UNIT - V

Software Testing Techniques

Black box testing

- Black-box or functional testing is one in which test conditions are developed based on the program or system's functionality; that is, the tester requires information about the input data and observed output, but does not know how the program or system works.

Black box testing continued…

Example:

As one does not have to know how a car works internally to drive it.

It is not necessary to know the internal structure of a program to execute it.

Black box testing continued..

- The tester focuses on testing the program's functionality against the specification.

- With black-box testing, the tester views the program as a black-box and is completely unconcerned with the internal structure of the program or system.

Black box testing continued…

Some examples in this category include:

- decision tables,
- equivalence partitioning,
- range testing,
- boundary value testing,
- database integrity testing,
- Cause effect graphing,
- orthogonal array testing,
- array and table testing,
- Exception testing,
- limit testing,
- random testing.

Advantages

- Tests are geared to what the program or system is supposed to do, and it is natural and understood by everyone.

Boundary value

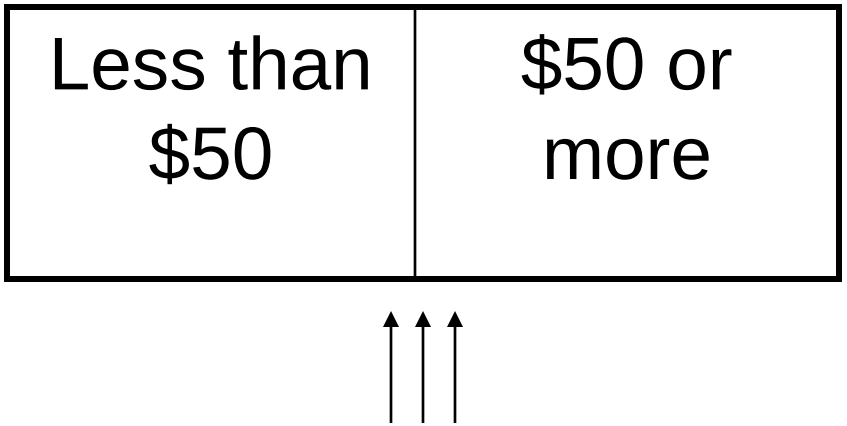
- “Bugs lurk in corners and congregate at boundaries.” - Boris Beizer,

- Programmers often make mistakes on the boundaries of the equivalence classes/input domain.
- As a result, we need to focus testing at these boundaries. This type of testing is called Boundary Value Analysis (BVA) and guides you to create test cases at the “edge” of the equivalence classes

Boundary value definition

- *Boundary value* is defined as a *data value that corresponds to a minimum or maximum input, internal, or output value specified for a system or component.*

Boundary value



- We should create test cases for $49,$50 and $51.
- These test cases will help to find common off-by-one errors, caused by errors like using >= when you mean to use >.

Boundary value

When creating BVA test cases, consider the following :

1. If input conditions have a range from **a** to **b** (such as a=100 to b=300), create test cases:
   - immediately below **a** (99)
   - at **a** (100)
   - immediately above **a** (101)
   - immediately below **b** (299)
   - at **b** (300)
   - immediately above **b**

Boundary value

2. If input conditions specify a *number* of values that are allowed, test these limits.

Bottom-up testing

- The bottom-up testing technique is an incremental testing approach where the lowest-level modules or system compo-nents are integrated and tested first.
- Testing then proceeds hierarchically to the top level. A driver, or temporary test program that invokes the test module or system component, is often required.

Bottom-up testing continued…

- Bottom-up testing starts with the lowest-level modules or system components with the drivers to invoke them.
- After these components have been tested, the next logical level in the program or system component hierarchy is added and tested driving upward.

Bottom-up testing continued…

- Bottom-up testing is common for large complex systems, and it takes a relatively long time to make the system visible.
- The menus and external user interfaces are tested last, so users cannot have an early review of these interfaces and functions
- A potential drawback is that it requires a lot of effort to create drivers, which can add additional errors

Cause effect graphing

- Cause-effect diagrams (also known as Ishikawa or Fishbone diagrams) are useful tools to analyze the causes of an un-satisfactory condition. They have several advantages
- One is that they provide a visual display of the relationship of one cause to another. This has proven to be an effective way to stimulate ideas during the initial search.

Cause effect graphing

- One application of cause-effect graphs was undertaken to understand the inspection process.
- It discovered that

  (1) excessive size of materials tobe inspected leads to a preparation rate that is too high,

  (2) a preparation rate that is too high contributes to an excessive rate of inspection, and

  (3) an excessive rate of inspection causes fewer defects to be found.

This analysis using cause-effect graphics provided insights to optimize the inspection process by limiting the size of materials to be inspected and the preparation rate.

Cause effect Methodology

1. Identify all the requirements.
2. Analyze the requirements and identify all the causes and effects.
3. Assign each cause and effect a unique number.
4. Analyze the requirements and translate them into a Boolean graph linking the causes and effects.
5. Convert the graph into a decision table.
6. Convert the columns in the decision table into test cases.

Example of Cause effect graphing

- A database management system requires that each file in the database have its name listed in a master index identifying the location of each file.
- The index is divided into ten sections. A small system is being developed that allows the user to interactively enter a command to displayany section of the index at the terminal.
- Cause-effect graphing is used to develop a set of test cases for the system. The specification for this system is explained in the following paragraphs

Specification

To display one of the ten possible index sections, a

command must be entered consisting of a letter and a digit.

The first character entered must be a D (for display) or an L (for list), and it must be in column

1. The second character entered must be a digit

   (0 through 9) in column

2. If this command occurs, the index section identified by the digit is displayed on the terminal. If the first character is incorrect, error message “Invalid Command” is printed. If the second character is incorrect, error message “Invalid Index Number” is printed.

Causes and Effects

Causes

1. Character in column 1 is D.
2. Character in column 1 is L.
3. Character in column 2 is a digit.

Causes and effects

Effects

1. Index section is displayed.
2. Error message “Invalid Command” is displayed.
3. Error message “Invalid Index Number” is displayed.

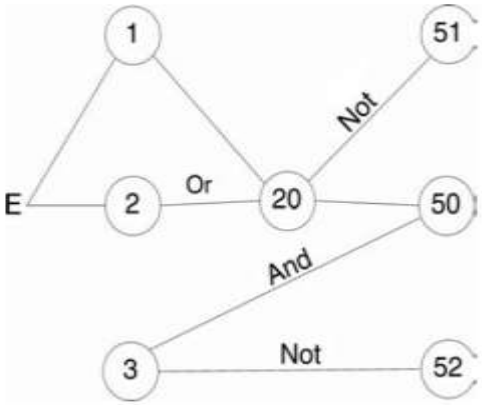
A Boolean graph is constructed through analysis of the specification. This is accomplished by

(1) representing each cause and effect by a node by its unique number;

(2) listing all the cause nodes vertically on the left side of a sheet of paper and listing the effect nodes on the right side;

(3) interconnecting the cause and effect nodes by analyzing the specification. Each cause and effect can be in one of two states: true or false. Using Boolean logic, set the possible states of the causes and determine under what conditions each effect is present; and

(4) annotating the graph with constraints describing combinations of causes and effects that are impossible because of syntactic or environmental constraints.



- Node 20 is an intermediate node representing the Boolean state of node1 or node 2.
- The state of node 50 is true if the states of nodes 20 and 3 areboth true.
- The state of node 20 is true if the state of node 1 or node 2 is true.
- The state of node 51 is true if the state of node 20 is not true. The state of node 52 is true if the state of node 3 is not true.
- Nodes 1 and 2 are also annotated with a constraint that states that causes 1 and 2 cannot be true simultaneously.

	Test Cases			
Causes	1	2	3	4
1	1	0	0	
2	0	1	0	
3	1	1		1
Effects				
50	1	1	0	0
51	0	0	1	0
52	0	0	0	1

(1) tracing back through the graph to find all combinations of causes that make the effect true for each effect,

(2) representing each combination as a column in the decision table, and

(3) determining the state of all other effects for each such combination. After completing this, each column in the figure represents a test case

Test Case Number	Input	Expected Results
1	D5	Index Section 5 is displayed
2	L4	Index Section 4 is displayed
3	B2	"Invalid Command"
4	DA	"Invalid Index Number"

- Cause-effect graphing can produce a useful set of test cases and can point out incompleteness and ambiguities in the requirement specification.
- It can be applied to generate test cases in any type of computing application when the specification is clearly stated and combinations of input conditions can be identified.
- Although manual application of this technique is tedious, long, and moderately complex, there are automated testing tools that will automatically help convert the requirements to a graph,decision table, and test cases.

CRUD

- A CRUD matrix, or process/data matrix,
- Is optionally developed during the analysis phase of application development, which links data and process models.
- It helps ensure that the data and processes are discovered and assessed. It identifies and resolves matrix omissions and conflicts and helps refine the data and process models, as necessary.
- The CRUD matrix is developed at the analysis level of development before the physical system or GUI (physical screens, menus,etc.) has been designed and developed.

- As the GUI evolves, a CRUD test matrix can be built
- It is a testing technique that verifies the life cycle of all business objects. each CRUD cell object is tested.
- When an object does not have full life cycle operations a“–” can be placed in a cell.
- CRUD stands for Create, Read, Update, Delete.

  -Data can be Created and added.

  -Data can be Accessed or Read.

  -Tester prepares CRUD matrix.

  -Tests object creation, reading, update and deleting objects.
- Creating test plans for create, read, update and delete (CRUD) functionality is a very common requirement.

Data Base Testing

- It does not matter at all whether it is web or desktop, client server or peer to peer, enterprise or individual business, database is working at backend. Similarly, whether it is healthcare of finance, leasing or retail, mailing application or controlling spaceship, behind the scene a database is always in action.
- for the applications with high frequency of transactions (e.g. banking or finance application), necessity of fully featured DB Tool is coupled.
- Currently, several **database tools** are available in the market e.g. MS-Access2010, MS SQL Server 2008 r2, Oracle 10g, Oracle Financial, MySQL, PostgreSQL, DB2 etc.

- All of these vary in cost, robustness, features and security. Each of these DBs possesses its own benefits and drawbacks. One thing is certain; a business application must be built using one of these or other DB Tools
- When the application is under execution, the **end user mainly utilizes the 'CRUD' operations facilitated by the DB Tool**.
- **C: Create** – When user 'Save' any new transaction, 'Create' operation is performed.
  R: Retrieve – When user 'Search' or 'View' any saved transaction, 'Retrieve' operation is performed.
  U: Update – when user 'Edit' or 'Modify' an existing record, the 'Update' operation of DB is performed.

- **D: Delete** – when user ‘Remove’ any record from the system, ‘Delete’ operation of DB is performed.

- It does not matter at all, which DB is used and how the operation is preformed. End user has no concern if any join or sub-query, trigger or stored-procedure, query or function was used to do what he wanted. But, the interesting thing is that all DB operations performed by user, from UI of any application, is one of the above four, acronym as **CRUD**.

How To Test Database:

1. Create your own Queries

In order to test the DB properly and accurately, first of all a tester should have very good knowledge of SQL and specially DML (Data Manipulation Language)

- statements. Secondly, the tester should acquire good understanding of internal DB structure of AUT. If these two pre-requisites are fulfilled, then the tester is ready to test DB with complete confidence.

2. Observe data table by table

If the tester is not good in SQL, then he or she may verify the result of CRUD operation, performed using GUI of the application, by viewing the tables (relations) of DB.

3.Get query from developer

This is the simplest way for the tester to test the DB. Perform any CRUD operation from GUI and verify its impacts by executing the respective SQL query obtained from the developer. It requires neither good knowledge of SQL nor good knowledge of application's DB structure.

Histograms

- A histogram is a graphical description of measured values organized according to the frequency or relative frequency of occurrence.
- In Exhibit G.39, the table consists of a sample of 100 client/server terminal response times (enter key until a server response) for an application. This was measured with a performance testing tool.
- The histogram in Exhibit G.40 illustrates how the raw performance data from the above table is displayed in a histogram. It should be noted that the design specification is for response times to be less than 3 seconds. It is obvious from the data that the performance requirement is not being satisfied and there is a performance problem.

Exhibit G.39. Response Time of 100 Samples (seconds)

2	4	1	6	5	12	4	3	4	10
5	2	7	2	4	1	12	4	2	1
1	2	4	3	5	1	3	5	7	12
5	7	1	2	4	3	1	4	1	2
1	3	5	2	1	2	4	5	1	2
3	1	3	2	6	1	5	4	1	2
7	1	8	4	3	1	1	2	6	1
1	2	1	4	2	6	2	2	4	9
2	3	2	1	8	2	4	7	2	2
4	1	2	5	3	4	5	2	1	2

Exhibit G.40. Response Time Histogram

Average = 3.47 seconds									
0	**23**	**25**	**10**	**16**	**10**	**4**	**5**	**2**	**5**
0 to.9	1 to 1.9	2 to 2.9	3 to 3.9	4 to 4.9	5 to 5.9	6 to 6.9	7 to 7.9	8 to 8.9	9 to ∞

- A histogram (or frequency distribution chart) is a bar graph that groups data by predetermined intervals to show the frequency of the data set. It provides a way to measure and analyze data collected about a process or problem, and may provide a basis for what to work on first.
- Histograms are also useful for displaying information such as defects by type or source, delivery rates or times, experience or skill levels, cycle times, or end user survey responses.

Gray Box Testing

- Black-box testing focuses on the program's functionality against the specification. White-box testing focuses on the paths of logic. Gray-box testing is a combination of black- and white-box testing.
- Gray-box testing is also known as *translucent testing.*
- The tester studies the requirements specifications and communicates with the developer to understand the internal structure of the system. The motivation is to clear up ambiguous specifications and "read between the lines" to design
- One example of the use of gray-box testing is when it appears to the tester that a certain functionality seems to be reused throughout an application. If the tester communicates with the developer and understands the internal design and architecture

many tests will be eliminated, because it may be possible to test the functionality only once.

- Another example is when the syntax of a command consists of seven possible parameters that can be entered in any order, as follows:
- Command parm1, parm2, parm3, parm4, parm5, parm6, parm7
- In theory, a tester would have to create 7! or 5040 tests. The problem is compounded further if some of the parameters are optional. If the tester uses gray-box testing, by talking with the developer and understanding the parser algorithm, if each parameter is independent, only seven tests may be required.

- The aim of this testing is to search for the defects if any due to improper structure or improper usage of applications.
- Gray-box testing is beneficial because it takes the straightforward technique of black-box testing and combines it with the code targeted systems in white-box testing

Application:

- Gray-box testing is well suited for **Web applications**. Web applications have distributed network or systems; due to absence of source code or binaries it is not possible to use white-box testing. Black-box testing is also not used due to just contract between customer and developer, so it is more efficient to use gray-box testing as significant information is available in Web Services Definition Language (WSDL).

- Gray-box testing is suited for **functional or business domain testing**. Functional testing is done basically a test of user interactions with may be external systems. As gray-box testing can efficiently suits for functional testing due to its characteristics; it also helps to confirm that software meets the requirements defined for the software

Inspections

- The inspection process was developed by Michael Fagan in the mid-1970s and it has later been extended and modified.
- Inspections are the most formal, commonly used form of peer review.
- The key feature of an inspection is the use of checklists to facilitate error detection.
- These checklists are updated as statistics indicate that certain types of errors are occurring more or less frequently than in the past.
- The most common types of inspections are conducted on the product design and code, although inspections may be used during any life cycle phase.

- Inspections should be short because they are often intensive; there fore, the product component to be reviewed must be small.
- Specifications or designs that result in 50 to 100 lines of code are usually manageable. This translates into an inspection of 15 minutes to one hour,
- although complex components may require as much as two hours. In any event, inspections of more than two hours are generally less effective and should be avoided.
- Participants are expected to study and make comments on the materials before the review.
- A primary goal of an inspection is to identify items that can be modified to make the component more understandable, maintainable, or usable.

- At the end of the inspection, an accept or reject decision is made by the group, and the coordinator summarizes all the errors and problems detected and gives this list to all participants.
- The individual whose work was under review (e.g., designer, implementer, tester) uses the list to make revisions to the component.
- When revisions are implemented, the coordinator

  and producer go through a minireview, using the problem list as a checklist.
- The coordinator then completes management and summary reports. The summary report is used to update checklists for subsequent inspections.

JAD's

- Technique that brings users and developers(IT Professionals) together to jointly design systems in facilitated sessions
- JADs stands for joint application design sessions
- JADs go beyond the one-on-one interviews to collect information.
- They promote communication, cooperation, and teamwork among the participants by placing the users in the driver's seat.

- JADs are logically divided into phases: customization, session, and wrap-up. Regardless of what activity one is pursuing in development, these components will always exist. Each phase has its own objectives**.**

Pareto Analysis

- A statistical technique for making decisions which is used for selecting a limited number of tasks which produce significant overall effect. Pareto Analysis uses the ‘Pareto Principle’ – an idea by which 80% of doing the entire job is generated by doing 20% of the work.
- When many possible courses of actions are completing the attention, the technique ‘Pareto Analysis’ is useful. In essence, the delivered benefit by each action is estimated by problem-solver, and selects the number of most effective actions which delivers the total benefit.

Random Testing

- Technique involving random selection from a specific set of input values where any value is as likely as any other

- Random testing is a technique in which a program or system is tested by selecting at random some subset of all possible input values
- It is not an optimal testing technique, because it has a low probability of detecting many defects.
- It does, however, sometimes uncover defects that standardized testing techniques might not. It should, therefore, be considered an add-on testing technique.
- Random testing is a form of functional testing
- Useful when the time needed to write & run directed tests is too long (or the complexity of the problem makes it impossible to test every combination).

Risk Based Testing

- The purpose of risk management testing is to measure the degree of business risk in an application system to improve testing.

- This is accomplished in two ways:
  - high-risk applications can be identified and subjected to more extensive testing.
  - risk analysis can help identify the error prone components of an individual application so that testing can be directed at those components.

- Risk analysis is a formal method for identifying vulnerabilities (i.e., areas of potential loss). Any area that could be misused, intentionally or accidentally, and result in a loss to the organization is a vulnerability.

- Risk-based testing is a technique in which test cases are created for every major risk factor that has been previously identified. Each condition is tested to verify that the risk has been averted.

- RBT is a type of software testing that prioritizes the tests of features and functions based on the risk of their failure.

Regression Testing

- Testing the modified build is know as Regression Testing
- Regression testing tests the application in light of changes made during a development spiral, debugging, maintenance, or the development of a new release.
- This test must be performed after functional improvements or repairs have been made to a system to confirm that the changes have no unintended side effects.
- Correction of errors relating to logic and control flow, computational errors, and interface errors are examples of conditions that necessitate regression testing.

- The purpose of regression testing is to confirm that a recent program or code change has not adversely affected existing features.

- Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that old code still works once the new code changes are done.

- Regression Testing is required
  - ✓ when there is a Change in requirements and code is modified according to the requirement
  - ✓ New feature is added to the software
  - ✓ Defect fixing
  - ✓ Performance issue fix

- **Regression Testing Techniques**
  - ❑ Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly . Therefore , Regression Testing becomes necessary. Regression Testing can be carried out using following techniques:
  - Retest All
    - **This is one of the methods for regression testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.**

- Regression Test Selection:
  - Instead of re-executing the entire test suite, it is better to select part of test suite to be run
  - Test cases selected can be categorized as 1) Reusable Test Cases 2) Obsolete Test Cases.
  - Re-usable Test cases can be used in succeeding regression cycles.
  - Obsolete Test Cases can't be used in succeeding cycles.

- Prioritization of Test Cases
  - Prioritize the test cases depending on business impact, critical & frequently used functionalities . Selection of test cases based on priority will greatly reduce the regression test suite.

Structured Walkthroughs

- Structured walkthroughs are more formal than the code-reading reviews.
- Distinct roles and responsibilities are assigned before the review.
- Another key feature of this review is that it is presented by the producer.
- The most common walkthroughs are those held during design and coding; however, recently they have been applied to specifications documentation and test results.
- The producer schedules the review and assembles and distributes input. In most cases, the producer selects the walkthrough participants (although this is sometimes done by management) and notifies them of their roles and responsibilities.

- The walkthrough is usually conducted with less than seven participants and lasts no more than two hours.
- If more time is needed, there should be a break or the product should be reduced in size.
- Roles usually included in a walkthrough are producer, coordinator, recorder, and representatives of user, maintenance, and standards organizations.
- Although the review is opened by the coordinator, the producer is responsible for leading the group through the product.
- In the case of design and code walkthroughs, the producer simulates the operation of the component, allowing each participant to comment based on that individual's area of specialization.

- A list of problems is kept, and at the end of the review, each participant signs the list, or other walkthrough form,
- Indicating whether the product is accepted as is, accepted with recommended changes, or rejected. Suggested changes are made at the discretion of the producer.
- If the walkthrough review is used for products throughout the life cycle, however, comments from past reviews can be discussed at the start of the next review.

Thread Testing

- Thread testing is a software testing technique that demonstrates key functional capabilities by testing a string of program units that accomplishes a specific business function in the application.
- A thread is basically a business transaction consisting of a set of functions.
- It is a single discrete process that threads through the whole system.
- Each function is tested separately, then added one at a time to the thread.
- The business transaction thread is then tested. Threads are in turn integrated and incrementally tested as subsystems, and then the whole system is tested.
- This approach facilitates early systems and acceptance testing.

Performance Testing

- Term often used interchangeably with “stress” and “load” testing. Ideally “performance” testing (and any other “type” of testing) is defined in requirements documentation or QA or Test Plans

- Verifies and validates that the performance requirements have been achieved; measures response times, transaction rates, and other time-sensitive requirements.
- It involves testing software applications to ensure they will perform well under their expected workload.

- Features and Functionality supported by a software system is not the only concern. A software application’s performance like its response time, do matter.
- Testing an application under heavy loads, such as testing of a Web site under a range of loads to determine at what point the system’s response time degrades or fail

The focus of Performance testing is checking a software program's :

- Speed – Determines whether the application responds quickly
- Scalability – Determines maximum user load the software application can handle.
- Stability – Determines if the application is stable under varying loads

Why do performance testing?

- Performance testing is done to provide stakeholders with information about their application regarding speed, stability and scalability.

- More importantly, performance testing uncovers what needs to be improved before the product goes to market.

- Without performance testing, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability

- Performance testing will determine whether or not their software meets speed, scalability and stability requirements under expected work loads.

Types of performance testing:

- **Load testing –** checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

- **Stress testing –** involves testing an application under extreme workloads to see how it handles high traffic or data processing .The objective is to identify breaking point of an application.

- **Endurance testing(Soak Testing) –** is done to make sure the software can handle the expected load over a long period of time.

- **Spike testing –** tests the software's reaction to sudden large spikes in the load generated by users.

- **Volume testing** – Under Volume Testing large no. of. Data is populated in database and the overall software system's behavior is monitored. The objective is to check software application's performance under varying database volumes.

Performance Test Tools

- There are a wide variety of performance testing tools available in market. The tool you choose for testing will depend on many factors such as types of protocol supported , license cost , hardware requirements , platform support etc. Below is a list of popularly used testing tools.

- HP Loadrunner **–** is the most popular performance testing tools on the market today. This tool is capable of simulating hundreds of thousands of users, putting applications under real life loads to determine their behavior under expected loads. Loadrunner features a virtual user generator which simulates the actions of live human users.

- HTTP Load - a throughput testing tool aimed at testing web servers by running several http or https fetches simultaneously to determine how a server handles the workload.

- Proxy Sniffer – one of the leading tools used for load testing of web and application servers. It is a cloud based tool that's capable of simulating thousands of users.

White Box Testing

- White-box testing, or structural testing, is one in which test conditions are designed by examining paths of logic.
- Also know as Clear Box testing, Transparent Testing and Glass box Testing
- White-box testing is a testing technique in which paths of logic are tested to determine how well they produce predictable results.
- Test cases are defined by examining the logic paths of a system
- For doing white box testing need the knowledge of the internal program structure and logic,
  - just as a mechanic knows the inner workings of an automobile.

- Specific examples in this category include basis path analysis, statement coverage, branch coverage, condition coverage, and branch/condition coverage,data flow testing, path testing.

Advantage:of white-box testing is that it is thorough and focuses on the produced code. Because there is knowledge of the internal structure or logic, errors or deliberate mischief on the part of a programmer have a higher probability of being detected.

Disadvantages: of white-box testing is that it does not verify that the specifications are correct; that is, it focuses only on the internal logic and does not verify the logic to the specification.

Pareto Analysis

- A statistical technique for making decisions which is used for selecting a limited number of tasks which produce significant overall effect. Pareto Analysis uses the ‘Pareto Principle’ – an idea by which 80% of doing the entire job is generated by doing 20% of the work.
- When many possible courses of actions are completing the attention, the technique ‘Pareto Analysis’ is useful. In essence, the delivered benefit by each action is estimated by problem-solver, and selects the number of most effective actions which delivers the total benefit.