Topic 3: Software Testing, Cost Estimation, Quality Management and Configuration management

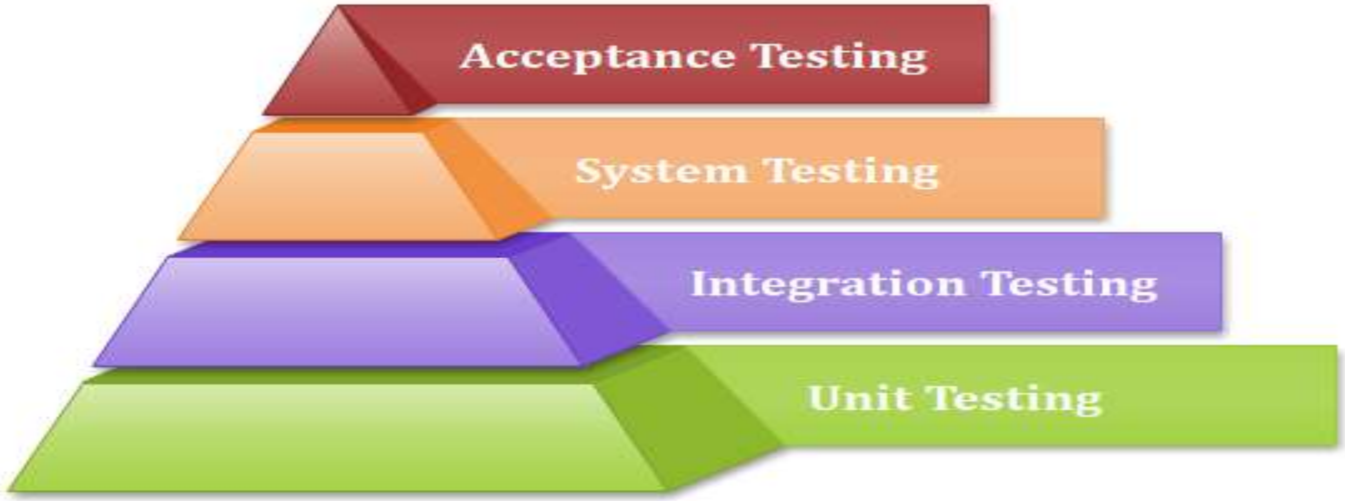
❖Unit Testing

- Individual components or functions of a software application are tested in isolation.
- Focusing on small, manageable parts of the application, unit testing helps identify and fix bugs early in the development process, significantly improving code quality and reliability.
- Verify its functionality, typically using automated testing frameworks typically automated written by developers using various frameworks/tools such as **JUnit**, **NUnit**, or **pytest.**

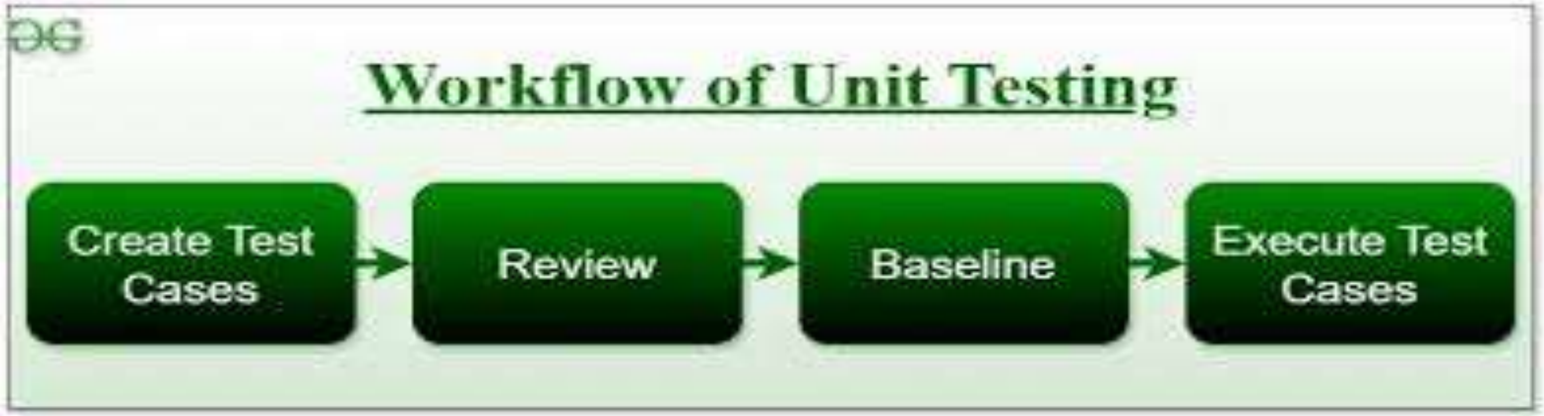
The objective of Unit Testing are follows:

- Isolate a section of code.

- Verify the correctness of the code and code reuse.
- Test every function and procedure.
- To fix bugs early and changes quickly in the development cycle and to save costs.
- In a testing level hierarchy, unit testing is the first level of testing done before integration and other remaining levels of the testing.



Workflow of unit testing



- **Unit Testing Techniques**
- There are 3 types of Unit Testing Techniques. They are follows

1. **Black Box testing:** This testing technique is used in covering the unit tests for input, user interface, and output parts.
2. **White Box Testing:** This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.

- **Gray Box Testing:** This technique is used in executing the relevant test cases, test methods, and test functions, and analyzing the code performance for the modules.

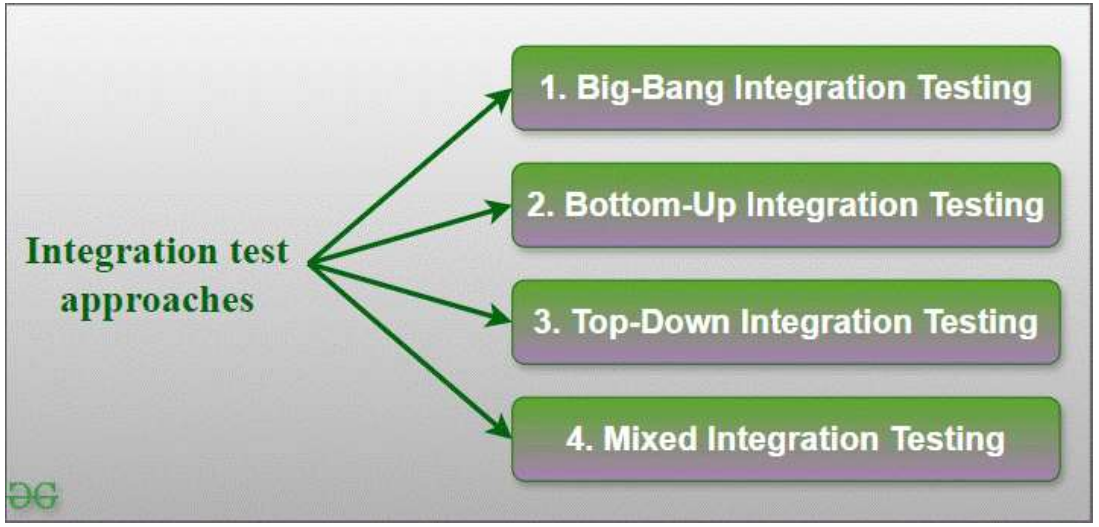
❖Integration Testing

- Verifying the interactions and data exchange between different components or modules of a software application.
- Verifies that individual software modules or components work together correctly as a whole system.
- Helps identify any compatibility or communication issues between different parts of the system.

Integration test approaches

1.Big –bang integration testing

- All the modules of the system are simply put together and tested.
- practicable only for very small systems**.**
- debugging errors reported during Big Bang integration testing is very expensive to fix.
- Used when there is low degree of interdependence between the components.


Integration test approaches
1. Big-Bang Integration Testing
2. Bottom-Up Integration Testing
3. Top-Down Integration Testing
4. Mixed Integration Testing

2.Bottom-up Integration Testing:

- Each module at lower levels are tested with higher modules until all modules are tested.
- The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem.

3.Top- down integration testing:

- High-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

4.Mixed integration Testing:

- Combination of top down and bottom-up testing approaches.

❖ System Testing

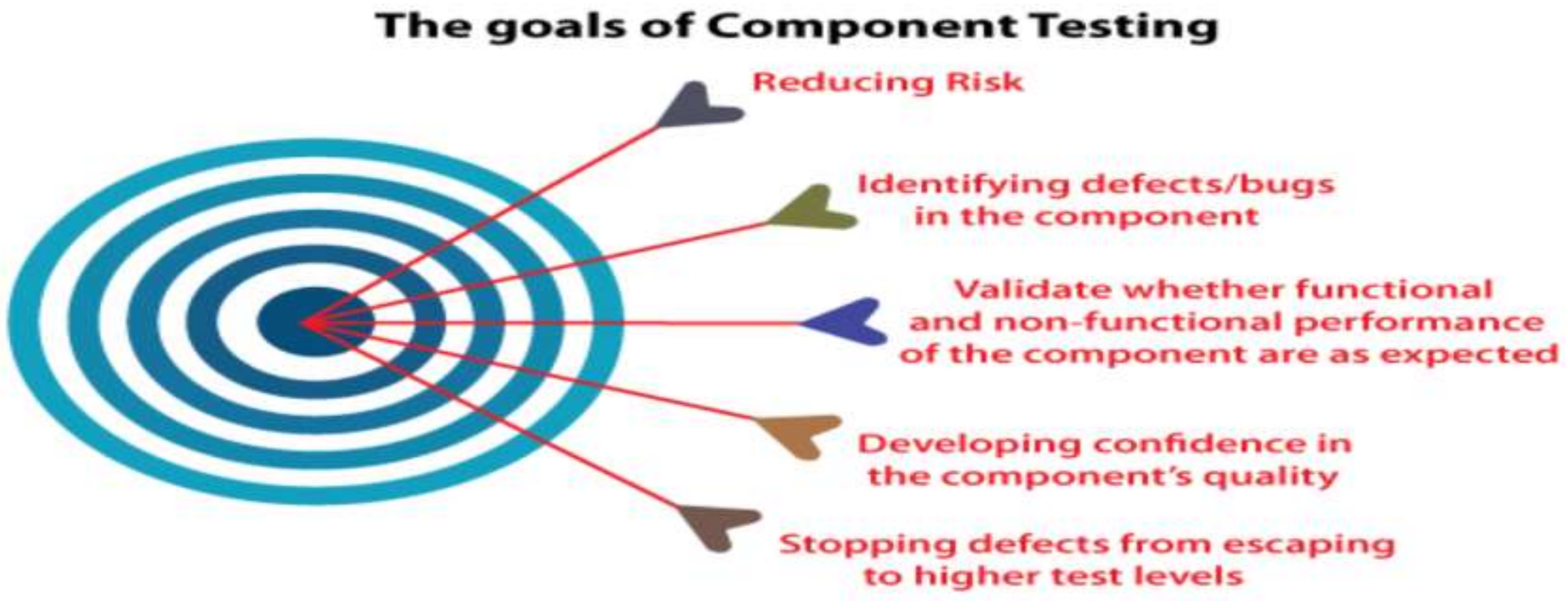
- Evaluates the overall functionality and performance of a complete and fully integrated software solution.
- System testing detects defects within both the integrated units and the whole system.
- Carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or the context of both.
- Some of the tools used for system testing are Jmeter, Gallen Framework, HP Quality Center/ALM,IBM Rational Quality Manager Microsoft Test Manager, Selenium, Appium, Apache Jserv etc.

Types of System Testing

- ➢**Performance Testing:** To test the speed, scalability, stability and reliability of the software product or application.
- ➢**Load Testing:** To determine the behavior of a system or software product under extreme load.
- ➢**Stress Testing:** To check the robustness of the system under the varying loads.
- ➢**Scalability Testing:** To check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

❖ Component testing

- Components separately as well as the usability testing; interactive valuation is also done for each specific component. It is further known as **Module Testing or Program Testing and Unit Testing.**


The goals of Component Testing
Reducing Risk
Identifying defects/bugs in the component
Validate whether functional and non-functional performance of the component are as expected
Developing confidence in the component's quality
Stopping defects from escaping to higher test levels

❖ Acceptance Testing

- Acceptance testing is formal testing based on user requirements and function processing.
- Types of acceptance testing:

1. User Acceptance Testing (UAT)

- User acceptance testing is used to determine whether the product is working for the user correctly. 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.

3. Contract Acceptance Testing (CAT)

- CAT is a contract that 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.

4. Regulations Acceptance Testing (RAT)

- RAT is used to determine whether the product violates the rules and regulations that are defined by the government of the country where it is being released.

5. Operational Acceptance Testing (OAT)

- It mainly includes testing of recovery, compatibility, maintainability, reliability, etc. OAT assures the stability of the product before it is released to production.

6. Alpha Testing

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

7. Beta Testing

- Beta testing is used to assess the product by exposing it to the real end-users, typically called beta testers in their environment.

❖ Test case design

- Test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Here are the general steps and considerations for designing effective test cases:

1. Understand Requirements and Specifications

- **Review Documentation:** Understand the requirements, specifications, and use cases of the software.
- **Identify Testable Requirements:** Determine which requirements can be tested and how they can be measured.

2. Define Test Objectives

- **Purpose of the Test:** Clearly define what each test case is supposed to achieve.
- **Scope:** Identify what part of the application is being tested.

3. Determine Test Conditions

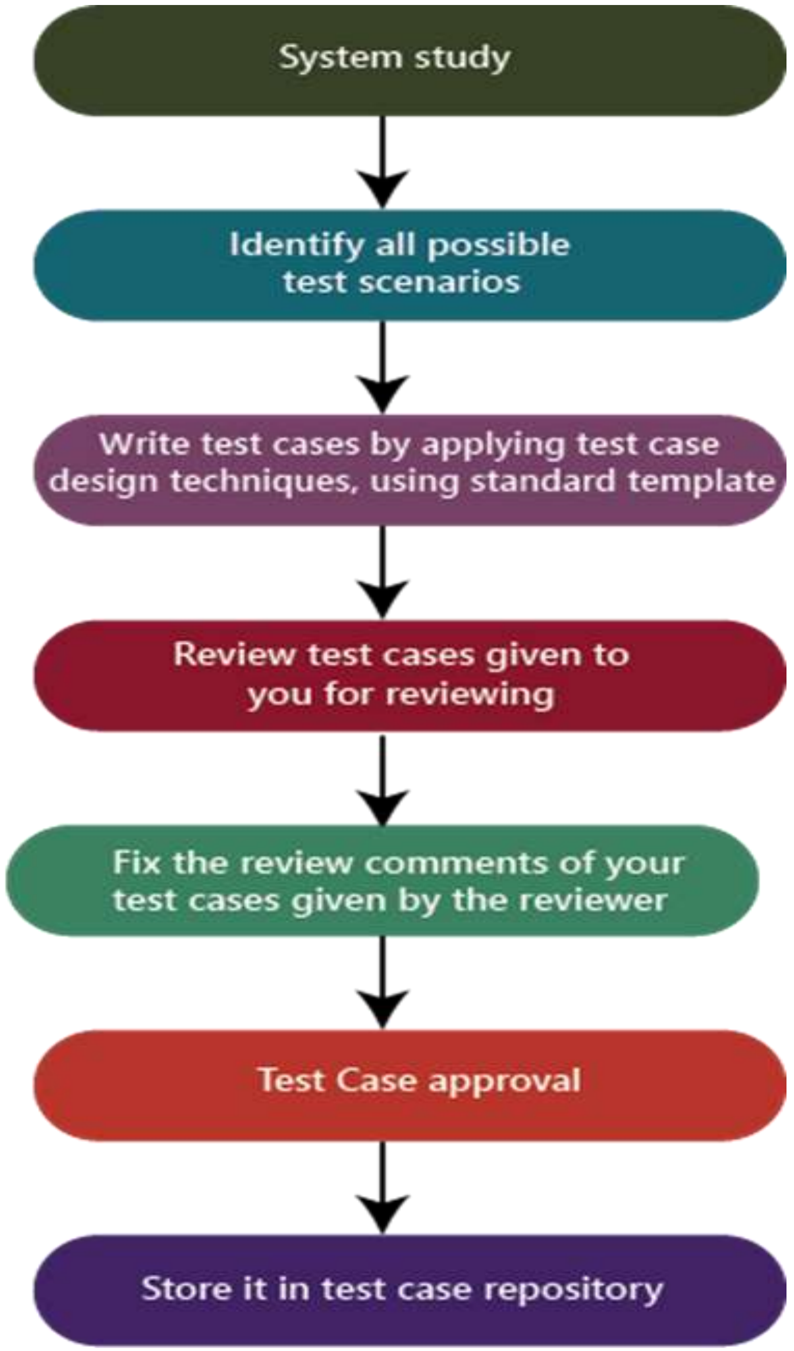
- **Identify Scenarios:** Determine the various scenarios under which the software should be tested.
- **Consider Edge Cases:** Think about unusual or extreme inputs and conditions.

4. Create Test Cases

- **Test Case ID:** Assign a unique identifier to each test case.
- **Title:** Provide a clear, descriptive title.
- **Description:** Write a brief description of what the test case will verify.
- **Preconditions:** List any conditions that must be met before executing the test.
- **Test Steps:** Detailed, step-by-step instructions on how to execute the test.
- **Test Data:** Specify the data to be used for testing.
- **Expected Result:** Clearly define the expected outcome of the test.
- **Actual Result:** Record the actual result after executing the test.
- **Pass/Fail Criteria:** Criteria to determine if the test has passed or failed.

5. Prioritize Test Cases

- **Risk-Based Testing:** Prioritize test cases based on the impact and likelihood of potential failures.
- **Frequency of Use:** Prioritize tests for features that will be used most frequently by end-users.
- **6. Review and Optimize**
- **Peer Review:** Have other team members review the test cases to ensure coverage and clarity.
- **Update Regularly:** Update test cases as requirements change and new features are added.


System study
Identify all possible test scenarios
Write test cases by applying test case design techniques, using standard template
Review test cases given to you for reviewing
Fix the review comments of your test cases given by the reviewer
Test Case approval
Store it in test case repository

❖ Test Automation

- Test automation in software testing involves using software tools to execute predefined test cases on the application under test comparing the actual outcomes with expected outcomes and reporting the results.
- Test automation is the practice of automatically reviewing and validating a software product such as a web application to make sure it meets predefined quality standards for code style, functionality (business logic), and user experience.

Test Automation Tools

- ➢**Web Applications:** Selenium, Cypress, Puppeteer.
- ➢**Mobile Applications:** Appium, Espresso, XCUITest.
- ➢**API Testing:** Postman, SoapUI, RestAssured.
- ➢**Unit Testing:** JUnit, TestNG, NUnit.

❖ Test Metrices

- Test metrics are essential in determining the software’s quality and performance.
- Software Testing Metrics are the quantitative measures used to estimate the progress, quality, productivity and health of the software testing process.
- A Metric defines in quantitative terms the degree to which a system, system component, or process possesses a given attribute

Importance of test Metrices

- Take decision for next phase of activities
- Evidence of the claim or prediction
- Understand the type of improvement required
- Take decision or process or technology change

Types of test metrices

- **Process Metrics:** It can be used to improve the process efficiency of the SDLC (Software Development Life Cycle)
- **Product Metrics:** It deals with the quality of the software product
- **Project Metrics:** It can be used to measure the efficiency of a project team or any testing tools being used by the team members

Manual test Metrices

➢Base Metrics

➢Calculated Metrics

❖ Algorithmic cost modelling

- A mathematical function is used to estimate the cost.
- Commonly used product attribute is LOC (code size).
- The function points considered are

✓External inputs and outputs.

✓User interactions.

✓External interfaces.

✓Files used by the system.

The productivity factors are

- ✓Application Domain experience
- ✓Process Quality
- ✓Product size
- ✓Technology support
- ✓Working Environment

COCOMO Model

The Constructive Cost Model (COCOMO) is a software cost estimation model that helps predict the effort, cost, and schedule required for a software development project.

Types of COCOMO model

1.Basic COCOMO model

- ➢Uses a simple mathematical formula to predict how many person-months of work are required based on the size of the project, measured in thousands of lines of code (KLOC).

- $E = a*(KLOC)^{b}\ PM$
- $Tdev = c*(E)^{d}$

Person required = Effort/ Time

E is effort applied in Person-Months

KLOC is the estimated size of the software product indicate in Kilo Lines of Code

Tdev is the development time in months

a, b, c are constants determined by the category of software project

2. Intermediate COCOMO model

- only a function of the number of lines of code and some constants evaluated according to the different software systems.
- Various other factors such as reliability, experience, and Capability. These factors are known as **Cost Drivers (multipliers)** and the Intermediate Model utilizes 15 such drivers for cost estimation.
- $E = a*(KLOC)^{b} * EAF\ PM$
- $Tdev = c*(E)^{d}$
- *Where*

E is effort applied in Person-Months

KLOC is the estimated size of the software product indicate in Kilo Lines of Code

EAF is the Effort Adjustment Factor (EAF) is a multiplier used to refine the effort estimate obtained from the basic COCOMO model.

Tdev is the development time in months

a, b, c are constants determined by the category of software project

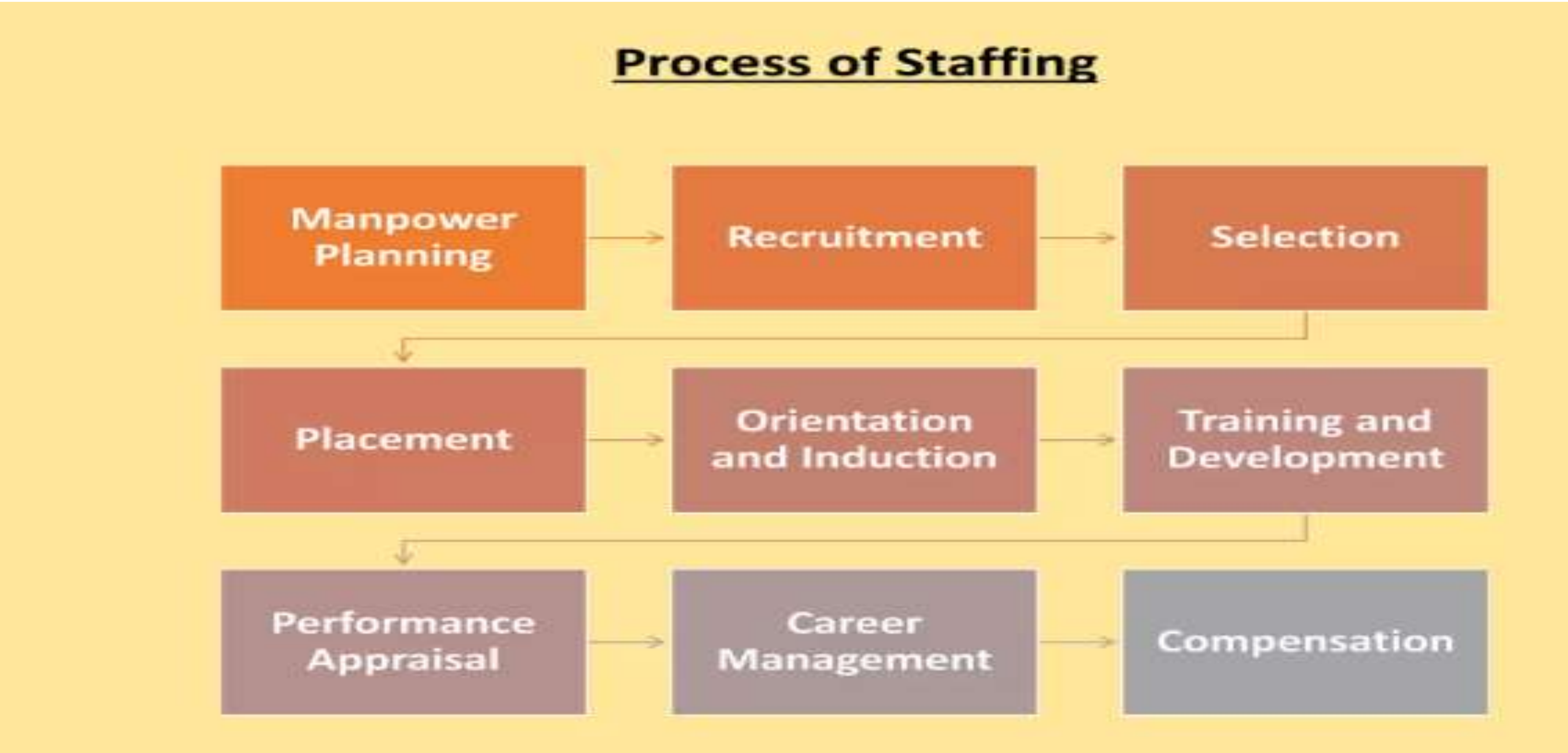
3. Detailed COCOMO model

- Considers a wider range of parameters, like team experience, development practices, and software complexity.
- By analyzing these factors in more detail, Detailed COCOMO provides a highly accurate estimation of effort, time, and cost for software projects.

❖ Project Duration and Staffing

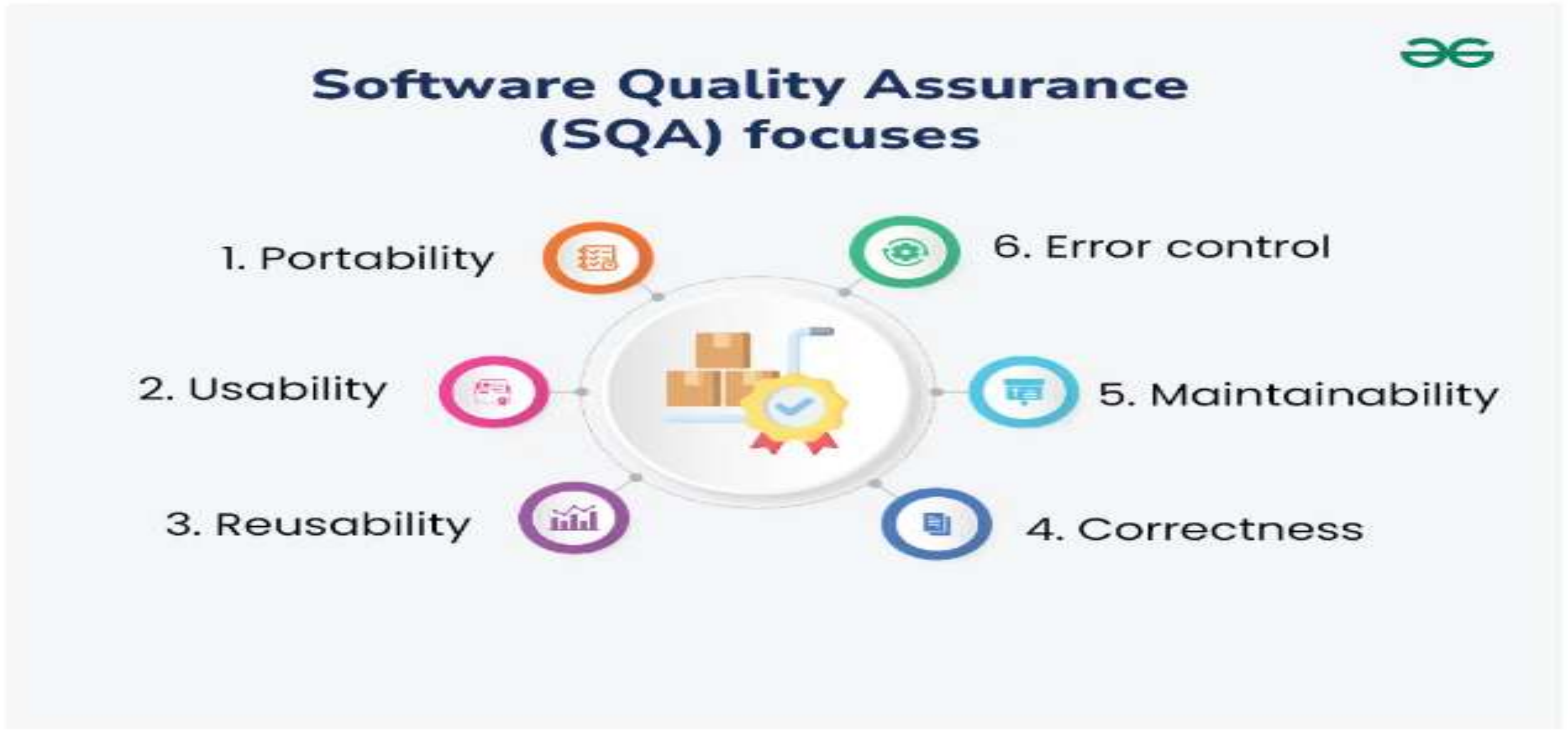
- As well as effort estimation, managers must estimate the calendar time required to complete a project and what number of staffs will be required.
- Calendar time can be calculated using COCOMO model formula.
- Staffing is the process of finding the right worker with appropriate qualification and experience and or recruiting them to fill the job position or role.
- Through this process , organization acquire deploy and retain a workforce of sufficient quantity and quality to create the positive impact on the organization effectiveness.

Process of staffing



❖ Software Quality Assurance

- Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.
- Software Quality assurance focuses on



Software Quality Assurance (SQA)

SQA encompasses

- A quality management approach
- Effective Software engineering technology (methods and tools)
- Formal technical reviews that are tested throughout the software process
- A multitier testing strategy.
- Control of software documentation and the changes made to it.
- A procedure to ensure compliances with software development standards
- Measuring and reporting mechanisms.

Kinds of Quality

- **Quality of Design:** Quality of Design refers to the characteristics that designers specify for an item.
- **Quality of Conformance:** Quality of conformance is the degree to which the design specifications are followed during development.

- **Software Quality:** Software Quality is defined as the conformance to explicitly state functional and performance requirements, explicitly documented development standards, and inherent characteristics.
- **Quality Control:** Quality Control involves a series of inspections, reviews, and tests used throughout the software process.
- **Quality Assurance:** Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully. focuses on how the engineering and management activity will be done.

❖ Formal technical Reviews

- are structured, systematic, and organized processes used to evaluate software products or deliverables in order to identify defects, ensure quality, and improve the development process.
- Finding flaws, making sure standards are followed, and improving the product or document under review's overall quality are the main objectives of a formal technical review (FTR).

Objectives of formal technical review (FTR)

- **Detect Identification:** Identify defects in technical objects by finding and fixing mistakes, inconsistencies, and deviations.
- **Quality Assurance:** To ensure high-quality deliverables and confirm compliance with project specifications and standards.

- **Risk Mitigation:** To stop risks from getting worse, proactively identify and manage possible threats.
- **Knowledge Sharing:** Encourage team members to work together and build a common knowledge base.
- **Consistency and Compliance:** Verify that all procedures, coding standards, and policies are followed.
- **Learning and Training:** Give team members the chance to improve their abilities through learning opportunities.

❖ Formal approaches to SQA

- **Quality Standards and Models**: Establishing and adhering to quality standards such as ISO/IEC 25000 series, CMMI (Capability Maturity Model Integration), and others. These provide frameworks for assessing and improving processes.
- **Process-Based Approaches**: Implementing formal processes like Agile, Scrum, Waterfall, or DevOps to manage and control the software development lifecycle (SDLC) effectively.
- **Testing and Validation**: Formal testing strategies, including unit testing, integration testing, system testing, and acceptance testing, are crucial for verifying that software meets functional and non-functional requirements.
- **Code Reviews and Inspections**: Formal reviews and inspections of code and design documents to detect defects and ensure adherence to coding standards and best practices.

- **Documentation**: Maintaining comprehensive documentation throughout the SDLC, including requirements specifications, design documents, test plans, and user manuals, to ensure clarity and maintainability.
- **Configuration Management**: Formal configuration management practices to manage changes systematically, including version control, baseline management, and configuration audits.
- **Metrics and Measurements**: Using formal metrics to quantify software quality attributes such as reliability, performance, security, and maintainability, enabling continuous improvement efforts.
- **Risk Management**: Formal approaches to identify, assess, and mitigate risks throughout the software development process, ensuring potential issues are addressed proactively.

❖ **Statistical software quality assurance**

- Information about software errors and defects is collected and categorizes.
- An attempt is mase to trace each error and defects to its underlying cause.
- Using the Pareto principle(80 % of the effects can be traces to 20% of all possible causes), isolate the 20%.
- Once the vital few causes have been identified, move on correct the problems that have caused the errors and defects.

- The software can encounter more defects and uncovered errors like

✓Incomplete and erroneous specification.

✓Violation of the programming standards.

✓Error in design logic.

✓Incomplete and erroneous testing.

✓Error in data representation.

✓Inconsistent component interface.

✓Ambiguous human computer interface.

- **Six sigma** is most widely used strategy for statistical quality assurance in industry.
- Six Sigma is a **methodology** that aims to improve the quality of processes by identifying and removing the causes of defects (errors) and minimizing variability in manufacturing and business processes.

❖ Framework for software metrices

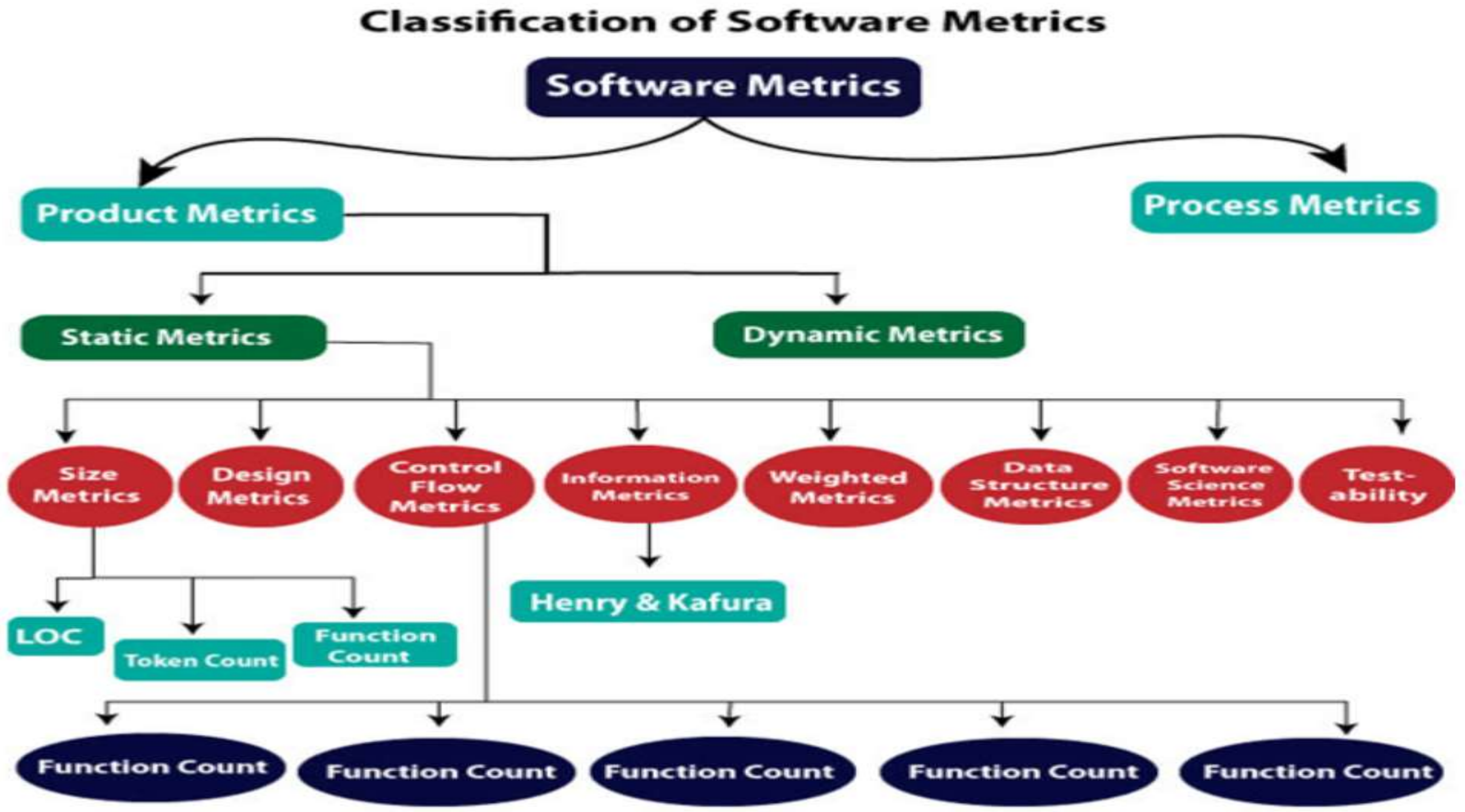
- A software metric is a measure of software characteristics which are measurable or countable.
- Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.
- **Software metrics can be classified into two types as follows:**

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.



Types of Metrices

• **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.

• **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.

• **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.

• **Project metrics:** Project metrics are the metrics used by the project manager to check the project's progress. 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.

❖ Matrices for analysis and design model

- Metrics for design modeling allows developers or software engineers to evaluate or estimate quality of design and include various architecture and component-level designs.
- The design complexities are :

➢ **Structural Complexity**

Structural complexity depends upon fan-out for modules.

➢ **Data Complexity**

Data complexity is complexity within interface of internal module. It is size and intricacy of data.

➢ **System Complexity**

System complexity is combination of structural and data complexity.

- When structural, data, and system complexity get increased, overall architectural complexity also gets increased.

- Analytical models, it's crucial to assess their effectiveness, accuracy, and performance.

Some of them are:

- **1. Regression Models**
- **Mean Absolute Error (MAE)**: Average of the absolute differences between predicted and actual values.
- **Mean Squared Error (MSE)**: Average of the squared differences between predicted and actual values.
- **Root Mean Squared Error (RMSE)**: Square root of MSE, providing a more interpretable error metric.
- **R-squared (R^2)**: Proportion of the variance in the dependent variable that is predictable from the independent variables.

2. Classification Models

- **Accuracy**: Proportion of correctly classified instances out of total instances.
- **Precision**: Proportion of true positive predictions out of all positive predictions made.
- **Recall (Sensitivity)**: Proportion of true positives correctly identified as such out of all actual positives.
- **F1-score**: Harmonic mean of precision and recall, balancing both metrics.
- **Receiver Operating Characteristic (ROC) Curve**: Plots the true positive rate against the false positive rate across various thresholds.
- **Area Under the ROC Curve (AUC)**: Quantifies the overall performance of a classification model.

❖ ISO standards

- Mission of the International Standard Organization is to market the development of standardization and its related activities to facilitate the international exchange of products and services.
- It also helps to develop cooperation within the different activities like spheres of intellectual, scientific, technological, and economic activity.
- Different types of ISO standards are :

➢**ISO 9000: 2000 –**

ISO 9000: 2000: contains Quality management systems, fundamentals, and vocabulary.

➢**ISO 9000-1: 1994 –**

This series of standards includes Quality management systems and Quality assurance standards. It also includes some guidelines for selection and use.

- ➢**ISO 9000-2: 1997 –**This series of standards also includes Quality management systems and Quality assurance standards. It also includes some guidelines for the application of ISO 9001, ISO 9002, and ISO 9003
- ➢**ISO 9000-3: 1997 –**
  This series contains Quality management systems, Quality assurance standards and also includes guidelines for the application of ISO 9001 to 1994 to the development, supply, installation, and maintenance of computer installation.
- ➢**ISO 9001**: **1994** –
  This series of standards has Quality systems and a Quality assurance model. This model helps in design, development, production, installation, and service.
- ➢**ISO 9001**: **2000 –**
  This series of standards also includes Quality management systems.

❖ CMMI

- Capability Maturity Model Integration, is a process improvement approach that provides organizations with essential elements for effective process improvement.
- It helps integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and measure progress.

➢ **Maturity Levels**:

- CMMI defines five maturity levels that represent a path of continuous improvement:
  - **Level 1 - Initial**: Processes are ad hoc and chaotic.
  - **Level 2 - Managed**: Processes are planned and executed in accordance with policies and procedures.

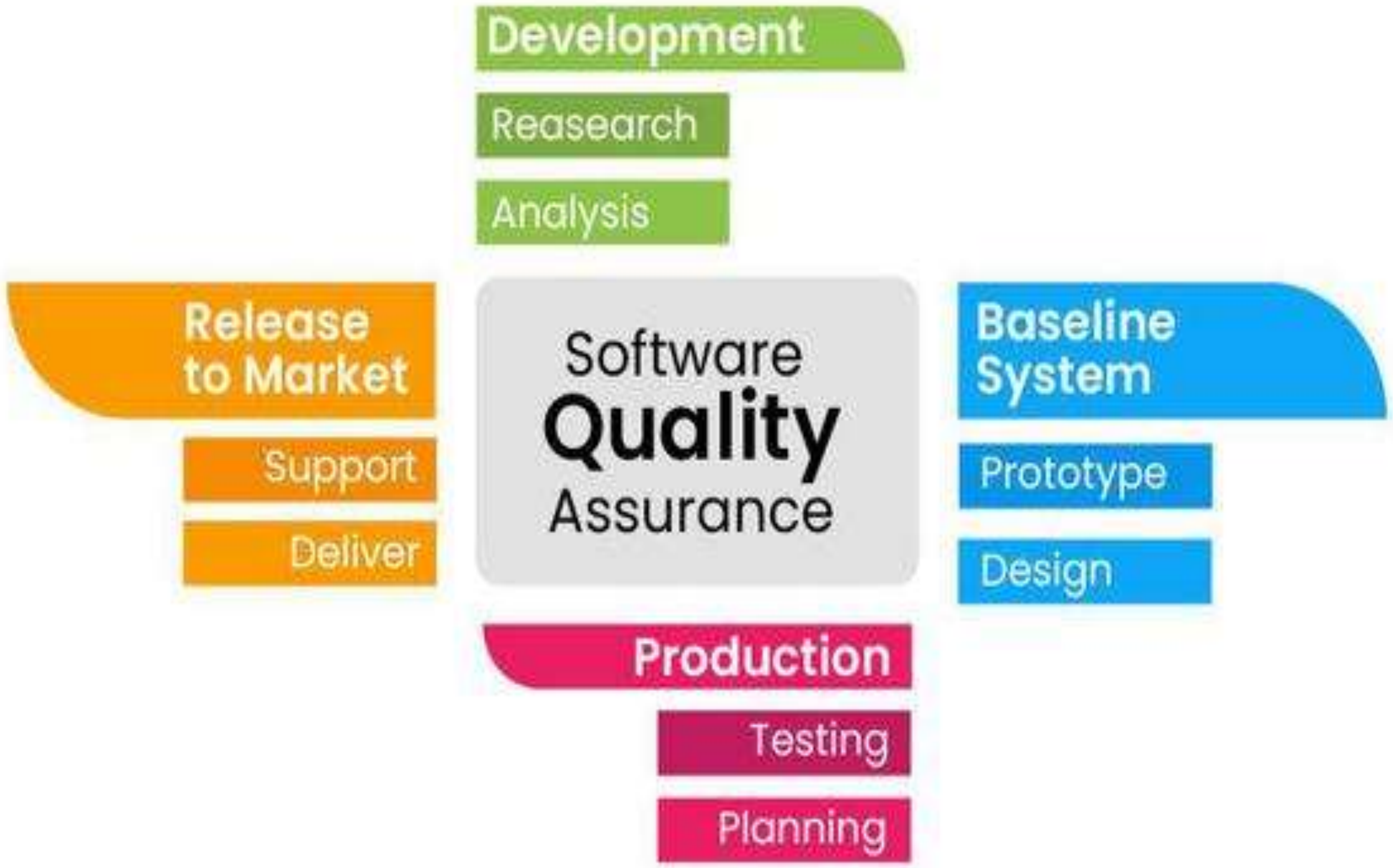
- **Level 3 - Defined**: Processes are documented, standardized, and integrated across the organization.
- **Level 4 - Quantitatively Managed**: Processes are controlled using statistical and quantitative techniques.
- **Level 5 - Optimizing**: Focus on continuous process improvement based on quantitative understanding.

➢**Capability Levels**:

- CMMI also defines capability levels for specific process areas within each maturity level:
  - **Level 0**: Incomplete.
  - **Level 1**: Performed.
  - **Level 2**: Managed.
  - **Level 3**: Defined.
  - **Level 4**: Quantitatively Managed.
  - **Level 5**: Optimizing.

❖ SQA Plan

- A software quality assurance plan’s main goal is to guarantee that the market’s product or service is trouble- and bug-free
- Fulfill the specifications listed in the SRS .
- A Quality Assurance Plan (QAP) is a document or set of documents that outlines the systematic processes, procedures, and standards for ensuring the quality of a product or service.
- For a software product or service, an SQA plan will be used in conjunction with the typical development, prototyping, design, production, and release cycle.
- An SQA plan will include several components, such as purpose, references, configuration and management, tools, code controls, testing methodology, problem reporting and remedial measures, and more, for easy documentation and referencing


Development
Reasearch
Analysis
Release to Market
Support
Deliver
Software Quality Assurance
Baseline System
Prototype
Design
Production
Testing
Planning

SQA Plan

❖ Configuration management planning

- When the software is developed the product undergoes many changes in their maintenance phase; we need to handle these changes effectively.
- The elements that comprise all information produced as a part of the software process are collectively called a software configuration.
- These are handled and controlled by SCM. This is where we require software configuration management.
- Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

SCM is the discipline which

- Identify change
- Monitor and control change
- Ensure the proper implementation of change made to the item.
- Auditing and reporting on the change made.

➢ Multiple people are working on software which is consistently updating. It may be a method where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently.

➢ It provides the tool to ensure that changes are being properly implemented.

➢ SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.

❖ Change management

- Change management in software engineering refers to the process of managing changes to software systems in a structured way.
- It involves planning, tracking, and controlling changes to ensure they are implemented smoothly and effectively. Here are some key aspects of change management in software engineering:
- Here are some key aspects of change management in software engineering:

➢ **Change Identification**: Recognizing the need for change, whether it's due to new requirements, bug fixes, enhancements, or regulatory requirements.

➢ **Change Request**: Formalizing the change request, which typically includes details such as the reason for the change, impact analysis, and proposed solution.

➢ **Impact Analysis**: Assessing the potential effects of the change on the system, including technical, operational, and business impacts.

- **Change Planning**: Developing a plan for implementing the change, which may involve scheduling, resource allocation, and risk assessment.
- **Change Implementation**: Executing the change according to the plan, ensuring that it is properly tested and validated before deployment.
- **Change Review**: Evaluating the effectiveness of the change after implementation, including gathering feedback and addressing any issues that may arise.
- **Change Documentation**: Documenting the change process, including all relevant details and decisions made throughout the lifecycle of the change.
- **Change Control**: Monitoring and controlling changes to ensure they align with the overall software development goals and standards

❖ Version and release management

- Identifying and keeping track of the versions of a system.
- Versions managers devise procedures to ensure that versions of a system may be retrieved when required and are not accidentally changed by the development team.
- Versions with only small differences are sometimes called **variants.**
- Versions are created with an organization for internal development or testing and are not intended for release to customers.
- To create a specific version of a system, specify the versions of the system components that ought to be included in it.
- In a large software system, there are hundreds to software components, each of which may exist in several different versions.

Three basic techniques are used for components version identification :

- **Version Numbering :** In version numbering scheme, a version number is added to the components or system name. If the first version is called 1.0, subsequent versions are 1.1, 1.2 and so on.
- **Attribute Based Identification :** If each version is identified by a unique set of attributes, it is easy to add new versions, that are derived from any of existing versions. These are identified using unique set of attribute values.
- **Change Oriented Identification :** Each component is known as in attribute-based identification but is additionally related to one or more change requests. That is, it is assumed that each version of component has been created in response to one or more change requests.

❖ CASE tools for configuration management

- CASE (Computer-Aided Software Engineering) tools for configuration management help manage and control software development processes. Some commonly used CASE tools for configuration management include:

1. **Git**: A distributed version control system widely used for tracking changes in source code during software development.
2. **SVN (Subversion)**: A centralized version control system that allows teams to collaborate on software development projects.
3. **Mercurial**: Another distributed version control system similar to Git, often used for managing software source code.
4. **Perforce**: A version control and collaboration platform that supports large-scale development environments.

5. **ClearCase**: A version control system that provides features for software configuration management, parallel development, and workspace management.
6. **Team Foundation Server (TFS)**: Provides version control, reporting, requirements management, project management, automated builds, testing, and release management capabilities.
7. **JIRA**: While primarily a project management tool, JIRA also includes features for issue tracking and agile project management, which are essential for configuration management in software development.

THANK YOU