1	First OS based on Unicode
Windows NT 4.0	Display of Unicode characters
Windows NT 5.0	Display and input of Unicode characters

- Locale
  - Each of the languages is spoken differently in different countries and states
  - There could be many countries speaking the same language, using the same characters, punctuations, etc.
  - But some conventions may be different (currency and date format)
  - For example, English is used widely in the US and India, but
    - Currency : $ and Rs.
    - $1,000,000 and Rs. 1,00,000
  - There could be multiple currencies in a country (Euro and Franc in France)
  - There could be multiple locale for a language in the same country

Terminology

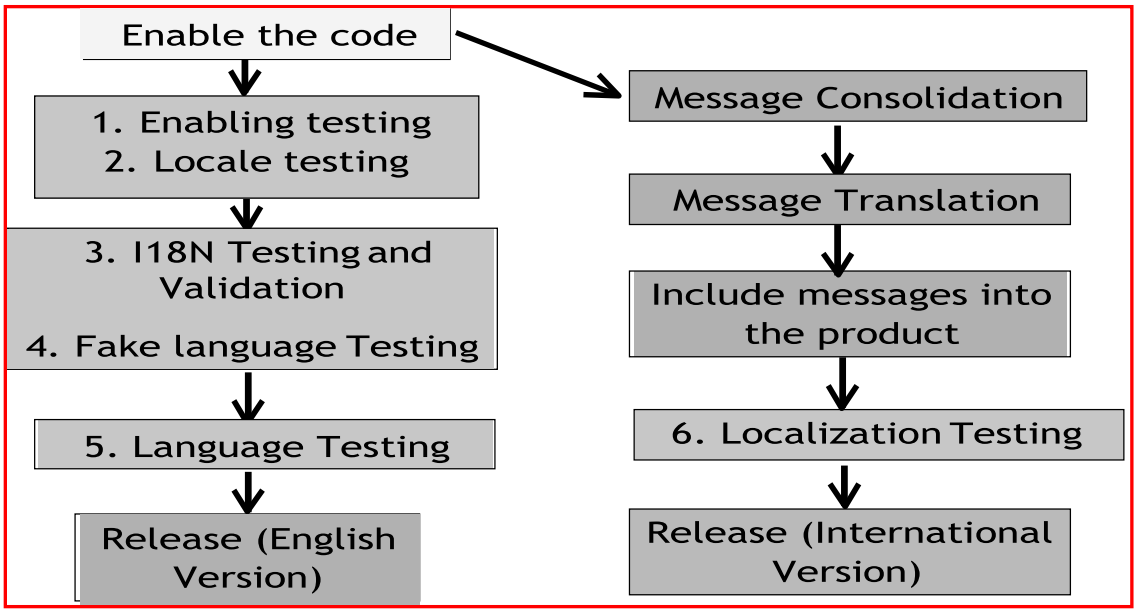
- **Internationalization(**I18*n*)
  - also called I18*n*, the subscript 18 is used to mean that there are 18 characters between "I" and the last "n" in the word Internationalization
  - Testing is done in various phases to ensure that all those activities are done right is called inter nationalization testing or I18*n* testing.
  - Represents all activities to make products available to the international market
  - Includes both DEV & testing activities
- **Localization (**L10*n)*
  - Also called L10*n*, the subscript 10 is used to indicate that there are 10 characters between "L" and "n" in the word Localization.
  - Translation of all product messages and documentation
  - Done by language experts
  - Includes both DEV & testing activities
- **Globalization**
  - Not very popular
  - Also called **G12*n***
  - Internationalization includes localization but some companies want to separate as the team that does

both are different, and hence this term

GLOBALIZATION= INTERNATIONALIZATION+ LOCALIZATION

Test Phases for Internationalization Testing :- Testing for Internationalization requires a clear understanding of all activities involved and their sequence. The job of testing is to ensure the correctness of activities done earlier by other teams.

The Major Activities in Internationalization Testing



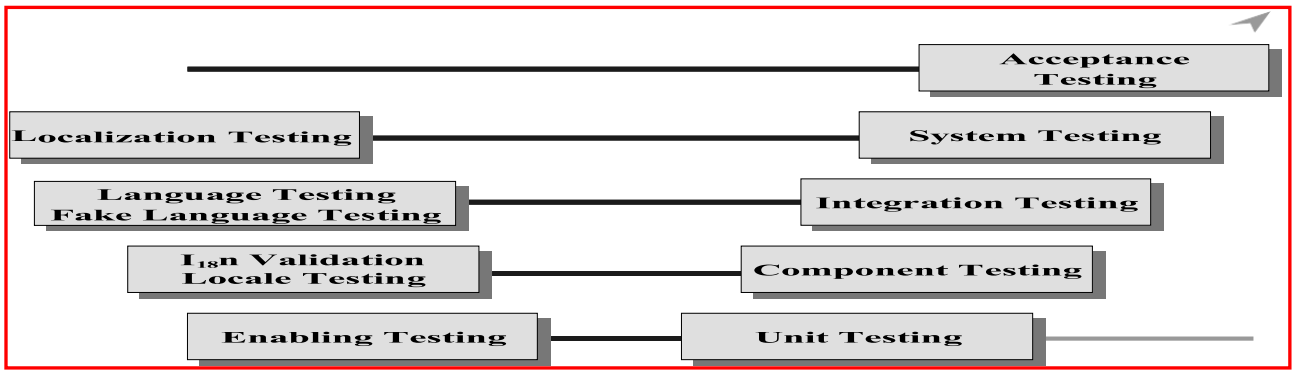
The Testing for internationalization is done in multiple phases in the project life cycle. The diagram below Elaborates the SDLC V model described and how the different phases of this model are related to various I18n testing Activities. Enabling testing is done by the developer as a part of the Unit testing Phase.

Some Important Aspects of Internationalization testing are:-

1. Testing the code for how it handles input, strings and sorting items;
2. Display of Messages for Various Languages; and
3. Processing of Messages for Various Languages and Conventions.

Acceptance Testing

Localization Testing System Testing



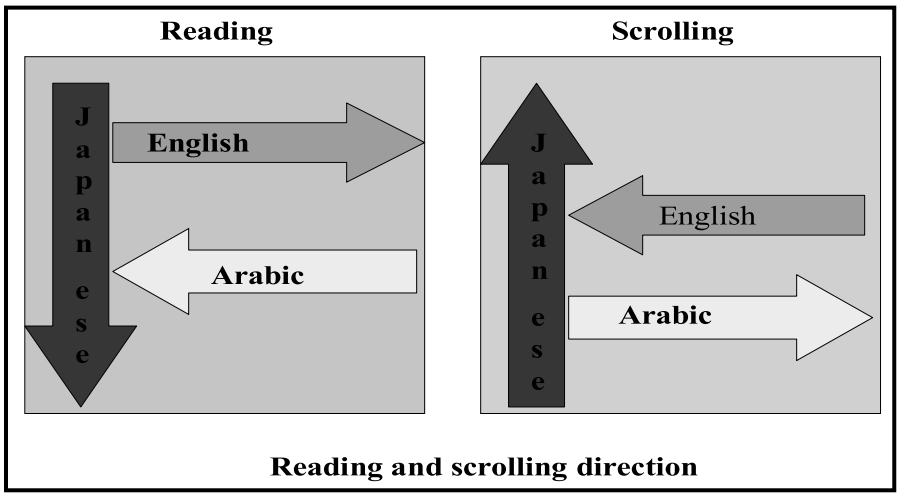
Enabling Testing:-

Enabling Testing is a white box testing methodology , which is done to ensure that the source code used in the software allows internationalization. A source code , which has hard coded currency format and date format, fixed length GUI screens or dialog boxes, read and print messages directly on the media is not considered enabled code. ***An activity of code review or code inspection mixed with test cases for unit testing, with an objective to catch I18n defects is called enabling testing.*** The year 2000 is a classic I18n defect. Enabling testing finds the majority of I18*n* defects If this is not done in the unit test phase, exponential effort has to be spent in later phases as it impacts code, design, etc. Also other I18*n* testing for fake language, l10*n* has to be repeated.

Enabling Testing – Checklist:-

- Find out those APIs/function calls that can't be used for I18n (printf, scanf) – NLSAPI, unicode, GNU gives some APIs instead
- Check the code for hard-coded date, currency format, ASCII usage or character constants.
- Check the code for arithmetic operations on date ie there is no computations (additions and subtractions) done on date variables or different format forced to the date in the code.
- Check that no format is forced to date field
- Check each field in the screen for extra space (normally 50% extra space is allotted)
- Ensure that region-based messages/slang are not used (e.g., Hi, references to colour)
- Ensure no string operations are performed on the code (substring search, concatenation); only APIs provided by I18n are to be used
- The code does not assume any predefined path, filename, directory name in NLS directory
- Check code doesn't make any assumptions about bit representation (8, 16, 32), and bit operations are not used
- Ensure adequate length is allocated to accommodate translated messages
- Check that pictures, logos and bitmaps do not have embedded text
- Ensure that all messages have code in-line comments for helping translators (e.g. pre-ponement of meeting)
- Ensure all resources are (dialog boxes, screen shots, bitmaps, etc.) Ensure technical jargons are not used and that the text may be understood by even the least skilled(e.g, pipe overflow)
- Ensure that the directions of reading / writing are opposite to scrolling, and that they follow the language

convention



Locale Testing:-

Locale Testing is not as elaborate procedure as enabling testing. The focus of Locale testing is Limited to :-

- Changing the different locale using the system settings or environment variables, and testing the software functionality, number, date, time and currency format is called locale testing.
- It is to used validate the effects of locale change in the product. A locale change affects date, currency format, the display of items in the screen, dialog boxes and the text.
- Black box methodology tests all component features for each locale.
- In Microsoft Windows 2000, you can change locale by clicking "Start->Settings->Control Panel->Regional options (demo).

Locale Testing focuses on testing the conventions for number, punctuations, date and time, and currency format.

Locale Testing - Checklist

- All features that are applicable to I18n are tested with different locales of the software for which it is intended.
- Some of the activities that need not be considered for I18n testing are auditing, debug code, log of activities and such features that are used only by English administrators and programmers.
- Hot keys, function keys and help screens are tested with different applicable locales (this is to check whether locale change would affect the keyboard settings).
- Date and time format is in line with the defined locale of the language. For example if the US English locale is selected, the software should display data in mm/dd/yyyy date format.
- Currency is in line with the selected locale and language. For example, currency should be AUS$ if the language is AUS English.
- Number format is in line with the selected locale and language. For example, the correct decimal punctuations are used and the punctuation is put at the right places.

- Time zone information and daylight saving time calculations (if used by the software) are consistent and correct.

Internationalization Validation:-

Objectives of I18*n* the validations is performed with the following objectives:-

1. The software is tested for functionality with ASCII, DBCS, and European characters•
2. The software handles string operations, sorting, sequencing operations as per the language and characters selected
3. The software display is consistent with characters that are non-ASCII in GUI and menus
4. The software messages are handled properly

I18n Validation – Input Method Editor

This is a soft keyboard used to enter non-English characters into the product. IME soft keyboard for Japanese.

I18n validation – Checklist:-

1. The functionality in all languages and locales are the same.
2. Sorting and sequencing the items are as per the conventions of language and locale. For example if $ is mentioned as the currency symbol for USA, sorting should take care of symbol & punctuations.
3. The input to the software can be in non-ASCII (Use of tools such as IME) and functionality is consistent with non-ASCII.•
4. The non-ASCII characters in the name are displayed as they were entered.
5. The cut or copy -and-paste of non-ASCII characters retains their style after pasting, and the software functions as expected.
6. The software functions correctly with different languages / words / names generated with IME and other tools; for example, Login should work with an English user name as well as with a German user name with some accented characters.
7. The documentation contains consistent documentation style and punctuations, and all language / locale conventions are followed for every target audience.

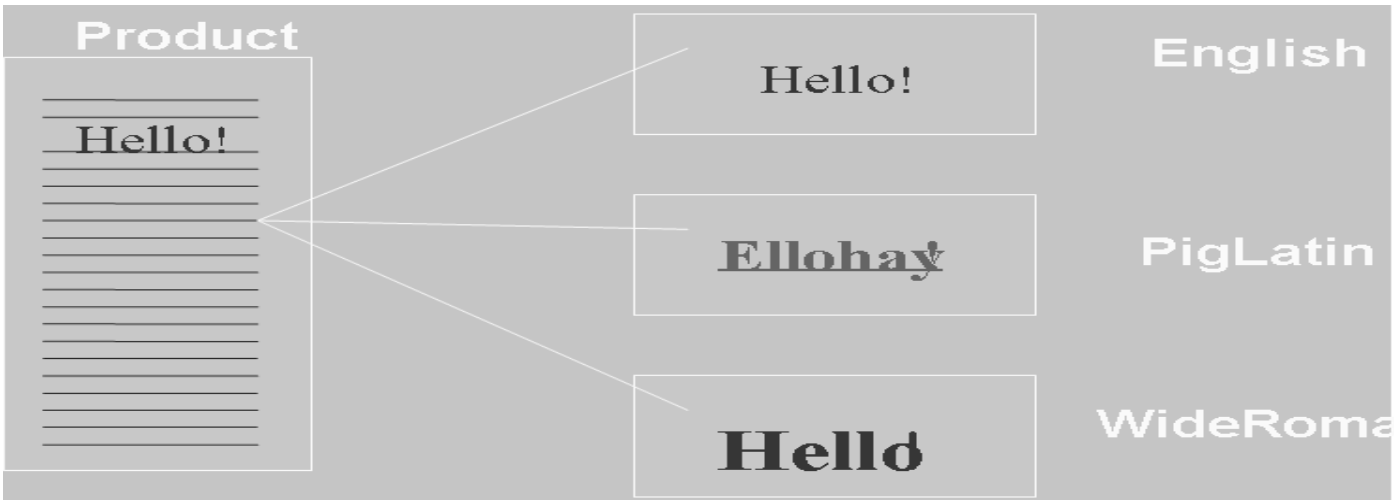
8. All the runtime messages in the software are as per the language, country terminology and usage along with proper punctuations; for example, the currency amount 123456789.00 should get formatted as 123,456,789.00 in the US and as 12,34,56,789.00 in India)

I18 *n* Validation Focuses on component functionality for Input/ Output of Non English Messages.

Fake Language Testing

Fake Language testing uses software translators to catch the translation and localization issues early. This also ensures that switching between languages works properly and correct messages are picked up from proper directories that have the translated messages. Fake Language testing helps in identifying the issues proactively before the product is localized. For this purpose , all messages are consolidated from the software , and fake language conversion are consolidated from the software, and fake language conversion are done by tools and tested. The Fake language translators use English like Target Languages, which are easy to understand and test. This type of testing helps English testers to find the defects that may otherwise found only by Language Experts during Localization Testing .

In the figure there are two English like Fake Languages used (Pig Latin and Wide Roman) A message in the program, "Hello" as "Ellohay" in Pig Latin and "Hello" in Wide Roman .This helps in identifying whether the proper target language has been picked up by the software when language is changed dynamically using system setting .



The Following items in the checklist can be used for Fake Language Testing:-

1. Ensure the software functionality is tested for at least one of the European single byte fake languages (e.g., Pig Latin) Ensure the software functionality is tested for at least one double byte language (e.g., Wide Roman)
2. Ensure all strings are displayed properly in the screen
3. Ensure the screen width, size of pop-ups and dialog boxes are adequate for string display with the fake languages.

Fake Language testing helps in simulating the functionality of the localized product for a different language using software translator.

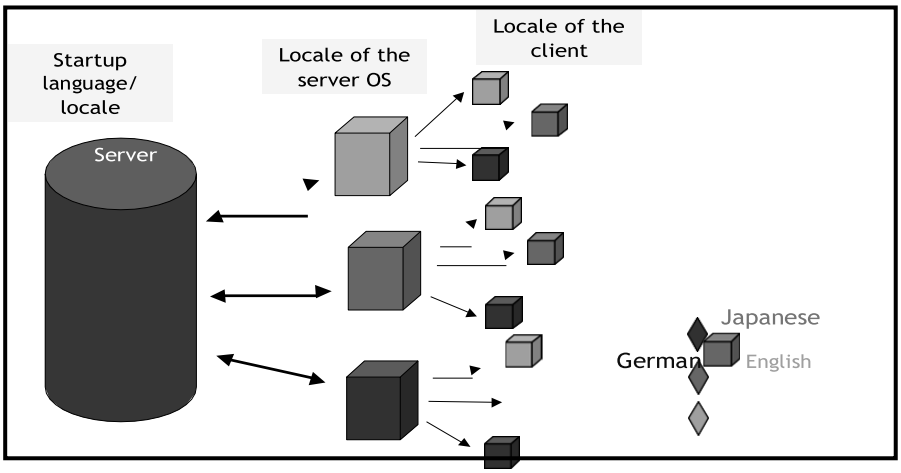
Language Testing:-

- Short form of "language compatibility testing"

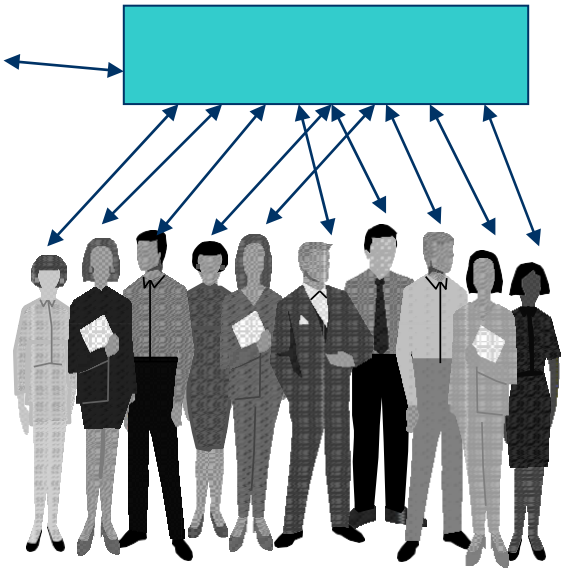
- This testing is done to ensure that on other language settings the functionality of the software is not broken and that it is still compatible across the network.
- When data is transmitted between machines or between softwares and operating systems, the code page, bit stream, message conversions taking place for internationalization.

Language Testing - Checklist

- *Check the functionality on one English, one non-English and one double-byte language platform combination.*
- *Check the performance of key functionality on different language platforms and across different machines connected in the network.*



I speak only English but can deal with anyone

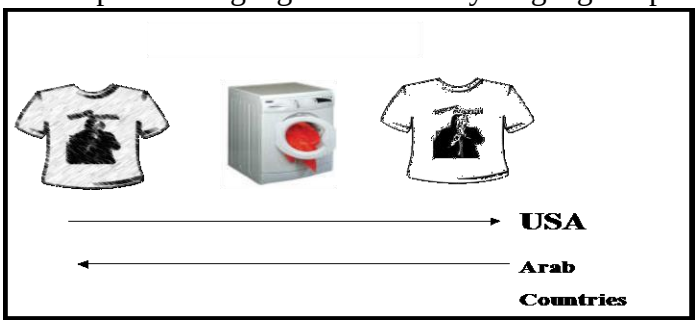


Localization Testing :-

1. Build tools consolidate all messages.
2. Documents and other artifacts are collected.
3. They are sent to language experts for translation.

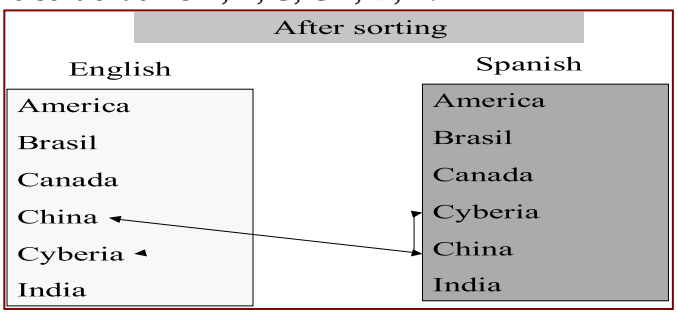
4. Process of localization is expensive.
5. Not all messages, documents need to be localized.

6. Process of localization also alters the GUI screens, dialog boxes, icons and bitmaps.
7. Process of customization.
8. The product is installed in a specific language and tested by language experts.



Localization Testing – Checklist

1. *All the messages, documents, pictures, screens are localized to reflect the native users and the conventions of the country, locale and language.*
2. *Font sizes and hot keys are working correctly in the translated messages, documents and screens.*
3. *Filtering and searching capabilities of software work as per the language and locale conventions.*
4. *Addresses, phone numbers, numbers and postal codes in the localized software are as per the conventions of the target user.*
5. *Sorting and case conversions are right as per language convention; for example, sort order in English is A, B, C, D, E, whereas in Spanish the sort order is A, B, C, CH, D, E.*



Sort Order in English and Spanish

Tools Used For Internationalization :-

There are several tools available for internationalization. These largely depend on the technology and platform used. For Example, the tools used for client server technology is different from those for web services technology using Java.

S.No	Sample list of tools for Microsoft OS	Name of tool for Linux OS	Purpose
1	MS localization Studio	GNU gettxt ()	Enabling and enabling testing
2	http://BabelFish.Altavista.com	http://BabelFish.Altavista.com	Fake language testing
3	MS regional settings	LANG and set of env variables	Locale testing
4	IME	Unicode IME (several variants exist from several companies)	I18n validation
5	http://www.snowcrest.net/donnelly/piglatin.html	http://www.snowcrest.net/donnelly/piglatin.html	Fake language testing Pig Latin)

ADHOC TESTING :-

- **Overview**
- ***Ad hoc* testing Vs planned testing**
- **Buddy testing**
- **Pair testing**
- **Exploratory testing**
- **Iterative testing**
- **Agile & extreme testing**
- **Defect seeding**
- **Defect bash**
- **Drawbacks of *ad hoc* testing**

All types of testing explained earlier are part of planned testing and are carried out using certain specific techniques (Boundary value Analysis) there are family of test types which are carried out in un planned manner hence it is named Adhoc Testing. Related Type of Adhoc Testing are

1. Buddy Testing
2. Exploratory Testing
3. Pair Testing
4. Iterative Testing
5. Agile and Extreme Testing
6. Defect Seeding

Issues of planned testing

a. Goes by level of understanding at the time of design
b. Validation of test cases happens at runtime
c. Lack of clarity on requirements impacts quality
d. Lack of skills affects quality of test cases
e. Lack of time for design affects completeness

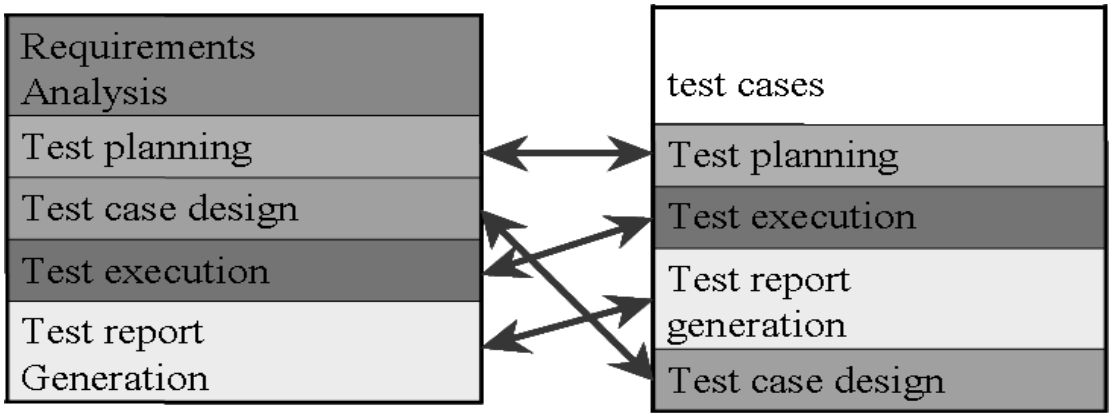
- After some of the Planned test cases are executed, the clarity on the requirement improves. Test cases written earlier may not reflect the better clarity gained in this process.
- After going through a round of planned test execution, the skills of the test engineers becomes but the test cases may not have been updated to reflect the improvement in skills.
- The lack of time for test design affects the quality of testing , as there could be missing perspectives.
- Planned Testing Enables catching certain types of defects. Though Planned tests help in boosting the testers Confidence , it is the testers “intuition” that often finds critical defects.

Definition: *Ad Hoc* Testing

Testing done without using any formal testing technique is called *ad hoc* testing.

Pesticide Paradox

One of the Principles of Software Testing explains the situation where the surviving pests in a farm creates resistance to a particular pesticide. The situation requires the farmer to use a different types of pesticide every time for the next crop cycle. Similarly, products defects gets tuned to planned test cases and those test cases may not uncover defects in the nest test cycles unless new perspectives are added. Planned Test Cases requires constant updates, sometimes even on a daily basis, incorporating the new learning. Updating test cases very frequently may become time consuming and tedious job .In such cases we have to follow Adhoc Testing.



Planned Testing ***Ad hoc Testing***

Constant interaction with developers and other project team members may lead to better understanding of the product from various perspectives. Since Adhoc tests require better understanding of the product, it is importance to stay "Connected".

Due to lack of communication, change in the requirements may not be informed to the test team. When test Engineer does not know the requirements changes, it is possible to miss few tests. This may result in a few undetected defects. It is possible to unintentionally miss some perspectives due to changed requirements.

Interaction with developers and other team members may help in getting only a set of perspectives. These type of interaction may bias the testing team. Hence it is important to constantly question the test cases and also interact with people outside the organization to find different ways of using the product and use them in adhoc testing.

Adhoc testing can be performed on a product at any time, but the return from adhoc testing are more if they are run after running planned test cases. Adhoc testing can be planned in one of two ways:-

1. After a Certain number of planned test cases are executed. In this case, the product is likely to be in a better shape and thus newer perspectives and defects can be uncovered. Since Adhoc testing does not require all the test cases to be documented immediately, this provides an opportunity to catch multiple missing perspectives with minimal time delay.
2. Prior to planned testing. This will enable gaining better clarity on requirement and assessing the quality of the product upfront.

Drawbacks of Adhoc Testing and Their Resolutions:-

Drawback	***Possible resolution***
Difficult to ensure the perspectives covered in ad hoc testing are used in future	• Document ad hoc tests after test completion
Large number of defects found in ad hoc testing	• Schedule a meeting to discuss defect impacts • Improve the test cases for planned testing
Lack of comfort on coverage of ad hoc testing	• When producing test reports combine the planned test and ad hoc test • Plan for additional planned test and ad hoc test cycles

Difficult to track the exact steps done	• Write detailed defect reports step by step manner • Document ad hoc tests after test execution
Lack of data for metrics analysis	• Plan the metrics collection for both planned tests and ad hoc tests, and testing and debugging

Buddy Testing :-

Def: A developer and tester working as buddies to help each other on testing and in understanding the specifications is called Buddy Testing

- Two team members (developer and a tester)are identified as buddies. The buddies mutually help each other, with a common goal of identifying defects early and correcting them
- This good working relationship as buddies overcome fear.
- Budding people with good working relationships yet having diverse backgrounds is a kind of a safety measure that improves the chance of detecting errors in the program very early
- Buddies should not feel mutually threatened or get a feeling of insecurity during buddy testing. They are trained on the philosophy and objective of buddy training.
- They also have to agree on the modalities and the terms of working before actually starting the testing work. They stay close together to be able to follow the agreed plan
- The Buddy can check for compliance to coding standards , appropriate variable definitions , missing code, sufficient inline code documentation , error checking.
- Buddy testing uses both white box and black box testing approaches.
- after testing generates specific review developers.
- The more specific the feedback, easier it is for the developer to fix the defects . The buddy may also suggest ideas to fix the code when pointing out an error in the work product. A buddy test may help avoid errors of omission, misunderstanding, and miscommunication by providing varied perspectives or interactive exchanges between the buddies,
- Buddy testing not only helps in finding errors in the code but also helps the tester to understand how the code is written and provides clarity on specifications. Buddy testing is normally done at the unit phase , where there are both coding and testing activities .

Pair Testing:-

Pair testing is testing done by two testers working simultaneously on the same machine to find defects in the product

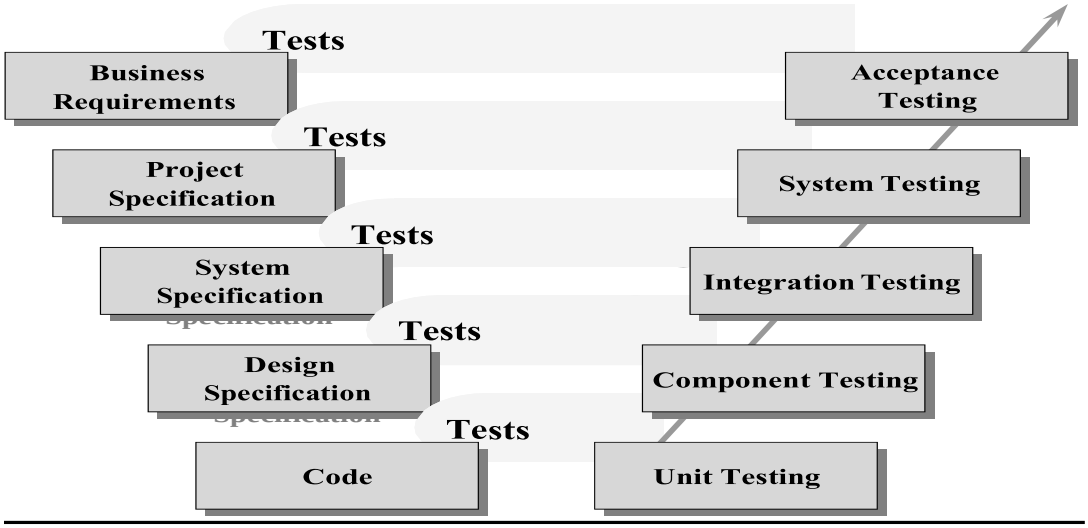
.**Example:-**

For e.g., two people traveling in a car to find a new place

- Two testers pair up to uncover new defects
- One person executes tests, and the other person observes
- Rotation of roles
- A session of one or two hours
- A senior person and a junior person make an ideal pair

- Pair testing takes advantage of the concept of the presence of one senior member can also help in pairing; this can cut down on the time spent on the learning curve of the product. It enables better training to be given to the team members; The impact of the requirements can be fully understood and explained to less experienced individuals.
- Pair testing can be done during any phase of testing. It encourages idea generating right from the requirements analysis phase, taking it forward to the design, coding and testing phases .
- Testers can pair together during the coding phase to generate various ideas to test the code and various components.
- After completion of component testing, during integration, tester can be paired to test the interfaces together. Pair testing during system testing ensures that product level defects are found and addressed.
- When the product is in new domain and not many people have the desired knowledge pair testing will be useful. Pair testing can track that vague defect that is not caught by a single person testing,
- A defect found during such pair testing may be explained better by representation of two members. Pair testing is extension of the "Pair Programming" concept used as a technique in the extreme programming model.
- Pair testing require interaction and exchange of ideas between two individuals. Team members pair with different persons during project life cycle, the entire project team can have a good understanding of each other ,

Situation when Pair Testing Becomes Ineffective:-



During pairing, teaming up individual high performers may lead to problem may be possible that during the

results. In case the pair of individuals in the team are ones who do not try to understand and respect each other testing may lead to frustration and domination. When one member is working on the computer and other is playing the role of scribe , if their speed of understanding and execution does not match, it may result in loss of attention. It may be difficult in the later stage.

Pairing up juniors with experienced members may result in the members may result in the former doing tasks that the senior may not want to do, At the end of the session, there is no accountability on who is responsible for steering the work, providing directions and delivering the results.

Exploratory Testing :-

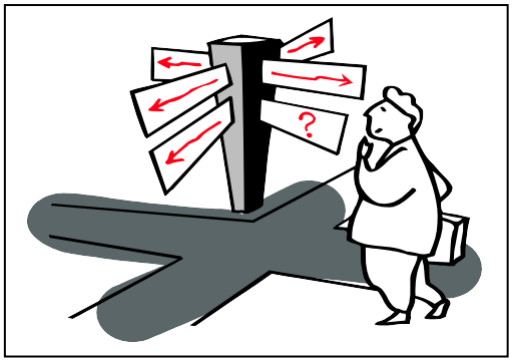
Technique used to find defects in Adhoc testing is to keep exploring the products, covering more depth and breadth. Exploratory testing tries to do that with specific objectives, tasks and plans. Exploratory testing can be done during any phase of testing.

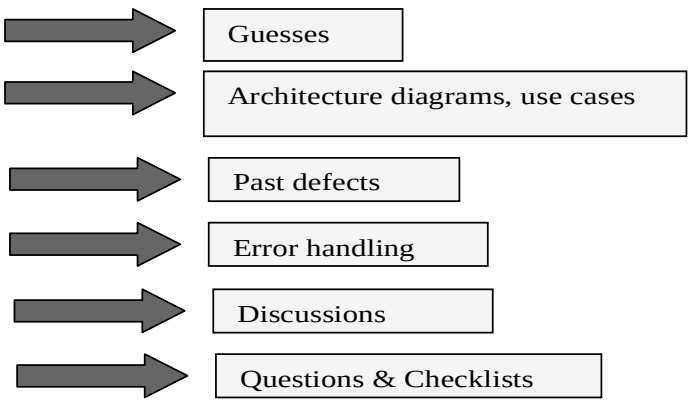
Exploratory testers may execute their test based on their past experiences in testing a similar product, or a product of similar domain, or a product in a technology area. Exploratory testing can be used to test software that is untested, unknown , or unstable. It is used when it is not obvious what the nest test should be and or when we want to go beyond the obvious tests.

Exploratory Testing Techniques:-

For e.g., driving the car in a new area. Common techniques used to reach the destination is

- Getting a map
- Asking pedestrians
- Random direction and search
- Calling up a friend
- Enquiring at gas stations
- Looking at boards / signs





Guesses are used to find the part of the program that is likely to have more errors. Because a tester would have already faced situations to test a similar product ort software. Those tests from guesses are used on the product to check for similar defects,

Architectural Diagrams and Use Cases depicts the interactions and relationships between different components and modules . us eases give an insight of the product's usage from the end users perspectives. Use case can explain a set of business events , the input requires, people involved in those events and the expected output.

Study of Past Defects studying defects reported in the previous releases helps in understanding of the error prone functionality / modules in a product development environment .

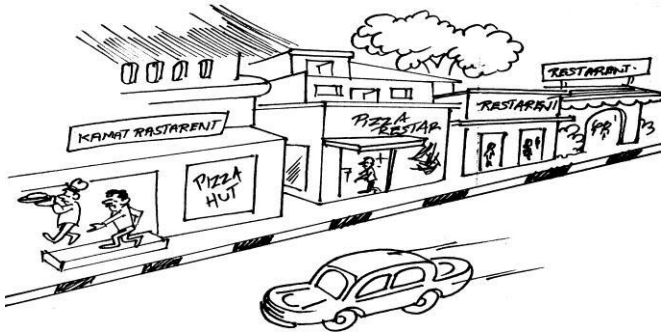
Error Handling is the product in another technique to explore. Error handling is a portion of the code which prints appropriate messages or provides appropriate action in case of failures. We can check using exploratory test for various scenarios for graceful error handling. For Example in the case of a catastrophic error, termination should be with a meaningful error message. Error Handling provides a message or corrective action in such situations. Test can be performed to simulate such situations to ensure that the products code take care of these aspects.

Discussion – Exploration may be planned based on the understanding of the system during project discussions or meetings. Plenty of information can be picked up during these meetings regarding implementation of different requirements for the products. They can be noted and used while testing.

Questionnaires and Checklists to perform the exploration. Questions like "What, When, How, Who and Why" can provide leads to explore areas in the product. To understand the implementation of functionality in a product , open-ended questions like "What does this module do", " When is it being called or used?", " how is the input processed" ," who are the users of this modules", etc.

Iterative Testing :-

For e.g., a person driving without a map trying to count the restaurants in a town

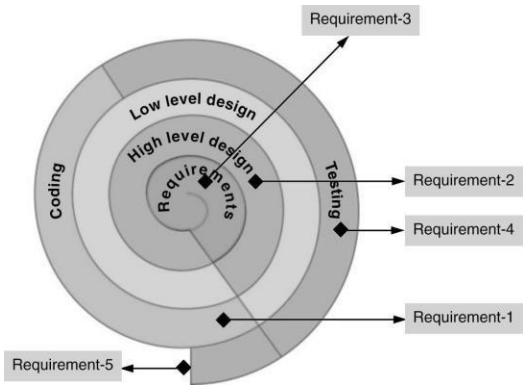


Customer will have a usable product at the end of every iteration .It is possible to stop the product development at any particular iteration and market the product as an independent entity.

Customer and Management can notice the impact of defects and the product functionality at the end of each iteration. They can take a call to proceed to the next level or not, base don the observations made in the last iterations. A test plan is created at the beginning of the first iterations and update for every subsequent iterations. This can be broadly defined the type and scope of testing to be done for each of the iterations. Developers create unit test cases to ensure that the

program developed goes through complete testing. Unit test cases are also generated from black box perspective and more more completely test the product. Regression Testing may be repeated at least every alternative iterations so that the current functionality is preserved since iterative testing involves repetitive test execution of tests that were run fro the previous iterations, it becomes a tire some exercise for the testers.

Assume that a defect was found in the second iteration and was not fixed until the fifth. There is a possibility that the defect may no longer be valid or could have become void due to revised requirements during the third, fourth and fifth iterations. In the example above the counting the number of restaurants starts from the first road visited , the results of the search can be published at the end of each iteration and released.



- Each of the requirements is at a different phase
- Testing needs to focus on the current requirement
- It should ensure that all requirements continue to work
- More re-testing effort

Agile and Extreme Testing :-

Call Attendant: Our process requires the person for whom the certificate is issued to come and sign the form.

Caller: I understand your process, but I am asking for the death certificate of my grand father.

Agile and Extreme (XP) models take the processes to the extreme to ensure that customer requirements are met in a timely manner. Customer partner with the project teams to go step by step in bringing the project to completion in a phased manner. The customer becomes part of the project team so as to clarify any doubts/questions.

Agile and Extreme (XP) methodology emphasizes the involvement of the entire team, and their interactions with each other, to produce workable software that can satisfy a given set of features. As a result of such interactions, all ideas are exchanged. Software is delivered as a small release with features being introduced in increments.

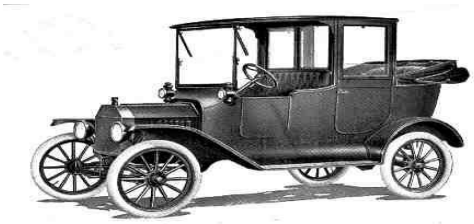
A typical XP project day start with a meeting called the Stand Up meeting . At the start of each day, the team meets to decide on the plans of actions for the day. During this meeting the team brings up any clarifications or concerns.

Policies / concepts of Agile and Extreme

- Cross boundaries

- Incremental progress – both product and process evolve in incremental way
- Travel light – least overhead
- Communicate – more focus on communication
- Write tests before coding – all unit tests run at 100%
- Make frequent small releases
- Involve customers all the time

Example for extreme testing

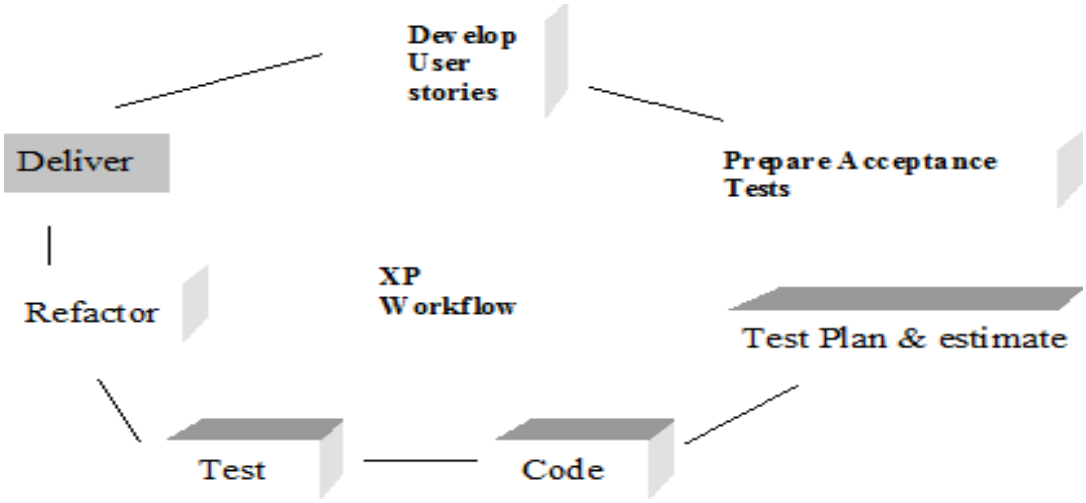


For e.g.,

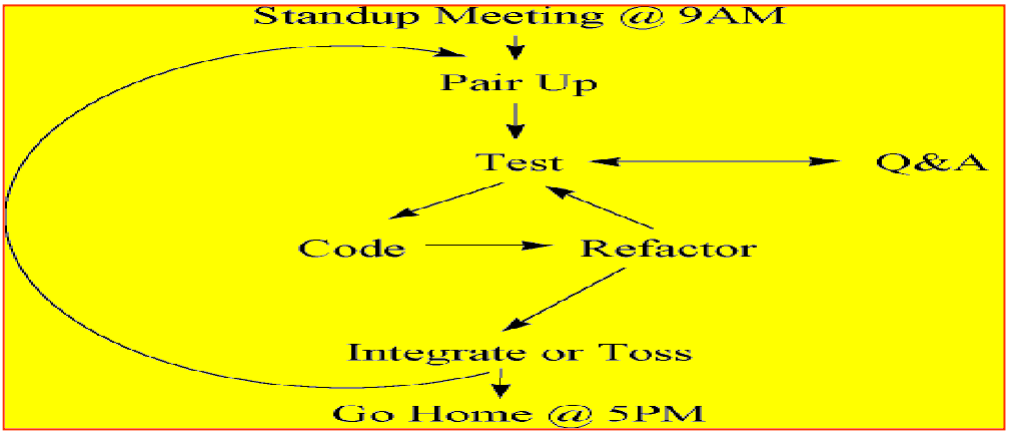
- Basic features : wheel, brake, pedal, tyres
- New features added incrementally
- Every year / every manufacturer release new models many times in a year with new features
- Thus Automobile industry keeps growing & Improving

- **Technically, driving a car using a joystick is easier , But customers are comfortable with steering wheels**

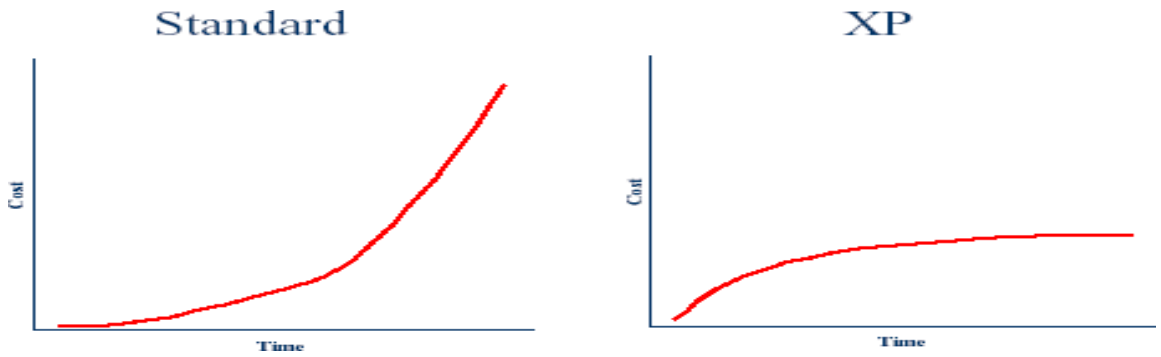
The basic steps that are carried out :-



- Develop and Understand User Story
- Prepare acceptance tests
- Test Plan and Estimation
- Code
- Test
- Refactor
- Automate
- Accepted and Delivered



Cost of Change:-



Defect Seeding:-

Def: Defect seeding is a method of intentionally introducing defects into a product to check the rate of detection and residual defects.

Error Seeding is also known as *Debugging* . It acts as a reliability measure for the release of the product. Usually one group members in the project injects the defects while an other group tests to remove them. The purpose of this exercise is while finding the known seeded defects, the unseeded/ un earthed defects may also be uncovered . Defects that are seeded are similar to real defects. Defects that can be seeded may vary from sever or critical defects to cosmetic errors. Defect Seeding may act as a guide to check the efficiency of the inspection or testing process. It serves as a confidence measure to know the percentage of defects removal rates. It acts as a measure to estimate the number of defect yet to be discovered in the system.

Defects that can be seeded may vary from severe or critical defects to cosmetic errors.

- For example : a team seeds 20 defects, and testing finds out 12 seeded defects and 25 other defects

Total latent defects = (defects seeded / defects seeded found) * Other defects found

- 20 / 12 * 25 = 41.67 = 42

Based on the above calculation , the number of estimated defects yet to be found is 42.

When a group knows that there are seeded defects in the system it acts as a challenge for them to find as many of them as possible. It adds a new energy into their testing .in case of manual testing, defects are seeded before the start of the testing process. When the tests are automated, defects can be seeded any time .

It may be useful to look at the following issues on defect seeding as well.

1. Care should be taken during the defect seeding process to ensure that all the seeded defects are removed before the release of the product.
2. The code should be written in such a way that the errors introduced can be identified easily, Minimum number of lines should be added to seed defects so that the effort involved in removal becomes reduced.

- It is necessary to estimate the effort required to clean up the seeded defect along with effort for identification. Effort may also be needed to fix the real defects

ALPHA, BETA TESTS

- Goal : allow users to evaluate the software in terms of clients expectations and goals.
- The acceptance tests must be planned carefully with input from the client/users. Acceptance test cases are based on requirements.
- The user manual is an additional source for test cases. System test cases may be reused.
- The software must run under real-world conditions on operational hardware and software.
- For continuous systems the software should be run at least through a 25-hour test cycle.
- Development organizations will often receive their final payment when acceptance tests have been passed.
- Acceptance tests must be rehearsed by the developers/testers. There should be no signs of unprofessional behavior or lack of preparation. Clients do not appreciate surprises. They should be provided with documents and other material to help them participate in the acceptance testing process, and to evaluate the results
- After acceptance testing the client will point out to the developers which requirement have/have not been satisfied. Some requirements may be deleted, modified, or added due to changing needs.
- If the client is satisfied that the software is usable and reliable, and they give their approval, then the next step is to install the system at the client's site. If the client's site conditions are different from that of the developers, the developers must set up the system so that it can interface with client software and hardware. Retesting may have to be done to insure that the software works as required in the client's environment. This is called installation test.
- If the software has been developed for the mass market , then testing it for individual clients/users is not practical or even possible in most cases. Very often this type of software undergoes
- two stages of acceptance test.
    - alpha test. □ test takes place at the developer's site. A cross-section of potential users and members of the developer's organization are invited to use the software. Developers observe the users and note problems.

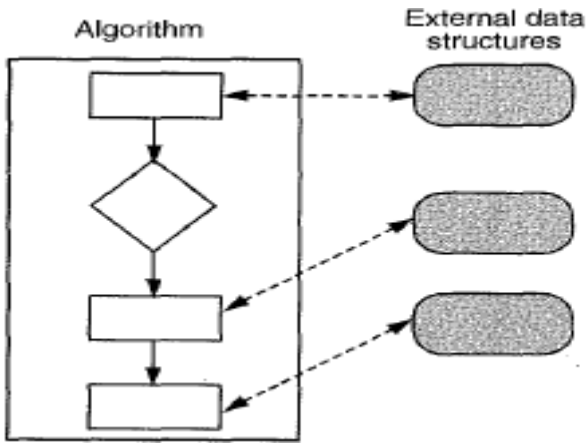
- Beta test□ sends the software to a cross-section of users who install it and use it under real world working conditions. The users send records of problems with the software to the development organization where the defects are repaired sometimes in time for the current release. In many cases the repairs are delayed until the next release.

TESTING OO SYSTEMS :-

In procedure-oriented languages

Algorithms +Data Structures =Programs.

These programming languages were algorithm-centric in that they viewed the program as being driven by an algorithm that traced its execution from start to finish, as shown in Figure Data was an external entity that was operated upon by the algorithm.



Conventional algorithm centric programming languages.

Fundamentally, this type of programming languages was characterized by

1.Data being considered as separate from the operations or program and

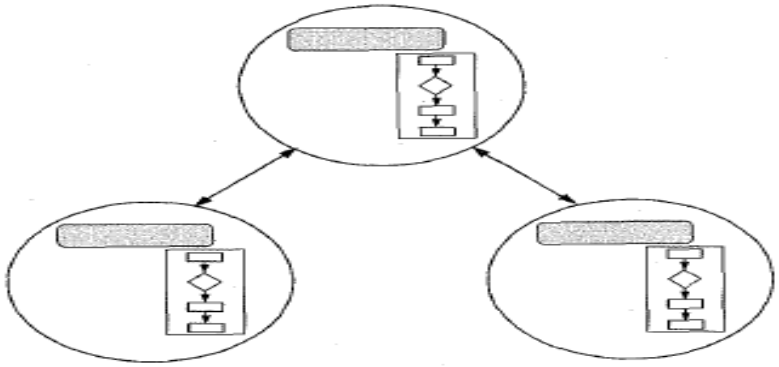
2.Algorithm being the driver, with data being subsidiary to the algorithm.

In OO languages

There are two fundamental paradigm shifts in OOlanguages and programming:

First□ the language is data- or object-centric.

Second□ The data and the methods that operate on the data go together as one indivisible unit.



Object centric language-algorithm and data tightly coupled

Some of the basic concepts of OO systems are relevant for testing

Classes :Classes form the fundamental building blocks for OO systems. A class is a representation of a real-life object. Each class (or the real-life object it represents) is made up of attributes or variables and methods that operate on the variables.

```
Class rectangle
{
private int length, breadth;
public:
new (float length, .float. breadth)
(
this->length = length;
this->breadth = breadth;
float area ()
{
return (length*breadth1;
return (2*(length+breadth))
}
} ;
```

Objects

Objects are the dynamic instantiation of a class. Multiple objects are instantiated using a given (static) class definition. Such specific instantiations are done using a constructor function.

Constructor

A constructor function brings to life an instance of the class. Each class can have more than one constructor function. Depending on the parameters passed or the signature of the function, the right constructor is called.

Encapsulation

Encapsulation provides the right level of abstraction about the variables and methods to the outside world.

Polymorphism

This property of two methods-in different classes-having the same name but performing different functions is called polymorphism.

Inheritance

Inheritance enables the derivation of one class from another without losing sight of the common features. This ability is called inheritance. The original class is called the parent class (or super-class) and the new class is called a child class (or derived class, or sub-class). Inheritance allows objects (or at least parts of the object) to be reused. A derived class inherits the properties of the parent class-in fact, of all the parent classes, as there can be a hierarchy of classes. Thus, for those properties of the parent class that are inherited and used as is, the development and
testing costs can be saved.

DIFFERENCES IN OO TESTING

From a testing perspective, the implication is that testing an oo system should tightly integrate data and algorithms .. The dichotomy between data and algorithm that drove the types of testing m procedure-oriented languages has to be broken. Testing OO systems broadly covers the following topics.

1. Unit testing a class
2. Putting classes to work together (integration testing of classes)
3. System testing
4. Regression testing
5. Tools for testing OO systems

Unit Testing a Set of Classes

Classes are the building blocks for an entire OO system.

Why classes have to be tested individually first:

reasons:

1. A class is intended for heavy reuse. A residual defect in a class can therefore, potentially affect every instance of reuse.

2. Many defects get introduced at the time a class (that is, its attributes and methods) gets defined. A delay in catching these defect makes them go into the clients of these classes. Thus, the fix for the defect would have to be reflected in multiple places, giving rise to inconsistencies.

3. A class may have different features; different clients of the class may pick up different pieces of the class. No one single client may use all the pieces of the class. Thus, unless the class is tested as a unit first, there may be pieces of a class that may never get tested.

4. A class is a combination of data and methods. If the data and methods do not work in sync at a unit test level, it may cause defects that are potentially very difficult to narrow down later on.

5. Unlike procedural language building blocks, an OO system has special features like inheritance, which puts more "context" into the building blocks. Thus, unless the building blocks are thoroughly tested stand-alone, defects arising out of these contexts may surface, magnified many times, later in the cycle.

Conventional methods that apply to testing classes

Some of the methods for unit testing that we have discussed earlier apply directly to testing classes. For example:

1. Every class has certain variables. The techniques of boundary value analysis and equivalence partitioning discussed in black box testing can be applied to make sure the most effective test data is used to find as many defects as possible.
2. As mentioned earlier, not all methods are exercised by all the clients, The methods of function coverage that were discussed in white box testing can be used to ensure that every method (function) is exercised.
3. Every class will have methods that have procedural logic. The techniques of condition coverage, branch coverage, code complexity, and so on that we discussed in white box testing can be used to make sure as many branches and conditions are covered as possible and to increase the maintainability of the code.
4. Since a class is meant to be instantiated multiple times by different clients, the various techniques of stress testing.

Integration testing

Since OOsystems are ·designed to be made up of a number of smaller components or classes that are meant to be reused (with necessary redefinitions), testing that classes work together becomes the next step, once the basic classes themselves are found to be tested thoroughly. In the case of OO systems, because of the emphasis on reuse and classes, testing this integration unit becomes crucial. in an OO system, the way in which the various classes communicate with each other is through messages. A message of the format

<instance name>.<method name>.<variables>

calls the method of the specified name, in the named instance, or object (of the appropriate class) with the appropriate variables.
Methods with the same name perform different functions polymorphism. From a testing perspective, polymorphism is especially challenging because it defies the conventional definition of code coverage and static inspection of code.

The various methods of integration

- top-down
- bottom-up
- big bang

System Testing and Interoperability of OO Systems

Object oriented systems are by design meant to be built using smaller reusable components (i.e. the classes). Some of the reasons for this added importance are:
1. A class may have different parts, not all of which are used at the same time. When different clients start using a class, they may be using different parts of a class and this may introduce defects at a later (system testing) phase
2. Different classes may be combined together by a client and this combination may lead to new defects that are hitherto uncovered.
3. An instantiated object may not free all its allocated resource. Thus causing memory leaks and such related problems, which shows up only in the system testing phase

Regression Testing of OO Systems

- Taking the discussion of integration testing further, regression testing becomes very crucial for OO systems. As a result of the heavy reliance of OO systems on reusable components, changes to an one component could have potentially unintended side-effects on the clients that use the component.
- Hence, frequent integration and regression runs become very essential for testing OO systems. Also, because of the cascaded effects of changes resulting from properties like inheritance, it makes sense to catch the defects as early as possible.

Tools for Testing of OO Systems

There are several tools that aid in testing OO systems. Some of these are

1. Use cases
2. Class diagrams
3. Sequence diagrams
4. Activity Diagrams
5.State charts

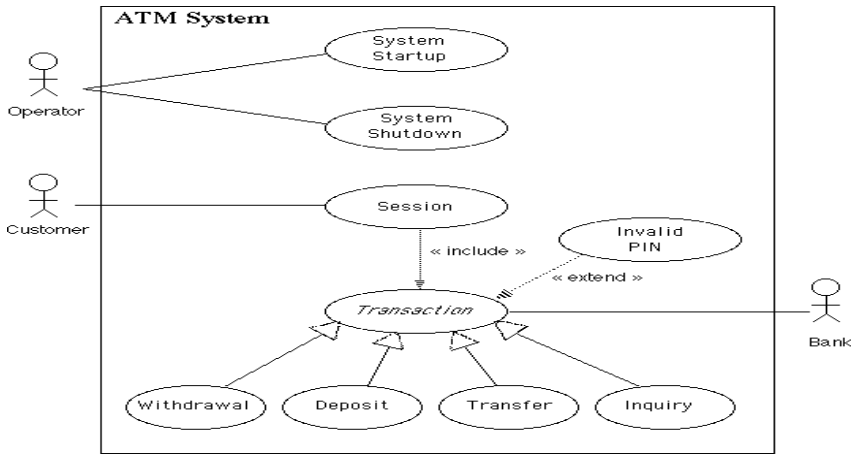
Use cases

Use cases represent the various tasks that a user will perform when interacting with the system. Use cases go into the details of the specific steps that the user will go through m accomplishing each task and the systemresponses for each steps. This fits in place for the object oriented paradigm, as the tasks and responses are akin to messages passed to the various objects.

Class diagram

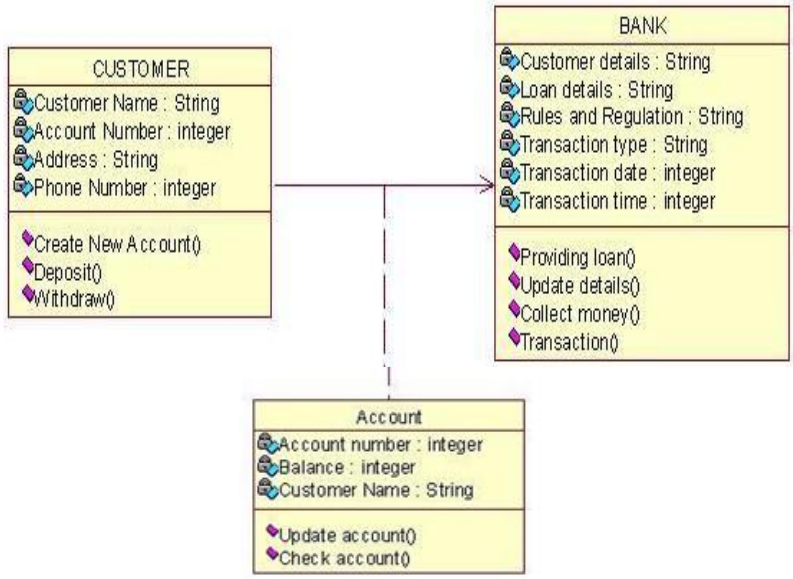
A class diagram is useful for testing in several ways.

1.It identifies the elements of a class and hence enables the identification of the boundary value analysis, equivalence partitioning, and such tests.



2.The associations help in identifying tests for referential integrity constraints across classes.

3.Generalizations help in identifying class hierarchies and thus help in planning incremental class testing as and when new variables and methods are introduced in child classes.



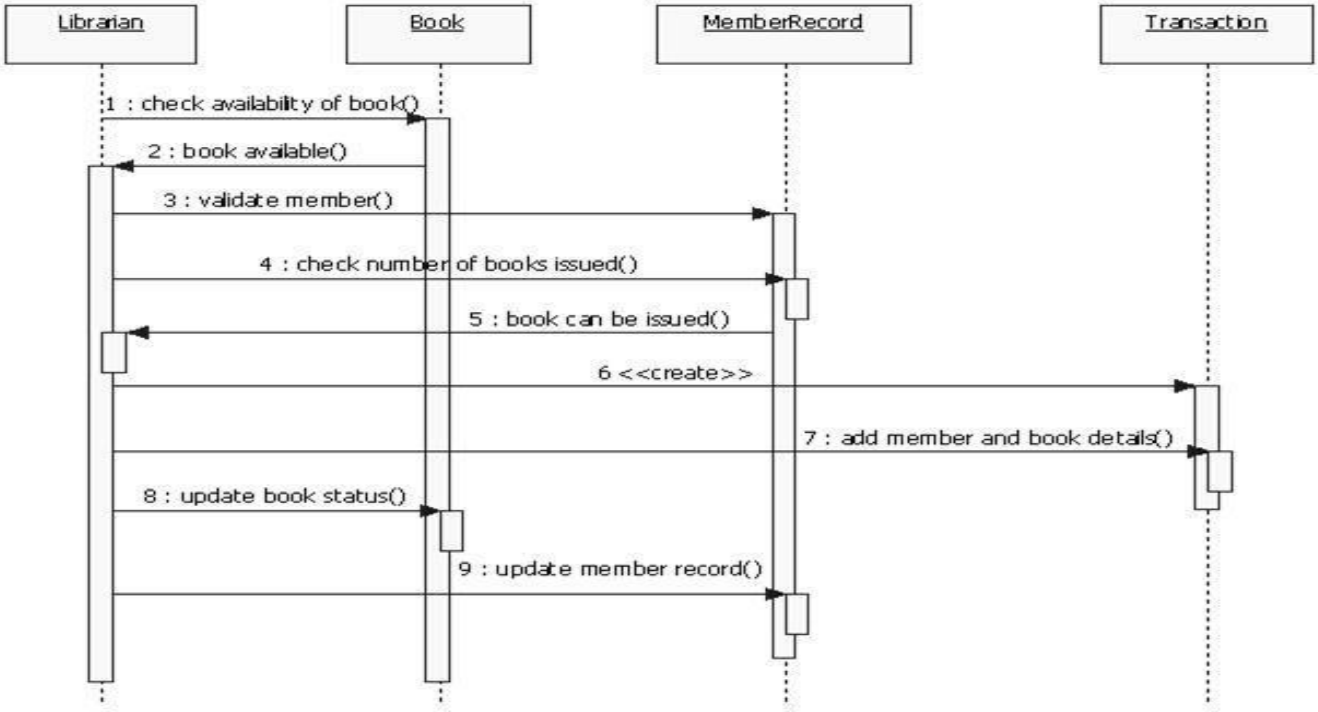
A sequence diagram

A sequence diagram helps in testing by

1. Identifying temporal end-to-end messages.Tracing the intermediate points in an end-to-end transaction, thereby enabling easier narrowing down of problems.
2. Providing for several typical message-calling sequences like blocking call, non-blocking call, and so on.

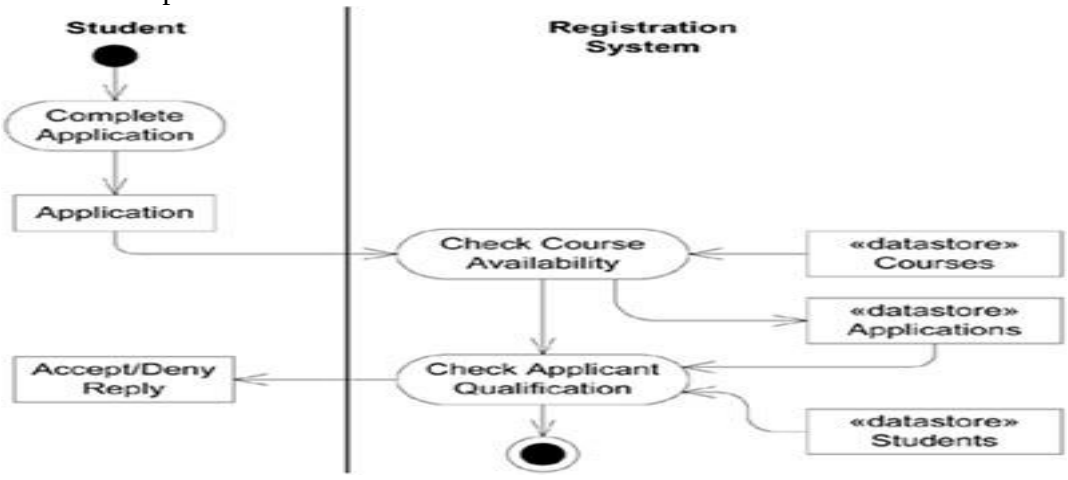
Sequence diagrams also have their limitations for testing-complex interactions become messy, if not impossible; to represent; dynamic binding cannot be represented easily.

Ex: Borrow Books in Library Information system



Activity diagram

While a sequence diagram looks at the sequence of messages, an activity diagram depicts the sequence of activities that take place. It is used for modeling a typical work flow in an application and brings out the elements of interaction between manual and automated processes. Since an activity diagram represents a sequence of activities, it is very similar to a flow chart and has parallels to most of the elements of a conventional flow chart.



Given that an activity diagram represents control flow, its relevance for testing comes from

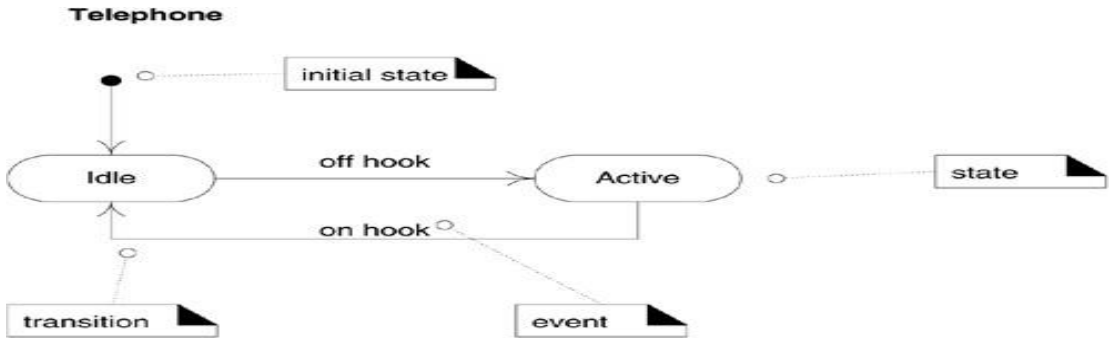
1. The ability to derive various paths through execution. Similar to the flow graph discussed in white box testing, an

activity diagram can be used to arrive at the code complexity and independent paths through a program code.

2. Ability to identify the possible message flows between an activity and an object, thereby making the message based, testing more robust and effective.

State Chart Diagram

When an object can be modeled as a state machine, then the techniques of state-based testing, in black box testing can be directly applied.



USABILITY AND ACCESSIBILITY TESTING

Usability Testing

Testing that validates ease of use , speed and aesthetics of the product from the user's point of view

Characteristics

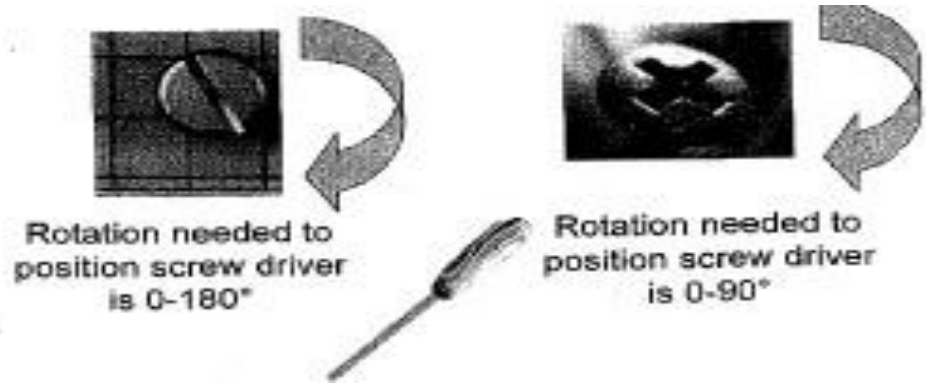
1. Usability testing tests the product from the users' point of view.
2. Usability testing is for checking the product to see if it is easy to use for the various categories of users.
3. Usability testing is a process to identify discrepancies between the user interface of the product and the human user requirements, in terms of the pleasantness and aesthetics aspects.

Conclusion

A view expressed by one user of the product may not be the view of another.

- easy for one user -->may not be easy for another
- fast (interms of say, response time) □e slow for another user
- beautiful by someone □ look ugly to another.

APPROACH TO USABILITY



For example , when a Philips (or a star) screwdriver was invented, it saved only few milliseconds per operation to adjust the screwdriver to the angle of the screw compared to a flat screwdriver.

People best suited to perform usability testing :

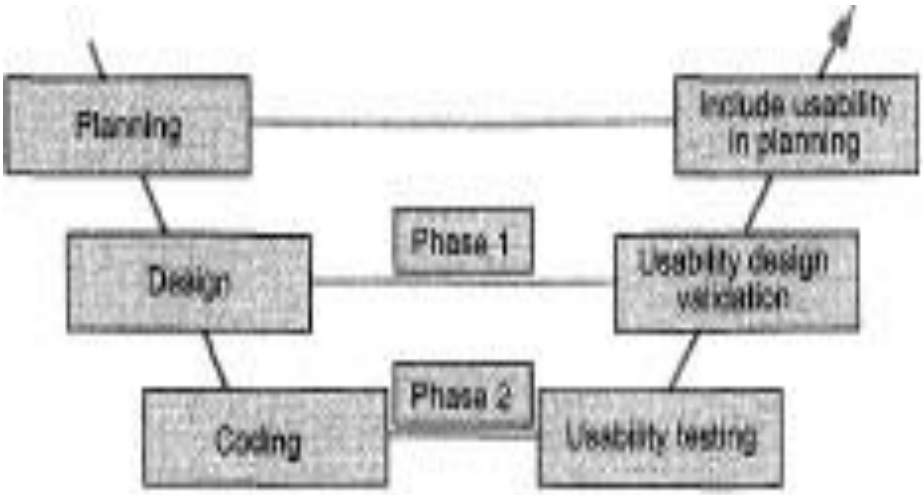
- representatives of the actual user segments who would be using the product
- People who are new to the product

WHEN TO DO USABILITY TESTING?

There are 2 phases in usability testing.

Phase 1 : Design Validation

Phase 2 : Usability testing done as a part of component and integration testing phases of a test cycle



Usability design is verified through several means, some of them are

- Style sheets
- Screen prototypes
- Paper designs
- Layout design

Web application interfaces are designed before designing functionality. That gives adequate time for doing two phases of usability testing.

Client Application	**Web Application**
Step1 : Design for functionality	**Step1 :** Design for User Interface
Step2 : Perform Coding for functionality	**Step2 :** Performa Coding for User Interface
Step3 : Design for User Interface	**Step3 :** Test User Interface (Phase 1)
Step4 : Perform coding for User Interface	**Step4 :** Design for Functionality
Step5 : Integrate user interface with functionality	**Step5 :** Perform coding for functionality
Step6 : Test UI along with functionality (Phase 1 & 2)	**Step6 :** Test UI along with functionality (Phase 2)

Development and Testing of Client Applications and Web Application

HOW TO ACHIEVE USABILITY?

Usability is a habit and a behavior, just like humans, the products are expected to behave differently and correctly with different users and to their expectations.

Checklists are created and verified during usability testing.

1. Do users complete the assigned tasks/operations successfully?
2. If so, how much time do they take to complete the tasks/operations?
3. Is the response from the product fast enough to satisfy them?
4. Where did the users get struck? What problems do they have?
5. Where do they get confused? Were they able to continue on their own? What helped them to continue?

QUALITY FACTORS FOR USABILITY

- **Comprehensibility** – when features and components are grouped in a product , they should be based on user terminologies not technology or implementation
- **Consistency** – A Product needs to be consistent with any applicable standards , platform look and feel , base infrastructure and earlier versions of the same product.
- **Navigation –** This helps in determining how easy it is to select the different operations of the product
- **Responsiveness-** How fast the product responds to the user request .

AESTHETICS TESTING

It ensures the product is pleasing to the eye.

Ex: A pleasant look for menus, pleasing colors, nice icons, and so on can improve aesthetics. It is generally considered as gold plating, which is not right.

ACCESSIBILITY TESTING

Verifying the product usability for physically challenged users

Accessibility to the product can be provided by two means.

1. Making use of accessibility features provided by the underlying infrastructure (for example, operating system), called basic accessibility, and
2. Providing accessibility in the product through standards and guidelines, called product accessibility.

I) Basic Accessibility

1) Keyboard accessibility
   - Sticky keys(ctrl, alt, del -->login , logout)
   - Filter keys
   - Toggle key sound
   - Sound keys
   - Arrow keys to control mouse
   - Narrator (text to audio)

2) Screen accessibility

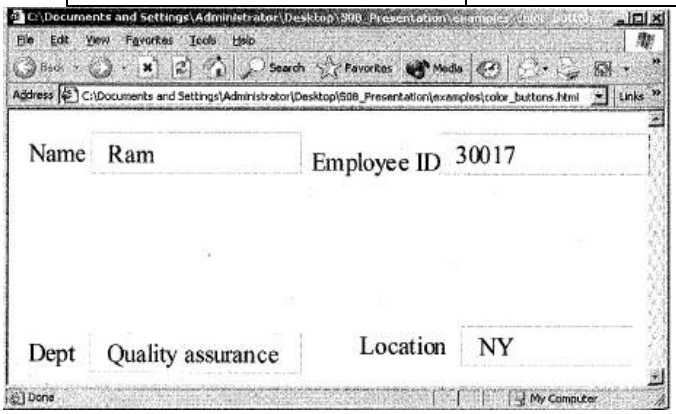
- Visual sound
- Enabling captions for multimedia

3) Other accessibility features

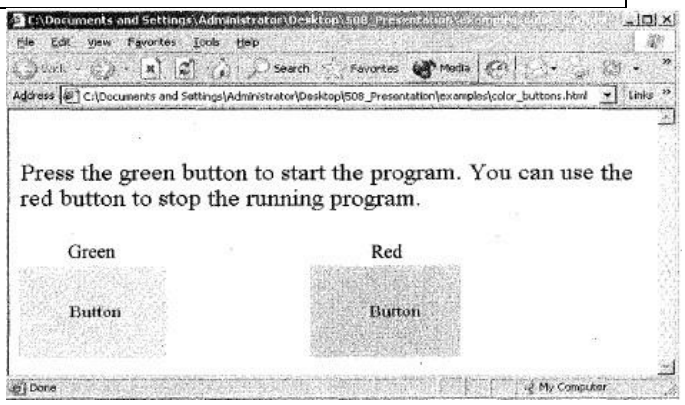
- Soft keyboard
- Easy reading with high contrast

II) Product Accessibility

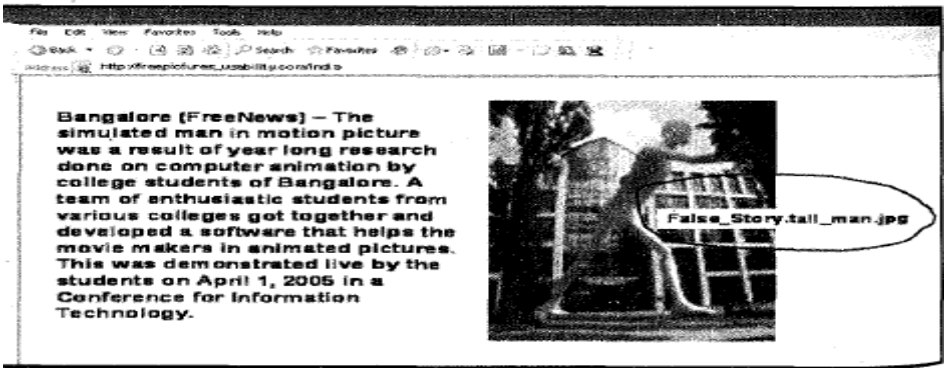
Sample Requirement 1	Text equivalent have to be provided for audio , video & picture images
Sample Requirement 2	Documents and fields should be organized (style sheets)
Sample Requirement 3	UI should be designed so that all info conveyed with color is also without color
Sample Requirement 4	Reduce the fictker rate , speed of moving text avoid flashes and blinking text
Sample Requirement 5	Reduce physical movements requirements for the user when designing the interface and allow adequate time for user response



Screen with 4 fields in the corner



Color as method of identification



Sample website with picture along with web site equivalent

TOOLS FOR USABILITY

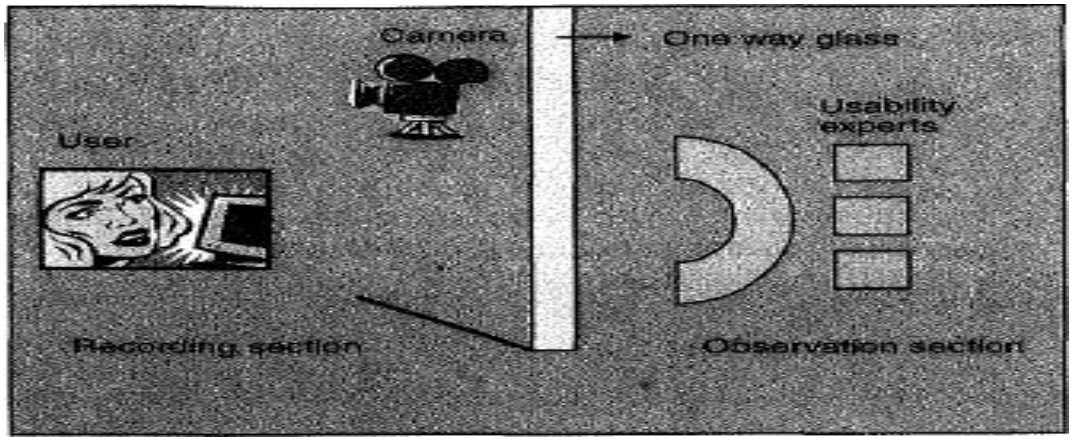
- **Jaws**
- **HTML Validator**
- **Style Sheet Validator**
- **Magnifier (enlarge the items)**

- **Soft Keyboard(display keyboard template on the screen)**

USABILITY LAB SETUP

This lab has 2 sections – recording sections and observation section.

- In the recording section of the lab - A user is requested to come to the lab with a prefixed set of operations that are to be performed with the product
- In the observations section of the lab - it is one way glass – the experts can see the user but the user cannot see the experts . some usability experts sit and observe the user for body language and associate the defects with the screens and events that caused it.



Configuration Testing (or refer page no 21 of unit 3 notes)

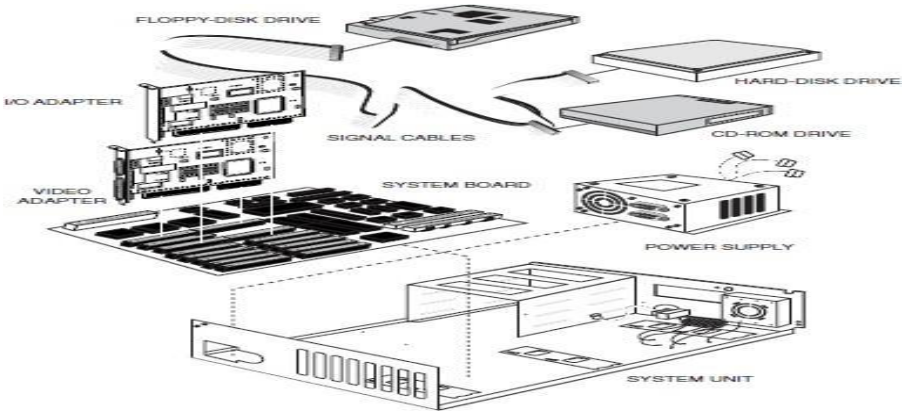
Configuration testing is the process of checking the operation of the software you're testing with all the various types of hardware.

Ex : Configuration bug

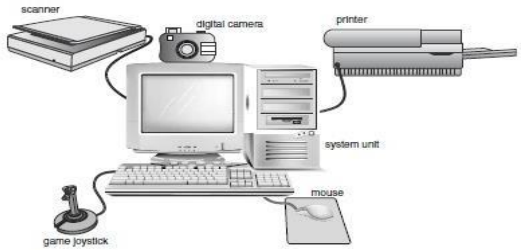
1. if your greeting card program works fine with laser printers but not with inkjet printers.
2. The hardware device or its device drivers may have a bug that only your software reveals. Maybe your software is the only one that uses a unique display card setting. When your software is run with a specific video card, the PC crashes.
3. if a specific printer driver always defaulted to draft mode and your photo printing software had to set it to high-quality every time it printed.

The PC.

Components - system boards, component cards, and other internal devices such as disk drives, CD-ROM drives, video, sound, modem, and network cards



monitors, cameras, joysticks, and other devices that plug into your system and operate externally to use the PC



Interfaces. The components and peripherals plug into your PC through various types of interface connectors .These interfaces can be internal or external to the PC. Typical names for them are ISA, PCI, USB, PS/2, RS/232, and Firewire. There are so many different possibilities that hardware manufacturers will often create the same peripheral with different interfaces. It's possible to buy the exact same mouse in three different configurations!

• **Options and memory.** Many components and peripherals can be purchased with different hardware options and memory sizes. Printers can be upgraded to support extra fonts or accept more memory to speed up printing. Graphics cards with more memory can support additional colors and higher resolutions.

• **Device Drivers.** All components and peripherals communicate with the operating system and the software applications through low-level software called device drivers. These drivers are often provided by the hardware device manufacturer and are installed when you set up the hardware. Although technically they are software, for testing purposes they are considered part of the hardware configuration.

configuration testing - the general process

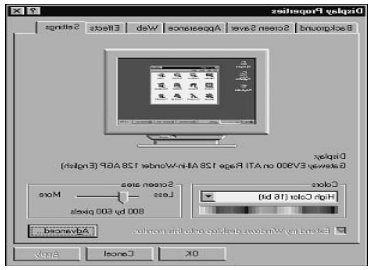
1. **Decide the Types of Hardware You'll Need**
   Put your software disk on a table and ask yourself what hardware pieces you need to put together to make it work.

2. **Decide What Hardware Brands, Models, and Device Drivers Are Available**

Decide what device drivers you're going to test with. Your options are usually the drivers included with the operating system, the drivers included with the device, or the latest drivers available on the hardware or operating system company's Web site.

3. **Decide Which Hardware Features, Modes, and Options Are Possible**

Color printers can print in black and white or color, they can print in different quality modes, and can have settings for printing photos or text. Display cards, as shown in Figure, can have different color settings and screen resolutions.



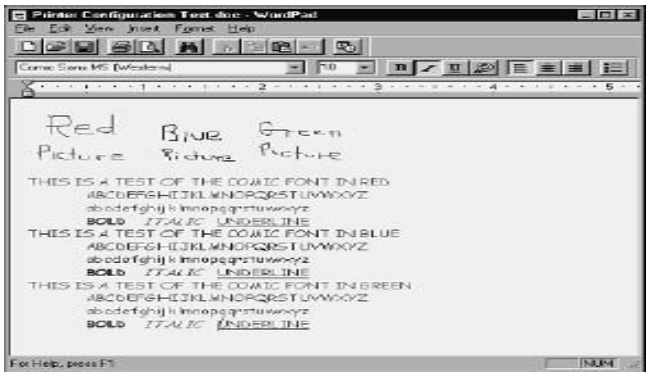
4. **Pare Down the Identified Hardware Configurations to a Manageable Set**

reduce the thousands of potential configurations into the ones that you're going to test.
put all the configuration information into a spreadsheet with columns for the manufacturer, model, driver versions, and options.

Popularity (1=most, 10=least)	Type (Laser / InkJet)	Age (years)	Manufacturer	Model	Device Driver Version	Options	Options
1	Laser	3	HAL Printers	LDIY2000	1.0	B/W	Draft Quality
5	InkJet	1	HAL Printers	IJDIY2000	1.0a	Color B/W	Draft Quality Draft Quality
5	InkJet	1	HAL Printers	IJDIY2000	2.0	Color B/W	Art Photo Draft Quality
10	Laser	5	OkeeDohKee	LJ100	1.5	B/W	100dpi 200dpi 300dpi
2	InkJet	2	OkeeDohKee	EasyPrint	1.0	Auto	600dpi

5. Identify Your Software's Unique Features That Work with the Hardware Configurations

For example, if you're testing a word processor such as WordPad), you don't need to test the file save and load feature in each configuration. File saving and loading has nothing to do with printing. A good test would be to create a document that contains different fonts, point sizes, colors, embedded pictures, and so on. You would then attempt to print this document on each chosen printer configuration



6. Design the Test Cases to Run on Each Configuration

1. Select and set up the next test configuration from the list.
2. Start the software.
3. Load in the file configtest.doc.
4. Confirm that the displayed file is correct.
5. Print the document.
6. Confirm that there are no error messages and that the printed document matches the standard.
7. Log any discrepancies as a bug

7. Execute the Tests on Each Configuration

run the test cases and carefully log and report your results to your team, and to the ardware manufacturers if necessary. You'll need to work closely with the programmers and white-box testers to isolate the cause and decide if the bugs you find are due to your software or to the hardware. If the bug is specific to the hardware, consult the manufacturer's Web site for information on reporting problems to them. Be sure to identify yourself as a software tester and what company you work for.

8. Rerun the Tests Until the Results Satisfy Your Team

It's difficult to run configuration testing the entire course of a project. Initially a few configurations might be tried, then a full test pass, then smaller and smaller sets to confirm bug fixes. Eventually you will get to a point where there

are no known bugs or to where the bugs that still exist are in uncommon or unlikely test configurations at that point you can call your configuration testing complete.

Compatibility Testing (Refer Unit 2 Notes)
Documentation Testing (Refer Unit 2 Notes)

Website Testing

- Web Page Fundamentals
- Black-Box Testing
- Gray-Box Testing
- White-Box Testing
- Configuration and Compatibility Testing
  - Usability Testing

Web Page Fundamentals

Internet Web pages are just documents of text, pictures, sounds, video, and hyperlinks
Web page features.

- Text of different sizes, fonts, and colors (okay, you can't see the colors in this book)
- Graphics and photos
- Hyperlinked text and graphics
- Varying advertisements
- Drop-down selection boxes
- Fields in which the users can enter data

features that make the Web site much more complex:

- Customizable layout that allows users to change where information is positioned
- onscreen
- Customizable content that allows users to select what news and information they want to see
- Dynamic drop-down selection boxes
- Dynamically changing text
- Dynamic layout and optional information based on screen resolution
- Compatibility with different Web browsers, browser versions, and hardware and software platforms
- Lots of hidden formatting, tagging, and embedded information that enhances the Web page's usability

Testing Techniques apply to Web page testing

- basic white-box and black-box techniques
- configuration and compatibility testing
- usability testing

1) Black-Box Testing

screen image of Apple's Web site, www.apple.com, a fairly straightforward and typical Web site. It has all the basic elements—text, graphics, hyperlinks to other pages on the site, and hyperlinks to other Web sites.

The easiest place to start is by treating the Web page or the entire Web site as a black box What would you test? What would you choose not to test?

When testing a Web site, you first should create a state table , treating each page as a different state with the hyperlinks as the lines connecting them. A completed state map will give you a better view of the overall task.

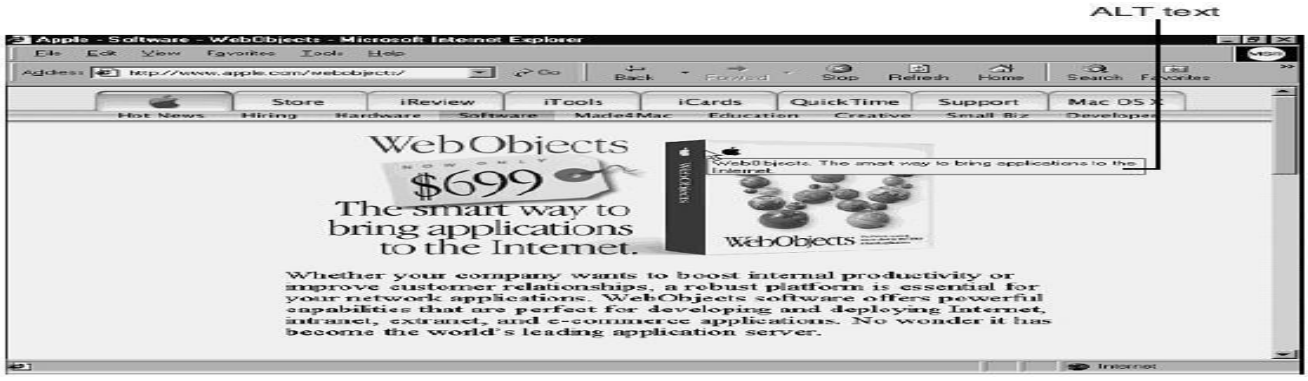
Web pages are made up of just text, graphics, links, and the occasional form. Testing them isn't difficult.

Text

Check the audience level,

- the terminology,
- the content and subject matter,
- the accuracy—especially of information that can become outdated—and
- always check spelling.
- each page has a correct title

An often overlooked type of text is called ALT text, for ALTernate text. Figure shows an example of ALT text. When a user puts the mouse cursor over a graphic on the page he gets a pop-up description of what the graphic represents. Web browsers that don't display graphics use ALT text. Also, with ALT text blind users can use graphically rich Web sites—an audible reader interprets the ALT text and reads it out through the computer's speakers.



Hyperlinks

Links can be tied to text or graphics. Each link should be checked to make sure that it jumps to the correct destination and opens in the correct window.

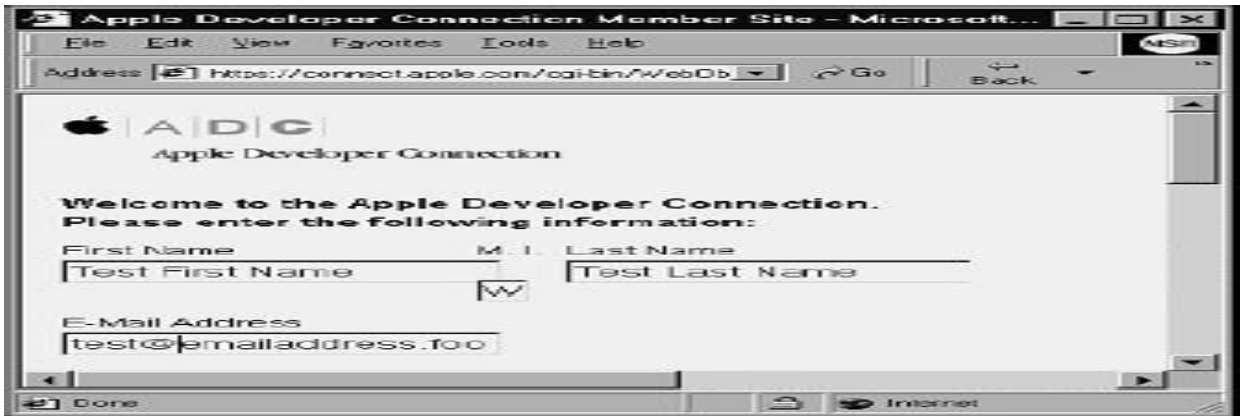
Check

- Text links are usually underlined, and the mouse pointer should change to a hand pointer when it's over any kind of hyperlink—text or graphic.
- Look for orphan pages, which are part of the Web site but can't be accessed through a hyperlink
- do all graphics load and display properly? If a graphic is missing or is incorrectly named, it won't load and the Web page will display an error where the graphic was to be placed.
- If text and graphics are intermixed on the page, make sure that the text wraps properly around the graphics. Try resizing the browser's window to see if strange wrapping occurs around the graphic.
- How's the performance of loading the page? Are there so many graphics on the page, resulting in a large amount of data to be transferred and displayed, that the Web site's performance is too slow?
- What if it's displayed over a slow dial-up modem connection on a poor-quality phone line?

If a graphic can't load onto a Web page, an error box is put in its location

Forms

Forms are the text boxes, list boxes, and other fields for entering or selecting information on a Web page. In the example a signup form for potential Mac developers. There are fields for entering your first name, middle initial, last name, and email address.



Make sure your Web site's form fields are positioned properly. Notice in this Apple Developer signup form that the middle initial (M.I.) field is misplaced.

Gray-Box Testing

graybox testing, is a mixture of the black box & white box testing —hence the name. You still test the software as a black-box, but you supplement the work by taking a peek (not a full look, as in white-box testing) at what makes the software work. Web pages provide themselves nicely to gray-box testing.

Most Web pages are built with HTML (Hypertext Markup Language). Listing shows a few lines of the HTML used to create the Web page

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
….

HTML and Web pages can be tested as a gray box because HTML isn't a programming language it's a markup language.

In the early days of word processors, you couldn't just select text and make it bold or italic. You had to embed markups, sometimes called field tags, in the text. For example, to create the bolded phrase

This is bold text.

you would enter something such as this into your word processor:
[begin bold]This is bold text.[end bold]

HTML works the same way. To create the line in HTML you would enter
This is bold text.
HTML has evolved to where it now has hundreds of different field tags and options, as evidenced by the HTML

2) White-Box Testing

Web page also has customizable and dynamic changing content. Remember, HTML isn't a programming language— it's merely a tagging system for text and graphics. To create these extra dynamic features requires the HTML to be

supplemented with programming code that can execute and follow decision paths popular web programming languages: DHTML, Java, JavaScript, ActiveX, VBScript, Perl, CGI, ASP, and XML.

the important bugs that you have some knowledge of the Web site's system structure and programming:

• **Dynamic Content.** Dynamic content is graphics and text that changes based on certain conditions—for example, the time of day, the user's preferences, or specific user actions.

Supported by

- Client side scripting :It's possible that the programming for the content is done in a simple scripting language such as JavaScript and is embedded within the HTML. apply gray-box testing techniques when you examine the script and view the HTML.

- server-side scripting : For efficiency, most dynamic content programming is located on the Web site's server; and would require to have access to the Web server to view the code.

• **Database-Driven Web Pages.** Many e-commerce Web pages that show catalogs or inventories are database driven. The HTML provides a simple layout for the Web content and then pictures, text descriptions, pricing information, and so on are pulled from a database on the Web site's server and plugged into the pages.

• **Programmatically Created Web Pages.** Many Web pages, especially ones with dynamic content, are programmatically generated—that is, the HTML and possibly even the programming is created by software. A Web page designer may type entries in a database and drag and drop elements in a layout program, press a button, and out comes the HTML that displays a Web page. If you're testing such a system, you have to check that the HTML it creates is what the designer expects.

• **Server Performance and Loading**. Popular Web sites might receive millions of individual hits a day. Each one requires a download of data from the Web site's server to the browser's computer. If you wanted to test a system for performance and loading, you'd have to find a way to simulate the millions of connections and downloads.

• **Security**. Web site security issues are always in the news as hackers try new and different ways to gain access to a Web site's internal data. Financial, medical, and other Web sites that contain personal data are especially at risk and require intimate knowledge of server technology to test them for proper security.

3) Configuration and Compatibility Testing

Configuration testing is the process of checking the operation of your software with various types of hardware and software platforms and their different settings.

Compatibility testing is checking your software's operation with other software.

Web pages are perfect examples of where you can apply this type of testing .Assume that you have a Web site to test. You need to think about what the possible hardware and software configurations might be that could affect the operation or appearance of the site.

Here's a list to consider:

• **Hardware Platform.** Is it a Mac, PC, a TV browsing device, a hand-held, or a wristwatch? Each hardware device has its own operating system, screen layout, communications software, and so on. Each can affect how the Web site appears onscreen.

• **Browser Software and Version.** There are many different Web browsers and browser

versions. Some run on only one type of hardware platform, others run on multiple platforms. Some examples are Netscape Navigator 3.04 and 4.05, Internet Explorer 3.02,
4.01, and 5.0, Mosaic 3.0, Opera, and Emacs.

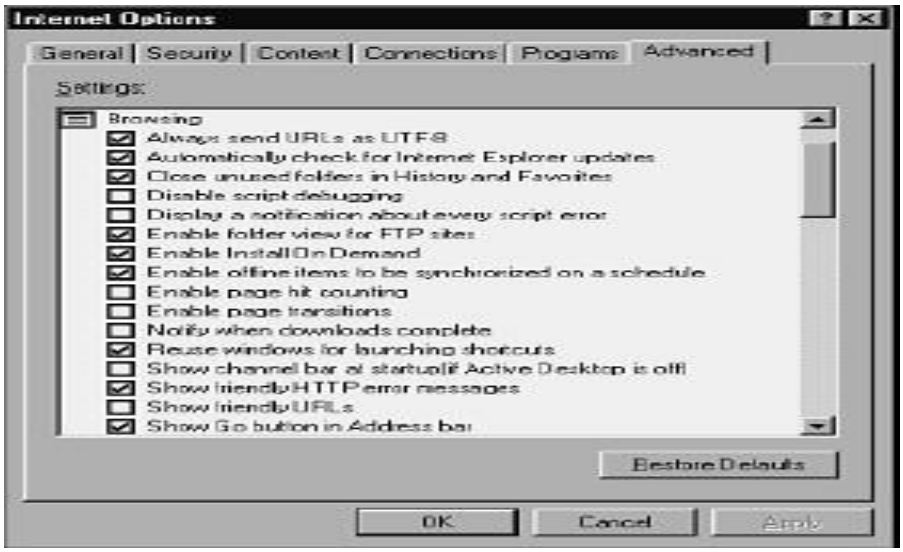
• **Browser Plug-Ins.** Many browsers can accept plug-ins or extensions to gain additional functionality. An example of this would be to play specific types of audio or video files.

• **Browser Options.** Most Web browsers allow for a great deal of customization. You can select security options, choose how ALT text is handled, decide what plug-ins to enable, and so on. Each option has potential impact on how your Web site operates—and, hence, is a test scenario to consider.

Video Resolution and Color Depth. Many platforms can display in various screen resolutions and colors. A PC running Windows, for example, can have screen dimensions of
640×480, 800×600, 1,024×768, 1280×1024, and up. Your Web site may look different, or even wrong, in one resolution, but not in another. Text and graphics can wrap differently, be cut off, or not appear at all. The number of colors that the platform supports can also impact the look of your site. There can be as few as 16 colors and as many as 224. Could your Web site be used on a system with only 16 colors?

• **Text Size.** Did you know that a user can change the size of the text used in the browser? Could your site be used with very small or very large text? What if it was being run on a small screen, in a low resolution, with large text?

• **Modem Speeds.** Enough can't be said about performance. Someday everyone will have high-speed connections with Web site data delivered as fast as you can view it. Until then, you need to test that your Web site works well at a wide range of modem speeds.



4) Usability Testing

The following list is adapted from his Top Ten Mistakes in Web Design:

• **Gratuitous Use of Bleeding-Edge Technology**. Your Web site shouldn't try to attract users by bragging about its use of the latest Web technology. When desktop publishing was young, people put 20 different fonts in their documents; try to avoid similar design bloat on the Web.

• **Scrolling Text, Marquees, and Constantly Running Animations**. Never allow page elements that move incessantly. Moving images have an overpowering effect on human peripheral vision.

• **Long Scrolling Pages.** Users typically don't like to scroll beyond the information visible onscreen when a page comes up. All critical content and navigation options should be on the top part of the page. Recent studies have shown that users are becoming more willing

to scroll now than they were in the early years of the Web, but it's still a good idea to minimize scrolling on navigation pages.

• **Non-Standard Link Colors**. Hyperlinks to pages that users haven't seen should be blue; links to previously seen pages should be purple or red. Don't mess with these colors because the ability to understand which links have been followed is one of the few navigational aids that's standard in most Web browsers. Consistency is key to teaching users what the link colors mean.

• **Outdated Information.** some pages are better off being removed completely from the server after their expiration date.

• **Overly Long Download Times.** Traditional human-factor guidelines indicate that 0.1 second is about the limit for users to feel that the system is reacting instantaneously. One second is about the limit for a user's flow of thought to stay uninterrupted. Ten seconds is the maximum response time before a user loses interest. On the Web, users have been trained to endure so much suffering that it may be acceptable to increase this limit to 15 seconds for a few pages. But don't aim for this—aim for less.

• **Lack of Navigation Support**. They will always have difficulty finding information, so they need support in the form of a strong sense of structure and place. Your site's design should start with a good understanding of the structure of the information space and communicate that structure explicitly to users. Provide a site map to let users know where they are and where they can go. The site should also have a good search feature because even the best navigation support will never be enough.

• **Orphan Pages**. Make sure that all pages include a clear indication of what Web site they belong to since users may access pages directly without coming in through your home page. For the same reason, every page should have a link to your home page as well as some indication of where they fit within the structure of your information space.

• **Complex Web Site Addresses (URLs).** Even though machine-level addressing like the URL should never have been exposed in the user interface, it's there and research has found that users actually try to decode the URLs of pages to infer the structure of Web sites. Users do this because of the lack of support for navigation and sense of location in current Web browsers. Thus, a URL should contain human-readable names that reflect the nature of the Web site's contents.

• **Using Frames**. Frames are an HTML technology that allows a Web site to display another Web site within itself, hence the name frame—like a picture frame. Splitting a page into frames can confuse users since frames break the fundamental user model of the Web page.