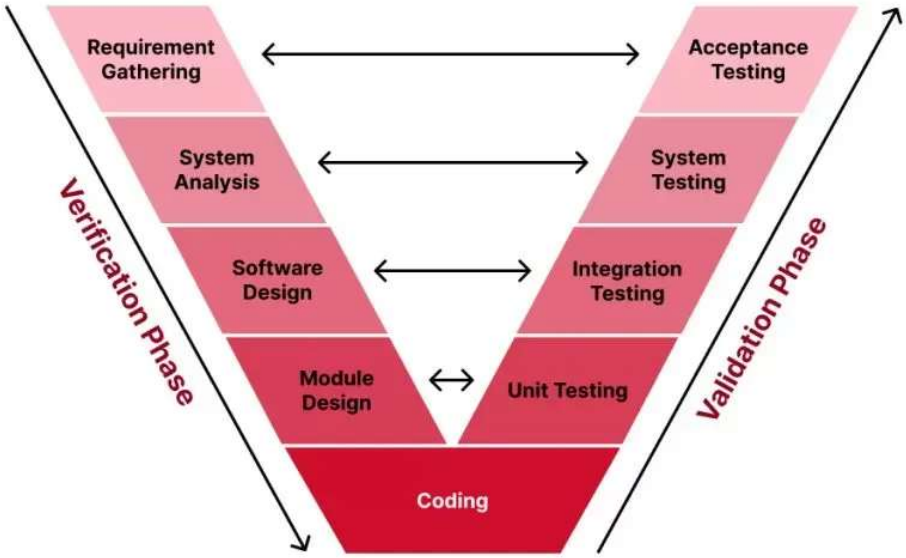
# Software Engineering

## V - Shaped Model –

**What is V-Model?**

- **V-Model (Verification and Validation Model) is an extension of the Waterfall Model.**
- **Each development phase is directly associated with a testing phase.**
- **Shaped like a "V" to show the development (left side) and testing (right side) activities.**
- **Focuses on early testing and defect prevention.**

**Phases of V-Model**




- **Verification Phases (Left Side of "V")**

1. **Business Requirement Analysis**
    - **Gather customer requirements.**

- Plan acceptance test cases.

2. System Design
    - Create high-level and detailed system design.
    - Clear understanding of product requirements.
3. Architectural Design (High-Level Design - HLD)
    - Decide system architecture.
    - Break system into modules.
    - Plan integration testing.
4. Module Design (Low-Level Design - LLD)
    - Design internal module logic.
    - Create unit test plans.
5. Coding Phase
    - Actual code writing.
    - Code reviews and optimizations.

---

- Validation Phases (Right Side of "V")

1. Unit Testing
    - Test individual modules.
    - Based on unit test plans.
2. Integration Testing
    - Test interactions between modules.
    - Focus on data flow and interfaces.
3. System Testing
    - Test entire application.
    - Functional and non-functional testing.

4. **User Acceptance Testing (UAT)**
    - **Final user-based testing.**
    - **Done in an environment like production.**

---

**Industrial Challenges**

- **Need for accurate requirement gathering.**
- **Building according to authorized requirements.**
- **Validating that the product meets business needs.**

---

**Importance of V-Model**

- **Early defect identification.**
- **Phase-wise testing mapped to development.**
- **Avoids "Big Bang" testing at the end.**
- **Improves team collaboration.**
- **Strengthens quality assurance.**

---

**Principles of V-Model**

- **Large to Small: Broad to detailed design.**
- **Data/Process Integrity: Consistency in design and processes.**
- **Scalability: Suitable for small to large projects.**
- **Cross Referencing: Requirements linked to tests.**
- **Tangible Documentation: Complete documentation at every stage.**

---

**When to Use V-Model**

- **Projects needing clear requirement traceability.**

- **Complex systems with interdependencies.**
- **Sequential (Waterfall-like) projects.**
- **Safety-critical systems (e.g., healthcare, aerospace).**

---

**Advantages**

- **Simple and easy to understand.**
- **Structured and disciplined process.**
- **Early error detection.**
- **Better tracking of project progress.**
- **Strong emphasis on testing and quality.**
- **Good traceability between requirements and product.**

---

**Disadvantages**

- **Rigid (not flexible for requirement changes).**
- **Time-consuming (lots of documentation and testing).**
- **Not suitable for:**
  - **Complex or object-oriented projects.**
  - **Projects with unclear requirements.**
- **No phase iteration (linear process).**

---

📝 **Conclusion**

- **V-Model is effective for small to medium-sized, clearly defined projects.**
- **Offers a structured approach for ensuring quality software through early and continuous testing.**
- **Best suited for stable environments with fixed requirements.**

## Spiral Model –

**What is Spiral Model**

- The **Spiral Model** is an SDLC model combining **iterative** development and **systematic** risk management.
- Diagram looks like a **spiral with multiple loops** (phases); the number of loops **varies** per project.
- Best for **large, complex**, and **high-risk** projects.
- This Model also Called Meta Model .

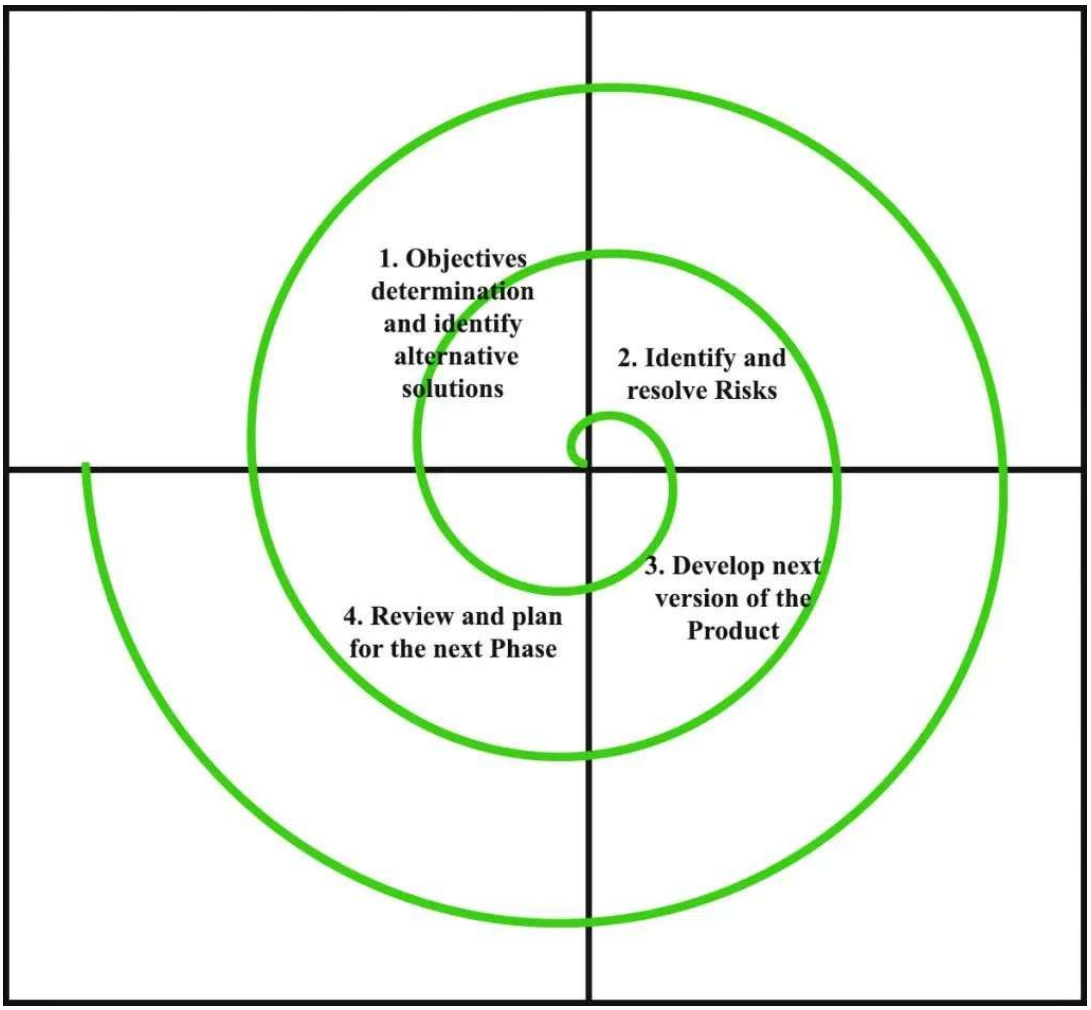
◆ **Key Points**

- **Dynamic number of phases** based on project risks.
- **Project Manager** plays a critical role in adjusting phases.
- Each loop = one full cycle: **requirements → design → implementation → testing → evaluation**.

◆ **Phases (Four Quadrants)**

1. **Objectives Defined**
    - Identify functional and non-functional requirements.
    - Propose alternative solutions.
2. **Risk Analysis and Resolving**
    - Analyze and resolve risks.
    - Build a **prototype** for best solution.
3. **Develop Next Version**
    - Develop features.
    - Test and validate.
4. **Review and Plan Next Phase**

- Customer evaluates current version.
- Plan the next iteration.




**Radius** of spiral = **cost so far**, **Angle** = **progress**.

- **Risk Handling**
  - Spiral Model supports **identifying and resolving risks** at every stage.
  - **Prototyping** helps manage unknown risks **after** development starts.
  - More flexible than traditional Prototyping Model.

- **Real-Life Example: E-Commerce Website**

- **First Spiral:** Gather basic requirements, build simple homepage prototype.
- **Second Spiral:** Refine design, add secure payment and shopping cart.
- **Third Spiral:** Implement order tracking, search, and test scalability.
- **Fourth Spiral:** Full deployment, monitor for issues.

🔹 **Advantages**

- Excellent **Risk Handling**.
- Ideal for **large projects**.
- **Flexible** to changes in requirements.
- **Customer Satisfaction** via early feedback.
- **Iterative and Incremental** approach.
- Focus on **Improved Quality** and **Communication**.

🔹 **Disadvantages**

- **Complex** and **Expensive**.
- Highly dependent on **Expert Risk Analysis**.
- **Time Estimation** is difficult.
- **Resource and Time Intensive**.

🔹 **When to Use**

- Large and complex projects.
- Projects requiring **frequent releases**.
- Projects with **moderate to high risk**.
- When **requirements are unclear**.
- When **prototyping** is essential.
- When economic priorities **may change** during project.

🔹 **Conclusion**

- Spiral Model is **risk-driven** and highly **flexible**.
- Ensures **continuous improvement** with **customer feedback**.
- Suitable for **high-priority risk management** and **large-scale** development.

## Agile Model -

**Definition**

The Agile Model was created to help projects quickly adapt to change requests and complete faster. Agility is achieved by customizing processes to the project and removing unnecessary activities, avoiding waste of time and effort. It refers to a group of development processes that share basic characteristics but differ subtly.

---

**Steps in the Agile Model**

**1. Requirement Gathering**

- Gather requirements through customer interaction.
- Plan the time and effort needed.
- Evaluate technical and economic feasibility.

**2. Design the Requirements**

- Create system architecture based on gathered requirements.
- Wireframe the user interfaces for better navigation and user-friendliness.
- Create high-level UML diagrams to show structure and flow.
- Develop prototypes to collect early feedback.

**3. Construction / Iteration**

- Begin coding based on planned features.
- Work in short cycles (1-4 weeks).
- Deliver a working product at the end of each cycle.

- Improve the product incrementally with each iteration.

**4. Testing / Quality Assurance**

- Unit Testing: Testing small units of code individually.
- Integration Testing: Checking the interaction between combined units.
- System Testing: Verifying the software meets user requirements in all scenarios.

**5. Deployment**

- Deploy working software after testing.
- Make the software accessible to end-users.
- Handle any deployment issues quickly.

**6. Feedback**

- Gather feedback from users, stakeholders, and customers.
- Identify bugs or improvements.
- Adjust the plan and software based on feedback.

---

**Agile SDLC Methods**

- **Test-Driven Development (TDD)**: Write unit tests before writing code.
- **Behavior Driven Development (BDD)**: Develop based on expected user behavior.
- **Exploratory Testing**: Testers freely explore the software to find unknown issues.
- **Acceptance Test-Driven Development (ATDD)**: Customers, developers, and testers discuss requirements together before coding.
- **Extreme Programming (XP)**: Focus on customer satisfaction and high-quality output.
- **Session-Based Testing**: Conduct uninterrupted, time-boxed testing sessions.

- **Dynamic Software Development Method (DSDM)**: Provides a framework for building and maintaining systems collaboratively.
- **Crystal Methodologies**: Focuses on people and interactions rather than processes and tools.

---

**Principles of the Agile Model (Based on Agile Manifesto)**

1. Satisfy the client through early and continuous delivery.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently.
4. Business people and developers must work together daily.
5. Build projects around motivated individuals.
6. Face-to-face conversation is the best way to communicate.
7. Working software is the primary measure of progress.
8. Maintain a constant pace of development indefinitely.
9. Focus on technical excellence and good design.
10. Simplicity is essential (maximize the amount of work not done).
11. The best designs emerge from self-organizing teams.
12. Reflect and adjust regularly for better efficiency.

---

**When to Use the Agile Model**

- When frequent changes and updates are expected.
- When low-cost implementation of new features is important.
- When limited upfront planning is desired.
- When dynamic requirements are anticipated.
- When collaboration and continuous feedback are priorities.

---

**Advantages of the Agile Model**

- Pair programming leads to well-written, error-free code.
- Reduces overall development time.
- Improves collaboration through face-to-face communication.
- Frequent product updates allow easier requirement changes.
- Ensures the product meets user needs.

---

**Disadvantages of the Agile Model**

- Lack of formal documentation may cause confusion.
- Not suitable for projects with complex dependencies.
- Heavy reliance on customer clarity and interaction.
- Difficult to forecast project timelines and costs.
- Requires high expertise from team members.
- Maintenance becomes hard without documentation.

---

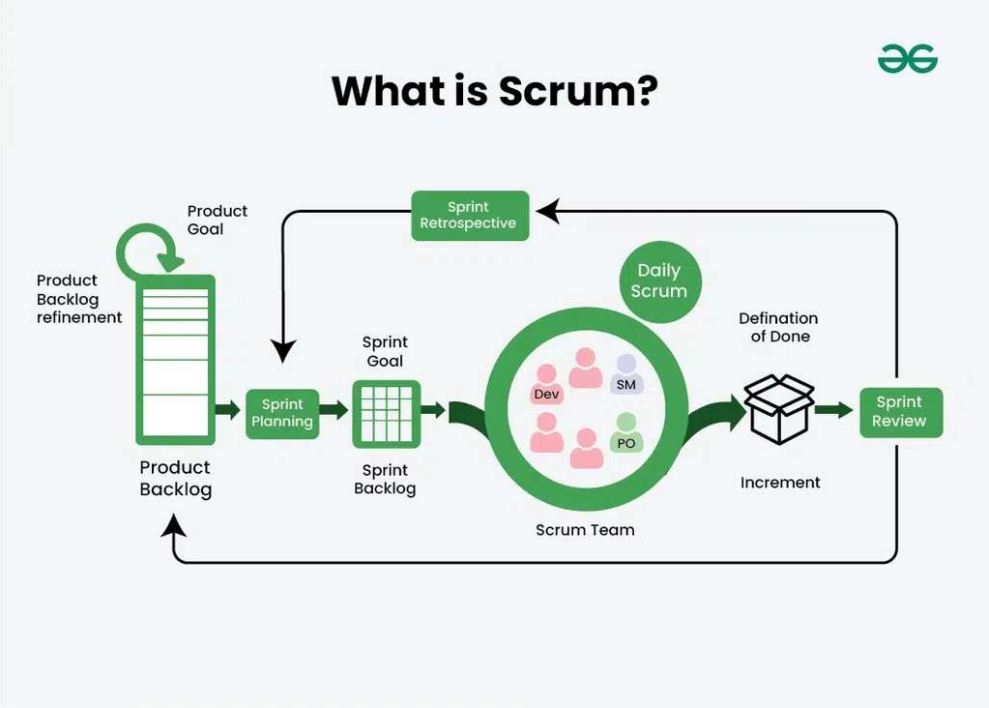
**Conclusion**

Agile focuses on flexibility, teamwork, and continuous improvement. It helps in building software that easily adapts to customer requirements and changes, emphasizing the delivery of real value to the users.

**SCRUM –**

Scrum is a way of managing projects, especially in software development. It's like a playbook that teams use to work together more effectively. Instead of doing everything at once, Scrum breaks work into smaller parts called "sprints." Each sprint focuses on completing a specific piece of the project, allowing

teams to adapt and improve as they go. It's all about teamwork, communication, and getting things done step by step.




**The Scrum Framework**

The Scrum framework is a simple, agile way to manage projects that helps teams work together to create high-quality products quickly. Here are the basic components and principles of Scrum, explained in simple terms.

**Scrum Team**

- **Product Owner:** Decides what the team should work on and makes sure it meets business and customer needs.
- **Scrum Master:** Helps the team follow Scrum rules and removes any obstacles they face.
- **Development Team:** A group of people with different skills who work together to build the product.

**Artifacts:**

- **Product Backlog**: A list of everything that needs to be done for the product, organized by priority.
- **Sprint Backlog:** A list of tasks the team plans to complete during the current sprint.
- **Increment:** The finished work at the end of a sprint that could be released to users.

**Events:**

- **Sprint:** A short, fixed period (usually 1-4 weeks) where the team works on specific tasks from the sprint backlog.
- **Sprint Planning:** A meeting at the start of the sprint where the team decides what to work on.
- **Daily Scrum:** A quick daily meeting where the team checks in on progress and plans the day's work.
- **Sprint Review:** A meeting at the end of the sprint where the team shows what they've accomplished and gets feedback.
- **Sprint Retrospective:** A meeting at the end of the sprint where the team discusses what went well and what could be improved for the next sprint.

What is Sprint Process?

Step 1: The scrum cycle is a complete cycle of product development which starts with the Product Owner, the product owner defines all the product backlog and requirements for further development.

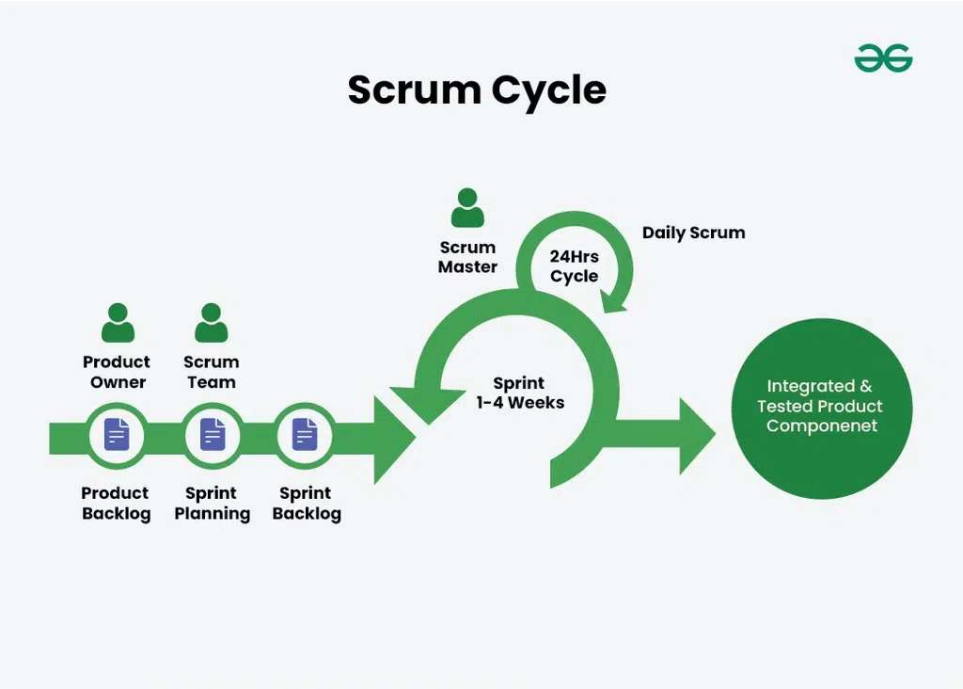
Step 2: The next step is handled by the scrum team and the scrum team manages the Sprint Planning in the sprint planning phase the team plans all the tasks and roles for the development of the product.

Step 3: Then in the next phase the team deals with the sprint backlogs which are due and deals accordingly with all the backlogs left.

Step 4: As we know the scrum master is someone who overseas all the tasks of the team and acts as a mentor for the team, so in this step the scrum master overseas and collaborates with the team to make sure the product is ready on time, this cycle typically lasts from 1 to 4 weeks but depending on the project size it varies a lot, which is why sometimes for larger projects, it's also divided into smaller part known as Daily Scrum.

Step 5: This is the final process stage, in this stage the team integrates all the modules of the project and tests the various components of the product.

Below is the visual representation of the Scrum's sprint process:




**Advantages of Scrum (Bullet Points)**

- Enables **faster and iterative product delivery** through short sprints.
- Promotes **better team collaboration** via daily stand-ups and regular sprint planning.
- Time - Efficient
- Increases **transparency and visibility** of project progress.

- Budget Friendly Planning

**Limitations of Scrum (Bullet Points)**

- Not Suitable for all Projects
- Requires Skilled and Experienced Team
- Limited Predictability
- Lack of detailed documentation
- Risk of Team Burnout

| Parameters | Agile | Scrum |
|---|---|---|
| **Methodology** | Agile is a set of principles that's iterative and incremental. | Scrum is an implementation of the agile methodology. |
| **Projects** | Suited for projects involving a small team of experts. | They are used in projects where the requirements are constantly changing. |
| **Leadership** | The project head takes care of all tasks is vital to the project. | There's no leader, the scrum master, and the team addresses the issues. It involves cross-functional, self-organizing teams. |

| Parameters | Agile | Scrum |
| --- | --- | --- |
| **Flexibility** | In agile, changes cannot be handled frequently. | It enables teams to react to changes quickly. |
| **Delivery** | The methodology requires frequent delivery to the end user. | With sprints, builds are delivered to clients for feedback. |
| **Collaboration** | Face-to-face interaction takes place between cross-functional teams. | Daily stand-up meetings help with collaboration. |

# Software Requirement Specifications ( SRS )

**Requirement Analysis –**

Requirement Analysis is the process of understanding, gathering, analyzing, and documenting what a project or system must do to satisfy the needs of users and stakeholders.

It’s a critical step because if you misunderstand the requirements, no matter how good the design or code is, the final product won't be what the client wants.

**Barriers to Effective Requirement Analysis**

Here are the major problems (barriers) people often face:

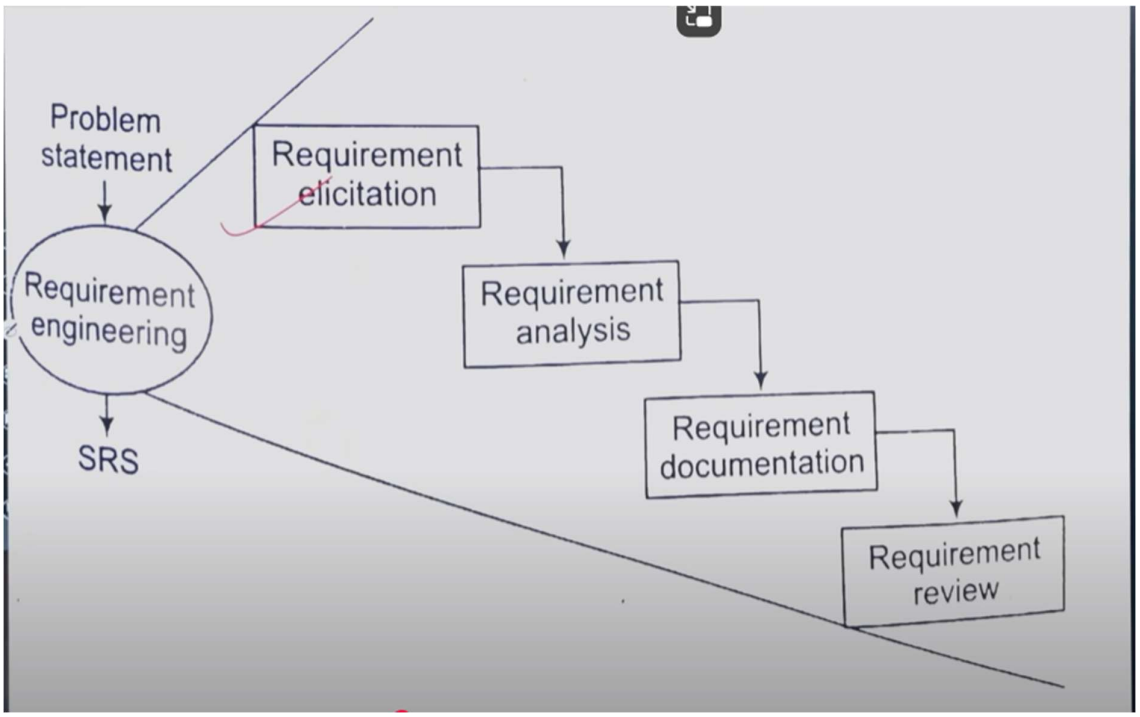
1. **Requirements are difficult to uncover**
    - No one Give the complete requirement in the first time.
2. **Changing Requirements (Scope Creep)**
    - New features keep getting added during the project without adjusting the timeline or budget.
3. **Communication Gaps**
    - Users and Developer have different technical background and tastes.
    - Stakeholders might explain things differently or developers might misunderstand technical jargon.
4. **Tight Project Schedule**
    - Have Insufficient time to do a decent job.
5. **Unclear Business Goals**
    - If the organization itself is not clear on what they want, the requirements will naturally be messy.
6. **Lack of Resources**

- There may not be enough resources to build software that can do everything the customer wants.

7. **Stakeholder Conflicts**
    - Different stakeholders want different or even **conflicting** things, and nobody agrees.

**<u>Types of Requirements –</u>**

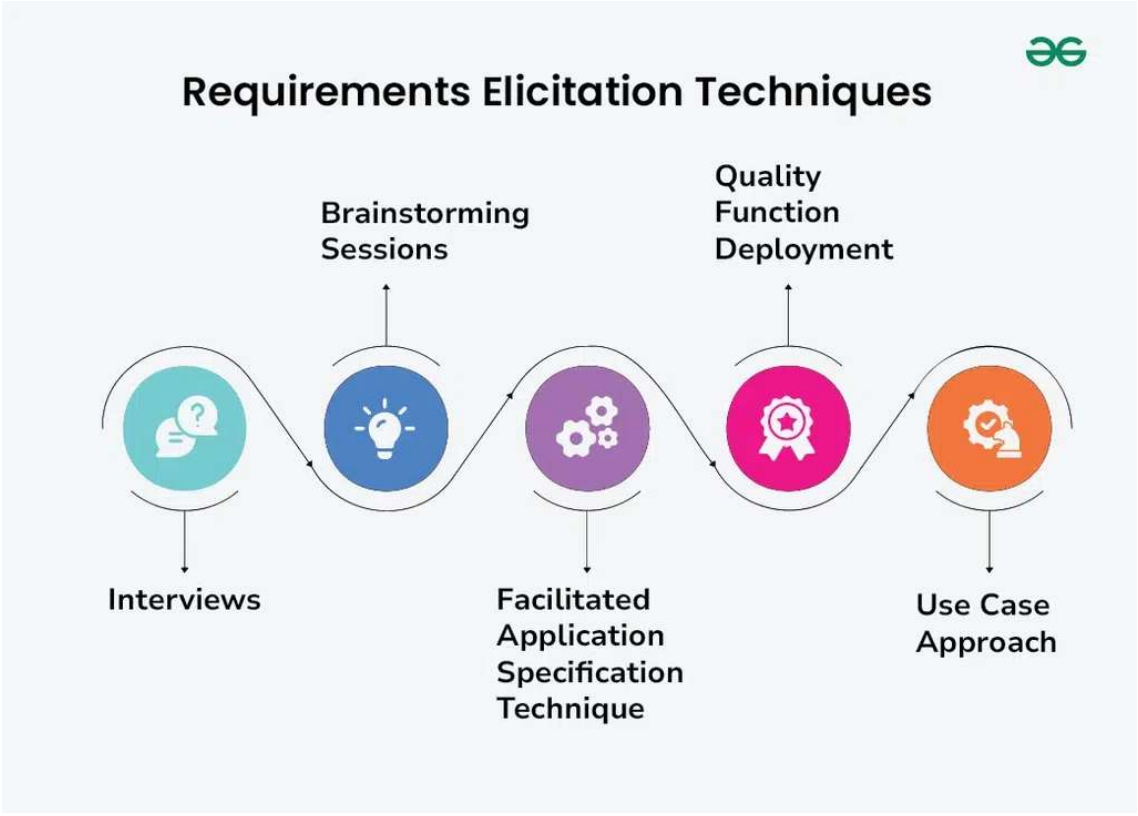
1. **Known Requirement :** which is already known to the stake holder.
2. **Unknown Requirement :** Forgotten by stake holder because that are not needed right now.
3. **Undreamt Requirement :** Stakeholder is unable to think of new requirement due to limited domain knowledge.



SRS contains following steps :

1. Requirement Elicitation – Gathering of Requirement
2. Requirement Analysis - Requirements are analysed to identify inconsistences , detects etc and to resolve conflicts.
3. Requirement documentation – Here we document all the refined requirement properly.
4. Requirement Review – Review is carried out to improve the quality of SRS.

**Methods of Requirement Elicitation –**



**1. Interviews**

The objective of conducting an interview is to understand the customer's expectations of the software.
It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility. Interviews may be open-ended or structured.

1. In open-ended interviews, there is no pre-set agenda. Context-free questions may be asked to understand the problem.
2. In a structured interview, an agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

**2. Brainstorming Sessions**

- Brainstorming Sessions is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

---

**3. Delphi Technique**

Here Participants are made to write the requirement on a piece of paper then these requirements are exchanged among participants who gave their comments to get a revised set of requirements. This Process is repeated till the final consensus is reached .

---

**4. Facilitated Application Specification Technique (FAST)**

- **Objective**: Bridge the **expectation gap** between developers and customers.
- **Approach**:
    - Team-oriented requirements gathering.
    - Each participant lists:
        - Objects part of the system's environment.
        - Objects produced by the system.
        - Objects used by the system.
    - Combine lists ➔ Remove redundancies ➔ Form mini-specifications in sub-teams ➔ Prepare a draft specification document.

---

**5. Quality Function Deployment (QFD)**

- **Objective**: Focus on **customer satisfaction** by prioritizing valuable requirements.
- **Types of Requirements**:
    - **Normal Requirements**:
        - Basic goals and objectives stated by customers.
        - *(e.g., entry of marks, result calculation in a result management system)*.
    - **Expected Requirements**:
        - Obvious features not explicitly stated but expected.
        - *(e.g., security from unauthorized access)*.
    - **Exciting Requirements**:
        - Surprising features that delight customers.

        - *(e.g., automatic backup on unauthorized access detection)*.

---

**6. Use Case Approach**

- **Objective**:
    - Combine **text and diagrams** to describe system functionality clearly.
- **Focus**:
    - Describes *"what"* the system does, not *"how"* it does it (functional view).
- **Components**:
    - **Actor**:
        - External entity interacting with the system (person, machine, etc.).
        - Represented as a **stick figure**.
        - *Primary Actor*: Needs system's help to achieve a goal.
        - *Secondary Actor*: System needs help from this actor.
    - **Use Case**:
        - Describes sequence of interactions between actor and system.
        - Specifies all ways users interact with the system.
    - **Use Case Diagram**:
        - Graphical representation:
            - **Stick figure** = Actor
            - **Oval** = Use case
            - **Line** = Relationship between actor and use case

**Steps of Requirements Elicitation**

1. **Identify Stakeholders**
    - Find users, developers, managers, customers, etc.
    - Early involvement is crucial.
2. **Gather Requirements**
    - Collect functional and non-functional needs through interviews, surveys, discussions.
3. **Prioritize Requirements**
    - Categorize as Must-have, Should-have, Could-have, Won't-have.
4. **Categorize Feasibility**
    - **Achievable**: Can be done within resources.
    - **Deferred**: Important but delayed.
    - **Impossible**: Unachievable, dropped.

---

**Features of Requirements Elicitation**

- Stakeholder engagement.
- Information gathering about system and users.
- Requirement prioritization.
- Clear documentation.
- Validation and verification.
- Iterative (refined continuously).
- Strong communication and collaboration.

- Flexibility to adapt to changes.

---

**Advantages**

- Clear and refined requirements.
- Improved stakeholder communication.
- Higher software quality.
- Avoids misunderstandings.
- Early risk identification.
- Helps accurate planning.
- Boosts user confidence.
- Finds new business opportunities.

---

**Disadvantages**

- Time-consuming and costly.
- Needs skilled professionals.
- Sensitive to changing needs.
- Affected by politics/organization issues.
- Risk of stakeholder disengagement.
- Conflicting priorities possible.
- Chance of incomplete/inaccurate requirements.
- May increase development costs.

## Requirement Analysis

Once Requirement Elicitation is complete (i.e., after gathering all the raw requirements from stakeholders), Requirement Analysis begins.

Requirement Analysis is about:

- Organizing,
- Clarifying,
- Understanding deeply, and
- Refining
  the collected requirements to make sure they are correct, complete, consistent, and feasible.

**Tools –**

1. **Data Flow Diagram -**

- A data flow diagram or bubble chart is  a graphical representation of the flow of data through a system . It clarifies systems requirements and identify major transformations.
- DFD represent a system at different level of abstraction . DFD may be partitioned into levels that represent increasing information flowand functional details.

**Main Components**:

- **Process**: Work done on data (drawn as a circle or rounded rectangle).
- **Data Flow**: Movement of data between parts (drawn as arrows).
- **Data Store**: Storage place for data (drawn as open-ended rectangles).
- **External Entity**: People/systems outside the boundary (drawn as squares).

**Levels**:

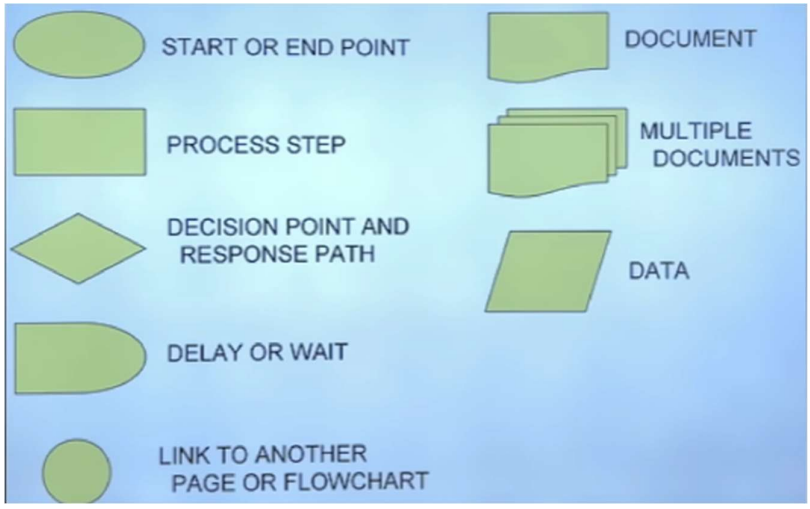
- **Level 0**: High-level overview (called **Context Diagram**).
- **Level 1, Level 2, etc.**: Breaks processes into detailed sub-processes.

**Advantages**:

- Easy to understand even without technical background.
- Helps visualize and improve business processes.
- Detects missing requirements early.

2. **<u>Control Flow Diagram / Control Flow Chart / Flow Chart</u>**

A flow chart is a graphical representation of how control flow during the execution of a program. It use the following symbols to represent a system's control flow.




**Focus :**
**What actions** happen and **in what order** — step-by-step instructions.

**Main Elements**:

- **Start/End**: Represented by an **oval**.
  (e.g., Start, End)
- **Process/Action**: Represented by a **rectangle**.
  (e.g., "Add two numbers")
- **Decision/Condition**: Represented by a **diamond**.
  (e.g., "Is X > 0?")

- **Arrows**: Show the **flow direction** between steps.

**Use Cases**:

- Describing algorithms.
- Representing business processes.
- Helping developers understand logic before coding.

**Advantages**:

- Simple and easy to follow.
- Highlights all possible decision paths.
- Good for spotting mistakes and logical errors early.

3. **Data Dictionary –**

A Data Dictionary is a central repository that stores definitions, formats, and descriptions of all the data elements in a system.

Purpose:
➔ To standardize and document information about the data — ensuring everyone understands it the same way.

What it includes:

- Data element name (e.g., User_ID)
- Description (e.g., "Unique identifier for a user")
- Data type (e.g., Integer, String, Date)
- Allowed values/ranges (e.g., "User type: Admin, Customer")
- Default values (if any)
- Relationships (e.g., "User_ID" links to "Orders" table)

Why it’s important:

- Avoids confusion about data meanings.
- Helps developers, analysts, and testers use data consistently.

- Makes it easier to update, maintain, or expand the system later.

When it's used:

- During system design.
- While writing requirements and specifications.
- During database development.

4. **Entity Relationship Diagram –**

An ER Diagram is a visual representation of entities (things) and their relationships in a system.

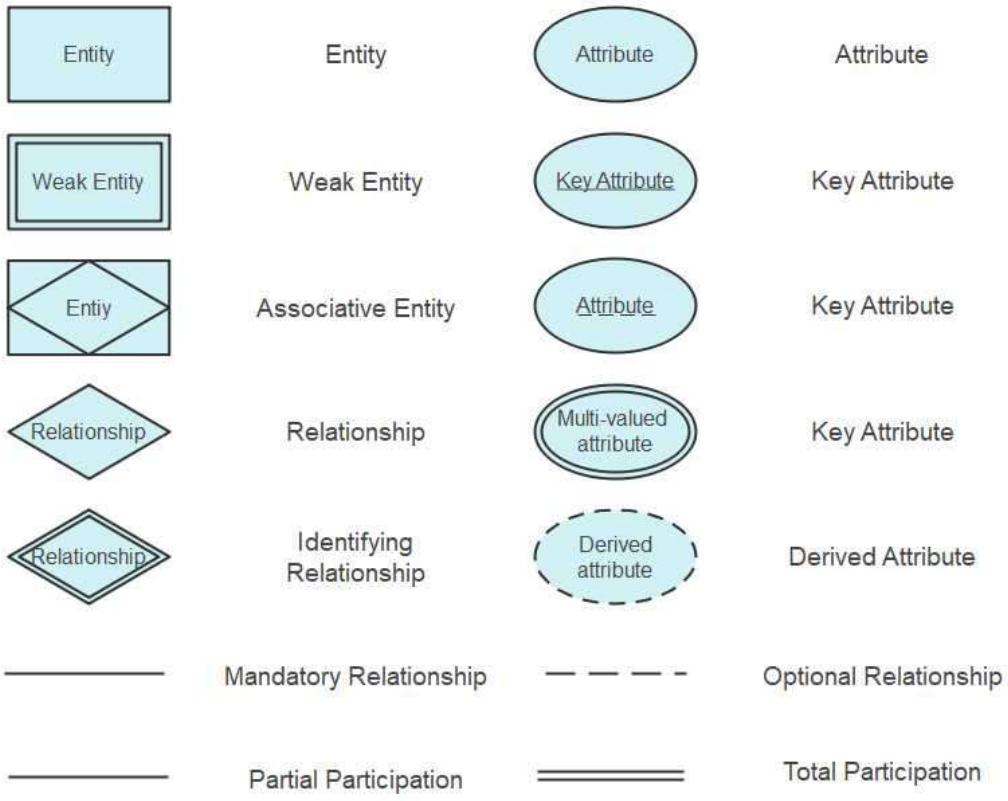
Purpose:
➔ To design and organize the structure of a database clearly before creating it.

Main Components:

- Entity:
  ➔ Objects or concepts (e.g., Student, Course).
  ➔ Represented by rectangles.
- Attribute:
  ➔ Properties or details about entities (e.g., Student has Name, Roll_No).
  ➔ Represented by ovals.
- Relationship:
  ➔ Describes how entities are related (e.g., Student enrolls in Course).
  ➔ Represented by diamonds.

Why it's important:

- Helps in planning databases.
- Makes complex systems easier to understand.
- Ensures data integrity.




**5. <u>Decision Tables –</u>**

A decision table is a tabular method to represent different conditions and the actions that follow from them.

Purpose:

➔ To systematically list all possible combinations of inputs (conditions) and their corresponding outputs (actions).

➔ Helps in complex decision-making where multiple conditions exist.

Structure:

Condition Stubs → List of all the conditions (inputs).

Action Stubs → List of all possible actions (outputs).

Condition Entries → Different combinations of true/false for each condition.

Action Entries → Action to be taken for each condition combination.

| **Conditions** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
|---|---|---|---|---|
| Condition 1 | Y | Y | N | N |
| Condition 2 | Y | N | Y | N |
| ---------- | -------- | -------- | -------- | -------- |
| Actions | Action A | Action B | Action C | Action D |

| **Aspect** | **Decision Table** | **Decision Tree** |
|---|---|---|
| **Structure** | Tabular format, lists conditions, actions, and rules. | Tree-like diagram, shows decision paths with branches. |
| **Decision Making** | Based on various combinations of conditions leading to specific actions. | Based on sequential decisions leading to outcomes. |
| **Ease of Use** | Simple for representing complex logical relationships, but can become cumbersome with many variables. | More intuitive for visualizing decision paths, but can become complex with many branches. |
| **Application** | Preferred when all possible combinations of conditions and outcomes need clear representation. | Ideal for situations where decision-making is more sequential and hierarchical. |

## Requirement Documentation

IEEE standards for SRS : IEEE830

IEEE standards for SRS –

1 – Introduction

- 1.1 Purpose
- 1.2 Scope / Intended Audience
- 1.3 Definition , Acronyms and Abbreviation
- 1.4 References / Contact Information / SRS Team member
- 1.5 Overview

2 Overall Description

- 2.1 Product Perspective
- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 General Constraints
- 2.5 Assumptions and dependencies

3 Specific Requirement

- 3.1 External Interface Requirement ( User / hardware / software interface )
- 3.2 Functional Requirements
- 3.3 Performance Requirements
- 3.4 Design Constraints ( Standards Compliance / hardware limitations )
- 3.5 Logical database Requirements
- 3.6 Software System attributes ( Reliability , availability , Security , Maintainability )

4 Change Management Process

5 Documents Approval

- 5.1 Tables , diagrams , and flowcharts
- 5.2 Appendices
- 5.3 Index

Verification and Validation :

Key Activities -
1. Tracking and Controlling Changes
2. Version Control
3. Traceability
4. Communication
5. Monitoring and Reporting

## **Software Quality Assurance –**

- SQA is a process ensuring software products meet quality standards and requirements.
- Objectives – Prevent Defects , Improve Quality , ensure customer satisfaction.
- Process – Implement Quality control and management activities throughout development.
- Techniques – Uses code reviews , inspections , audits , and testing to evaluate software quality .
- Standards – Adheres to international standards like ISO 9001 , IEEE 730 , and CMMI .
- Quality Attributes – Focuses on reliability , maintainability , usability , efficiency , and functionality.
- Continuous Improvements – Monitors Processes and products to identify and implement improvements.
- Metrics – Employs metrics like defect density and code coverage to evaluate quality.
- Training and Documentation


## Validation / Black Box –

What it is:
Checks whether the right product is built (i.e., are we building what the user actually wants?).

Focus:
Focuses on the external behavior of the software — inputs and outputs, without looking at the internal code.

Example:
Testing a login page by entering username and password without worrying how the backend code works.

Also known as:
Black Box Testing because the internal logic is hidden.

## Verification / White Box –

**What it is:**
Checks **whether the product is built right** (i.e., is the product being built correctly according to specifications?).

**Focus:**
Focuses on the **internal structure, design, and coding** — how the code works.

**Example:**
Checking if the login function correctly hashes passwords and handles errors in the code.

**Also known as:**
**White Box Testing** because the internal workings are visible and tested.

- Validation = Are we building the right product? (Black Box)
- Verification = Are we building the product right? (White Box)

## Software Quality Factors –

The various factors, which influence the software, are termed as software factors. They can be broadly divided into two categories.

- The first category of the factors is of those that can be measured directly such as the number of logical errors.
- Second category clubs those factors which can be measured only indirectly. For example, maintainability but each of the factors is to be measured to check for the content and the quality control.

Several models of software quality factors and their categorization have been suggested over the years. The classic model of software quality factors, suggested by McCall in 1977). The 11 factors are grouped into three categories.

- Product operation factors – Correctness, Reliability, Efficiency, Integrity, Usability.
- Product revision factors – Maintainability, Flexibility, Testability.
- Product transition factors – Portability, Reusability, Interoperability.