|  |  |  |
| --- | --- | --- |
| **BASIS FOR COMPARISON** | **TOP-DOWN APPROACH** | **BOTTOM-UP APPROACH** |
| Basic | Breaks the massive problem into smaller subproblems. | Solves the fundamental low-level problem and integrates them into a larger one. |
| Process | Submodules are solitarily analysed. | Examine what data is to be encapsulated, and implies the concept of information hiding. |
| Communication | Not required in the top-down approach. | Needs a specific amount of communication. |
| Redundancy | Contain redundant information. | Redundancy can be eliminated. |
| Programming languages | Structure/procedural oriented programming languages (i.e. C) follows the top-down approach. | Object-oriented programming languages (like C++, Java, etc.) follows the bottom-up approach. |
| Mainly used in | Module documentation, test case creation, code implementation and debugging. | Testing |

The  **top-down** approach basically divides a complex problem or algorithm into multiple smaller parts (modules). These modules are further decomposed until the resulting module is the fundamental program essentially be understood and can not be further decomposed. After achieving a certain level of modularity, the decomposition of modules is ceased. The top-down approach is the stepwise process of breaking of the large program module into simpler and smaller modules to organise and code program in an efficient way. The flow of control in this approach is always in the downward direction. The top-down approach is implemented in the “C” programming language by using functions.

Thus, the top-down method begins with abstract design and then sequentially this design is refined to create more concrete levels until there is no requirement of additional refinement.

Definition of Bottom-up Approach

The **bottom-up** approach works in just opposite manner to the top-down approach. Initially, it includes the designing of the most fundamental parts which are then combined to make the higher level module. This integration of submodules and modules into the higher level module is repeatedly performed until the required complete algorithm is obtained.

Bottom-up approach functions with layers of abstraction. The primary application of the bottom-up approach is testing as each fundamental module is first tested before merging it to the bigger one. The testing is accomplished using the certain low-level functions.

Programming refers to the method of creating a sequence of instructions to enable the [computer](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) to perform a task. It is done by developing logic and then writing instructions in a programming language. A program can be written using various programming practices available. A **programming practice**refers to the way of writing a program and is used along with coding style guidelines. Some of the commonly used programming practices include top-down programming, bottom-up programming, structured programming, and [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) hiding.

## [Programming Practices](http://ecomputernotes.com/images/Programming-Practices.jpg)

## Top-down Programming

Top-down programming focuses on the use of modules. It is therefore also known as modular programming. The program is broken up into small modules so that it is easy to trace a particular segment of code in the software program. The modules at the top level are those that perform general tasks and proceed to other modules to perform a particular task. Each module is based on the functionality of its functions and procedures. In this approach, programming begins from the top level of hierarchy and progresses towards the lower levels. The implementation of modules starts with the main module. After the implementation of the main module, the subordinate modules are implemented and the process follows in this way. In top-down programming, there is a risk of implementing data structures as the modules are dependent on each other and they nave to share one or more functions and procedures. In this way, the functions and procedures are globally visible. In addition to modules, the top-down programming uses sequences and the nested levels of commands.

## Bottom-up Programming

Bottom-up programming refers to the style of programming where an application is constructed with the description of modules. The description begins at the bottom of the hierarchy of modules and progresses through higher levels until it reaches the top. Bottom-up programming is just the opposite of top-down programming. Here, the program modules are more general and reusable than top-down programming.

It is easier to construct functions in bottom-up manner. This is because bottom-up programming requires a way of passing complicated arguments between functions. It takes the form of constructing abstract [data type](http://ecomputernotes.com/java/data-type-variable-and-array/explain-data-types-in-java)s in languages such as C++ or Java, which can be used to implement an entire class of applications and not only the one that is to be written. It therefore becomes easier to add new features in a bottom-up approach than in a top-down programming approach.

## Structured Programming

Structured programming is concerned with the structures used in a computer program. Generally, structures of computer program comprise decisions, sequences, and loops. The **decision structures**are used for conditional execution of statements (for example, 'if statement). The **sequence structures**are used for the sequentially executed statements. The **loop structures**are used for performing some repetitive tasks in the program.

Structured programming forces a logical structure in the program to be written in an efficient and understandable manner. The purpose of structured programming is to make the software code easy to modify when required. Some languages such as Ada, Pascal, and dBase are designed with features that implement the logical program structure in the software code. Primarily, the structured programming focuses on reducing the following statements from the program.

1. 'GOTO' statements.
2. 'Break' or 'Continue' outside the loops.
3. Multiple exit points to a function, procedure, or subroutine. For example, multiple 'Return' statements should not be used.
4. Multiple entry points to a function, procedure, or a subroutine.

Structured programming generally makes use of top-down design because program structure is divided into separate subsections. A defined function or set of similar functions is kept separately. Due to this separation of functions, they are easily loaded in the memory. In addition, these functions can be reused in one or more programs. Each module is tested individually. Aftertesting, they are integrated with other modules to achieve an overall program structure. Note that a key characteristic of a structured statement is the presence of single entry and single exit point. This characteristic implies that during execution, a structured statement starts from one defined point and terminates at another defined point.

## 

## Information Hiding

Information hiding focuses on hiding the non-essential details of functions and code in a program so that they are inaccessible to other components of the software. A software developer applies information hiding in software design and coding to hide unnecessary details from the rest of the program. The objective of information hiding is to minimize complexities among different modules of the software. Note that complexities arise when one program or module in software is dependent on several other programs and modules.

Information hiding is implemented with the help of interfaces. An interface is a medium of interaction for software components that are using the properties of the software modules containing data. The implementation of interfaces depends on the syntax and process. Examples of interface include constants, data types, types of procedures, and so on. Interfaces protect other parts of programs when a software design is changed.

Generally, the interfaces act as a foundation to modular programming (top-down programming) and object-oriented programming. In object-oriented programming, interface of an object comprises a set of methods, which are used to interact with the objects of the software programs. Using information hiding, a single program is divided into several modules. These modules are independent of each other and can be used interchangeably in other software programs.

To understand the concept of information hiding, let us consider an example of a program written for 'car'. The program can be organized in several ways. One is to arrange modules without using information hiding. In this case, the modules can be created as 'front part', 'middle part', and 'rear part'. On the other hand, creating modules using information hiding includes specifying names of modules such as 'engine' and 'steering'.

On comparison, it is found that modules created without using information hiding affect other modules. This is because when a module is modified, it affects the data, which does not require modification. However, if modules are created using information hiding, then modules are concerned only with specific segments of the program and not the whole program or other parts of the program. In our example, this statement means that the module 'engine' does not have any affect on the module 'steering'.

**Code documentation** is a manual-cum-guide that helps in understanding and correctly utilizing the software code. The coding standards and naming conventions written in a commonly spoken language in code documentation provide enhanced clarity for the designer. Moreover, they act as a guide for the software maintenance team (this team focuses on maintaining software by improving and enhancing the software after it has been delivered to the end user) while the software maintenance process is carried out. In this way, code documentation facilitates code reusability.

 While writing a software code, the developer needs proper documentation for reference purposes. Programming is an ongoing process and requires modifications from time to time. When a number of software developers are writing the code for the same software, complexity increases. With the help of documentation, software developers can reduce the complexity by referencing the code documentation. Some of the documenting techniques are comments, visual appearances of codes, and programming tools. **Comments**are used to make the reader understand the logic of a particular code segment. The **visual appearance of a code**is the way in which the program should be formatted to increase readability. The **programming tools** in code documentation are algorithms, flowcharts, and pseudo-codes.

Code documentation contains source code, which is useful for the software developers in writing the software code. The code documents can be created with the help of various coding tools that are used to auto-generate the code documents. In other words, these documents extract comments from the source code and create a reference manual in the form of text or HTML file. The auto-generated code helps the software developers to extract the source code from the comments. This documentation also contains application programming interfaces, data structures, and algorithms. There are two kinds of code documentation, namely, internal documentation and external documentation.

Documentation which focuses on the [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) that is used to determine the software code is known as internal documentation. It describes the data structures, algorithms, and control flow in the programs. There are various guidelines for making the documentation easily understandable to the reader. Some of the general conventions to be used at the time of internal documentation are header comment blocks, program comments, and formatting. Header comment blocks are useful in identifying the purpose of the code along with details such as how the c0ge functions and how each segment of code is used in the program.

Since software code is updated and revised several times, it is important to keep a record of the code information so that internal documentation reflects the changes made to the software code. Internal documentation should explain how each code section relates to user requirements in the software. Generally, internal documentation comprises the following information.

1. Name, type, and purpose of each variable and data structure used in the code
2. Brief description of algorithms, logic, and error-handling techniques
3. Information about the required input and expected output of the program
4. Assistance on how to test the software
5. Information on the upgradations and enhancements in the program.

Documentation which focuses on general description of the software code and is not concerned with its detail is known as external documentation. It includes information such as function of code, name of the software developer who has written the code, algorithms used in the software code, dependency of code on programs and libraries, and format of the output produced by the software code. Generally, external documentation includes structure charts for providing an outline of the program and describing the design of the program.

External documentation is useful for software developers as it consists of information such as description of the problem along with the program written to solve it. In addition, it describes the approach used to solve the problem, operational requirements of the program, and user interface components. For the purpose of readability and proper understanding, the detailed description is accompanied by figures and illustrations that show how one component is related to another. External documentation explains why a particular solution is chosen and implemented in the software. It also includes formulas, conditions, and references from where the algorithms or documentation are derived. External documentation makes the user aware of the errors that occur while running the software code. For example, if an array of five numbers is used, it should be mentioned in the external documentation that the limit of the array is five.

## Code Documentation Tools

While writing software code documentation, it is important to consider the code documentation tools required for writing the software code. The software documentation tools conform to standards by generating the required elements automatically with configurable format and style. These tools combine the selected comment sections with the software code to generate a usable documentation with the essential level of details in it. Some of the code documentation tools are listed in Table.

**Table Code Documentation Tools**

|  |  |  |
| --- | --- | --- |
| **Documentation Tools** | **Language Supported** | **Description** |
| Cocoon | C++ | Used to process C++ library files and generates web pages that are useful to document the libraries, classes, and global functions. |
| CcDoc | C++ | Used for implementing the document standards in Java and C++. |
| CxRef | C | Used to generate documents in HTML, RTF, and so on. It also includes cross-references from source code of C programs. |
| DOC++ | C, C++, Java | Used for providing output for the documentations produced in C, C++, and Java. |
| JavaDoc | Java | Used as a standard for documentation in Java. |
| Perceps | C++ | Used to break C and C++ header files into separate header files. It generates documentation in various formats according to class definitions, declarations, and comments included in those files. |
| RoboDoc | Assembler, C, Perl, LISP, Fortran, Shell scripts, COBOL | Used to convert formatted documentation into cross-referenced set of HTML pages, which describe the software code. |
| DocJet | Java, C, C++, Visual Basic | Used to generate documentation from comments in the source code. |
| ObjectManual | C++ | Used to generate documentation in the form of HTML, XML, and RTF pages. |
| Together | Java, C++ | Used to generate documentation from UML and its source code. |
| Doc-o-matic | C++, C#, ASP.NET, VB.NET, Java, JavaScript, JSP | Used to create documentations such as source code documentation, online help, and user manuals. It is integrated with easy to use interface for managing the documentation projects. |

Code documentation tools should be simple to use because easy-to-use documentation tools provide rapid feedback. However, the basic features of software code documentation tools are listed below.

1. **Target media:**Views software code easily in a web browser. The target media is useful in displaying the structure and layout of the page with sufficient precision. It is required for code documentation, so that the software code can be easily used in the web browser. An example of target media is **HTML.**
2. **Documentation structure:**Includes the index to pages and chapters. The chapters in the documentation should include information such as title, introduction, table of contents, and sections.
3. **Comment extraction capabilities:**Extracts the software code comments regardless of the style used in the software code.
4. **Languages supported:**Makes the code documentation consistent while writing the software code. The code documentation tools support multiple programming languages and are preferred for concentrating on a particular language.
5. **Formatting and style elements:**Make the format of the software code proper. Special elements such as tags or mark-ups are required to determine the layout, structure, and style of the code documentation.
6. **Code readability:**Makes the software code consistent and easily readable. This is because code documentation is itself not sufficient and requires the comments in the software code to make the code documentation readable and understandable.

**In**addition to the above mentioned features, the amount of detail provided is also an important feature. Too much detail makes the code documentation inefficient and proves unnecessary. The level of details should be according to the software developer and not according to the coding tools used in the software code

The Differences Between [Black Box Testing](http://softwaretestingfundamentals.com/black-box-testing/) and [White Box Testing](http://softwaretestingfundamentals.com/white-box-testing/) are listed below.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Black Box Testing** | **White Box Testing** |
| Definition | Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester | White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. |
| Levels Applicable To | Mainly applicable to higher levels of testing:[Acceptance Testing](http://softwaretestingfundamentals.com/acceptance-testing/)  [System Testing](http://softwaretestingfundamentals.com/system-testing/) | Mainly applicable to lower levels of testing:[Unit Testing](http://softwaretestingfundamentals.com/unit-testing/)  [Integration Testing](http://softwaretestingfundamentals.com/integration-testing/) |
| Responsibility | Generally, independent Software Testers | Generally, Software Developers |
| Programming Knowledge | Not Required | Required |
| Implementation Knowledge | Not Required | Required |
| Basis for Test Cases | Requirement Specifications | Detail Design |

| **Parameter** | **Black Box testing** | **White Box testing** |
| --- | --- | --- |
| **Definition** | It is a testing approach which is used to test the software without the knowledge of the internal structure of program or application. | It is a testing approach in which internal structure is known to the tester. |
| **Alias** | It also knowns as data-driven, box testing, data-, and functional testing. | It is also called structural testing, clear box testing, code-based testing, or glass box testing. |
| **Base of Testing** | Testing is based on external expectations; internal behavior of the application is unknown. | Internal working is known, and the tester can test accordingly. |
| **Usage** | This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing. | Testing is best suited for a lower level of testing like Unit Testing, Integration testing. |
| **Programming knowledge** | Programming knowledge is not needed to perform Black Box testing. | Programming knowledge is required to perform White Box testing. |
| **Implementation knowledge** | Implementation knowledge is not requiring doing Black Box testing. | Complete understanding needs to implement WhiteBox testing. |
| **Automation** | Test and programmer are dependent on each other, so it is tough to automate. | White Box testing is easy to automate. |
| **Objective** | The main objective of this testing is to check what functionality of the system under test. | The main objective of White Box testing is done to check the quality of the code. |
| **Basis for test cases** | Testing can start after preparing requirement specification document. | Testing can start after preparing for Detail design document. |
| **Tested by** | Performed by the end user, developer, and tester. | Usually done by tester and developers. |
| **Granularity** | Granularity is low. | Granularity is high. |
| **Testing method** | It is based on trial and error method. | Data domain and internal boundaries can be tested. |
| **Time** | It is less exhaustive and time-consuming. | Exhaustive and time-consuming method. |
| **Algorithm test** | Not the best method for algorithm testing. | Best suited for algorithm testing. |
| **Code Access** | Code access is not required for Black Box Testing. | White box testing requires code access. Thereby, the code could be stolen if testing is outsourced. |
| **Benefit** | Well suited and efficient for large code segments. | It allows removing the extra lines of code, which can bring in hidden defects. |
| **Skill level** | Low skilled testers can test the application with no knowledge of the implementation of programming language or operating system. | Need an expert tester with vast experience to perform white box testing. |
| **Techniques** | Equivalence partitioning is Black box testing technique is used for Blackbox testing.   Equivalence partitioning divides input values into valid and invalid partitions and selecting corresponding values from each partition of the test data.   Boundary value analysis   checks boundaries for input values. | Statement Coverage, Branch coverage, and Path coverage are White Box testing technique.  Statement Coverage validates whether every line of the code is executed at least once.   Branch coverage validates whether each branch is executed at least once   Path coverage method tests all the paths of the program. |
| **Drawbacks** | Update to automation test script is essential if you to modify application frequently. | Automated test cases can become useless if the code base is rapidly changing. |

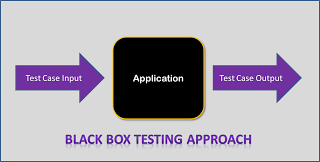
Software testing is one of the best means to affirm the quality of software and deliver an error-free application. Over the years, [software testing](https://www.testing-whiz.com/) has matured into a separate discipline giving way to several different testing techniques that have been introduced, analyzed and studied in this area. Black box testing and white box testing are two such testing approaches that are quite commonly used by software testers.

## What is Black Box Testing?

Crudely put, when the tester has no idea of the internal working of the system which he is testing, that approach is called black box testing.

In this case, the system under test is viewed as a “black box”.

Requirements Document or Functional Specification Document forms the basis of this testing, which requires the user to understand the processes within the software.



### How to write Test Cases for Black Box Testing?

* The tester examines requirements and specifications of the system.
* The tester explores the system’s UI and functionality to understand how the processes on the system are expected to work.
* Tester designs test cases with valid inputs and the corresponding expected outputs.
* Tester also includes some negative test cases with invalid inputs and expected outputs (error messages/program termination) as applicable.

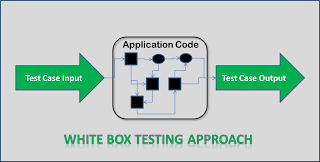
### Techniques of Black Box Testing

In case of black box testing, inputs to the test cases are the driving factor. Any one of the three techniques discussed below can be used to choose the inputs during the black box testing process

* **Boundary Value Analysis**: This approach is focused on testing the boundary values associated with the system. This approach aims at testing the boundaries of the input domain that have the highest probability of giving erroneous outputs.
* **Equivalence Class Partitioning**: In this approach, a limited set of functions is identified along with its corresponding valid and invalid inputs and expected outputs. This approach aims at identifying classes of errors and therefore reducing the number of test cases required.
* **Error Guessing**: An experienced tester most often uses this approach to first identify the defects and then develop corresponding test cases.

## What is White Box Testing?

In [white box testing](https://www.testing-whiz.com/manual-testing-services.html) methodology, the tester has the knowledge of the internals of a system and knows how the system is implemented. The tester uses this knowledge to develop test cases that will examine the control flow, information flow, data flow, exception and error handling as well as coding practices of the system.



### How to write Test Cases for White Box Testing?

* The tester analyzes and understands the structure of the system by examining its code.
* The tester understands the weak spots within the code that is most prone to defects.
* The tester develops test cases to cover individual data/information/ control flows and branches within the code.
* The tester also develops test cases to test proper working of all the functionalities and error handing of the system.

### Techniques of White Box Testing

When it comes to white box testing, the knowledge that the tester possesses about the system is the driving factor, which helps the tester to devise test cases aimed at discovering defects with the internal working of the system.

* **Statement Tests**: All the statements within the code must have a test case associated with it such that each statement must be executed at least once during the testing cycle.
* **Decision Tests**: All the decision directions must be executed at least once during the testing life cycle.
* **Branch Condition Tests**: All the conditions in a specific decision must be tested for proper working at least once.
* **Decision/Condition Tests**: All the combination of the possible conditions within a specific decision for all the decisions is to be tested.
* **Data Flow Tests**: This will ensure that all the variables and data that are used within the system are tested by passing the specific variables through each possible calculation.
* **Multiple Condition Tests**: This will ensure that each point of entry with in the code is tested at least once during the testing life cycle.

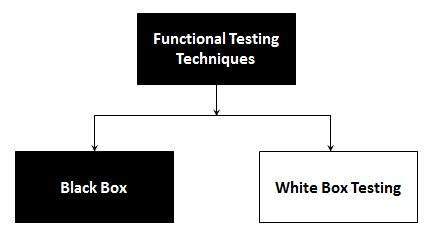
## What is Grey Box Testing?

This is an approach that lies between white box and black box testing. In this, the test cases are written with the knowledge of internal source code of the application. However, the testing is done at the interface level only.

Functional Testing is a testing technique that is used to test the features/functionality of the system or Software, should cover all the scenarios including failure paths and boundary cases.

## Functional Testing Techniques:

There are two major Functional Testing techniques as shown below:



The other major Functional Testing techniques include:

* Unit Testing
* Integration Testing
* Smoke Testing
* User Acceptance Testing
* Localization Testing
* Interface Testing
* Usability Testing
* System Testing
* Regression Testing
* Globalization Testing

##### **What is Functional Testing?**

[Functional Testing](https://www.softwaretestingclass.com/functional-testing/) is the type of testing done against the business requirements of application. It is a black box type of testing.

It involves the complete integration system to evaluate the system’s compliance with its specified requirements. Based on the functional specification document this type of testing is to be carried out. In actual testing, testers need to verify a specific action or function of the code. For functional testing either manual testing or automation tools can be used but functionality testing would be easier using manual testing only. Prior to non Functional testing the Functional testing would be executed first.

Five steps need to be keeping in mind in the Functional testing:

1. Preparation of test data based on the specifications of functions
2. Business requirements are the inputs to functional testing
3. Based on functional specifications find out of output of the functions
4. The execution of test cases
5. Observe the actual and expected outputs

To carry out functional testing we have numerous tools available, here is the list of [Functional testing tools](https://www.softwaretestingclass.com/software-testing-tools-list/).

In the **types of functional testing** following testing types should be cover:

* Unit Testing
* [Smoke testing](https://www.softwaretestingclass.com/smoke-testing/)
* [Sanity testing](https://www.softwaretestingclass.com/sanity-testing/)
* Integration Testing
* Interface Testing
* System Testing
* [Regression Testing](https://www.softwaretestingclass.com/regression-testing-definition/)
* UAT

##### **What is non Functional Testing?**

The non Functional Testing is the type of testing done against the **non functional requirements**. Most of the criteria are not consider in functional testing so it is used to **check the readiness of a system.** Non-functional requirements tend to be those that reflect the quality of the product, particularly in the context of the suitability perspective of its users. It can be started after the completion of Functional Testing. The **non functional tests** can be effective by using testing tools.

The testing of software attributes which are not related to any specific function or user action like performance, scalability, security or behavior of application under certain constraints.

Non functional testing has a great influence on customer and user satisfaction with the product. Non functional testing should be expressed in a testable way, not like “the system should be fast” or “the system should be easy to operate” which is not testable.

Basically in the non functional test is used to major **non-functional attributes of software systems.** Let’s take **non functional requirements** examples; in how much time does the software will take to complete a task? or how fast the response is.

Following testing should consider in **non functional testing types**:

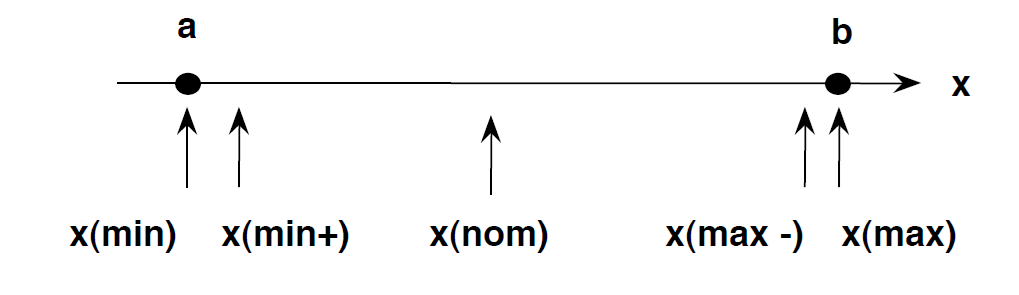
* Availability Testing
* Baseline testing
* Compatibility testing
* Compliance testing
* Configuration Testing
* Documentation testing
* Endurance testing
* Ergonomics Testing
* Interoperability Testing
* Installation Testing
* Load testing
* Localization testing and Internationalization testing
* Maintainability Testing
* Operational Readiness Testing
* Performance testing
* Recovery testing
* Reliability Testing
* Resilience testing
* Security testing
* Scalability testing
* Stress testing
* Usability testing
* Volume testing

**What is Boundary Testing?**

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

* So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
* The basic idea in boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum

[](https://www.guru99.com/images/3-2016/032316_0620_Equivalence1.png)

 In Boundary Testing, Equivalence Class Partitioning plays a good role

 Boundary Testing comes after the Equivalence Class Partitioning.

## II What is Boundary Testing?

Boundary value analysis is a type of black box or specification based testing technique in which tests are performed using the boundary values.

## Example:

An exam has a pass boundary at 50 percent, merit at 75 percent and distinction at 85 percent. The Valid Boundary values for this scenario will be as follows:

49, 50 - for pass

74, 75 - for merit

84, 85 - for distinction

Boundary values are validated against both the valid boundaries and invalid boundaries.

The Invalid Boundary Cases for the above example can be given as follows:

0 - for lower limit boundary value

101 - for upper limit boundary value

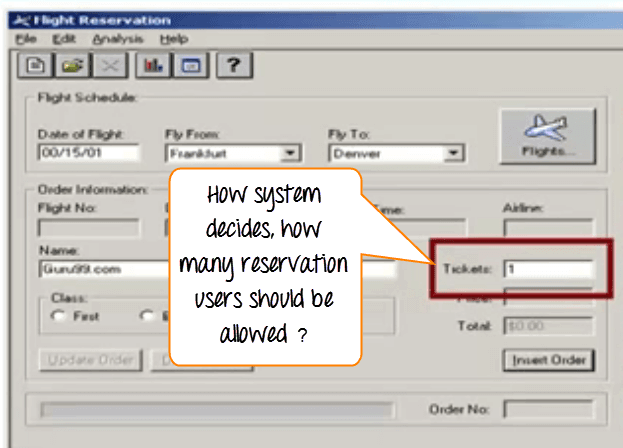
**What is Equivalent Class Partitioning?**

Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same.

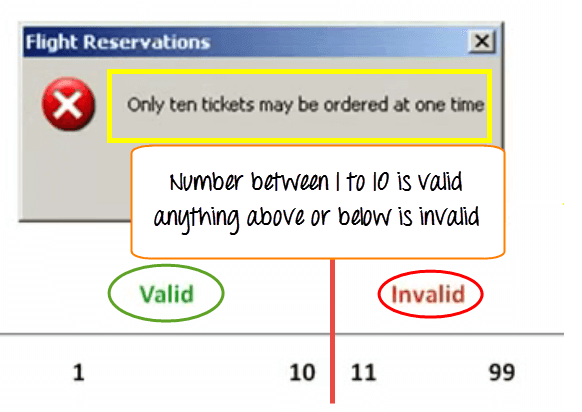
* It divides the input data of software into different equivalence data classes.
* You can apply this technique, where there is a range in input field.

**Example 1: Equivalence and Boundary Value**

* Let's consider the behavior of tickets in the Flight reservation application, while booking a new flight.

[](https://www.guru99.com/images/3-2016/032316_0620_Equivalence2.png)

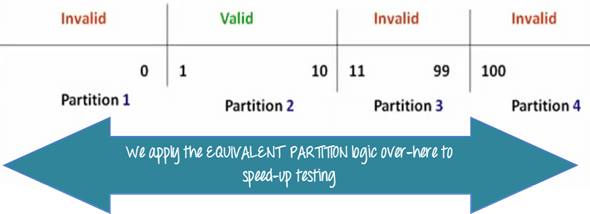
* Ticket values 1 to 10 are considered valid & ticket is booked. While value 11 to 99 are considered invalid for reservation and error message will appear, **"Only ten tickets may be ordered at one time."**

[](https://www.guru99.com/images/3-2016/032316_0620_Equivalence3.png)

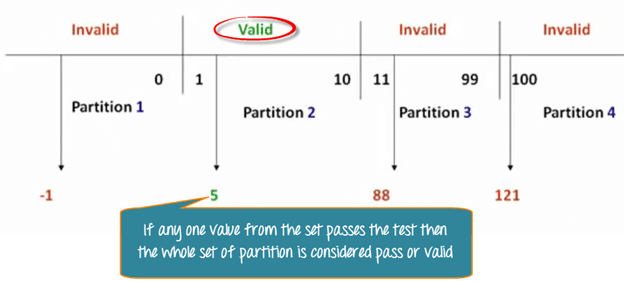
**Here is the test condition**

1. Any Number greater than 10 entered in the reservation column (let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

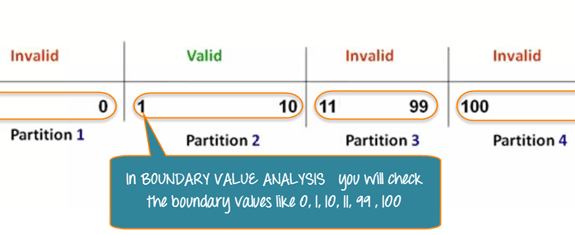
We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.

[](https://www.guru99.com/images/3-2016/032316_0620_Equivalence4.png)

The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.

[](https://www.guru99.com/images/3-2016/032316_0620_Equivalence5.png)

**Boundary Value Analysis**- in Boundary Value Analysis, you test boundaries between equivalence partitions

[](https://www.guru99.com/images/3-2016/032316_0620_Equivalence6.png)

In our earlier example instead of checking, one value for each partition you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at**both valid and invalid boundaries**. Boundary Value Analysis is also called**range checking**.

Equivalence partitioning and boundary value analysis are closely related and can be used together at all levels of testing.

**Example 2: Equivalence and Boundary Value**

Suppose a password field accepts minimum 6 characters and maximum 10 characters

That means results for values in partitions 0-5, 6-10, 11-14 should be equivalent

|  |  |  |
| --- | --- | --- |
| **Test Scenario #** | **Test Scenario Description** | **Expected Outcome** |
| 1 | Enter 0 to 5 characters in password field | System should not accept |
| 2 | Enter 6 to 10 characters in password field | System should accept |
| 3 | Enter 11 to 14 character in password field | System should not accept |

**Examples 3: Input Box should accept the Number 1 to 10**

Here we will see the Boundary Value Test Cases

|  |  |
| --- | --- |
| **Test Scenario Description** | **Expected Outcome** |
| Boundary Value = 0 | System should NOT accept |
| Boundary Value = 1 | System should accept |
| Boundary Value = 2 | System should accept |
| Boundary Value = 9 | System should accept |
| Boundary Value = 10 | System should accept |
| Boundary Value = 11 | System should NOT accept |

**Why Equivalence & Boundary Analysis Testing**

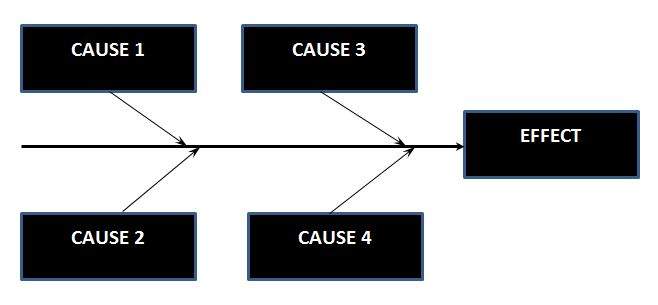
1. This testing is used to reduce very large number of test cases to manageable chunks.
2. Very clear guidelines on determining test cases without compromising on the effectiveness of testing.
3. Appropriate for calculation-intensive applications with large number of variables/inputs

## What is Cause-Effect Graph?

Cause Effect Graph is a black box testing technique that graphically illustrates the relationship between a given outcome and all the factors that influence the outcome.

It is also known as Ishikawa diagram as it was invented by Kaoru Ishikawa or fish bone diagram because of the way it looks.

## Cause Effect - Flow Diagram



## Circumstances - under which Cause-Effect Diagram used

* To Identify the possible root causes, the reasons for a specific effect, problem, or outcome.
* To Relate the interactions of the system among the factors affecting a particular process or effect.
* To Analyze the existing problems so that corrective action can be taken at the earliest.

## Benefits :

* It Helps us to determine the root causes of a problem or quality using a structured approach.
* It Uses an orderly, easy-to-read format to diagram cause-and-effect relationships.
* It Indicates possible causes of variation in a process.
* It Identifies areas, where data should be collected for further study.
* It Encourages team participation and utilizes the team knowledge of the process.
* It Increases knowledge of the process by helping everyone to learn more about the factors at work and how they relate.

## Steps for drawing cause-Effect Diagram:

* **Step 1 :**Identify and Define the Effect
* **Step 2 :**Fill in the Effect Box and Draw the Spine
* **Step 3:**Identify the main causes contributing to the effect being studied.
* **Step 4 :**For each major branch, identify other specific factors which may be the causes of the EFFECT.
* **Step 5 :**Categorize relative causes and provide detailed levels of causes.