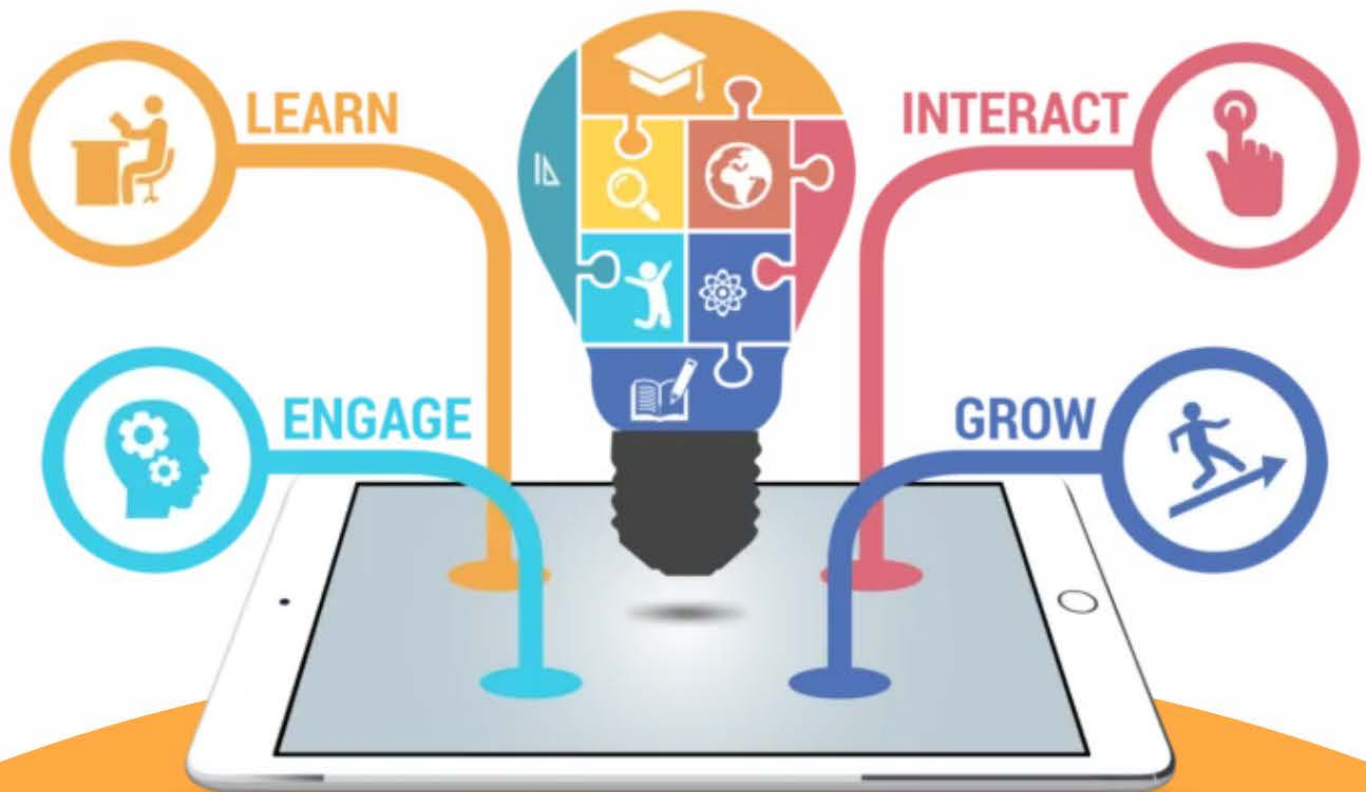


BSC-IT SEM 6



SOFTWARE QUALITY ASSURANCE



With lots of efforts, research, reviews we have launched the prerecorded series of academics for multiple universities, bundled in userfriendly application "The Shikshak"

The Shikshak EdTech

Students should do **topic-wise study** rather than question-wise study for several reasons:

1. **Comprehensive understanding:** Topic-wise study allows students to have a thorough understanding of a particular topic. It helps in building a strong foundation of knowledge on a subject. Once they have a good understanding of a particular topic, they can answer any question related to it.
2. **Efficient use of time:** When students study topic-wise, they can cover a range of questions related to a particular topic in one go. This way, they can utilize their time more efficiently instead of jumping from one question to another and losing focus.
3. **Better retention:** Studying a topic in-depth helps students retain the information for a longer time. It is because they learn the concepts in a logical sequence, making it easier for them to remember.
4. **Effective exam preparation:** Most exams are organized based on topics or units, so studying topic-wise will enable students to be well-prepared for the exam. They will have a good grasp of all the topics that will appear on the exam.
5. **Build analytical skills:** When students study topic-wise, they develop their analytical skills by understanding how various concepts in a subject connect with each other. This helps them develop a deeper understanding of the subject, making them better problem solvers.

In conclusion, studying topic-wise is more beneficial for students as it enables them to develop a better understanding of a subject, retain information better, utilize their time more efficiently, and be well-prepared for exams.

TheShikshak Edu App is an online learning platform that offers a range of resources and tools to help students pursuing BScIT and BScCS programs. Here are some ways in which TheShikshak Edu App can benefit BScIT and BScCS students:

1. **Comprehensive course material:** TheShikshak Edu App offers comprehensive course material for BScIT and BScCS students, covering all topics and concepts required in these programs.
2. **Track their progress :** analytics program helps student to know which topics are remaining and which are lowest watched lectures
3. **Expert guidance:** TheShikshak Edu App has a team of experienced instructors who provide expert guidance and support to students. Students can get their doubts clarified and receive personalized feedback on their performance.

Unit 1

Chapter 1: Introduction to Quality

- There are two general meanings of the term quality.
 1. “Quality means meeting requirements. These requirements must be measurable.”
- **Explanation:** If we consider this definition and think about quality as a binary term, the product can be either a quality product or not. This is the producer’s requirement of quality as meeting the producer’s requirement or specification.
 2. Quality is product meets customers’ needs and satisfaction. Also, we can say, “Fit for Use”.

Quality- A Historical Perspective

- Quality has changed from Inspection to Quality Control to Quality Assurance to Total Quality Management and now we have an excellent model of Six Sigma, ISO-9001, and ISO 9001-9015 and so on.
- **1920-1950 Inspection (Craftsman, Walter Shewhart):**
 - 1920
 - Walter Shewhart developed Statistical Process Control (SPC) and use of Control Chart. Use of Control chart was less initially but now backbone of Six Sigma.
 - 1946
 - American Society established ASQ (American Society for Quality)
- **1950-1970: Quality Control (Deming, Feigenbaum & Juran):**
 - Feigenbaum introduced the concept of quality Cost.
 - Juran published Quality Control Handbook
 - Deming trained hundreds of Japanese engineers in Statistical process control and concept of quality.
 - Kaouru Ishikawa introduced seven tools of quality.
 - Quality management in Japan (Deming & Juran).
 - Developed Quality tools.
 - Quality becomes strategic (USA).
- **1970-1990 Quality Assurance (Philip Crosby):**
Introduced LEAN,
 - 1979 – Philip Crosby published “Quality is Free”. Better Productivity Higher Sell
 - 1986 – Motorola introduced Six Sigma considered Top Quality product solving problem projectby project.
 - 1988 – Two Quality awards were introduced. MBNQ Malcolm Baldrige National Award
 - 1991 – European Foundation for Quality Management.
- **2000-2018 International Standard Organization:**
 - 2000: Introduced third edition of ISO 9001
 - 2008: Introduced the fourth issue of ISO 9001
 - ISO 9001: 2015: Introduced the fifth issue of ISO 9001

- 2013: Tie up with Motorola Learning Solution
- 2014: Introduction of TQML's own six sigma Master Black Belt Belongs to product and services.

Quality: A Fact or Perspective:

- Quality requires commitment from the top management to staff.
- There is a belief that quality can be achieved by assessing only one completed product.
- But it can be used to prevent quality defects or deficiencies and make the product accessible by quality assurance measures.
- Quality is an abstract perception, and it has quantitative measures. $Q = P/E$
Where, **P: Performance** (measured by manufacturing groups) and **E: Expectation** of customers.

Verification versus Validation:

Core Components of Quality:

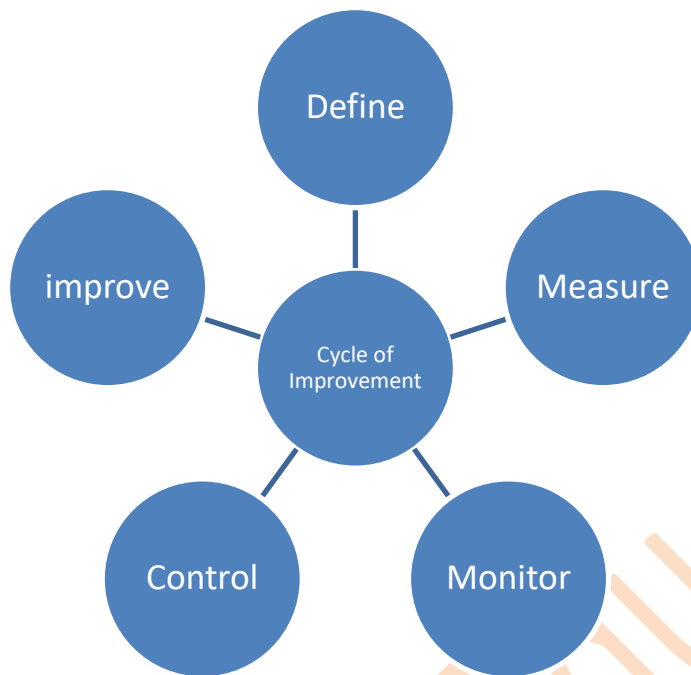
1. Quality is based on Customer Satisfaction.
2. The organization must define quality parameters before it can be achieved.
3. Management must lead the organization through improvement efforts.
4. Continuous process improvement necessary.

1. Quality is based on Customer Satisfaction:

- It talks about the ability of the product/service to satisfy customer needs/requirement/purpose.

2. The organization must define quality parameters before it can be achieved:

- If the customer requirements are clear, it becomes easier for the manufacturer to provide quality product.
- If the quality product is to be delivered and if it is defined in measurable terms, it helps the manufacturer to determine whether the quality product is achieved or not.
- If improvement is needed, we need to follow the following cycle of improvement



Define:

- Feature, functionalities and attribute of the product must be defined in quantitative terms and it must be part of requirement specification.
- Supplier must know 'Must be', what 'Should be' and what could be present in the product.

Measure:

- Measurement gives gap between what is expected by the customer and what is delivered to him.

Monitor:

- There should be some mechanism to monitor the progress used in development, testing and delivering product to the customer and there outcome.
- Deviation from the specification must be analyzed and reason of these deviations must be sorted out to improve quality of product.

Control:

- Ability to provide desired results and avoid unwanted things going to the customer.
- Controlling function called quality control or verification and validation, overall organizational control is entrusted with the management called quality assurance.

Improve:

- In order to maintain customer satisfaction continuous improvements are necessary.

3. Management must lead the organization through improvement efforts:

- Quality of product should be driven by management and participated by employees.
- Quality improvement can be brought in to the organization by means of vision, mission, policies, objectives, strategies, goals and values of the organization.
- Every word and action of the management should be seen and adapted by the employees,
- Quality improvement is also termed as "Cultural change brought in by management"

4. Continuous process improvement necessary:

- For improving the competitive cost advantage to producer as well as customer, quality must be produced with an aim of first time right and must be improved continuously.

Quality View

- Quality is viewed differently by the various stakeholders. There are six different categories of stakeholders. They are affected directly/indirectly. They get benefitted if the project is successful and suffer if project fails. These six categories are:



1. **Customer:** Main stakeholder for any project who pays to satisfy his requirements. Interested in product delivery with all features on defined scheduled time.
2. **Supplier:** Gives inputs for making our project. When more and more project are executed by the organization it becomes successful and suppliers make more business, profit and expansion.
3. **Employee:** People working on a project in an organization is termed as employee.
4. **Management:** People who manage the organization or project is termed as management. Management is divided into project management, staff management, senior management investors, etc. Management is concerned about more profit, recognition, turnover improvement.
5. **Society:** Successful organization generate more employment and wealth for the people. It also affects the resource availability at local as well as global level like water, roads, power supply etc.
6. **Government:** Government may get higher taxes, export benefits, foreign currency etc. from successful organization. People living in these areas get employment and overall wealth for nation improves. Also there may be pressure on resources like water, power etc.

Financial Aspect of Quality

- Price is decided by the manufacturer. While deciding price cost of inspection, testing, sorting is considered.
 - Sales price can be defined as: **Sales Price = Cost of manufacturing + Cost of Quality + Profit**
 - Cost decides the quality of sales, reduced price might get more volume of sales.
 - Profit = Sales Price – (Cost of Manufacturing + Cost of Quality)
 - There are four types of costs associated while ensuring a quality product:
 1. Cost of Manufacturing
 2. Cost of Prevention
 3. Cost of Appraisal
 4. Cost of Failure
1. **Cost of Manufacturing:** Cost required for developing right product by right method at the first time. Cost involved in requirement analysis, designing, development, other resources and coding. Cost can be reduced through improvements in technology and productivity.
 2. **Cost of Prevention:** Investment by an organization by defining process, guidelines, standards for development and testing. This is termed as “Green Money.” Generally it is believed that “1 part of cost of prevention can reduce 10 part of cost appraisal 100 part of cost of failure.”
 3. **Cost of Appraisal:** The cost incurred in first time reviews and testing is called as cost of appraisal. This is termed as “Blue Money.” 1 part of cost of appraisal can reduce 100 parts of costs of failure.
 4. **Cost of Failure:** Rework, retesting, sorting, scrapping, regression testing, sales under concession etc. represent cost of failure. It is termed as “Red money.” It affects profit.

Customers, Suppliers and Processes

- Every organization has suppliers supplying the inputs and customers who are buying the outputs produced. Suppliers and customers are internal as well as external to the organization.

Internal Customer:

- Functions and projects serviced and supported by some other functions and projects. Each function must try to fulfill its customer's requirements.
- If internal customers are satisfied they will satisfy external customers.

External Customers:

- External people to the organization who will be paying for the services offered by the organization.
- These are people who will be buying from the organization.

Total Quality Management

- Total Quality Management is an extensive and structured organization management approach that focuses on continuous quality improvement of products and services by using continuous feedback.
- In TQM effort all members of an organization participate in improving processes, products, services and the culture in which they work. Emphasis is given on fact-based decision making coupled with performance metrics to monitor progress.

1. **Customer-focused:** The customer determines whether the efforts took by the organization to

[For detailed Video Lecture Download The Shikshak Edu App](#)

maintain quality are worthwhile or not.

2. **Total Employee Involvement:** All employees are working towards a common goal.
3. **Process-centered:** A process is a series of steps take inputs from suppliers (internal or external) and transforms them into outputs that are delivered to customers (internal or external).
4. **Integrated System:** Micro processes add up to form a large business process required for defining and implementing strategy.
5. **Strategic and Systemic Approach:** Includes the formulation of strategic plan that integrates quality as a core component.
6. **Continual Improvement:** Continual improvement drives an organization to be both analytical and creative in finding ways to become more competitive and more effective at meeting stakeholder expectations.
7. **Fact based decision making:** TQM requires that an organization continually collects and analyze data in order to improve decision making accuracy, achieve consensus and allow prediction based on history.
8. **Communications:** Effective communication plays a crucial role in maintaining morale and motivating employees at all levels. Communication involves strategies, methods and timelines.

Quality Management through Statistical Process

Method of measuring and controlling quality through monitoring the manufacturing process. Quality data is collected in the form of product or process measurements or readings from various machines or instruments. The data is collected and used to evaluate, monitor and control a process. SPC is an effective method for continuous improvement. There are three part of approach:

1. Quality planning at all level.
 2. Quality control.
 3. Quality improvement.
1. Quality Planning at all level:
 - Quality improvement must be planned at all organizational levels. Basically it happens at two levels quality planning at organizational and quality planning at unit level.
 - **Quality planning at organizational level:** Quality must be planned based on the policy definition and strategic quality plans based on the basis of vision, mission. The needs of the present or future must be expressed in numeric form to be measurable.
 - **Quality planning at unit level:** Done by the people responsible for managing the unit. Project plan and quality plan at unit level must be consistent with the strategic quality plan at organizational level.
 1. **Quality Control:** Examines product at various levels as per the defined standard. Removing defect improve their capability help to reach new level of improved quality.
 2. **Quality Improvement:** Continuously improve the quality of the processes used for producing the products.

How to Use SPC?

- The manufacturing process must be evaluate to check the main areas of waste before

implementing SPC.

- Examples of manufacturing process waste are rework, scrap and excessive inspection time. One should apply SPC tools to these areas first.
- During SPC, all dimensions cannot be monitored due to expense, time and production delays that can occur.
- Prior to SPC implementation the key or critical characteristics of the design or process should be identified by a cross functional team (CFT).

Quality Management through Cultural Changes

- To maintain quality culture over the course of time one must make the following cultural changes in the organization :
 1. Maintain an awareness of quality as a key critical issue.
 2. There is plenty of evidence of management's leadership.
 3. Empower employees and encourage self-development and self-initiative.
 4. High level support essential
 5. Recognize and reward the behaviors that tend to nurture and maintain quality culture.
 6. Management lead by example.
 7. Make changes so attractive that people want to be a part of it.
 8. Changes to be grafts of new values to old culture.
 9. Three sets of attributes associated with culture, time and effort.
 10. Change must be measured in years, not weeks or months.

Continuous (Continual) Improvement Cycle:

- Continuous improvement is initially invented to make changes and improvements in the existing systems to produce better outcomes.
- In order to do so we have devised an approach **PDCA (Plan-Do-Check-Act)**. PDCA is an iterative, four-stage approach for continually improving processes, products or services and for resolving problems.
- Involves systematically testing possible solutions, assessing the results and implementing the ones that have shown to work. Also called **PDSA(Plan-Do-Study-Act)**, Deming cycle, Shewhartcycle

Plan -

- Identify the problem.
- Collect relevant data.
- Understand the problems root cause.
- Develop a hypothesis for what issues may be.
- Decide which one to test.

Do –

- An organization must work in direction set by the plan in earlier phase for improvements.
- Develop and implement a solution; decide upon a measurement and its effectiveness, test the potential solution and measure the results.

Check –

- Compare the outcomes of do stage with reference or expected results.
- Done periodically to access the direction.
- Confirm the results before and after data comparisons.
- Study the result measure effectiveness and decide whether the hypothesis or plan is supported or not.

Act –

- If any difference is observed in actual outcomes the organization may need to take proper actions to resolve it.
- Document the results.
- Inform others about process change.
- Make recommendations for future PDCA cycles.
- If the solution was effective implement.
- Tackle the next problem and repeat the PDCA cycle.

Benchmarking and Metrics

- Benchmarking is referred to as the process by which an organization measures their products, services, and practices against its most difficult competitors.
- Benchmarking is the process of measuring an organization's internal processes then identifying, understanding, and adapting outstanding practices from other organizations considered to be best-in-class.
- Measuring our performance against that of best-in-class companies, determining how the best-in-class achieve those performance levels and using the information as a basis for our own company's targets, strategies and implementation.

Why Benchmarking?

- Promoting improvements in performance.
- Establish Competitiveness.

Process of Benchmarking:

- Organizations that benchmark, adapt the process to best fit their own needs and culture.
- The number of steps may vary depending upon the requirements.
- There are generally six steps involved into it:
 1. What to Benchmark?
 2. Understand Current Performance
 3. Plan
 4. Study Others
 5. Learn from Data
 6. Use findings

Quality Metrics:

- Quality metrics should be measurable, actionable, tractable, maintained updated and tied to business goals. Metrics, objective standards for measuring your product and the quality and efficiency of the manufacturing process.

- Performance metrics assess the organization's overall performance in a wide range of areas (these metrics are most closely tied to outputs, customer requirements, and business needs for the process).
- Diagnostic metrics are internally focused and usually associated with internal process steps and inputs received from suppliers. The good metrics should be specific, measurable, actionable, relevant and timely.

Problem Solving Techniques

- In TQM problem solving, there are 7 steps involved falling into the categories Plan, Do, Check and Act.
- The problem solving process is a logical sequence for solving problems and improving the quality of decisions. Also acts as a guide to identifying which tools and techniques to apply. The steps are as follows :
 1. Producing a clear statement of the identified problem/opportunity.
 2. Gathering all necessary information associated with a problem/opportunity.
 3. Analyzing collected data and providing a clear statement of the root causes of a problem or the benefits of an opportunity.
 4. Producing a list of all potential solutions to a problem/opportunity.
 5. Selecting the best solution to fit the problem/opportunity.
 6. Implementing and testing a plan.
 7. Establishing a process for continuous improvement and holding the gains.

Implementation/Application of Problem-Solving Process: The problem solving process can be applied to any problem or deviation from requirements but can also be used to tackle an opportunity.

Problem Solving Software Tools:

- There are two types of problem-solving tools:
 1. Statistical Tools
 2. Non-Statistical Tools
- 1. **Statistical Tools:**
 - Includes Pareto Chart, pie Chart, bar chart, run chart, scatter Plot, histogram, control Chart, balance Scoreboard.
- 2. **Non-Statistical Tools:**
 - It includes Checklist, Check sheet, Brainstorming, Motivating, Cause & Effect Diagram, Flowcharting, Force Field Analysis, Tree Diagram and Relationship Diagram.

Chapter 2: Software Quality

Constraints of Software Product Quality Assessment

- It Requirement specification is done by business analysts and system analysts.
- Tester may or may not have direct access to the customer may get information through requirements' statements, queries etc.
- To overcome the limitations of in this scenario software industry decide similarities between

[For detailed Video Lecture Download The Shikshak Edu App](#)

anytwo products.

- All aspect of software cannot be tested fully as of number of permutation and combinations.
- Exhaustive testing is neither feasible nor justifiable with respect to cost.
- There are numerous possibilities and all of them may not be tried in the entire life cycle.

Customer is King

- Customer is the most important asset in any process of developing a product and using it. Anyone who is provided with a good, product, service or idea is a customer. "Customer is king because he keeps every business afloat."
- "Customers are generally a key to your business success. " Whether an organization offers a product or service, it cannot remain in business if it cannot find a group of people willing to become its customers.
- Factors determining the success: To be a successful organization one must consider following factors:
 1. Internal Customer & Internal Supplier
 2. External Customer & External Supplier
 - **Internal Customer and Internal Supplier:** The principle of TQM is 'Internal Customer Satisfaction.'
 - **External Customer and External Supplier:** External customer may be final users. External suppliers are the entities who are external to the organization and who are supplying to the organization.

Requirements of Product

- There are 3 types of requirements:
 1. **Stated/Implied Requirements**
 2. **General/Specific Requirements**
 3. **Present/Future Requirements**
 1. **Stated/Implied Requirements:** Some requirements' are specifically documented in requirement specifications while few others are implied ones. Functional and non-functional requirements specified by a customer and business analyst. As a part of development team one must understand stated as well as intended/implied requirements from the users.
 2. **General/Specific Requirements:** Some requirements are generic in nature while some are very specific for the product. Many times generic requirements may not be mentioned in requirement specification while specific requirements are stated ones as those are present in requirement specifications.
 3. **Present/Future Requirements:** When an application is used in present circumstances while future requirements are for future needs. Present as well as future requirements may be specifically define by customer business/system analysts. On the basis of user's perspective, requirements categorized in different ways as follows:
 - i. **Must' and Must Not' Requirements or Primary Requirements:** This requirements denoted by priority "P1" indicating highest priority also known

as Primary Requirements. Value of the product is defined on the basis of 'Must' requirements

- ii. 'Should be' and 'should Not be' requirements or Secondary Requirements: This requirements denoted by 'P2' indicating lower priority requirement than 'Must' requirement. This requirements appreciated by customer if they are present in product. Customer may pay little bit extra for the satisfaction of these requirements.
- iii. 'Could be' and 'couldnot be' Requirements or Tertiary requirements: This requirements known as 'P3' indicating lowest priority. 'Could be' requirements are requirements which may add a competitive advantage to the product. But may not add value in terms of price paid by a customer. If two products have everything same then 'could be' requirements may help in better appreciation of product by users.

Organization Culture

- Culture of an organization is an understanding of the organization virtue about its people, customers, suppliers and all stockholders. Quality improvement program is based on are based on the ability of the organization to bring about the changes in culture.

Characteristics of Software

- **Transferability:** How easily a new team or team member can become productive when assigned to work on the application.
- **Changeability:** How easily and quickly an application can be modified.
- **Robustness:** Risks of failures or defects that can result by changing an application.
- **Performance:** Risk of performance issues of an application based on architectural designs.
- **Security:** Risk of security breaches for an application.
- **Maintainability Index:** Cost and difficulty/ease to maintain an application in future.
- **Technical Size:** Gives the technical size understood by developers, testers or integrators. Measured through lines of codes, Number of Programs, Number of forms, Number of Classes, Number of Function Points, Methods...
- **Functional Weight:** Size of an application from the user standpoint. The number of "features" provided to the users. Based on Lines of Code and/or on the application architecture related measures and application statistical metrics.

Software Development Process

- A software development process is the process of dividing software development work into various phases to improve design, product management, and project management.
- Also known as a software development life cycle (SDLC). The stages in SDLC are given below:
 1. **Planning:** Planning gives us an overview of management activities that determine the specific goals and allocate adequate resources for the various phases of development.
 2. **Analysis:** The performance of the software at various stages is analyzed and notes on additional requirements are made.

3. **Design:** Once the analysis is complete, the step of designing takes over which is basically building the architecture of the project.
4. **Development & Implementation:** The actual task of developing the software starts here with data recording going on in the background. Once the software is developed, implementation comes in where the product goes through a pilot study to see if it's functioning properly.
5. **Testing:** The testing stage assesses the software for errors and documents bugs if there are any.
6. **Maintenance:** Once the software passes through all the stages without any issues, it is to undergo a maintenance process wherein it will be maintained and upgraded from time to time to adapt to changes.

Following are the process model:

1. Waterfall Development Model.
2. Iterative Development Model/Incremental Development Model
3. Spiral Development Model.
4. Prototyping Development Model.
5. Rapid Application Development Model
6. Agile Model

Water fall Development Model:

- Also referred to as a linear-sequential life cycle model.
 - Each phase must be completed fully before the next phase.
1. **Requirements:** The first phase involves understanding what needs to be design and what is its function, purpose etc.
 2. **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and helps in defining overall system architecture.
 3. **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase.
 4. **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing each unit. Testing is done so that the client does not face any problem during the installation of the software.
 5. **Deployment of System:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.
 6. **Maintenance:** This step occurs after installation and involves making modifications to the system or an individual component to alter attributes or improve performance. The client is provided with regular maintenance and support for the developed software.

Iterative Development Model (Incremental Model):

- In incremental model the whole requirement is divided into various builds.
- Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle.

- Cycles are divided up into smaller, more easily managed modules.
- In this model, each module passes through the requirements, design, implementation and testing phases.
- A working version of software is produced during the first module, so you have working software early on during the software life cycle.
- Each subsequent release of the module adds function to the previous release.
- The process continues till the complete system is achieved.

Spiral Model:

- The spiral model is similar to the incremental model placed on risk analysis.
 - The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.
 - A software project repeatedly passes through these phases in iterations (called Spirals in this model)
 - The baseline spiral, starting in the planning phase, requirements is gathered and risk is assessed.
 - Each subsequent spiral builds on the baseline spiral.
- The following steps are involved here:
1. **Planning Phase:** Requirements are gathered during the planning phase.
 2. **Risk Analysis:** In the risk analysis phase, a process is undertaken to identify risk and alternate solutions.
 3. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.
 4. **Engineering Phase:** In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.
 5. **Evaluation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Prototyping Development Model :

- Prototype model is developed based on the currently known requirements.
- Prototype model is a software development model
- By using this prototype, the client can get an "actual feel" of the system.
- Since the interactions with prototype can enable the client to better understand the requirements of the desired system.
- Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.
- The prototype is usually not complete systems and many of the details are not built in the prototype.
- The goal is to provide a system with overall functionality.

RAD (Rapid Application Development) Model:

- RAD or Rapid Application Development process is an adoption of the waterfall model.
- It targets at developing software in a short span of time.
RAD follows the iterative
- It focuses on input-output source and destination of the information.

- It emphasizes on delivering projects in small pieces.
- The larger projects are divided into a series of smaller projects.
- The main features of RAD model are that it focuses on the reuse of templates, tools, processes, and code.

SDLC RAD model has following phases:

1. Business modeling.
2. Process Modeling
3. Testing and Turnover
4. Data Modeling
5. Application Generation

Agile Model:

- Agile development model is also a type of Incremental model.
- Software is developed in incremental, rapid cycles.
- This results in small incremental releases with each release building on previous functionality.
- Each release is thoroughly tested to ensure software quality is maintained.
- It is used for time critical applications.
- Extreme Programming (XP) is currently one of the most well-known agile development life cycle model.

Types of Product

- There are various scheme of grouping the product on the basis of criticality to the users few of them are listed below:
 1. Critical Product
 2. Second Critical Product
 3. Third Rank Critical Product
 4. No Criticality

Critical Product/Life Affecting Products:

- Some Product directly or indirectly affect human life and are considered as most critical products. Testing is very critical for such product as failures may result into loss of life, this type of product further classified into 5 different categories.
 1. Failure of product resulting into death of person. (Most critical product)
 2. Failure of product which may cause permanent disablement to a patient (Second Level Critical Product)
 3. Any product failure may cause temporary disablement to patient.
 4. Some product does not affect health or safety.
 5. Any product failure which may cause minor injury.

Product Affecting Huge Sum of Money:

- Some product has direct relationship with loss of huge sum of money.
- Such products require large testing efforts. Security, confidentiality and accuracy.
- Such product requires high confidence level and huge testing.
- For Example: Ecommerce and E-Business

Product which can be tested only by Simulators:

- Product which cannot be tested in real life scenario simulated environments for testing are third in ranking of criticality
- Such product needs huge testing although lesser than the earlier two types
- For Example, Product used in aeronautics, space research etc.

Other Products/ No criticality:

- All other product which cannot be categories in any of the above scheme may be put in these categories.
- For Example: Auto piloting software
- It cannot be tested in real environments.
- From user perspective product failure which disrupts the entire business.

Problematic Areas of Software Development Life Cycle

- Requirement phase introduces maximum defects in the product. Problems associated with requirement gathering are, requirements are not easily communicated.
- Many times, the customer may not understand the benefit of selecting a specific technology over the other option and problems of using these technically specified configurations.
- There may be numerous business rules which are defined by customers or users for doing business. Development team must understand the rules and regulations applicable for a particular product and business.
- Operational Requirements these may be functional as well as non-functional requirements. They tell the development team, what the intended software must do/must not do when used by the normal user.
- System requirements specific security requirements such as strong password encryption, and privileges definitions, which are also declared by the customer.
- Requirements change very frequently requirements are very dynamic in nature. There are many complaints by development teams that requirement change is very frequent.

Software Quality Management

- Quality management involves management of all inputs and processing to the process defined
- It talks about three levels of handling problems, namely,
 1. Correction
 2. Corrective Actions
 3. Preventive
- 1. **Correction:** Talks about the condition where defects found in the product or service are immediately sorted and fixed. This is a natural phenomenon which occurs when a tester defines any problem found during testing. This is mainly a quality control approach.
- 2. **Corrective Actions:** Every defect needs an analysis to find the roots' causes for introduction of a defect in the system. Situation where the root cause analysis of the defect is done and actions are initiated to remove the root causes so that the same defect does not recur in future are termed as corrective action.

This is a quality assurance approach where process-related problems are found and resolved to avoid recurrence of similar problems again and again.

3. **Preventive Actions:** Preventive action means that there are potential weak areas where defects have not been found till that point. But there exists a probability of finding the defect. In this situation, similar scenarios are checked, and actions are initiated so that other potential defects can be eliminated before they occur.

Quality management is a set of planned and systematic activities which ensures that the software processes and products conform to requirements, standards and process defined by management, customer, and regulatory authorities. The output of the process must match the expectations of the users.

Why Software has Defects?

- Sometimes requirements changes are very dynamic.
- Technologies are responsible for introducing few defects.
- There are many defect introduced due to browsers, platforms, database, etc.
- Customer may not be aware of all requirements and ideas develop as product is used
- "Prototyping is used for clarifying requirements to overcome the problem to some extent.

Process Related To Software Quality

- **Vision:** Vision defines what the organization wishes to achieve in the given time horizon. Every organization must have a vision statement. Clearly defining the ultimate aim.
- **Mission:** Success of all these mission is essential for achieving the organization vision. Mission are expected to support each other to achieve the overall vision. Mission may have different lifespan and completion dates.
- **Policy:** Policy Statement talks about a way of doing business as defined by senior management. It helps the employees, suppliers, and customers to understand the thinking of management. Example of policy may be security policy, quality policy and human resource development policy.
- **Objectives:** objectives define an expectation from each mission can be used to measure the success /failure at the end of define time period.
- **Strategy:** It defines the way of achieving a particular mission. Policy is converted into actions through strategy.
- **Goals:** Goals define the milestone to be achieved to make the mission successful.
- **Values:** value can be defined as the principles or way of doing a business by the management. 'Treating the customer with Courtesy' can be a value for an organization.

Quality Management System Software

1. **Quality Policy**
2. **Quality Objective**
3. **Quality Manual**

1. **1st Tier Quality Policy:** Quality Policy sets the wish, intent and direction by the management about how activities will be conducted by the organization.
2. **2nd Tier Quality Objectives:** Is used to define progress and achievements in numerical way. An improvement in quality determines highly achievement. This objective must be compared with

[For detailed Video Lecture Download The Shikshak Edu App](#)

planned levels expected and results and deviation.

3. **3rd- Tier -Quality Manual:** Quality manual also termed as policy manual is established and published by the management of the organization. It sets the framework for other processes definitions, and is a quality planning at organization level.

Pillars of Quality Management System

There are 7 Pillars of TQM:

P1: Creation of quality management:

- Environment Committed leadership.
- Identification of starting point Communication of TQM to employees.

P2: Development of teamwork:

- Development of cross functional working teams.
- Effective information sharing.
- Involvement employees in decision making.

P3: Practice of quality control tools and techniques:

- Educate the employees about Statistical Process Control (SPC) tools and techniques.
- Practice of Failure Mode and Effect Analysis (FMEA).
- Adoption of Total Productive Maintenance (TPM).

P4: Focus on customer satisfaction:

- Customer feedback Employee motivation.
- Focus on supplier relationship.
- Identification of the best suppliers.
- Development of trust-based relationship.
- Adoption of advanced resources.

P5: Benchmarking:

- Identification of deviation from the set targets.
- Identification of best practice.
- Implementation of the best practices.

P6: Improvement of processes:

- Identification of process deviation
- Development of zero defect mentality
- Implementation of PDCA cycle
- Adoption of refinement and renovation

P7: Involvement of employees:

- Recognition and rewards
- Employee empowerment
- Training and development

Important Aspects of Quality Management

The following are the important aspects of Quality Management:

- Quality planning at organizational level

- Quality planning at project level
- Resource management
- Work environment
- Customer related processes
- Quality management system document and data control
- Verification and Validation
- Software Project Management
- Software Configuration Management
- Software Metrics and Measurement
- Software Quality Audits
- Subcontract Management
- Information Security Management
- Management Review

TheShikshak Edu App

Unit 2

Chapter 1: Fundamentals of Testing

Definition of Testing

- Execution of a work product with intention to find a defect.
- The primary goal of testing is to find hidden defects so that they can be fixed.

Necessity of Testing

- Different entities are involved in different phases of software development. All entities work may not be matching with each other. Gaps between requirement design and coding may not be traceable unless testing is performed.
- Developers may have excellent skills of coding but integration issues can be present when different units do not work together. One must bring individual units together and make the final product. There is a possibility of blindfold. Testers have to challenge each assumption and decision taken during development.
- Development people assume that whatever they have developed is as per customer requirements and will always work. But it is imperative to create real life scenario and undertake actual execution of a product at each level of software building to assess whether it really works or not.
- Analysis of customer expectations.

What is testing? Definition of Testing & What it includes?

- Testing is defined as “Execution of a work product with intent to find a product.”

Testing Process may Include:

- An activity of identification of the difference between expected results and actual results produced, during execution of software applications. This difference may be due to the possibility of a defect in the process and/or work product.
- Process of executing a program with the intention of finding defects. It is expected that these defects may be fixed by the development team during correction, and the root cause of the defects are also found and closed during correction activity.
- Requirement mismatches must be detected by testing.
- Establish confidence that a program does what it is supposed to do.
- Acceptance testing is an activity defining whether the software has been accepted or not.
- Process of evaluating processes used in software development. Every failure/defect indicates a process failure. This can be used to improve development processes.
- Verifying that the system satisfies its specified requirements as defined and is fit for normal use.
- Confirming that program performs its intended functions correctly.

- Software testing is the process of analyzing a software item to detect the difference between existing and required conditions and to evaluate the feature of the software item.

Approaches to Testing

There are two approaches of testing :

1. Big Bang Testing
2. Total Quality Management
 1. **Big Bang Approach** : In Big-Bang testing all the modules of the software are integrated simultaneously thus creating a big collection of modules and all the modules are tested together, therefore the name Big-Bang.

Characteristics :

- Involves testing software system after development work is completed.
- Testing is the last part of software development as per waterfall methodology.
- It may not be able to detect all defects as all permutations and combinations cannot be tested in system testing due to various constraints like time

Disadvantages :

- Defects present at the interfaces of components are identified at very late stage as all components are integrated in one shot.
- It is very difficult to isolate the defects found.
- There is high probability of missing some critical defects which might pop up in the production environment.

2. **Total Quality Management**: An approach that seeks to improve quality and performance which will meet or exceed customer expectations.

Total Quality Management(TQM) as against the Big Bang Approach :

- **Stage 0**: There may be a stage of maturity in an organization where no verification/validation is required to certify the product quality. The cost involved is 'zero' or the benefits derived by investment in process definition, optimization and deployment make quality free. Quality is free
- **Stage 1**: Even if some defects are produced during any stage of development in such quality environment, then an organization may have very good verification processes which may detect the defects at the earliest possible stage and prevent defect percolation
There will be a small cost of verification and fixing, represented by 10'. This is the cost which reduces the profitability of the organization due to scrap, rework and re-verification.
- **Stage 2**: If some defects escape the verification process, still there are capable validation processes for filtering the defects before the product goes to a customer. Cost of validation is much higher than verification. This cost is represented by ' 100' One may have to go to the stage where defect was introduced in the product and correct all the stages from that point onward till the defect-detection point. The cost is much higher but till that point of time the defect has not reached the customer, it may not affect customer feelings or goodwill.

- **Stage 3:** At the bottom of the pyramid, there is a highest cost associated with the defects found by customer during production or acceptance testing. This cost is represented by 1000 There may be customer complaints, loss of goodwill, etc.

TQM in Cost Perspective

- Total Quality Management (TQM) aims at reducing, the cost of development and cost of quality through continual improvement. TQM defined the cost incurred in development divides the quality into three parts:
 1. **Green Money also known as Cost of Prevention**
 2. **Blue Money also known as Cost of Appraisal**
 3. **Red Money also known as Cost of Failure**
- **1. Green Money /Cost of Prevention:** Green Money is considered as an investment by the organization in doing quality work. It is a cost spent in definition of processes, training people, developing foundation for quality etc. It gives return on investment. If an organization has defined and optimized processes trained people and fixed guidelines and standards for doing work which are followed, then the return on such investment can be seen in terms of less inspection and testing and higher customer satisfaction with repeat orders. This improves profitability of an organization.
- **2. Blue Money/ Cost of Appraisal:** Blue money is a cost incurred by the organization during development in the form of first time review/ testing which gets returned in future. It does not earn profit, but an essential part of development process to ensure the process capability. First time testing helps in certifying that nothing has gone wrong and work product can go to the nextstage.
- **3. Red Money/ Cost of Failures:** Red money is a pure loss for the organization. It involves money lost in scrap, rework, sorting etc. It directly reduces the profit for the organization andthe customer may not pay for it.

Testing During Development Life Cycle

- There are 3 types of testing involved here :
 1. **Requirement Testing**
 2. **Design Testing**
 3. **Code Testing**
- 1. **Requirement Testing :**
 - Involves mock running of future application using the requirement statements to ensure that requirements meet their acceptance criteria. This type of testing is used to evaluate whether all requirements are covered in requirement statement or not.

- Requirements must be prioritized as “must”, “should be” and “could be” requirements.

2. Design Testing :

- Similar to developing data flow diagrams from the designs where the flow of information is tracked from start to finish.
- Wherever the flow is not defined, or not clear about where it will lead to, there are defects with design which must be corrected.

3. Code Testing

Code review is done to ensure that code files are:

- Readable and Maintainable in future
- Testable in unit testing
- Traceable with requirements and designs
- Testable in integration and system testing
- Optimized to ensure better working of software.
- Reusability creates a lighter system.

Test Scenario And Test Case Testing:

- Written by tester to address testing needs of a software application.
- Test Scenarios can be functional as well as structural depending on type of requirement and design they are addressing.
- Test Scenario must be clear and complete, representing end-to-end relationship of what is going to happen and also the possible outcomes of such processing.
- Test cases are derived from test scenarios.

Requirement Traceability Matrix:

- Doing the complete mapping for the software. One can expect a blueprint of an entire application using traceability matrix. Requirement Traceability assures good 'Quality' of the application as the features are tested.
- Quality control can be achieved as software gets tested for unforeseen scenarios with minimal defects and all functional and non-functional requirements being satisfied. There are 3 traceability matrix:
 - Horizontal Traceability
 - Vertical Traceability
 - Bidirectional Traceability

Advantages:

- Entire software development can be tracked completely through traceability matrix.
- Test-case failure can be tracked requirements, design, coding etc.
- Any changes in requirements can be affected through entire work product upto cases and also test case failure can be traced back to requirements.
- The application becomes maintainable as one has complete relationship requirement till test results available.

Disadvantages :

- If number of requirements are huge. It is very difficult to create traceability matrix manually.
- There may be one-to-many. Many-to-one, and many-to-many relationships between various elements of traceability matrix, when we are trying to connect columns and rows of traceability matrix, and maintaining these relationships need huge efforts.
- Requirements change frequently, and one needs to update the traceability whenever there is a change.
- A customer may not find value in it and may not pay for it.

1. Horizontal Traceability:

- When an application can be traced from requirement through design and coding will test scenario and test cases up to test results, it is termed as horizontal traceability.
- On failure of any test case, we must be able to find which requirements have not been met. Horizontal traceability is an aspect identifying nonhierarchical similarities, mutual properties, interactions, etc. among requirements and work products.

2. Vertical Traceability :

- Vertical traceability identifies the origin of items and follows these same items as they travel through the hierarchy of the work breakdown structure to the project teams and eventually to the customer.
- Traceability may exist in individual column as the requirements may have some interdependencies between them, or these may be child and parent relationship.

3. Bidirectional Traceability :

- Implemented both forward and backward from requirements to end products and from end products back to requirements.

Essentials of Software Testing

Viewed as exercise of doing SWOT (Strength, Weakness, Opportunity, Threats)

- **Strengths:** Some areas of software are very strong, and no defects are found during testing of such areas. It supports to develop a good product, We can always rely on these processes and try to deploy them in other areas.
- **Weakness:** There are processes in these areas representing some problems. An organization needs to analyse such processes and the organization such as training, communication, etc.
- **Opportunity:** Some areas of the software which satisfy requirements as defined by the customer, or implied requirements but with enough space available for improving it further. These improvements represent ability of the developing organization to help the customer and give competitive advantage.
- **Threats:** Threats are the problems or defects with software which result into failures. They represent the problems with some processes in the organization such as requirement clarity, knowledge and expertise.

Workbench

- A Workbench is a method of documenting how a particular activity must be fulfilled.
- A workbench is referred to a stages, steps, and assignments.
- A workbench comprises some procedures defined for doing a work and some procedures. Defined to check the outcome of the work done.
- While checking /testing a work product in a workbench, if one finds derivations between expected result and actual result, it may be considered as defect, and the work product and the process used for development needs to be reworked.

Tester's Workbench: Tester's workbench include:

1. Input to tester's workbench
 2. Do Process
 3. Check Process
 4. Output
 5. Standards And Tools
 6. Rework
1. **Input to Tester's workbench:** Input form is the very first phase of any documentation associated with a product, test environment, or test plan depending upon the location of workbench in life cycle. In the workbench technique, input form or workbench technique. Input may be test scenario, work product, entrance criteria is the foremost stages.
 2. **Do Process:** The software undergoes testing as per defined test case and test procedure. Do Process' must guide the normal tester while doing the process.
 3. **Check Process:** Evaluation of testing process to compare the achievements as defined in test objectives is done by 'check process. Check process helps in finding whether Do Process' have worked correctly or not.
 4. **Output:** Must be available as required in form of test report and test log from the test process. Output of the tester's workbench needs to have an exit criteria.
 5. **Standards And Tools :** Include how to install the application, which steps are to be followed while doing testing, how to capture defects.
 6. **Rework:** If 'Check Processes' find that "Do Processes' are not able to achieve the objective defined for them, it must follow the route of work.

Important Feature of Testing Process

- Testing is a destructive process but it is constructive destruction.
- Testing needs a Sadistic approach without a Consideration that there is a defect.
- If the test does not detect a defect present in the system than it is a unsuccessful test.
- A test that detects a defect is valuable investment for development as well as customer.
- It is risky to develop software and not to test it before delivery.
- With High Pressure to deliver the software as quickly as possible test process must provide maximum value in shortest timeframe.
- Organization aim must be defect prevention rather than finding and fixing a defect.

Misconceptions about testing

- Anyone can do testing and no special skills are required for testing.
- Defects found in testing are blamed on developers.
- Defects found by customer are blamed on tester.

Principles of Software Testing

- Programmers/team must avoid testing their own work products.
- Thoroughly inspect results of each test to find potential improvements.
- Initiate actions for correction, corrective actions and prevention actions.

Salient Features of Good Testing

- **Capturing user requirements:** Involve technical, economical, legal, and operational and system requirements.
- **Capturing user needs:** User needs may include present and future requirements and other requirements.
- **Design objectives:** Design objectives state why a particular approach has been selected for developing software.
- **User Interfaces:** Way in which the user interacts with the system.
- **Execution of Code:** Testing is execution of a work product to ensure that it works as intended by the customer or user, and is prevented from any probable misuse or risk of failure.

Test Plan	Test Strategy(Test Approach)
In Test Plan, test focus and project scope are defined. It deals with test coverage scheduling, features to be tested, features not to be tested, estimation and resource management	Test strategy is a guideline to be followed to achieve the test objective and execution of test types mentioned in the testing plan. It deals with test objective, test environment, test approach, automation tools and strategy, contingency plan, and risk analysis

Test Planning: First activity of test team. Test plans are intended to plan for testing throughout SDLC. Test plan should be realistic and talk about the limitations and constraints of testing. It should talk about the risks and assumptions done during testing.

- Plan testing efforts adequately with an assumption that defects are there.
- If defects are found, it is failure of testing activity.

- Successful tester is not one who appreciates development but one finds defects in the product.
- Testing is not a formality to be completed at the end of development life cycle.

Difference between Verification and Validation

Verification	Validation
An activity where we check the work products with reference to standards, guidelines and procedures	An activity to find whether the software achieves whatever is defined by requirements
Verification is prevention based. It tries to check the process adherence	Validation is detection based. It checks the product attributes
Verification talks about a process, Validation talks about the product standards and guidelines	Talks about the product
Verification is also termed 'white box testing' or static testing as the work product undergoes a review	Validation is also termed as 'black box testing' or dynamic testing' as work product is executed
Verification can find about 60% of the defects	Validation can find about 30% of the defects
Verification may be based on opinion of reviewer and may change from person to person	Validation is based on facts and is generally independent of a person
Verification involves the following Reviews Walkthrough Inspection Audits	Validation involves the following : System testing User interface testing Stress testing
Verification can give the following : Statement coverage Decision coverage Path coverage	Validation can give the following Requirement coverage Feature coverage Functionality coverage

Testing Process and Number of Defects Found in Testing

- Testing is intended to find more defects. Generally, it is believed that there are fixed number of defects in a product and as testing finds more defects, chances of the customer finding the defect will reduce.
- Actually the scenario is reverse. As we find more and more defects in a product, there is a probability of finding some more defects.
- This is based on the principle that every application had defects and every test team has some efficiency of finding defects. It is governed by the test team's defect-finding ability.

Test Team Efficiency

- A test team must have 100% efficiency for finding defects.
- But it is not possible practically so in order to be a good test team it must be very near to 100%.

- If it moves away from 100% the test team becomes more and more unreliable.
- Test team is dependent on organization culture and may not be improved unless organization makes some deliberate efforts.

Mutation Testing

Mutation testing is used to check the capability of test program and test cases to find defects. Test cases are designed and executed to find defects. If test cases are capable of finding defects, it is a loss for an organization. This is also termed 'test case efficiency'.

Example:

- A program is written, and set of test cases are designed and executed on the program. The test team may find out few defects, the original program is changed and some defects are added deliberately, this is called 'mutant of the first program' and process is termed 'mutation'. It is subjected to the same test case execution again.
- It is very difficult to get a test team with 100% efficiency of finding defects and test cases with 100% efficiency of finding defects. Some of the reasons for deviation are listed below:
 1. **Camouflage Effect:** It may be possible that one defect may camouflage another defect, and the tester may not be able to see that defect, or test case may not be able to locate the hidden defect. It is called 'Camouflage Effect' or Compensation defects as two defects compensate each other. Thus, defect introduced by developer may not be seen by the tester while executing a test case
 2. **Cascading Effect:** It may be possible that due to existence of a certain defect, few more defects are introduced or seen by the tester. Though there is no problem in the modification, defects are seen due to cascading effect of one defect. Thus, defects not introduced by developer may be seen by tester while executing a test case.
 3. **Coverage Effect:** It is understood that nobody can test 100%, and there may be few lines of code or few combinations which are not tested at all due to some reasons.
 - a. **Redundant Code:** There may be parts of code, which may not get executed under any condition, as the conditions may be impossible to occur, or some other condition may take precedence over it. If developer introduces a defect in such parts, testers will not be able to find defect as that part of code will never get executed.

Challenges in Testing

- **Testing** is a challenging job. Management may have problems with understanding testing approach and may consider it as an obstacle to be crossed before delivering the product to the customer.
- There may be problems related to testing process as well as Independent development process.
- **Major challenges faced by test teams are as follows:**
 - Requirements are not clear, complete, consistent, measurable and testable. These may create some problems in defining test scenario and test case.
 - Requirements may be wrongly documented and interpreted by business analyst and system analyst. Code logic may be difficult to capture. Often, testers are not able to

understand the code due to lack of technical knowledge. Sometimes, testers do not have access to code files. Error handling may be difficult to capture.

Test Team Approaches

There are four approaches:

1. Approach based on location of test teams in an organization.
 - I. **Independent test team:** Independent test team report independently to senior management or customer.
 - II. **Test team reporting to development manager :** If the test team is reporting to development manager
2. **Developers becoming testers:** Sometimes those who work as developers in initial stage take the role of tester too.
3. **Domain experts doing software testing:** Organizations employ domain experts for software testing.

Process Problems faced by testing

- **People - Many** people are involved in software development and testing Customer, business analyst, and test managers. There is a possibility that a few instances some personal attributes and capabilities may create problems.
- **Material** - Testers need requirement document standards and test standard, guidelines, and other material which add to their knowledge about a prospective system.
Machines - Testers try to build real-life scenario using various machines, these may include computers, hardware, software, and printers.
Methods - Methods for doing test planning, risk analysis, defining test scenarios, test cases, and test data may not be proper.

Cost Aspect of Testing

- The cost of customer dissatisfaction is inversely proportional to testing efforts. It means more investment on testing efforts reduces the cost of customer unhappiness.
- Cost of Customer dissatisfaction is guided by the following aspects :
 - Cost of customer dissatisfaction mainly depends upon customer-supplier relationship. Customer dissatisfaction curves tend to be parallel to "Y axis. On the other hand, if the customer is finicky (fussy/overcritical) about defects and past performance of an organization is not good, it may tend to be parallel to X axis.
 - Cost of customer dissatisfaction also depends on the type of product and its mission.
- Efforts spent by organization in developing and testing of an application are converted at some predefined rates to arrive at the total cost of a product.

Establishing Test Policy

- **Good** testing is a deliberate planned effort by the organization. Test Strategy or approach must define what steps are required for performing an effective testing. How the test environment will be created, what tools will be used for testing, defect capturing, defect reporting and number of test cycles required will be a part of test strategy. It must talk about the depth and breadth of testing to ensure adequate confidence levels

for users. Test objectives define what testing will be targeting to achieve.

- Generally, methods applied for testing efforts are defined at organizational levels. They are generic in nature and hence, need customization. They are customized into our test plan, and any tailoring required to suit a specific project. Management directives establish methods applied for testing. It includes what part will be tested / not tested, and how it will be tested. Which tools will be used for testing, defect tracking mechanism and so on.
- Testing strategy may be discussed with customers to get their views in testing, It may be accomplished through meetings and memorandums. All stakeholders for the project must be made aware that 'zero defects' is an impossible condition and acceptance criteria for the project. Methods of using data or inputs provided by a customer must be analyzed for sufficient and correctness.

Structural Approach to Testing

Testing is a lifecycle activity and should be carried out in all the phases of testing. If testing is carried only at the end of deployment phase then it can be costly. Four components of waste can be generated here:

1. **Waste in Wrong Development:** Wrong specification used for development or testing will result into wrong product and wrong testing.
2. **Waste in testing defects**
3. **Wastage as wrong specifications, designs, codes and documents must be replaced by correct specifications, designs, codes and documents.**
4. **Wastage as system must be retested to ensure that the corrections are correct.**

Categories of defect

Mistake	Error	Defect
Issue identified while reviewing our own documents or peer reviews	Issue identified internally or in unit testing	Issue identified in black box testing or by customer
Very low cost of finding mistakes and can be fixed immediately	Slightly more cost of finding an error and needs more time for fixing	Costly and needs longer time for fixing defects
Problems and resolutions are not documented properly	Problems and resolutions are documented but may not be used for process improvements	Problems and resolutions are officially documented and used for process improvements.

Developing Test Strategy

Test planning included developing a strategy about the test team will perform testing. Some key components of testing strategy are as follows:

- Test factors required in particular phase of development.
- Test phase corresponding to development phase.

Process of developing test strategy goes through the following stages:

- Select and rank test factors for the given application.
- Identify system development phases and related test factors.
- Identify associated risks with each selected test factor in case if it is not achieved.
- Identify phase in which risks of not meeting a test factor need to be addresses.

Developing testing methodologies (Test Plan)

Developing test tactics is the job of project – level testmanager/test lead. Different projects may need different tactics as per type of product/customer.

- Designing and defining of test methodology may take the following route.
- Acquire and study test strategy as define earlier.
- Determine the type of development project being executed.
- Determine the type of software system being made.
- Identify tactical risks related to development.
- Determine when testing must occur during life cycle.
- Steps to develop customized test strategy.
- Type of development metrology impact test plan decisions.

Testing Process

The following are the milestones (steps) in the process of testing:

- **Defining Test Policy :** Test policies are defined by senior management of the organization or test management.
- **Defining Test Strategy:** Test Strategy provides the actions to the intents defined by test policy.
- **Preparing Test Plan:** Test plan tries to answer six basic questions: what, where, why, when, which and how.
- **Establishing Testing To Be Achieved :** Measure the effectiveness and efficiency of a testing process.
- **Developing Test Scenarios and test cases:** A test scenario represents user scenario which acts as a framework for defining test cases.
- **Writing/reviewing test cases:** These are done by senior testers or test leads for the project.
- **Defining test data:** Test data may be defined on the basis of different techniques available for the purpose. Includes boundary value analysis, error guessing, and equivalence partitioning and state transition.
- **Creation of test bed:** Test bed defines some assumptions in a test plan which may include certain risks of testing.
- **Executing test cases:** Execution of actual test cases with the test data defined for testing the software involves applying test cases as well as test data and trying to get the actual result.

- **Test Result:** One needs to make sure that the expected results are traceable to requirements.
- **Test Result Analysis:** At the end of testing, the test team must recommend the next step after testing is completed to the project manager- whether the software is ready to go to the next stage or it needs further rework and retesting.
- **Performing retesting/regression testing when defects are resolved by development team :** Retesting is done to find out whether the defects declared as fixed and verified by the development team are really fixed or not. Regression testing is done to confirm that the changed part has not affected any other parts of software, which was working earlier.
- **Root cause Analysis and corrective/preventive actions :** Root cause analysis is required to initiate corrective actions.

Attitude towards Testing

Some of the views about test team are as follow:

- New members of development team are not accustomed to view testing as a discovery process. The defects found are taken as personal blames rather than system/process lacunae.
- We take pride on what we developed' or 'we wish to prove that it is right' or 'it is not my fault' are very common responses. Developers may not accept the defect in the first place.
- Conflict between developer and tester can create differences between project team and test teams. In reality, the sole aim of development and testing must be a customer satisfaction, and defects must be considered as something which prevents achievements of this objective.

Test Methodologies/Approaches

The main methodologies (approaches) for testing is as follows:

1. Black Box Testing
2. White Box Testing

Besides this there is one more approach for testing: **Grey Box Testing**

Black Box Testing	Grey Box Testing	White Box Testing
The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
Also known as closed box testing, data driven testing and functional testing	Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
Testing is based on external expectations- Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly

This is the least time consuming and exhaustive	Partly time consuming and exhaustive.	The most exhaustive and time consuming type of testing.
Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing
This can only be done by trial and error method	Data domains and Internal boundaries can be tested, if known	Data domains and Internal boundaries can be better tested

People Challenges in Software Testing

- The tester is responsible for improving testing process to ensure better products with less number of defects going to customer, thus enhancing customer satisfaction. All defects must be found and the confidence level must be built in the process that can give customer satisfaction.
- Testing needs trained and skilled people who can deliver products with minimum defects to the stakeholders. Testers have to improve their skills through continuous learning.
- The tester needs a positive team attitude for creative destructive of software. Defect in the software is an opportunity to improve the product.
- Testing is a creative work and challenging task.
- Programmers and testers are essential to improve the quality of software developed and delivered to customer, and the process used for software development and testing. The ultimate aim is customer satisfaction.
- Every defect is considered as a process shortcoming. Defect closure needs retesting and regression testing to find whether the defect is really fixed or not, and to ensure that there is no negative impact of a defect on existing functions.
- Testing needs patience, fairness, ambition, creditability, capability, and diligence on part of testers. Every defect must be seen from the business perspective.

Raising Management Awareness For Testing

Tester's Role:

- While establishing a test function in an organization, the management has some objectives to be achieved.
- Test team needs to understand these objectives and fulfill them.
- Calculate testing cost, effectiveness of testing and ensure that management understands the same.
- By doing good design, number of customer complaint must reduce and cost of failure must go down
- Highlights needs and benefits of training-in test team as well as development team-on testing activities and skills, so that testers can perform better.
- Collect and distribute information on testing to all team members as well as development team/organization which can be used for improvement.
- Get involved in testing budget.

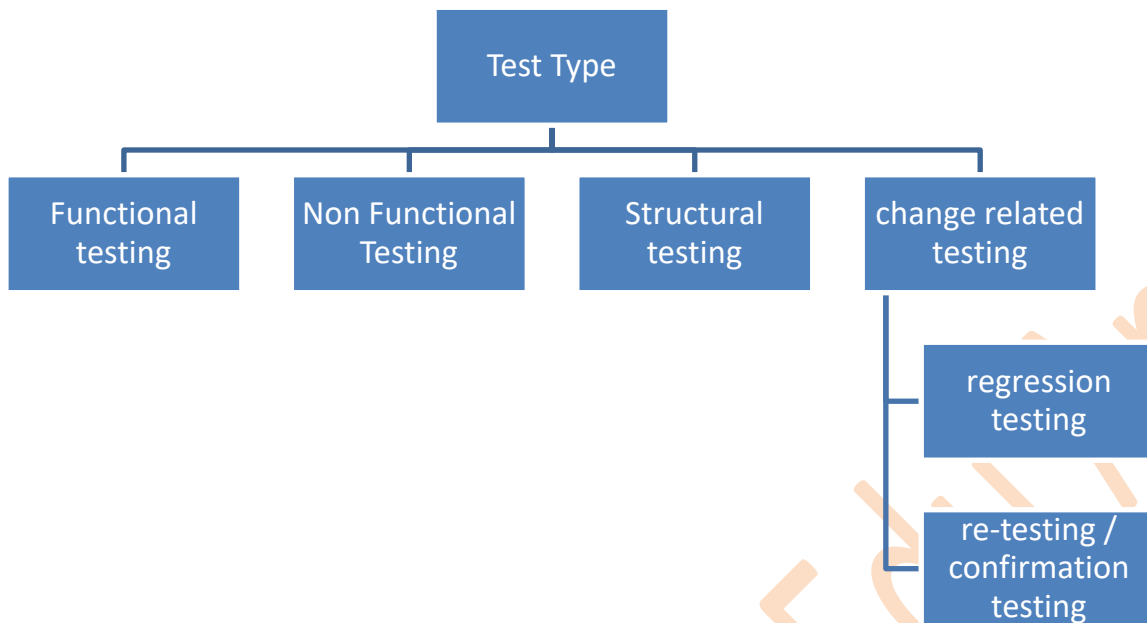
Skills Required By Tester

General Skills	Testing Skills
Written and verbal presentation skill	Concepts of testing
Effective listening skill	Levels of testing
Facilitation skill	Techniques for validation and verification
Software development, operations and maintenance	Selection and use of testing tools
Continuous education	Knowledge of testing standards
	Risk assessment and management
	Developing test plan
	Defining acceptance criteria
	Checking of testing processes
	Execution of test plan
	Continuous improvement of testing process

Test Levels: A level of software testing is a process where every unit or component of software/system is tested. These are the various test levels:

1. Unit Test
2. Integration Test
3. System Test
4. Acceptance Test
 - 1) **Unit Testing:** Unit testing: A Unit is a smallest testable portion of system or application which can be compiled, linked, loaded, and executed. Helps to test each module separately. Aim is to test each part of the software by separating it. Checks that component are fulfilling functionalities or not. Performed by developers.
 - 2) **Integration Testing:** Integration means combining. In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing. Checks the data flow from one module to other modules. Performed by testers.
 - 3) **System Testing:** System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing. System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.
 - 4) **Acceptance Testing:** Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

Test Types



Functional Versus Non-Functional Testing:

Functional Testing	Non-Functional Testing
Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements	Non-Functional testing checks the Performance, reliability, scalability and other non-functional aspects of the software system.
Functional testing is executed first	Nonfunctional testing should be performed after functional testing
Manual Testing or automation tools can be used for functional testing	Using tools will be effective for this testing Performance parameters like speed scalability are inputs to non-functional testing
Business requirements are the inputs to functional testing	Performance parameters like speed, scalability are inputs to non-functional testing
Functional testing describes what the product does	Nonfunctional testing describes how good the product works
Easy to do Manual Testing	Tough to do Manual Testing
Types of Functional testing are Smoke Testing Integration Testing Black Box testing Regression Testing Unit Testing Sanity Testing White box testing User Acceptance testing	Types of Nonfunctional testing are Performance Testing Volume Testing Security Testing Penetration Testing Migration Testing Load Testing Stress Testing Installation Testing Compatibility Testing

Structural Testing: Structural testing, also known as glass box testing or white box testing is an approach where the tests are derived from the knowledge of the software's structure or internal implementation.

Structural Testing Techniques:

- **Statement Coverage** - This technique is aimed at exercising all programming statements with minimal tests.
- **Branch Coverage** - This technique is running a series of tests to ensure that all branches are tested at least once.
- **Path Coverage** - This technique corresponds to testing all possible paths which means that each statement and branch are covered.

Change Related Testing: This testing is the last target of testing. This category is different from the above categories because if we make any changes in the software then we can identify the changes in its functions and structure.

Regression Testing	Retesting
Regression testing verifies that a recent change in program code has not affected the existing feature	Retesting verifies the failed test cases in the final execution especially when the defects are fixed.
Verification of defect is not a part of Regression testing.	Verification of defect is a part of the Retesting.
This testing is performed for passed test cases	This testing is performed for failed test cases.
This testing is only completed when any change becomes compulsory.	Retesting executes faults with same data and environment with various inputs.

Maintenance Testing

The testing of the changes that may occur after the software is developed is called maintenance testing, and is triggered by:

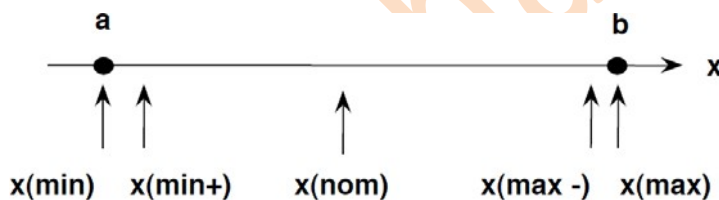
- Planned enhancements or upgrades to business systems.
- Corrective and emergency changes.
- Changes to the operating environment such as system upgrades.
- Upgrade of commercial-off-the-shelf software.
- Migration of applications from one platform to another.
- Retirement of legacy software systems.

Unit-3

Chapter 1: Unit Testing: Boundary Value Testing

Boundary Value 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".
- Boundary value analysis is a type of black box or specification-based testing technique in which tests are performed using the boundary values.
- There are four types of Boundary value testing:
 - a) Normal boundary value Testing.
 - b) Robust boundary value Testing.
 - c) Worst-case boundary value Testing.
 - d) Robust Worst-case boundary value 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



- In Boundary Testing, Equivalence Class Partitioning plays a good role
- Boundary Testing comes after the Equivalence Class Partitioning.
- 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

Normal Boundary Value Testing

In the general application of Boundary Value Analysis can be done in a uniform manner. The basic form of implementation is to maintain all but one of the variables at their nominal (normal or average) values and allowing the remaining variable to take on its extreme values. The values used to test the extremities are:

Min..... - Minimal

Min+ - Just above Minimal

Nom - Average

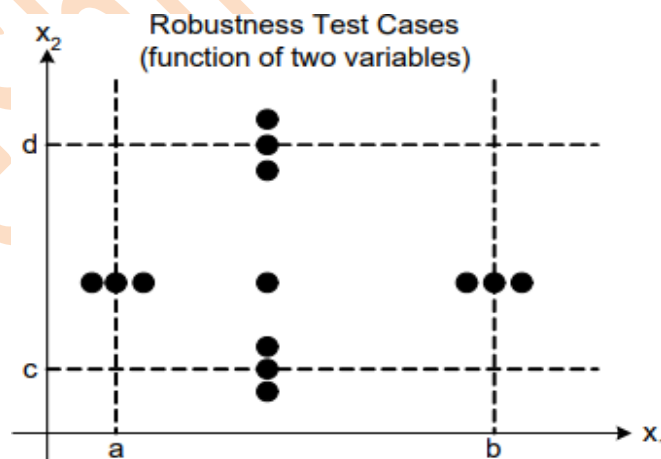
Max- - Just below Maximum

Max - Maximum

Robust boundary value Testing

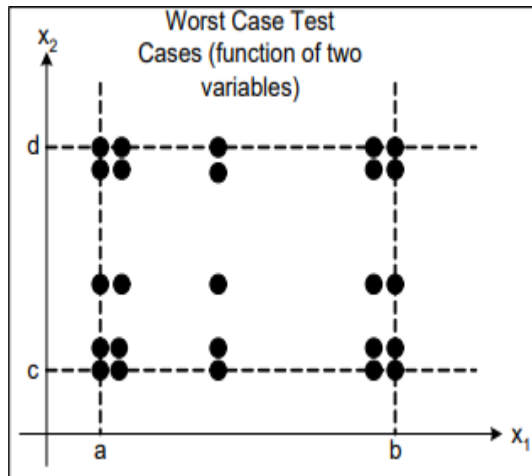
- Robustness testing can be seen as an extension of Boundary Value Analysis. The idea behind Robustness testing is to test for clean and dirty test cases.
- In addition to the aforementioned 5 testing values (min, min+, nom, max-, max) we use two more values for each variable (min-, max+), which are designed to fall just outside of the input range.
- If we adapt our function f to apply to Robustness testing we find the following equation:
- $f = 6n + 1$
- Robustness testing ensures a sway in interest, where the previous interest lied in the input to the program, the main focus of attention associated with Robustness testing comes in the expected outputs when an input variable has exceeded the given input domain. For example the NextDate problem when we have an entry like the 31st June we would expect an error message to the effect of "that date does not exist; please try again".
- Robustness testing has the desirable property that it forces attention on exception handling. Although Robustness testing can be somewhat awkward in strongly typed languages it can show up alterations. In Pascal if a value is defined to reside in a certain range then values that fall outside that range result in the run time errors that would terminate any normal execution. For this reason exception handling mandates Robustness testing.

$$f(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 \text{ is min- and } x_2 \text{ is min-} \\ 2 & \text{if } x_1 \text{ is min- and } x_2 \text{ is min+} \\ 3 & \text{if } x_1 \text{ is min- and } x_2 \text{ is nom} \\ 4 & \text{if } x_1 \text{ is min- and } x_2 \text{ is max-} \\ 5 & \text{if } x_1 \text{ is min- and } x_2 \text{ is max+} \\ 6 & \text{if } x_1 \text{ is min+ and } x_2 \text{ is min-} \\ 7 & \text{if } x_1 \text{ is min+ and } x_2 \text{ is min+} \\ 8 & \text{if } x_1 \text{ is min+ and } x_2 \text{ is nom} \\ 9 & \text{if } x_1 \text{ is min+ and } x_2 \text{ is max-} \\ 10 & \text{if } x_1 \text{ is min+ and } x_2 \text{ is max+} \\ 11 & \text{if } x_1 \text{ is nom and } x_2 \text{ is min-} \\ 12 & \text{if } x_1 \text{ is nom and } x_2 \text{ is min+} \\ 13 & \text{if } x_1 \text{ is nom and } x_2 \text{ is nom} \\ 14 & \text{if } x_1 \text{ is nom and } x_2 \text{ is max-} \\ 15 & \text{if } x_1 \text{ is nom and } x_2 \text{ is max+} \\ 16 & \text{if } x_1 \text{ is max- and } x_2 \text{ is min-} \\ 17 & \text{if } x_1 \text{ is max- and } x_2 \text{ is min+} \\ 18 & \text{if } x_1 \text{ is max- and } x_2 \text{ is nom} \\ 19 & \text{if } x_1 \text{ is max- and } x_2 \text{ is max-} \\ 20 & \text{if } x_1 \text{ is max- and } x_2 \text{ is max+} \\ 21 & \text{if } x_1 \text{ is max+ and } x_2 \text{ is min-} \\ 22 & \text{if } x_1 \text{ is max+ and } x_2 \text{ is min+} \\ 23 & \text{if } x_1 \text{ is max+ and } x_2 \text{ is nom} \\ 24 & \text{if } x_1 \text{ is max+ and } x_2 \text{ is max-} \\ 25 & \text{if } x_1 \text{ is max+ and } x_2 \text{ is max+} \end{cases}$$



Worst-case boundary value Testing

- Boundary Value analysis uses the critical fault assumption and therefore only tests for a single variable at a time assuming its extreme values. By disregarding this assumption, we are able to test the outcome if more than one variable were to assume its extreme value. In an electronic circuit this is called Worst Case Analysis. In Worst-Case testing we use this idea to create test cases.
- To generate test cases, we take the original 5-tuple set (min, min+, nom, max-, max) and perform the Cartesian product of these values. The end product is a much larger set of results than we have seen before.



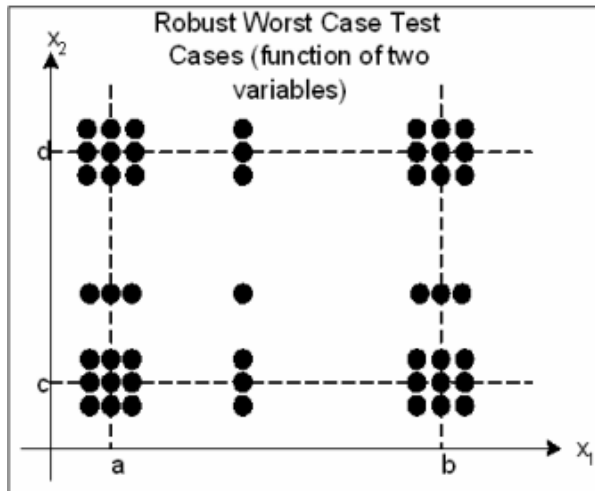
$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1nom}, x_{2nom} \rangle$
$\langle x_{1min}, x_{2min+} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min}, x_{2min+} \rangle$	$\langle x_{1max}, x_{2min+} \rangle$
$\langle x_{1min}, x_{2nom} \rangle$	$\langle x_{1max}, x_{2nom} \rangle$
$\langle x_{1min}, x_{2nom} \rangle$	$\langle x_{1max}, x_{2nom} \rangle$
$\langle x_{1min}, x_{2max-} \rangle$	$\langle x_{1max}, x_{2max-} \rangle$
$\langle x_{1min}, x_{2max-} \rangle$	$\langle x_{1max}, x_{2max-} \rangle$
$\langle x_{1min+}, x_{2min} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min+}, x_{2min} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min+}, x_{2min+} \rangle$	$\langle x_{1max}, x_{2min+} \rangle$
$\langle x_{1min+}, x_{2min+} \rangle$	$\langle x_{1max}, x_{2min+} \rangle$
$\langle x_{1min+}, x_{2nom} \rangle$	$\langle x_{1max}, x_{2nom} \rangle$
$\langle x_{1min+}, x_{2nom} \rangle$	$\langle x_{1max}, x_{2nom} \rangle$
$\langle x_{1min+}, x_{2max-} \rangle$	$\langle x_{1max}, x_{2max-} \rangle$
$\langle x_{1min+}, x_{2max-} \rangle$	$\langle x_{1max}, x_{2max-} \rangle$
$\langle x_{1nom}, x_{2min} \rangle$	$\langle x_{1nom}, x_{2max-} \rangle$
$\langle x_{1nom}, x_{2min+} \rangle$	$\langle x_{1nom}, x_{2max-} \rangle$

- We can see from the results in figures that worst case testing is a more comprehensive testing technique. This can be shown by the fact that standard Boundary Value Analysis test cases are a proper subset of Worst-Case test cases.
- These test cases although more comprehensive in their coverage, constitute much more endeavor. To compare we can see that Boundary Value Analysis results in $4n + 1$ test case where Worst-Case testing results in $5n$ test cases. As each variable has to assume each of its variables for each permutation (the Cartesian product) we have 5 to the n test cases.
- For this reason Worst-Case testing is generally used for situations that require a higher degree of testing (where failure of the program would be very costly) with less regard for the time and effort required as for many situations this can be too expensive to justify.

Robust Worst-case boundary value Testing:

- If the function under test were to be of the greatest importance we could use a method named Robust Worst-Case testing which as the name suggests draws it attributes from Robust and Worst-Case testing.
- Test cases are constructed by taking the Cartesian product of the 7-tuple set defined in the Robustness testing chapter. Obviously this results in the largest set of test results we have seen so far and requires the most effort to produce.
- The function f (to calculate the number of test cases required) can be adapted to calculate the amount of Robust Worst-Case test cases. As there are now 7 values each variable can

- assume we find the function f to be: $f = 7n$ This function has also been reached in the paper A Testing and analysis tool for Certain 3-Variable functions [2]. The results for the continuing example can be seen in figures:



$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1nom}, x_{2nom} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1nom}, x_{2max} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1nom}, x_{2max} \rangle$	$\langle x_{1max}, x_{2nom} \rangle$
$\langle x_{1min}, x_{2nom} \rangle$	$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1nom}, x_{2max} \rangle$	$\langle x_{1max}, x_{2max} \rangle$
$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1max}, x_{2min} \rangle$	$\langle x_{1max}, x_{2max} \rangle$
$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1min}, x_{2nom} \rangle$	$\langle x_{1max}, x_{2min} \rangle$	$\langle x_{1max}, x_{2max} \rangle$
$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1max}, x_{2min} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1max}, x_{2nom} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1max}, x_{2max} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min}, x_{2min} \rangle$	$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1max}, x_{2max} \rangle$	$\langle x_{1max}, x_{2min} \rangle$
$\langle x_{1min}, x_{2nom} \rangle$	$\langle x_{1nom}, x_{2min} \rangle$	$\langle x_{1max}, x_{2max} \rangle$	$\langle x_{1max}, x_{2max} \rangle$
$\langle x_{1min}, x_{2nom} \rangle$	$\langle x_{1nom}, x_{2min} \rangle$	$\langle x_{1max}, x_{2max} \rangle$	$\langle x_{1max}, x_{2max} \rangle$
$\langle x_{1min}, x_{2max} \rangle$	$\langle x_{1nom}, x_{2min} \rangle$	$\langle x_{1max}, x_{2min} \rangle$	$\langle x_{1max}, x_{2max} \rangle$

Special value testing

- It is the form of functional testing.
- Testing where Tester domain experience, knowledge on similar products/projects, intuition helps in testing the application successfully.
- It is the most intuitive and the least uniform.

Advantages:

- Soft Spots error can be identified with minimal efforts.

Disadvantages:

- 1) Domain Knowledge is required.
- 2) There is no Co-ordination with developers.

Examples:

Triangle problem

Standard Boundary Value Analysis test cases:

min = 1 min+ = 2 nom = 100 max- = 199 max = 200

Boundary Value Analysis Test Cases				
Case	a	b	c	Expected Output
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a Triangle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	199	100	Isosceles
9	100	200	100	Not a Triangle
10	1	100	100	Isosceles
11	2	100	100	Isosceles
12	199	100	100	Isosceles
13	200	100	100	Not a Triangle

Worst-Case Analysis test cases:

Worst Case Test Cases (60 of 125)									
Case	a	b	c	Expected Output	Case	a	b	c	Expected Output
1	1	1	1	Equilateral	31	2	2	1	Isosceles
2	1	1	2	Not a Triangle	32	2	2	2	Equilateral
3	1	1	100	Not a Triangle	33	2	2	100	Not a Triangle
4	1	1	199	Not a Triangle	34	2	2	199	Not a Triangle
5	1	1	200	Not a Triangle	35	2	2	200	Not a Triangle
6	1	2	1	Not a Triangle	36	2	100	1	Not a Triangle
7	1	2	2	Isosceles	37	2	100	2	Not a Triangle
8	1	2	100	Not a Triangle	38	2	100	100	Isosceles
9	1	2	199	Not a Triangle	39	2	100	199	Not a Triangle
10	1	2	200	Not a Triangle	40	2	100	200	Not a Triangle
11	1	100	1	Not a Triangle	41	2	199	1	Not a Triangle
12	1	100	2	Not a Triangle	42	2	199	2	Not a Triangle
13	1	100	100	Isosceles	43	2	199	100	Not a Triangle
14	1	100	199	Not a Triangle	44	2	199	199	Isosceles
15	1	100	200	Not a Triangle	45	2	199	200	Scalene
16	1	199	1	Not a Triangle	46	2	200	1	Not a Triangle
17	1	199	2	Not a Triangle	47	2	200	2	Not a Triangle
18	1	199	100	Not a Triangle	48	2	200	100	Not a Triangle
19	1	199	199	Isosceles	49	2	200	199	Scalene
20	1	199	200	Not a Triangle	50	2	200	200	Isosceles
21	1	200	1	Not a Triangle	51	100	1	1	Not a Triangle
22	1	200	2	Not a Triangle	52	100	1	2	Not a Triangle
23	1	200	100	Not a Triangle	53	100	1	100	Isosceles
24	1	200	199	Not a Triangle	54	100	1	199	Not a Triangle
25	1	200	200	Isosceles	55	100	1	200	Not a Triangle
26	2	1	1	Not a Triangle	56	100	2	1	Not a Triangle
27	2	1	2	Isosceles	57	100	2	2	Not a Triangle
28	2	1	100	Not a Triangle	58	100	2	100	Isosceles
29	2	1	199	Not a Triangle	59	100	2	199	Not a Triangle
30	2	1	200	Not a Triangle	60	100	2	200	Not a Triangle

Again this is only up to 60 of 125 test cases.

Random Testing

Random Testing, also known as monkey testing, is a form of functional black box testing that is performed when there is not enough time to write and execute the tests.

Random Testing Characteristics:

- Random testing is performed where the defects are NOT identified in regular intervals.
- Random input is used to test the system's reliability and performance.
- Saves time and effort than actual test efforts.
- Other Testing methods Cannot be used to.

Random Testing Steps:

- Random Inputs are identified to be evaluated against the system.
- Test Inputs are selected independently from test domain.
- Tests are Executed using those random inputs.
- Record the results and compare against the expected outcomes.
- Reproduce/Replicate the issue and raise defects, fix and retest.

Guidelines for Boundary Value Testing

- Boundary value analysis can also be used for internal variables, such as loop control variables, indices and pointers.
- The test method based on the input domain of a function are the most rudimentary of all specification-based method.
- They share the common assumption that the input variable are truly independent;
- And when this assumption is not warranted the method generate unsatisfactory testcases.
- Robustness testing is a good choice for testing internal variable.

Chapter 2: Equivalence Class Testing

Equivalence Class Testing

- It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.

The use of equivalence classes is appropriate in situations like:

- When exhaustive testing is desired.
- When there is a strong need to avoid redundancy.

Types of equivalence class testing:

- 1) Weak Normal Equivalence Class Testing.
- 2) Strong Normal Equivalence Class Testing.
- 3) Weak Robust Equivalence Class Testing.
- 4) Strong Robust Equivalence Class Testing.

Advantages:

- Equivalence class testing helps reduce the number of test cases without compromising

[For detailed Video Lecture Download The Shikshak Edu App](#)

the test coverage.

- Reduces the overall test execution time as it minimizes the set of test data.

Disadvantages:

- It does not consider the conditions for boundary value.
- The identification of equivalence classes relies heavily on the expertise of testers.

Traditional Equivalence Class Testing

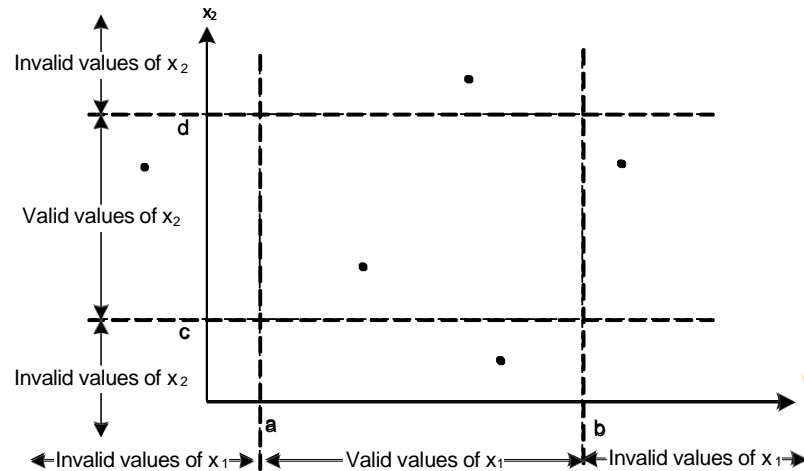
- In the 1960s and 1970s, programmers often had very detailed input data requirements.
- If input data didn't comply, it was the user's fault
- The popular phrase—Garbage In, Garbage Out (GIGO)
- Programs from this era soon developed defenses: as much as 75% of code verified input formats and values
- It is nearly identical to weak robust equivalence class testing.
- "Traditional" equivalence class testing focuses on detecting invalid input.
- Example of Traditional Equivalence Class Testing (only 2-dimensions for drawing purposes) easy to extend to more variables

$F(x_1, x_2)$ has these classes...

- valid values of x_1 : $a \leq x_1 \leq b$
- invalid values of x_1 : $x_1 < a, b < x_1$
- valid values of x_2 : $c \leq x_2 \leq d$
- invalid values of x_2 : $x_2 < c, d < x_2$

Process:

- test F for valid values of all variables,
- then test one invalid variable at a time
- (note this makes the single fault assumption)



Improved Equivalence Class Testing

- Equivalence Class Testing, which is also known as **Equivalence Class Partitioning (ECP)** and **Equivalence Partitioning**, is an important software testing technique used by the team of testers for grouping and partitioning of the test input data, which is then used for the purpose of testing the software product into a number of different classes.
- These different classes resemble the specified requirements and common behaviour or attributes of the aggregated inputs. Thereafter, the test cases are designed and created based on each class attributes and one element or input is used from each class for the test execution to validate the software functioning, and simultaneously validates the similar working of the software product for all the other inputs present in their respective classes.
- The equivalence class testing can be categorized into four different types, which are integral part of testing and cater to different data set. These types of equivalence class testing are:
 - Weak Normal Equivalence Class Testing:** In this first type of equivalence class testing, one variable from each equivalence class is tested by the team. Moreover, the values are identified in a systematic manner. Weak normal equivalence class testing is also known as **single fault assumption**.
 - Strong Normal Equivalence Class Testing:** Termed as **multiple fault assumption**, in strong normal equivalence class testing the team selects test cases from each element of the Cartesian product of the equivalence. This ensures the notion of completeness in testing, as it covers all equivalence classes and offers the team one of each possible combinations of inputs.
 - Weak Robust Equivalence Class Testing:** Like weak normal equivalence, weak robust testing too tests one variable from each equivalence class. However, unlike the former method, it is also focused on testing test cases for invalid values.
 - Strong Robust Equivalence Class Testing:** Another type of equivalence class testing, strong robust testing produces test cases for all valid and invalid elements of the product of the equivalence class. However, it is incapable of reducing the redundancy in testing.

Edge case

- When a programmer makes an error (mistake), which results in a defect in the software source code.

[For detailed Video Lecture Download The Shikshak Edu App](#)

- If this defect is executed, system will produce wrong results, causing a failure. A defect can be called fault or bug.
- QA (Quality Assurance Team) person uses many test cases based on the requirement and also does create a test case for edge case or tail end case testing. This testing is known as when an user normally will not enter into this situation.
- Consider simple example of inputting the employee information:
 - When you key in the employee information, the maximum age limit would be 70 or 80. But if you key in a year that results in employee age as more than 200, the software will not function correctly. This error should be caught while doing edge case testing. Even if we release this product to the client with out testing this case, it will continue to work until the user intentionally enters the age above 200.

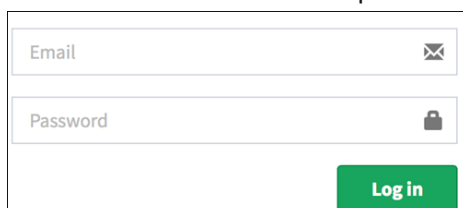
Guidelines and Observations

- Equivalence Class Testing is appropriate when input data is defined in terms of intervals and set of discrete values.
- Equivalence Class Testing is strengthened when combined with Boundary Value Testing
- Strong equivalence takes the presumption that variables are independent. If that is not the case, redundant test cases may be generated.
 - e.g., February 30 and 31 for three different types of years can be redundant test cases
- Complex functions, such as the NextDate program, are well-suited for Equivalence Class Testing
- Several tries may be required before the “right” equivalence relation is discovered
 - If the equivalence classes are chosen wisely, the potential redundancy among test cases is greatly reduced
 - The key point in equivalence class testing is the choice of the equivalence relation that determines the classes

Chapter 3: Decision table-Based Testing

Decision table

- Decision table testing is a software testing technique used to test system behavior for different input combinations.
- This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form.
- That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage.
- A Decision Table is a tabular representation of inputs versus rules/cases/test conditions.



Email	✉
Password	🔒
<button>Log in</button>	

Example

The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

- T – Correct username/password
- F – Wrong username/password
- E – Error message is displayed
- H – Home screen is displayed

Interpretation:

- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

While converting this to test case, we can create 2 scenarios,

- Enter correct username and correct password and click on login, and the expected result will be the user should be navigated to homepage

And one from the below scenario

Enter wrong username and wrong password and click on login, and the expected result will be the user should get an error message

- Enter correct username and wrong password and click on login, and the expected result will be the user should get an error message
- Enter wrong username and correct password and click on login, and the expected result will be the user should get an error message

Decision table technique

- Decision table technique is one of the widely used case design techniques for black box testing. This

[For detailed Video Lecture Download The Shikshak Edu App](#)

is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form.

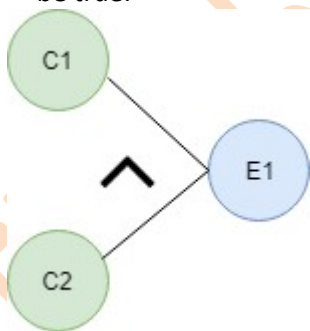
- That's why it is also known as a cause-effect table. This technique is used to pick the testcases in a systematic manner; it saves the testing time and gives good coverage to the testing area of the software application.
- Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs.
- This technique is related to the correct combination of inputs and determines the result of various combinations of input. To design the test cases by decision table technique, we need to consider conditions as input and actions as output.

Cause and Effect Graph

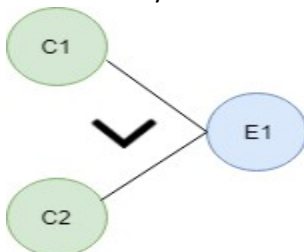
- Cause-effect graph comes under the black box testing technique which underlines the relationship between a given result and all the factors affecting the result. It is used to write dynamic test cases.
- The dynamic test cases are used when code works dynamically based on user input. For example, while using email account, on entering valid email, the system accepts it but, when you enter invalid email, it throws an error message. In this technique, the input conditions are assigned with causes and the result of these input conditions with effects.
- Cause-Effect graph technique is based on a collection of requirements and used to determine minimum possible test cases which can cover a maximum test area of the software.
- The main advantage of cause-effect graph testing is, it reduces the time of test execution and cost.
- This technique aims to reduce the number of test cases but still covers all necessary test cases with maximum coverage to achieve the desired application quality.
- Cause-Effect graph technique converts the requirements specification into a logical relationship between the input and output conditions by using logical operators like AND, OR and NOT.

Notations used in the Cause-Effect Graph

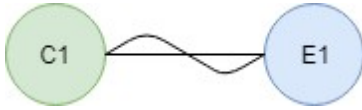
AND - E1 is an effect and C1 and C2 are the causes. If both C1 and C2 are true, then effect E1 will be true.



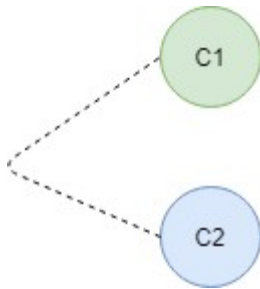
OR - If any cause from C1 and C2 is true, then effect E1 will be true.



NOT - If cause C1 is false, then effect E1 will be true.



Mutually Exclusive - When only one cause is true.



Guidelines and Observations

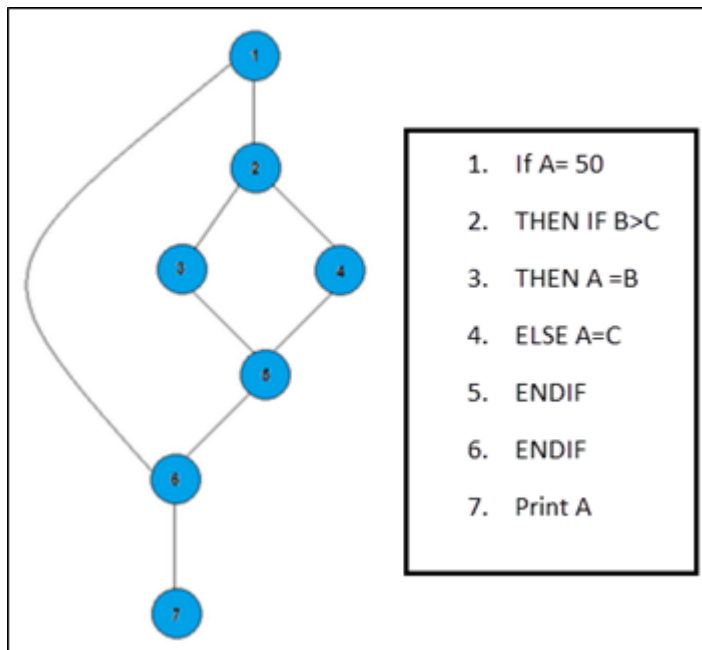
- Decision Table testing is most appropriate for programs where
 - There is a lot of decision making
 - There are important logical relationships among input variables
 - There are calculations involving subsets of input variables
 - There are cause and effect relationships between input and output
 - There is complex computation logic (high cyclomatic complexity)
- Decision tables do not scale up very well.
 - May need to
 - Use extended entry decision tables
 - Algebraically simplify tables
- Decision tables can be iteratively refined
 - The first attempt may be far from satisfactory
- Look for redundant rules
 - More rules than combination count of conditions
 - Actions are the same
 - Too many test cases
- Look for inconsistent rules
 - More rules than combination count of conditions
 - Actions are different for the same conditions
- Look for missing rules
 - Incomplete table

Chapter 4: Path testing

Program graphs

- Program graph is a graphical representation of the source code of a program. The statements of a program are represented by nodes and flow of control by edges in the program graph.
- Program graph is a directed graph in which nodes are statement fragments, and edges represent flow of control. (A complete statement is a “default” statement fragment.)
- Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code.
- This method is designed to execute all or selected path through a computer program.

Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.



In the above example, we can see there are few conditional statements that are executed depending on what condition it suffices. Here there are 3 paths or conditions that need to be tested to get the output,

- ☐ **Path 1:** 1,2,3,5,6, 7
- ☐ **Path 2:** 1,2,4,5,6, 7
- ☐ **Path 3:** 1, 6, 7

DD-Paths

Originally defined by E. F. Miller (1977?)

“DD” is short for “decision to decision”

Original definition was for early (second generation) programming languages

- Similar to a “chain” in a directed graph
- Bases of interesting test coverage metrics

☐ A DD-Path (decision-to-decision) is a chain in a program graph such that

Case 1: it consists of a single node with indeg = 0,

Case 2: it consists of a single node with outdeg = 0,

Case 3: it consists of a single node with indeg ≥ 2 or outdeg ≥ 2 ,

Case 4: it consists of a single node with indeg = 1 and outdeg = 1,

Case 5: it is a maximal chain of length ≥ 1 .

Example

```
#include <stdio.h>

#include <conio.h>int main ( ) {

int a, b, c, boolean = 0;

printf (“\n Enter side-a :”);

scanf (“%d”, & a);

printf (“\n Enter side-b :”);

scanf (“%d”, & b);

printf (“\n Enter side-c:”);

scanf (“%d”, & c);

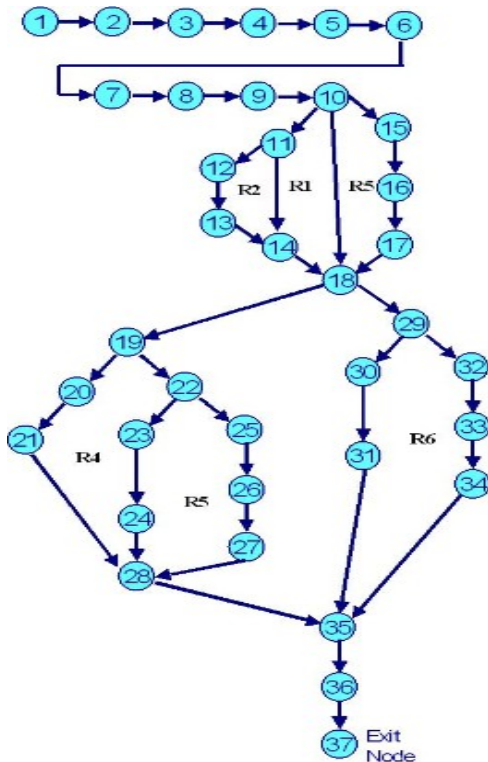
if ((a > 0) && (a < - 100) && (b > 0) && (b < . 100) && (c > 0) && (c < =100)) {

if ((a + b) > c) && ((c + a) > b) && ((b + c) > a)) {

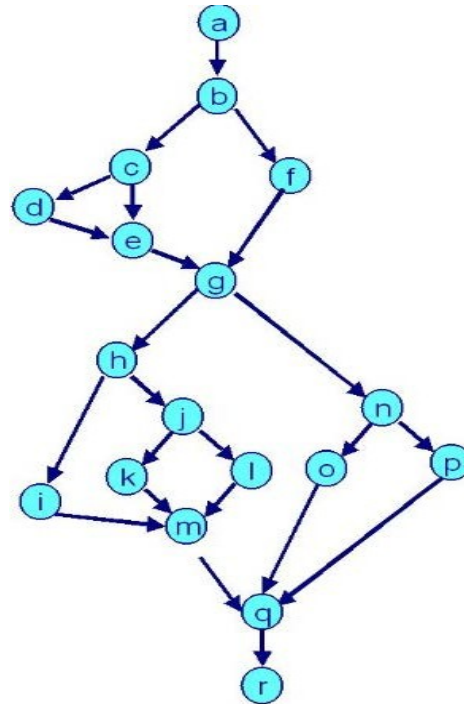
boolean = 1;
```

```
}  
  
}  
  
else {  
  
    boolean = -1;  
  
}  
if (boolean == 1) {  
    if ((a == b) && (b == c)) {  
        printf ("Triangle is equilateral");  
    }  
    else if ((a == b) || (b == c) || (c == a)) {  
  
        print ("Triangle is isosceles");  
  
    }  
    else {  
        printf("Triangle is scalene");  
    }  
    }  
    else if (boolean == 0) {  
  
        printf ("Not a triangle");  
  
    }  
    else  
        printf ("\n invalid input range");  
    getch ( );  
  
    return -1;  
  
}
```

Draw the following Flow Graph



Draw the following DD Path Graph



Test Coverage Metrics

- Used to evaluate a given set of test cases
- Test Coverage Metrics are a device to measure the extent to which a set of test cases covers (for exercises) a program.
- Program graph based coverage metrics:
 - Given a set of test cases for a program, they constitute node coverage if, when executed on the program, every node in the program graph is traversed. it is denoted by G_{node} where the G stand for program graph.
 - Given a set of test cases for a program, they constitute edge coverage if, when executed on the program, every edge in the program graph is traversed. it is denoted by G_{edge} .
 - Given a set of test cases for a program, they constitute chain coverage if, when executed on the program, every chain of length greater than or equal to 2 in the program graph is traversed. it is denoted by G_{chain} .
 - Given a set of test cases for a program, they constitute path coverage if, when executed on the program, every path from the source node to the sink in the program graph is traversed. it is denoted by G_{path} .

Basis Path Testing

- Basis path testing, a structured testing or white box testing technique used for designing test cases intended to examine all possible paths of execution at least once. Creating and executing

tests for all possible paths results in 100% statement coverage and 100% branch coverage.

- Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid of branch testing and path testing methods.
- The objective behind basis path in software testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly.

Example:

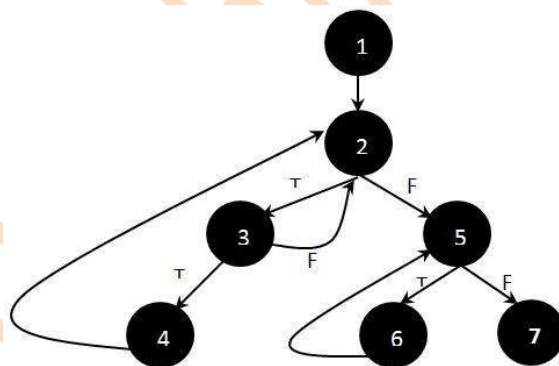
```
Function fn_delete_element (int value, int array_size, int array[])
{
    int i;
    location = array_size + 1;

    for i = 1 to array_size
    if ( array[i] == value )
        location = i;
    end if;
    end for;

    for i = location to array_size
        array[i] = array[i+1];
    end for;
    array_size --;
}
```

Steps to Calculate the independent paths

Step 1 : Draw the Flow Graph of the Function/Program under consideration as shown below:



Step 2 : Determine the independent paths.

Path 1: 1 - 2 - 5 - 7
 Path 2: 1 - 2 - 5 - 6 - 7
 Path 3: 1 - 2 - 3 - 2 - 5 - 6 - 7
 Path 4: 1 - 2 - 3 - 4 - 2 - 5 - 6 - 7

Chapter 5: Data Flow Testing

Data Flow Testing

- ✓ **Data Flow Testing** is a type of structural testing. It is a method that is used to find the test paths of a program according to the locations of definitions and uses of variables in the program. It has nothing to do with data flow diagrams. It is concerned with:

- Statements where variables receive values,
- Statements where these values are used or referenced.

- ✓ Data Flow Testing uses the control flow graph to find the situations that can interrupt the flow of the program.
- ✓ Reference or define anomalies in the flow of the data are detected at the time of associations between values and variables. These anomalies are:
- A variable is defined but not used or referenced,
 - A variable is used but never defined,
 - A variable is defined twice before it is used

Advantages of Data Flow Testing:

Data Flow Testing is used to find the following issues-

- To find a variable that is used but never defined,
- To find a variable that is defined but never used,
- To find a variable that is defined multiple times before it is used,
- Deallocating a variable before it is used.

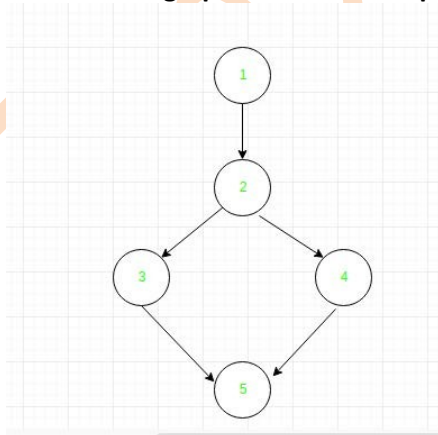
Disadvantages of Data Flow Testing

- Time consuming and costly process
- Requires knowledge of programming languages

Example:

```
1. read x,y;  
2.if(x>y) 3.a=x+1  
   else  
4.a=y-1  
print a;
```

Control flow graph of above example:



Define/use of variables of above example

VARIABLE	DEFINED AT NODE	USED AT NODE
X	1	2, 3
Y	1	2, 4
A	3, 4	5

Slice based testing

- Slices are simpler than the original program and simplify the process of testing of the program.
- Slicing or program slicing is a technique used in software testing which takes a slice or a group of program statements in the program for testing particular test conditions or cases and that may affect a value at a particular point of interest. It can also be used for the purpose of debugging in order to find the bugs more easily and quickly.
- Slicing techniques were originally defined by Mark Weiser and they were only static in nature at that time. Afterwards, Bogdan Korel and Janusz Laski introduced dynamic slicing, which can work for a particular execution of the program.

Program Slicing Tool

- **Spyder**: A debugging tool based on program slicing.
- **Unravel**: A program slicer for ANSI C.
- **Kamkar**: A program slicer for Pascal
- **CodeSonar**: A program slicer for C C++
- **Indus/ Kaveri**: A program slicer for java
- **jSlice**: A program slicer for java
- **SeeSlice**: A program slicer for C

Unit 4

Chapter 1: Software Verification and Validation

Software Verification & Validation

- These are two branches of software testing.
- They are complementary to each other and not substitute of each other.
- Completely dependent on each other.
- Developer's view of quality also known as 'Conformance requirement' is known as Verification
- Customer's view of quality also known as 'Fitness for use' is known as Validation.

Verification

- Verification is an approach to evaluate whether the software product fulfills the requirements or conditions imposed on them by standards or processes, also called as static technique.
- Does not involve execution of any code, program and work product.
- Systematically reading and accessing the contents of work products.
- Helps in identification of defects and its location which then assists in correction.

Advantages of Verification:

- Ensures that the software is following processes correctly defined by organization or customer.
- Not only fixes the defect but also conducts root cause analysis so same defects are not repeated.
- Reduce the cost of finding and fixing defects.
- Each product is corrected and reviewed faster.
- Cost of fixing defect is less.
- Minimum impact on other parts of software.

Disadvantages of Verification:

- Cannot show whether the developed software is correct or not.
- Actual working software may not be accessed by verification as it does not cover any kind of execution.

Verification Workbench

- Verification Workbench is where verification activities are conducted.
- It can be physical or virtual entity.

For every workbench the following entities are required:

1. Input
2. Output
3. Verification Process

4. Check Process
5. Standards, Tools & Guidelines
 1. **Inputs:** There must be some entry criteria definition when inputs are entering the workbench. Definitions must match with the output criteria.
 2. **Outputs:** There must be some exit criteria from work bench as similar to entry criteria, which should match with input criteria for the next work bench. Include review comments and the work product with defects.
 3. **Verification Process:** Describe step-by-step activities to be conducted in a work bench. Describes the activities done with verifying the work product under review.
 4. **Check Process:** Describe how the verification process has been checked. Not a verification of the work product but it is a verification of the process used for verification. Quality plan must define the objectives to be achieved and check process must verify that the objectives have been really achieved.
 5. **Standards, Tools and Guidelines:** These may be termed the tools available for conducting verification activities. There may be coding guidelines or testing standards available for verifications. Sometimes checklist is used for doing verifications.

Methods of Verification

There are 5 methods of verification:

1. Self-Review
 2. Peer Review
 3. Superior Review
 4. Walkthrough
 5. Inspection
1. **Self-Review:** Not considered as an official review in software verification description, as it is assumed that everybody does as self-check before giving work product for further verification. Self-review is excellent in defect prevention through self-learning or retrospection. Defects found in self can help in self-education and self-improvement.

Advantage of Self review:

- Self-reviews are highly flexible with respect to time and defect finding.
- Self-review is an excellent tool for self-learning.
- There is no ego involved as finding is not shared with anybody in self review.

Disadvantages of Self Reviews

- People involved in self review may not conduct a review in reality due to time and focus issues.
 - Something which was implemented correctly at initial development may get changed due to personal issues.
 - Some people with less self confidence may change the correct implementation.
2. **Peer Review :** Author and a peer are involved. Frequently conducted in SDLC. Review is done by peer and review records are maintained. Here code review can be done by fellow developer and test case review can be done by fellow tester. There are two kinds of peer review :

- 1) Online Peer Review
- 2) Offline Peer Review

1) Online Peer Review (Peer-to-Peer Review): In such kind of review, the author and reviewer meet together and review the work product jointly. The extreme programming recommends online review as defects are found and corrected jointly by the peers.

2) Offline Peer Review (Peer Review) : In this kind of review, the author informs the reviewer that the work product is ready for review. The reviewer may review the work product as per his time availability. The review report is sent to the author along with the defects, if any. The author may accept/reject the review comments. This review is prone to mistakes as there is no joint review.

- **Advantages:**

- Peer reviews are highly informal and unplanned in nature. It can happen at any time during development testing life cycle. Mostly, defects are discussed and decision is reached very informally
- Peer review is an excellent tool for educating peers about the artifact.

- **Disadvantages:**

- The other person who is doing the review may or may not be expert and his/her suggestion may not be valid. Peers may fix the defects without recording them, as they may not like to show defects of their partners/friends.
- Approach-related defects may not be fixed, if both the author and the reviewer are at the same status and understanding things in a similar way.

3. **Superior Review:** Superior review is also called as peer review, as a superior may be considered as peer of the author with some more experience and visibility. Superior review is conducted to avoid some limitations of peer review related to approach and visibility etc. A superior such as team leader or module leader may have to review the artifacts made by subordinates.

- **Advantages of Superior Review:**

Approach related defects can be found easily as superiors are expected to have a better view of the entire project. A superior may have better overall requirements and design than the author. As a superior is an experienced person so he can share better ways of doing things whatever he has learned from his own experience.

- **Disadvantages of Superior Review:**

Superior review may become a bottleneck if everything developed by a team needs to be received by the superior. For e.g. if there are 10 people in a team of module leader, then reviewing their work product may hamper his own work. It doesn't work efficiently when the superior is also new to the domain/approach or if the project is in research and development kind of environment where exact solution or even approach may not be known to anybody.

4. **Walkthrough:** Walkthrough is semi formal type of review as it involves large teams along with the author reviewing a work project. In a typical walkthrough some members of a project team are involved in examining an artifact under review. Walkthrough can be effectively used for communication between the team.

- **Advantages of Walkthrough:**
Walkthrough is excellent tool for collaboration where joint decisions can be made by the team. Involvement of each and every member is must in decision making. Walkthrough is also useful for training the entire team at once. Suggestions can be received from the team for further improvement in work product.
 - **Disadvantages of Walkthrough:**
Distributed team is a major challenge in walkthrough. Team members may not be experts in giving comments and may need some training and basic knowledge about the project. Time can be constraint where schedules are tight.
5. **Inspection:** Inspection is a formal way of reviewing a work product. Agenda of inspection is decided in advance. The author of the work product is not allowed/expected to be present during inspection. The people present in the inspection are experts in the domain under consideration. Sometimes, inspector may be external to the organization.
- **Advantages of Inspection:** Expert opinion is available as they are present during review. Inspection should be time effective as availability of expert is limited. Defects are recorded but solutions are not given by the expert
 - **Disadvantages of inspection :** Expert may not be ready with comments before inspection and time may not be utilized properly. Expert opinion may vary from realities it is up to their judgment as expert does not know the need of the customer. The final aim of the organization to get input from the inspector.

Types of Review on the basis of stage/phase

1. In-process review
2. Milestone review
3. Post-implementation review

1. **In process Review :** Ongoing reviews which are conducted during different phases of software life cycle are defined as 'in process review'. They are intended to check whether inputs given are correct or not whether all processes/procedures are following correctly or not. Review takes place any active phase of SDLC (in-between any process of SDLC).

Steps of in process review:

- Work product, and metrics undergoing review are created and submitted to the stakeholders in advanced.
 - Inputs are received from stakeholder to initiate various actions.
 - Risks identified by the project are reviewed and actions may be initiated as required.
 - Risks which no longer exist may be closed.
 - Issues related to stakeholders are noticed and actions are initiated. Issues like software availability, hardware availability, changing requirements, training required etc.
 - Reviews are generated at the end of the review which acts as basis of next review.
2. **Milestone review:** Milestone reviews are conducted on periodic basis depending on the completion of a particular phase or a time frame defined or a milestone achieved during a software development life cycle. Example of such review is requirement review at end of requirement phase, design review at the end of design phase.

Three types of milestones reviews are as follows:

1. Phase End Review
2. Periodic Review

3. Percent Completion Review

1. **Phase End Review:** Phase end review is conducted at the end of a development phase under review such as requirement phase, design phase, coding phase, and testing phase. Typical waterfall cycle is suitable for doing a phase-end review where difference between phases is clearly defined. It may be used during iterative development where one phase completion may be reviewed. Phase-end review also termed 'gate reviews' by some organizations as the gate of phase end opens only when the output criteria of the phase are achieved by the work product.
2. **Periodic Review:** We may have review on some periodic basis such as weekly, monthly, quarterly, etc. Project plan must define various periods when review of project related activities are conducted. Stakeholder may be required to review the progress and take actions if required.
3. **Percent Completion Review:** Percent completion review is a combination of periodic review and phase-end review where the project activities or product development activities are accessed on the basis of percent completion e.g. 5% completion. For high level activity like month may be long duration to get review on lots of things may change in a month. Percentage is the better way to get review.

3. Post-implementation Review:

- Post-implementation reviews are conducted after the project is over and delivered to the customer. It is used by an organization to populate its statistical information.

List of entities performing verification

Verification	Entities Performing Verification
Requirement Review	Business Analysis, System Analysis, Project Team
Design	Project Team, Customer/User
Code	Development Team, Customer/User
Project Plan	Project Team, Customer/user, Suppliers
Test Artifacts	Test team, Customer/User, Development team

SDLC phase and verification techniques used:

Phase	Verification Techniques Used
Planning Documents	Inspection, Walkthrough, Peer Review
Requirements	Inspection, Peer Review
Design	Walkthrough, Peer Review
Coding	Peer Review, Superior Review
Test Plan	Inspection, Walkthrough, Peer Review
Test Scenario	Walkthrough, Peer Review
Test Cases	Peer Review, Superior Review
Test Results	Walkthrough, Peer Review

Review in testing Lifecycle

- Software testing has its own life cycle termed as 'software testing life cycle(SDLC)'. Software testing has various phases which are defined in test planning domain. Here are some reviews which are associated with execution of testing :
 - Test Reading Review (Includes Pre-requisites testing, Updation testing, un-installation testing)
 - Test Completion Review

1. **Test Reading Review** : Test reading reviews are generally conducted by test managers or senior tester with project manager. Test reading review is done at each stage of development such as unit testing, integration testing, interface testing, system testing, and acceptance testing. Test reading review includes the following stages :
 - Review comments for different work products indicate the weak areas in software development and problematic area where defects can be found
 - Test manager must review all comments of the artifacts such as peer review comments, walkthrough or inspection to check whether all comments are closed or not.
 - Review of unit test reports to check whether all unit testing defects are closed or not. If unit testing defects are open then the same defects can be found in system testing.
 - Most of the software requires installation. One need to install software to check whether installation testing has been done successfully. If software cannot be installed, then it cannot be used by the users. The installation defect is called 'showstopper' as they can stop usage of software
 - **Prerequisites Testing**: Software installation has some prerequisites such as operating system, database, and report testing. If requirement specifies that installation then must check for availability of such prerequisites
 - **Updating Testing**: It is checked whether operating system already exist and what is the version of that operating system while installation of operating system.
 - **Un-installation Testing**: Un-installation testing is done when the development organization wishes to check that un-installation is clean. When an application is uninstalled, all the files must be removed from the disk
2. **Test Completion Review**: Test completion review is conducted when a testing cycle is completed and test results are created. Outcome of such review includes analysis of testing process, number and types of defects founds, and number of iterations completed. Final outcome of test completion is a recommendation about status of an application. Through this final outcome the development team would know whether the product can go to the next phase or it needs any repairing, retesting and regression testing.

Coverage in Verification (Test Designing)

Different instances of verification offer different ways of measuring coverage achieved in verification.

1. Statement Coverage
2. Path Coverage
3. Decision Coverage
4. Decision-to-Decision path coverage
 1. **Statement Coverage**: Program includes numerous 'statements.' Definitions of statement' must be done by an organization before calculating any coverage because some people consider ';' as a sign of statement while some people consider a code line with or without ';' as a statement. Sometimes, non-executable lines are also considered as statement as those are also important from maintainability point of view, for example comments. Statement coverage will represent the verified lines against the total number of lines is available in given unit, when a verification activity is conducted. For a unit 100% coverage is targeted but for an application it may not be possible to verify each and every line in an application. In such cases coverage may be less than 100%.
 2. **Path Coverage**: Path represents the sequence of control flow from entry to exit in a given

code. Application may have several paths in which the flow may progress while executing the application. If there are no decision loops in the program, then it will have single path in which it can progress. If decisions are mutually exclusive then number of path increases as per number of decisions. But if decisions are dependent on each other then increase in number of path is exponential. In verification activity, it may give 100% path coverage, if each path is considered for verification. Each path has different probability of happening and some paths have lesser probability of getting executed than the others. In such cases, Path coverage may be less than 100%.

3. **Decision Coverage:** An application is getting executed at several places it may have to make decision depending upon the situation each decision may have several paths. It is not possible to verify all the paths then only decision paths are considered. If outcome is verified on an assumption that all other outcome will be corrected if the selected one is correct then the decision coverage will be less than 100%.
4. **Decision- To-Decision Path Coverage:** There may be a part in between two decisions as if decisions are mutually exclusive then there may be single path between two decisions but if decision is depending upon another decision then there may be several paths between two decisions. Decision coverage may be restricted to the decision part but flow from one decision to next decision is done under Decision To-Decision Path Coverage.

Concerns of Verification

There are few concerns associated with verification activities:

1. Use of Right Verification Technique
2. Integration of Verification Activities in SDLC
3. Resources and Skills Available for Verification
 1. **Use of Right Verification Technique:** Every verification has some advantages and disadvantages One should use right verification techniques like for test cases and code files, Peer-to-Peer review may be more beneficial from cost perspective while for requirement statement, and inspection may be selected as a technique.
 2. **Integration of Verification Activities in SDLC:** Every phase of SDLC is associated with verification and validation Selection on verification technique must find the defect as early as possible This can prevent stage conflict.
 3. **Resources and Skills Available for Verification:** Very important inputs for verification are time and people with required skills If reviewer/inspector is capable with sufficient knowledge and ability, verification can be effective.

Validation:

- Validation is an approach to evaluate whether the final built software product fulfills its specification. It is defined to demonstrate that the product fulfills its intended use when deployed on appropriate environment.
- Validation is also called as 'dynamic testing' as the application is executed during validation with the intention to find defects.
- Validation must be done by independent user, functional experts or black box tester to ensure independence of testing from development activities.
- Validation is basically done by the testers during testing.

Advantages of Validation:

- Black box testing approach is used for system, integration, and acceptance testing.
- It represents actual user interaction with the system without any consideration of internal structures or how the system is built.
- Validation helps in building the right product as per the customer's requirements and helps in satisfying their needs.

Disadvantages of Validation:

- If defects are not found by applying the defined test cases, then it is concluded that there is no defect in the set of transaction executed.
- Stubs and drivers are needed to be designed and used during unit testing and module testing.
- Sometimes, it may result into redundant testing as the tester is not aware of internal structure, and same part of code is executed again and again during validation.

Validation Workbench

A validation work bench is a place where validation activities are conducted on the work products. For every work bench, the following basic things are required. These consists of the following:

1. Input
2. Output
3. Validation Process
4. Check Process
5. Standards, tools and Processes
 1. **Inputs:** There must be some entry criteria definition when inputs are entered into the work bench. This definition should match with the output criteria of earlier work bench.
 2. **Outputs:** There must be some exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include validated with products, defects, and test logs.
 3. **Validation Process:** Validation process must describe step-by-step activities to be conducted in a work bench. It must describe the activities done while validating the work under testing.
 4. **Check Process:** Check process describes how the validation process has been checked. Test plan defines the objectives to be achieved during validation and check process verifies that the objectives have been achieved successfully.
 5. **Standards, Tools, and Guidelines:** These are termed as tools for testing. There may be testing guidelines and testing standards available. Sometimes checklists are used for doing validation.

Levels of Validation

Validation can be applied on various stages of software development.

- **Unit Testing:** Software is made up of many units. Individual unit need to be tested to find whether they have implemented the design correctly or not. Unit testing is done by developer as they are capable of understanding individual units under testing. Thus, unit testing needs a definition, coding, and use of stubs and drivers for testing.
- **Integration Testing:** Integration testing involves testing of many units combining them together to form a sub-module or module. If designing of stubs/drivers are needed to execute integrated parts, then in such cases integration is done by developer. And if integration is executed

independently, then integration is done by tester. Integration testing mainly refers to detail design or low-level design.

- **Interface Testing:** Interface testing involves testing of software with the environment factors (such as database and operating system), where an application is supposed to work. If an application is communicating with other application then third party components (such as communication software) combining all these application and then conducting testing is termed as 'interface testing.'
- **system Testing :** System testing involves end-to-end testing of a system to find the behavior of a system with respect to expectations. System testing is actually a set of processes which may include functionality, user interface, performance and security testing.

Coverage In Validation(Prioritization/Slice Testing)

Following are the famous coverage of Validation:

- **Requirement Coverage:** Requirements are defined in 'requirement specification.' All the requirements are not mandatory. They can be put in different classes such as 'must', 'should be', and 'could be' and are prioritized accordingly. All high-priority requirements expressed as 'must' and lower level priority requirements expressed as 'should be' and 'could be'

- **Requirement Coverage Definition:**

Priority of Requirement	Coverage Offered
P1(Must)	100%
P2(Should be)	50%
P3(Could be)	25%

- **Functionality coverage:** Sometimes, requirements are expressed in terms of functionality required for the application to work successfully. Functionalities are also defined at different priorities as per their requirement.

- **Functionality Coverage Definition:**

Priority of Functionalities	Coverage Offered
P1(Must)	100%
P2(Should be)	50%
P3(Could be)	25%

- **Feature Coverage:** An application may need some features as defined in requirements. Features are a group of functionalities doing similar things. The same feature may have different feature of doing things and each of them is functionality for the application. If we saving a file in Microsoft Word, then 'saving' represents the features while 'different ways to save the file' may represent different functionalities.

- **Feature Coverage Definition:**

Priority of Features	Coverage Offered
P1(Must)	100%
P2(Should be)	50%
P3(Could be)	25%

Acceptance Testing

Acceptance testing is generally done by the user and/or customer to understand whether the software satisfies their requirement or not. There are 3 levels of acceptance testing :

1. Alpha testing

2. Beta testing
3. Gamma Testing
 - **Alpha Testing:** Alpha testing is done by the customer in development environment. The testing is done at development side on dummy data. Alpha testing may be done by tester in front of customer to show that software is working properly. There is no direct impact of such testing on customer as testing is done off-site.
 - **Beta Testing:** Beta testing is done at customer side. Testing is actually conducted by customer in production/semi-production environment. Tester/Developer may be appointed to help customer in testing the application and recording the problem, if any
 - **Gamma Testing:** Gamma testing is used for limited liability testing at selected places. Gamma check is performed when the application is ready for release to the specified requirements and this is performed directly without going through all the testing activities at home. The application is given to few people for using it in production, and feedback is obtained from them.

Management of Verification and Validation(V&V)

The steps in verification and validation are as follows:

1. Defining the processes for verification and validation
2. Prepare Plans for Execution of Process
3. Initiate Implementation Plan
4. Monitor Execution Plan
5. Report Progress of the Process
6. Ensure Product Satisfies Requirements
7. Defining the processes for verification and validation:
 1. **Defining the processes for verification and validation:** The processes involved here are as follows :
 - **Software Quality Assurance Process:** These processes concentrate on developing the techniques/procedures for development and testing of the project. Generally, quality assurance process forms a part of prevention cost, where care is taken so that minimum (zero) defects should introduce in the project. Software Quality Assurance is oriented towards Prevention.
 - **Software Quality Control Process:** These Processes concentrate on developing the approach for verification and validation activities during software development and testing. Quality control process must ensure that defects are found near its origin, so that stage contamination can be reduced.
Software Quality Control is oriented towards defects.
 - **Software Development Process:** These processes concentrate on the approach of developing software with minimum (zero) defects. They may cover quality control processes along with other SDLC processes.
 - **Software Life Cycle Definition:** Software life cycle definition is essential to establish development and testing process for the software. One may select a waterfall or iterative model as per requirement of customer.
 2. **Prepare Plans for Execution of Process:** The Plans involved are as follows :
 - Software development plan and schedule which includes responsibilities and time schedule for verification/validation activities must be defined at the start of the project

- It must define entry and exit criteria for each activity
 - Software Quality Plan and Software Test Plan must be refined if required.
 - Software acceptance testing plan must be defined where acceptance criteria is finalized
 - Software acceptance plan must cover the entry and exit at each stage of development
3. **Initiate Implementation Plan:** Plan should be made at the time of proposal/contract. Implemented during development life cycle of a project. There should be formal records of requirement reviews, design reviews, code reviews, unit testing, integration testing, interface testing, system testing and acceptance testing.
 4. **Monitor Execution Plan:** Project manager and test manager must supervise that the activities defined in various plans are being executed properly and the result are being logged in test log. Analyse problems discovered during execution:
 - Execution of verification and validation process may discover many problems in the product as well as processes associated with development and testing.
 - An organisation is expected to perform a root cause analysis, and take preventive actions to remove the chances of occurrence of the same problem.
 5. **Report Progress of the Process:** The outcome of verification and validation activities as planned must be formally reported to management, customer, and development team to make them aware of the project progress and problems faced by the product/processes during life cycle.
 6. **Ensure Product Satisfies Requirements:** The ultimate aim of verification and validation activities undertaken during project execution is to achieve customer satisfaction. The requirements specified by the customer and defined by the organization must be tested fully to ensure that software meets with the customers expectation and requirements.

Software Development and Validation Activities

The following activities are involved in software development and validation activities:

1. Conceptualization
2. Requirement analysis
3. Design
4. Coding
5. Integration
6. Testing
7. Installation
8. Documentation

1. Conceptualization: Conceptualization is the first phase of development or concepts into reality. Conceptualization means converting the thoughts or concepts into reality. In Conceptualization, the main stress of verification and validation activity is to determine feasibility of an approach for the project. Feasibility may be depending upon various factors such as technical feasibility, economic feasibility and skill availability.

2. Requirement Analysis: Requirement phase may start from conceptualization. Requirement verification and validation shows the feasibility of requirements and gives inputs to design phase. Requirement reviews helps in understanding different aspects of customer needs as well as various trade-off done.

3. Design: Requirements are implemented through design. Verification and validation of

design ensures that design is complete in all aspects and matches with requirements. Design validation involves dummy execution of a product design through data flow diagrams and prototyping, to find whether the design reflects the requirements with feasibility.

4. Coding: Coding involves code review, Verification and validation of coding ensures that requirement and design are correctly implemented.

5. Integration: Integration verification and validation shows that the individually tested units work correctly when brought together to form a module or sub-module

6. Testing: Test artifacts such as test plan, test scenario, test bed, test cases and test data must be subjected to verification and validation activities. Test plan must be covering all the aspects of testing expected by the customers. Test bed must mimic real life scenario. Test scenario must be completed and feasible, Test cases for acceptance testing must be validated by customer/user/business analyst.

7. Installation: Application must be tested for installation, if installation is required by the customer. The documentation for installation of an application must be covering all the instruction and guidance.

8. Documentation: There are many documents given along with a software product such as installation guide and user manual. Document must be complete, detailed, and informative, So that it can be referred by common user.

Chapter 2: V-test Model

- It starts when proposal of software development is made and ends when application is finally delivered.
- In case of maintenance, the cycle may get repeated for every instance of change in present system.
- For Customer, it starts from problem statement or conceptualization of new product, and ends with satisfactory product, acceptance and usage.
- There is a testing activity associated with development activity, to achieve milestones with minimum problem. This is termed as 'certification of testing or gate approach of testing.
- Each phase of software development activities must consider corresponding testing activity associated with it.
- Project plan and estimations in terms of time and effort for a project must consider all the activities of testing along with development activities corresponding to different phases.

V-model for software

- It is a Validation Model. Validation model describes the validation activities associated with different phases of software development.
- At the requirement phase, during system testing and acceptance testing, the system tester and users confirm that requirements have been really met or not.
- Design phase is associated with interface testing which covers design specification testing as well as structural testing.
- Program-level designs are associated with integration testing

- At code level, unit testing is done to validate individual units.

1. **Requirement Analysis:** Requirements are gathered, analyzed and studied in this phase. It is not important how the system is implemented but important is what the system is supposed to do. Here, the product requirements are understood from customer's perspective. It involves detailed communication with the customer to understand his expectations and exact requirements.

2. **System Design:** It is a time to design the complete software, after getting the product requirements. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design.

3. **Architectural Design:** In this phase, the data transfer and communication between the internal modules and other system is clearly understood.

4. **Module Design:** In this phase the system breaks down into small modules. The detail design of system modules is specified, known as Low-Level Design (LLD)

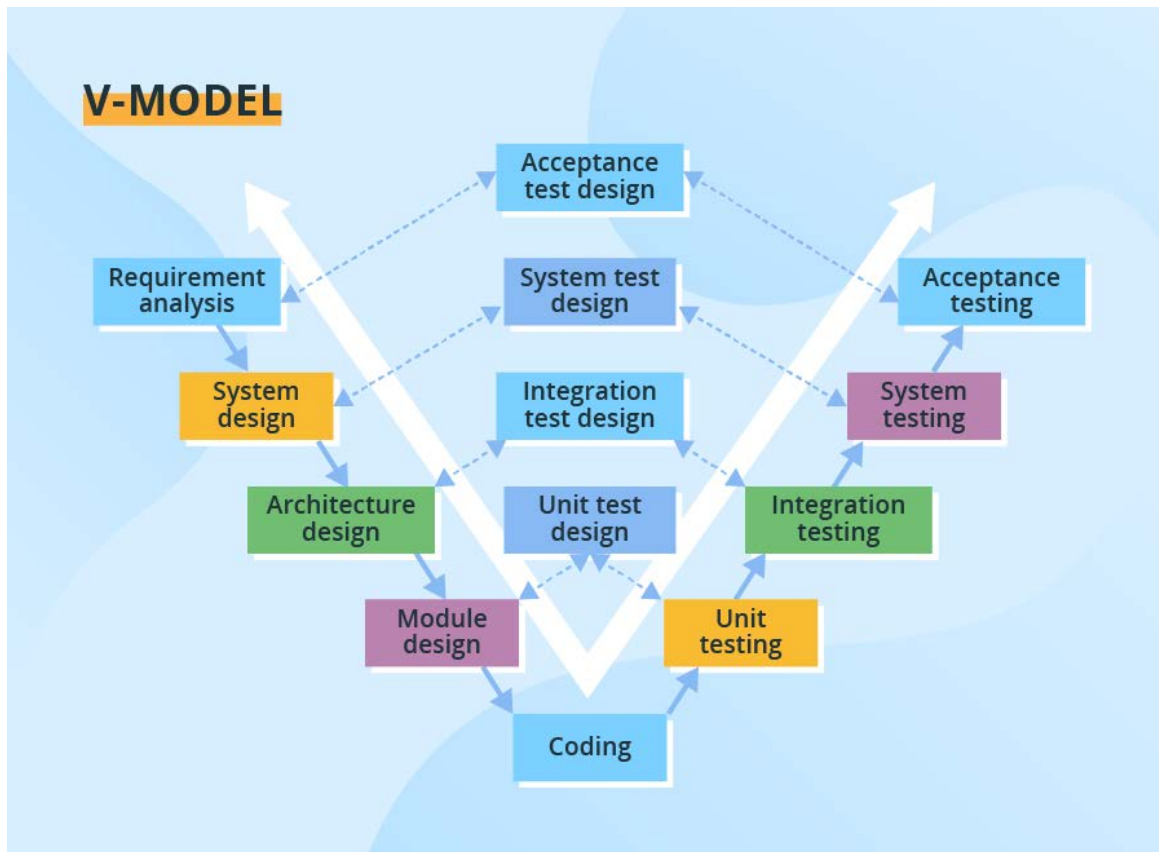
5. **Coding phase:** The actual coding of the system modules in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

6. **Unit Testing:** Unit testing is a white box testing technique, where a piece of code is written which invokes a method (or any other piece of code) to test whether the code fragment is giving the expected output or not. This testing is basically performed by the development team. In case of any anomaly, defects are logged and tracked.

7. **Integration Testing:** In this phase the integration test cases are executed which were created in the defects are logged and tracked. Integration testing is a technique where the unit tested modules are integrated and tested whether the integrated modules are rendering the expected results. In simpler words, it validates whether the components of the application work together as expected.

8. **System Testing:** In this phase all the system test cases, functional test cases and nonfunctional test cases are executed. In other words, the actual and full fledge testing of the application takes place here. Defects are logged and tracked for its closure. Progress reporting is also a major part in this phase. The traceability metrics are updated to check the coverage and risk mitigated.

9. **Acceptance Testing:** Acceptance testing is basically related to the business requirements testing. Here testing is done to validate that the business requirements are met in the user environment. Compatibility testing and sometimes nonfunctional testing are also done in this phase.



Testing During Proposal Phase

- Requirement starts from system proposal.
- A proposal is created when the customer asks for information, quotation, proposal etc.
- At proposal stage, system description may not be very clear, it must talk about the major problems to be solved, and the possible solution for which the system is designed.
- It may also consider major constraints related to people, technology etc.
- Feasibility may or may not be part of the proposal, sometimes, feasibility study also be considered as it may involve some cost and efforts.
- Feasibility study is done by the Customer as well as supplier in search of a possible solution to solve the problem faced by the customer.

Testing During Requirement Stage

This stage covers all the technical, economic, legal, and operational and system requirements. Characteristics of good requirements are mentioned below:

- Adequate
- Clear/Unambiguous
- Measurable
- Feasible
- Not Conflicting With Each Other
- **Adequate:** The requirement must explain the entire system covering end-to-end scenario from the user's side. All the assumptions and constraints/limitations in the system must be defined and documented with possible solutions. Requirement should not talk about any happening

which is impossible thing.

- **Clear/Unambiguous:** Customer's expectation must be reflected in requirements clearly. Requirements must be clear for designer as well as developer. It must describe various functions, outputs in various forms, system performance requirement, and interface with other system. There should not be any doubt about the outcome of the system working for the customer as well as for developing organization.
- **Measurable:** Tests defined from requirements must have the expected results, possibly in numerical form or at least in measurable terms which can be verified during testing. Requirement can be a definite numbers or a range with boundaries defined with terms like minimum and maximum, etc.
- **Feasible:** Requirement must be feasible It must not refer to thing which is impossible or not implemented with given technology, approach, software or system configuration.
- **Not Conflicting with each other:** Two requirements must not specify anything which is contrary with each other. Generally requirements are prioritized to avoid any conflict. If any conflict is there, then requirement with greater with priority would be implemented.

Testing During Test Planning Phase

Verification of testing artifacts may include the following:

1. Generate Test Plan to Support Development Activities.
2. Generate Test Cases Based on System Structure.
3. Analyze Requirement/Design coverage.

1. Generate Test Plan to Support Development Activities: The Test plan must consist of application development methodology, schedule and deliverable. Test plan must refer to test approach or test strategy, test objectives, scope of testing, assumptions and risk associated with testing. It must contain methods used for defining test data, test cases, test scenario, and execution of testing activities.

2. Generate Test Cases Based on System Structure: Functional test cases must be based on functional requirement, and structural test cases must be defined on the basis of design and non-functional requirement of the system. It must be used to define test data using different techniques available such as boundary value analysis, equivalence partitioning, error guessing, and state transition.

Test cases and Test data must be derived from Test scenario.

3. Analyse Requirement/Design coverage: The test manager decides the objectives of requirement coverage and design coverage in test plan, Customer must be involved in making decisions. Coverage less than 100% indicates a risk. In case of any shortage in coverage with respect to objectives defined, the test team must analyse the situation and perform risk-benefit analysis of lesser coverage with the help of customer and take corrective measures, if required.

Testing during design phase

Verification and Validation of design may include the following:

1. Consistency with respect to requirements
2. Analyses Design for Error
3. Analyze Error Handling
4. Developers verify information flow and logical structure.
5. Testers inspect design in detail.

1. **Consistency with respect to requirements:** Design must be consist of requirement defined. Sometimes, customer expectations as defined in requirements may be contradicting with each other, one must perform trade-off. If there is any trade-off between the requirements for implementation it must be mentioned clearly in design, and customer approval should be taken
2. **Analyse Design For Error:** Design generated must be reviewed and tested for completeness and accuracy. Design is implemented by coding. Errors in design will directly reflect the errors in coding and application developed. Reusability can create robust design and flexible code. Design must be optimized.
3. **Analyze Error Handling:** A design must define the error-handling process during use of application. It must consider all possible errors such as erroneous data input and invalid transactions, and how design must handle them. All kinds of error handling must be covered in designing aspects. An organisation must have standards for error handling, error messages and user interaction so that designers should be clear to handle them during designing.
4. **Developers verify information flow and logical structure:** The designs must be analyzed for logical flow of information during transactions. Data flow diagrams and dummy execution of the system is done, and derived outputs are measured against expected output. Design must be consistent with the requirement and user expectation.
5. **Testers inspect design in detail:**
Testers use design for defining structural test scenario, test cases, and structural test data. Test scenario must be end-to-end scenario considering data flow in the system must contain valid as well as invalid conditions, and error handling during user interactions with the system.

Aspects to be checked during design phase:

- Missing test cases
- Faulty Logic
- Module interface match
- Data Structure Inconsistency

Testing During Coding

Aspects to be checked:

- **Coding Standards/Guidelines Implementation:** Many organizations have coding standards defined. While reviewing code one must make sure that all these guidelines and standards are fulfilled.
- **Coding Optimization:** Coding standards must also talk about how nesting is done, how variables are declared and how functions are declared.
- **Unit Testing:** Done by developer to ensure that written code is working as expected.

VV Model

- VV Model is known as Verification and Validation Model. VV model considers all the activities related to verification and validation. It is also termed as Quality Model. Various Activities of VV

Model associated with each phase of software development life cycle :

- **Requirement:** Includes requirement verification and requirement validation.
- **Design :** Includes design verification and design validation
- **Coding:** Includes code verification and code validation.

Critical Roles and Responsibilities

- Depending upon the organization size, maturity and type of project the following roles are performed by different entities:
 1. **Development:** Project planning activities include requirement elicitation, estimation project planning, scheduling, and definition of quality attributes required by the customer. Project planning is the foundation of project's success. Generally senior people in development (such as project manager) may have the responsibility to create project plan. Organisation must have adequate resources which are required by the project like the number of people, machines, hardware, software, and tools.
 2. **Testing:** Interaction with customer and other stakeholder to gather requirements for project. Defining policies and procedures for creating development work, verification and validation activities to ensure is built properly, and delivering it to test team/customer. Test planning including test strategy definition, and test case writing Test planning may include estimation of efforts and resources required for testing. Interacting with customer and other stakeholder as per project requirements. It may include asking queries, responding to customer/development team. Defining policies and procedures for creating and executing tests as per test strategy and test plan. Testers may have to take part in verification and validation activities related to test artifacts.
 3. **Customer:** Customer may be the final user group, or people who are actually sponsoring the project Customer can be internal to an organization or external to the organization Roles and Responsibilities of a customer may include:
It may also include solving any queries or issues raised by development/test team. Participating in acceptance testing as per roles and responsibilities defined in acceptance test plan. A customer may be responsible for alpha, beta, and gamma testing. Review and approve various artifacts as defined by contract or statement of work.

Unit 5

Chapter 1: Levels of Testing

Levels of Testing

- A process where every unit or component of a software/system is tested. Following are the various levels of testing:

1. Proposal Testing:

- Proposal is made on the basis of Request for Proposal (RFP) or Request for Information(RFI) or Request for Quotation(RFQ) by customer.
- Proposal Review includes customer, business analyst, system analyst and project manager and most important member's organization committees.
- Proposal testing includes customer, business analyst, system analyst and project manager.
- Proposal testing includes technical review, commercial review and validation proposal.

2. Technical Review:

- Technical Feasibility of proposed system or application
- Availability and requirement of skills sets, hardware and software and other requirements of the system.
- Technical proposal is overall approach as per the discussion with the customer.

3. Commercial Review:

- Focuses on gross margins of product fund in terms of money going out and coming in development organization.
- Several iteration of Proposal change before all parties involved in Request For Quotation (RFQ) and proposal agrees on some terms and conditions for doing a project.

4. Validation Proposal:

- A proposal sometimes involves development of prototypes or proof of concept to explain the proposed approach to the problem of the customer.
- After this process the proposal gets converted into contract.

Requirement Testing

- Here test cases, conditions and data are derived from requirements. Includes functional and non-functional attributes like performance, reusability or usability. Addresses to major issues:
 - Validating that the necessary requirements are correct, complete, unambiguous and logically inconsistent.
 - Designing a necessary and sufficient set of test cases from the requirements.
- **Stages in Requirements based Testing:**
 - **Defining Test Completion Criteria** - Testing is completed only when all the functional and non-functional testing is complete.
 - **Design Test Cases** - A Test case has five parameters namely the initial state or precondition, data setup, the inputs, expected outcomes and actual outcomes.
 - **Execute Tests** - Execute the test cases against the system under test and document the results.
 - **Verify Test Results** - Verify if the expected and actual results match each other.
 - **Verify Test Coverage** - Verify if the tests cover both functional and non-functional aspects

of the requirement.

- **Track and Manage Defects** - Any defects detected during the testing process goes through the defect life cycle and are tracked to resolution. Defect Statistics are maintained which will give us the overall status of the project.

- **Requirements Testing process:**

- Testing must be carried out in a timely manner.
- Testing process should add value to the software life cycle, hence it needs to be effective.
- Testing the system exhaustively is impossible hence the testing process needs to be efficient as well.
- Testing must provide the overall status of the project, hence it should be manageable.

Design Testing

- Creation of Data Flow Diagram, Activity Diagram, Information Flow Diagram & State Transition Diagram Software design gives three types of results :
 - i. **Architectural Design:** Identifies that software as a system with many components interacting with each other.
 - ii. **High-level Design:** Focuses on how system along with its components can be implemented in the form of modules.
 - iii. **Detailed Design:** Logical structure of each module and their interface to communicate with other modules.
- The characteristics of good design are as follows:
 1. Clarity
 2. Complete
 3. Traceable
 4. Implementable

Code Review, Unit Testing

- Reviewing Code File, database schema, classes, object definition, procedures and methods applied to ensure that design is implemented correctly by developers and all the guidelines and standards are followed.
- Characteristics of Code Review Testing are as follows:
 1. Clarity
 2. Complete
 3. Traceable
 4. Maintainable

Unit Testing

- Individual software component or module
- Done by programmer as it involves detailed knowledge of internal program design and code.
- Unit testing is performed in debugger mod.
- Gray box testing is also considered as "Gray Box Unit Testing".

Module Testing

- Units come together and form a module.
- Module works on its own or many component.
- Testing the individual subprograms, subroutines, classes or procedures in a program.
- White box oriented.
- **Objective:** Proper functioning of the module but to demonstrate the presence of an error in the module.
- Allows Parallelism as multiple modules can be tested simultaneously.

Integration Testing

- Individual units are combined and tested as group.
- Expose faults in the interaction between integrated units.
- Verify the functional, performance and reliability between the modules that are integrated.
- Approaches of Integration Testing:
 1. Top Down
 2. Bottom Up

Big Bang Testing

- All components are integrated together at once and tested.
- The testing team receives the entire software in a bundle.
- Individual modules are not integrated until and unless the modules are ready.
- All modules are integrated without performing any integration testing and then its executed to know whether all the integrated modules are working fine or not.
- Because of integrating everything at one time if any failure occur then it becomes very difficult for the programmers to know the root cause of that failure.

Sandwich Testing

- This approach is a hybrid of top-down and bottom-up methodology.
- Stub and drivers are used for incomplete or not developed modules.
- **Testing Approach:** A middle layer is identified from which bottom-up and top-down testing is done. The middle layer is also known as target layer.
- **Top down testing** starts from middle layer and moves downwards towards lower level modules. This layer below middle layer is known as bottom layer.
- **Bottom-up testing** also starts from middle layer and move up towards top layer modules. This layer above middle is known as Top Layer.

Critical Path First Testing

- Development team concentrates on design, implementation and testing of critical path of a system first.
- This is also term as 'skeleton development and testing.' It is applied where critical path of a system is important for system performance and for business from customer's perspective.
- One must understand the criticality of system functions from user perspective or from business perspective and decide on the priority of testing.
- Critical path first is generally used where complete system testing is impossible.

Sub System Testing, System Testing, Testing Stages

- **Sub-System Testing:** Same as integration testing. Involves collection of units, submodules and modules which have been integrated to form subsystems.
The subsystem can independently and implemented also can be separately executable.
- **System testing:** System testing representing final testing done on system before it deliver to customer. It is done on integrated subsystem that make up the entire system ,or the final system getting deliver to the customer. System testing goes into following two stages:
 - **Functional testing :** It checks all the functions as per requirementsdefinition are working or not.
 - **User interface system:** Once the functionalities are set correct, next step isto set the user interface correct.
- Once system is tested for functionalities and user interface it is ready to go for other types of testing such as security, load and capability. 'It depends upon the scope of testing.
- **Testing Stages:** Generally an organization performs unit testing first part of testing. Input from unit testing is use to identify and eliminate defect at unit level so that they will not go further down. Once unit testing over and units are ready for integration than integration or module testing is done. Then integration module testing is done. Module testing may be done by the development team. After Module testing the system goes subsystem testing is done module testing may done by test team depending upon requirements. Once system is through with system it is taken for acceptance testing to check whether it fulfill that acceptance criteria.

Chapter 2: Special Tests

- The Special testing must be included in a scope statement for testing and in the requirement specification defined by the customer or businessanalysts, and system analysts. Such type of testing is termed as "Special Testing "as it may or may not be done in general, for any application Special Testing may need some special testing skills; tool, and techniques.
- 1. **Specialized system and application:** Some specialized system or special user requirement testing is considered in this type of testing. These system may be made and used for special Purpose, and need to have some special characteristics example of such system may be e-business system, security system antivirus application, operating and databases.
- 2. **Complex Testing:** Complexity testing is a verification technique where complexity of system design and coding is verified through reviews, walkthroughs or inspections as per planned arrangement.
- 3. **Control Flow Graph:** Describes the logical structure of software unit, wheredecisions can go while executing instructions.

Graphical User Interface (GUI) Testing

GUI include the following:

- All the color combination should be used properly ,the user be able to identify entities on the screen correctly and efficiently.
- Very small or very large font, caption not aligned properly, or spelling mistake can create a

[For detailed Video Lecture Download The Shikshak Edu App](#)

negative impact about the application , so all words ,fonts and alignments should be used properly .

- Error message and information given to user must be meaningful and helpful. Message Must guide user to perform the correct actions.
- GUI defects are considered a high priority defects.

Compatibility Testing

- Any System is made up of many components such as different type of machines, Printers, hardware, and different supporting software such as operating system, database and communication system.
- If the product cannot work with other variables in the user environment, then it restricts the market .so, the aim of manufacturer should be the product is being developed must be working on all possible scenario of these components.
- Compatibility testing refers to testing the software on multiple configurations to checks the behaviors of different system components and their combinations.
- The Variable Can be:
 - Operating Systems
 - Databases
 - Browsers
 - Languages
- The Hardware can be:
 - Machine and servers
 - Routers
 - Printers
- Integration with other communication system can be:
 - Mailing Software
 - Messages Software
- Different Languages and across the globe can be:
 - Chinese, Korean, or Japanese
 - Hindi, Urdu, or Hebrew
 - English, German and French
- Multiplatform testing is an example of compatibility Testing.
- Multiplatform Testing: Multiplatform testing is one set of testing in compatibility testing Where system is expected to work with various platforms. Multiplatform testing is a testing of software to check their performance on different platforms. Various Platform gets updated frequently as technology changes. Multiplatform testing involves verification and validation activities that software does its intended function in the same/similar way when platform is changed.
- **Types of Compatibility**: For Multiplication testing, there are three main types of compatibility:
 - Friend Compatibility
 - Neutral Compatibility
 - Enemy Compatibility

Security testing

- Security testing is a special type of testing intended to check the level of security and protection offered by an application to the users against unfortunate incidences. The incidence could be loss of privacy, loss of data etc. There are some weak Points in a system which is vulnerable to outside attacks/unauthorized entry in the system. Some Definitions associated with security are given below:
 - Vulnerability
 - Threats
 - Perpetrators

Performances Testing

- Performance testing is intended to find whether the system meets its Performance requirement under normal level of activities. Performance criteria must be measurable in quantitative terms such as time in "Milliseconds". Some examples of performance testing can be given below:
 - Adding a new records in databases must take maximum five milliseconds. It means that when the record is added into database must take maximum five milliseconds. It means that when the record is added into the database, then it may take less or equal to five milliseconds .
 - Database containing one million records must not take more than one second while searching a record in database.
 - Sending information across the system size of 1 MB with a network of 512 KBPS must not take more than one minute.

Volume Testing

- Volume testing means maximum volume or load. It can be described in terms of concurrent users, number of connections, maximum size of an attachment to be transferred over a network, etc. performance is not checked under volume testing, only factor to be assessed is whether the system is living or dead due to increased volume or load on the system.

Stress Testing

- Stress testing talks about the system resources required by the transactions that have been planned to be undertaken. If the system has limited resources available then the response of the system may go down due to unavailability or loading of resources. Stress testing is governed by 'factor of safety' expected by the user to protect system failures due to resource constraints. Stress testing is used to define the resource level required by the system for its efficient and optimum performance. Example of stress testing: Simple transactions are created, and the system resources such as processor size, RAM size and bandwidth are reduced to find the minimum level of resources where system is living. Transaction selected must represent very high probability of happening.

Recovery Testing

- Recovery testing is intended to find how the entire system recovers from a disaster. There are various methods of disasters recovery and system requirements/designs must specify the methods expected to be used during recovery, Disaster recovery is a subset of business continuity planning. Recovery testing may include:

- System Recovery
- Machine Recovery

(1) System Recovery:

- System Returns to the points of integrity after meeting disaster.
- Storing data in temporary location.
- Completing the transaction

(2) Machine Recovery:

- Cold recovery
- Warm recovery
- Hot recovery

Installation Testing

- Involves installation process, un-installation process, upgradation process and requirement testing.
- **Installation Process:** Installation may be done through devices like CD, pen drives etc, or it done by using networks, or from one machine to several machines at a time. Sometimes, remote installation is also required. Installation process can be fully automatic or it may need user action.
- **Un-Installation Process :** Un-Installation testing is used where user requirements define the same in requirement document. Un-Installation must clean all the components and files installed during installation
- **Upgradation Testing :** An application may need an upgradation, in order to upgrade from old version to new version. Upgradation may be done by using patches released by the manufacturer time to time. Upgradation also follow similar process as that of installation.

Requirement Testing (Specification Testing)

- Requirement testing is also known as Specification Testing.
- Objectives of requirement testing include the following:
 - User requirements are implemented correctly or not
 - Correctness is maintained as required by customer
 - Processing compiles with organizations and customers policies and procedure.
- Methods of requirement testing include the following:
 - During software development, need to trace the requirements to determine whether all requirements are implemented or not.
 - Tracing process may include designs, coding, test cases and finally test results.

Regression Testing

- Intended to determine whether the changed components have any error in unchanged components of the system. Regression testing may not be considered as special testing in development plan. But often, maintenance projects may consider it as special testing as regression testing of entire application may be very costly.

Error Handling Testing

- Error handling testing is done to determine the following :
 - It has ability to properly process erroneous transactions and protect the user from making any mistakes during data entry
 - It is possible that the user may be prevented to enter erroneous transaction by giving error messages to user, when such kind of entry or processing is detected by the system.
 - All expected error are recognized when they occur in the system and the appropriate error message may be given to use. It must help user to identify and rectify the error defined by requirement statement.
- There may be several types of error messages such as following :
 - **Preventive Messages:** When user entering some wrong data it is identified by the system and preventing the system.
 - **Auto-Corrective Messages:** Auto corrective message is when the user typing some wrong text and it is corrected automatically.
 - **Suggestive Messages :** In case of suggestive messages, system tells the user about what is wrong and provides suggestions about correcting it. User can overrule the suggestion and may re-enter transaction again or may accept the suggestion.
 - **Detective Messages:** Detective message is when system tells the user about what is wrong but there is no guidance available about correct transaction.

Manual Testing

- Manual Testing is intended to test the interface between people as users of an application and application system. Manual Testing is used to determine whether:
 - 1) Manual support Procedures are documented, complete and available to user. It can be in the form of online help or user manual or trouble shooting manual.
 - 2) Manual support people are well trained to handle various conditions of working including entering data, processing, taking output as well as handling errors. People must be able to work with systems independently.
 - 3) Manual support and automated segments are properly interfaced within the application. "Help" available must provide the guidance to common user when some problem is encountered by them.
 - 4) User must be able to work with the system without assistance of system personnel

Intersystem Testing

- There are Possibilities that the system developed may have to interact with many other supporting system or with existing system before the system is installed.
- System testing is designed to determine whether,
 - 1) Parameters and data are correctly passed between application and other system to avoid any communication failure. Acceptance of Data and output of data must match with the requirements.
 - 2) Documentation for system must be accurate , complete, and matching expected inputs and outputs. It must define parameter passing and bridge of communication between various

system.

- 3) System testing must be conducted whenever there is a change in the parameters between applications communicating with each other.
- 4) set of transactions is prepared on one system then passed to another system for processing and the results are verified for the corrections.

- Manual verification of documentation is done to understand the relationship between different system .

Control Testing

Control testing is done to determine the following:

- 1) User must complete all mandatory fields before saving the record, if mandatory fields are not completed then it must stop the user by sending messages and transfer controls on mandatory fields
- 2) Transactions entered in the system are authorized by identifying the users permission unauthorized person may not able to access the records ,or may not able to save it, modify it or delete it.
- 3) Process must meet the user needs.It must be supported by requirement statement.
- 4) Transactions must be traceable from start to end in entire data processing life cycle
- 5) Ensure integrity of processing of transactions and data from entry till exit.Data must not be lost , modified or added during the transaction Processing.
- 6) Identify risks associated with different users using the system ,and their ability to interact with the system
- 7) Create risk conditions in the test laboratory to access the level of control mechanism
- 8) Controls must be effective , efficient and must justify their presence.

Smoke Testing

- Generally ,these tests are performed without any user input .steps involved in smoke testing can be installation navigating through the applications , invoking or accessing some major functionalities.
- Smoke testing is not approving or certifying the application. It just tells the tester whether the application but not ensure that the normal user will be able to work with it . If smoke testing fails, user will not be able to work with the applications. Such failure may result into rejection of an application without going further.
- Generally, smoke testing is done by test manager or senior tester. Smoke testing is also termed as “smell test” as test manager may have to make a judgement about the system.

Adhoc Testing (Monkey Testing, Exploratory Testing, Random Testing)

- Adhoc testing is performed without any formal test plan, test scenario, test cases or test data. It is also called 'exploratory testing' or 'monkey testing or random testing'. Tester try to test the system with different combinations of functionalities on the basis of error guessing

and experienced on similar application in past. This helps in identifying hidden defects that might have been missed in all previous test efforts.

Parallel Testing

- Done by comparing the existing system with newly designed system to validate it against existing system. It is used to compare results from two different systems. Two different systems are run in parallel to find out the similarities and differences between them. Parallel testing is done in acceptance phase, typically in beta testing or business pilot where existing system or manual operations are compared with the new system being developed.

Execution Testing

- Execution testing is performed to ensure that system achieves desired level of ability in production environment when normal users are using in normal circumstances. Execution testing may be considered as alpha testing if it is done in development environment, or beta testing if it is done in user environment. Execution testing may be conducted by using hardware/software monitors, by simulating the functioning of the system, and by creating programs to evaluate performance of completed system.

Operations Testing

- An operation testing is performed to check that operating procedures are correct as documented in user manual, and application is executed properly using the documentation available with it. Operation testing is done to determine whether the system documentation is complete of user manual, etc. People must be able to work with the system by referring to these documentations. Complete training is given to user, if required as per requirements and user can use the system on the basis of training provided.
- Effectiveness of training may be evaluated in this testing. Operation testing must occur prior to placing the application into production environment. User must be capable of working with system. Operation testing must be conducted without any assistance.

Compliance Testing

- Compliance testing is intended to check the application/development processes with the standards applicable to such applications. There may be some regulatory or statutory requirements enforced by different agencies in the environment where application is expected to work. Compliance testing is done to check whether system is developed with prescribed standards, procedures and guidelines applicable to them as per domain, technology, customer, etc.
- Compliance testing is done to determine whether:
 - Development and maintenance methodologies are followed correctly or not while development/maintaining system belonging to different domain.
 - Documentation must be clear and complete and must be complying with standards applicable
 - Customer requirement must include definition of regulatory and statutory requirements
- Compliance testing is done using the following:
 - Checklist prepared for evaluation or assessment of product.
 - Peer reviews to verify that the standards are met.

- SQA reviews by quality professionals.
- Internal audits.

Usability Testing

- Usability testing is done to check 'ease of use' of an application to a common user who will use the application in production environment. It involves user guide and help manual (including online help) available with application.
- **Usability testing is done by:**
 - Direct observation of people using the system, their interaction with system and ease with user can work with the system under testing.
 - Conducting usability surveys by checking the development or implementation of system in production environment.
 - Beta testing or business pilot of application in user environment.
- **Usability testing checks for human factor problem such as:**
 - Whether outputs from the system such as printouts and reports are meaningful or not. They must be evaluated from user perspective of fit for use.
 - Error messaging must help user in using system.
 - Is the application easy to use to user?
 - System must provide online help or user manual to get self service
 - Consistent in its function and overall design

Decision Table Testing (Axiom Testing)

- Decision table is a good way to capture system requirements that contain logical conditions, and to document internal system design handling various conditions faced by the system. Specifications are analysed, and conditions and actions of the system are identified.
- Decision table contains the triggering conditions, often combinations of true or false for all input conditions, and resulting actions for each combination of conditions.
- Axiom testing is a part of decision table where system works on some relationships between variables, when two variables X and Y are tested which are related to each other with a relationship expressed as $Y=f(X)$, it may be termed as axiom testing.

Documentation Testing

- Software development life cycle generates many artifacts which are not part of final system, or executable, but are very important to understand the system in future. This is termed as 'System Documentation'.

Training Testing

- Sometimes, vendor is expected to give training to user who will be using the system. A best approach is to test training as part of system testing.

Rapid Testing

- Powerful technique that can be used to complement conventional structure testing. Based on exploratory testing techniques, and is used when there is little time available to obtain full test

coverage using conventional methodologies. Rapid testing finds the biggest bug in shortest time.

- Areas where Rapid Testing is used :
 - Proof of concept test early in the development cycle
 - Prior to, or following migration from development to the production environment.
 - Sign-off development milestones to trigger funding or investments.
 - Prior to public release or delivery to the customer.

Control Flow Graph

- Control Flow Graph is flow of a control when a program is getting executed. If a program does not have any kind of decision, flow happens in a single direction. Whenever there is any decision to be taken by a program, there is a possibility of different flow graphs possible. Each decision includes multiple paths in a program while it is getting executed.
 - Dominators: When we begin from the start of a program, and there exists a set of code which will always be executed when path is selected, then it is termed as dominator.
 - Post-Dominator : From any point in an application if we go to the end of the application, If we have to go through particular set of code then it is defined as post-dominator'.
- **Program Dependence Graph:** When the program is getting executed and execution may be dependent on two factors, data dependence and control dependence.
- **Category Partition Method:** Category partition method is used to generate the test cases from requirements. The following steps are involved in generating test cases by category partitioning methods:
 - **Analyze the requirements:** The requirements are put into different sets like functions, user interface, and perform requirements. These may be tested independently and they are put in different test suites.
 - **Identify Categories:** Some specified input are analyzed which will have expected result on categories of output.
 - **Partition the Categories:** Input with similar output and output with similar input are put into different partitions or classes.
 - **Identification of Constraints:** Some categories of inputs and outputs which cannot exist together or which are impossible. They must be identified.
 - **Creating Test Cases Accordingly:** Test cases are created to check that possible constraint have been defined in user requirement.
 - **Processing the Test Cases:** The test cases defined above are executed and results are captured.
 - **Evaluate the Output:** The output is verified with expected output.
 - **Generate the test sets:** If the expected output and obtained output are mismatched then it may be added in test set.

Generating Tests on the basis of Combinatorial Design

- An application is expected to work under various environmental configurations such as hardware, browser and operating system. This is called as Test Configuration.
- Combinatorial Test Design Process includes modeling the input space and test environment, generating combinatorial object and generate test and test configurations.

State Graph

- State graph and state table are useful for describing software behavior under various input conditions. State testing approach is based upon the finite state machine model for the structures and specifications of an application under testing. If we draw a state graph, states are represented by nodes, whenever an input is provided the system changes its state. This is also called as 'State Transition'

Risk Associated With Technologies

Explain risk associated with new technologies

- Technology itself is defective or not matured enough to use Technology is inefficient as it is not matured.
- Technology is incompatible with other relevant technology in use New technology makes existing technology obsolete.
- Variation between technologies and delivered and documentation provided.
- Lack of knowledge and Skill for optimal technology usage.

Process Maturity Level of Technology

- Level 1: Adhoc Usage of New Technology
- Level 2: Manages Usage of New Technology
- Level 3: Defined usage of new technology
- Level 4: Quantitatively managed usage of new technology
- Level 5: Optimized use of technology

Testing Adequacy of Control in Few Technology Usage

- Testing Actual performance achieved as against stated performance of the technology by manufacturer of technology.
- Tester must ensure the following:
 - Documentations given by technology vendor along with new represent actual technology execution.
 - Training courses about new technology must be effective and complete, so that transfer of needed knowledge to use the technology is effective.
 - New technology is compatible with existing technology, so that customer may retain old applications along with new one. Compatibility issues, if any, must be addressed
 - Expected test processes and tools are effective in testing new technology.
- **Test the Adequacy of Current Process Definition Available to Control Technology:** Access Adequacy of staff skills to effectively use technology.

Object-Oriented Application Testing

- Object oriented development has made a dramatic change in development methodologies. One object may be used at several instances giving a benefit of optimization, reusability and flexibility.
- There are many effective approaches to testing object-oriented software. A test process that complements object-oriented design and programming can significantly increase reuse, quality and productivity of development and testing process.

Testing of Internal Controls

- It includes the Testing of Transaction Processing Control, Transaction Origination, Transaction Entry in System, Transaction Communication Within/Outside the system, Transaction processing by system, Storage and retrieval of Data from the Database, Transaction Output, Testing Security Controls, Attributes of Effective Security Controls, Simplicity of Control and /Usage, Failure Self Control, Open design for control, Separation on privileges of users, Psychological Acceptability of Controls by user, Layered defense in system and Compromised recording.

COST Testing

- COST stands for Commercially Off The Shelf. These softwares are readily available in the market and user can buy them directly.
- Why Software Organizations Use COST?
 - Line of Business
 - Cost-Benefit Analysis Expertise/Domain Knowledge Delivery Schedule
- **Features of COSTS Testing:** COSTS are developed with general requirements collected from the market. It may not exactly match with organization's needs and expectations. Some COSTS may need changing business process to suite the COSTS implementation in organization. This is another way of Business Process Reengineering (IBPR) for an organization where intentionally proven practices can be implemented by using COST. Sometimes, COSTS may need configuration of software or system to suite business model.
- **Challenges in Testing COSTS:** Some of the aspects which testers must remember while testing COSTS are as follow:
 - Requirement statement and design may not be available to testers as product manufacturer never share with any customer.
 - Verification and validation records prepared during SDLC are very important for system testing and acceptance testing.
- **COSTS Test Process:**
 - Assure completeness of need specification.
 - Define critical success factor of buying.
 - Determine compatibility with environment variables.
 - Assure that COSTS can be integrated with business.
 - Demonstrating COSTS in operations.

- Evaluate people fit.

Client-Server Testing

- Client-server is an initial improvement from stand-alone applications where there are several clients communicating with the server.
- There are many advantages of client-server over stand-alone application. The architecture gives an opportunity to work with software at a time.
- **Testing Approach of Client-Server System:** Involves component testing and integration testing followed by various specialized testing, as per scope of testing involved. These testing are:
 1. Component Testing
 2. Integration Testing
 3. Performance Testing
 4. Concurrency Testing
 5. Disaster Recovery/Business Continuity Testing
 6. Testing For Extended Periods

Web Application Testing

- It is further improvement in client-server applications where the clients can communicate with servers through virtual connectivity. It has many advantages over a client-server application a multiple server networks can be accessed at a time from the same client. It improves communication between people at different places significantly.

Testing Approach of Client-Server System: Involves component testing and integration testing followed by various specialized testing, as per scope of testing involved. These testing are:

1. Component Testing
2. Integration Testing
3. Performance Testing
4. Concurrency Testing
5. Disaster Recovery/Business Continuity Testing
6. Testing For Extended Periods
7. Compatibility Testing

Mobile Application Testing (PDA Devices)

- Now-a-days pocket devices are widely used for communication and computing.
- Interfaces Design of PDAs:
 - The smaller size and resolution of the PDA screen presents usability challenges to testers. Reading from the screen is not easy.
 - Scrolling up and down is very inconvenient.
 - Instructions and other text must be used carefully only when necessary. As they may occupy the screen and scrolling may become necessary.
 - Links must be very brief and containing only necessary keywords.

eBusiness/eCommerce Testing

- There are minor difference between ecommerce and ebusiness application. Ecommerce mainly deals with money transactions, ebusiness concerns with all aspects of business including money, advertising and sales.
- Distance part of eBusiness :
 - Information Access
 - Self Service
 - Shopping Service
 - Interpersonal Communication Services
 - eBusiness
- **Testing Approach for eBusiness/eCommerce :**
 - Software applications are challenged to meet expectations of usability, performance and reliability which are critical success factors for such applications. People may not use the application or perform any transactions if it is not user friendly.
 - Generally, these applications are controlled by regulatory and statutory requirements imposed by various governments at different places and at different times.

Agile Development Testing

- Development is becoming a famous word in software development. Agile development talks about agile policy which works of some principles.
- Some of agile principles are given below:
 - Delivering working software to users in place of getting requirements signed off from customer.
 - Adopting, and acceptance changes in requirements in place of scope definitions and change management to deliver what is required for the project.
 - More stress on communicating effectively between various stakeholders.
- **Agile Testing:** There are various ways under the umbrella of agile development such as scrum, extreme programming, features-driven development, and test-driven development etc.
- **Critical Points For Agile Testing:** Testing is very critical from agile perspective. There are many changes and test team must be capable of handling huge regression testing cycle as one progresses from iteration to iteration.
- **Competencies/Maturity of Agile Development and Test Team:**
 - For undertaking agile, one must have the teams, customer and management who psychologically accept agile approach. It talks about ability to change very fast, build good working product and communicate with team members and stakeholders effectively as well as efficiently.
- **Development and Test Process Variability**
 - One must be able to plot development and test process and try to remove personal factors from the processes.
- **Change Management and Communication**
 - Change is predictable in agile. It flows from customer to development team and goes back to customer. There must be a very close communication between development team, test team, customer, and other stakeholders to adapt to changing scenario.

- **Testing Process Flexibility**
 - Changes is must in agile, and one may have to adapt to the changes. Test process is not an exception to it. Different parts of software need different strategies of testing. There must be different test plans or one may keep flexibility in a test plan to adapt to these changes.
- **Focus on Business Objectives:**
 - There is always a time pressure in agile development. Pressure may come from stakeholders to deliver things faster or it may come from development, if they get delayed. Cost-benefit analysis may be done when it comes to defect fixes and release of software

Data Warehousing Testing

Repository of an organization's electronically stored data. Data warehousing focuses on data storage, to retrieve and analysis data, to extract, transform and load data and to manage the data dictionary. ETL testing is performed in five stages:

- Identifying data sources and requirements.
- Data acquisition.
- Implement business logics and dimensional modelling .
- Build and populate data.
- Build Reports.