**UNIT-II**

# Black Box Testing

* **What is Black-Box Testing**

Also known as functional testing. A [software](http://www.webopedia.com/TERM/S/software.html) testing technique whereby the internal workings of the item being tested are not known by the tester.

For example, in a black box test on a software design the tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs.

The tester does not ever examine the programming [code](http://www.webopedia.com/TERM/C/code.html) and does not need any further knowledge of the program other than its specifications.

**The advantages of this type of testing include:**

* The test is unbiased because the designer and the tester are independent of each other.
* The tester does not need knowledge of any specific programming languages.
* The test is done from the point of view of the user, not the designer.
* Test cases can be designed as soon as the specifications are complete.

**The disadvantages of this type of testing include:**

* The test can be redundant if the software designer has already run a test case.
* The test cases are difficult to design.
* Testing every possible input stream is unrealistic because it would take a inordinate amount of time; therefore, many program paths will go untested.

### Why Black Box Testing

Black box testing helps in the overall functionality verification of the system under test.

Black box testing is done based on requirements It helps in identifying any incomplete, inconsistent requirement as well as any issues involved when the system is tested as a complete entity.

* **When to do Black Box Testing**

Black box testing activities require involvement of the testing team from the beginning of the software project life cycle, regardless of the software development life cycle model chosen for the project.

Testers can get involved right from the requirements gathering and analysis phase for the system under test. Test scenarios and test data are prepared during the test construction phase of the test life cycle, when the software is in the design phase.

### How to do Black Box Testing

Since we are testing external functionality in black box testing, we need to arrive at a judicious set of tests that test as much of the external functionality as possible, uncovering as many defects as possible, in as short a time as possible.

While this may look like a utopian wish list, the techniques we will discuss in this section facilitates this goal. This section deals with the various techniques to be used to generate test scenarios for effective black box testing.

Black box testing exploits specifications to generate test cases in a methodical way to avoid redundancy and to provide better coverage.

## Requirements-based testing

**Step 1:** For the requirements-based test, you first import a requirement baseline that contains all the requirements relevant to the test, for example in a tree structure. In the imbus TestBench, you can not only access the requirement texts, but also you have access to further fields, such as “status” or “priority” – in short: access to all the information that you need for planning and designing tests.

**Step 2:** Test managers or test designers then link requirements with – in some cases newly created – test themes or test case sets. In this way, the structure for the testing of requirements is drawn up and finally determined, which tests validate which requirements. Requirements that have not yet been assigned are optically highlighted so that it is possible to check at a glance whether all requirements have been covered by tests.

**Step 3:** In the following step, test designers specify particular tests within the given structure. At the same time, test managers can easily get an overview on all requirements for which tests are still missing.

* **Positive and negative testing**

In positive testing our intention is to prove that an application will work on giving valid input data. i.e. testing a system by giving its corresponding valid inputs.  
  
 In negative testing our intention is to prove that an application will not work on giving invalid inputs.

* **Boundary value analysis**
* Boundary value analysis (BVA) is based on testing at the boundaries between partitions.
* Here we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions).
* As an example, consider a printer that has an input option of the number of copies to be made, from 1 to 99. To apply boundary value analysis, we will take the minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case). In this example we would have three equivalence partitioning tests (one from each of the three partitions) and four boundary value tests. Consider the bank system described in the previous section in equivalence partitioning.

# Decision table in software testing

The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs.

However, if different combinations of inputs result in different actions being taken, this can be more difficult to show using equivalence partitioning and boundary value analysis, which tend to be more focused on the user interface.

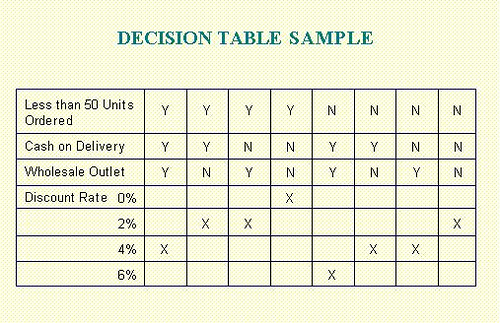
The other two specification-based [software testing](http://istqbexamcertification.com/what-is-a-software-testing/) techniques, decision tables and state transition testing are more focused on business logic or business rules.

A decision table is a good way to deal with combinations of things (e.g. inputs). This technique is sometimes also referred to as a ’cause-effect’ table.

The reason for this is that there is an associated logic diagramming technique called ’cause-effect graphing’ which was sometimes used to help derive the decision table (Myers describes this as a combinatorial logic network [Myers, 1979]).

However, most people find it more useful just to use the table described in [Copeland, 2003].

* Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers.
* Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules.
* It helps the developers to do a better job can also lead to better relationships with them. Testing combinations can be a challenge, as the number of combinations can often be huge. Testing all combinations may be impractical if not impossible. We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is also important. If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.



# Equivalence partitioning in Software testing

* Equivalence partitioning (EP) is a specification-based or black-box technique.
* It can be applied at any level of testing and is often a good technique to use first.
* The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence ‘equivalence partitioning’. Equivalence partitions are also known as equivalence classes – the two terms mean exactly the same thing.
* In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.
* **State based testing**

State based testing (SBT) is Functional Testing technique. The boundary value and equivalence class partitioning technique is more related to the various input combinations and their result. But state based technique is different from this.

In any system, it will have a set of inputs and corresponding actions. SBT is making use of this. You need to identify all possible valid and invalid transitions in the system and test them.

1. Identify various COMPONENTS in the system   
2. Identify the various possible STATE of these components   
3. Identify the possible ACTION ( ie : action which causes a transition from one state to other )   
4. Draw a state diagram based on these   
5. Test all valid and invalid transitions.

* **Compatibility testing**
* It is a type of non-functional testing.
* Compatibility testing is a type of [software testing](http://istqbexamcertification.com/what-is-a-software-testing/) used to ensure compatibility of the system/application/website built with various other objects such as other web browsers, hardware platforms, users (in case if it’s very specific type of requirement, such as a user who speaks and can read only a particular language), operating systems etc. This type of testing helps find out how well a system performs in a particular environment that includes hardware, network, operating system and other software etc.
* It is basically the testing of the application or the product built with the computing environment.
* It tests whether the application or the software product built is compatible with the hardware, operating system, database or other system software or not.
* **User Documentation Testing**

User Documentation  covers all the manuals, user guides, installation guides, setup guides, read me files, software release notes, and online help that are provided along with the software to help the end user to understand the software system. User Documentation Testing should have two objectives:-

1. To check if what is  stated in the document is available in the software
2. To check if what is there in the product is explained correctly in the document

This testing is plays a vital role as the users will refer this document when they start using the software at their location. a badly written document can put off a user and bias them against the product even the product offers rich functionality.

Defects found in the user documentation need to be tracked to closure like any regular software defect. Because these documents are the first interactions the users have with the product. A good User Documentation  aids in reducing customer support calls. The effort and money spend on this effort would form a valuable investment in the long run for the organisation.

* **Domain testing**

Domain testing is one of the most widely practiced software testing techniques. It is a method of selecting a small number of test cases from a nearly infinite group of candidate test cases. Domain knowledge plays a very critical role while testing domain-specific work.

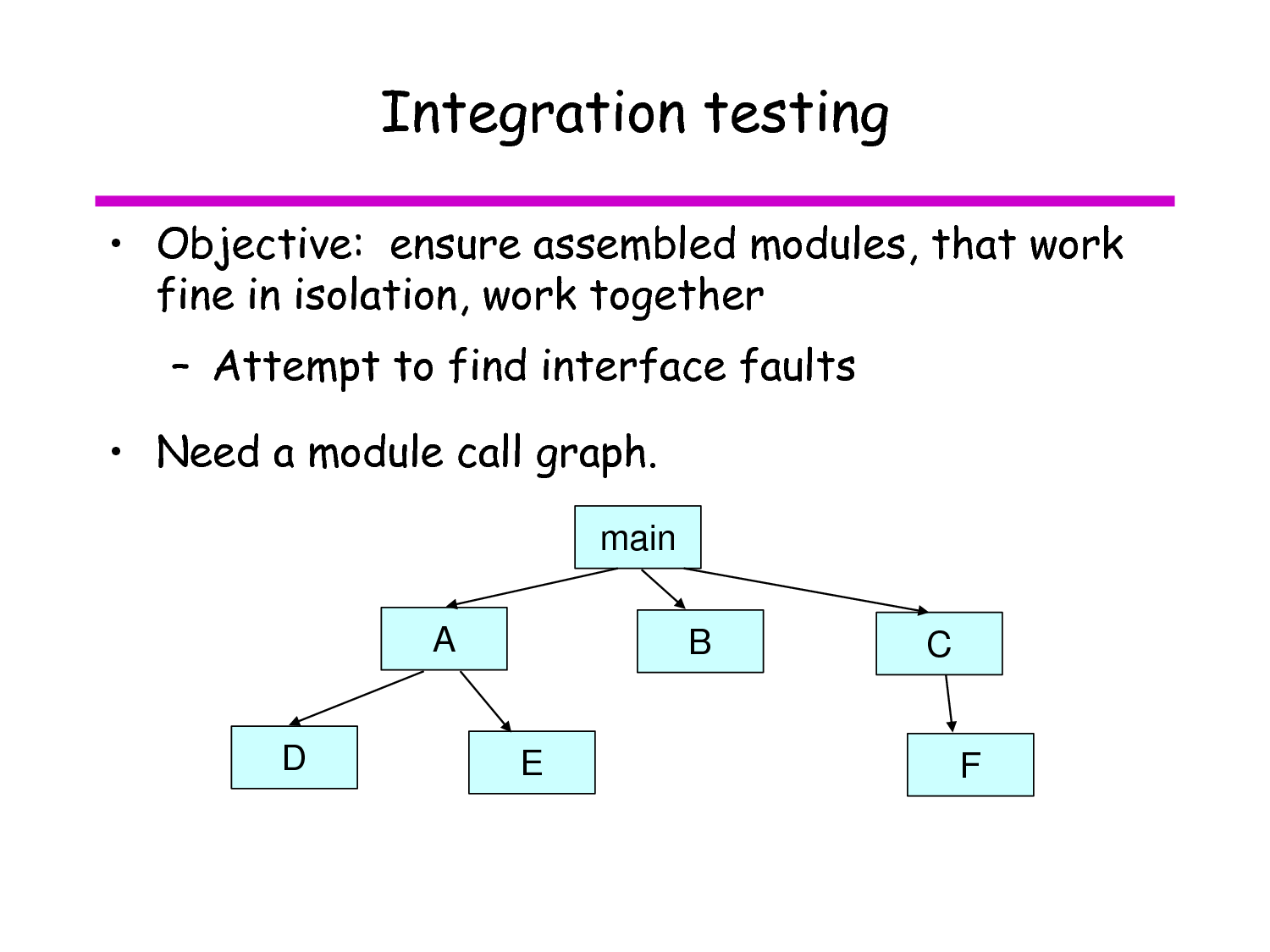
### Integrated testing

Integration testing is the testing process in [software testing](http://software-testing-tutorials-automation.blogspot.com/2011/06/what-is-software-testing.html) to verify that when two or more modules are interact and produced result satisfies with its original functional requirement or not.

Integrated testing is fall in Black box testing. Integrated testing will start after completion of unit testing.  [Software testing](http://software-testing-tutorials-automation.blogspot.com/2011/06/what-is-software-testing.html) engineer is performing integration testing.

# Integration Testing

Integration Testing is defined as the communication between two dependent modules. The pre requsitive for the integration Testing is having two different modules having tested for unit Testing.



* **Integration testing as a Type of Testing**

There are four types of methodologies available for the Integration Testing, they are

1. Top – Down Integration Testing.  
2. Bottom – Up Integration Testing.  
3. Bi – Directional Integration Testing.  
4. System Testing.

* Top – Down Integration Testing :

Top down Integration Testing involves, testing the High level or complex modules at the very first and later on moving to the simple modules in the application.

By Testing these way the developers will integrate the high level modules with the Low level modules inorder to complete integration process.

At times the development of low level modules will not be complete. At this stage, the developers will use a duplicate modules known as Stub and integrate with the Higher level modules. After the development process is over the stubs are removed.

**Advantages**

* Driver do not have to be written when top down testing is used.
* It provides early working module of the program and so design defects can be found and corrected early.

**Disadvantages**

* Stubs have to be written with utmost care as they will simulate setting of output parameters.
* It is difficult to have other people or third parties to perform this testing, mostly developers will have to spend time on this.
* **Bottom up integration**

In bottom up integration testing, module at the lowest level are developed first and other modules which go towards the 'main' program are integrated and tested one at a time.

Bottom up integration also uses test drivers to drive and pass appropriate data to the lower level modules. As and when code for other module gets ready, these drivers are replaced with the actual module.

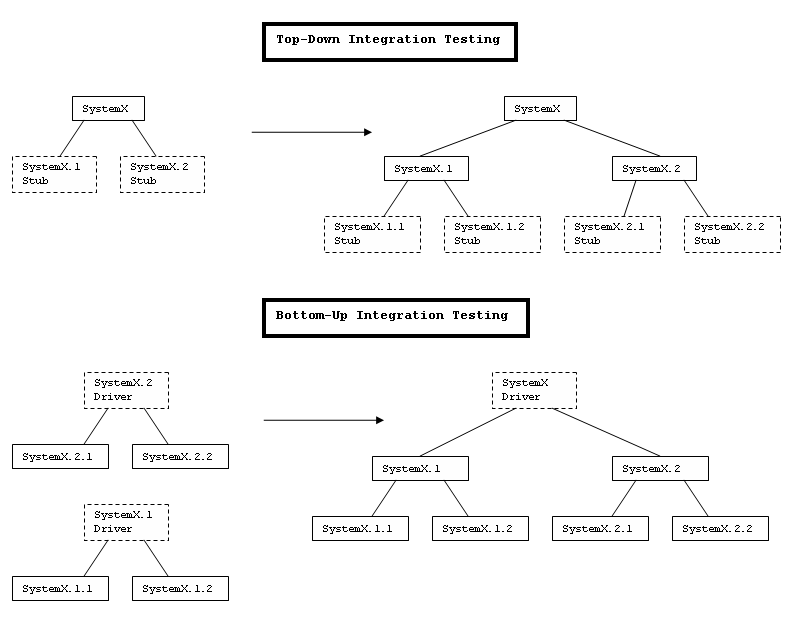
In this approach, lower level modules are tested extensively thus make sure that highest used module is tested properly.

**Advantages**

* Behavior of the interaction points is crystal clear, as components are added in the controlled manner and tested repetitively.
* Appropriate for applications where bottom up design methodology is used.

**Disadvantages**

* Writing and maintaining test drivers or harness is difficult than writing stubs.
* This approach is not suitable for the software development using top down approach.



* Bi Directional Integration Testing :

 Bi Directional Testing can be defined as the binding of Bottom up and Top down approaches. Bi Directional Testing does not guarantee the approach it is going to implement for the particular project.

* System Integration :

System Integration Testing can be defined as the testing of modules with complete workflow. When all the modules are combined instantly with other modules at a time and a complete testing is done. It is called as System Integration.

* It tests the interactions between different systems and may be done after system testing.
* It verifies the proper execution of software components and proper interfacing between components within the solution.
* The objective of SIT Testing is to validate that all software module dependencies are functionally correct and that data integrity is maintained between separate modules for the entire solution.
* As testing for dependencies between different components is a primary function of SIT Testing, this area is often most subject to Regression Testing.
* **Integration testing as a phase of testing**

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.

It occurs after unit testing phase and before system testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested. A component, in this sense, refers to an integrated aggregate of more than one unit.

In a realistic scenario, many units are combined into components, which are in turn aggregated into even larger parts of the program. The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups.

# Scenario testing

Scenario testing is a [software testing](http://en.wikipedia.org/wiki/Software_testing) activity that uses [scenarios](http://en.wikipedia.org/wiki/Scenario_(computing)): hypothetical stories to help the tester work through a complex problem or test system.

The ideal scenario test is a credible, complex, compelling or motivating story the outcome of which is easy to evaluate. These tests are usually different from [test cases](http://en.wikipedia.org/wiki/Test_case) in that test cases are single steps whereas scenarios cover a number of steps.

### System scenarios

In this method only those sets of realistic, user activities that cover several components in the system are used as scenario tests. Development of system scenario can be done using:

1. Story lines
2. State transitions
3. Business verticals
4. Implementation story from customers

### Use-case and role-based scenarios

In this method the focus is on how a user uses the system with different roles and environment.

* **Defect bash**

Defect bash is an ad hoc testing where people performing different roles in an organization test the product together at the same time.

This is very popular among application development companies, where the product can be used by people who perform different roles. The testing by all the participants during defect bash is not based on written test cases.

What is to be tested is left to an individual’s decision and creativity. They can also try some operations which are beyond the product specifications. Defect bash brings together plenty of good practices that are popular in testing industry. They are as follows.  
 It involves several steps:

Step1: choosing the frequency and duration of defect bash.

Step2: selecting the right product build.

Step3: communicating the objective of each defect bash to everyone.

Step4: setting up and monitoring the lab for the defect bash.

Step5: taking actions and fixing issues.

Step6: optimizing the effort involved in defect bash.

**ONE MARK QUESTIONS**

1. ------------------- is done without the knowledge of the internals of the system under test
2. **Black box testing** b. white box testing c. stress testing d. none
3. SRS stands for---------------------
4. **Software Requirements Specification** b. system requirement specification c. state required specification d. none
5. --------------- is the different ways of testing the requirements
6. **Test conditions** b. test cases c. a and b d. none
7. ---------------can be used to complete the mapping between test cases and the requirements
8. **Test case id** b. test condition c. a and b d. none
9. -------------------test the products compliance to the requirements specifications
10. **Requirements based testing** b. white box testing c. stress testing d. none
11. Most of the defects in software products hover around---------------------
12. Conditions b. boundaries c. **a and b** d none
13. --------------------is useful when input and output data can be expressed as Boolean conditions
14. **Decision table** b. boundary value analysis c. negative testing d. none
15. The set of input values that generate one single expected output is called-----------------
16. **Partition** b. equivalence c. a and b d. none
17. Testing done to ensure that the product features work consistently with different infrastructure components is called---------
18. **Compatibility testing** b. decision table c. stress testing d. none
19. ----------------- testing exploits the fasters domain knowledge to test the suitability of the product to what the users do an a typical day
20. **Domain** b. decision c. stress d. none
21. --------------testing is testing the interfaces
22. **Integration** b. domain c. compatibility d. none
23. SKD stands for---------------
24. **Software Development Kits** b. system development kit c. software deployment kit d. none
25. ---------------------- is the opposite of top down integration
26. **Bottom up integration** b. bi directional c. system d. all
27. All the components of the system are integrated and tested as a single unit is called----------
28. **System integration** b. bi direction c. top down d. none
29. -------------matrix gas as its columns various parameters the combinations of which have to be tested
30. **Compatibility** b. horizontal c. intelligent d. none
31. -------------- is done to ensure the documentation matches the product and vice-versa
32. **User documentation** b. domain c. compatibility testing d. none
33. ------------------- testing without applying a methodology is similar to looking at the map without knowing where you are and what your destination is.
34. **Performing black box** b. compatibility c. domain testing d. none
35. The functionality of different components are combined and tested together for a sequence for related operations are called------------------
36. **Scenarios** b. system scenario c. battle ground d. none
37. ------------------------is a stepwise procedure on how a user intends to use a system, with different user roles and associated parameters
38. **Use case scenario** b. system scenario c. integration testing d. none
39. ------------------ is an ad hoc testing where people performing different roles in an organization test the product together at the same time
40. **Defect bash** b. scenario c. system scenario d. none

**ANSWER THE FOLLOWING QUESTIONS**

5 MARKS

1. What is black box testing?
2. When to do black box testing?
3. What is domain testing?
4. What is Integration testing?
5. Short note on top-down and bottom-up integration?

8 MARKS

1. Explain how to do black box testing?
2. Explain about domain testing?
3. Describe integration testing as a type of testing?
4. Explain about system integration?
5. Explain integration testing as a phase of testing?
6. Explain system and use case scenario?