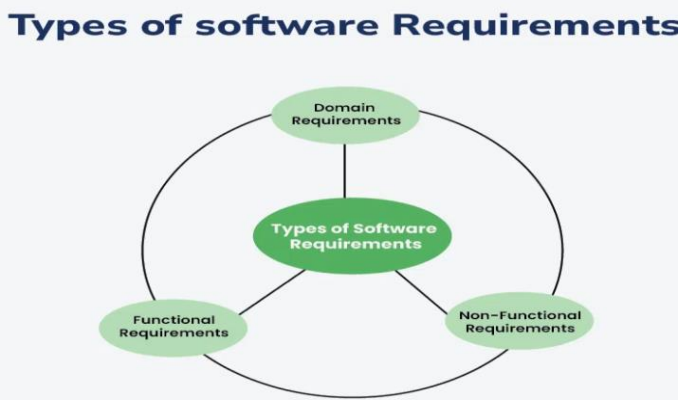
UNIT - 2

 # SOFTWARE REQUIREMENTS:

- Requirements analysis is an essential process that enables the success of a system or software project to be assessed.
- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
- The main types of Software Requirements are functional, non-functional, and domain requirements.



1. FUNCTIONAL REQUIRENMENT:

- Functional requirements describe what the software should do. They define the functions or features that the system must have.
- Functional Requirements are the requirements that the end user specifically demands as basic facilities that the system should offer. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality that defines what function a system is likely to perform.
- They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. For example, in a hospital management system, a doctor should be able to retrieve the information of his patients.

Examples:

- **User Authentication**: The system must allow users to log in using a username and password.
- **Search Functionality**: The software should enable users to search for products by name or category.

- **Report Generation**: The system should be able to generate sales reports for a specified date range.

Explanation: Functional requirements specify the actions that the software needs to perform. These are the basic features and functionalities that users expect from the software.

2. NON-FUNCTIONAL REQUIREMENT:

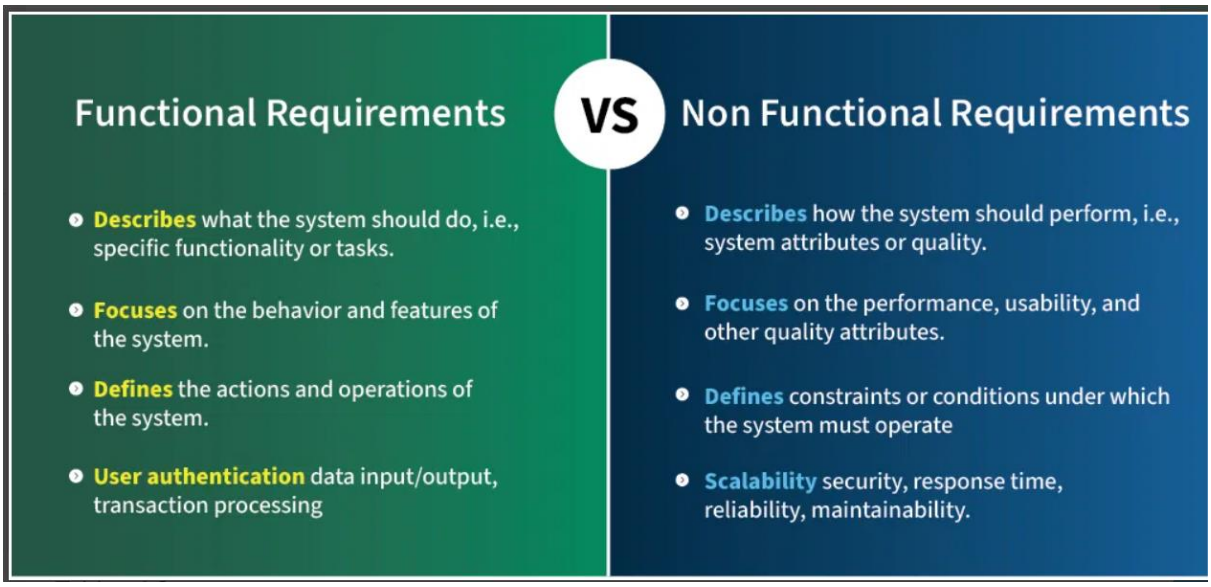
- Non-functional requirements describe how the software performs a task rather than what it should do. They define the quality attributes, performance criteria, and constraints.
- These are basically the quality constraints that the system must satisfy according to the project contract. Nonfunctional requirements, not related to the system functionality, rather define how the system should perform
  They basically deal with issues like:
- **Portability**
- **Security**
- **Maintainability**
- **Reliability**
- **Scalability**
- **Performance**
- **Reusability**
- **Flexibility**

  Explanation: Non-functional requirements are about the system's behavior, quality, and constraints. They ensure that the software meets certain standards of performance, usability, reliability, and security.

3. DOMAIN REQUIREMENTS:

- Domain requirements are specific to the domain or industry in which the software operates. They include terminology, rules, and standards relevant to that particular domain.
- Domain requirements can be functional or nonfunctional. Domain requirements engineering is a continuous process of proactively defining the requirements for all foreseeable applications to be developed in the software product line.
  Examples:
- **Healthcare**: The software must comply with HIPAA regulations for handling patient data.
- **Finance**: The system should adhere to GAAP standards for financial reporting.
- **E-commerce**: The software should support various payment gateways like PayPal, Stripe, and credit cards.

Explanation: Domain requirements reflect the unique needs and constraints of a particular industry. They ensure that the software is relevant and compliant with industry-specific regulations and standards.



Below are the differences between Functional Requirements and Non-Functional Requirements:

Aspect	Functional Requirements	Non-Functional Requirements
Definition	Describes what the system should do, i.e., specific functionality or tasks.	Describes how the system should perform, i.e., system attributes or quality.
Purpose	Focuses on the behavior and features of the system.	Focuses on the performance, usability, and other quality attributes.
Scope	Defines the actions and operations of the system.	Defines constraints or conditions under which the system must operate.

Aspect	Functional Requirements	Non-Functional Requirements
Examples	User authentication, data input/output, transaction processing.	Scalability, security, response time, reliability, maintainability.
Measurement	Easy to measure in terms of outputs or results.	More difficult to measure, often assessed using benchmarks or SLAs.
Impact on Development	Drives the core design and functionality of the system.	Affects the architecture and overall performance of the system.
Focus on User Needs	Directly related to user and business requirements.	Focuses on user experience and system performance.
Documentation	Typically documented in use cases, functional specifications, etc.	Documented through performance criteria, technical specifications, etc.
Evaluation	Can be tested through functional testing (e.g., unit or integration tests).	Evaluated through performance testing, security testing, and usability testing.
Dependency	Determines what the system must do to meet user needs.	Depends on how well the system performs the required tasks.

Advantages of Classifying Software Requirements:

Better organization: Classifying software requirements helps organize them into groups that are easier to manage, prioritize, and track throughout the development process.

Improved communication: Clear classification of requirements makes it easier to communicate them to stakeholders, developers, and other team members. It also ensures that everyone is on the same page about what is required.

Increased quality: By classifying requirements, potential conflicts or gaps can be identified early in the development process. This reduces the risk of errors, omissions, or misunderstandings, leading to higher-quality software.

Improved traceability: Classifying requirements helps establish traceability, which is essential for demonstrating compliance with regulatory or quality standards.

Disadvantages of classifying software requirements:

Complexity: Classifying software requirements can be complex, especially if there are many stakeholders with different needs or requirements. It can also be time-consuming to identify and classify all the requirements.

Rigid structure: A rigid classification structure may limit the ability to accommodate changes or evolving needs during the development process. It can also lead to a siloed approach that prevents the integration of new ideas or insights.

Misclassification: Misclassifying requirements can lead to errors or misunderstandings that can be costly to correct later in the development process.

USER REQUIRENMENT:

- These requirements describe what the end-user wants from the software system. User requirements are usually expressed in natural language and are typically gathered through interviews, surveys, or user feedback.
- Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
- User requirements are defined using natural language, tables and diagrams.

A good user requirement should be:

- ✓ **Clear –** Easy to understand without ambiguity.
- ✓ **Complete –** Covers all aspects of user needs**.**
- ✓ **Consistent** – Does not conflict with other requirements.
- ✓ **Feasible** – Technically and financially realistic.
- ✓ **Testable –** Can be verified during software testing.

SYSTEM REQUIREMENTS:

- These requirements specify the technical characteristics of the software system, such as its architecture, hardware requirements, software components, and interfaces.
- System requirements are typically expressed in technical terms and are often used as a basis for system design.

Requirement Type	Readers
User Requirements	Client Managers, System End-users, Client Engineers, Contractor Managers, System Architects
System Requirements	System End-users, Client Engineer, System Architects, Software Developers
Software Design Specification	Client Engineers (perhaps), System Architects, Software Developers

INTERFACES SPECIFICATION:

- These requirements specify the interactions between the software system and external systems or components, such as databases, web services, or other software applications.
- **Types interface requirements:**

  - ✓ **User Interface (UI) Specification**
  - ✓ **Application Programming Interface (API) Specification**
  - ✓ **Hardware Interface Specification**
  - ✓ **Database Interface Specification**

Business requirements:

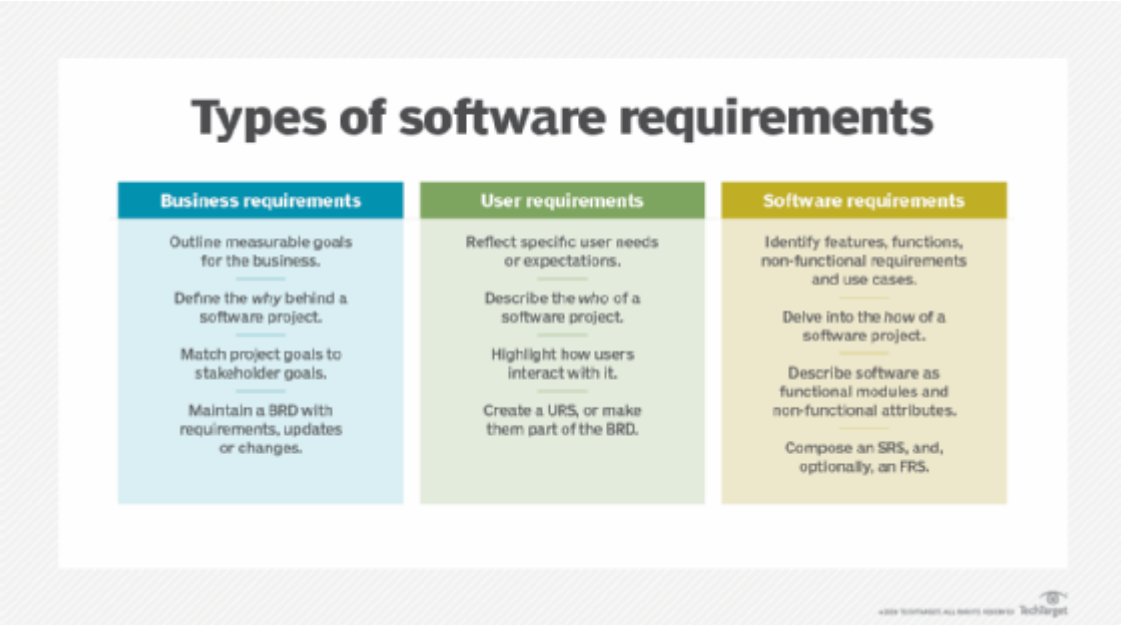
These requirements describe the business goals and objectives that the software system is expected to achieve. Business requirements are usually expressed in terms of revenue, market share, customer satisfaction, or other business metrics.

Regulatory requirements:

These requirements specify the legal or regulatory standards that the software system must meet. Regulatory requirements may include data privacy, security, accessibility, or other legal compliance requirements.

Design requirements:

These requirements describe the technical design of the software system. They include information about the software architecture, data structures, algorithms, and other technical aspects of the software.



SOFTWAR REQUIREMENTS SPECIFICATION

- A Software Requirements Specification (SRS) is like a blueprint or a recipe for building software.
- It tells developers, designers, and testers what the software should do, how it should work, and any rules it must follow.
- Imagine you want to build a mobile app. Before making it, you need to write down all the details about how the app should function, what it should look like, and what users can do with it. That's exactly what an SRS document does.
- Software Requirement Specification (SRS) Format as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system.
- These requirements can be functional as well as non-functional depending upon the type of requirement. The interaction between different customers and contractors is done because it is necessary to fully understand the needs of customers.

1.Introduction

Purpose of this Document – At first, main aim of **why** this document is necessary and what's purpose of document is explained and described.
Scope of this document – In this, overall working and main objective of document and **what value** it will provide to customer is described and explained. It also includes a description of development cost and time required.
Overview – In this, description of product is explained. It's simply summary or overall review of product.

2.General description

In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

Functional Requirements

In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order. Functional requirements specify the expected behavior of the system-

which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, detailed description all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

Interface Requirements

In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

Performance Requirements

- In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc. The performance requirements part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified.
- There are two types of performance requirements: static and dynamic. Static requirements are those that do not impose constraint on the execution characteristics of the system. Dynamic requirements specify constraints on the execution behaviour of the system.

3.Design Constraints

In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc. There are a number of factors in the client's environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

Non-Functional Attributes

In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

4.Preliminary Schedule and Budget

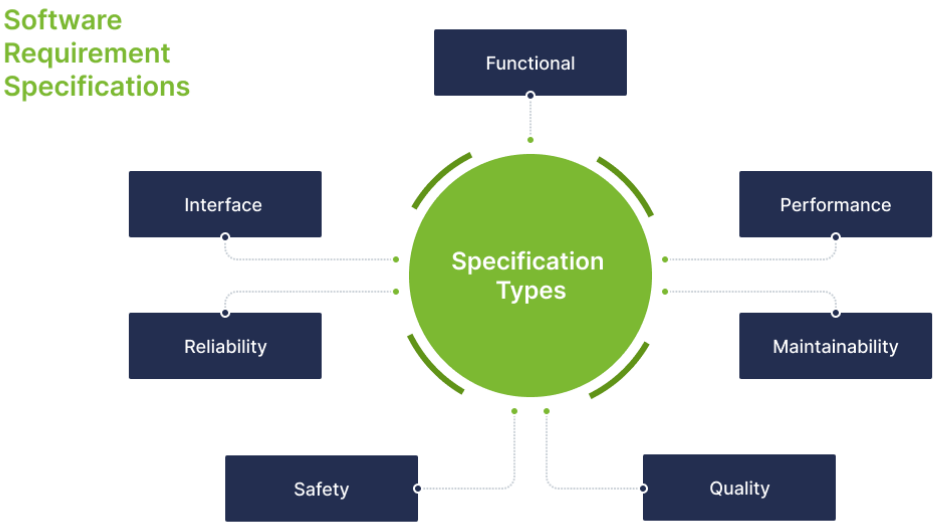
In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

5.Appendices

In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

Uses of SRS document

- Development team require it for developing product according to the need.
- Test plans are generated by testing group based on the describe external behaviour.
- Maintenance and support staff need it to understand what the software product is supposed to do.
- Project manager base their plans and estimates of schedule, effort and resources on it.
- customer rely on it to know that product they can expect.
- As a contract between developer and customer.
- in documentation purpose.

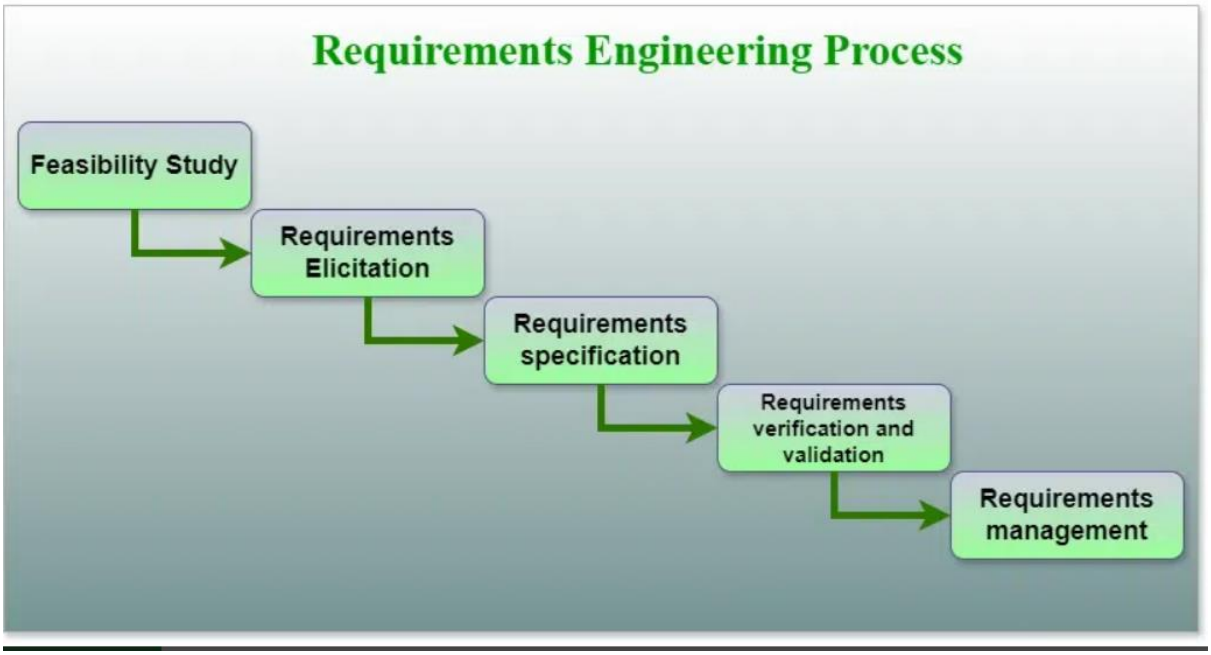


#REQUIREMENT ENGINEERING:

- Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process.
- It gives the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.

- A systematic and strict approach to the definition, creation, and verification of requirements for a software system is known as requirements engineering.
- To guarantee the effective creation of a software product, the requirements engineering process entails several tasks that help in understanding, recording, and managing the demands of stakeholders.

Requirements Engineering Process



1. **Feasibility Study**
2. **Requirements elicitation**

3. **Requirements specification**
4. **Requirements for verification and validation**
5. **Requirements management**

1. Feasibility :

- The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.
- **Types of Feasibility:**

1.Technical Feasibility - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

2.Operational Feasibility - Operational feasibility assesses the range in which the required software performs a series of levels to solveiness problems and customer requirements.
3.Economic Feasibility - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

4. Legal Feasibility: In legal feasibility, the project is ensured to comply with all relevant laws, regulations, and standards. It identifies any legal constraints that could impact the project and reviews existing contracts and agreements to assess their effect on the project's execution..
5.Schedule Feasibility: In schedule feasibility, the project timeline is evaluated to determine if it is realistic and achievable. Significant milestones are identified, and deadlines are established to track progress effectively.

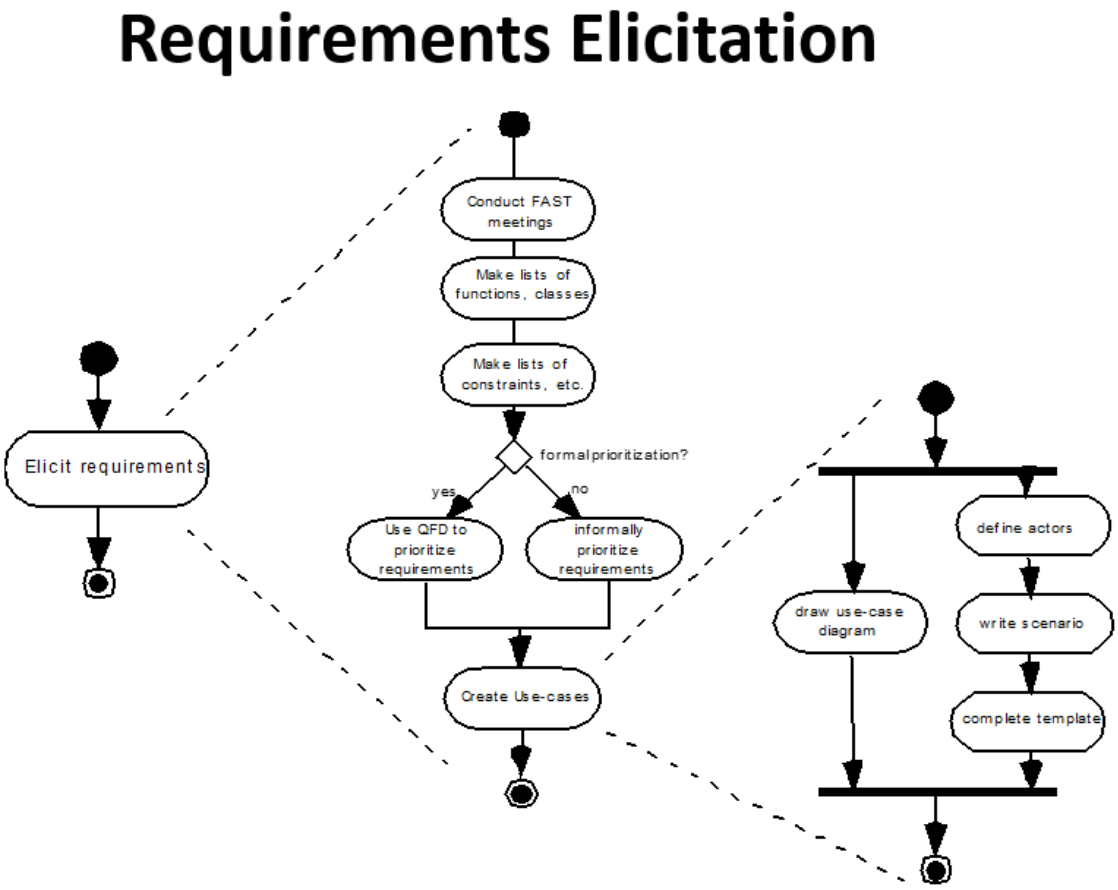
2.. Requirement Elicitation and Analysis:

1. This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes.
2. Also known as Requirements elicitation or requirements discovery.
3. Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
4. May involve end-users, managers, engineers involved in maintenance, domain experts etc. These are called *stakeholders*.

Several techniques can be used to elicit requirements, including:

- **Interviews**: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.
- **Surveys**: These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.

- **Focus Groups**: These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
- **Observation**: This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
- **Prototyping**: This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.



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.
- Developed by Barry Boehm in 1981, COCOMO uses a mathematical formula based on the size of the software project, typically measured in lines of code (LOC)
- The COCOMO Model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.

  1. Effort: Amount of labor that will be required to complete a task. It is measured in person-months units.

  2. **Schedule:** This simply means the amount of time required for the completion of the job.

Types of Projects in the COCOMO Model

1. **Organic Model**:

- Used for small, simple, and well-understood projects.
- Teams have experience working on similar systems, and requirements are clear.
- Example: A simple payroll system or a small business website.

2. **Semi-Detached Model**:

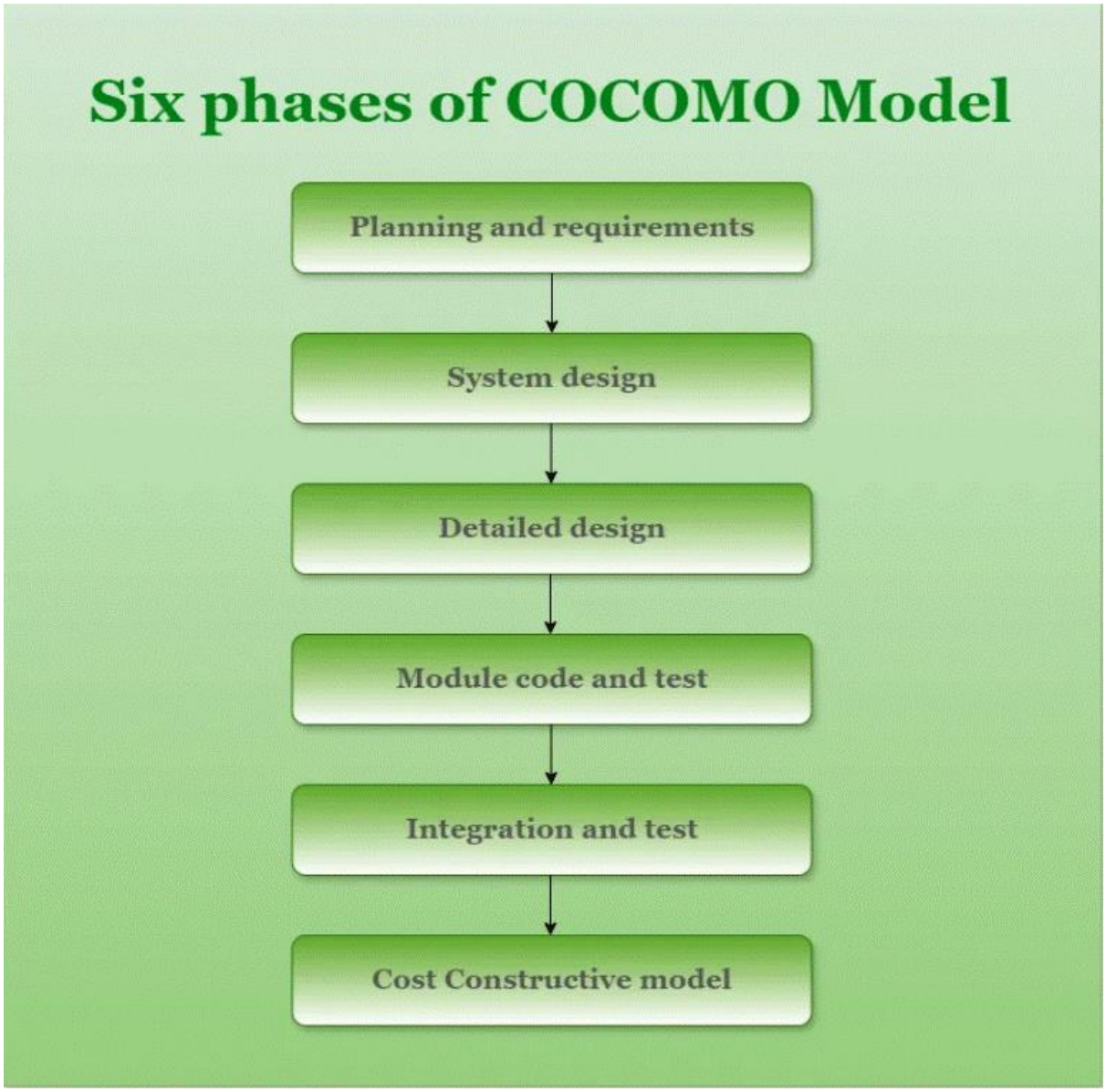
- Used for medium-sized projects with moderate complexity.
- A mix of experienced and new developers working together.
- Example: A real-time database system or a larger business application.

3. **Embedded Model**:

- Used for complex, highly regulated, and hardware-integrated systems.
- Requires strict adherence to requirements, and developers have limited flexibility.
- Example: Airplane control systems, military-grade software, or medical device software.

Comparison of these three types of Projects in COCOMO Model

Aspects	Organic	Semidetached	Embedded
Project Size	2 to 50 KLOC	50-300 KLOC	300 and above KLOC
Complexity	Low	Medium	High
Team Experience	Highly experienced	Some experienced as well as inexperienced staff	Mixed experience, includes experts
Environment	Flexible, fewer constraints	Somewhat flexible, moderate constraints	Highly rigorous, strict requirements
Effort Equation	E = 2.4(400)1.05	E = 3.0(400)1.12	E = 3.6(400)1.20
Example	Simple payroll system	New system interfacing with existing systems	Flight control software



1. **Planning and requirements:** This initial phase involves defining the scope, objectives, and constraints of the project. It includes developing a project plan that outlines the schedule, resources, and milestones
2. **System design:** : In this phase, the high-level architecture of the software system is created. This includes defining the system's overall structure, including major components, their interactions, and the data flow between them**.**
3. **Detailed design:** This phase involves creating detailed specifications for each component of the system. It breaks down the system design into detailed descriptions of each module, including data structures, algorithms, and interfaces.
4. **Module code and test:** This involves writing the actual source code for each module or component as defined in the detailed design. It

includes coding the functionalities, implementing algorithms, and developing interfaces.

5. **Integration and test:** This phase involves combining individual modules into a complete system and ensuring that they work together as intended.

Importance of the COCOMO Model

1. **Cost Estimation:** To help with resource planning and project budgeting, COCOMO offers a methodical approach to software development cost estimation.
2. **Resource Management:** By taking team experience, project size, and complexity into account, the model helps with efficient resource allocation.
3. **Project Planning**: COCOMO assists in developing practical project plans that include attainable objectives, due dates, and benchmarks.
4. **Risk management**: Early in the development process, COCOMO assists in identifying and mitigating potential hazards by including risk elements.
5. **Support for Decisions**: During project planning, the model provides a quantitative foundation for choices about scope, priorities, and resource allocation.
6. **Benchmarking**: To compare and assess various software development projects to industry standards, COCOMO offers a benchmark.
7. **Resource Optimization:** The model helps to maximize the use of resources, which raises productivity and lowers costs.

Types of COCOMO Model

There are three types of COCOMO

- **Basic Model** – Provides a rough estimate of effort and cost.
- **Intermediate Model** – Includes additional factors like project attributes.
- **Detailed Model** – Adds deeper analysis, considering individual phases

1. Basic COCOMO Model

The Basic COCOMO model is a straightforward way to estimate the effort needed for a software development project. It 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).

It estimates effort and time required for development using the following expression:

$E = a*(KLOC)_b\ PM$

$Tdev = c*(E)_d$

Person required = Effort/ Time
Where,

E is effort applied in Person-Months
KLOC is the estimated size of the software product indicate in Kilo Lines of Code

2. Intermediate COCOMO Model

- The basic COCOMO model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software systems.
- However, in reality, no system's effort and schedule can be solely calculated based on Lines of Code.
- For that, 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.
- This model improves upon the basic model by considering cost drivers (factors affecting effort). It refines the effort estimation with 15 effort multipliers, such as:
- **Product attributes (complexity, reliability needs).**
- **Hardware attributes (memory constraints, response time).**
- **Personnel attributes (experience, capability).**
- **Project attributes (development tools, schedule constraints)**
-

3.Detailed Cocomo model:

- Detailed COCOMO goes beyond Basic and Intermediate COCOMO by diving deeper into project-specific factors. It 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.
- It's like zooming in on a project's unique characteristics to get a clearer picture of what it will take to complete it successfully

- This model extends the **intermediate model** by estimating effort and cost at **each phase** of software development (design, coding, testing, etc.). It breaks down:

- Effort per phase.
- Cost per phase.
- Personnel allocation.

It applies cost driver adjustments to **each phase separately** rather than to the whole project.

#LOC(LINE OF CODE):

- LOC (Lines of Code) is a software metric used to measure the size of a program by counting the number of lines in its source code. It helps estimate **effort, time, and cost** in software development.
- **Features of Lines of Code (LOC)**
- **Change Tracking:** Variations in LOC as time passes can be tracked to analyze the growth or reduction of a codebase, providing insights into project progress.
- **Limited Representation of Complexity:** Despite LOC provides a general idea of code size, it does not accurately depict code complexity. It is possible for two programs having the same LOC to be incredibly complex.
- **Ease of Computation:** LOC is an easy measure to obtain because it is easy to calculate and takes little time.
- **Easy to Understand:** The idea of expressing code size in terms of lines is one that stakeholders, even those who are not technically inclined, can easily understand.

Advantages of Lines of Code (LOC)

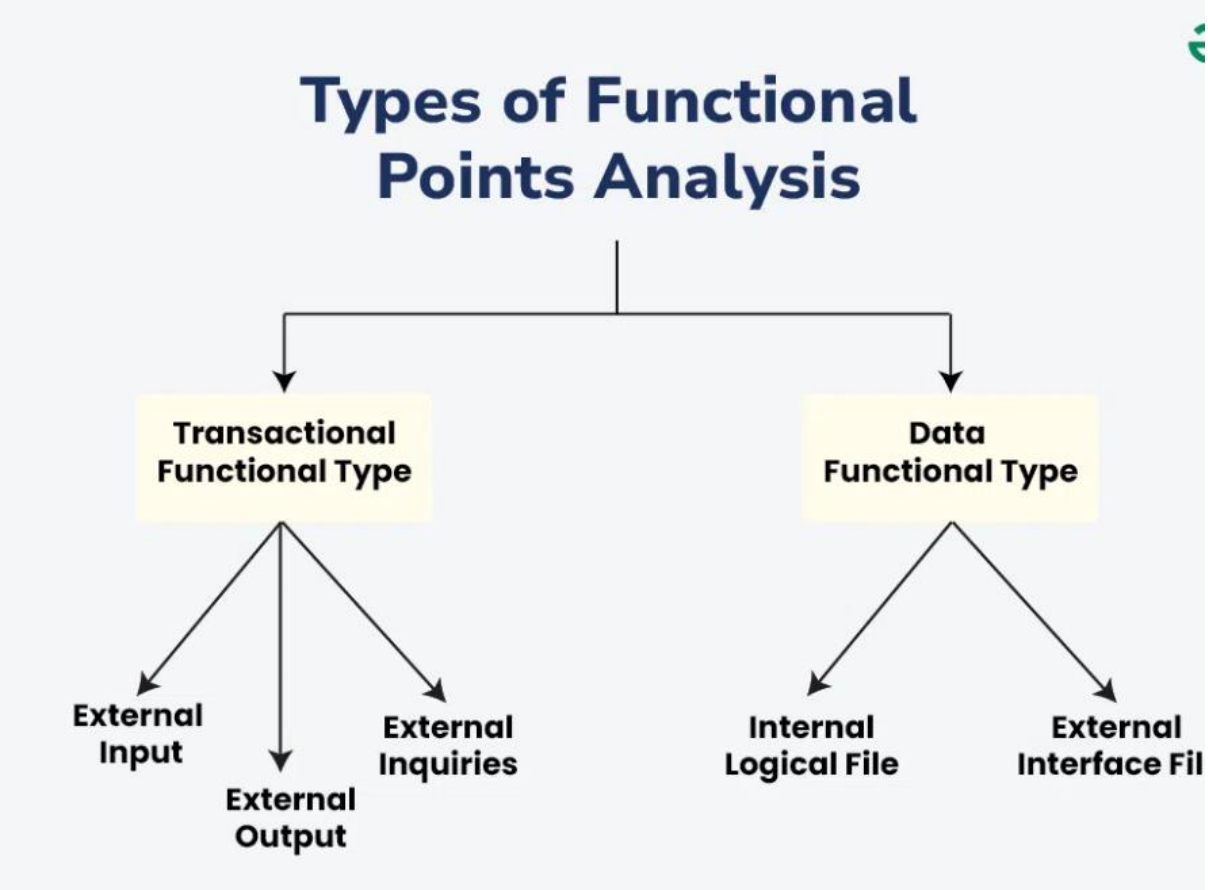
- Cost estimation
- Comparative Analysis
- Benchmarking tool

DISADVANTAGES:

- Challenges in Agile Work Environments
- **Not Considering Into Account External Libraries**
- **Challenges with Maintenance:**

FUNCTION POINT METRIC:

- Functional Point Analysis (FPA) is a software measurement technique used to assess the size and complexity of a software system based on its functionality.
- It involves categorizing the functions of the software, such as input screens, output reports, inquiries, files, and interfaces, and assigning weights to each based on their complexity. By quantifying these functions and their associated weights, FPA provides an objective measure of the software's size and complexity.


Types of Functional Points Analysis
Transactional Functional Type
Data Functional Type
External Input
External Output
External Inquiries
Internal Logical File
External Interface Fil

ADVANTAGES:

- **Technological Independence**
- **Better Accurate Project Estimation**
- **Improved Interaction**
- **Making Well-Informed Decisions**

DISADVANTAGES:

- **Low Accuracy**
- **Steep Learning Curve**
- **Time Consuming**
- **Cost**