Software Testing

Unit-1

1.1 What is Software Quality?

- The quality of software can be defined as the ability of the software to function as per user requirement.
- When it comes to software products it must satisfy all the functionalities

Key aspects that conclude software quality include,

•**Good design**–It's always important to have a good and aesthetic design to please users

•**Reliability**–Be it any software it should be able to perform the functionality impeccably without issues

•**Durability-**Durability is a confusing term, In this context, durability means the ability of the software to work without any issue for a long period of time.

•**Consistency**–Software should be able to perform consistently over platform and devices

•**Maintainability**–Bugs associated with any software should be able to capture and fix quickly and news tasks and enhancement must be added without any trouble

•**Value for money –**customer and companies who make this app should feel that the money spent on this app has not foneto waste.

1.2 What is Software Quality Model?

- Software quality models were proposed to measure the quality of any software model.

- There are five widely accepted modelswhen it comes to measuring software quality

1.McCall's Quality Model

2.Boehm quality model

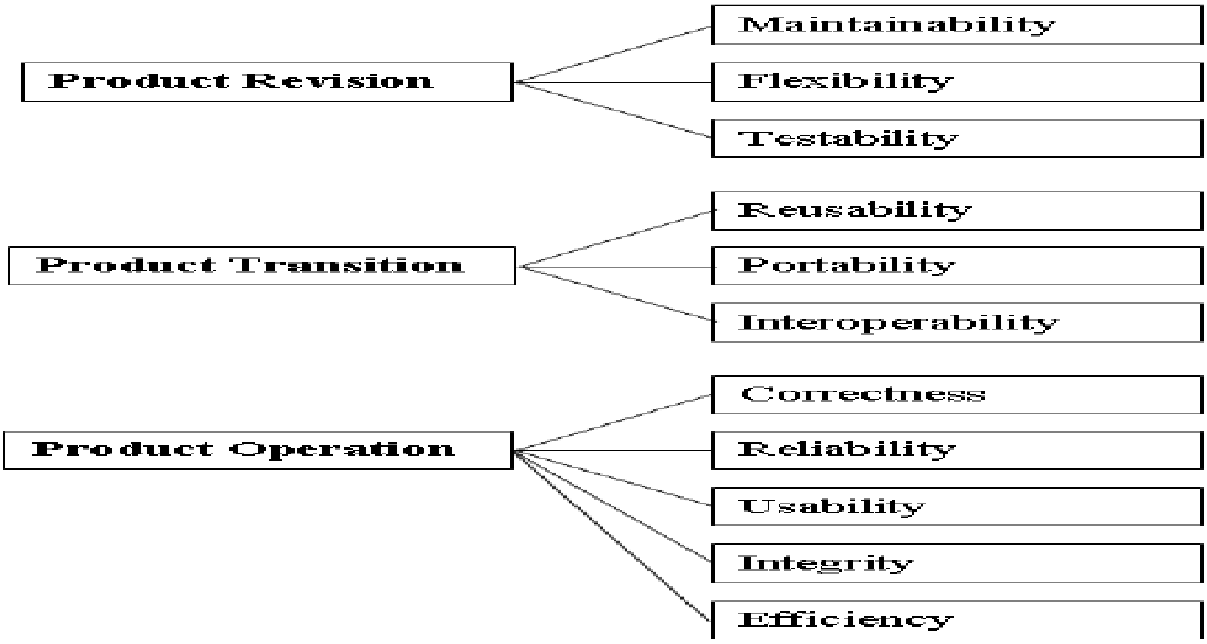
3.Dromey'squality model

4.FURPS model

5.ISO 9126MODEL

1. Mc call's Model

- Mc Call's model was first introduced in the US Airforcein the year 1977.
- The main intention of this model was to maintain harmony between users and developers.



2. Boehm Model

- Boehm software quality model was introduced in the year of 1978.
- The model is used to represent a hierarchical model that structures around high level characteristics, intermediate level characteristics, and primitive characteristics.
- The high level of characteristics is made in such a way that answers following questions:

As-Is Utility:: It defines the way a utility signifies the as-is utility. It creates a question of how easily, reliably and efficiently an as can be utilized.

Maintainability:: This aspect decides how convenient it is to understand, change or re-evaluate a process.

Portability: This aspect helps in deciding an effective way to change an environment.

- The intermediate level characteristic represents Boehm's 7 quality factors that together represent the qualities expected from a software system:

  • Portability(General utility characteristics): Code possesses the characteristic portability to the extent that it can be operated easily and well on computer configurations other than its current one.

  • Reliability(As-is utility characteristics): Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily.

  • Efficiency(As-is utility characteristics): Code possesses the characteristic efficiency to the extent that it fulfillsits purpose without waste of resources.

  • Usability(As-is utility characteristics, Human Engineering): Code possesses the characteristic usability to the extent that it is reliable, efficient and human-engineered.

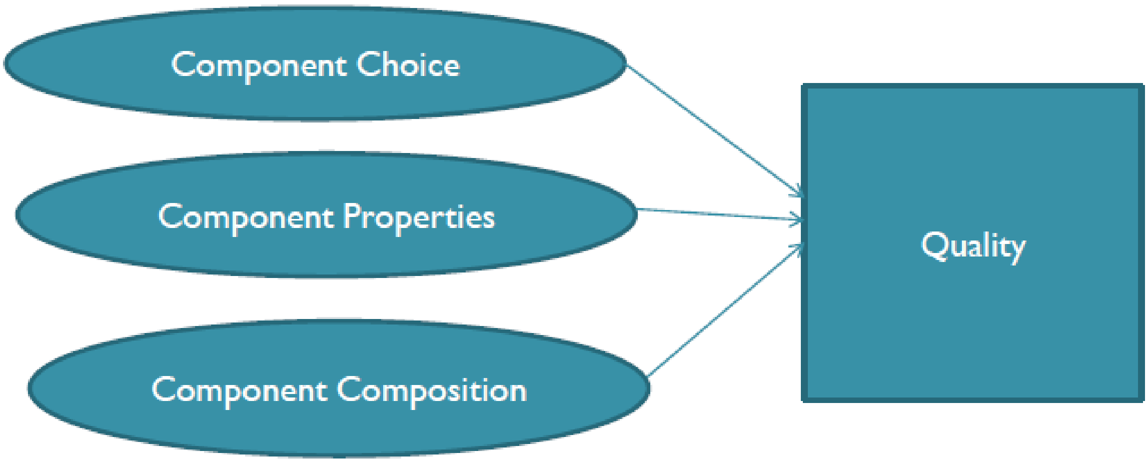
  • Testability(Maintainability characteristics): Code possesses the characteristic testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance.

  • Understandability(Maintainability characteristics): Code possesses the characteristic understandabilityto the extent that its purpose is clear to the inspector.

  • Flexibility(Maintainability characteristics, Modifiability): Code possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined.

3. Dromey'squality model

- Dromey'sproposed in 1996,a quality evaluation framework that analyses the quality of software based on the components.
- According to this model, product quality is determined by the fig below:



- Choice of components that make up the product and how they are implemented, Tangible properties of individual components and Tangible properties associated with components composition.
- The possible **violations that affect product quality** are:
- Using the wrong component in given context,
- Improperly implementation a components and
- Misusing a components in relation to other components.

- The quality framework is based on 3 quality models:

- Implementation quality model

- Requirements quality model

- Design quality model

- Dromey'squality model involves 5 steps:

1.Choose a set of high level quality attributes like functionality,reliability,accuracyand understandability.

2.Identify product components.

3.Identify quality–carringproperties for components/modules.

4.Determine how each property affects the quality attributes.

5.Evaluate the model and identify its weaknesses.

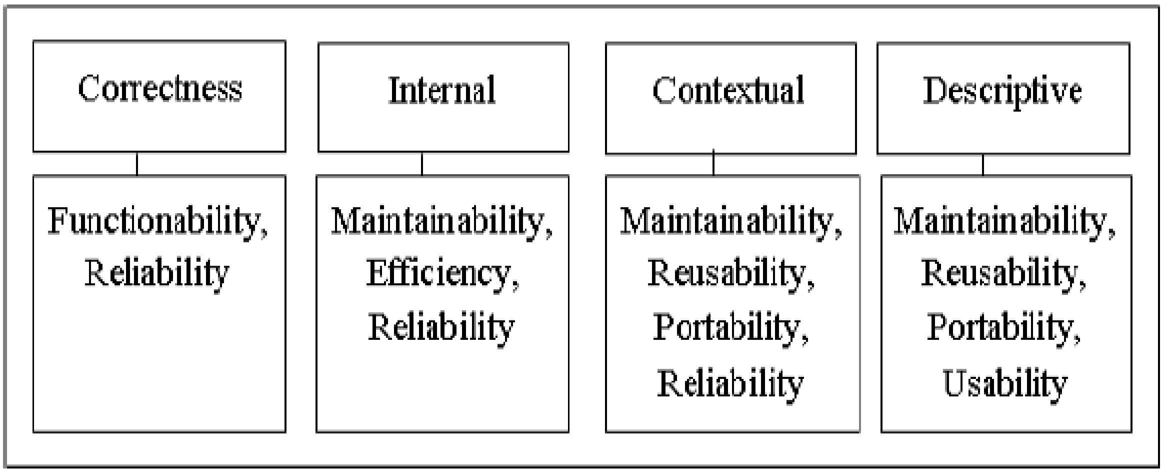
- According to dromey, these components have intrinsic properties that can be classified into 4 Categories:

1.Correctness: Evaluates if some basic principle are violated.

2.Internal: Measures how well components has been depolyed

3.Contextual: Deals with the external influences by and on the use of components.

4.Descriptive: Measure the descriptiveness of a component



Correctness	Internal	Contextual	Descriptive
Functionability, Reliability	Maintainability, Efficiency, Reliability	Maintainability, Reusability, Portability, Reliability	Maintainability, Reusability, Portability, Usability

4. FURPS Model

- Robert Grady and Hewlett Packard proposed the FURPS model that uses characteristics from functional and non-functional requirement.
- FURPS stands for
  - **F**unctionality-Features set, Capabilities, Generality,Security , **U**sability-Human Factors, Aesthetics, Consistency, Documentation, Responsiveness
- **R**eliability-Availability (Failure Frequency (Robustness/Durability), Failure Extent & Time-Length (Recoverability/Survivability)), Predictability (Stability), Accuracy (Frequency/Severity of Error)
- **P**erformance-Speed, Efficiency, Resource Consumption (power, ram, cache, etc.), Throughput, response time.
- **S**upportability-Testability, Flexibility, Adaptability, Extensibility, Maintainability,compatibility.

5. ISO 9126 Model

- ISO 9126 is an international standard for evaluation of software first issued in 1991 and reissues in 2001.

- This model is based on MCCall, Boehm, FURPS

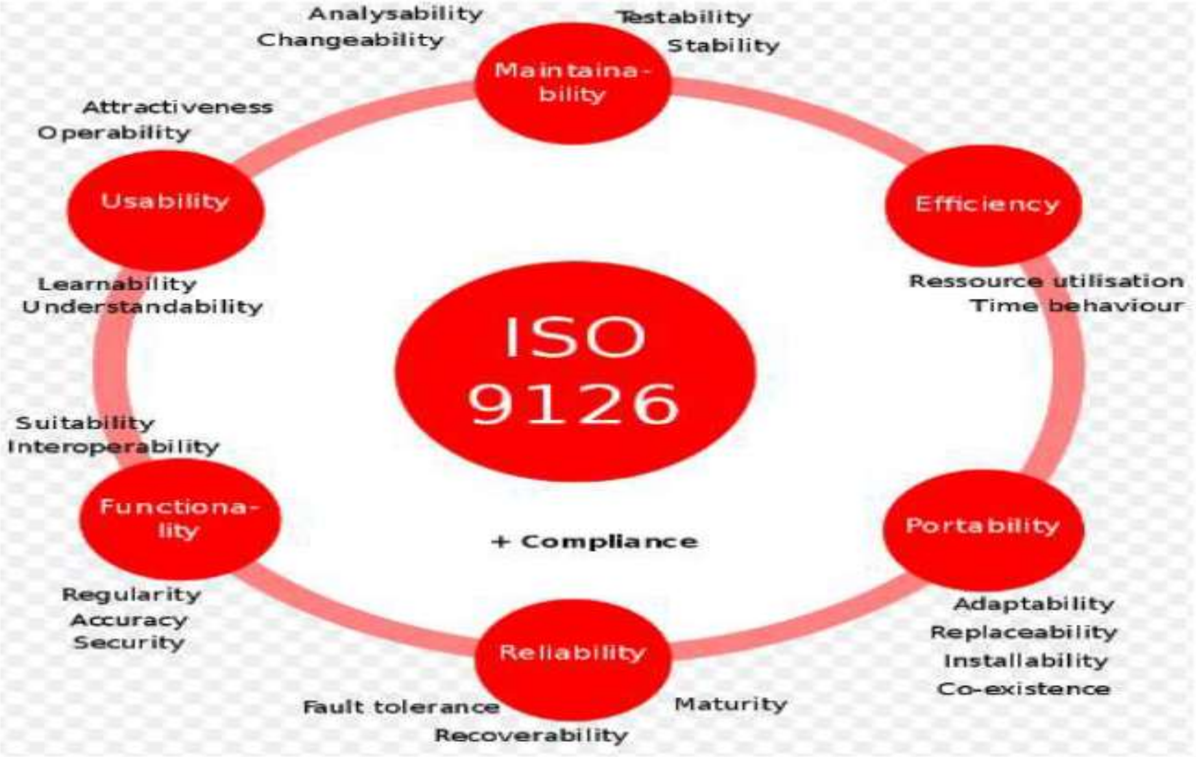
- ISO/IEC Standard 9126:2001 divides software quality into

1.Process Quality –express the degree to which defined process are followed and completed.

2.Product Quality –to which the developed software meets the defined requirement.

3.Quality in use –to which a product is fit for purpose when exposed to particular context of use.

- As per ISO 9126,shown in fig.



- The most important parameter of software product are:

- **Functionality**-"A set of attributes that bear on the existence of a set of functions and their specified properties.

- **Reliability-**"A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time."

- **Usability-**"A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users."

- **Efficiency**-"A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions."

- **Maintainability-**"A set of attributes that bear on the effort needed to make specified modifications."

- **Portability**-"A set of attributes that bear on the ability of software to be transferred from one environment to another."
- Benefits of software quality

- ➢ Delivering quality software products and services benefits in the following ways:
- ➢ Higher quality and better services allows an organization to **change more** from their customers.

- ➢ Correct job done right the first time as per customer expectations ensures quality and saves cost of rework, thereby **increasing profitability.**

- ➢ Good quality product **increases word-of-mouth marketing**and referral business, thereby decreasing external advertising costs.

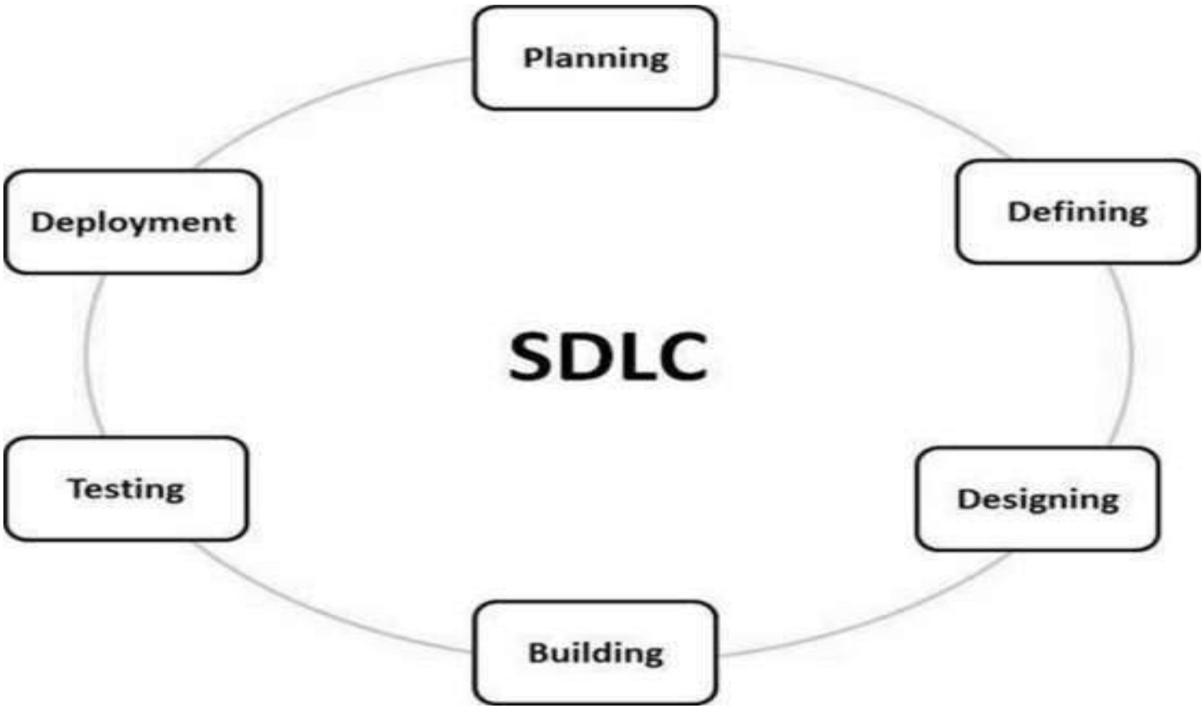
1.3 Software development life cycle model

- ➢ SDLC is a process followed for a software project, within a software organization.

- It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software.

- The life cycle defines a methodology for improving the quality of software and the overall development process.

- The following figure is a graphical representation of the various stages of a typical SDLC



- There are various approach in software development namely,

  1.Waterfall Model

  2.Prototyping Model

  3.Incremental Model

  4.Iterative Model

  5.SprialModel

  6.Agile Methodology

- Each process model follows a particular workflow in order to successfully develop the software.

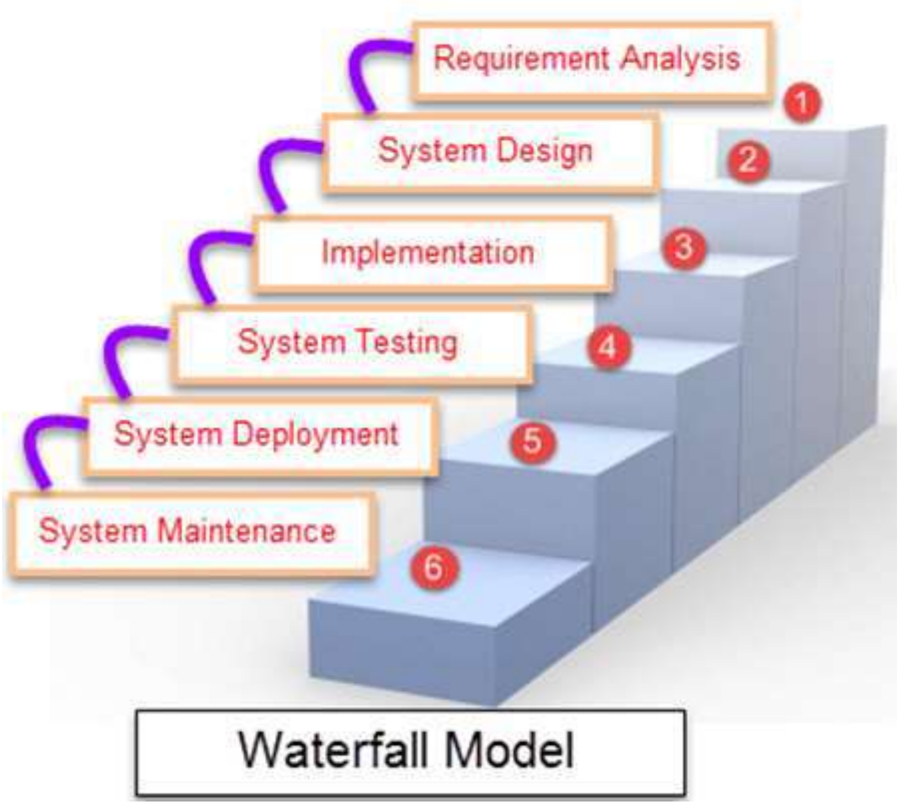
WATERFALL MODEL

- WATERFALL MODELis a sequential model that divides software development into pre-defined phases.

- Each phase must be completed before the next phase can begin with no overlap between the phases.

- Each phase is designed for performing specific activity during the SDLC phase.

- It was introduced in 1970 by Winston Royce.



Different Phases of Waterfall Model in Software Engineering

- **Different phases Activities performed in each stage :**

1.Requirement Gathering stage : During this phase, detailed requirements of the software system to be developed are gathered from client.

2.Design Stage : Plan the programming language, for ExampleJava,PHP, .netor database like Oracle, MySQL, etc. Or other high-level technical details of the project.

3.Built Stage: After design stage, it is built stage, that is nothing but coding the software.

4.Test Stage : In this phase, you test the software to verify that it is built as per the specifications given by the client.

5.Deployment stage : Deploy the application in the respective environment.

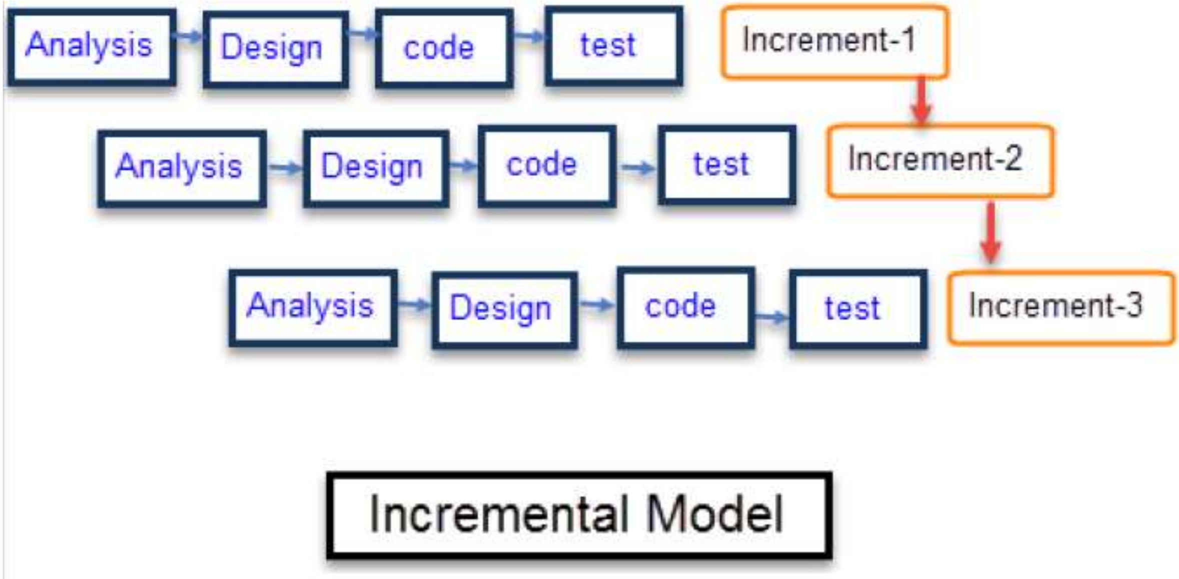
6.Maintenance stage : Once your system is ready to use, you may later require change the code as per customer request.

- **When to use SDLC Waterfall Model**

- Waterfall model can be used when
- Requirements are not changing frequently
- Application is not complicated and big
- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable
- Resources are available and trained

INCREMENTAL MODEL

- Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle.
- Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.
- Each iteration passes through the**requirements, design, coding and testing phases**.
- And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.



- The system is put into production when the first increment is delivered.
- The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments.
- Once the core product is analyzed by the client, there is plan development for the next increment.

When to use Incremental models?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental models

- Errors are easy to be recognized.
- Easier to test and debug

- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Advantage of Incremental models

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning and design.
- Total Cost is high.
- Well defined module interfaces are needed.

Prototyping Model

- ThePrototypingModelisoneofthemostpopularlyusedSoftwareDevelopmentLifeCycleModels(SDLCmodels).

- Thismodelisusedwhenthecustomersdonotknowtheexactprojectrequirementsbeforehand.

- Inthismodel,aprototypeoftheendproductisfirstdeveloped,testedandrefinedaspercustomerfeedbackrepeatedlytillafinalacceptableprototypeisachievedwhichformsthebasisfordevelopingthefinalproduct.

- Theprocessstartsbyinterviewingthecustomersanddevelopingtheincomplete high-levelpapermodel.
- Oncethecustomerfiguresouttheproblems,theprototypeisfurtherrefinedtoeliminatethem.Theprocesscontinuestilltheuserapprovestheprototypeandfinds theworkingmodeltobesatisfactory.

- There are 2 approaches for this model:

  1.Rapid Throwaway Prototyping –This technique offers a useful method of exploring ideas and getting customer feedback for each of them.

  2.Evolutionary Prototyping –In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort.

Iterative Model

- An**iterative life cycle model**does not start with a full specification of requirements.

- In this model, the development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements.

- Each release of Iterative Model is developed in a specific and fixed time period, which is called iteration.

- Iterative Model allows accessing previous phases, in which the changes are made accordingly. The final output of the product is revived at the end of the**Software Development Life Cycle (SDLC)**.

Advantages of Iterative Model:

- The biggest advantage of this model is that, it is implemented during the earlier stages of software development process,which allows developers and testers to find functional or design related flaws as early as possible,which further allows them to take corrective measures in a limited budget.

Spiral Model

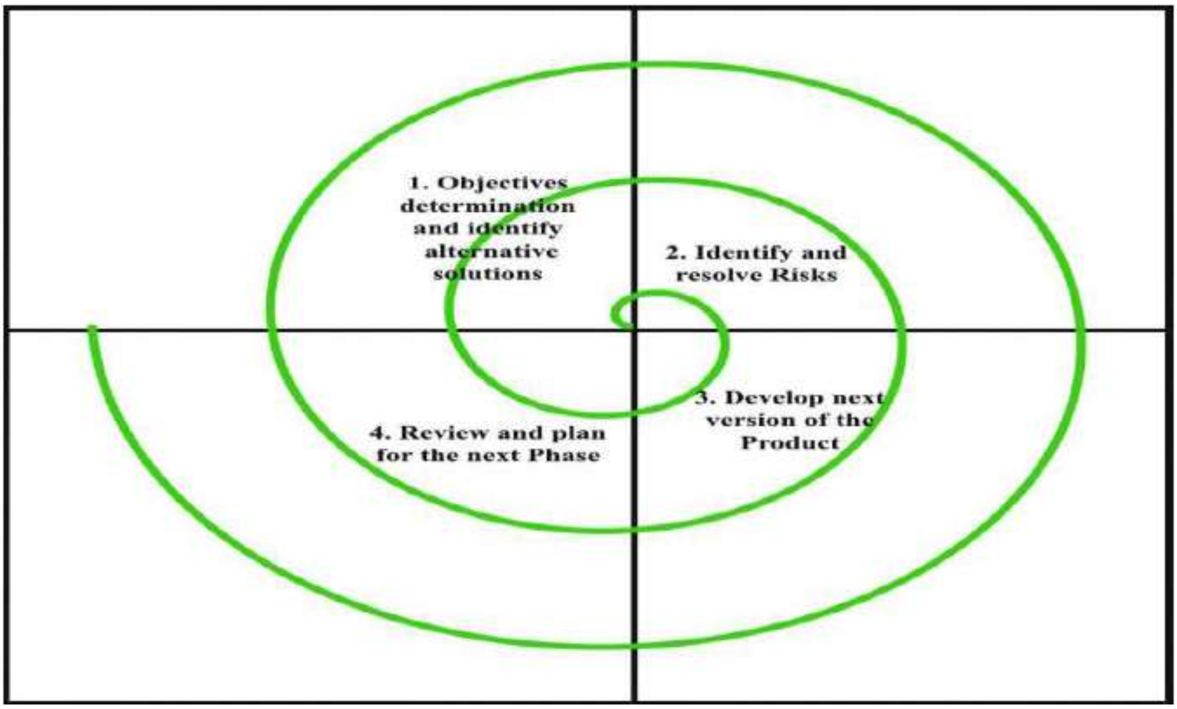
- **Spiral model**is one of the most important Software Development Life Cycle models, which provides support for**Risk Handling**.

- The exact number of loops of the spiral is unknown and can vary from project to project.

- **Each loop of the spiral is called a Phase of the software development process.**

- The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.
- The Radius of the spiral at any point represents the expenses (cost) of the projects and the angular dimension represents the progress mades of the current phase.

- Below diagram shows the different phases of the Spiral Model:


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

- Each phase of Spiral Model is divided into four quadrants as shown in the above figure.

1.Objectives determination and identify alternative solutions: Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase.

2.Identify and resolve Risks: During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified.

3.Develop next version of the Product: During the third quadrant, the identified features are developed and verified through testing.

4.Review and plan for the next Phase: In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Agile Methodology

- Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

- Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation,
- A leadership philosophy that encourages teamwork,self-organization and accountability,
- A set of engineering best practices intended to allow for rapid delivery of high-quality software, and
- A business approach that aligns development with customer needs and company goals.
- Agile development refers to any development process that is aligned with the concepts of the Agile Manifesto.
- The12 agile principles stated in the agilemanifesto forms the basic of this software development methodology.
- The guiding principles behind agile software development are:

1.Our highest priority is to satisfy the customerthrough early and continuous delivery of valuable software.

2.Welcome changing requirements, even late indevelopment. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from acouple of weeks to a couple of months, with reference to the shorter timescale.

4.Business people and developers must work together daily throughout the project.

5.Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6.The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8.Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9.Continuous attention to technical excellence and good design enhances agility.

10.Simplicity--the art of maximizing the amount of work not done--is essential.

11.The best architectures, requirements, and design merge from self-organizing teams.

12.At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

1.4 Types of Defects

- Defects can be introduced in the software in any phases as requirements development, design, coding and testing.

- Defect type may help inidentifying the phase in which it might have been introduced by a method called Orthogonal Defect Classification.

- Defects are injected or introduced in the software by members of the development team due to inadequate training, lackofdesign and development skills, poo r communication among team members etc.

- Defects can be assigned to four major classes in the software life cycle:

1.Requirements/specifications defects
2.Design defects
3.Coding defects and
4.Testing defects

1. Requirements/specifications defects :

- A valid requirement which is mandatory and supposed to code (or implement) is missed.

- The root cause of this defect is due to not capturing this requirement in the specification document.
- These types of observations are categorized as '**Missed requirements defects**' since the requirements are missed from requirement specification document.

2. Design defects

- This type of defects occurs when interactions between system components, interactions between the components and external software/hardware or interactions with user are incorrectly designed.
- Defects in the design of algorithms, control, logic, data elements and external software/hardware/user interface descriptions are all due to design defects.

3. Coding defects

- Coding defects are due to faulty implementation of the code.
- Code defects if code is used for low level design which is provided to programmers to code.
- Coding defects happen due to lack of programming skills and miscommunication with the designers.
- Omission of programming constructs and incomplete implementation of data structures also lead to defects.

4. Testing defects

- Test plans, test cases, test harnesses and test procedures can also contain defects,

- Test cases are supposed to detect all defects, incomplete or poorly design test cases fail to detect defects and some defects escape to the customer.

1.5 Definitions used for software quality engineering

- The following definitions used for software quality engineering:

1. Incident: The questionable behavior in some situations is called an incident.

- It may or may not be result of a defect.

2. **Error:** Human mistakes cause error.

- Errors can be typographical and syntactic errors which can be detected easily during complication.
- But logical errors in design and coding are difficult to diagnose without proper verification methods.
- This causes need to be analyzed and appropriate corrective action taken

3. **Defects:** These are improper program conditions generally the result of an error.

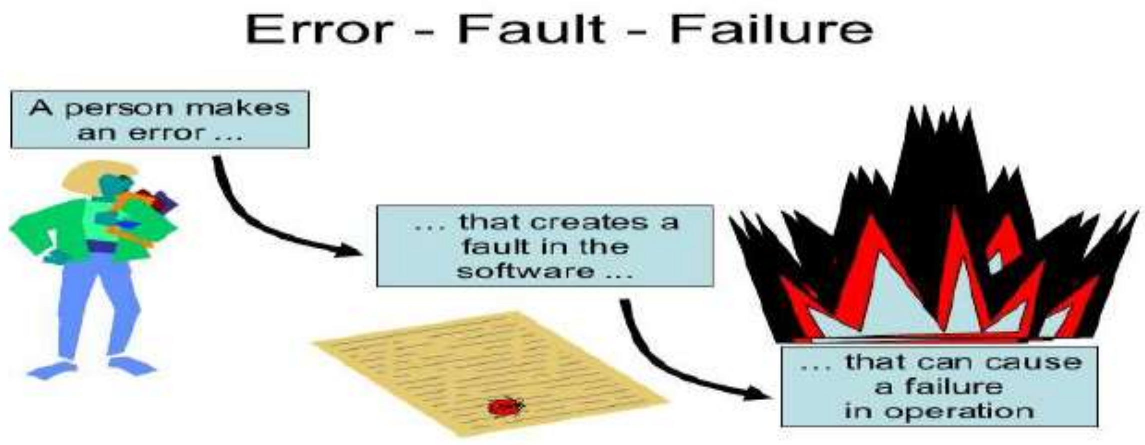
- Not all errors produce defects,
- e.g incorrect comments, errors in documentation and others.

4.**Bug :** Developer accepts the defect is called bug.

- It a program defects that is encountered during operation (either during test or during use).
- Bugs result from defects but all defects do not cause bugs.

5. **Failure :** When a defect reaches the end customer is called failure.

- It is malfunction of a user's installation. It may result from a bug, incorrect installation, hardware failure etc.
- The relationship between error, defect and failure is shown below fig.


Error - Fault - Failure
A person makes an error ...
... that creates a fault in the software ...
... that can cause a failure in operation

6. **Verification :**

- It is the process of checking the correctness of the product developed based on the understanding of the customer requirements by testing.

- Verification may be done by static analysis and dynamic analysis by executing the program in a test.

- Verification can be expressed by the query ‘Are you building the thing right?’ check the documented development processes were followed and output is per specifications documented.

7. **Validation :**

- It is an attempt to find errors by executing the program in real environment.
- It checks the products from users perspective to ensure that the product as per user expectations.

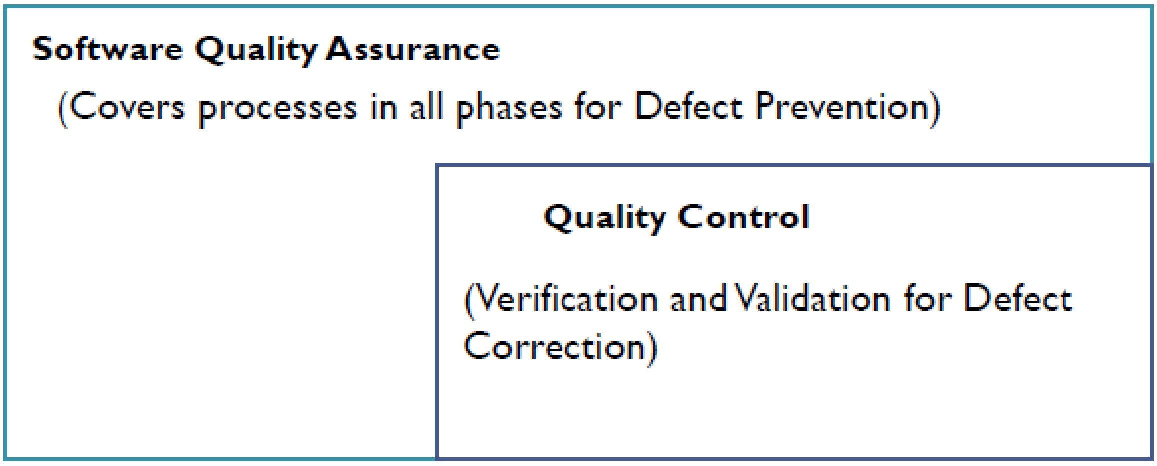
- Validation can be expressed by the query ‘Are you building the right thing?’ refer to the users needs

8. **Debugging :**

- It involves finding the precise nature of a defects and then correcting it.

1.6 Software Quality Assurance And Quality Control

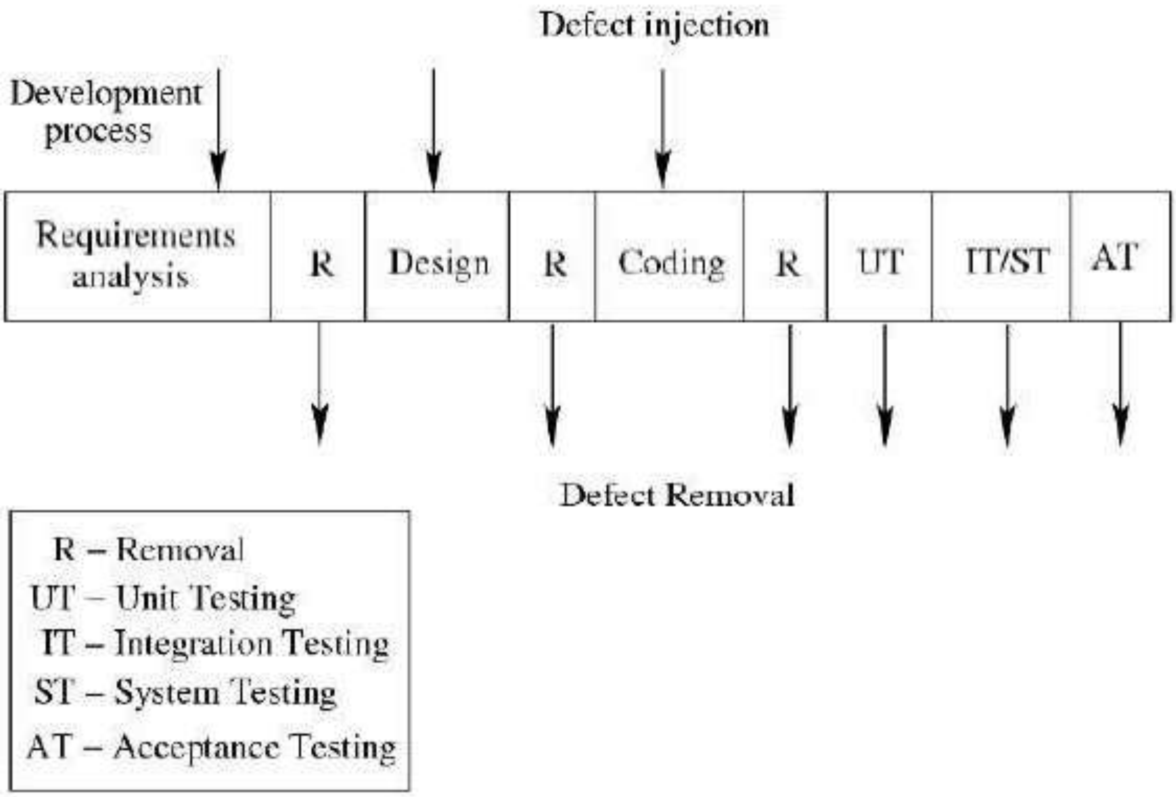
- The software quality needs intelligence efforts in both aspects of Software Quality Assurance (SQA) And Quality Control (QA) both which comprise software quality engineering as in below fig.


Software Quality Assurance
(Covers processes in all phases for Defect Prevention)
Quality Control
(Verification and Validation for Defect Correction)

Software Quality Assurance(SQA) is a set of activities for ensuring quality in software engineering processes.

- It ensures that developed software meets and complies with the defined or standardized quality specifications.
- SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets the desired quality measures.
- SQA incorporates and implements software testing methodologies to test the software.
- Rather than checking for quality after completion, SQA processes test for quality in each phase of development, until the software is complete.
- With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards.
- SQA also encompasses processes for quality control as shown in figure.

- Total Quality Management(TQM),the quality of the product depends upon the quality of processes defined and followed in its development.
- SQA activities involves following the process standards defined by International Organization for Standardization (ISO),Software Engineering Institute (SEI) and others.
- QC deals with a set of activities designed to evaluate a developed work product and correct any defects detected.
- QC activities focus on finding defects in deliverables including that in documents and in code.
- Reviews, inspections, walkthroughs and testing are part of QC activities.
- The QC activities in a SDLC as shown below:



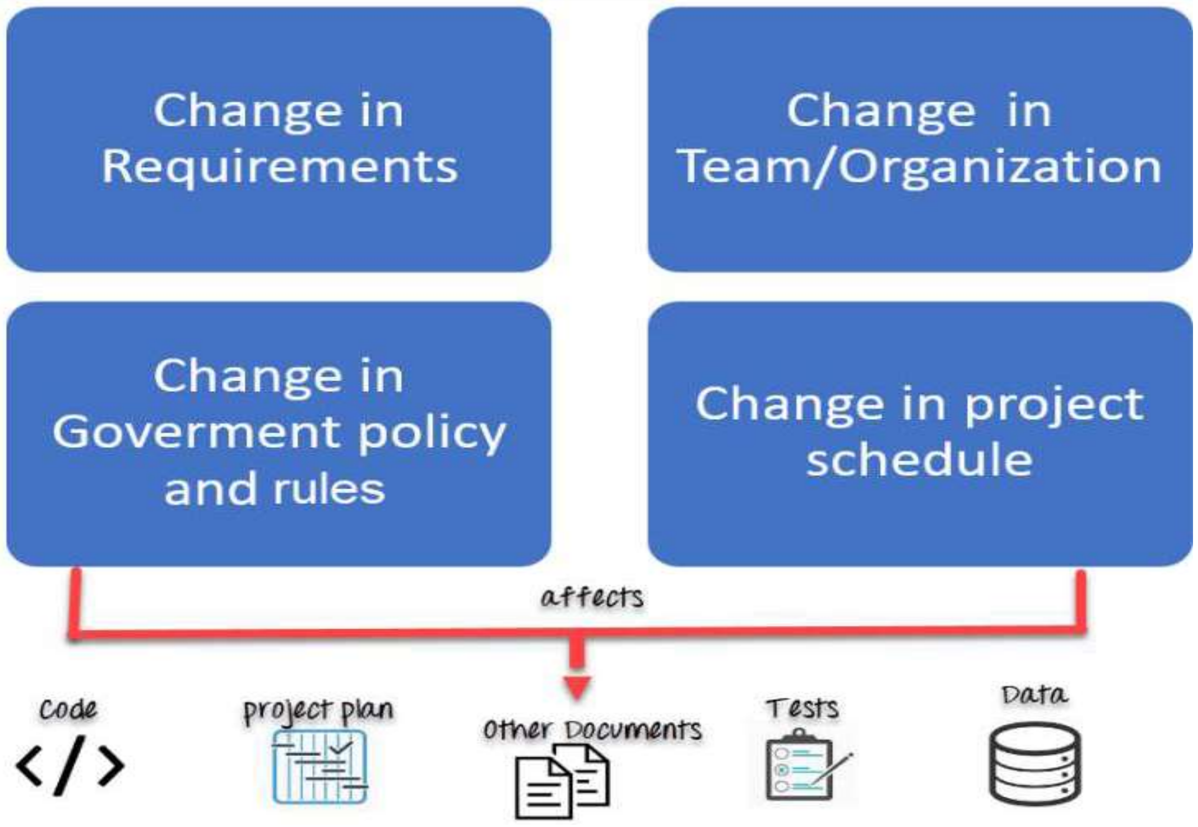
- Defects may be introduced/injected in requirements development, design and coding stages.

  •These defects are removed by reviews/inspection and by testing.

  •The process of testing is a dynamic vertification techniques which can only be done by running the code.

1.7 Software Configuration Management

- In Software Engineering,**Software Configuration Management(SCM)**is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.
- The primary goal is to increase productivity with minimal mistakes.
- SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.
- The primary reasons for Implementing Technical Software Configuration Management System are:
  - There are multiple people working on software which is continually updating
  - It may be a case where multiple version, branches, authors are involved in a software configproject, and the team is geographically distributed and works concurrently
  - Changes in user requirement, policy, budget, schedule need to be accommodated.
  - Software should able to run on various machines and Operating Systems
  - Helps to develop coordination among stakeholders
  - SCM process is also beneficial to control the costs involved in making changes to a system


Change in Requirements
Change in Team/Organization
Change in Goverment policy and rules
Change in project schedule
affects
Code
project plan
Other Documents
Tests
Data

- Any change in the software configuration Items will affect the final product.
- Therefore, changes to configuration items need to be controlled and managed.

1.8 Software Quality Assurance

- **Software Quality Assurance (SQA)**is simply a way to assure quality in the software.
- It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly.
- Software Quality Assurance is a process which works parallel to development of a software.
- It focuses on improving the process of development of software so that problems can be prevented before they become a major issue.
- Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.

Software Quality Assurance have:

- A quality management approach
- Formal technical reviews
- Multi testing strategy
- Effective software engineering technology
- Measurement and reporting mechanism

Benefits of Software Quality Assurance (SQA):

- SQA produce high quality software.
- High quality application saves time and cost.
- SQA is beneficial for better reliability.
- SQA is beneficial in the condition of no maintenance for long time.
- High quality commercial software increase market share of company.
- Improving the process of creating software.
- Improves the quality of the software.

1.9 Role of SQA

- While SQA team facilitates implementation of processes and monitors if the prescribed standards and procedures are followed by the development team, ensuring quality of deliverables is the responsibility of the project team.

- Besides having procedures and guidelines to ensure quality, the top management of an organisation needs to be committed to delivering high quality products and services as per customer requirements.

- Quality management is a separate and important role and functioning of SQA team must be given due important in the organisation.

Functions of SQA

- The SQA team must work independently and assists the management in improving quality.

- It must be staffed with components professionals.

- The function of SQA team are to:

- Define the processes in the organization as per widely used quality frameworks.

- Monitor if all concerned members of a project team are adhering to the defined practices.

- Plan reviews after requirements development, design, coding and test design.

- Ensure that reviews are done at all planned stages and evaluated coding practices.
- Check the documentation done by the development team on the design and coding
- Conduct internal audits on the defined processes and inform the top management of the results
- Measure the effectiveness of the processes defined.
- Redefine the processes, if required to make more effective.

SQA People

- Getting right people into SQA is one of the challenge filled by the management of an organisation.

- The job of SQA team members does not involve being too knowledgeable about software technology and programming, they need familiar with the software development and engineering methologies.

- When an organization is going for process certification, the organization is benefitted if the member of its SQA team have understanding ISO 9001:2008 and other client specific process frameworks in addition to basic principles of software quality.

SQA Plan

- The SQA plan is a document that specifies the process to be followed in each step of the software development and the procedures to be followed in each activity of such a process.

- The objective of SQA plan is to ensure that the development of the software is based on a course of action and that from time to time the development can be measured controlled and monitored with respect to such a course of action-so that the end product is as per the specifications.

- The plan is governed by several quality standards, policies and models.

- **The SQA plan document consists of the below sections:**
  - Purpose section
  - Reference section
  - Software configuration management section
  - Problem reporting and corrective action section
  - Tools, technologies and methodologies section
  - Code control section
  - Records: Collection, maintenance and retention section
  - Testing methodology

1.10 What is Process?

- One of the most important concepts in the area of sqais the concept of process.
- A process contains a set of practices performed to achieve a given objective.
- Input to a process may be information, materials , data etc. ,and output can be information, data, product, service etc.,
- The output of a process depends not only on the defined practices but also on the interaction of hardware and software, procedures, specifications, skills, people and nature as shown in fig.

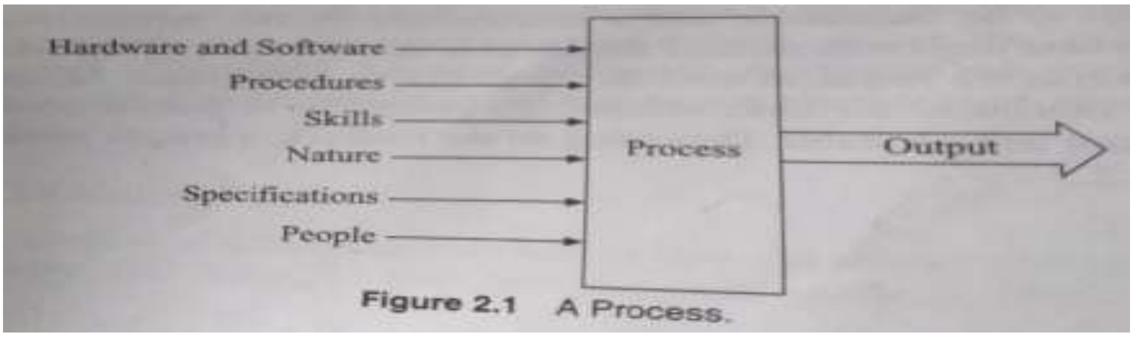
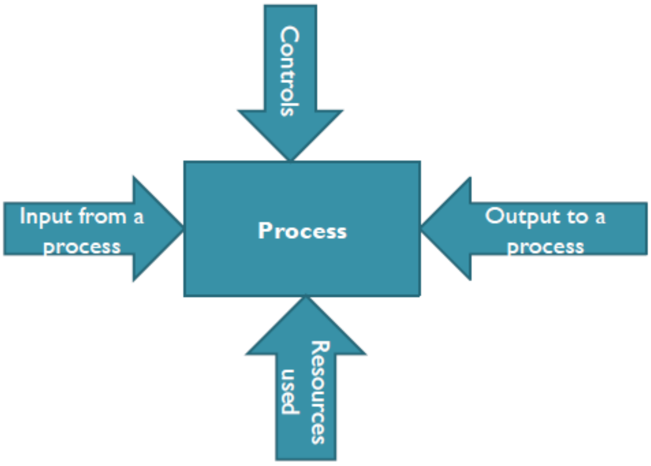


Figure 2.1 A Process.

- Each process has inputs, outputs of one process may be the inputs to the next.
- Execution of the process requires resources as shown below



Input / outputs of a process

1.11 Process Framework

- A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of size or complexity.
- It also includes a set of umbrella activities that are applicable across the entire software process.
- Each framework activity is populated by a set of software engineering actions –a collection of related tasks that produces a major software engineering work product (e.g. design is a software engineering action).
- Each action is populated with individual work tasks that accomplish some part of the work implied by the action.
- The following generic process framework is applicable to the vast majority of software projects :

1. Communication: This framework activity involves heavy communication and collaboration with the customer (and other stakeholders) and encompasses requirements gathering and other related activities.

2. Planning: This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced and a work schedule.

3. Modeling: This activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.

4. Construction: This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

5. Deployment: The software is delivered to the customer who evaluates the delivered product and provides feedback based on evaluation.

- Some most applicable framework activities are described below.

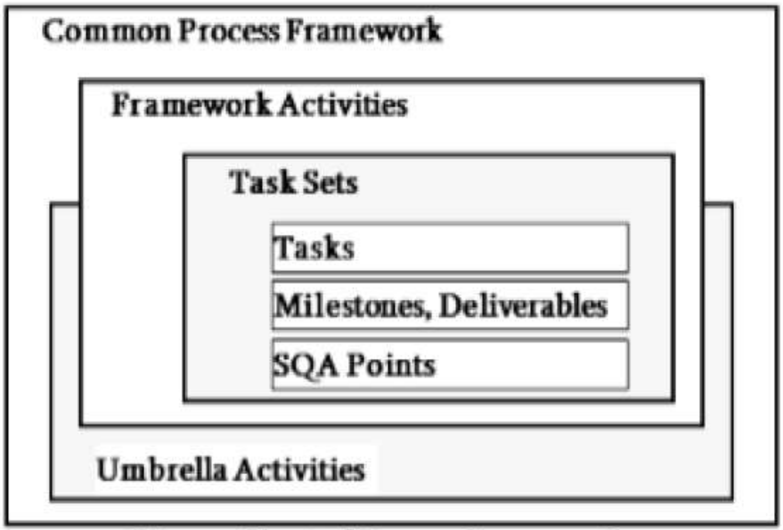


Figure: Chart of Process Framework

1.12 PDCA (Plan-Do-Check-Act) Cycle

- PDCA cycle became a widespread framework for constant improvements in manufacturing, management, and other areas.

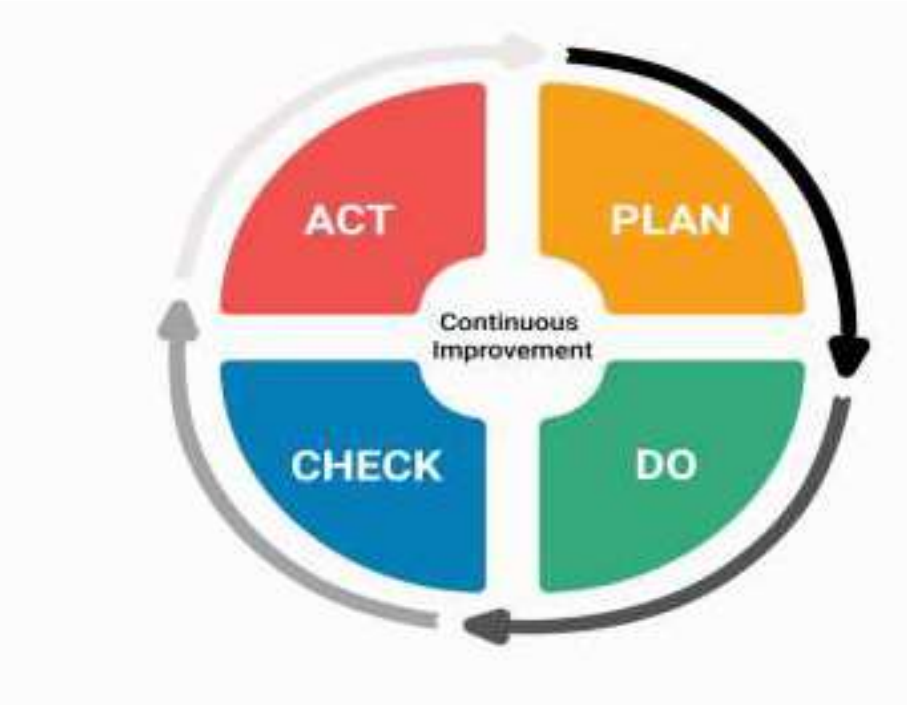
- PDCA is a simple four-stage method that enables teams to avoid recurring mistakes and improve processes.

- As we've explained the PDCA's meaning let’s get deeper into the topic and learn more about the cycle.

- PDCA cycle is an iterative process for continually improving products, people, and services. It became an integral part of what is known today asLean management.

- The Plan-Do-Check-Act model includes solutions testing, analyzing results and improving the process.

- For example, imagine that you have plenty of customer’s complaints about slow response rate of your support team. Then you will probably need to improve the way your team works in order to keep customers satisfied. That is the point where PDCA comes into play.



- A closer look at the four stages of the PDCA process.

Plan

- At this stage, you will literally plan what needs to be done. Depending on the size of the project, planning can take a major part of your team's efforts.
- It will usually consist of smaller steps, so you can build a proper plan with fewer possibilities of failure.

DO

- After you have agreed on the plan, it is time to take action. At this stage, you will apply everything that has been considered during the previous stage.
- Be aware that unpredicted problems may occur at this phase. You may first try to incorporate your plan on a small scale and in a controlled environment.

CHECK

- This is probably the most important stage of the PDCA cycle. If you want to clarify your plan, avoid recurring mistakes, and apply continuous improvement successfully, you need to pay enough attention to the CHECK phase.

ACT

- Finally, you arrive at the last stage of the Plan-Do-Check-Act cycle. Previously, you developed, applied, and checked your plan. Now, you need to act.

- If everything seems perfect and your team managed to achieve the original goals, then you can proceed and apply your initial plan.

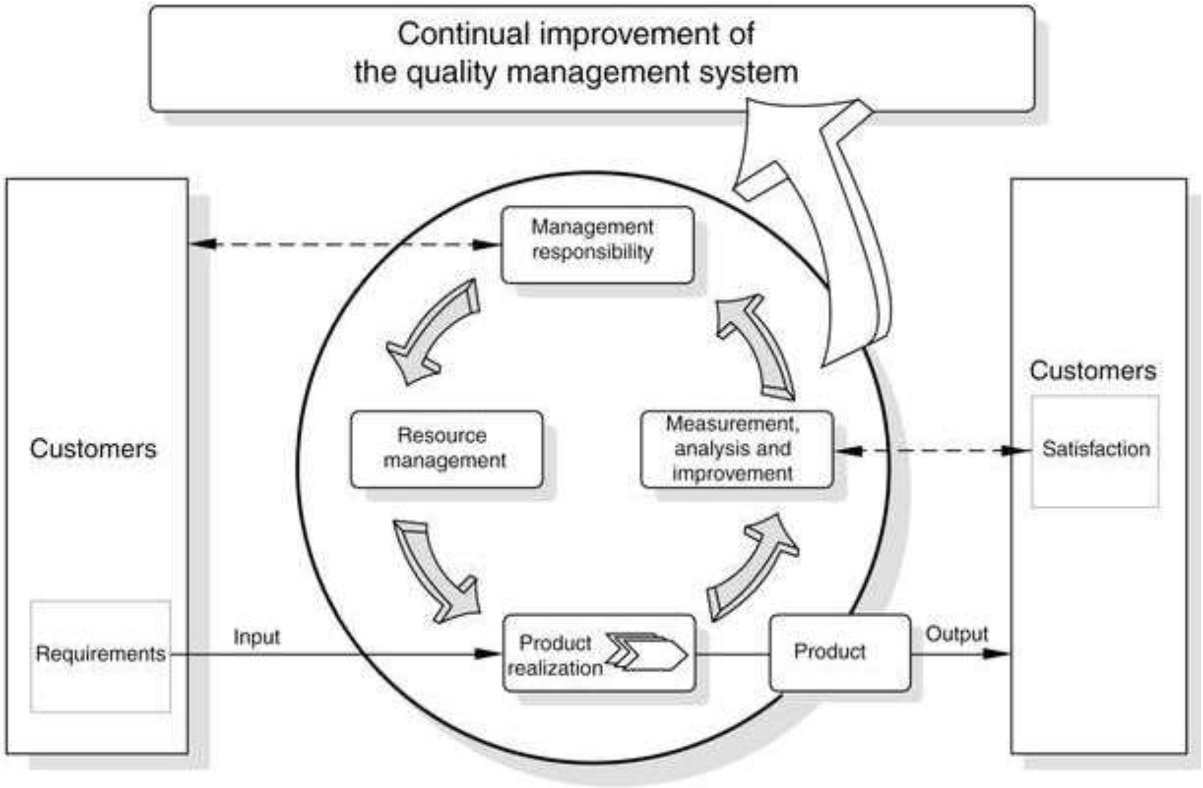
i. ISO 9001:2008

- International Organization for Standardization(ISO) introduced the latest version ISO 9001:2008 in the year 2008.

- ISO 9001:2008 sets out the criteria for a QMS(Quality Management System) and is the only standard in the ISO 9000 family that can be certificate to.

- It can be used by any organization, large or small, regardless of its field of business.

- It implemented by over one million companies and organization to ensure that products or services satisfy the customers quality requirement and with any regulations applicable in making these products or delivering the services.
- ISO 9001:2008 has five major sections and focuses on customer satisfaction, data analysis and continual improvement.
- For application to computer software, a set of guidelines has been provided in ISO 9001 by international organization standardization.
- The structure of ISO 9001:2008 is shown below


Continual improvement of the quality management system
Customers
Requirements
Input
Management responsibility
Resource management
Measurement, analysis and improvement
Product realization
Product
Output
Customers
Satisfaction

ISO 9001:2008 framework

- The first four clauses from clause 0 (Introduction and Scope) to clause 3 (Terms and Definitions) only provide background information on the standard: its purpose, concepts and principles used.
    - Clause 4 through clause 8 provide the mandatory requirements that QMS must implement and records that are maintained for implementation.
    - Each clause has several sub clause.
- The purpose of these five major clause of ISO 9001:2008 standard is given above:

Clause 4: Quality Management System(QMS): discusses requirements to identify, plan, document, operate and control QMS processes and to continually improve the effectiveness of QMS.

Clause 5: Management Responsibility specifies the responsibilities of top management, which is expected to demonstrate its commitment to develop, implement and continually improve quality.

Clause 6: Resource Management specifies the commitment of the management to determine, provide and control the various resources needed to operate and manage QMS processes

Clause 7: Product realization deals with the processes essential to plan, operate and control the specific processes.

Clause 8: Measurement, Analysis and improvement : QMS must give enough emphasis on continual improvement and requirements to plan, measure, analyses and improve processes that demonstrate the product quality.

ii. SEI's CMMI

- Federally funded body based at Carnegie Mellon University
- Established in 1986 by the federal government to improve the state of the practice of software engineering
- SEI concepts are now widely used in both commercial and government organizations.

1.13 What is CMM ?

- CMM stands for**C**apability**M**aturity**M**odel.
- Focuses on elements of essential practices and processes from various bodies of knowledge.
- Describes common sense, efficient, proven ways of doing business (which you should already be doing) -not a radical new approach.
- CMM is a method to evaluate and measure the maturity of the software development process of an organizations.
- CMM measures the maturity of the software development process on a scale of 1 to 5.
- CMM v1.0 was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in USA.

- CMM was originally developed for Software Development and Maintenance but later it was developed for :

  ◦Systems Engineering

  ◦Supplier Sourcing

  ◦Integrated Product and Process Development

  ◦People CMM

  ◦Software Acquisition

  ◦Others...

What is CMMI ?

- CMM Integration project was formed to sort out the problem of using multiple CMMs. CMMI Product Team's mission was to combine three**Source Models**into a single improvement .These three Source Models are :
  - Capability Maturity Model for Software (SW-CMM) -v2.0
  - Electronic Industries Alliance Interim Standard (EIA/IS) -731 Systems Engineering
  - Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98
- **CMM Integration:**

  -builds an initial set of integrated models.

  -improves best practices from source models based on lessons learned.

  -establishes a framework to enable integration of future models.
- CMMI -Maturity Levels
  - A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.
  - CMMI models with staged representation, have five maturity levels designated by the numbers 1 through 5. They are −

1.Initial

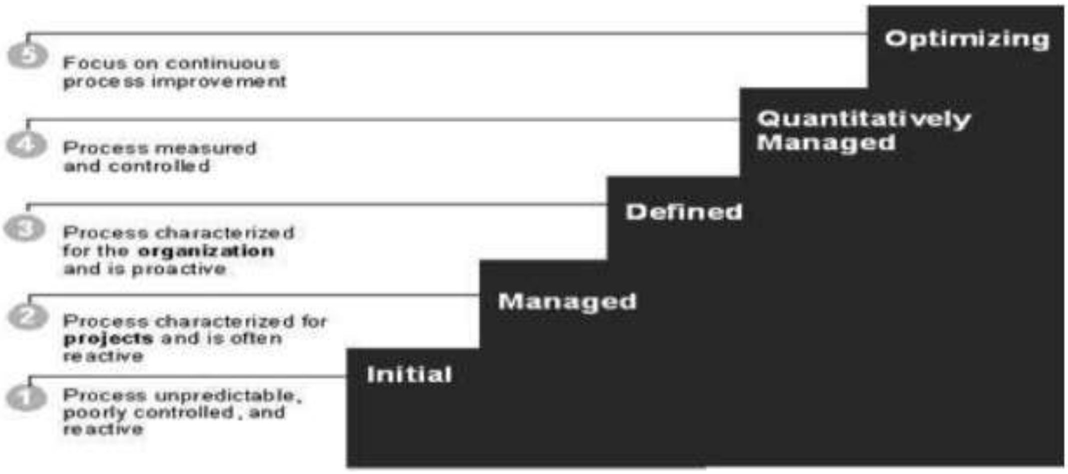
2.Managed

3.Defined

4.Quantitatively Managed

5.Optimizing

- CMMI Staged Representation Maturity Levels
- The following image shows the maturity levels in a CMMI staged representation.



Optimizing
5
Focus on continuous process improvement
Quantitatively Managed
4
Process measured and controlled
Defined
3
Process characterized for the organization and is proactive
Managed
2
Process characterized for projects and is often reactive
Initial
1
Process unpredictable, poorly controlled, and reactive

Maturity Level 1 Initial

- At maturity level 1, processes are usually ad hoc and chaotic.
- The organization usually does not provide a stable environment.
- Success in these organizations depend on the competence and heroics of the people in the organization and not on the use of proven processes.
- Maturity level 1 organizations often produce products and services that work

Maturity Level 2 Managed

- At maturity level 2, an organization has achieved all the**specific**and**generic goals**of the maturity level 2 process areas.

- In other words, the projects of the organization have ensured that requirements are managed and that processes are planned, performed, measured, and controlled.

- The process discipline reflected by maturity level 2 helps to ensure that existing practices are retained during times of stress.

Maturity Level 3 Defined

- At maturity level 3, an organization has achieved all the**specific**and**generic goals**of the process areas assigned to maturity levels 2 and 3.

- At maturity level 3, processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

- A critical distinction between maturity level 2 and maturity level 3 is the scope of standards, process descriptions, and procedures.

Maturity Level 4 Quantitatively Managed

- At maturity level 4, an organization has achieved all the**specific goals**of the process areas assigned to maturity levels 2, 3, and 4 and the**generic goals**assigned to maturity levels 2 and 3.

- At maturity level 4, sub-processes are selected that significantly contribute to the overall process performance.

- These selected sub-processes are controlled using statistical and other quantitative techniques.

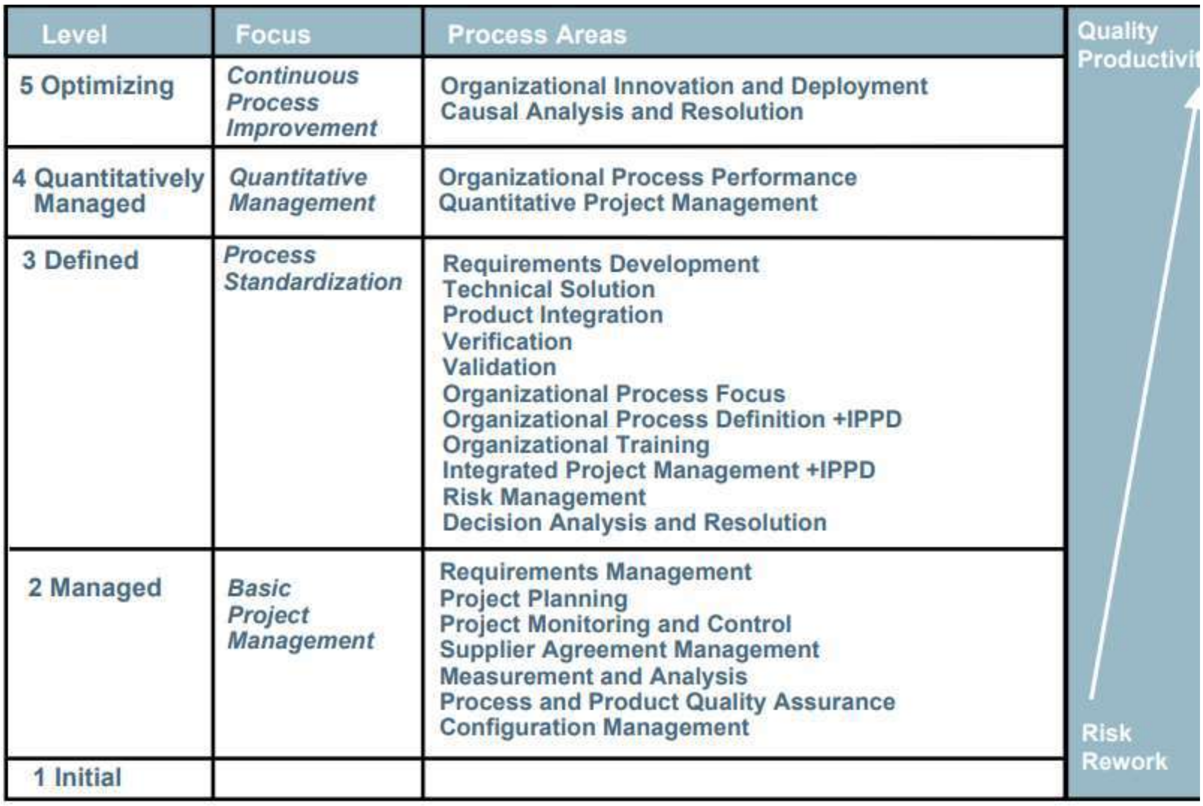
Maturity Level 5 Optimizing

- At maturity level 5, an organization has achieved all the**specific goals** of the process areas assigned to maturity levels 2, 3, 4, and 5 and the**generic goals**assigned to maturity levels 2 and 3.

- Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes.

- This level focuses on continually improving process performance through both incremental and innovative technological improvements.

Maturity Levels and Process Areas



Level	Focus	Process Areas	Quality Productivit
5 Optimizing	*Continuous Process Improvement*	Organizational Innovation and Deployment Causal Analysis and Resolution	
4 Quantitatively Managed	*Quantitative Management*	Organizational Process Performance Quantitative Project Management	
3 Defined	*Process Standardization*	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition +IPPD Organizational Training Integrated Project Management +IPPD Risk Management Decision Analysis and Resolution	
2 Managed	*Basic Project Management*	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management	
1 Initial			Risk Rework

1.14 Review

- In a review, the producer and other individuals examine the work product for defects.
- The work product is deliverable or the outcome of any phase in the software development lifecycle.
- An example of work products would be requirement models.
- The objective of review is to detect any defect on the outcome of each software development lifecycle phase.
- Reviews can be formal or informal as briefly described below.

i. Formal Reviews

- This type of review is conducted at the end of each phase in software development life cycle.
- Formal review is unlike informal review, since it has formal agenda and definite schedule.

- Moreover, the date and time for performing formal review is addressed in the project plan.

- In the formal review, materials must be well organized and prepared, such as a software requirements review.

ii. informal review

- An informal review is a process in which the software tester asks a member of the software team to review the work products of each phase in software development .

- Informal reviews can be conducted whenever needed.

- This type of review is done without a formal agenda or definite schedule.

- The date and the time of conducting any informal review process will not be addressed in project plan.

- Therefore, the software member can conduct informal review anytime as needed.

- The material of informal review may be as “informal as a computer listing or hand-written documentation.”

iii. Walkthrough

- Walkthrough is considered as the form of "software peer review" in which a work product is examined by the author of work product along with one or more colleagues to evaluate technical parts of software product.

- In a walkthrough, the author presents a full description of software product to participants and asks them to write some comments regarding the technical quality or document content.

- The walkthrough review is informally conducted to gain feedback about the technical content of the software product document.

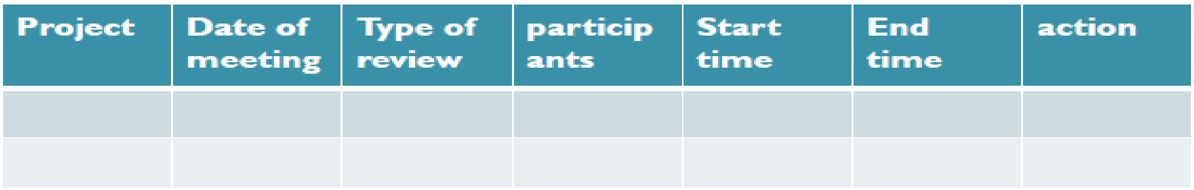
1.15 INSPECTIONS

- Inspection is one type of ‘formal peer review' in the software development life cycle .
- Formal inspection review is the most effective technique for software peer review .
- This type of software review aims to find defects and problems mostly in documents, such as requirements, specifications, test plans, test cases, and coding.
- Another benefit of using inspection review is to find problems in each phase of software development and report these problems without fixing them.
- Inspection teams are consisted of a moderator, author, reader, inspector and recorder .
- Each one has a different responsibility. The moderator is responsible for managing the inspection, reporting and scheduling meetings, collecting data and overseeing the follow-up.
- The author's responsibility is to define the artifactsto be inspected.
- The author also provides an overview of the product and may respond to questions raised about the product during the inspection.
- While the reader is responsible for leading the team through the artifactsbeing inspected during the meeting, the inspector identifies defects in each work product.
- Finally, the recorder is responsible for documenting all defects in the list during the inspection meeting.

Inspection forms and check lists

- A forms and checklists vary from organisation to organisation.

- A sample form for recording the proceedings of a review meeting given below:



Project	Date of meeting	Type of review	participants	Start time	End time	action

- The inspection process should always have a checklist of common errors before the inspection starts and this should be done by discussion with experienced members.

- For code inspection, different checklists should be prepared for different programming languages and organizations generally prepared their own checklists.

Rate Inspection

- The amount of code which can be inspected in a given time depends upon the experience of the inspection team, the programming language and the application domain.

- Large size of material to be inspected needs a preparation rate that is too high.

- Perparationrate that is too high leads to a high rate of inspection and high rate of inspection causes fewer defects to be found.

- Measurement data collected at IBM has revealed that during individual preparation, about 125 source code statements per hour is acceptable and about 90-125 statements per hour can be inspected during formal inspection meeting.

- It has suggested that the maximum time spent on an inspection meeting should be about two hours as the efficiency of the defects detection process falls off after that time.

Inspection Metrics

- The data collected during inspection of a software program is used to compute a set of metrics that help in improvement of the inspection process which in turn helps in the improvement of quality.
- The metrics required to be computed during inspection need to be defined by SQA team of an organization.
- Many different metrics including the following can be calculated during an inspection process.

Number of major and minor defects found.

- The number of major defects found to total found. (If the proportion of minor defects to major defects is too large a moderator may request that the reviewer repeat the review, focusing on major defects, prior to commencing the logging meeting)
- The size of the artifact(pages, LOC, ...)
- The rate of review -the size of the reviewed artifactdivided by time (normally expressed in hours) (e.g., 15 pages /hour).
- The defect detection rate -the number of major defects found per review hour.
- Total Number of Defects Found and Defects Density -Total number of defects found is the sum of the total number of defect s found by each reviewer, minus the number of common defects found.

  Total Defects Found = A + B -C

  Where A and B are the number found by reviewer A and B respectively and C is the number found by both A and B.
- Defect density is the ratio of the number of defects found to the size of the artifact. It is given by

  Defect Density = Total Defects Found / Size

Where the size of the artifactis measured in number of pages, loc, or other size measure.

- ➢ Estimated Total Number of Defects

The estimated total number of defects is the sum of the total number of defects found and the estimated total number of defects remaining.

- ➢ Population Sampling Approach -Suppose we have a fish farm and we want to estimate the total number of fish we have. We could apply the capture-recapture method:

1. Capture a number of fish, tag them and release them (let this number be S1).
2. Allow time for the first sample population to redistribute.
3. Capture a second number of fish (let this number be S 2).
4. Count the number of tagged fish in the second population (let this number be S T).
5. Calculate the proportion of tagged fish in the second population (let this number be T, then T= S T/ S 2).
6. We assume that T is representative of the proportion of tagged fish in the total population (POP), so T * POP= S1, or for our purposes POP= S1 /T.

- ➢ Using the population sampling approach to estimate the number of defects remaining leads to the following steps: -
- ➢ Let the number of defects found by one reviewer be the tagged population (A).

1. Assume an even likelihood of finding all defects (even distribution,...)

2. Count the number of defects found by the second reviewer (B).

3. Count the number of defects found by the second reviewer that were also found by the first (C the common defects).

4. Calculate the proportion of common defects in the second reviewers defects (T=C/B).

5. We assume that T is representative of the proportion of common defects in the total number of defects (EstTot), so T * EstTot=A, or for our purposes EstTot= A/T = (A * B)/C.

So assuming that defects are equally likely to be found in an artifactand each reviewer is equally likely to find every defect, then:

Estimated Total Defects = (A * B)/C

UNIT – II

2.1 TESTING TECHNIQUES

- Testing is a crucial element of software development.
- It can also be a complex activity to structure correctly, and in a way that supports maximum efficiency.
- These are the principles that have been collated and established by the ISTQB as testing and software development has evolved over the years, and are recognised as the absolute core of testing.
- The seven principles of testing

1. *Testing shows the presence of defects, not their absence*
2. *Exhaustive testing is impossible*
3. *Early testing saves time and money*
4. *Defects cluster together*
5. *Beware of the pesticide paradox*
6. *Testing is context dependent*
7. *Absence-of-errors is a fallacy*

2.2 Software Tester Role

- A Software tester (software test engineer) should be capable of designing test suites and should have the ability to understand usability issues.
- It is very important for a software tester to have great communication skills so that he can interact with the development team efficiently.
- The roles and responsibilities for a usability software tester are as follows:

1. A Software Tester is responsible for designing testing scenarios for usability testing.
2. He is responsible for conducting the testing, thereafter analyze the results and then submit his observations to the development team.
3. He may have to interact with the clients to better understand the product requirements or in case the design requires any kind of modifications.
4. Software Testers are often responsible for creating test-product documentation and also has to participate in testing related walk through.

- A software tester has different sets of roles and responsibilities. He should have in depth knowledge about software testing.
- He should have a good understanding about the system which means technical (GUI or non-GUI human interactions) as well as functional product aspects.
- In order to create test cases it is important that the software tester is aware of various testing techniques and which approach is best for a particular system
- The responsibilities of the software tester include:

1. Creation of test designs, test processes, test cases and test data.
2. Carry out testing as per the defined procedures.
3. Participate in walkthroughs of testing procedures.
4. Prepare all reports related to software testing carried out.
5. Ensure that all tested related work is carried out as per the defined standards and procedures.

Software Test Manager Role

- Managing or leading a test team is not an easy job.
- The company expects the test manager to know testing methodologies in detail.

- A test manager has to take very important decisions regarding the testing environment that is required, how information flow would be managed and how testing procedure would go hand in hand with development.
- He should have sound knowledge about both manual as well as automated testing so that he can decide how both the methodologies can be put together to test the software.
- A test manager should have sound knowledge about the business area and the client's requirement, based on that he should be able to design a test strategy, test goal and objectives.
- He should be good at project planning, task and people coordination, and he should be familiar with various types of testing tools. Many people get confused between the roles and responsibilities of a test manager and test lead.
- The responsibilities of the test manager are as follows:

1. Since the test manager represents the team he is responsible for all interdepartmental meetings.
2. Interaction with the customers whenever required.
3. A test manager is responsible for recruiting software testing staff. He has to supervise all testing activities carried out by the team and identify team members who require more training.
4. Schedule testing activities, create budget for testing and prepare test effort estimations.
5. Selection of right test tools after interacting with the vendors. Integration of testing and development activities.
6. Carry out continuous test process improvement with the help of metrics.
7. Check the quality of requirements, how well they are defined.
8. Trace test procedures with the help of test traceability matrix.

2.3 Essential skills of a tester

Technical and Non-Technical required to become a Software Tester.

Non-Technical Skills

Following skills are essential to becoming a good software tester. Compare your skill set against the following checklist to determine whether Software Testing is a reality for you-

- **Analytical skills**: A good software tester should have sharp analytical skills. Analytical skills will help break up a complex software system into smaller units to gain a better understanding and create test cases.
- **Communication skill**: A good software tester must have good verbal and written communication skill. Testing artifacts (like test cases/plans, test strategies, bug reports, etc.) created by the software tester should be easy to read and comprehend.
- **Time Management & Organization Skills:** Testing at times could be a demanding job especially during the release of code. A software tester must efficiently manage workload, have high productivity, exhibit optimal time management, and organization skills
- **GREAT Attitude:** To be a good software tester you must have a GREAT attitude. An attitude to 'test to break', detail orientation, willingness to learn and suggest process improvements.
- **Passion:** To Excel in any profession or job, one must have a significant degree of the passion for it. A software tester must have a passion for his / her field

Technical Skills

This list is long, so please bear with us

- **Basic knowledge of Database/ SQL:** Software Systems have a large amount of data in the background. This data is stored in different types of databases like Oracle, MySQL, etc. in the backend. In that case, simple/complex SQL queries can be used to check whether proper data is stored in the backend databases.
- **Basic knowledge of Linux commands:** Most of the software applications like Web-Services, Databases, Application Servers are deployed on Linux machines.So it is crucial for testers to have knowledge about Linux commands.
- **Knowledge and hands-on experience of a Test Management Tool:**Test Management is an important aspect of Software testing. Without proper test management techniques, software testing process will fail. Test management is nothing but managing your testing related artifacts.

  For example - A tool like Testlink can be used for tracking all the test cases written by your team.

2.4 Types of Software Testing

Software testing is generally classified into two main broad categories: functional testing and non-functional testing. There is also another general type of testing called maintenance testing.

Functional Testing

Functional testing involves the testing of the functional aspects of a software application. When you’re performing functional tests, you have to test each and every functionality. You need to see whether you’re getting the desired results or not.

There are several types of functional testing, such as:

- Unit testing
- Integration testing
- End-to-end testing
- Smoke testing
- Sanity testing
- Regression testing
- Acceptance testing
- White box testing
- Black box testing
- Interface testing

Functional tests are performed both manually and using automation tools. For this kind of testing, manual testing is easy, but you should use tools when necessary.

Some tools that you can use for functional testing are Micro Focus UFT (previously known as QTP, and UFT stands for Unified Functional Testing), Selenium, JUnit, soapUI, Watir, etc.

Non-functional Testing

Non-functional testing is the testing of non-functional aspects of an application, such as performance, reliability, usability, security, and so on. Non-functional tests are performed after the functional tests.

There are several types of non-functional testing, such as:

- Performance testing
- Security testing
- Load testing
- Failover testing
- Compatibility testing

- Usability testing
- Scalability testing
- Volume testing
- Stress testing
- Maintainability testing
- Compliance testing
- Efficiency testing
- Reliability testing
- Endurance testing
- Disaster recovery testing
- Localization testing
- Internationalization testing

Unit Testing

Testing each component or module of your software project is known as unit testing. To perform this kind of testing, knowledge of programming is necessary. So only programmers do this kind of tests, not testers.

You have to do a great deal of unit testing as you should test each and every unit of code in your project.

Integration testing

After integrating the modules, you need to see if the combined modules work together or not. This type of testing is known as integration testing. You need to perform fewer integration tests than unit tests.

Some good tools for unit and integration testing are Jasmine, Mocha, etc.

End-to-end Testing

End-to-end testing is the functional testing of the entire software system. When you test the complete software system, such testing is called end-to-end testing. You need to perform fewer end-to-end tests than integration tests.

Cucumber, Protractor, Jasmine, Karma, etc. are some great end-to-end testing tools.

User Interface Testing

User interface testing involves the testing of the application's user interface. The aim of UI tests is to check whether the user interfaces have been developed according to what is described in the requirements specifications document.

By running UI tests, you can make the application's user interfaces more user-friendly and appealing to the eyes.

Some great automated user interface testing tools are Monkey test for Android, Saucelabs, and Protractor.

Accessibility testing

Testing whether your software is accessible to disabled people or not is termed as accessible testing. For this type of tests, you need to check if disabled people such as those who are color blind, blind, and deaf can use your application.

The right choice of color and contrast need to be made to make your software accessible to color-blind people.

Alpha testing

Alpha testing is a kind of testing to look for all the errors and issues in the entire software.

This kind of test is done at the last phase of app development and is performed at the place of the developers, before launching the product or before delivering it to the client to ensure that the user/client gets an error-free software application.

Alpha testing is run before the beta testing, which means that after performing alpha testing, you need to run beta testing.

Alpha testing is not performed in the real environment. Rather, this kind of tests is done by creating a virtual environment that resembles a real environment.

Beta testing

Beta testing is done before the launch of the product. It is carried out in a real user environment by a limited number of actual customers or users, in order to be certain that the software is completely error-free and it functions smoothly.

After collecting feedback and constructive criticism from those users, some changes are made to make the software better.

So when the software is under beta testing, it is called beta version of the software. After this testing is complete, the software is released to the public.

Performance testing

Performance tests are run to check if the software's performance is good or not. There are performance testing tools that analyze your app's performance and show you the performance issues. By fixing those issues, you'll be able to increase the performance of your software application.

Some great performance testing tools, also known as load testing tools, for web applicationsare WebLOAD, LoadView, NeoLoad, LoadNinja, Appvance, Load Runner, Apache JMeter, Loadster, LoadImpact, Testing Anywhere, SmartMeter.io, Tricentis Flood, Rational Performance Tester, LoadComplete, etc.

Load testing

Load testing is one kind of performance testing that tests how much load a system can take before the software performance begins to degrade. By running load tests, we can know the capacity of taking load of a system.

You can run load tests using tools like LoadRunner, WebLoad, JMeter, etc.

Black box testing

Performed by the QA team of a company, black box testing is a testing technique that involves the checking of the application's functionality without having any technical knowledge of the application, like the knowledge of the code's logic, how the code works, knowledge of the internal structure, etc.

White box testing

Performed by the development team, white box testing is a testing method that requires a good understanding of the application's code. It requires great knowledge of the app's internal logic.

Security testing

Security tests are performed to ensure the security of your application, in order that security breaches can be prevented. Security experts run this kind of tests to see how much your software is secure from attacks and to find security issues so that the app's security can be strengthened.

The top website security testing tools include Grabber, Arachni, Iron Wasp, Nogotofail, SQLMap, W3af, Wapiti, Wfuzz, Zed Attack Proxy, etc.

Acceptance testing

The client who will purchase your software will perform acceptance testing (also known as User Acceptance Testing) to see if the software can be accepted or not by checking whether your software meets all the client's requirements and preferences.

2.5 Evaluating the quality of test cases

- ➢ Delivering quality software also require the testers to evaluate the quality of test cases to make sure that they are able to detect defects in the software under test.
- ➢ Two methods are used for the purpose:

1. **Error Seeding :**

- ➢ Error Seeding is the process of deliberately introducing errors within a program to check whether the test cases are able to capture the seeded errors.
- ➢ This technique aims to detect errors in order to find out the ratio between the actual and artificial errors.
- ➢ Artificial faults are the unknown faults and actual faults are the injected faults, hence test cases are used to check presence of such faults.
- ➢ The reason for error seeding is that both testers and developers get a chance to challenge their respective responsibilities.
- ➢ It is basically an estimation technique which helps to figure out the presence of real errors on the basis of the number of seeded errors found.

2. **Mutation Testing**

- **Mutation Testing** is a type of software testing in which certain statements of the source code are changed/mutated to check if the test cases are able to program should be kept extremely small that it does not affect the overall objective of the program.
- Mutation Testing is also called Fault-based testing strategy as it involves creating a fault in the program and it is find errors in source code.
- The goal of Mutation Testing is ensuring the quality of test cases in terms of robustness that it should fail the mutated source code.
- The changes made in the mutant a type of White Box Testing which is mainly used for Unit Testing.
- Mutation was originally proposed in 1971 but lost fervor due to the high costs involved. Now, again it has picked steam and is widely used for languages such as Java and XML.

2.6 **White Box Testing**

White box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and transparent box testing**.

It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing.

In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

WHITE BOX TESTING
INPUT
OUTPUT

Control Flow Testing

Control flow testing is a testing technique that comes under white box testing. The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure.

The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. It is mostly used in unit testing. Test cases represented by the control graph of the program.

Control Flow Graph is formed from the node, edge, decision node, junction node to specify all possible execution path.

Notations used for Control Flow Graph

1. Node
2. Edge
3. Decision Node
4. Junction node

Node

Nodes in the control flow graph are used to create a path of procedures. Basically, it represents the sequence of procedures which procedure is next to come so, the tester can determine the sequence of occurrence of procedures.

Edge

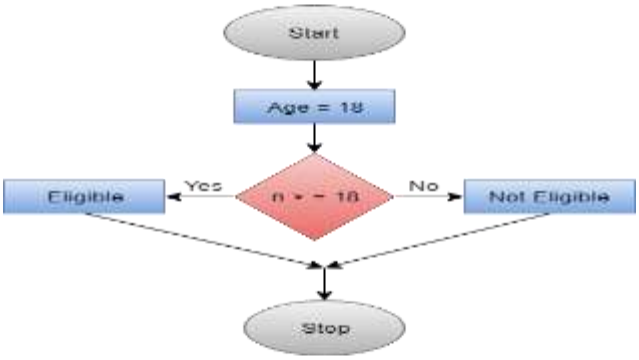
Edge in control flow graph is used to link the direction of nodes.

Decision node

Decision node in the control flow graph is used to decide next node of procedure as per the value.

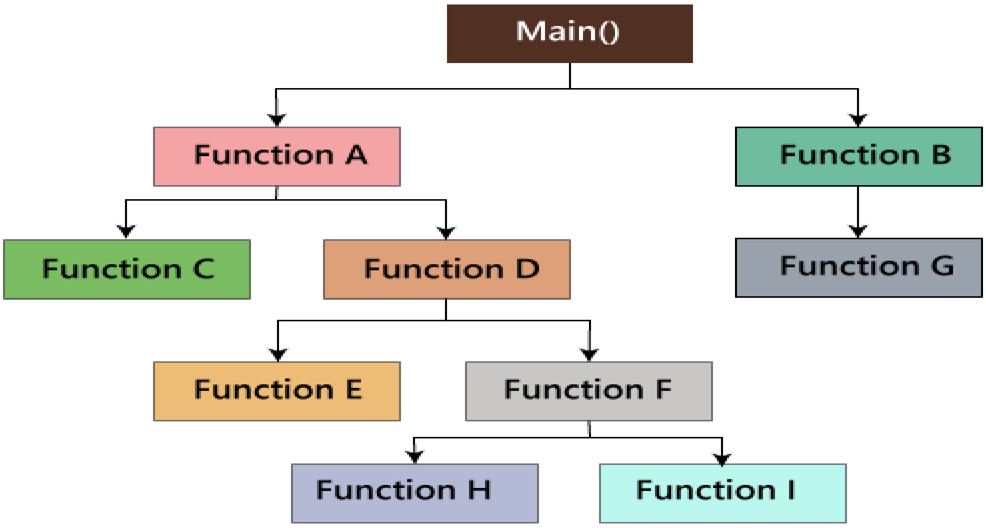
Junction node

Junction node in control flow graph is the point where at least three links meet.



Path testing

In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the program and also show how every program is added with one another as we can see in the below image:



And test all the independent paths implies that suppose a path from main() to function G, first set the parameters and test if the program is correct in that particular path, and in the same way test all other paths and fix the bugs.

LCSAJ Testing

LCSAJ stands for Linear Code Sequence and Jump, a white box testing technique to identify the code coverage, which begins at the start of the program or branch and ends at the end of the program or the branch.

LCSAJ consists of testing and is equivalent to statement coverage.

LCSAJ Characteristics:

- 100% LCSAJ means 100% Statement Coverage
- 100% LCSAJ means 100% Branch Coverage
- 100% procedure or Function call Coverage
- 100% Multiple condition Coverage

Loop Testing

Loop testing a white box testing technique performed to validate the loops. There are four kinds of loops as mentioned below:

- Simple Loops
- Nested Loops
- Concatenated Loops
- Unstructured Loops

What is tested in Loop Testing?

- Loops Testing reveals loops initialization problems.
- By going through the loop once, the uninitialized variables in the loop can be determined.
- Testing can also fix loop repetition issues.
- Loops can also reveal capacity/performance bottlenecks.

Data flow Testing

- The programmer can perform numerous tests on data values and variables. This type of testing is referred to as data flow testing.
- It is performed at two abstract levels: static data flow testing and dynamic data flow testing.
- The static data flow testing process involves analyzing the source code without executing it.
- Static data flow testing exposes possible defects known as data flow anomaly.
- Dynamic data flow identifies program paths from source code.

Steps of Data Flow Testing

- creation of a data flow graph.
- Selecting the testing criteria.

- Classifying paths that satisfy the selection criteria in the data flow graph.
- Develop path predicate expressions to derive test input.

The life cycle of data in programming code

- **Definition:** it includes defining, creation and initialization of data variables and the allocation of the memory to its data object.
- **Usage:** It refers to the user of the data variable in the code. Data can be used in two types as a predicate(P) or in the computational form(C).
- **Deletion:** Deletion of the Memory allocated to the variables.

Types of Data Flow Testing

- **Static Data Flow Testing**

No actual execution of the code is carried out in Static Data Flow testing. Generally, the definition, usage and kill pattern of the data variables is scrutinized through a control flow graph.

- **Dynamic Data Flow Testing**

The code is executed to observe the transitional results. Dynamic data flow testing includes:

- Identification of definition and usage of data variables.
- Identifying viable paths between definition and usage pairs of data variables.
- Designing & crafting test cases for these paths.

Advantages of Data Flow Testing

- Variables used but never defined,
- Variables defined but never used,
- Variables defined multiple times before actually used,
- DE allocating variables before using.

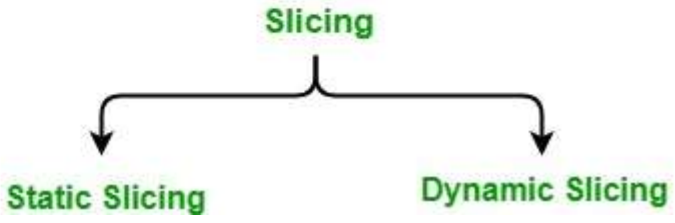
Data Flow Testing Limitations

- Testers require good knowledge of programming.
- Time-consuming
- Costly process.

Slice based Testing

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.



1.Staticslicing:

- A static slice of a program contains all statements that may affect the value of a variable at any point for any arbitrary execution of the program.

- Static slices are generally larger.
- It considers every possible execution of the program.

2. Dynamic slicing:

- A dynamic slice of a program contains all the statements that actually affect the value of a variable at any point for a particular execution of the program.
- Dynamic slices are generally smaller.

2.7 **Overview of white box testing**

Generic steps of white box testing

- Design all test scenarios, test cases and prioritize them according to high priority number.
- This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations and so on.
- In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
- This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
- In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing

- It identifies internal security holes.
- To check the way of input inside the code.
- Check the functionality of conditional loops.

- To test function, object, and statement at an individual level.

Advantages of White box testing

- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing

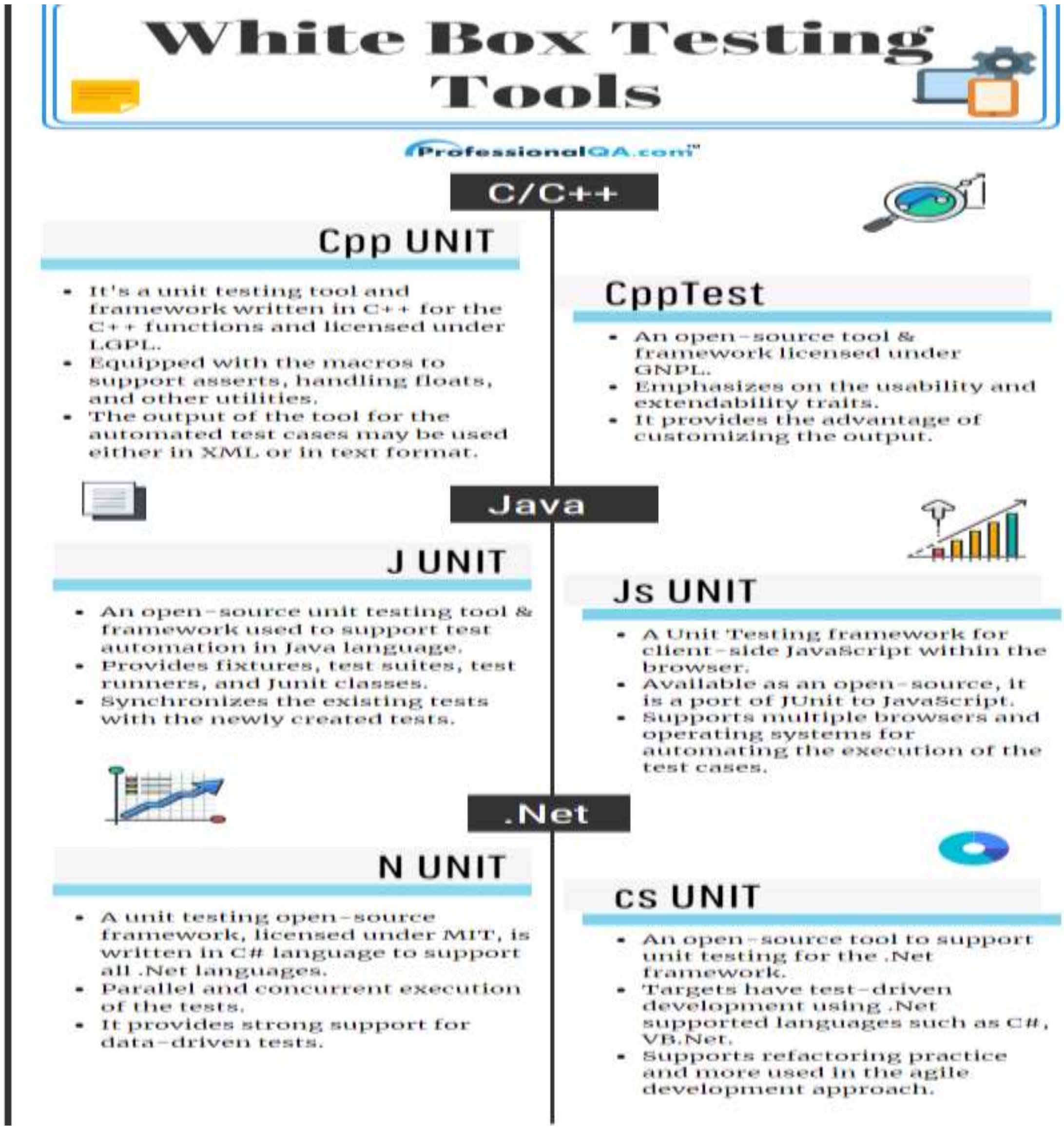
- White box testing is too much time consuming when it comes to large-scale programming applications.
- White box testing is much expensive and complex.
- It can lead to production error because it is not detailed by the developers.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

Techniques Used in White Box Testing

Data Flow Testing	Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.
Control Flow Testing	Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a

	particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.
Branch Testing	Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
Statement Testing	Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.
Decision Testing	This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

White Box Testing Tools



White Box Testing Tools

ProfessionalQA.com™

C/C++

Cpp UNIT

- It's a unit testing tool and framework written in C++ for the C++ functions and licensed under LGPL.
- Equipped with the macros to support asserts, handling floats, and other utilities.
- The output of the tool for the automated test cases may be used either in XML or in text format.

CppTest

- An open-source tool & framework licensed under GNPL.
- Emphasizes on the usability and extendability traits.
- It provides the advantage of customizing the output.

Java

J UNIT

- An open-source unit testing tool & framework used to support test automation in Java language.
- Provides fixtures, test suites, test runners, and Junit classes.
- Synchronizes the existing tests with the newly created tests.

Js UNIT

- A Unit Testing framework for client-side JavaScript within the browser.
- Available as an open-source, it is a port of JUnit to JavaScript.
- Supports multiple browsers and operating systems for automating the execution of the test cases.

.Net

N UNIT

- A unit testing open-source framework, licensed under MIT, is written in C# language to support all .Net languages.
- Parallel and concurrent execution of the tests.
- It provides strong support for data-driven tests.

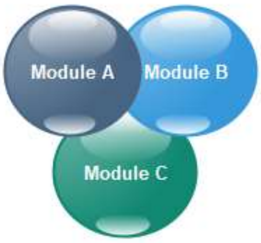
cs UNIT

- An open-source tool to support unit testing for the .Net framework.
- Targets have test-driven development using .Net supported languages such as C#, VB.Net.
- Supports refactoring practice and more used in the agile development approach.

2.8 .Integration testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group.

The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

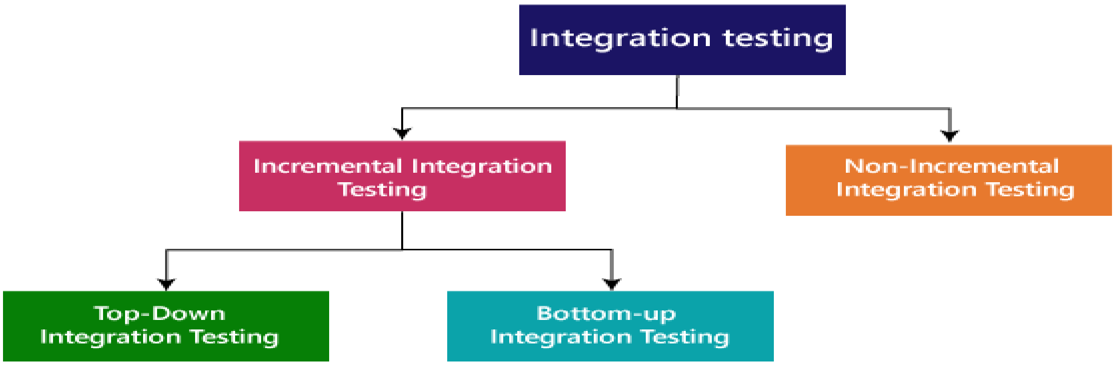


Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as **integration testing**.

Types of Integration Testing

Integration testing can be classified into two parts:

- **Incremental integration testing**
- **Non-incremental integration testing**


Integration testing
Incremental Integration Testing
Non-Incremental Integration Testing
Top-Down Integration Testing
Bottom-up Integration Testing

Incremental Approach

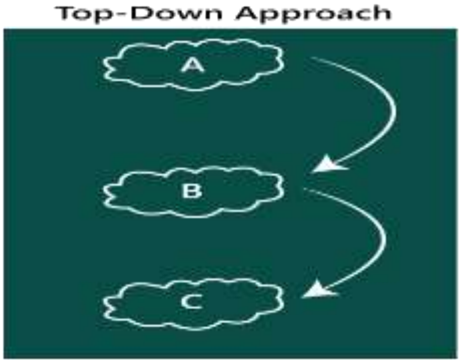
In the Incremental Approach, modules are added in ascending order one by one or according to need. The selected modules must be logically related. Generally, two or more than two modules are added and tested to determine the correctness of functions. The process continues until the successful testing of all the modules.

Incremental integration testing is carried out by further methods:

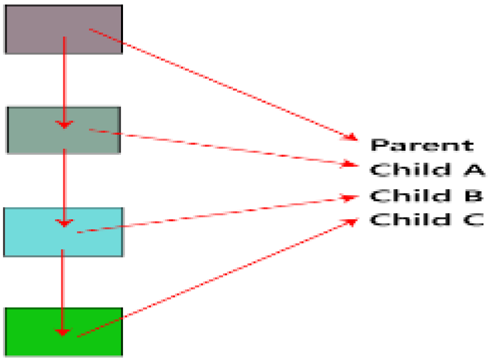
- Top-Down approach
- Bottom-Up approach

Top-Down Approach

The top-down testing strategy deals with the process in which higher level modules are tested with lower level modules until the successful completion of testing of all the modules. Major design flaws can be detected and fixed early because critical modules tested first. In this type of method, we will add the modules incrementally or one by one and check the data flow in the same order.



In the top-down approach, we will be ensuring that the module we are adding is the **child of the previous one like Child C is a child of Child B** and so on as we can see in the below image:



Advantages:

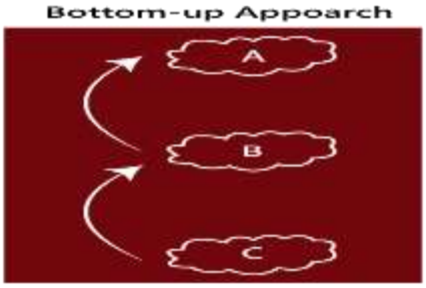
- Identification of defect is difficult.
- An early prototype is possible.

Disadvantages:

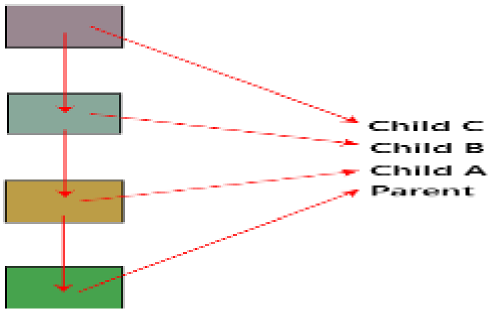
- Due to the high number of stubs, it gets quite complicated.
- Lower level modules are tested inadequately.
- Critical Modules are tested first so that fewer chances of defects.

Bottom-Up Method

The bottom to up testing strategy deals with the process in which lower level modules are tested with higher level modules until the successful completion of testing of all the modules. Top level critical modules are tested at last, so it may cause a defect. Or we can say that we will be adding the modules from **bottom to the top** and check the data flow in the same order.



In the bottom-up method, we will ensure that the modules we are adding **are the parent of the previous one** as we can see in the below image:



Advantages

- Identification of defect is easy.
- Do not need to wait for the development of all the modules as it saves time.

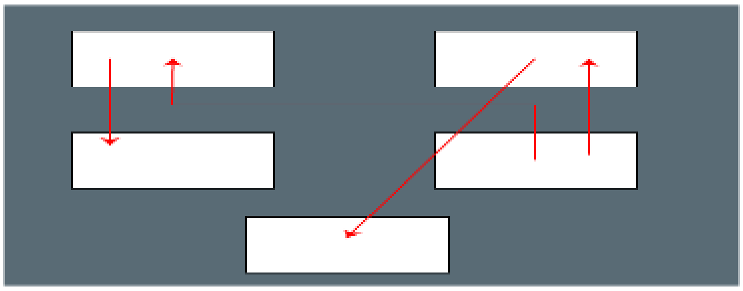
Disadvantages

- Critical modules are tested last due to which the defects can occur.
- There is no possibility of an early prototype.

In this, we have one addition approach which is known as **hybrid testing**.

Non- incremental integration testing

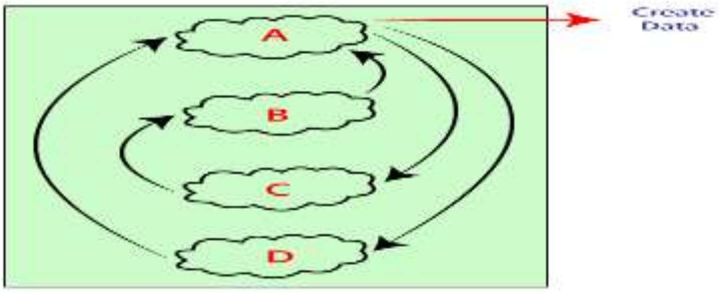
when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. Hence, it is also known as the **Big bang method**.



Big Bang Method

In this approach, testing is done via integration of all modules at once. It is convenient for small software systems, if used for large software systems identification of defects is difficult.

Since this testing can be done after completion of all modules due to that testing team has less time for execution of this process so that internally linked interfaces and high-risk critical modules can be missed easily.



Advantages:

- It is convenient for small size software systems.

Disadvantages:

- Identification of defects is difficult because finding the error where it came from is a problem, and we don't know the source of the bug.
- Small modules missed easily.
- Time provided for testing is very less.

UNIT -III

Functional Testing

- ➢ Functional testing is a form of black box testing and involves testing of all functions that the software is expected to perform.
- Functional testing involves
- ✓ Identification of functions specified in the requirement document.
- ✓ Design test scenarios
- ✓ Identification of test cases from test scenarios and
- ✓ Execution of test cases with data.
- ➢ Functional testing proceeds in Two Steps:

1. **Designing test scenarios** :is done from the specification by constructing a formal or informal model of the system.
2. **Designing test case with appropriate test data:**Test case with the test data are designed thereafter

- ➢ A test case consists of a set of input values,execution preconditions,expected results and post condition,developed to cover certain test conditions.
- ➢ A test scenarios may have multiple test cases.

3.1 Logic Based Testing

- ➢ Logic based testing handles very complex businss rules based on a set of defined conditions.
- ➢ Logic based testing is used if the specifications are dominated by logic(AND,OR,NOT operations) and predefined rules decide the actions that take places based on combinations of conditions.
- ➢ The rules are translated into either Decision table or graphs based on Boolean Algebra.
- **Decision Table**
- ➢ Decision Table is a precise and compact way to express complicated logic between inputs and outputs.

- It is used for expressing the rules that govern the handling of transactional situations and provide a means of specifying the actions that result based on different combinations of inputs.

- A Decision Table lists all possible "conditions"(inputs) and all possible "actions (Outputs).

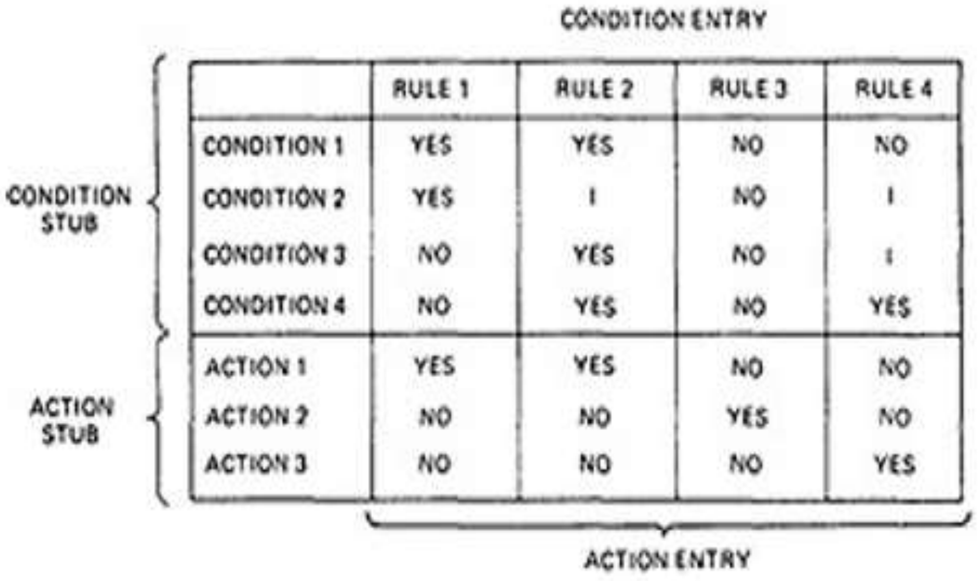
- There is a rule for each possible combination of "conditions" and the resulting actions.

- Example:

- The below table is a limited - entry decision table. It consists of four areas called the condition stub, the condition entry, the action stub, and the action entry.

- Each column of the table is a rule that specifies the conditions under which the actions named in the action stub will take place.

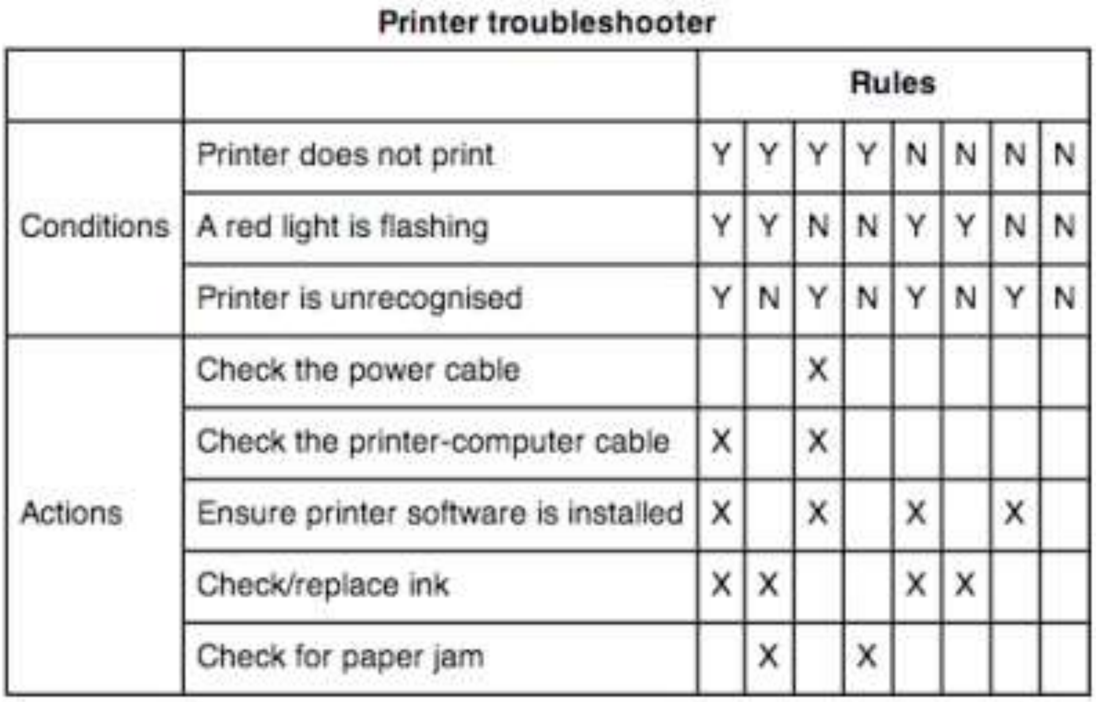
- The condition stub is a list of names of conditions.


CONDITION ENTRY
CONDITION STUB
ACTION STUB
ACTION ENTRY

	RULE 1	RULE 2	RULE 3	RULE 4
CONDITION 1	YES	YES	NO	NO
CONDITION 2	YES	I	NO	I
CONDITION 3	NO	YES	NO	I
CONDITION 4	NO	YES	NO	YES
ACTION 1	YES	YES	NO	NO
ACTION 2	NO	NO	YES	NO
ACTION 3	NO	NO	NO	YES

Fig. 1 Examples of Decision Table.

- A more general decision table can be as below:



Printer troubleshooter

		Rules							
Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognised	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

- A rule specifies whether a condition should or should not be met for the rule to be satisfied. "YES" means that the condition must be met, "NO" means that the condition must not be met, and "I" means that the condition plays no part in the rule, or it is immaterial to that rule.
- The action stub names the actions the routine will take or initiate if the rule is satisfied. If the action entry is "YES", the action will take place; if "NO", the action will not take place.
- The table in Figure 1 can be translated as follows:
- Action 1 will take place if conditions 1 and 2 are met and if conditions 3 and 4 are not met (rule 1) or if conditions 1, 3, and 4 are met (rule 2).
- "Condition" is another word for predicate.
- Decision-table uses "condition" and "satisfied" or "met". Let us use "predicate" and TRUE / FALSE.
- Now the above translations become:

✓ Action 1 will be taken if predicates 1 and 2 are true and if predicates 3 and 4 are false (rule 1), or if predicates 1, 3, and 4 are true (rule 2).

   ✓ Action 2 will be taken if the predicates are all false, (rule 3).

   ✓ Action 3 will take place if predicate 1 is false and predicate 4 is true (rule 4).

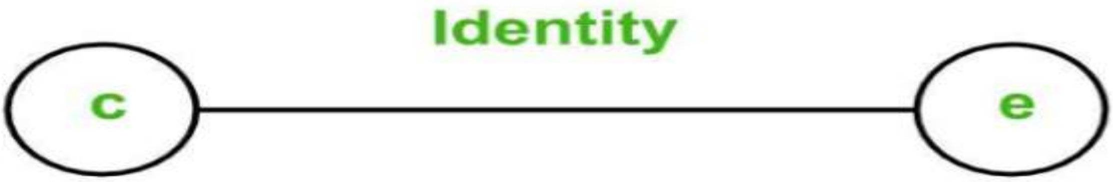
3.1.1 **Cause Effect Graph**

- **Cause Effect Graphing based technique** is a technique in which a graph is used to represent the situations of combinations of input conditions.
- The graph is then converted to a decision table to obtain the test cases.

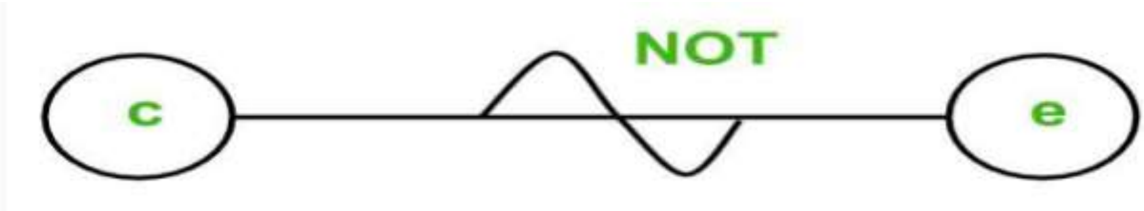
Cause-effect graphing technique is used because boundary value analysis and equivalence class partitioning methods do not consider the combinations of input conditions.

- The Cause-Effect Graph technique restates the requirements specification in terms of the logical relationship between the input and output conditions.
- Since it is logical, it is obvious to use Boolean operators like AND, OR and NOT.
- **Basic Notations used in Cause-effect graph:**
  Here **c** represents **cause** and **e** represents **effect**.
- The following notations are always **used between a cause and an effect**:

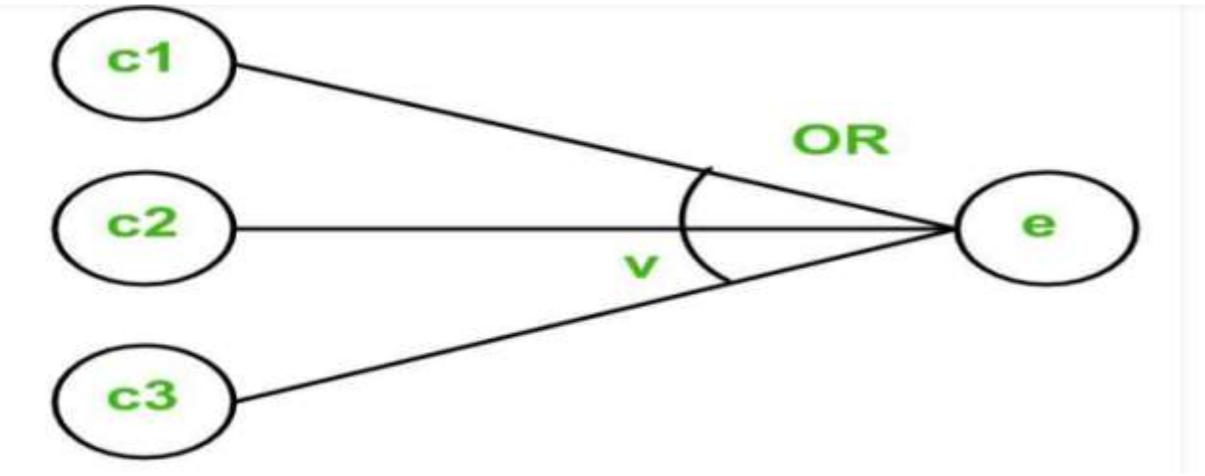
  1. Identity Function: if c is 1, then e is 1. Else e is 0.



2. NOT Function: if c is 1, then e is 0. Else e is 1.



3. OR Function: if c1 or c2 or c3 is 1, then e is 1. Else e is 0



3.2 State Transition Testing

- State transition testing is used where some aspect of the system can be described in what is called a 'finite state machine'.
- This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the 'machine'.
- This is the model on which the system and the tests are based.

➢ Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

➢ A finite state system is often shown as a **state diagram** below.

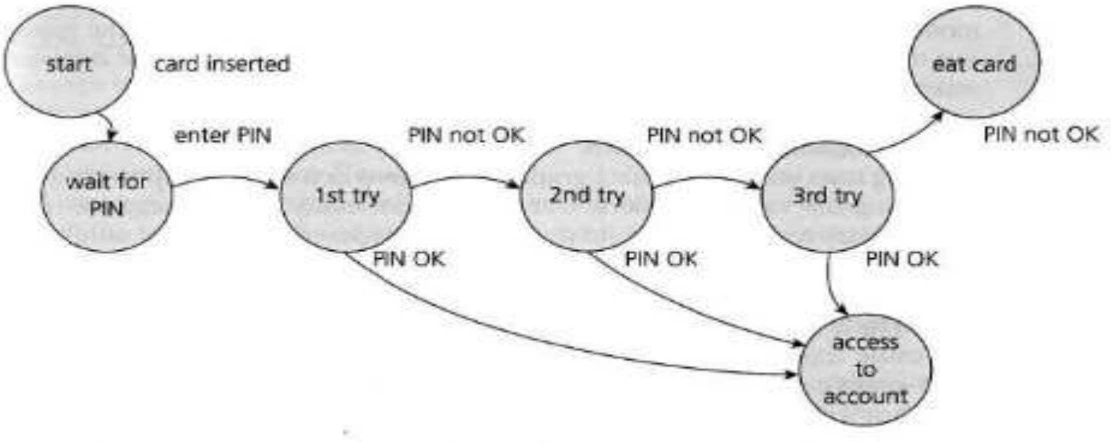
➢ One of the advantages of the state transition technique is that the model can be as detailed or as abstract as you need it to be.

➢ A **state transition model has four basic parts:**

1. *The states that the software may occupy (open/closed or funded/insufficient funds);*
2. *The transitions from one state to another (not all transitions are allowed);*
3. *The events that cause a transition (closing a file or withdrawing money);*
4. *The actions that result from a transition (an error message or being given your cash).*
5. Hence we can see that in any given state, one event can cause only one action, but that the same event – from a different state – may cause a different action and a different end state.

6. For example, if you request to withdraw $100 from a bank ATM, you may be given cash.

7. A state diagram can represent a model from the point of view of the system, the account or the customer.



- ➢ In deriving test cases, we may start with a typical scenario.

- First test case here would be the normal situation, where the correct PIN is entered the first time.

- A second test (to visit every state) would be to enter an incorrect PIN each time, so that the system eats the card.

- A third test we can do where the PIN was incorrect the first time but OK the second time, and another test where the PIN was correct on the third try. These tests are probably less important than the first two.

- Note that a transition does not need to change to a different state (although all of the transitions shown above do go to a different state). So there could be a transition from ‘access account’ which just goes back to ‘access account’ for an action such as ‘request balance’.

3.3 Use Case Based Testing

- Use Case Testing is a functional black box testing technique that helps testers to identify test scenarios that exercise the whole system on each transaction basis from start to finish.

Characteristics of Use Case Testing:

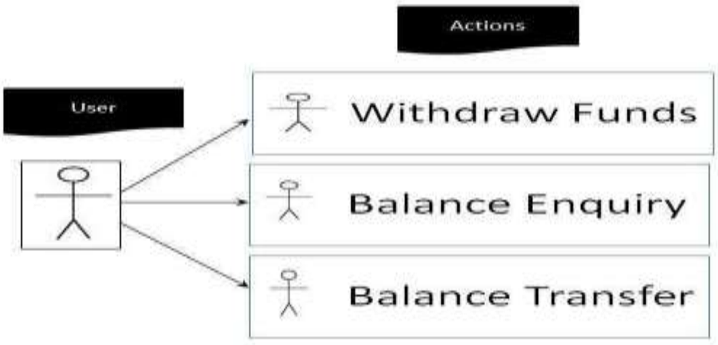
- Use Cases capture the interactions between 'actors' and the 'system'.

- 'Actors' represents user and their interactions that each user takes part into.

- Test cases based on use cases and are referred as scenarios.
- Capability to identify gaps in the system which would not be found by testing individual components in isolation.
- Very effective in defining the scope of acceptance tests.
- A use case is a tool for defining the required user interaction.
- If you are trying to create a new application or make changes to an existing application, several discussions are made.
- One of the critical discussion you have to make is how you will represent the requirement for the software solution.
- Business experts and developers must have a mutual understanding about the requirement, as it’s very difficult to attain. Any standard method for structuring the communication between them will really be a boon.
- It will, in turn, reduce the miscommunications and here is the place where Use case comes into the picture.

Example:

- The Below example clearly shows the interaction between users and possible actions.



3.4 Syntax Testing

➢ Syntax Testing is a type of **black box testing** technique which is used to examine the format and the grammar of the data inputs used in the software application, either external or input, which may be formally described in technical or established & specified notations such as BNF(Backus-Naur Form) and could be used to design input validation tests.

- ➢ Syntax testing is performed to verify and validate the both internal and external data input to the system, against the specified format, file format, database schema, protocol and other similar things.

- ➢ Syntax Testing - Steps:

- Identify the target language or format.

- Define the syntax of the language.

- Validate and Debug the syntax.

- ➢ Limitations:

- Sometimes it is easy to forget the normal cases.

- Syntax testing needs driver program to be built that automatically sequences through a set of test cases usually stored as data.

- Syntax is defined in BNF as a set of definitions. Each definition may in-turn refer to other definitions or to itself.

- The LHS of a definition is the name given to the collection of objects on the RHS.

  – ::= means "is defined as".

  – | means "or".

  – * means "zero or more occurrences".

  – + means "one or more occurrences".

  – means "n repetitions of A".

- BNF Example

special_digit ::= 0 | 1 | 2 | 5

other_digit ::= 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

ordinary_digit ::= special_digit | other_digit

exchange_part ::= other_digit ^{2} ordinary_digit

number_part ::=ordinary_digit ^{5}

phone_number ::= exchange_part number_part

Correct phone numbers: – 3469900, 9904567, 3300000 •

Incorrect phone numbers: – 0551212, 123, 8, ABCDEFG

3.5 Domain Testing:

- **Domain Testing** is a Software Testing process in which the application is tested by giving a minimum number of inputs and evaluating its appropriate outputs.
- The primary goal of Domain testing is to check whether the software application accepts inputs within the acceptable range and delivers required output.
- It is a Functional Testing technique in which the output of a system is tested with a minimal number of inputs to ensure that the system does not accept invalid and out of range input values
- It is one of the most important White Box Testing methods. It also verifies that the system should not accept inputs, conditions and indices outside the specified or valid range.
- Domain testing differs for each specific domain so you need to have domain specific knowledge in order to test a software system.
- Domain might involve testing of any one input variable or combination of input variables.
- Practitioners often study the simplest cases of domain testing less than two other names, "boundary testing" and "equivalence class analysis."

Boundary testing - Boundary value analysis (BVA) is based on testing at the boundaries between partitions. We will be testing both the valid and invalid input values in the partition/classes.

Equivalence Class testing - The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning.'

- That simplified form applies for Domain testing –

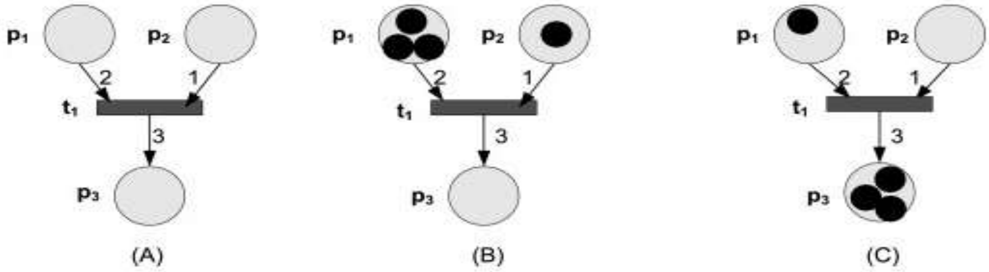
1. Only to tests of input variables
2. Only when tested at the system level
3. Only when tested one at a time
4. Only when tested in a very superficial way

3.6 Petri Net Based Testing

- Petri nets (PN) were introduced in 1962 by the PhD dissertation of Carl Adams Petri, at Technical University of Darmstandt, Germany.
- The original theory was developed as an approach to model and analyze communication systems.
- Petri Nets (PNs) are a graphic and mathematical modeling tool that can be applied in several types of systems and allow the modeling of parallel, concurrent, asynchronous and non-deterministic systems.

Petri nets model the dynamic behavior of systems.

- The places of a Petri Net contain tokens; firing of transitions removes tokens from the *input places* of the transition and adds them to its *output places*, see Figure below.
- Petri Nets can model different activities in a distributed system.
- A *transition* may model the occurrence of an event, the execution of a computational task, the transmission of a packet, a logic statement, and so on.



Petri Nets firing rules.

(A) An unmarked net with one transition t_1 with two input places, p_1 and p_2, and one output place, p_3.

(B) The marked net, the net with places populated by tokens; the net before firing the enabled transition t_1.

(C) The marked net after firing transition t_1, two tokens from place p_1 and one from place p_2 are removed and transported to place p3.

3.7 Non functional testing

- Non functional testing is just as critical as functional testing.
- And because teams need to conduct a mix of different types of testing, you need to be doing both.
- Non functional testing makes applications more usable and more reliable.
- Unfortunately, it can often be rushed in an effort to meet release deadlines.
- When non functional testing is overlooked, performance and UX defects can leave users with a bad experience and cause brand damage.
- Accessibility defects can result in compliance fines and their security could be at risk.

Non Functional Testing Types

Non functional testing is an umbrella term. There are many non functional testing types. Here are a few.

- **Accessibility testing** — Tests how usable the app is to users with disabilities, such as vision impairment.
- **Availability testing** — Tests how often the app is accessible and readily available for use.
- **Compliance testing** — Tests whether an app meets specified requirements or regulations.

- **Configuration testing** — Tests an app against software and hardware variations.
- **Disaster recovery testing** — Tests recovery of business-critical applications in emergency situations.
- **Endurance testing** — Tests an app under a heavy load over an extended period of time.
- **Failover testing** — Tests an app's backup system in the event of a system failure.
- **Geolocation testing** — Tests location-based scenarios on an app.
- **Internationalization testing** — Tests if an app can adapt to regional languages and other factors based on location.
- **Load testing** — Tests an app's performance under peak conditions.
- **Maintainability testing** — Tests the app's ability to update.
- **Performance testing** — Tests the speed and responsiveness of an app under various conditions.
- **Portability testing** — Tests how an app transfers from one software or operational environment to another.
- **Resilience testing** — Tests an app's ability to perform under stressed conditions.
- **Security testing** — Tests an app's security mechanisms to reveal vulnerabilities.
- **Scalability testing** — Tests an app's ability to scale up or down as user requests vary.
- **Stress testing** — Tests an app's stability under heavy loads or extreme conditions.
- **Usability testing** — Tests an app's ease of use.

3.8 Acceptance Testing

- ➢ Acceptance testing, a testing technique performed to determine whether or not the software system has met the requirement specifications.
- ➢ The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.

There are various forms of acceptance testing:

1. **User Acceptance Testing (UAT):**

   User acceptance testing is used to determine whether the product is working for the user correctly. Specific requirements which are quite often used by the customers are primarily picked for the testing purpose. This is also termed as *End-User* Testing.
2. **Business Acceptance Testing (BAT):**

   BAT is used to determine whether the product meets the business goals and purposes or not. BAT mainly focuses on business profits which are quite challenging due to the changing market conditions and new technologies so that the current implementation may have to being changed which result in extra budgets.

3. **Contract Acceptance Testing (CAT):**

   CAT is a contract which specifies that once the product goes live, within a predetermined period, the acceptance test must be performed and it should pass all the acceptance use cases.

   Here is a contract termed as Service Level Agreement (SLA), which includes the terms where the payment will be made only if the Product services are in-line with all the requirements, which means the contract is fulfilled.

4. **Alpha Testing:**

   Alpha testing is used to determine the product in the development testing environment by a specialized testers team usually called alpha testers.

5. **Beta Testing:**

   Beta testing is used to assess the product by exposing it to the real end-users, usually called beta testers in their environment. Feedback is collected from the users and the defects are fixed. Also, this helps in enhancing the product to give a rich user experience.

Use of Acceptance Testing:

- To find the defects missed during the functional testing phase.
- How well the product is developed.
- A product is what actually the customers need.
- Feedbacks help in improving the product performance and user experience.
- Minimize or eliminate the issues arising from the production.

3.9 REGRESSION TESTING

- **REGRESSION TESTING** is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.
- Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
- This testing is done to make sure that new code changes should not have side effects on the existing functionalities.
- It ensures that the old code still works once the latest code changes are done.
- The **Need of Regression Testing** mainly arises whenever there is requirement to change the code and we need to test whether the modified code affects the other part of software application or not.
- Moreover, regression testing is needed, when a new feature is added to the software application and for defect fixing as well as performance issue fixing.

Unit-4

INCORPORATING SPECIALIZED TESTING TECHNIQUES

4.1.Testing Object-Oriented Systems

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, subsystem testing, and system testing.

Unit Testing

In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free. Unit testing is the responsibility of the application engineer who implements the structure.

Subsystem Testing

This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside. Subsystem tests can be used as regression tests for each newly released version of the subsystem.

System Testing

System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

Object-Oriented Testing Techniques

Grey Box Testing

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are −

- **State model based testing** − This encompasses state coverage, state transition coverage, and state transition path coverage.
- **Use case based testing** − Each scenario in each use case is tested.
- **Class diagram based testing** − Each class, derived class, associations, and aggregations are tested.
- **Sequence diagram based testing** − The methods in the messages in the sequence diagrams are tested.

Techniques for Subsystem Testing

The two main approaches of subsystem testing are −

- **Thread based testing** − All classes that are needed to realize a single use case in a subsystem are integrated and tested.
- **Use based testing** − The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing

- **Alpha testing** − This is carried out by the testing team within the organization that develops software.

- **Beta testing** − This is carried out by select group of co-operating customers.
- **Acceptance testing** − This is carried out by the customer before accepting the deliverables.

4.2. Agile Software Testing

Agile Testing is a type of software testing that follows the principles of agile software development to test the software application.

All members of the project team along with the special experts and testers are involved in agile testing. Agile testing is not a separate phase and it carried out with all the development phases i.e. requirements, design and coding and test case generation. Agile testing takes place simultaneously through the Development Life Cycle.

Agile Testing Principles:

- **Shortening feedback iteration:**

  In Agile Testing, testing team get to know the product development and its quality for each and every iteration. Thus continuous feedback minimizes the feedback response time and the fixing cost is also reduced.

- **Testing is performed alongside:**

  Agile testing is not a different phase. It is performed alongside the development phase. It ensures that the features implemented during that iteration are actually done. Testing is not kept in pending for a later phase.

- **Involvement of all members:**

  Agile testing involves the each and every member of the development team and the testing team. It include various developers and experts.

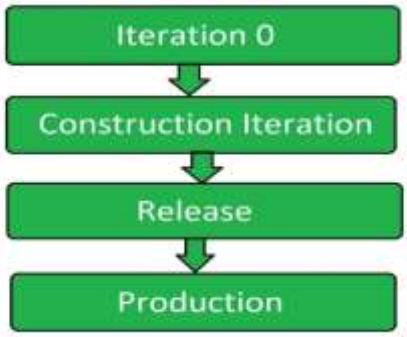
- **Documentation is weightless:**

  In place of global test documentation, agile testers use reusable checklists to suggest tests and focus on the essence of the test rather than the incidental details. Lightweight documentation tools are used.

- **Clean code:**

  The defects that are detected are fixed within the same iteration. This ensures clean code at any stage of the development.

Agile Testing Life Cycle:



1. **Iteration 0:**

   It is the first stage of the testing process and initial setup are performed in this stage. Testing environment is set in this iteration.

2. **Construction Iteration:**

   It is the second phase of the testing process. It is the major phase of the testing and most of the works are performed in this phase. It is a set of iterations to build an increment of the solution.

3. **Release:**
   This phase includes the full system testing and the acceptance testing. To finish the testing stage, the product is tested more relentless while it is in construction iterations. In this phase testers work on the defect stories.

4. **Production:**
   It is the last phase of the agile testing. Product id finalized in this stage after removal of all defects and issues raised.

Agile Testing Activities:

Agile testing includes the following activities:

- Participating in iteration planning
- Estimating tasks from the view of testing
- Writing test cases using the feature descriptions
- Unit Testing
- Integration Testing

- Feature Testing
- Defect Fixing
- Integration Testing
- Acceptance Testing
- Status Reporting on Progress of Testing
- Defect Tracking

4.3.UML Diagrams

The underlying premise of UML is that no one diagram can capture the different elements of a system in its entirety. Hence, UML is made up of nine diagrams that can be used to model a system at different points of time in the software life cycle of a system. The nine UML diagrams are:

- **Use case diagram:**

  The use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with each use case.
- **Class diagram:**

  The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes.

  The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

- **Object diagram:**

  The object diagram is a special kind of class diagram. An object is an instance of a class. This essentially means that an object represents the state of a class at a given point of time while the system is running. The object diagram captures the state of different classes in the system and their relationships or associations at a given point of time.

- **State diagram:**

  A state diagram, as the name suggests, represents the different states that objects in the system undergo during their life cycle. Objects in the system change states in response to events. In addition to this, a state diagram also captures the transition of the object's state from an initial state to a final state in response to events affecting the system.

- **Activity diagram:**

  The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.

- **Sequence diagram:**

  A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

- **Collaboration diagram:**

A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.

- **Component diagram:**

  The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.

- **Deployment diagram:**

  The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

UML Diagram Classification—Static, Dynamic, and Implementation

A software system can be said to have two distinct characteristics: a structural, "static" part and a behavioral, "dynamic" part. In addition to these two characteristics, an additional characteristic that a software system possesses is related to implementation. Before we categorize UML diagrams into each of these three characteristics, let us take a quick look at exactly what these characteristics are.

- **Static:** The static characteristic of a system is essentially the structural aspect of the system. The static characteristics define what parts the system is made up of.
- **Dynamic:** The behavioral features of a system; for example, the ways a system behaves in response to certain events or actions are the dynamic characteristics of a system.

- **Implementation:** The implementation characteristic of a system is an entirely new feature that describes the different elements required for deploying a system.

The UML diagrams that fall under each of these categories are:

- Static
    - Use case diagram
    - Class diagram

  Dynamic
    - Object diagram
    - State diagram
    - Activity diagram
    - Sequence diagram
    - Collaboration diagram

  Implementation
    - Component diagram
    - Deployment diagram

Finally, let us take a look at the 4+1 view of UML diagrams.

View of UML Diagrams

These different views are:

- **Design View:**

  The *design view* of a system is the structural view of the system. This gives an idea of what a given system is made up of. Class diagrams and object diagrams form the design view of the system.
- **Process View:**

  The dynamic behavior of a system can be seen using the *process view*. The different diagrams such as the state diagram, activity diagram, sequence diagram, and collaboration diagram are used in this view.

- **Component View:**

  Next, you have the *component view* that shows the grouped modules of a given system modeled using the component diagram.

- **Deployment View:**

  The deployment diagram of UML is used to identify the deployment modules for a given system. This is the *deployment view* of the

- **Use case View:**

  Finally, we have the *use case view*. Use case diagrams of UML are used to view a system from this perspective as a set of discrete activities or transactions.

4.4. What is Test Management?

The aim of this process is to prioritize tasks that are imperative and to analyze who is best suited for which task. Moreover, with its assistance one can allocate resources accordingly and forecast risks involved in testing. An adept tester is expected to apply his/her ingenuity and try applying the most suitable resource required at the right place. Additionally, the process of test management can also be carried out with the assistance of test management tools that allow automatic generation of **requirement traceability matrix (RTM)**, which signify the functional coverage of the application under test (AUT).

Roles and Responsibilities of Test Management:

1. **Strategic Management:** One of the most important role of test management to strategize the whole testing process by defining its goals,

initiatives, analyzing the competitive environment, identifying benchmarks, managing financial as well as human resources, and more. Moreover, it helps define the metrics for measuring the performance of the software and allows merging of test strategies with client's requirements and needs.

2. **Operational Test Management:** This process deals with project management, managing external relationships like determining types of external relationships, addressing contractual issues, defining communication strategies, project management aspects like estimation of time and effort, performing risk assessment, test project and evaluation include tracking information, internal and external reporting , and finally evaluating the effectiveness of all the previous steps.

3. **Test Team Management:** Another significant role of test management is team management, which covers organizational level management and management at testing level. During the latter, management involves solving testing issues, assigning roles to responsible individuals, deciding tools, and other such activities. Whereas, in the former, the focus shifts from basic planning and moves to monitoring testing team in the testing environment and handling ethical issues.

Reasons for Test Management:

If we consider the roles and responsibilities of test management, we can easily conclude that the reasons for test management can be several and can vary from project to project and organization to organization. Hence, here is a list of some of the reasons for test management:

- Can be used to set-up well defined outlines, which can improve the software quality.

- To have easy access to test data for the teams working across different geographical locations.

- For managing the rapidly increasing software, along with the increment in complexities and attributes associated with the software.

- Reduces coding & designing errors.

- Ensures effective and efficient usage of test resources.

- Helps complete project within deadlines.

- Makes the task of tracking test cases, easier and simpler.

- Helps get a faster, smoother and unified testing process.

Test Management Process:

1. **Test Organization:-** The first stage of the process, test organization involves defining the roles and responsibilities of the team members, while managing test functions, test facilities, and test activities. Moreover, it addresses how teams track dependencies and relationships among various test assets, such as test cases, test scripts, test data, test software and test hardware. Here, the main emphasis is on elaborating the competencies and knowledge of the people involved, as well as organizing & maintaining an inventory of test items. This further involves two stages, which are:

    - **Independent Testing:** To avoid author bias and to get more accurate results, individual testing is performed, wherein individual members of the team perform software testing independently. This improves the quality of defect detection, which further makes the process of testing effective.

- **Tasks of Test Leader & Tester:** Before the commencement of the testing process, it is the responsibility of the team leaders, managers, and testers to perform some important tasks, which are specifically allocated to them. Some of these tasks are:

    - Leading the testing team and encouraging them to effectively perform testing.

    - Defining the scope of testing as per the release and delivery of the project.

    - Deploying and managing resources of testing.

2. **Test Planning &Estimation:-** Test planning and estimation are vital for the smooth functioning of the whole testing process. These are the documents that define various factors of testing like the techniques, methods, and tools testing, as well as the scope, schedule and resources of testing. Moreover, with the help of estimation one can determine the time and cost of testing, even before the beginning of the process. Test planning and estimation are further dependent on few factors, which are mentioned below:

    - **Test Planning:** As stated above, test planning is mainly concerned about the following components:

        - Test policy, strategy.

        - Estimating techniques.

    - Test Plan.

- **Test Approaches:** Another important factor that is considered while preparing the test planning and estimation document is various test approaches, which are decided on the basis of the testing requirement and other crucial elements like:

    - Analytical.
    - Model Based.
    - Methodological.
    - Process Compliant or Standard Compliant.
    - Dynamic & Heuristic.
    - Regression Averse.
    - Consultative.

- **Entry & Exit Criteria:** The entry and exit criteria helps streamline the entire testing life cycle and improves the quality of the software. These are usually determined with the assistance of system testing, acceptance testing, integration testing, and unit testing. Moreover, the entry and exit criteria can be utilized to analyze the outcomes of the test strategy.

    - **Entry Criteria:** Specify the conditions or on-going activities that need to be present during each testing phase, such as test data, test plans, testing tools, testable codes, and more.

        - **Exit Criteria:** These are the activities or requirements that need to be fulfilled before the culmination of the testing process. These include identifying & resolving defects, ensuring all critical test cases are passed, retesting and closing all the high priority defects, among other things.

3. **Test Progress Monitoring and Control:** To ensure proper testing and functioning of the whole testing process, test progress monitoring and controlling are extremely necessary. These allow the team leads to monitor and evaluate the testing activities and provide a feedback, which can further help with the improvement and correction process.

    - **Test Monitoring:** The major objective of test monitoring during the process of test management is to offer visibility into the testing activities and to give a credible feedback, which can be collected either manually or automatically and then used for measuring the exit criteria.

    - **Test Reporting & Control:**

        - **Test Control:** Refers to any guiding and improvement actions taken in regard to the gathered and recorded information. These make decision based on the metrics of test monitoring and and can affect the testing activities.

        - **Test Reporting:** The focus here is to summarize the whole testing procedure including the details about detected defects and the ways of tackling the various issues. This is mainly used to determine the progress of testing.

    - **Test Summary Report:** Created after the commencement of test monitoring and controlling, this consists of all the necessary details about the testing process. From the tested metrics to the ways of testing, every minute detail about the process is recorded here.

4. **Risks & Testing:** Determines the possible risks of testing and the various approaches used to get the required results. All aspects of testing are considered like the testing approaches, methods, tools and more to ensure the probability of the defects or any other risk is minimum.

    - **Risk Probability/Likelihood & Impact:** To ensure minimum probability and impact of various testing risks, the risk management is implemented by the management team. Here, the team considers all the possible risks and take precautions to tackle them.

    - **Project & Product Risks:** To ensure the success of the software product, project and product risks are identify, understood and mitigated. Delays in testing and fixing of bugs, unavailability of test environment, lack of system domain, and other software and system risks, which can impact the quality of the product and user experience.

    - **Risks Based Testing Approach:** Here, the team decides on various approaches that can simplify the process of testing as well as reduced the probability of risk in the system. This includes risk analysis, response planning, scope changes, reviewing documents, risk data quality assessment, etc.

5. **Configuration Management:** This establishes and maintains the integrity of the products and ensures that all the items of testware are

identified, version controlled, tracked for changes, and related to each other. The main objective of this stage of test management process is to verify that system is performing as it is intended to and is documenting functional capabilities and interdependencies.

6. **Incident Management:** The last stage of test management process, incident management is the process of identifying, analyzing and correcting hazards by the organization to prevent future occurrence of a similar problem. These if not tackled, can lead to a huge disaster and can disrupt business operations.

    - **Incident Management & Logging:** The aim here is to log and record all the incidents that can impact business operations as well as prohibit users from getting optimum services. Therefore, every incident is logged along with exact date and time of occurence, a small description, and more.

    - **Test Incident Report:** Here all the deviations and variations observed in the software behaviour are reported. Furthermore, all the defects and bugs found during the process of testing are also included with details about their severity, priority, ways of resolving them, and more.

Test Management Tools:

- **qTest:** The best test management tool used by agile testing team, qTest provides easy to learn, easy to use, and scalable test management solutions, which allow easy centralization, organization, and helps in making the process of test management speedy.

- **PractiTest:** A flexible and innovative test management tool that offer impeccable customization and seamless integration. Visualize your data using the most advanced dashboards and reports.

- **QAComplete:** This is a comprehensive test management tool with enterprise level capabilities. It is extremely easy to use and offers ultimate flexibility to revolutionize your team quality assurance strategies.

- **Meliora Testlab:** A powerful test management tool with ALM features. This tool offers excellent assistance during software designing and testing phase and is extremely easy to use. With Meliora TestLab plan, track, and proactively manage manual, automated, and API tests in one repository to mitigate risks.

4.5.Cloud computing

The term cloud refers to a network or the internet. It is a technology that uses remote servers on the internet to store, manage, and access data online rather than local drives. The data can be anything such as files, images, documents, audio, video, and more.

There are the following operations that we can do using cloud computing:

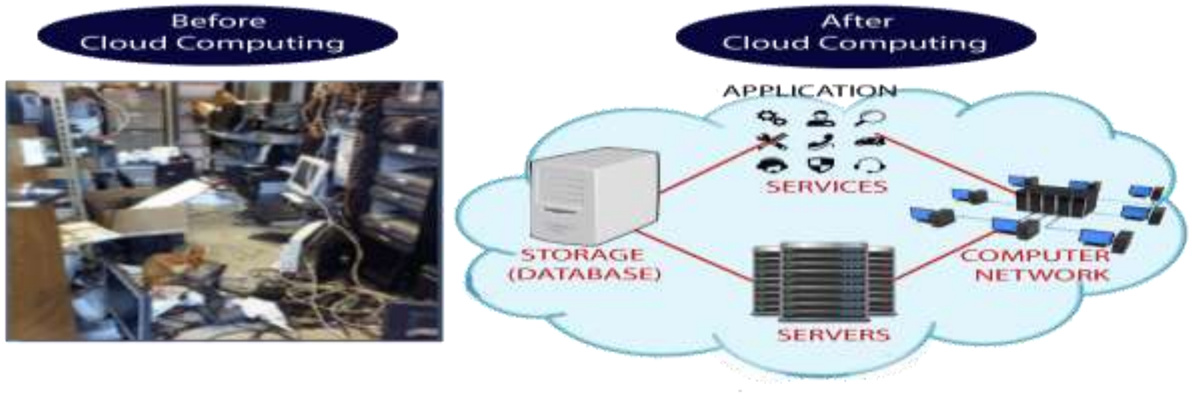
- Developing new applications and services
- Storage, back up, and recovery of data
- Hosting blogs and websites
- Delivery of software on demand
- Analysis of data
- Streaming videos and audios

Why Cloud Computing?

Small as well as large IT companies, follow the traditional methods to provide the IT infrastructure. That means **for any IT company, we need a Server Room that is the basic need of IT companies**.

In that server room, there should be a database server, mail server, networking, firewalls, routers, modem, switches, QPS (Query Per Second means how much queries or load will be handled by the server), configurable system, high net speed, and the maintenance engineers.

To establish such IT infrastructure, we need to spend lots of money. To overcome all these problems and to reduce the IT infrastructure cost, Cloud Computing comes into existence.



Characteristics of Cloud Computing

The characteristics of cloud computing are given below:

1) Agility

The cloud **works in a distributed computing environment**. It shares resources among users and works very fast.

2) High availability and reliability

The availability of servers is high and more reliable because the **chances of infrastructure failure are minimum**.

3) High Scalability

Cloud offers **"on-demand" provisioning of resources on a large scale**, without having engineers for peak loads.

4) Multi-Sharing

With the help of cloud computing, **multiple users and applications can work more efficiently** with cost reductions by sharing common infrastructure.

5) Device and Location Independence

Cloud computing enables the users to access systems using a web browser regardless of their location or what device they use e.g. PC, mobile phone, etc. **As infrastructure is off-site** (typically provided by a third-party) **and accessed via the Internet, users can connect from anywhere**.

6) Maintenance

Maintenance of cloud computing applications is easier, since they **do not need to be installed on each user's computer and can be accessed from different places**. So, it reduces the cost also.

7) Low Cost

By using cloud computing, the cost will be reduced because to take the services of cloud computing, **IT company need not to set its own infrastructure** and pay-as-per usage of resources.

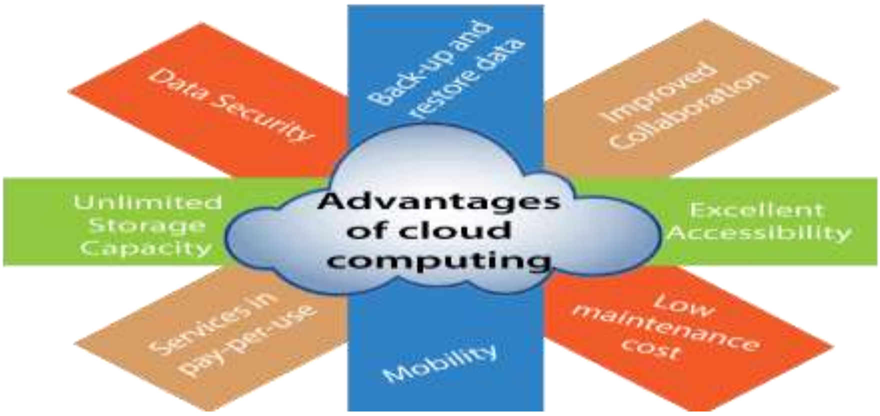
8) Services in the pay-per-use mode

Application Programming Interfaces **(APIs) are provided to the users so that they can access services on the cloud** by using these APIs **and pay the charges as per the usage of services**.

Advantages and Disadvantages of Cloud Computing

Advantages of Cloud Computing

Here, we are going to discuss some important advantages of Cloud Computing-



1) Back-up and restore data

Once the data is stored in the cloud, it is easier to get back-up and restore that data using the cloud.

2) Improved collaboration

Cloud applications improve collaboration by allowing groups of people to quickly and easily share information in the cloud via shared storage.

3) Excellent accessibility

Cloud allows us to quickly and easily access store information anywhere, anytime in the whole world, using an internet connection. An internet cloud infrastructure increases organization productivity and efficiency by ensuring that our data is always accessible.

4) Low maintenance cost

Cloud computing reduces both hardware and software maintenance costs for organizations.

5) Mobility

Cloud computing allows us to easily access all cloud data via mobile.

6) IServices in the pay-per-use model

Cloud computing offers Application Programming Interfaces (APIs) to the users for access services on the cloud and pays the charges as per the usage of service.

7) Unlimited storage capacity

Cloud offers us a huge amount of storing capacity for storing our important data such as documents, images, audio, video, etc. in one place.

8) Data security

Data security is one of the biggest advantages of cloud computing. Cloud offers many advanced features related to security and ensures that data is securely stored and handled.

Disadvantages of Cloud Computing

A list of the disadvantage of cloud computing is given below -

1) Internet Connectivity

cloud computing, every data (image, audio, video, etc.) is stored on the cloud, and we access these data through the cloud by using the internet connection. If you do not have good internet connectivity, you cannot access these data. However, we have no any other way to access data from the cloud.

2) Vendor lock-in

Vendor lock-in is the biggest disadvantage of cloud computing. Organizations may face problems when transferring their services from one vendor to another. As different vendors provide different platforms, that can cause difficulty moving from one cloud to another.

3) Limited Control

cloud infrastructure is completely owned, managed, and monitored by the service provider, so the cloud users have less control over the function and execution of services within a cloud infrastructure.

4) Security

Although cloud service providers implement the best security standards to store important information. But, before adopting cloud technology, you should be aware that you will be sending all your organization's sensitive information to a third party, i.e., a cloud computing service provider. While sending the data on the cloud, there may be a chance that your organization's information is hacked by Hackers.

UNIT – V

TEST AUTOMATION & QUALITY METRICS

5. Test Automation:

- Software Test automation makes use of specialized tools to control the execution of tests and compares the actual results against the expected result. Usually, regression tests, which are repetitive actions, are automated.
- Testing Tools not only helps us to perform regression tests but also helps us to automate data set up generation, product installation, GUI interaction, defect logging, etc. Automation tools are used for both Functional and Non-Functional testing.

Criteria for Tool Selection:

For automating any application, the following parameters should be considered:

- Data driven capabilities
- Debugging and logging capabilities
- Platform independence
- Extensibility & Customizability
- E-mail Notifications
- Version control friendly
- Support unattended test runs

Types of Frameworks:

Typically, there are 4 test automation frameworks that are adopted while automating the applications:

- Data Driven Automation Framework
- Keyword Driven Automation Framework
- Modular Automation Framework
- Hybrid Automation Framework

5.1 Advantages of Automated Testing:

Automated Testing has the following advantages:

1. Automated testing improves the coverage of testing as automated execution of test cases is faster than manual execution.
2. Automated testing reduces the dependability of testing on the availability of the test engineers.
3. Automated testing provides round the clock coverage as automated tests can be run all time in 24*7 environment.
4. Automated testing takes far less resources in execution as compared to manual testing.
5. It helps to train the test engineers to increase their knowledge by producing a repository of different tests.
6. It helps in testing which is not possible without automation such as reliability testing, stress testing, load and performance testing.
7. It includes all other activities like selecting the right product build, generating the right test data and analyzing the results.
8. It acts as test data generator and produces maximum test data to cover a large number of input and expected output for result comparison.
9. Automated testing has less chances of error hence more reliable.

10. As with automated testing test engineers have free time and can focus on other creative tasks.

Disadvantages of Automated Testing :

Automated Testing has the following disadvantages:

1. Automated testing is very much expensive than the manual testing.
2. It also becomes inconvenient and burdensome as to decide who would automate and who would train.
3. It has limited to some organisations as many organisations not prefer test automation.
4. Automated testing would also require additionally trained and skilled people.
5. Automated testing only removes the mechanical execution of testing process, but creation of test cases still required testing professionals.

5.2 Activities in test Automation

Following steps are followed in an Automation Process

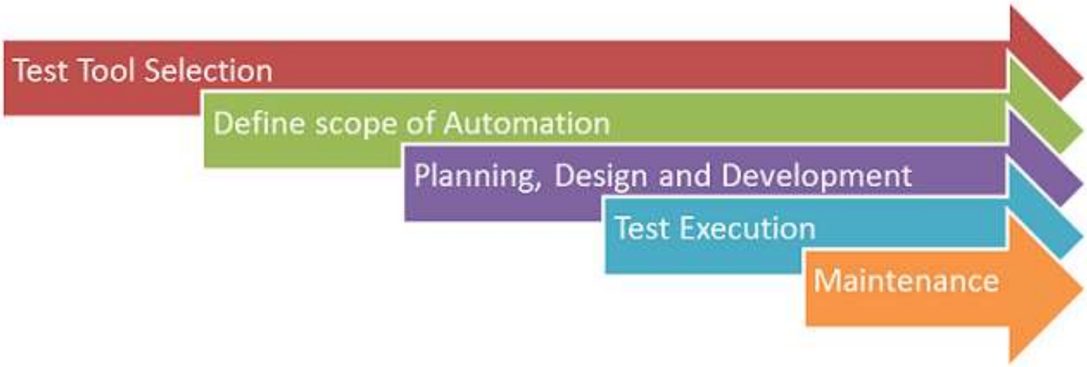
Step 1) Test Tool Selection

Step 2) Define scope of Automation

Step 3) Planning, Design and Development

Step 4) Test Execution

Step 5) Maintenance



5.3 Automation Framework

- Automation Framework is not a single tool or process, but it is a collection of tools and processes working together to support automated testing of any application.

- It integrates various functions like libraries, test data, and various reusable modules.

Importance of Automation Framework in Software Testing

- Framework supports automation testing as a technical implementation guideline.
- For example, consider a scenario where a testing team includes members who are based on different automation testing code. And, they are not able to grasp the common pieces of code and scripts updated by a team member in a project.
- The automation framework not only offers the benefit of reusing the code in various scenarios, but it also helps the team to write down the test script in a standard format.
- Hence, the test automation framework handles all the issues.
- Besides, there are many other benefits of using automation framework testing as listed below:

- Maintain a well-defined strategy across the test suites
- Enhanced speed at which testing progresses
- Maintaining the test code will be easy
- The URL or Application can be tested accurately
- Continuous testing of coding and delivery will be achieved
- Test automation framework is helpful when you need to execute the same test scripts multiple times with different builds to examine the application and validate output.

- ➢ It is better to avoid automated testing for functionality, which you used only once since building automation script itself is time-consuming.

Test Automation Framework Types

Each automation framework has its own architecture, advantages, and disadvantages. Some of these frameworks are:

- Linear Automation Framework
- Modular Driven Framework
- Behavior Driven framework
- Data-Driven Framework
- Keyword-Driven Framework
- Hybrid Testing Framework

1) Linear Automation Framework

The linear Automation framework is commonly used in the testing of small applications. This framework is also called as a Record and playback framework.

2) Modular Driven Framework

In this Framework, the tester can create test scripts module wise by breaking down the whole application into smaller modules as per the client requirements and create test scripts individually.

3) Behavior Driven Development Framework

Behavior Driven Development framework is to create a platform, which allows every person, like Developers, Testers, business analyst, etc., to participate actively. It also increases collaboration between the tester and the developers on your project.

4) Data-driven Testing Framework

Generally, Test Data is read from the external files like Excel Files, Text Files, CSV Files, ODBC Sources, DAO Objects and they are loaded into the variables inside the Test Script. The data-driven framework allows us to create test automation scripts by passing different sets of test data.

5) The Keyword-Driven Testing Framework

The keyword-Driven Testing framework is also known as table-driven testing. This framework is suitable only for small projects or applications. The automation test scripts performed are based on the keywords specified in the excel sheet of the project.

6) The Hybrid test Automation Framework

Hybrid Framework is used to combine the benefits of Keyword Driven and Data-Driven frameworks.

5.4 Tools for Test Automation

Success in any test automation depends on identifying the right tool for the project. With a plethora of open-source and commercial automation tools to choose from, automation Testing tool selection can become tricky. Here is a curated list of top automated tools –

1) ACCELQ

ACCELQ is the only cloud-based codeless test automation platform that seamlessly Automates API and web testing, achieving continuous testing for Enterprises.

2) TestProject

TestProject is the world's first free cloud-based, community-powered test automation platform that enables users to test Web, Android and iOS applications on all operating systems, effortlessly. Easily collaborate with your team using Selenium and Appium to ensure quality with speed.

3) Ranorex

Over 14,000 users worldwide accelerate testing with Ranorex Studio, an all-in-one tool for test automation. Ranorex is easy for beginners with a codeless click-and-go interface, but powerful for automation experts with a full IDE for C# or VB.NET, and open APIs.

4) TestCraft

TestCraft is a codeless Selenium test automation platform. The revolutionary AI technology and unique visual modeling allows for faster test creation and execution while eliminating test maintenance overhead. Testers create fully automated test scenarios without coding. Customers find bugs faster, release more frequently, integrate with CI/CD and improve overall quality of their digital products.

5) Subject7

Subject7 is a cloud-based, no-code platform supporting end-to-end automation for web, mobile, desktop, database, web services, load, security, and accessibility testing. The interface enables non-coders to author robust test flows, with minimal training/support. Customers include major government agencies, and enterprises of all sizes.

6) ZeuZ

ZeuZ Automation is an AI-assisted click-and-test automation framework that testers enjoy using! It's scriptless and simple for manual testers, but also robust, with all-in-one features desired by experts. Equipped with CI/CD integration, intelligent debugging, rich reporting, collaboration features, ZeuZ delivers on the promise of true end-to-end automation.

5.5 Script Languages In Test Automation.

- A **TEST SCRIPT** is a set of instructions that is performed on a system under test to verify that the system performs as expected.
- They can be written in either a human language (used for manual testing) or a scripting / programming language (used for automated testing).
- A test script is a part of a test case and a test case can have either a single test script or multiple test scripts.
- Multiple test scripts are associated with a single test case when:

- Each test script provides a different way to test the scenario in the test case.

- For instance, a test case might require several scripts to test the scenario in different test environments.

- Both manual and automated scripts exist to exercise the test case.

ISTQB Definition

- **Test script:** A sequence of instructions for the execution of a test.

- Some scripting languages used in automated testing are:

- JavaScript
- Perl
- Python
- Ruby
- Tcl
- Unix Shell Script
- VBScript

  - There are also many Test Automation Tools/Frameworks that generate the test scripts for you; without the need for actual coding.
  - Many of these tools have their own scripting languages (some of them based on a core scripting language)

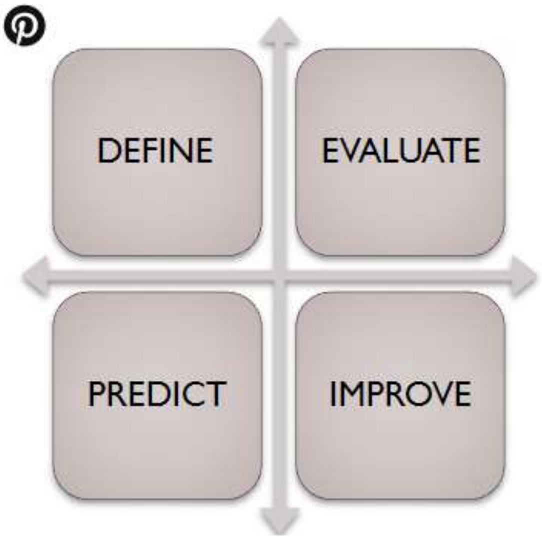
5.6 Metrics for Software Quality

- The word '**metrics**' refer to standards for measurements. Software Quality Metrics means measurement of attributes, pertaining to software quality along with its process of development.

- The term **"software quality metrics"** illustrate the picture of measuring the software qualities by recording the number of defects or security loopholes present in the software.

- However, quality measurement is not restricted to counting of defects or vulnerabilities but also covers other aspects of the qualities such as maintainability, reliability, integrity, usability, customer satisfaction, etc.

Why Software Quality Metrics?



1. To define and categorize elements in order to have better understanding of each and every process and attribute.

2. To evaluate and assess each of these process and attribute against the given requirements and specifications.

3. Predicting and planning the next move w.r.t software and business requirements.

4. Improving the Overall quality of the process and product, and subsequently of project.

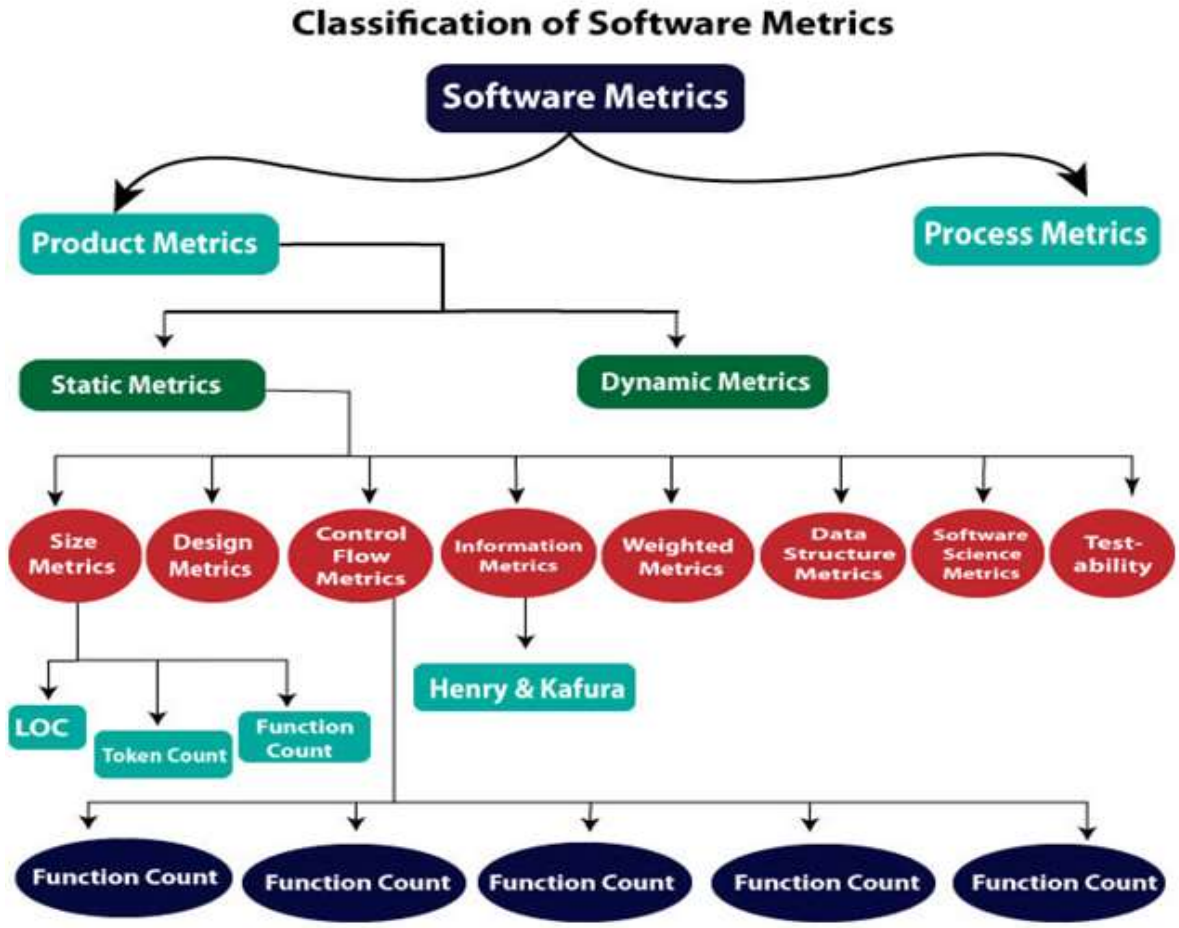
5.7 Classification of Software Metrics

- **Software metrics can be classified into two types as follows:**

1. Product Metrics: These are the measures of various characteristics of the software product. The two important software characteristics are:

1. Size and complexity of software.
2. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.



2. Process Metrics: These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.

5.7 Types of Metrics

1.Internal metrics:

- Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer.
- For example, Lines of Code (LOC) measure.

2.External metrics:

- External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user,

- e.g., portability, reliability, functionality, usability, etc.

3.Hybrid metrics:

- Hybrid metrics are the metrics that combine product, process, and resource metrics.
- For example, cost per FP where FP stands for Function Point Metric.

4. Project metrics:

- Project metrics are the metrics used by the project manager to check the project's progress.
- Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software.
- Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time

5. Process Metrics:

- It is used to improve the efficiency of the process in the SDLC (Software Development Life Cycle).

6. **Product Metrics:**

- It is used to tackle the quality of the software product.

5.9 Basic Quality Control Tool

The **7 basic quality tools** are as follows:

1. **Flow Chart**
2. **Histogram**
3. **Cause-and-Effect Diagram**
4. **Check Sheet**
5. **Scatter Diagram**
6. **Control Charts**
7. **Pareto Charts**

1.Flow charts:

- Flow charts are one of the best process improvement tools you can use to analyze a series of events.

- They map out these events to illustrate a complex process in order to find any commonalities among the events.

- Flow charts can be used in any field to break down complex processes in a way that is easy to understand.

- Then, you can go through the business processes one by one, identifying areas for improvement.

2.Histogram:

- A histogram is a chart with different columns. These columns represent the distribution by the mean.

- If the histogram is normal then the graph will have a bell-shaped curve.

- If it is abnormal, it can take different shapes based on the condition of the distribution.

- Histograms are used to measure one thing against another and should always have a minimum of two variables.

3.Cause-and-effect Diagram (also known as Fishbone diagram):

- Cause-and-effect diagrams can be used to understand the root causes of business problems.
- Because businesses face problems daily, it is necessary to understand the root of the problem so you can solve it effectively.

4.Check Sheet:

- A check sheet is a basic tool that gathers and organizes data to evaluate quality.
- This can be done with an Excel spreadsheet so you can analyze the information gathered in a graph.

5.Scatter Diagram:

- Scatter diagrams are the best way to represent the value of two different variables.
- They present the relationship between the different variables and illustrate the results on a Cartesian plane.
- Then further analysis can be done on the values.

6.Control Charts:

- A control chart is a good tool for monitoring performance and can be used to monitor any process that relates to the function of an organization.
- These charts allow you to identify the stability and predictability of the process and identify common causes of variation.

7.Pareto Charts:

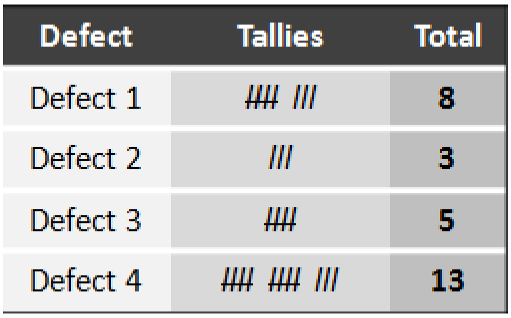
- Pareto charts are charts that contain bars and a line graph.
- The values are shown in descending order by bars and the total is represented by the line.
- They can be used to identify a set of priorities so you can determine what parameters have the biggest impact on the specific area of concern.

5.10 Check sheet

- Check sheets are manual data collection forms used to collect data in real time at the location where the data is generated.
- They can be used on a temporary basis (for example during a project) or be established for routine activities.
- Check sheets are often used for collecting failure information at specific process steps.
- These are often referred to as **Failure Check Sheets**. There are also the **Visual Check Sheets** (or the **Measles Charts**) that use pictures of the process or product to record where an event occurred.
- These are also known as a defect maps or a measles charts.
- A **Traveler Check Sheet** stays with the product or service throughout the entire process, collecting information at each stage.
- It is often used when collecting process lead times.
- Other common forms of check sheets are: **Tally Charts** and **Checklists**.

Tally Charts:

- A Tally Chart is an easy and helpful way to track and record data.
- It is a table that records the frequency with which different events are observed.
- The collected data is quickly understood as it is displayed in an easy-to-count groups of five.



Defect	Tallies	Total
Defect 1	𝍸 |||	8
Defect 2	|||	3
Defect 3	𝍸	5
Defect 4	𝍸 𝍸 |||	13

Checklists:

- A checklist is a basic check sheet with even more structure. It is simply a list of tasks to be performed and includes small check boxes next to each task.
- Listed items are ticked off as they are executed or when they are available.
- One of the benefits of checklists is that they compensate for the limitation of human memory and attention.

5.11 Cause And Effect Diagram

- A cause and effect diagram is a tool that helps you do this. The ‘effect’ is the problem you are working on, for example ‘waiting time’.
- The tool can help you identify major causes and indicate the most fruitful areas for further investigation. It will help you understand the problem more clearly.
- By going through the process of building the diagram with colleagues, everybody gains insights into the problem, alongside possible solutions.
- The people involved benefit from shared contributions, leading to a common understanding of the problem.
- The cause and effect diagram is sometimes called a fishbone diagram (because the diagram looks like the skeleton of a fish) or an Ishikawa diagram

How to use it

- Firstly, identify the problem. Write it in a box and draw an arrow pointing towards it. Think about the exact problem in detail. Where appropriate, identify who is involved, what the problem is, and when and where it occurs.

- Identify the major factors and draw four or more branches off the large arrow to represent main categories of potential causes. Categories could include: equipment, environment, procedures, and people. Make sure that the categories you use are relevant to your particular problem / delay.
- Take each of the main categories and brainstorm possible causes of the problem. Then, explore each one to identify more specific 'causes of causes'. Continue branching off until every possible cause has been identified. Where a cause is complex, you might break it down into sub-causes. Show these as lines coming off each cause line.

- Analyse your diagram. By this stage you should have a diagram showing all the possible causes of your delay / problem. Depending on the complexity and importance of the problem, you can now investigate the most likely causes further. This may involve setting up interviews (see getting patient perspectives), carrying out process mapping or surveys which you can use to decide whether the causes identified are correct.

5.12 Pareto Chart

- ➢ A Pareto chart is essentially a bar chart showing how often different categories of events/incidents take place.

- ➢ The most common is ordered to the left, and the least common is ordered to the right. It also includes a line showing the cumulative frequency (adding each column's value to the previous ones in turn).

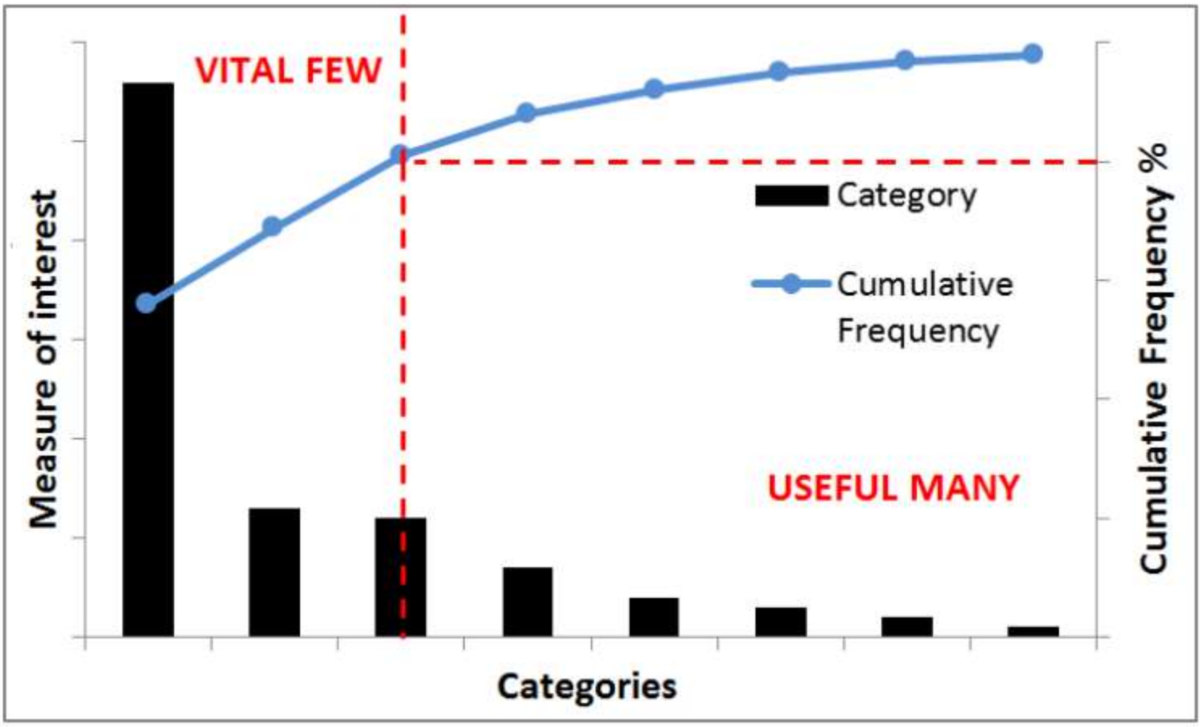
- ➢ A Pareto chart follows the 80/20 principle developed by Vilfredo Pareto, an Italian economist in the early 20th century.

- The 80/20 principle asserts that for many events, roughly 80% of the effects come from 20% of the causes.

- This then allows you to focus on where improvement projects are needed most and will have the biggest impact.

Characteristics of a Pareto chart

- On the X axis you have the area of interest (categories) e.g. as ward names.
- On the left Y axis you have the number of events e.g. number of falls.
- On the right Y axis you have the cumulative frequency.
- Essentially, a Pareto chart is a bar and line graph combined. The bars display the number of events per area of interest whilst the line displays the cumulative % of events.
- Categories contributing to 80% of the problems are often referred to as the 'vital few' whereas the others are labelled the 'useful many'.



Key points about a Pareto chart

- They work best with 30 observations across the categories. Smalls numbers of data can be misleading due to random change.
- The classification by categories is only a guide to where to focus attention of your improvement project.
- Identifying categories where 80% of the problems are occurring is not the only thing to consider when deciding where to focus improvement efforts.
- Pareto charts don't explain what sort of variation is being observed. To distinguish random (common cause) and non-random (special cause) variation you need to use either a run chart or control charts.

5.13 Histogram

- A histogram is a graphical representation of data.
- The data is represented by columns on a graph that vary in height depending on the frequency (how many times) the specific range of data occur.

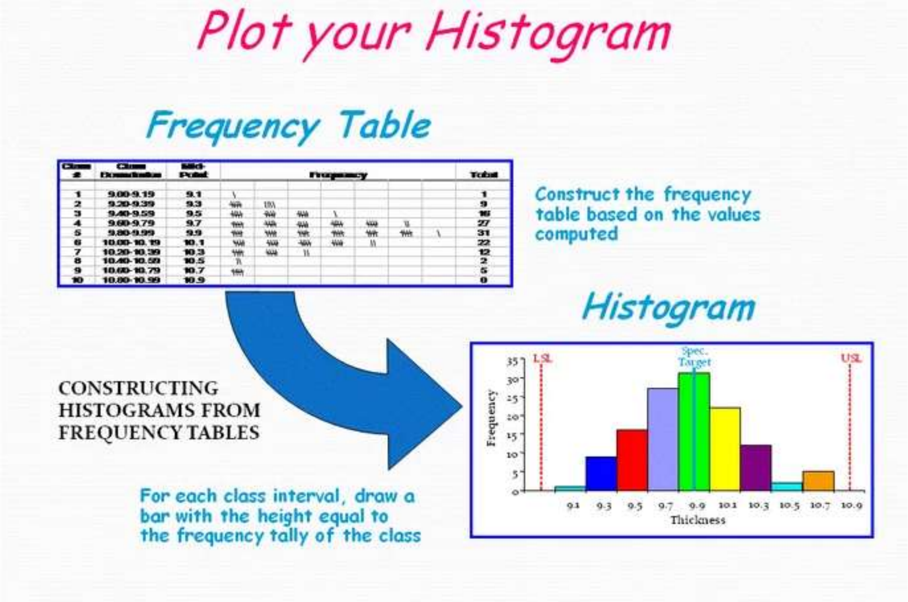
Use A Histogram As A Quality Tool

1. Displays data in an easy-to-interpret graphical manner
2. Shows frequency of occurrence of data values
3. Reveals the centering, variation and shape of the data
4. Illustrates the underlying distribution of the data
5. Enables future prediction of process performance
6. Enables identification in changes in processes parameters
7. Allows you to answer the question: "Is the process capable of meeting the customer requirements?"

Make a Histogram

- The first thing to do is to collect your data.
- We can collect data using a tally chart, recording occurrences of specific ranges of measurement or we can just create a table of results when we take the measurements.
- To use this quality tool we must draw the histogram, for this we need to know the number of “class intervals” (number of columns) and the “interval width” (the width of each column on our bar chart).

Plotting Your Histogram



Plot your Histogram

Frequency Table

Class #	Class Boundaries	Mid-Point	Frequency	Total
1	9.00-9.19	9.1	|	1
2	9.20-9.39	9.3	卌 ||||	9
3	9.40-9.59	9.5	卌 卌 卌 |	16
4	9.60-9.79	9.7	卌 卌 卌 卌 卌 ||	27
5	9.80-9.99	9.9	卌 卌 卌 卌 卌 卌 |	31
6	10.00-10.19	10.1	卌 卌 卌 卌 ||	22
7	10.20-10.39	10.3	卌 卌 ||	12
8	10.40-10.59	10.5	||	2
9	10.60-10.79	10.7	卌	5
10	10.80-10.99	10.9		0

Construct the frequency table based on the values computed

Histogram

CONSTRUCTING HISTOGRAMS FROM FREQUENCY TABLES

For each class interval, draw a bar with the height equal to the frequency tally of the class

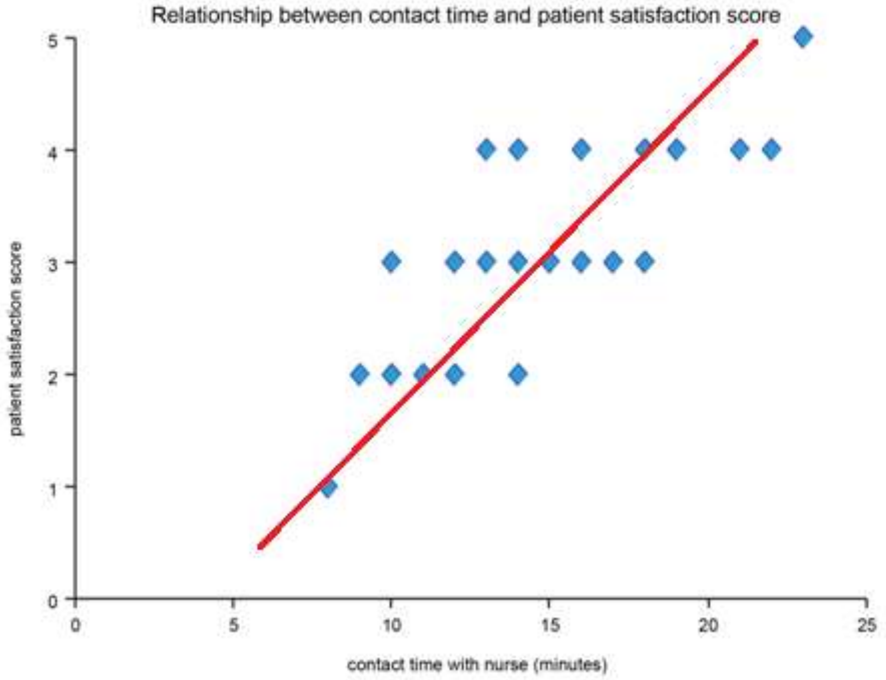
Class Intervals on Your Bar Chart

- To define the number of class intervals, the “official” method is to take the square root of the total number of measurements, for example if you have 400 measurements then the class interval will be 20.
- However, if you are not too comfortable with square roots the following table can be used as a simple guide.
- Number of Samples Class Intervals

- Under 50: 5–7
- 50–100: 6 - 10
- 100–250: 7–12
- Over 250: 10–20

5.14 Scatter Plot

- A scatter plot is a graph used to look for relationships between two variables Scatter plots show the relationship between the two variables in pairs of observations.
- One variable is plotted on the horizontal axis (usually the one that you are trying to control) and the other on the vertical axis (usually the one you expect to respond to the changes you are making).



- If the vertical variable increases as the horizontal one does (as the example above shows) then we say there is a positive correlation.

- This may indicate cause and effect but it may not be that simple. If the vertical variable decreases as the horizontal one increases we say there is negative correlation.

- ➢ This may also suggest a cause and effect relationship for further investigation.

- ➢ If the dots are scattered all over the graph then there is no evidence for a relationship between the variables.

5.15 Run chart

- ➢ Run charts are one of the most useful tools in quality improvement. They allow us to:

- Monitor the performance of one or more processes over time to detect trends, shifts or cycles.
- Compare a performance measure before and after implementation of a solution to measure its impact.
- Focus attention on truly vital changes in the process.
- Assess whether improved performance has been sustained.

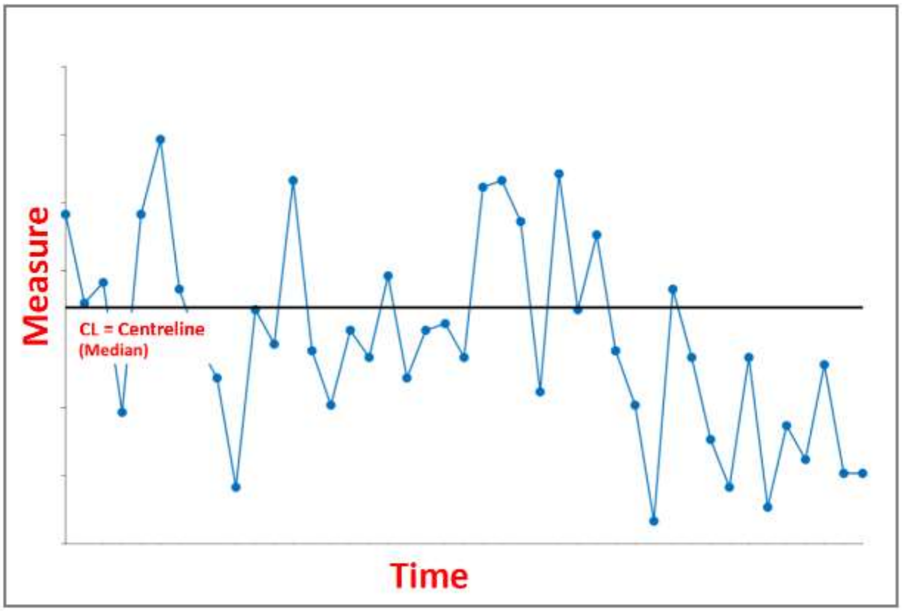
- ➢ Run charts are a valuable tool at the beginning of a project, as it reveals important information about a process before you have collected enough data to create a Stewhart control chart.

Characteristics of a run chart

- On the X axis you have data in some sort of chronological order e.g. Jan, Feb, Mar
- On the Y axis you have the measure of interest e.g. %, count
- Once the data points are connected you put a centre line (CL) between the graph. For a run chart the CL is called the **Median**.

*The **median** is the number in the middle of the data set when the data are reordered from the highest to the lowest value. If the number of observations is even, the median is the average of the two middle values.*

- ➢ A Typical run chart looks like this:



How to create a run chart

Step 1 – State the question that the run chart will answer and obtain data necessary to answer this question.

For example, if you were looking at how long it takes to travel to work in the morning you will make note of the time taken (in minutes) to get to work over a period of a month.

Step 2 – Gather data, generally collect 10-12 data points to detect meaningful patterns.

Step 3 – Create a graph with vertical line (y axis) and a horizontal line (x axis).

- On the vertical line (y axis), draw the scale related to the variable you are measuring.

Please note: it is good practice to ensure the y axis covers the full range of the measurements and then some (e.g. 1 ½ times the range of data). This is to ensure the chart can accommodate any future results.

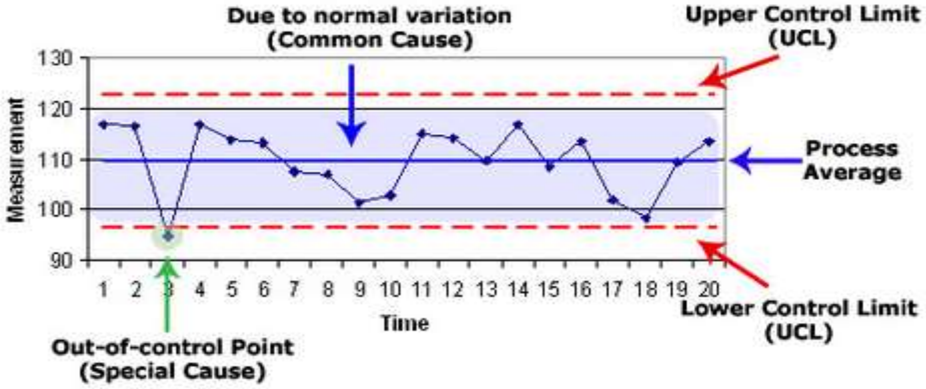
- On the horizontal line (x axis), draw the time or sequence scale.

Step 4 – Plot the data, calculate the median and include into the graph.

Step 5 – Interpret the chart. Four simple rules can be used to distinguish between random and non-random variations

5.16 Control charts

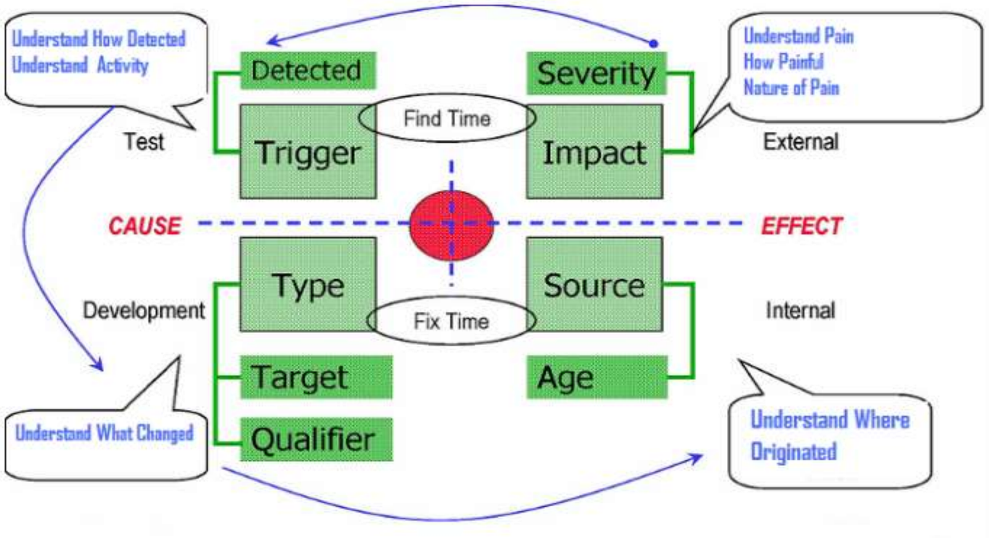
- Control charts have long been used in manufacturing, stock trading algorithms, and process improvement methodologies like Six Sigma and Total Quality Management (TQM).
- The purpose of a control chart is to set upper and lower bounds of acceptable performance given normal variation.
- In other words, they provide a great way to monitor any sort of process you have in place so you can learn how to improve your poor performance and continue with your successes.
- The control chart serves to "sound the alarm" when a process shifts (for instance, a machine suddenly breaking on a factory floor) or if someone has a breakthrough that needs to be documented and standardized across the larger organization.
- Simply put (without taking anomalies into consideration), you'll know something needs to be fixed if you're below your lower control limit or above your upper control limit.
- See the control chart example below:



5.17 Orthogonal Defect Classification (ODC)

- ODC is a systematic framework for Software Defect Classification developed by IBM in the early 1990’s.
- ODC is a concept that enables in-process feedback to developers by extracting signatures on the development process from defects.
- ODC uses of semantic information from defects to extract cause-effect relationships in the development process.
- ODC has built-in mechanisms for Phase Escape and Root Cause Analysis.
- ODC is referred to as an “MRI” on a Software Defect.

ODC (Orthogonal Defect Classification) Builds Cause — Effect Relationships



ODC (Orthogonal Defect Classification) Values

- Provide fast & effective feedback to developers.
- Captures information from defects that occurred through development phases & field usage.
- Allows understanding of defect trends over the lifecycle phases due to consistency of defect types.
- Through multi-dimensional measurement & analysis ODC assists developers to properly manage their development processes & product quality.

ODC (Orthogonal Defect Classification) Sections

A Defect passes through TWO sections with ODC

- **Opener Section:** When you find a defect, the following attributes can be classified:
- **Activity**: This is the actual activity performed at the time of defect discovery. For example, during function test phase, an engineer might decide to do a code inspection. The phase would be function test but the activity is code inspection.
- **Trigger**: The environment or condition that had to exist for the defect to surface. What is needed to reproduce the defect? During Review and Inspection activities, choose the selection which best describes what you were thinking about when you discovered the defect. For other defects, match the description with the environment or condition which was the catalyst for the failure.

- **Impact**: For in-process defects, select the impact which you judge the defect would have had upon the customer if it had escaped to the field. For field reported defects, select the impact the failure had on the customer.
- **Closer Section:** when you know how the defect was fixed, the following attributes can be classified:
- **Target**: Represents the high level identity of the entity that was fixed.
- **Defect Type**: Represents the nature of the actual correction that was made.
- **Qualifier** *(applies to the Defect Type)*: Captures the element of either a nonexistent or wrong or irrelevant implementation.
- **Source**: Identifies the origin of the Target (i.e. Design/Code, ID, etc.) which had the defect.
- **Age**: Identifies the history of the Target (i.e. Design/Code, ID, etc.) which had the defect.