

1. Explain Software with its characteristics

Ans: Software is a collection of programs, data, and documentation that perform specific tasks on a computer system. It enables users to interact with hardware and perform desired functions. Computer software is the product that software professionals build and then support over the long term. It also includes a set of documents, such as the software manual, meant for users to understand the software system. Today's software comprises the source code, Executable, Design Documents, Operations and System Manuals and installation and Implementation Manuals.

Characteristics of Software:

1. Software is Developed or Engineered, Software is not manufactured:

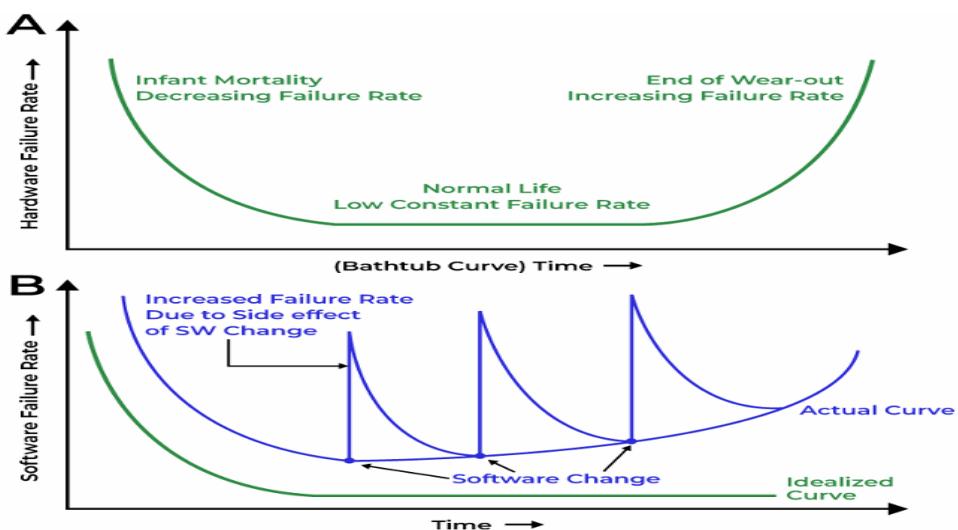
It is developed through design and coding. The two activities (software development and hardware manufacturing) are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems.

2. Software doesn't "wear out" like hardware and it is not degradable over a period.

The following figure depicts failure rate as a function of time for hardware. The relationship often called the "bathtub curve", indicates that hardware exhibits relatively high failure rates early in its life. The failure rate curve for software should take the form of an "idealized curve" as shown in the above figure. Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. Hence the software doesn't wear out, but it does deteriorate.

3. Most of the Software are Custom-built or generic rather than assembled from existing components:

Software can be tailored or commercially available. Although the industry is moving toward component – based construction, most software continues to be custom built. A software component should be designed and implemented so that it can be reused in many different programs

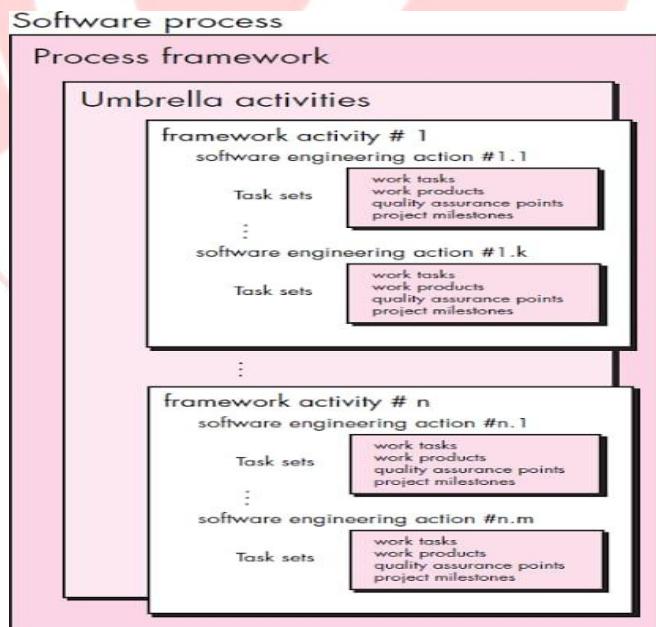


Other Characteristics are:

4. Complex: Often contains millions of lines of code.
5. Easy to modify: Can be updated with new features or bug fixes.
6. Intangible

2. Describe Generic process Framework activities and Umbrella activities.

Ans. These are the core activities that are part of almost every software development process, regardless of the development model (Waterfall, Agile, Spiral, etc.).



Communication

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Involves interacting with stakeholders to understand the requirements.
- Includes activities like requirement gathering, meetings, and interviews.

Planning

- Defines a roadmap for the project.
- Includes estimating time, cost, resources, and setting milestones.

Modeling

- Focuses on designing the architecture and system structure.
- Includes data modeling, process modeling, interface design, etc.

Construction

- Involves actual coding and testing of the software.
- Focuses on writing source code, unit testing, integration testing, etc.

Deployment

- Involves delivering the final product to the end-user.
- May include installation, user training, and feedback collection.
- Updates and patches are handled during post-deployment maintenance.

Umbrella Activities

Umbrella Activities in Software Engineering Umbrella activities are supportive tasks that occur throughout the software development life cycle (SDLC). They ensure that the core development activities are well-managed and high in quality.

✓1. Software Project Tracking and Control

- Monitors project progress to ensure it's on time and within budget.
- Helps take corrective actions if deviations occur.

✓2. Risk Management

- Identifies, analyzes, and prepares for potential risks that could impact the project.
- Plans strategies to reduce or avoid risks.

✓3. Software Quality Assurance (SQA)

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Ensures the software meets the defined quality standards.

- Includes activities like audits, reviews, and testing.

✓4. Formal Technical Reviews (FTRs)

- Structured reviews of software artifacts (design, code, documents)

Helps in early error detection and correction.

✓5. Measurement and Metrics

- Collects quantitative data to assess process and product quality.
- Used for improvement, estimation, and evaluation.

✓6. Software Configuration Management (SCM)

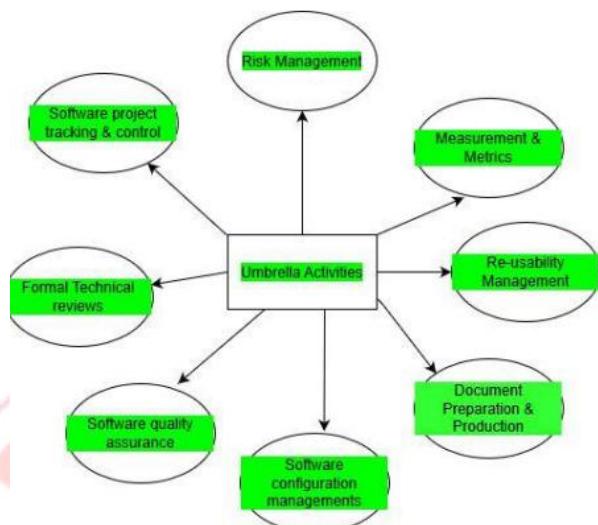
- Manages changes in software versions and components.
- Maintains integrity and traceability of project artifacts.

✓7. Reusability Management

- Identifies and promotes reuse of existing software components.
- Saves time and improves reliability.

✓8. Documentation

- Ensures proper recording of all development activities, plans, and decisions
- Helps in maintenance, training, and future development.



3. Describe the types and categories of software.

Ans: 1. System Software: It is a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) processes complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data.

2. Application Software: Stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making

3. Programming Software: Used by developers to write code. Examples: Compilers, editors, debuggers.

4. Embedded Software: Embedded software can perform limited functions (e.g., key pad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems). Used in embedded systems (non-PC devices). Examples: Software in washing machines, traffic lights.

5. Web-based Software: In their simplest form, WebApps can be little more than a set of linked hypertext files that present information using text and limited graphics. Runs on web browsers and accessed via the internet. Examples: Gmail, Google Docs.

6. AI Software: Designed to simulate intelligence. Examples: Chatbots, recommendation systems.

7. Engineering/scientific software: Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing Example: CAD software

8. Product-line software: Designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, database management).

4. Explain Software Engineering as a layered approach.

Ans: Software Engineering as a Layered Approach

1. Quality Focus (Base Layer)

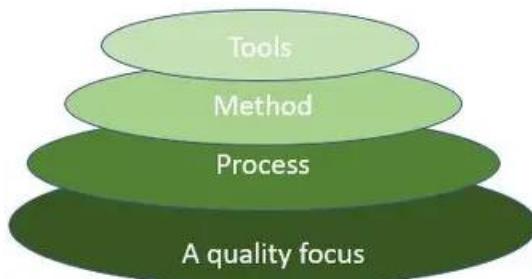
This is the foundation of all software engineering activities.

- It ensures the final product meets customer requirements and performs reliably.
- Quality is maintained through activities like quality assurance (QA), reviews, and testing.
- Without this layer, all other efforts may lead to unreliable or unsatisfactory software.
- Total quality management, six sigma and similar philosophies foster a continuous process improvement culture.
- The bedrock that supports software engineering is a quality focus. 2 Process Layer
- This layer defines the framework for managing and controlling the software development process.
- It provides a structured approach using Software Development Life Cycle (SDLC) models like Waterfall, Agile, Spiral, etc.
- Ensures planning, monitoring, and evaluation of all phases in software development.
- The software process forms the basis for management of software projects.

3 Methods Layer

This layer includes the technical methods used to build software.

- Covers requirement analysis, system design, coding, testing, and maintenance.
- Methods guide how software is developed technically to ensure correctness and efficiency.



4 Tools Layer

This is the topmost layer, consisting of automated tools that support the methods and processes.

- Tools improve productivity, accuracy, and efficiency.
- Examples include IDEs (like VS Code), testing tools (like Selenium), version control systems (like Git).

5. Explain the Core Principles of Software Engineering.

Ans: These principles are fundamental guidelines to help engineers build software that is reliable, efficient, maintainable, and user-focused

- ❖ The Reason It All Exists
 - Always focus on adding value to the customer or end-user.
 - Software is created to solve a problem or meet a specific need.
- ❖ KISS (Keep It Simple, Stupid) .
 - Simple solutions are often better and easier to maintain.
 - Avoid unnecessary features and complexity
- ❖ Maintain the Vision
 - Keep a clear goal throughout development.
 - Ensure every team member understands the core purpose of the product.
- ❖ What You Produce, Others Will Consume
 - Write clear code and documentation that others (like future developers or testers) can understand and use.
- ❖ Be Open to the Future
 - Design software to be scalable and adaptable, so it can evolve over time without major rewrites
- ❖ Plan Ahead for Reuse

- Create modular, well-documented components that can be reused in other projects or systems.
- ❖ Think!
 - Always think before you code. Analyze the problem, assess solutions, and anticipate challenges

6. Explain Software Engineering practices.

Ans: Involves the continuous exchange of accurate and complete information between stakeholders (clients, users, developers, testers).

i) Communication Principle

1. Listen: Try to focus on the speaker's words. Ask for clarification if something is unclear, but avoid constant interruptions.
2. Prepare before you communicate: Spend the time to understand the problem before you meet with others. If necessary, do some research to understand business domain jargon.
3. Someone should facilitate the activity: Every communication meeting should have a leader to keep the conversation moving in a productive direction.
4. Face-to-Face communication is best: It usually works better when some other representation of the relevant information is present. For eg. A participant may create a document that serves as a focus for discussion.
5. Take notes and document decisions: Someone participating in the communication should serve as a "recorder" and write down all important points and decisions.
6. Strive for collaboration: Each small collaboration serves to build trust among team members and creates a common goal for the team.
7. Stay focused; modularize your discussion: The facilitator should keep the conversation modular; leaving one topic only after it has been resolved.
8. If something is unclear, draw a picture: A sketch or drawing can often provide clarity when words fail to do the job.
9. A) Once you agree to something, move on. B) If you can't agree to something, move on C) If a feature or function is unclear and cannot be clarified at the moment, move on.
10. Negotiation is not a contest or a game. It works best when both parties win.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | [v2vedtech.com](https://www.v2vedtech.com)

ii) Planning Principle

1. Understand the scope of the project: It's impossible to use a road map if you don't know where you are going. Scope provides the software team with a destination.
2. Involve stakeholders in the planning activity: Stakeholders define priorities and establish project constraints.
3. Recognize that the planning is iterative: When the project work begins it's likely that few things may change. To accommodate these changes the plan must be adjusted, as a consequence.
4. Estimate based on what you know: The purpose of estimation is to provide an indication of the efforts, cost, and task duration. If the information is vague or unreliable, estimates will be equally unreliable.
5. Consider the risk as you define the plan: The team should define the risks of high impact and high probability. It should also provide a contingency plan.
6. Be realistic: The realistic plan helps in completing the project on time including the inefficiencies and change.
7. Adjust granularity as you define the plan: Granularity refers to the level of details that is introduced as a project plan is developed.
8. Define how you intend to ensure quality: The plan should identify how the software team intends to ensure quality.
9. Describe how you intend to accommodate change: Even the best planning can be obviated by uncontrolled change.
10. Track and monitor the plan frequently and make adjustments if required: Software projects fall behind schedule one day at a time.

iii) Modeling Principle

Analysis Modelling Principles:

1. The information domain of a problem must be represented and understood.
2. The functions that the software performs must be defined.
3. The behavior of the software (as a consequence of external events) must be represented.
4. The models that depict information function and behavior must be partitioned in a manner that uncovers detail in a layered fashion.
5. The analysis task should move from essential information toward implementation detail.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

iv) Construction Principle

Preparation Principles:

1. Understand of the problem you're trying to solve.
2. Understand basic design principles and concepts.
3. Pick a programming language that meets the needs of the software to be built and the environment in which it will operate.
4. Select a programming environment that provides tools that will make your work easier.
5. Create a set of unit tests that will be applied once the component you code is completed.

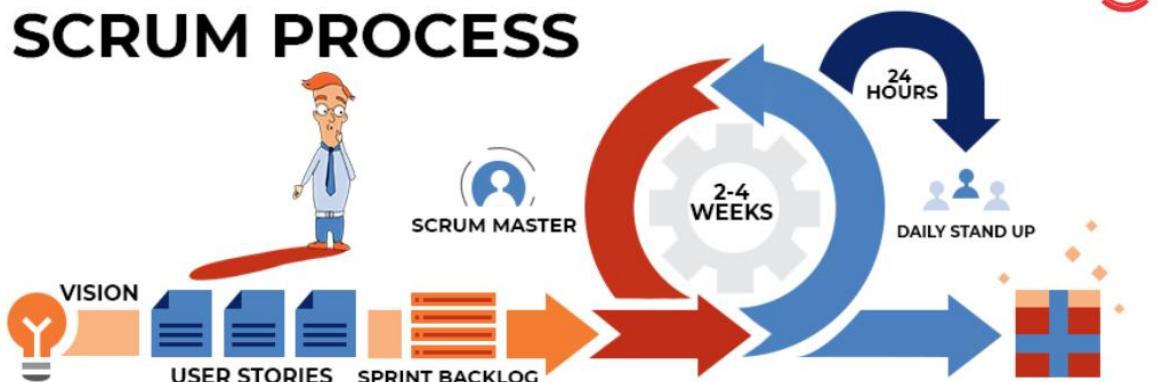
v) Deployment Principle

1. Customer expectations for the software must be managed.
2. A complete delivery package should be assembled and tested.
3. A support regime must be established before the software is delivered.
4. Appropriate instructional materials must be provided to end users.
5. Buggy software should be fixed first, delivered later.

7. Explain Scrum, XP (Extreme Programming), Agile Model, and Spiral Model , DSDM , ASD , AUP

Ans:

i)Scrum Model



Definition: Scrum is an Agile framework that organizes development in short iterations (called sprints), typically lasting 1–4 weeks. The team works collaboratively to complete selected features during each sprint.

Key Roles in Scrum:

- Product Owner: Represents the customer, defines product backlog
- Scrum Master: Facilitates the process, removes blockers
- Development Team: Self-organizing team that delivers the product

Key Elements

- Product Backlog: List of features or requirements
- Sprint Backlog: Tasks to be completed in a sprint
- Sprint Planning: Decide what to deliver in the sprint
- Daily Stand-up (Daily Scrum) : 15-minute daily progress meeting
- Sprint Review & Retrospective: Evaluate work and process after each sprint

Advantages of Scrum:

- Faster and incremental delivery of features.
- Encourages continuous feedback and improvement.
- Quick identification and resolution of problems through daily stand-ups.
- Promotes transparency and accountability within the team.

Disadvantages of Scrum:

- Not suitable for uncooperative teams
- Scope creep can occur if backlog is not managed well
- Requires experienced team for best results

Applications:

- Web and mobile app development
- Product-based companies with regular releases
- Projects requiring frequent updates or customer involvement

ii) XP Model

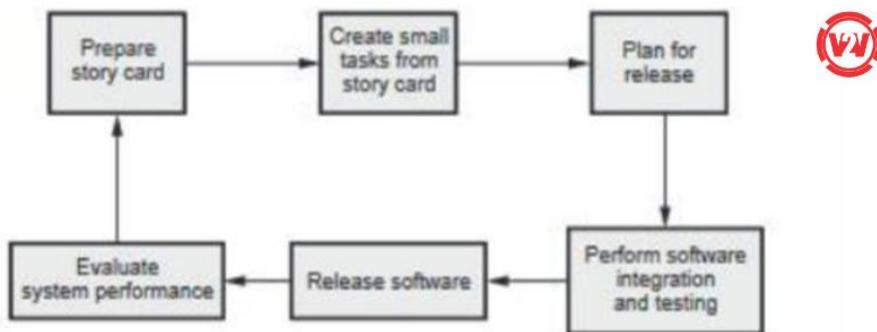


Fig. 1.8.1 Extreme programming release cycle

Definition: XP is an Agile method focused on improving software quality and responsiveness to customer requirements using frequent releases and technical practices. Prioritizes code simplicity, communication, and continuous improvement.

Practices include Test-Driven Development (TDD), Pair Programming, Continuous Integration, and Refactoring.

Core Practices of XP:

- Pair Programming – Two developers code together on the same computer
- Test-Driven Development (TDD) – Write tests before coding
- Continuous Integration – Code is integrated and tested frequently
- Refactoring – Regular code improvement
- Small Releases – Deliver functional software often
- Simple Design – Avoid unnecessary complexity

Advantages of XP:

- High-quality code due to frequent testing and continuous integration.
- Enables fast and flexible responses to changing requirements.
- Promotes strong team communication through pair programming and stand-ups.
- Frequent releases keep customers engaged and informed.

Disadvantages of XP:

- Requires high discipline and skilled developers
- Continuous feedback may be overwhelming
- Not ideal for very large teams or long-term planning

Applications:

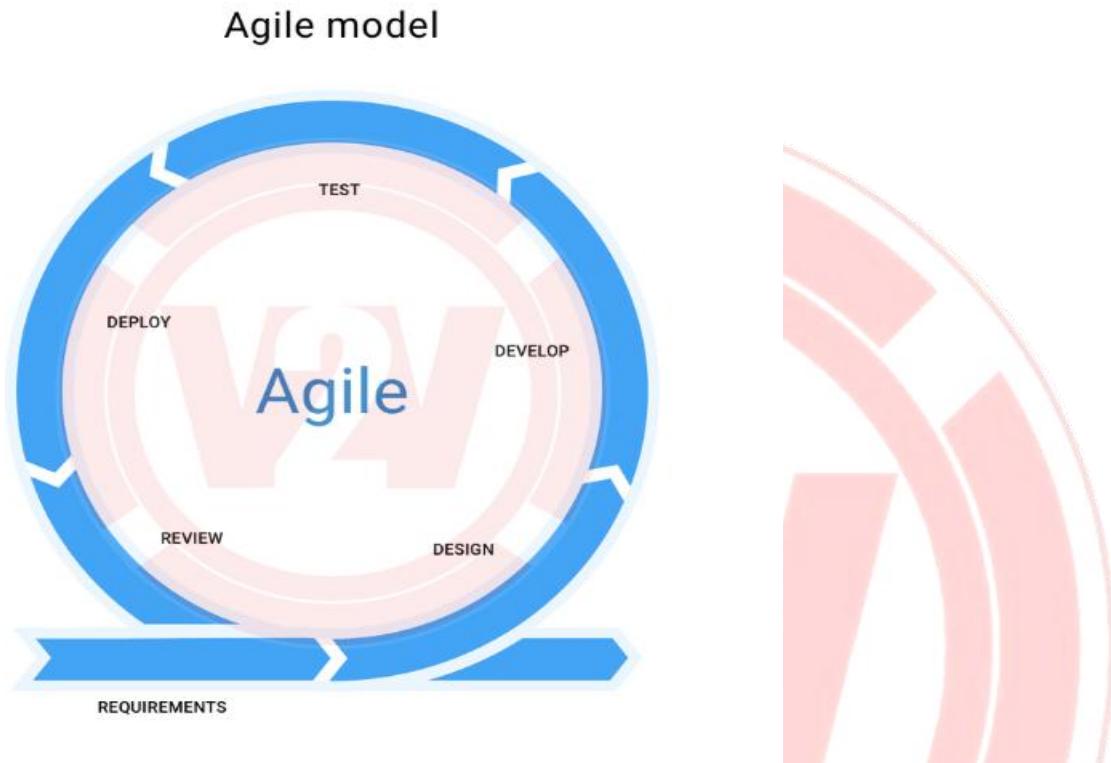
- Small to medium-sized software projects

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Real-time and evolving applications
- Startups needing quick MVP (Minimum Viable Product)

iii) Agile Model



Agile is a flexible, iterative, and collaborative software development methodology. Agile is a lightweight and iterative software development methodology focused on flexibility, customer collaboration, frequent delivery, and responding to change. Instead of a rigid plan, Agile encourages continuous improvement, short development cycles (sprints), and early delivery. Divides work into small units called iterations or sprints (typically 1–4 weeks). Continuous planning, testing, and integration.

2. Core Principles of Agile (Based on Agile Manifesto)

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

3. Importance of Agile

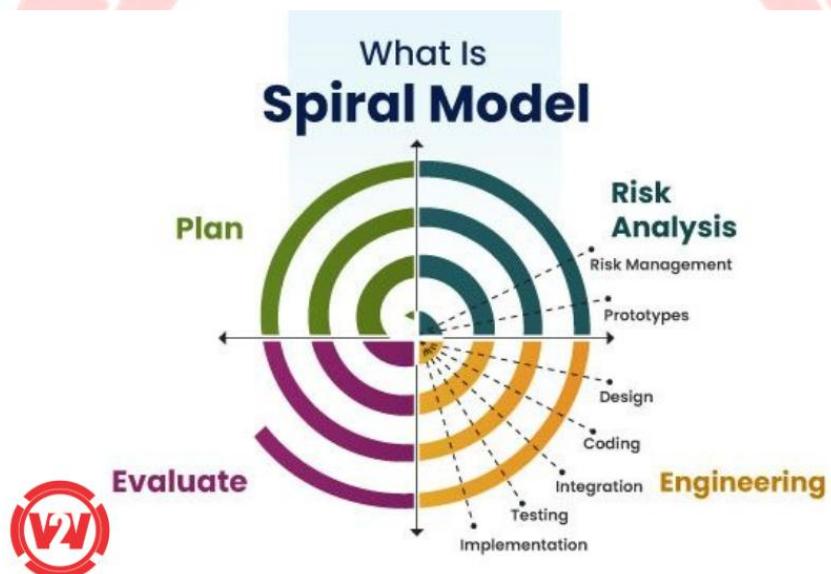
- Fast Delivery: Delivers usable software early and frequently.
- Mob No : 9326050669 / 9372072139 | Youtube : @v2vedtechllp

- Customer Satisfaction: Involves customer feedback at every stage.
- Flexibility: Easily adapts to changing requirements.
- Improved Quality: Regular testing during each iteration.
- Better Team Collaboration: Promotes face-to-face communication and teamwork.
- Risk Reduction: Problems are identified early, reducing the risk of failure.

4. Agile Process – Key Activities

1. Requirement gathering in the form of user stories
2. Planning for short time-boxed iterations (called sprints)
3. Design and Development in cycles (each sprint produces working software)
4. Testing and Review after each sprint
5. Retrospective to reflect and improve in the next iteration

iv) **Spiral Model**



Definition: The Spiral Model combines features of Waterfall and Prototyping, focusing on risk analysis. It involves repeated cycles (spirals), each with planning, risk analysis, development, and evaluation. Excellent for large, complex, high-risk projects. Allows changes at any stage. More costly and complex to manage.

Advantages:

- Focuses on risk analysis and mitigation at every phase.
- Supports flexible and iterative development.
-

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Well-suited for large, high-risk, and complex projects.

Disadvantages:

- Allows progressive refinement of requirements.
- Complex to manage and implement
- Expensive for small projects
- Requires expertise in risk assessment

Applications:

- Aerospace, military, and safety-critical software
- Long-term or high-budget systems
- R&D projects

V)DSDM Model

The DSDM is an agile software development approach that provides a framework for building and maintaining systems, which meet tight time constraints through the use of incremental prototyping in a controlled project environment.

DSDM is based on nine key principles that basically revolve around business needs and values, active user involvement, empowered teams, frequent delivery, integrated testing, and stakeholder collaboration.

The main features of the DSDM method are as follows:

1. User involvement.
2. Iterative and incremental development.
3. Increased delivery frequency.
4. Integrated tests at each phase.
5. The acceptance of delivered products depends directly on fulfilling requirements.

DSDM specifically calls out "fitness for business purpose" as the primary criteria for delivery and acceptance of a system, focusing on the useful 80% of the system that can be deployed in 20% of the time.

DSDM projects are prioritized using MOSCOW Rules which is stated as follows:

M-Must have requirements.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

S-Should have if at all possible.

C-Could have but not critical.

W-Would not have this time, but potentially later.

All critical work must be completed in a DSDM project.

Within each time-box, less critical items are included so that if necessary, they can be removed to keep from impacting higher priority requirements on the schedule.

The DSDM project framework is independent of, and can be implemented in conjunction with, other iterative methodologies such as Programming (XP) and the Rational Unified Process (RUP).

Advantages of the DSDM:

1. DSDM provides a technique-independent process and is flexible in terms of requirement evolution.
2. DSDM is designed from the ground up by business people, so business value is identified and expected to be the highest priority deliverable.
3. DSDM incorporates stakeholders into the development process.
4. An emphasis as DSDM on testing is so strong that at least one tester is expected to be on each project team. Which improves software quality and performance.

Disadvantages of the DSDM

1. DSDM focuses on RAD, which can lead to decrease in code robustness.
2. It requires significant user involvement.
3. DSDM requires a skilled development team in both the business and technical areas.

vi) ASD Model

- Adaptive Software Development (ASD) is a technique for building complex software and systems. ASD is a design principle for the creation of software systems. The principle focuses on the rapid creation and evolution of software systems.
- The goal of the ASD model is to consider midstream changes as the natural flow of projects and to incorporate them into the development model. Adaptive software development is made of three steps, each revolving around the coding of a program.
- The first step in ASD is speculation. During this phase, coders attempt to understand the exact nature of the software and the requirements of the users. This phase relies on bug and user reports to guide the project.

- The second step is collaboration. Collaboration in ASD is a balance between managing the doing and creating and maintaining the collaborative environment needed for output.
- - First, he or she has to decide which parts of the project are predictable.
 - For parts that are unpredictable, a manager has to establish an environment in which a team basically opens, collaborates..

Learning is the last step in adaptive software development. Learning activities expose those products to a variety of stakeholders to ascertain value.

Advantages of ASD:

1. The ASD model works very well for smaller teams (four to eight people) when the requirements technologies are uncertain or domain knowledge is missing.
2. The ASD allows the team to speculate about the final product without providing a detailed plan up front. In this way, the team can continuously adjust the final product over the course of the project.
3. The ASD includes processes for tracking progress and identifying problem areas.

Disadvantages of ASD:

1. In areas of predictable requirements and technologies, the traditional methods will probably work more efficiently than the Adaptive model.
2. The project is very different from managing projects developed with the traditional models because the Adaptive approach is result oriented rather than flow oriented.

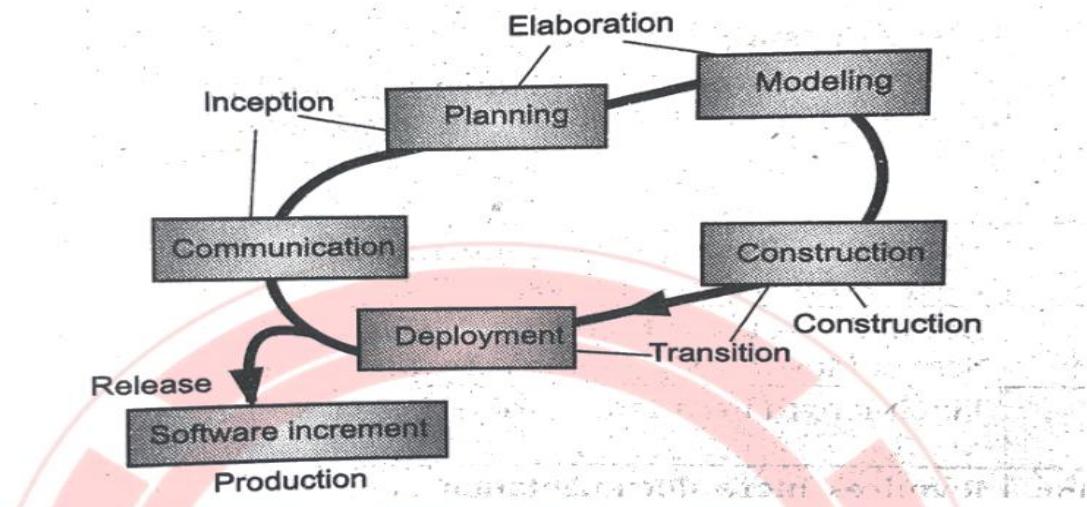
vii)AUP Model

Unified Process (UP) is a software development process framework that provides guidelines to design and build high-quality softwares. Agile Unified Process (AUP) is a simplified version of the Rational Unified Process (RUP). It describes a simple, easy to understand approach to developing business application software using agile techniques.

The AUP is an iterative, and adaptable software development methodology that combines the best practices of agile development with the disciplined approach of the Unified Process (UP). AUP is designed to deliver high-quality software that meets the changing needs of the stakeholders in an efficient and effective manner. RUP is an iterative and incremental approach to software development. The unified process is a "use-case driven, architecture-centric, iterative and incremental" software process designed as a framework for UML methods and tools.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com



- Fig. 1.23 shows the "phases" of the Unified Process (UP). The Unified Process is an incremental model 1 .

1. An inception phase that encompasses both customer communication and planning activities and emphasizes the development and refinement of use-cases as a primary model.
2. An elaboration phase that encompasses the customer communication and modeling activities focusing on the creation of analysis and design models with an emphasis on class definitions and architectural representations.
3. A construction phase that refines and then translates the design model into implemented software components.
4. A transition phase that transfers the software from the developer to the end-user for beta testing and acceptance.
5. A production phase in which on-going monitoring and support are conducted

Advantages of AUP:

1. Adaptability
2. Flexibility
3. High-Quality Software
4. Faster Time-to-Market
5. Risk Management

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

Disadvantages of AUP:

1. Lack of Formal Process: AUP may not provide the level of structure and formal process that some organizations require.
2. Lack of Documentation: AUP is a lightweight methodology that emphasizes working software over comprehensive documentation.
3. Limited Scalability: AUP may not be suitable for large, complex projects that require a high degree of coordination and management.
4. Dependency on Team Experience: AUP relies on the experience and expertise of the development team to deliver high-quality software.

- Ideal for small or well-defined projects
- No feedback until final stage
- Difficult to handle changes once a phase is complete

Disadvantages:

- No flexibility once the project is in the testing phase
- No overlapping of stages; each must be completed before the next begins.
- Poor model for long or complex projects
- Difficult to handle changes in requirements

Applications:

- Government and defense projects
- Projects with clear, fixed requirements

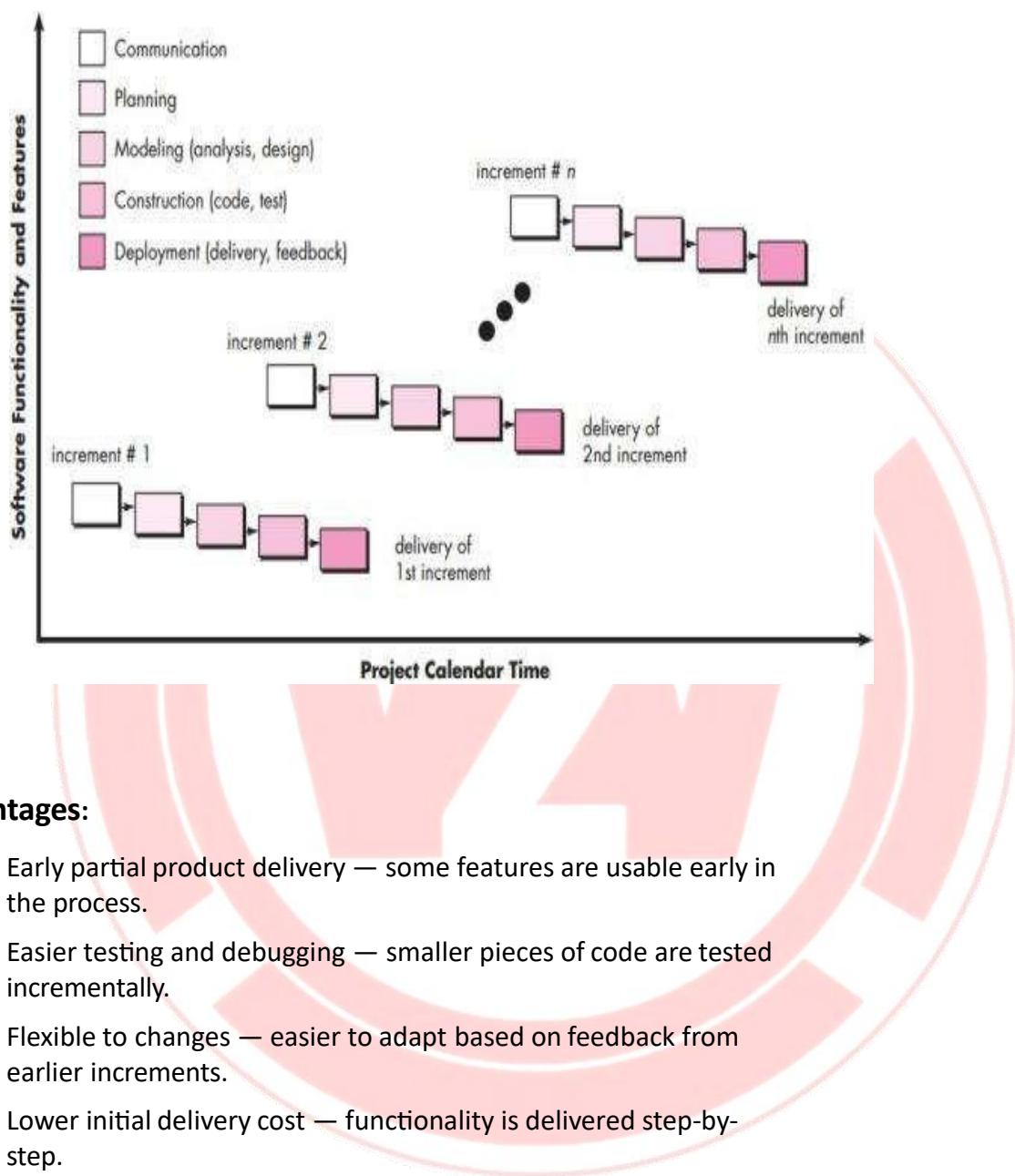
(viii). Incremental Model

Definition: The Incremental Model develops software in **small, manageable parts (increments)**. Each increment adds functionality until the final product is complete. Easier to test and debug. Early delivery of partial functionality. Each increment includes design, coding, and testing. Useful when requirements are well understood, but may evolve.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

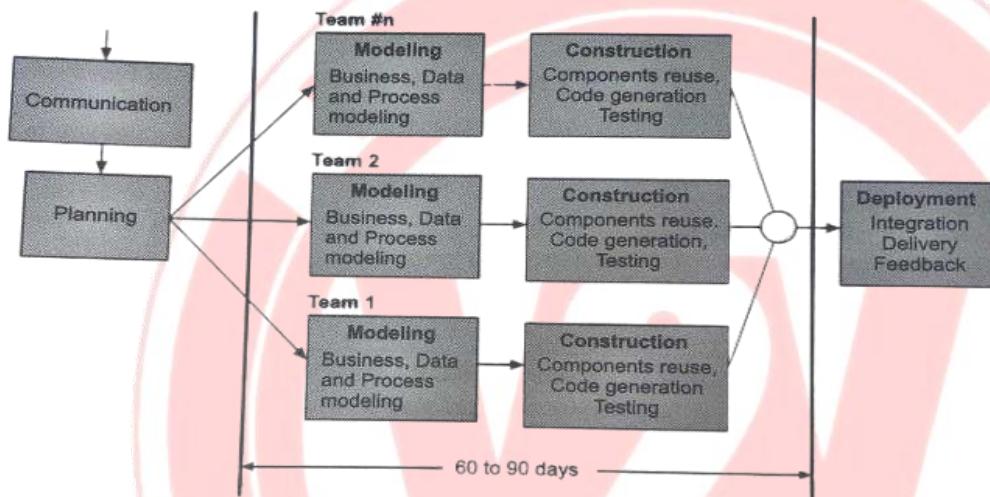
Helps reduce risk by identifying issues early.



Applications:

- Web applications
- E-learning platforms
- Projects with evolving requirements

ix. RAD Model (Rapid Application Development)



Definition: RAD focuses on **quick development and delivery** by using reusable components, prototyping, and minimal planning. It emphasizes **rapid iterations with user feedback**. Very fast development using reusable components. Requires active user involvement. Best for small to medium-sized projects

Advantages:

- Faster development due to component reuse and parallel tasks.
- Customer involvement is high with constant feedback and review.
- Prototyping helps refine requirements early in the project.
- Highly flexible to changing requirements.

Disadvantages:

- Not suitable for large-scale or complex systems

- Requires skilled developers and designers
- Poor documentation due to rapid pace

Applications:

- Short-term projects
- Mobile apps, internal tools
- UI/UX-heavy applications needing quick changes

10. Define Requirement Engineering with its task

Ans: Requirement Engineering is the process of gathering, analyzing, documenting, and validating the needs and constraints of the stakeholders for a software system.

Its goal is to ensure that the software meets user expectations and business needs.

From a software process perspective, requirements engineering is a major software engineering action that begins during the communication activity and continues into the modeling activity. It must be adapted to the needs of the process, the project, the product, and the people doing the work.

Main Tasks of Requirement Engineering:

- 1. Requirement Elicitation**
 - Involves gathering requirements from stakeholders, users, customers, and other sources.
 - Techniques: Interviews, questionnaires, brainstorming, observation, document analysis.
- 2. Requirement Analysis**
 - Examines and refines the gathered requirements to remove conflicts, inconsistencies, and ambiguities.
 - Ensures that requirements are clear, complete, and feasible.
- 3. Requirement Specification**
 - Documents the analyzed requirements in a clear and structured format (usually in a **Software Requirement Specification - SRS** document).
 - This becomes the official agreement between developers and clients.
- 4. Requirement Validation**

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#)

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)

- Ensures that the documented requirements accurately reflect stakeholder needs.
- Techniques: Reviews, inspections, prototypes, and test cases.

5. Requirement Management

- Handles changes in requirements throughout the software development lifecycle.
- Involves tracking, version control, and impact analysis of changing requirements.

11. Describe the different types of requirements

Ans: Types of Requirements Requirements are generally classified into three main types:

A. Functional Requirements

- Define what the system should do.
- Describe system behavior under specific conditions.
- Functional requirements describe the interaction of software with its environment and specify the inputs, outputs, external interfaces and the functions that should be included in the software.
- These requirements should be complete and consistent.
- Completeness implies that all the user requirements are defined.
- Consistency implies that all requirements are specified clearly without any contradictory definition.
- Example: "The system shall allow users to log in using email and password."

B. Non-Functional Requirements

- Define how the system performs tasks (quality attributes).
- Includes performance, usability, reliability, and security.
- These requirements arise due to user requirements, budget constraints, organizational policies, and so on. These requirements are not related directly to any particular function provided by the system.

Different type of Non Functional Requirement

1. Product Requirements:
 - Reliability Requirements describe the acceptable failure rate of the software.
 - Usability Requirements describe the ease with which users are able to operate.
 - Efficiency Requirements describe the extent to which the software makes resources, the speed with which the system executes and the memory it consumes for its operation.
 - Portability Requirements describe the ease with which the software can be transferred from one platform to another

2. Organizational Requirements:

- (a) Implementation Requirements describe requirements such as programming language and design method.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#)

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)

(b) Standards Requirements describe the process standards to be used during software development. For example, ISO and IEEE standards.

(c) Delivery Requirements specify when the software and its documentation are to be delivered to the user.

3. External Requirements:

(a) Interoperability Requirements defines the way in which different computer-based systems interact with each other in one or more organizations.

(b) Legislative Requirements ensures that the software operates within the legal jurisdiction. For example, pirated software should not be sold.

(c) Ethical Requirements specifies the rules and regulations of the software so that they are acceptable to users.

Example: "The system must handle 1000 transactions per second.

4. Domain Requirements

- Specific to the domain or industry of the software.
- Often driven by regulations, standards, or specific rules.
- Example: "A banking system must comply with RBI data security policies."

12. Explain Software Requirement Specification (SRS) with its needs and importance.

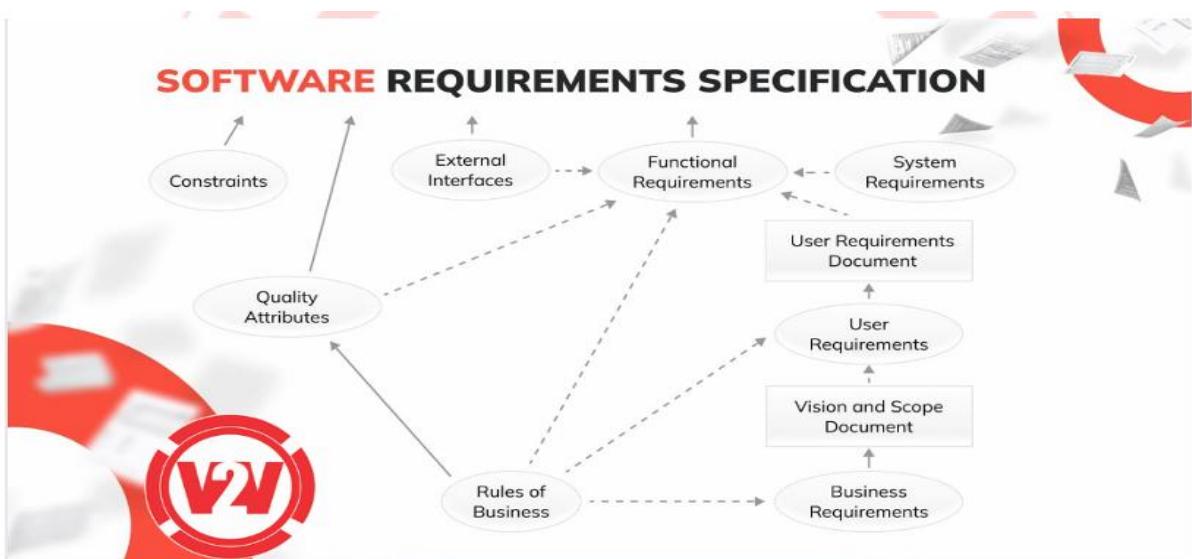
Ans: - SRS is a formal document that describes the functional and non-functional requirements of the software system to be developed. It acts as an agreement between the customer and the development team. The document completely describes what the proposed software should do without describing how the software will do it.

Need for SRS (Why SRS is Important)

- Clear Communication: It serves as a contract between stakeholders and developers.
- Requirement Clarity: It helps avoid misunderstandings and assumptions.
- Project Planning: It provides the foundation for cost, effort, and time estimation.
- Design & Development Base: It is used by designers and developers to build the system.
- Testing & Validation: Testers use it to create test cases and validate functionality.
- Maintenance Reference: It helps in future enhancements and debugging.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com



13. Explain format & characteristics of a good SRS.

Ans: An SRS (Software Requirement Specification) is a formal document that clearly describes what the software system should do. It defines the functional and non-functional requirements of the system and acts as a bridge between the client and the development team.

The characteristics of a good SRS are:

- Correctness: It should accurately describe the actual needs of the stakeholders.
- Completeness: It should include all significant requirements.
- Unambiguity: Each requirement should have only one interpretation.
- Consistency: It should contain no conflicting requirements or definitions.
- Verifiability: Each requirement must be testable.
- Modifiability: The document should be easy to update and modify.
- Traceability: Each requirement should be traceable to its source.
- Rank for Importance: Requirements should be prioritized.

Format / Structure of SRS

A standard SRS follows the IEEE 830:1998 format (commonly used in software engineering).

1. Introduction

Mob No : [9326050669](tel:9326050669) / [9372072139](tel:9372072139) | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech_](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

Describes the basic idea and purpose of the software.

Includes:

- Purpose of the document
- Scope of the system
- Definitions, acronyms, and abbreviations
- References
- Overview of the document

2. Overall Description

Gives an overview of the system, its functions, and constraints.

Includes:

- Product perspective (how it fits in environment)
- Product functions (summary of major features)
- User characteristics (who will use it)
- Constraints (hardware, software, legal, etc.)
- Assumptions and dependencies

3. Specific Requirements

Contains detailed functional and non-functional requirements.

Includes:

- Functional Requirements: Describe specific behaviors or functions.
Example: "System shall allow users to log in using email and password."
- Non-functional Requirements: Describe quality attributes.
Example: "System should respond within 2 seconds."
- Interface Requirements: Define user, hardware, and software interfaces.
- Performance Requirements: Specify speed, accuracy, capacity, etc.
- Security Requirements: Define access control, data protection, etc.

4. Appendices

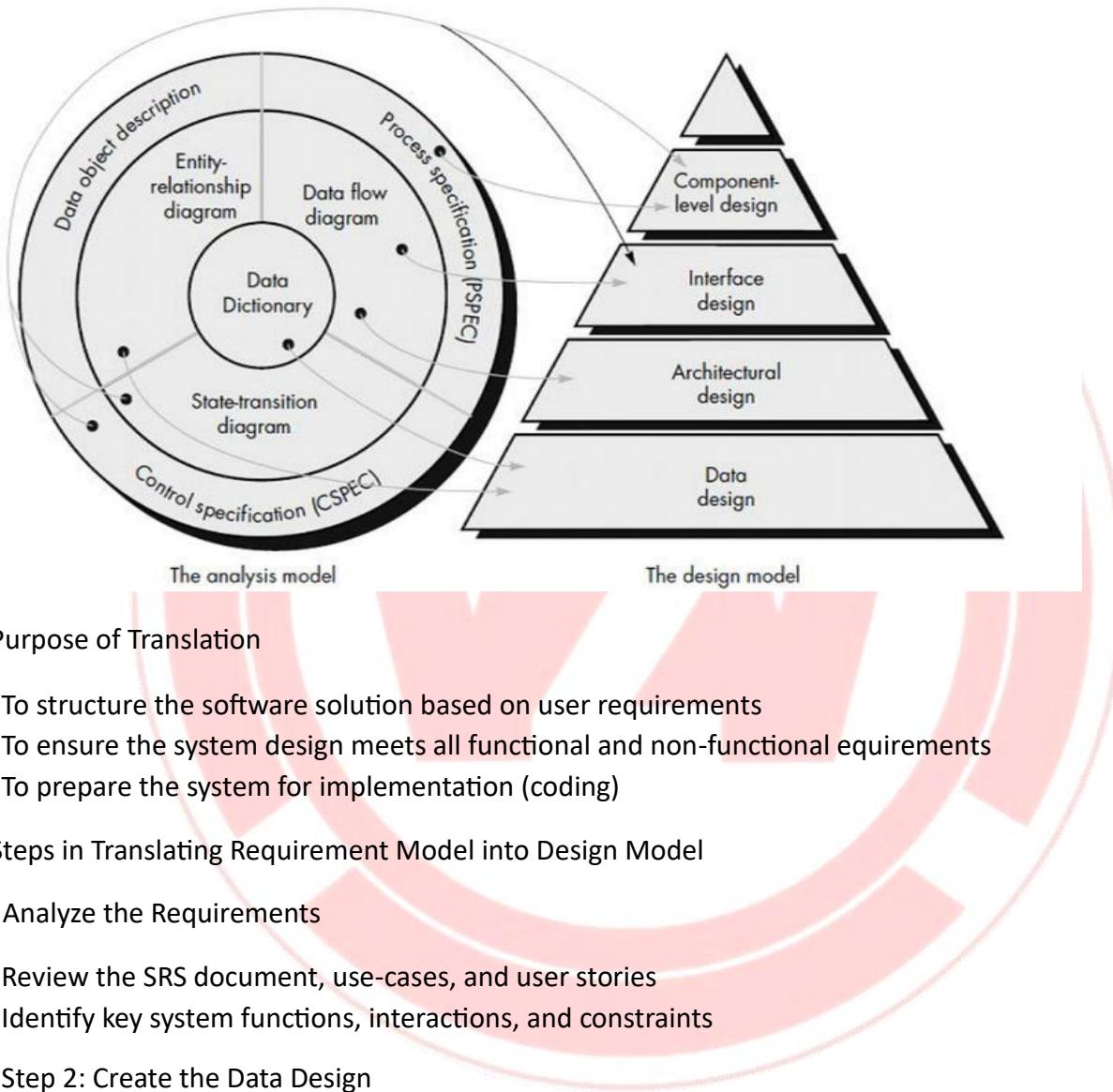
- Additional information such as diagrams, tables, data dictionaries, and references.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#)

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)

14. Explain the process of translating a Requirement Model into a Design model

Ans: - Translating the requirement model into the design model means converting the "what" (requirements) into the "how" (design). It bridges the gap between user needs (captured in the SRS or use-case model) and the technical solution (architecture and components of the system).



2. Purpose of Translation

- To structure the software solution based on user requirements
- To ensure the system design meets all functional and non-functional requirements
- To prepare the system for implementation (coding)

3. Steps in Translating Requirement Model into Design Model

Step 1: Analyze the Requirements

- Review the SRS document, use-cases, and user stories
- Identify key system functions, interactions, and constraints

Step 2: Create the Data Design

- Define data structures and data storage needs
- Use tools like ER Diagrams, Class Diagrams, or Data Dictionaries

Step 3: Develop the Architectural Design

- Decide how the system will be structured
- Identify main components, their responsibilities, and communication

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Output: Architecture Diagram or Component Diagram

Step 4: Design Interfaces

- Design user interfaces (UI) and system interfaces
- Consider input forms, navigation, and user interaction

Step 5: Create Procedural/Component Design

- Define modules, functions, and logic flow
- Describe control flow using Activity Diagrams, Sequence Diagrams, or Flowcharts

Step 6: Verify and Validate the Design

- Ensure the design satisfies all the requirements
- Conduct reviews, walkthroughs, or prototyping

15. What is a Data Flow Diagram (DFD)? Explain DFD Level 0 and DFD Level 1 with their symbols

Ans: A Data Flow Diagram is a graphical representation of the flow of data through a system, showing processes, data stores, external entities, and data flows.

Basic Components of DFD:

Symbol	Element	Purpose
■ Rectangle	External Entity	Source or destination of data (e.g., user)

5

Symbol	Element	Purpose
● Circle/Oval	Process	A function that transforms data
↗ Open rectangle	Data Store	Storage of data (e.g., database, file)
→ Arrow	Data Flow	Direction of data movement

 **Levels of DFD:**

Level	Description
Level 0	Context Diagram – Shows the system as a single process and all external entities
Level 1	Shows main sub-processes and data flow between them
Level 2+	Further breaks down Level 1 processes into detailed sub-processes

1. DFD Level 0 (Context Diagram)

- Also called the **Context Level DFD**.
- It is the **highest level** in a DFD (topmost view).
- Shows the **entire system as a single process** and its **interaction with external entities**.
- Gives an **overall picture** of the system's boundary and data flow.

Features:

- Only **one process** (representing the whole system).
- External entities** connected through data flows.
- No data stores** shown at this level.

2. DFD Level 1

- Expands** the single process of Level 0 into **multiple subprocesses**.
- Shows **major functions or modules** within the system.
- Includes **data stores**, which represent where information is held.
- Provides **more detail** about how the system operates internally.

16. Draw DFD diagrams for the following systems:

- Banking System
- Hospital Management System
- College Management System
- Railway Management
- Library

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

Ans:

i) Banking System .

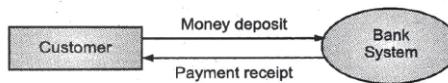


Fig. 2.16: Context Diagram of Bank System

Level 0 DFD

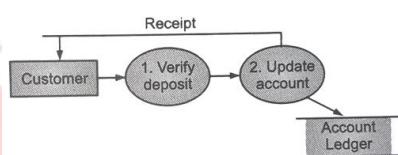


Fig. 2.17: Level 1 DFD of Bank System

Level 2 DFD

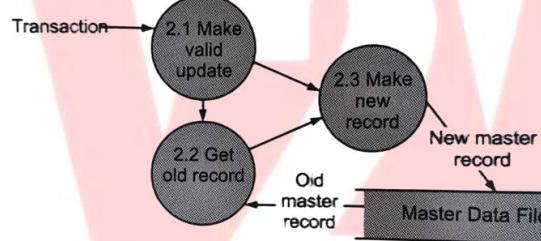


Fig. 2.18: Level 2 DFD of (Update account) of Bank System

ii) Hospital Management System

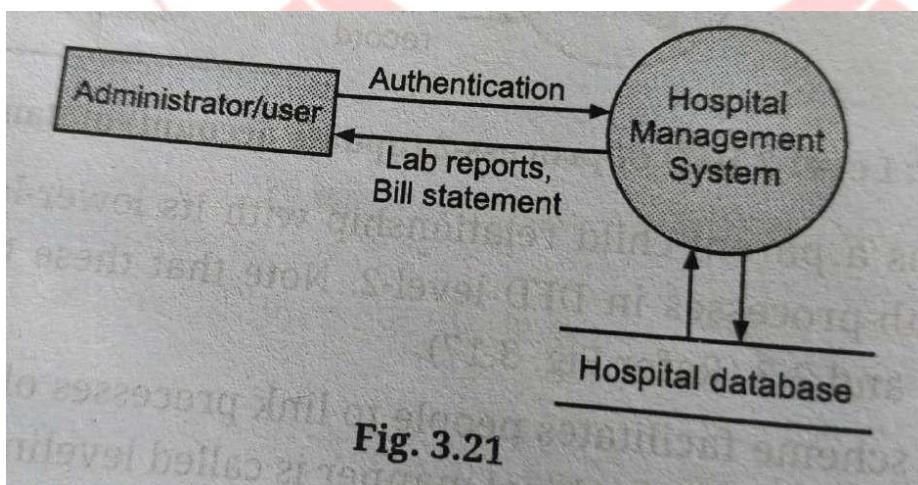
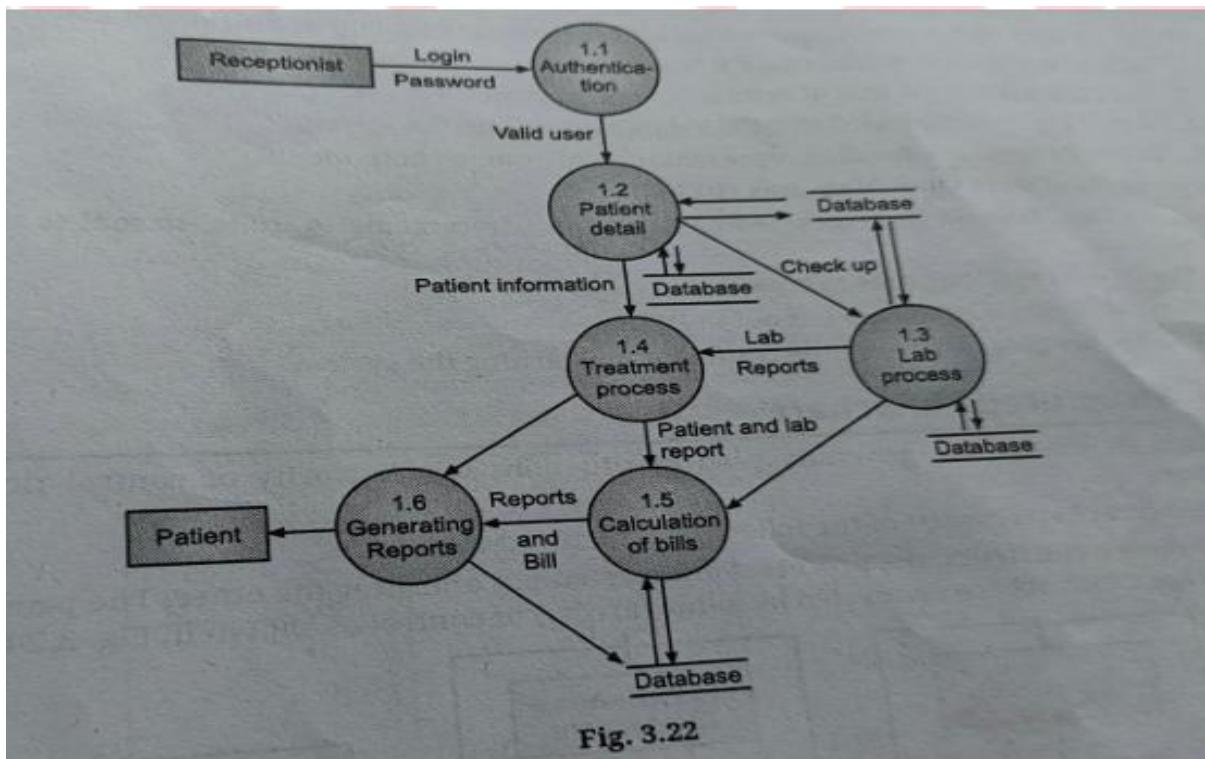


Fig. 3.21



Level 1 DFD

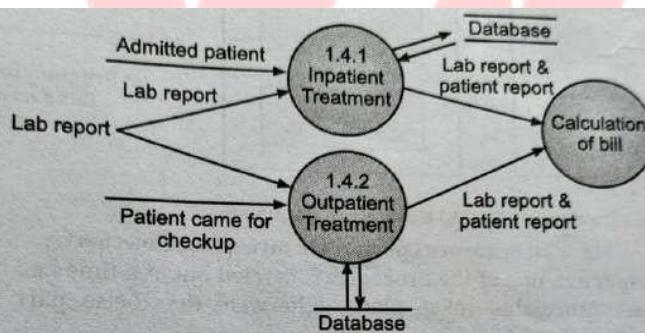


Fig. 3.23 (a): Level 2 DFD

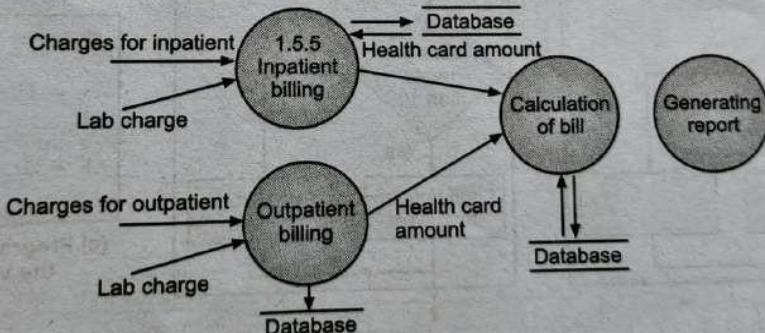
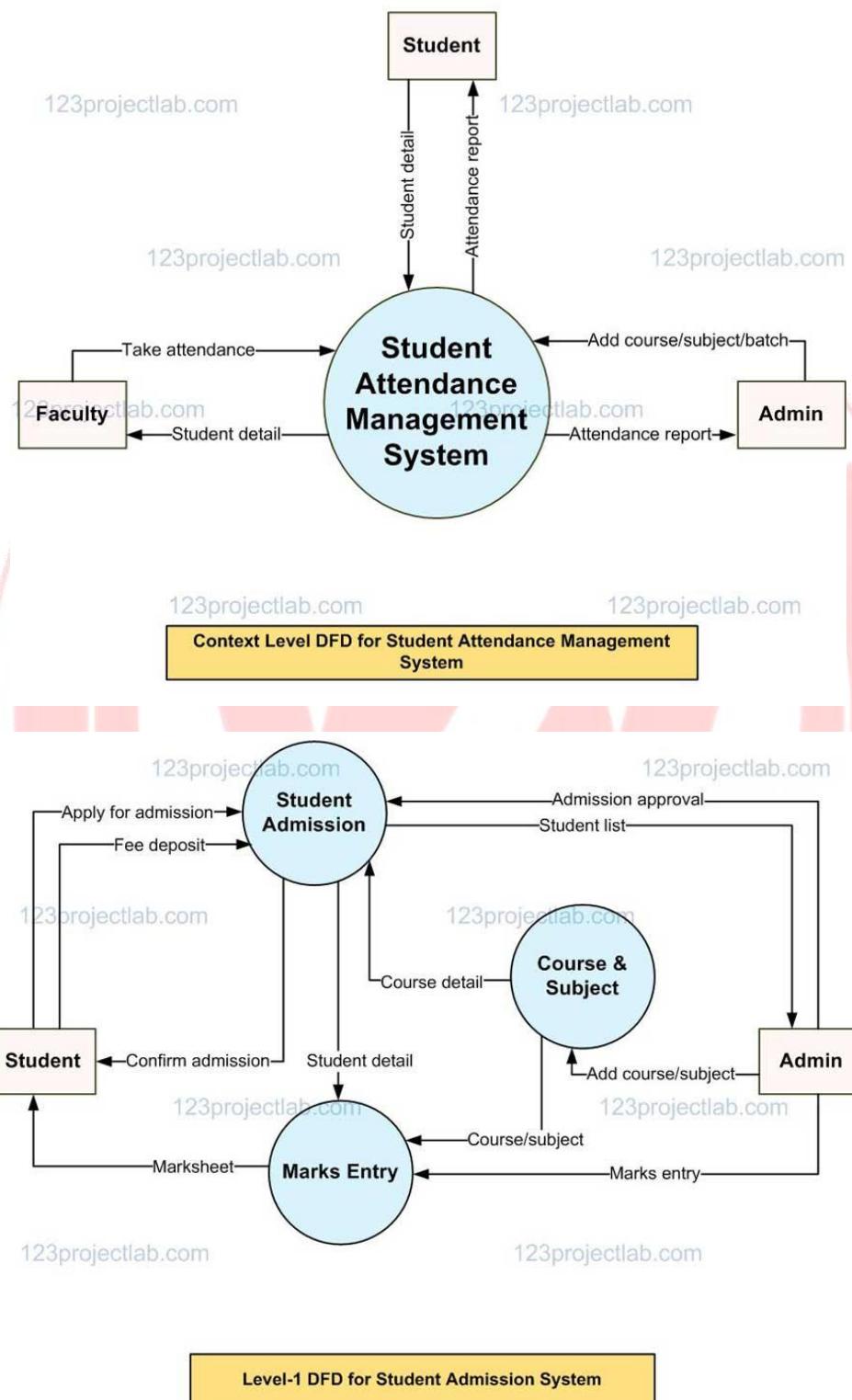


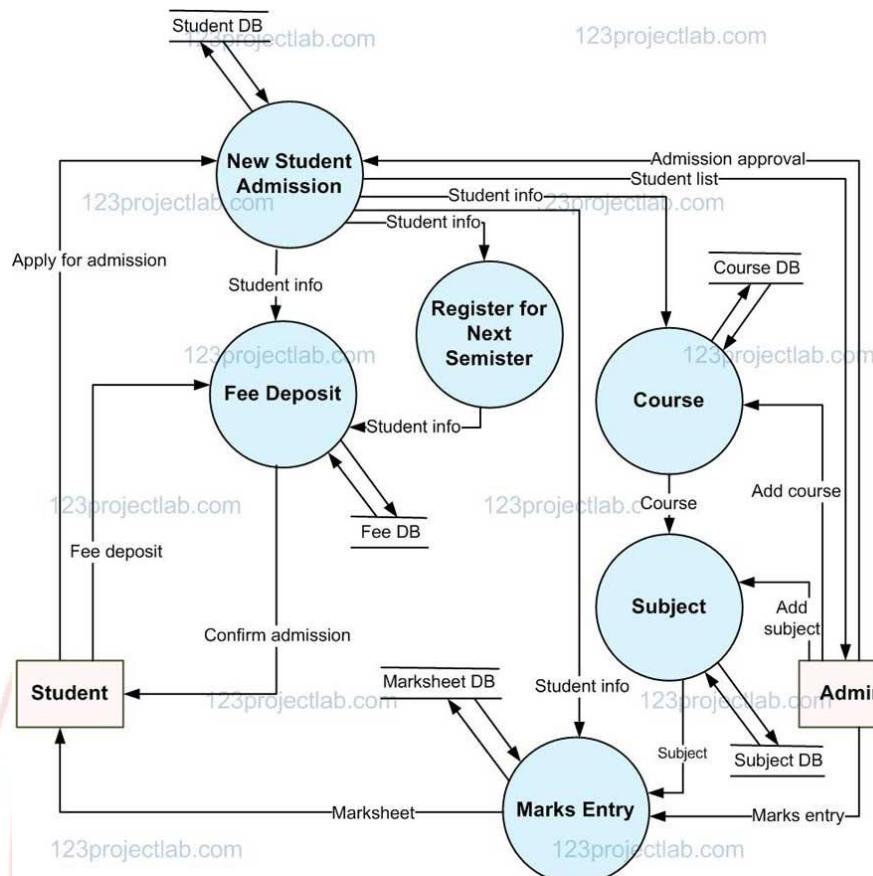
Fig. 3.23 (b): Level 3 DFD

iii) College Management System



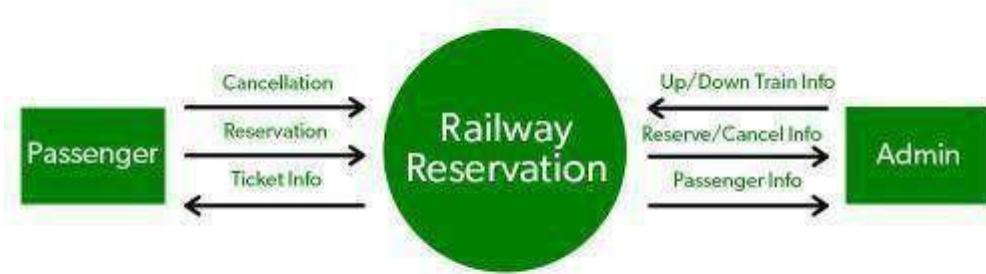
Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#)

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)



Level-2 DFD for Student Admission System

iv) Railway Management System

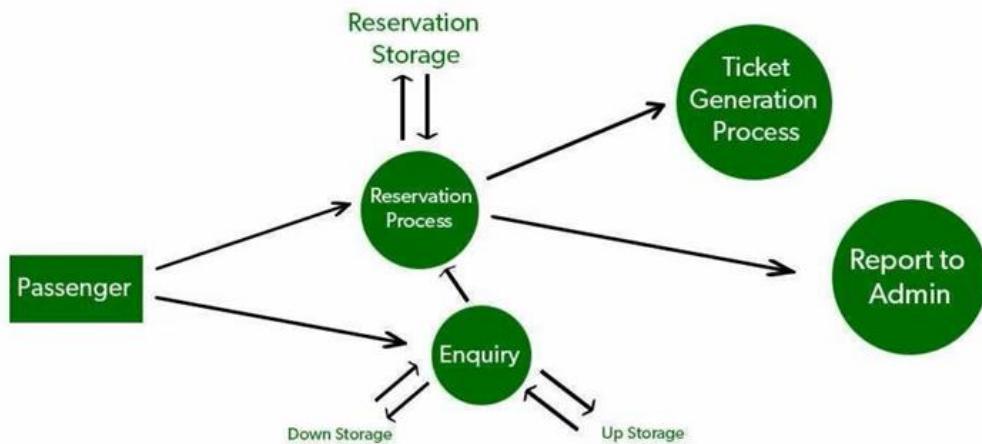


O-LEVEL DFD

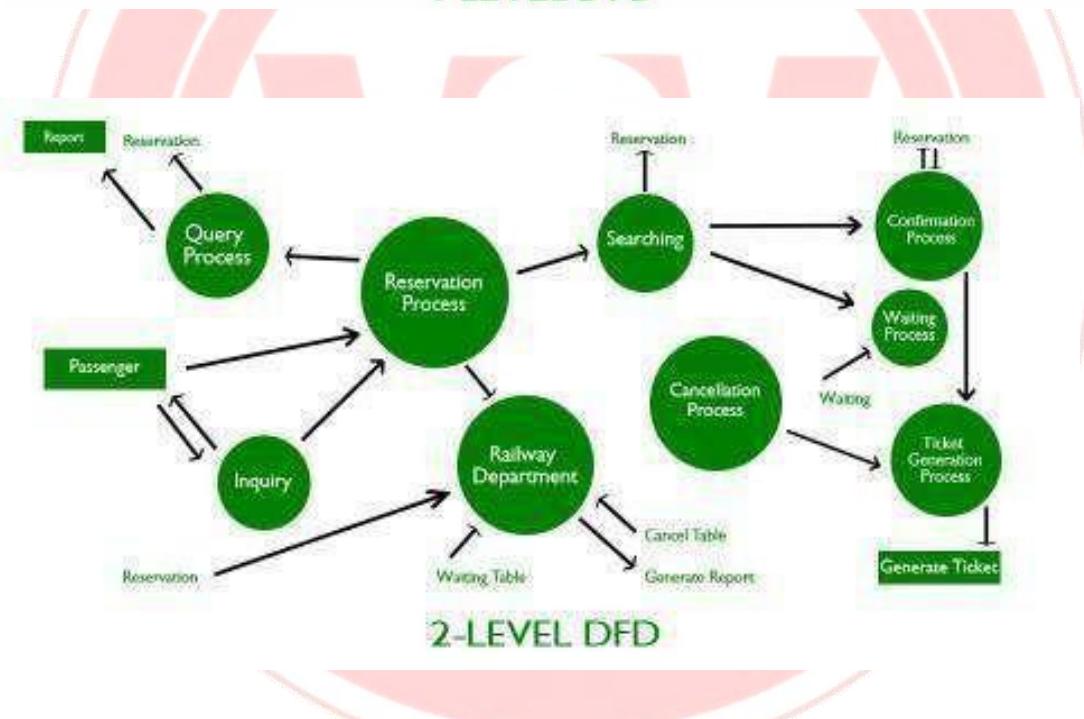
Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech_](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

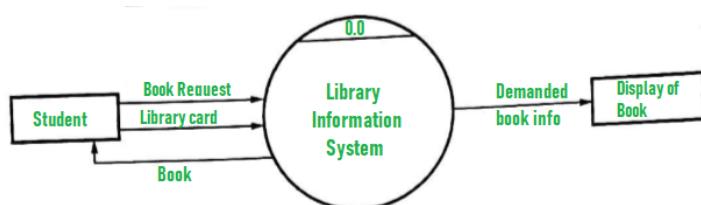
● Level 1 DFD

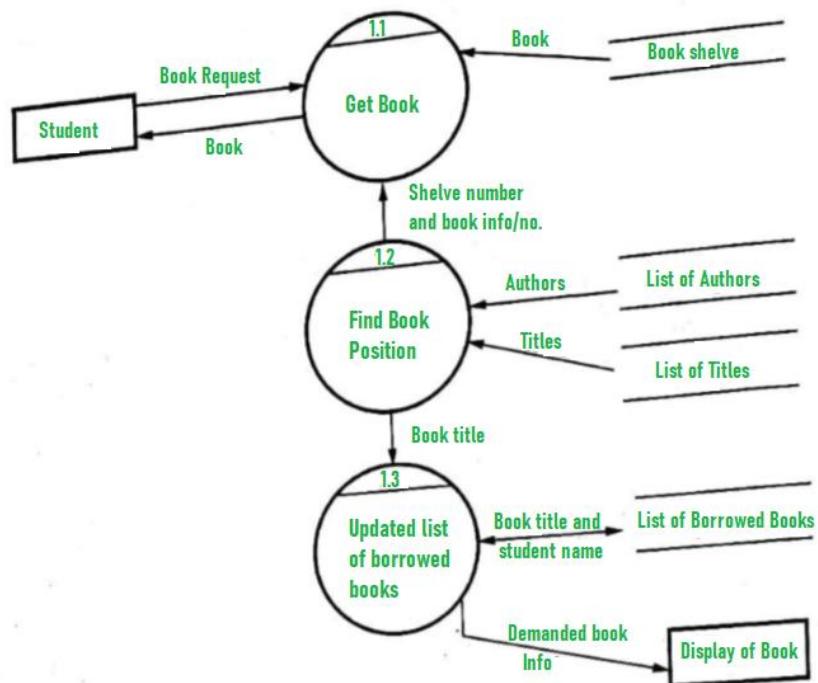
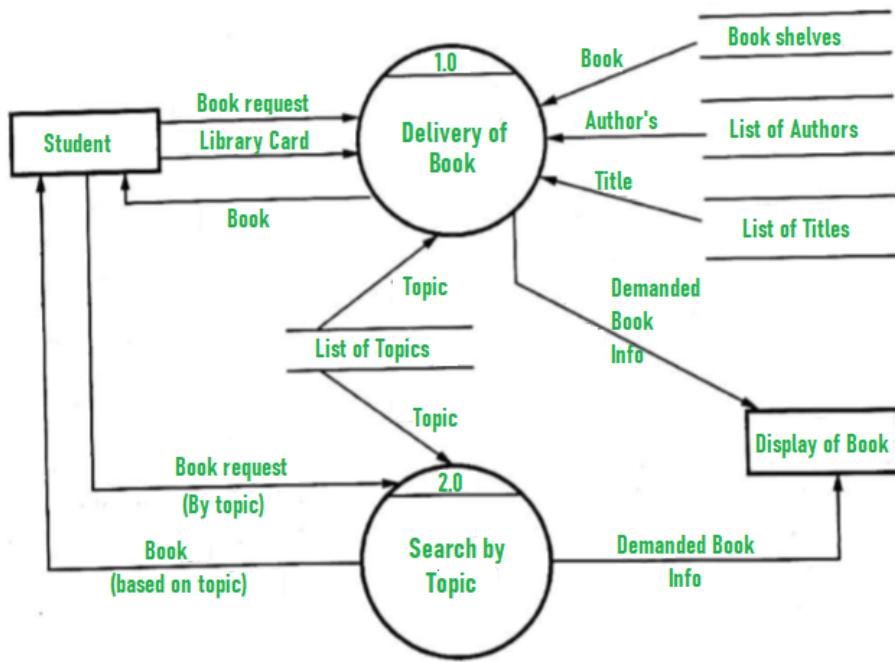


1-LEVEL DFD



2-LEVEL DFD





17. Explain Structured flowchart and its elements.

Ans: A structured flowchart is a diagram that shows the step-by-step logic or flow of a program or process using standard symbols.

Basic Symbols Used:

Symbol	Meaning	Purpose
Oval	Terminator	Start/End of the process
█ Rectangle	Process / Task	A specific operation or instruction
△ Diamond	Decision / Condition	Yes/No or true/false decisions
→ Arrow	Flowline	Shows the direction of control or process flow
□ Parallelogram	Input/Output	Data input or output operation

A flowchart is a graphical representation of an algorithm or process that shows the sequence of steps needed to solve a problem or perform a task.

It uses symbols and arrows to represent different operations and the flow of control in a program or system.

Purpose of a Flowchart

- To illustrate logic clearly and simply.
- To analyze, design, and document a process or algorithm.
- To detect errors or inefficiencies in logic before coding.
- To communicate processes easily among team members.

Advantages of Flowcharts:

- Clearly shows the logic of a process
- Helps in debugging and testing
- Improves communication between developers and clients
- Visual Clarity
- Effective Problem-Solving: By visualizing the entire process, flowcharts help identify bottlenecks, inefficiencies, and unnecessary steps
- Efficiency in Analysis: They enable teams to analyze workflows systematically, identify flaws logic or execution, and optimize the process

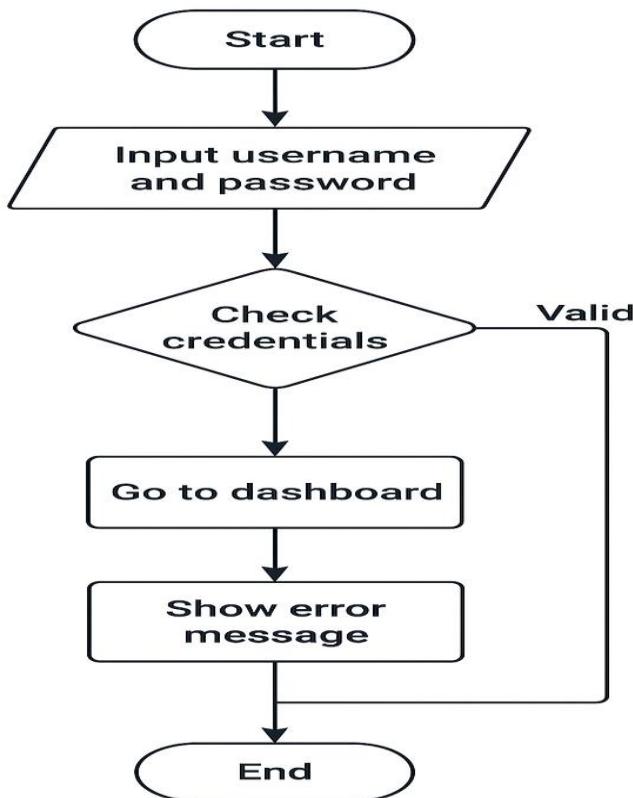
Example: Login System Flowchart

1. Start
2. Input username and password
3. Check credentials
4. If valid → Go to dashboard

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

5. If invalid → Show error message
6. End



18. Explain decision table and its fundamental design concepts (VIMP).

Ans: A Decision Table is a tabular representation of conditions and their corresponding actions. It is used to represent complex business logic or decision-making situations in a clear and structured way.

In simple terms:

A Decision Table helps determine which actions to perform based on different combinations of input conditions.

Purpose of a Decision Table

- To analyze and document complex decision logic.
- To eliminate ambiguity and ensure all possible conditions are considered.
- To serve as a reference for designing or testing software logic.

Structure of a Decision Table

A decision table typically has four parts:

Part	Description
Condition Stub	Lists all the conditions or inputs that affect the decision.
Condition Entries	Specifies the possible values (e.g., Yes/No, True/False) for each condition.
Action Stub	Lists all the possible actions or outcomes that can occur.
Action Entries	Defines which actions should be taken for each combination of conditions.

Example: Login System Decision Table

Conditions	Rule 1	Rule 2	Rule 3
Username correct?	Y	Y	N
Password correct?	Y	N	-
Actions			
Grant access	X		
Show "Invalid password" error		X	
Show "Invalid username" error			X

Explanation:

- Rule 1: Both username and password are correct → Grant access.
- Rule 2: Username correct, password wrong → Show “Invalid password.”
- Rule 3: Username wrong → Show “Invalid username.”

Advantages of Decision Tables

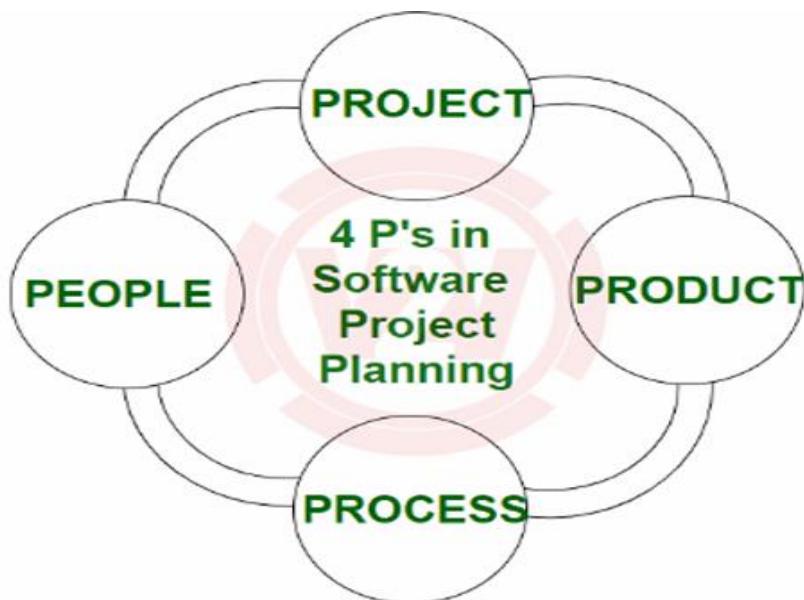
- Clarifies complex decision logic.
- Helps in test case design and error detection.
- Provides a clear, systematic approach to handling multiple conditions.
- Easy to convert into code or flowcharts.

19. Explain Management Spectrum or 4P's

Ans: In software engineering, the Management Spectrum refers to the four essential elements that influence the success or failure of a software project. It focuses on the 4P's. People, Product, Process and Project. These are known as the 4 P's

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com



1. People

People are the most important resource in any software project. This includes project managers, developers, testers, customers, users, and other stakeholders. Effective communication, collaboration, motivation, and leadership play a vital role in managing people.

Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

People connected directly or indirectly are the dynamic and live factor in project management. Project management has to deal and interact with different levels of people within and outside the organization.

People from within the organization are developers, analysts, designers, technology specialists, HR, managers and senior management.

2. Product

The product is the software system to be developed. It includes all functional and non-functional requirements specified by the customer. The product in the context of software is the scope of the software that is proposed to solve the requirement of the user.

To develop successfully, product objectives and scope should be established, alternative solutions should be considered, and technical and management constraints should be identified.

Without this information, it is impossible to define reasonable (and accurate) estimates of the cost.

Understanding the product clearly is essential for correct estimation of resources, time, and cost.

The software and its scope are a product that is defined and described by following four factors:

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

1. Objective : Objective is beneficial to users to solve business problems.
2. Performance: This stipulates non-functional requirements of speed, quality and features such as ease of use, adaptability, interoperability.
3. Context: The scenario or situation in business which has problems and which need solutions within the given domain.
4. Functions: Software system processes and functions it provides.

3. Process

A process is the set of activities and tasks used to develop the software. Process refers to methodologies to be followed for developing the software. It is the framework for establishment of a comprehensive plan and strategies for software development. Choosing an appropriate process model (e.g., Waterfall, Agile, Spiral) ensures that the project is well-managed, efficient, and structured.

The process specifies the policies, procedures, tools and techniques to be used for software development.

4. Project

The project includes the overall management and execution of work. It involves planning, monitoring, scheduling, resource allocation, and risk management to ensure timely and successful delivery.

A project is defined as an activity that has a definite start and a definite end require multiple resources and is full of uncertainties on a number of counts.

Project management is "the application of knowledge, skills, tools and techniques to manage resources to deliver customer needs within agreed terms of cost, time, and scope of delivery".

20. Explain size estimation techniques and their types (VIMP).

Ans: Size estimation is an important part of project management. It helps estimate effort, cost, and duration of the software project. Estimating the size of the software to be developed is the very first step to make an effective estimation of the project. A customer's requirements and system specification forms a baseline for Estimation of the size of software. It helps to determine or estimate the size of a software application of a component. Two widely used size metrics are:

Project	LOC	Effort	\$(000)	Pp doc	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6

Fig. Size-Oriented Metrics

❖ Line of Code (LoC)

Definition: Line of Code (LoC) is a metric that measures the number of lines of source code in a software program. It does not include blank lines and comments.

Counting the number of lines of code in computer programs is the most traditional and widely used software metric. It is also the most controversial. Lines Of Code (LOC) is one of the widely used methods for size estimation.

Lines of code written for assembly language are more than lines of code written in C++ From LOC.

Advantages:

- Simple and easy to measure after coding is done
- Useful for measuring programmer productivity
- Helps estimate effort using historical data

Disadvantages:

- Language-dependent (LoC varies by programming language)
- Doesn't measure functionality or logic
- May encourage writing more lines instead of efficient code

Example: A module with 500 LoC can be used to estimate time and cost using past productivity data.

❖ Function Points (FP)

Definition: Function Point is a functionality-based size metric that estimates the size of software by evaluating user-visible features, not lines of code.

Key Components Considered:

- External Inputs (EI)
- External Outputs (EO)
- External Inquiries (EQ)
- Internal Logical Files (ILF)
- External Interface Files (EIF)

Each component is assigned a weight based on complexity (Low, Medium, High), and then total function points are calculated.

Advantages:

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech_](https://www.instagram.com/v2vedtech/) | App Link | [v2vedtech.com](https://www.v2vedtech.com)

- Language-independent
- Can be estimated early in development
- Focuses on business functionality, not implementation

Disadvantages:

- Estimation may vary based on evaluator's judgment
- Requires proper training to calculate accurately

Example: If a system has 5 inputs, 3 outputs, and 2 files, function points can be calculated using a weighted formula.

Difference Between LoC and FP

Aspect	Lines of Code (LoC)	Function Points (FP)
1. Basis	Counts physical lines of code	Measures functionality delivered to the user
2. Language Dependence	Yes, depends on the programming language	No, independent of programming language
3. Timing of Use	Useful after coding starts	Useful in the early stages of project (before coding)
4. Accuracy	Can be misleading (same logic may use more or fewer lines in different languages)	More consistent and standardized
5. Measurement	Easy to count using tools	Requires functional analysis and understanding of scope
6. Usability	Good for maintenance projects	Better for new project estimation and planning

21. Explain UML Diagrams:

- Use Case Diagram
- Sequence Diagram
- Class Diagram

Ans: UML (Unified Modeling Language) is a standard visual modeling language used to analyze, design, and document software systems.

It provides different types of diagrams to represent structure and behavior of a system.

UML diagrams are mainly divided into:

- Structural Diagrams → Show system's static structure (e.g., Class Diagram).
- Behavioral Diagrams → Show system's dynamic behavior (e.g., Use Case, Sequence Diagrams).

1. Use Case Diagram

Definition:

A Use Case Diagram represents the functional requirements of a system — showing how users (actors) interact with system functions (use cases).

It answers “What does the system do?” (not how).

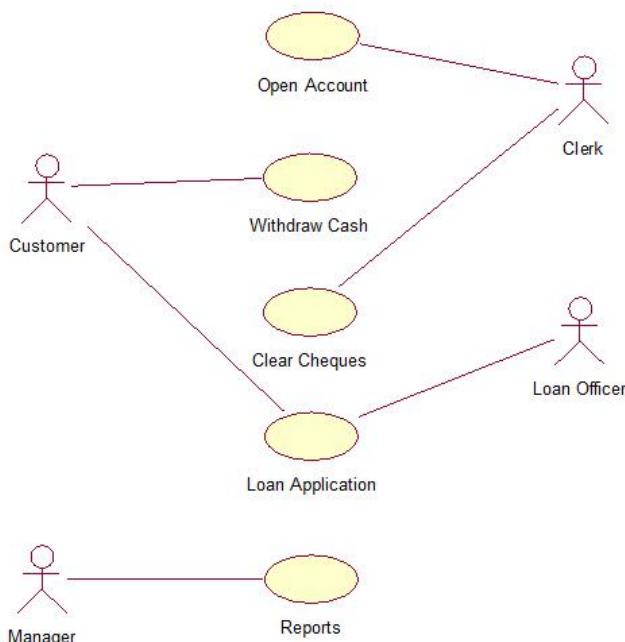
Key Components:

Symbol	Meaning
>User icon with a camera icon	Actor
Circle	Use Case
Circle with a minus sign	System Boundary
Line	Association Line
Include / Extend Relationship	Include: mandatory sub-function. Extend: optional behavior.

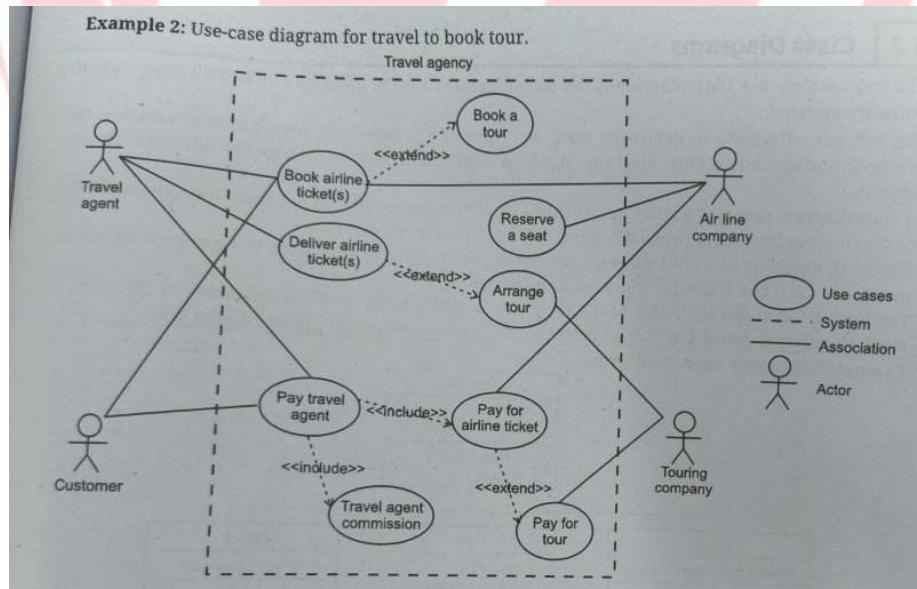
Example: Online Banking System

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com



Example 2



2. Sequence Diagram

Definition:

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

A Sequence Diagram shows how objects interact with each other in a particular order over time. It focuses on the message flow between system components.

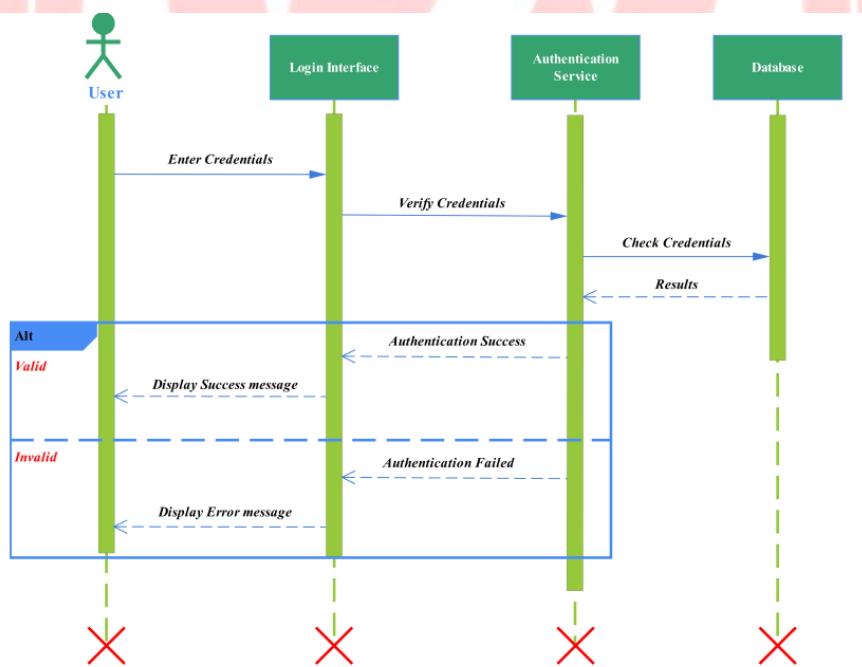
It answers "How does the system behave step by step?"

Key Components:

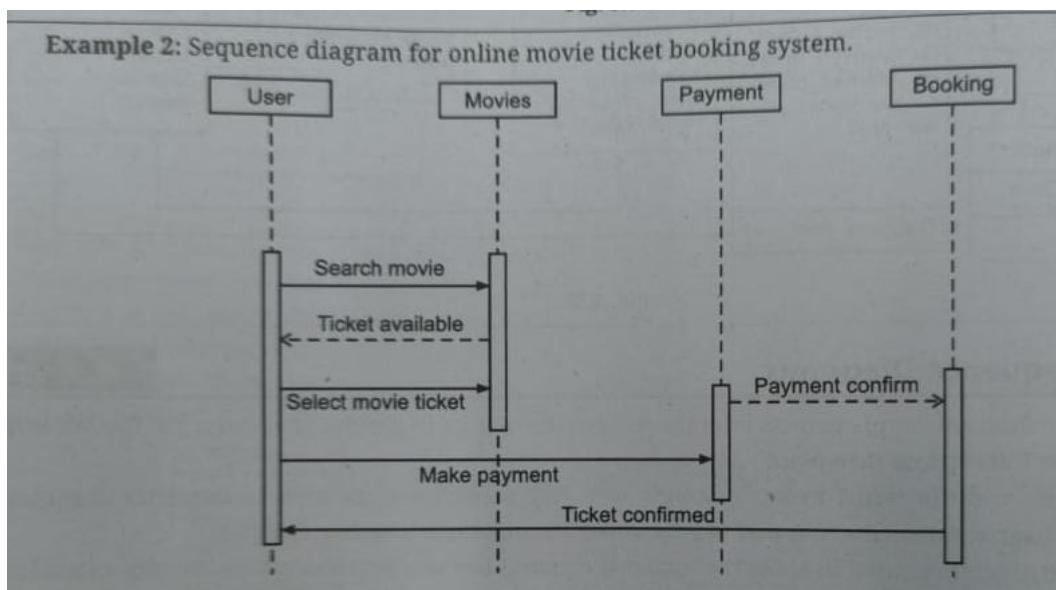
Symbol	Meaning
Actor/Object	Entities participating in the interaction.
Lifeline	A vertical dashed line showing the life of an object.
Activation Bar	A narrow rectangle showing when an object is active.
Message Arrow	Horizontal arrow showing communication between lifelines.

Example: Login Process Sequence Diagram

Step	Interaction
1	User → Login Page: Enter credentials
2	Login Page → Server: Validate credentials
3	Server → Database: Check username/password
4	Database → Server: Send validation result
5	Server → User: Display "Success" or "Error"



Example 2: -



3. Class Diagram

Definition:

A Class Diagram represents the static structure of a system.

It shows classes, their attributes, operations (methods), and relationships between them.

It answers “What are the components of the system?”

Key Components:

Component	Description
Class	Blueprint containing attributes and methods.
Attributes	Variables (data members) of a class.
Methods (Operations)	Functions or behaviors of a class.
Relationships	Connections among classes: Association, Inheritance, Aggregation, Composition.

Example: Library Management System

Classes:

- Book
- Member
- Librarian

Class	Attributes	Methods
Book	title, author, ISBN	issueBook(), returnBook()

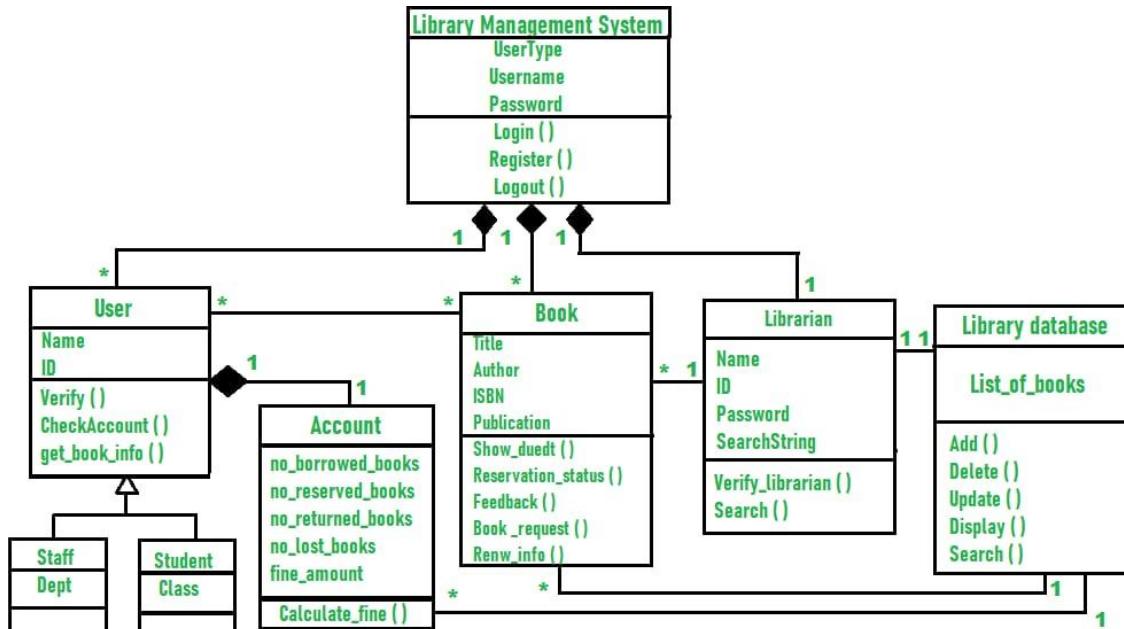
Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

Member	memberID, name	register(), borrowBook()
Librarian	empID, name	addBook(), removeBook()

Relationships:

- Member borrows Book → Association
- Librarian manages Book → Aggregation



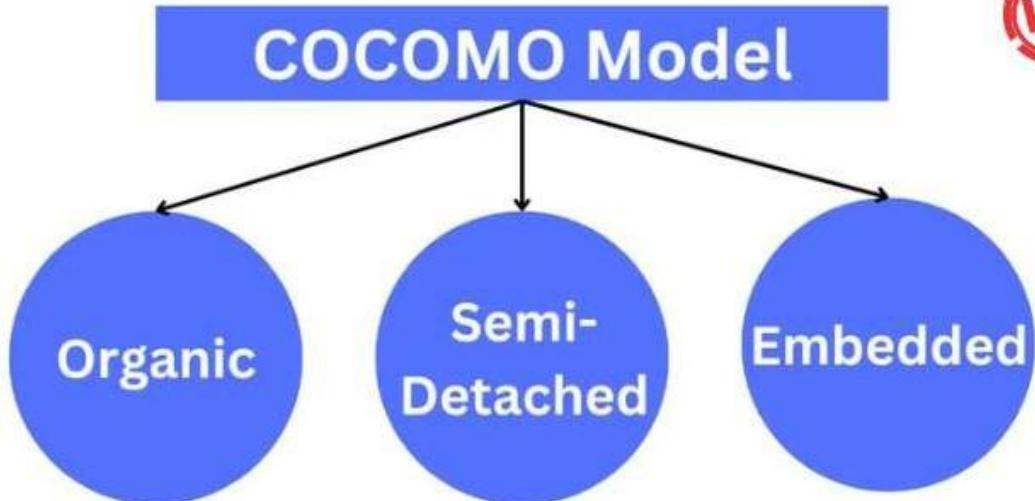
CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

22. Explain COCOMO I & COCOMO II or sums on COCOMO I or II.

Ans: COCOMO I

COCOMO (Constructive Cost Model) is a **cost estimation model** used in software engineering to estimate the **effort, cost, and time** required to develop a software project.

Developed by **Barry Boehm** in **1981**, COCOMO is based on historical project data and uses mathematical formulas.



Types of Projects (Modes):

Mode	Description
Organic	Small projects, experienced teams, well-defined requirements
Semi-Detached	Medium-size projects, mixed experience, moderately flexible
Embedded	Large, complex projects, strict constraints and tight coupling

Purpose:

To estimate:

- **Effort** (in person-months)
- **Development Time** (in months)
- **Cost** (in labor units)

	A	B	C	D
organic	2.4	1.05	2.5	0.38
Semi	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

$0.6+ \leftarrow$ $\rightarrow 0.07$ $\rightarrow 0.03-$
 Effort = $a * (kloc)^b$
 Time = $c * (E)^d$
 Size = E/d
 Productivity = $\frac{kloc}{E}$

Note: - For Embedded, B value $1.12 + 0.07 = 1.19$, But actual value is 1.20, please make a note of this

Problem Statement: Estimate the Effort and Development Time for a software project of 50 KLoC using:

- (a) Organic Mode
- (b) Semi-Detached Mode
- (c) Embedded Mode
- ◊ (a) Organic Mode

Formulas: Effort = $2.4 \times (KLoC)^{1.05}$

Time = $2.5 \times (\text{Effort})^{0.38}$

Step 1: Effort

$$\text{Effort} = 2.4 \times (50)^{1.05}$$

$$50^{1.05} \approx 60.80$$

$$\text{Effort} \approx 2.4 \times 60.80 = \boxed{145.92 \text{ Person - months}}$$

Step 2: Time

$$\text{Time} = 2.5 \times (145.92)^{0.38}$$

$$(145.92)^{0.38} = 6.64$$

$$\text{Time} \approx 2.5 \times 6.64 = \boxed{16.6 \text{ Months}}$$

(b) Semi-Detached Mode

Formulas: Effort = $3.0 \times (\text{KLoC})^{1.12}$

Time = $2.5 \times (\text{Effort})^{0.35}$

Step 1: Effort

$$\text{Effort} = 3.0 \times (50)^{1.12}$$

$$50^{1.12} \approx 79.95$$

$$\text{Effort} \approx 3.0 \times 79.95 = \boxed{239.85 \text{ Person - months}}$$

Step 2: Time

$$\text{Time} = 2.5 \times (200.1)^{0.35}$$

$$200.1^{0.35} \approx 6.38$$

$$\text{Time} \approx 2.5 \times 6.38 = \boxed{15.95 \text{ Months}}$$

(c) Embedded Mode

Formulas: Effort = $3.6 \times (KLoC)^{1.20}$

Time = $2.5 \times (\text{Effort})^{0.32}$

Step 1: Effort

$$\text{Effort} = 3.6 \times (50)^{1.20}$$

$$50^{1.20} \approx 102.33$$

$$\text{Effort} \approx 3.6 \times 102.33 = \boxed{368.38 \text{ Person - months}}$$

Step 2: Time

$$\text{Time} = 2.5 \times (368.38)^{0.32}$$

$$(368.38)^{0.32} = 6.62$$

$$\text{Time} \approx 2.5 \times 6.62 = \boxed{16.55 \text{ Months}}$$

COCOMO II

Definition: The **Post-Architecture Model** is the **most detailed** and widely used model in the **COCOMO II suite**. It is used after the project's **architecture and major requirements have been finalized**.

It helps estimate:

- **Effort** (in person-months)
- *Schedule (time)*
- **Cost**

It supports modern software practices like:

- **Component-based development**
- **Agile and iterative models**
- **Reuse and Object-Oriented Design**

When is it used?

After:

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Requirements are stable
- High-level architecture is defined
- Key project decisions are made (language, platform, team, etc.)

Estimate the effort for a project with:

- Size = 50 KSLOC
- Σ (Scale Factors) = 18
- EAF = 1.15
- A = 2.94

◆ Step 1: Calculate Exponent B

$$B = 0.91 + 0.01 \times 18 = 0.91 + 0.18 = \boxed{1.09}$$

◆ Step 2: Calculate Effort

$$\text{Effort} = 2.94 \times (50)^{1.09} \times 1.15$$

$$(50)^{1.09} \approx 71.10$$

$$\text{Effort} = 2.94 \times 71.10 \times 1.15 = \boxed{240.39 \text{ Person Months}}$$

◆ Step 3: (Optional) Development Time

Assume S = 0.33:

$$\text{Time} = 3.67 \times (240.39)^{0.33} = 3.67 \times 6.10 = \boxed{22.38 \text{ Months}}$$

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp) |

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

23. Explain the Cost Estimation Techniques

Ans.

i) Heuristic cost estimation approach: This technique basically use the concept of learning from the previous project and estimate the cost. The objective is to find a similar system produced earlier and through knowing how the properties of the new system vary from the existing one.

Two classes of different heuristic Estimation Techniques:

- Single variable model
- Multi variable model

1. Single Variable Estimation Models:

It provides a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size.

A single variable estimator model takes the following form:

Estimated Parameter = $c_1 * e_1$

e = characteristic which already have been calculated.

Estimated parameter is the dependent parameter to be estimated. The dependent parameters to be estimated could be effort, duration, staff size etc. c_1 and d_1 are constants- calculated from past projects. COCOMO is one of this type of models example.

2. Multi variable Cost Estimation Model:

It has the following form

Estimated Resources = $c_1 * e_1 d_1 + c_2 * e_2 d_2 + \dots$

e_1 and e_2 are the basic independent characteristics of the software already estimated. c_1, c_2, d_1, d_2 , are constants.

Multivariable Estimation Models are expected to give more accurate estimate compared to the Single Variable Models, since a project parameters is typically influenced by several independent parameters. The independent parameters influence the dependent parameter to different extents.

This is modeled by the constants $c_1, c_2, d_1, d_2, \dots$

ii) Analytical cost estimation approach

Analytical estimation techniques derive the required results starting with basic assumptions regarding the project.

- Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis.
- Halstead's software science is an example of an analytical technique.
- Halstead's software science can be used to derive some interesting results starting with a few simple assumptions. Halstead's software science is especially useful for estimating software maintenance efforts.
- In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts.
- Halstead's Software Science – An Analytical Technique Halstead's software science is an analytical technique to measure size, development effort, and development cost of software products.
- Halstead used a few primitive program parameters to develop the expressions for overall program length, potential minimum value, actual volume, effort, and development time.

iii) Empirical cost estimation approach

Empirical estimation techniques are based on making an educated guess of the project parameters. While using this technique, prior experience with development of similar products is helpful. Although empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years. Two popular empirical estimation techniques are:

1. Expert judgment technique and
2. Delphi cost estimation.

1. Expert Judgment Technique

Expert judgment is one of the most widely used estimation techniques.

- In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly.
- Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate.
- However, this technique is subject to human errors and individual bias.
- Also, it is possible that the expert may overlook some factors inadvertently.
- Further, an expert making an estimate may not have experience and knowledge of all aspects of a project.

2. Delphi cost estimation

- Delphi cost estimation approach tries to overcome some of the short comings of the expert judgment approach.
- Delphi estimation is carried out by a team comprising of a group of experts and a coordinator.
- In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate.
- Estimators complete their individual estimates anonymously and submit to the coordinator. In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation.
- The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators.
- Based on this summary, the estimators re-estimate.

24. Define risk & explain risk management process.

Ans: A risk is a potential problem or uncertain event that may negatively affect the success of a software project. It could cause delays, increased costs, reduced quality, or project failure.

Principles of Risk Management

Here are the main principles that guide effective risk management in software engineering:

1. Risk Identification

- The first and most crucial step.
- Identify all possible risks that could affect the project.
- Risks can be technical, managerial, financial, or external.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#) |

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)

Examples:

- Unrealistic deadlines
- Lack of skilled developers
- Technology change
- Requirement changes

2. Risk Analysis

- Analyze each identified risk to understand its probability (likelihood) and impact (severity).
- Classify risks as low, medium, or high priority.

3. Risk Prioritization

- Rank the risks according to their severity and impact on the project.
- Use a risk matrix to determine which risks need immediate attention.

Example:

Probability	Impact	Priority
High	High	Critical
Medium	High	Major
Low	Low	Minor

4. Risk Planning (Response Planning)

- Prepare strategies to handle each risk.
- There are four main response strategies:

Strategy	Meaning	Example
Avoidance	Eliminate the cause of risk	Use stable technology instead of a new one
Reduction/Mitigation	Reduce the likelihood or impact	Provide training to reduce human error
Transfer	Shift the risk to another party	Take insurance or outsource risky components
Acceptance	Accept and monitor the risk	Acknowledge minor risks with low impact

5. Risk Monitoring and Control

- Continuously track and review identified risks.
- Detect new risks and update the risk management plan regularly.
- Compare actual results with expected outcomes.

6. Communication and Documentation

- Keep all stakeholders informed about risks and mitigation plans.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Maintain a Risk Register that records:
 - Risk ID
 - Description
 - Probability
 - Impact
 - Response strategy
 - Responsible person

25. Explain types of risk (VIMP).

Ans: Risks in software projects are generally divided into the following major categories:

1. Project Risks

These are risks that **affect the project schedule, resources, or management.**

Examples:

- Unrealistic project deadlines
- Budget overruns
- Inadequate planning or poor project estimation
- Lack of skilled manpower
- Miscommunication among team members

Impact:

May cause **delays, increased cost, or failure to meet deadlines.**

2. Technical Risks

These are risks related to the **technology, tools, or technical performance** of the software system.

Examples:

- Use of **new or untested technology**
- Integration issues between software modules
- Performance problems (slow response time)
- Technical failure or design errors

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Incompatibility with hardware or software platforms

Impact:

Can lead to **system failure, rework, or poor software quality**.

3. Business (Organizational) Risks

These are risks that affect the **organization or business goals** of the project.

Examples:

- Change in **business priorities or strategies**
- **Budget cuts or funding withdrawal**
- **Client dissatisfaction or change in requirements**
- **Market competition or regulatory issues**

Impact:

May cause **project cancellation, reduced profits, or loss of reputation**.

4. Operational Risks

These are risks related to **daily operational activities** and **process management**.

Examples:

- Lack of proper process documentation
- Poor communication between teams
- Misuse or loss of critical data
- Hardware or network failures

Impact:

Can affect **team efficiency, data integrity, or system reliability**.

5. External Risks

These are risks that come from **outside the organization's control**.

Examples:

- Natural disasters (floods, earthquakes)
- Political instability
- Changes in government regulations or taxation

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

- Vendor or supplier failures
- Economic downturns

Impact:

May **delay** or **suspend** the project unexpectedly.

6. Security Risks

These are risks that affect the **confidentiality, integrity, and availability** of software data and systems.

Examples:

- Data breaches or cyberattacks
- Unauthorized access to confidential data
- Weak authentication or encryption mechanisms
- Malware or virus attacks

Impact:

7. Requirement Risks

These arise when the **requirements are unclear, incomplete, or frequently changing**.

Examples:

7. Ambiguous or misunderstood requirements
8. Frequent requirement changes by clients
9. Poor requirement documentation

Impact:

Can lead to **rework, scope creep, or customer dissatisfaction**.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

26. Explain RMMM (Risk Mitigation, Monitoring, and Management).

Ans:



RMMM stands for:

Risk Mitigation, Risk Monitoring, and Risk Management

It is a structured risk-handling approach used in software engineering to identify, reduce, track, and respond to potential risks in a project.

Goal of RMMM Strategy

The main goal of the RMMM strategy is:

“To identify risks early in the software process, minimize their impact through planning, monitor them continuously, and handle them effectively when they occur.”

It ensures:

- Project continuity
- Reduced cost and schedule impact
- Higher quality and reliability

Components of RMMM Strategy

1. Risk Mitigation

- Goal: Prevent the risk or reduce its probability or impact
- Involves creating preventive plans

Examples:

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#)

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)

- o Use experienced developers for complex modules
- o Backup plans for third-party tools
- o Allocate extra time in schedule for high-risk tasks

2. Risk Monitoring

- Goal: Track the risk and observe warning signs
- Ongoing activity throughout the project
- Includes regular status reports, risk checklists, and review meetings

3. Risk Management

- Goal: Take proper action when the risk actually occurs
- Execute contingency plans

Examples:

- o Reassign tasks if a developer leaves
- o Use alternative tool if preferred tool fails
- o Adjust schedule/resources when delays happen

27. Explain project scheduling with its principles.

Ans: Project Scheduling is the process of planning tasks, allocating resources, and setting timelines to ensure a software project is completed on time and within budget.

Basic Principles of Project Scheduling

1. Compartmentalization – Divide the project into smaller manageable tasks or modules.
2. Interdependency – Determine relationships between tasks (what depends on what).
3. Time Allocation – Estimate the time required for each task.
4. Effort Validation – Ensure enough resources are available for each task.
5. Defined Responsibilities – Assign responsibilities to team members.
6. Milestones and Deadlines – Mark key dates and review points.
7. Monitoring – Track progress and update the schedule regularly.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

28. Explain PERT and CPM techniques.

Ans: PERT is used to handle uncertainty in project scheduling by using probabilistic time estimates

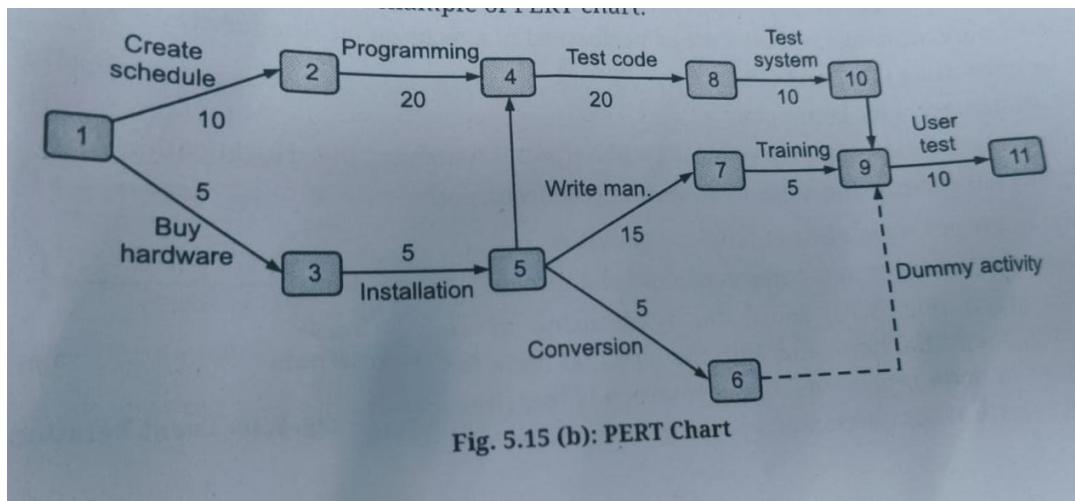


Fig. 5.15 (b): PERT Chart

Key Concepts:

- Each task has three time estimates:
 - Optimistic time (O) – best-case
 - Most likely time (M) – expected case
 - Pessimistic time (P) – worst-case

PERT Steps:

1. List all activities and estimate O, M, P times
2. Calculate Expected Time (TE) for each activity
3. Build the network diagram
4. Find the critical path based on TE values

Advantages:

- Useful for complex or uncertain projects
- Better time estimation using probabilities
- Helps in risk analysis

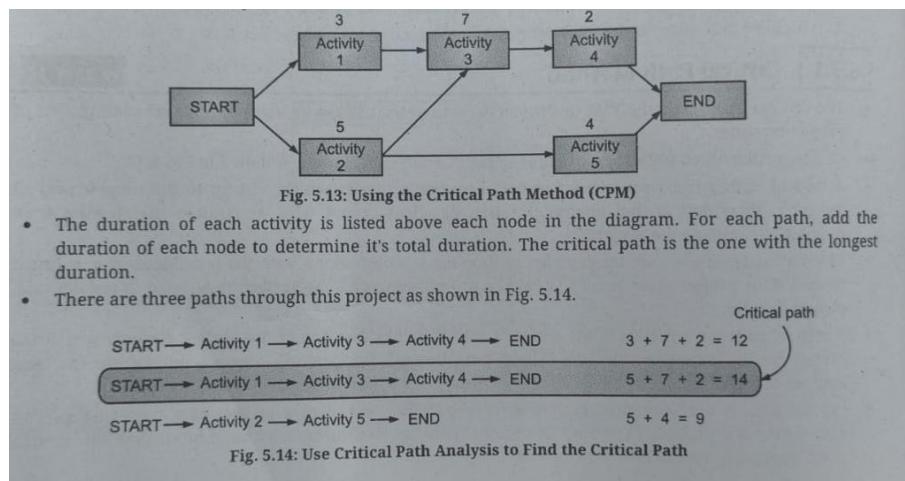
Disadvantages:

- Requires accurate estimates for O, M, P
- More complex than CPM

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

CPM is a technique used to find the longest sequence of dependent tasks (called the critical path) that determines the minimum project duration.



Key Concepts:

- Tasks are represented as nodes or activities
- Each task has a fixed duration
- No uncertainty in task time estimates

CPM Steps:

1. List all tasks and their durations
2. Identify dependencies between tasks
3. Draw a network diagram
4. Calculate the critical path
5. Highlight tasks with zero slack (delays in these will delay the project)

Advantages:

- Helps identify most important (critical) tasks
- Easy to use for well-defined projects
- Optimizes time and resource allocation

Disadvantages:

- Assumes fixed time estimates
- Less effective for uncertain or risky projects

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#) |

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)

29. Define project tracking and explain timeline & Gantt chart

Ans: Project tracking is the process of monitoring and controlling the software project's progress to ensure it stays on schedule, within budget, and meets quality expectations.

It involves:

- Tracking task completion
- Identifying delays
- Updating progress regularly
- Making adjustments if needed

Purpose of Project Tracking:

- Compare planned vs actual progress
- Identify and fix bottlenecks
- Ensure resource efficiency
- Keep the team and stakeholders informed

1. Timeline Chart

Definition: A Timeline Chart shows the sequence of events or tasks in a software project over time. It is a simple horizontal bar representation of the schedule.

Characteristics:

- Time is shown on the horizontal axis
- Tasks/milestones are placed in sequence
- Good for showing major events and deadlines

Advantages:

- Easy to understand
- Great for high-level planning
- Visualizes the order of tasks

Disadvantages:

- Does not show task dependencies
- Not suitable for complex projects

1. Timeline Chart –

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

Example Scenario:

You are developing a student management system.

Milestone	Planned Date
Requirements Gathering	Day 1
Design Completion	Day 4
Coding Start	Day 6
Testing Start	Day 10
Deployment	Day 14

Timeline Chart (Text Representation)

css

Day → 1 4 6 10 14

↓ ↓ ↓ ↓ ↓

Event → [Req] [Design] [Code] [Test] [Deploy]

Gantt Chart

Definition: A Gantt Chart is a detailed bar chart that shows:

- Start and end dates of tasks
- Duration of each task
- Dependencies between tasks
- Progress tracking (completion percentage) It is the most widely used tool for project tracking.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

2. Components of Gantt Chart:

- Vertical axis: Tasks/Activities
- Horizontal axis: Time scale (days/weeks/months)
- Bars: Represent the duration of each task
- Arrows/Lines: Show dependencies between tasks
- Progress shading: Shows % of task completed

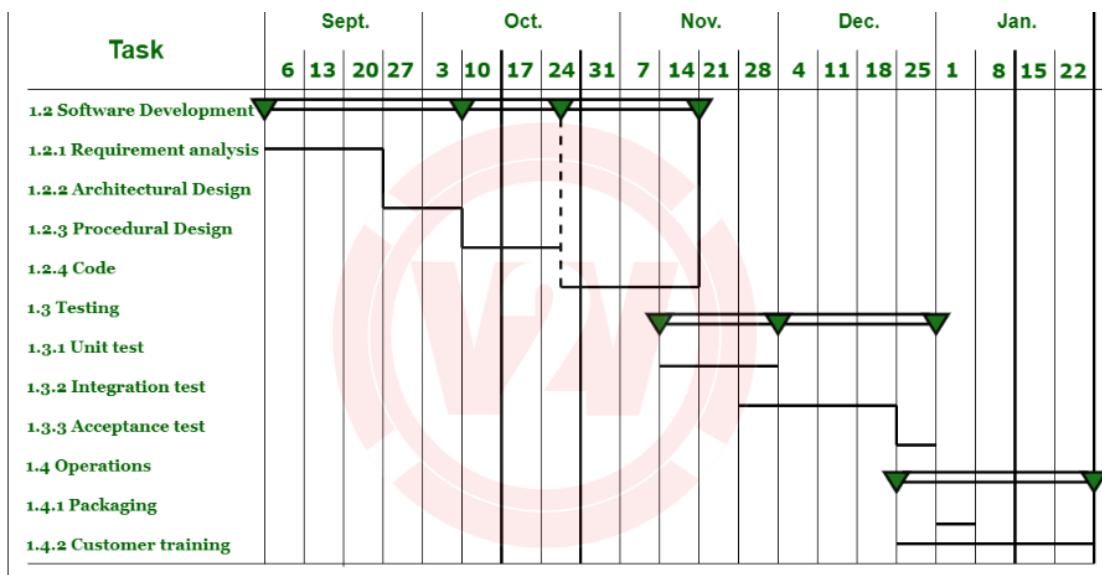
Advantages:

- Clearly shows project schedule and status
- Tracks progress visually
- Helps with resource and time management

Disadvantages:

- Can become complex for large projects
- Needs to be updated regularly

Gantt Chart (Example)



Gantt Chart

30. Explain black-box testing and white-box testing.

Ans: 1. *Black Box Testing*

Definition: Black Box Testing is a **testing approach where the tester does not know the internal structure or code** of the application. The focus is entirely on the **inputs and expected outputs**.

Think of it as testing from the outside.

You treat the software like a “black box” — you don’t care how it works internally.

Types of Black Box Testing:

- **Functional Testing** – verifies what the software does
- **Non-Functional Testing** – includes performance, usability, security
- **Regression Testing** – checks old functionalities after changes

Advantages:

- Testers do **not need programming knowledge**
- Tests are based on **requirements**, not code
- Effective for **system-level and acceptance testing**
- Helps uncover **missing functions or usability issues**

Disadvantages:

- Limited **test coverage** of internal paths
- Cannot find **hidden or logical errors in code**
- Test case creation may be difficult without clear specifications

Example:

Testing the login form:

- Input: Username and Password
 - Output: Login success or error message
- Tester doesn't check **how** authentication happens internally.

2. White Box Testing

Definition: White Box Testing (also called **Glass Box** or **Clear Box Testing**) is a **testing technique** that considers the internal logic, code structure, and flow of the program.

Think of it as testing from the inside.

The tester knows how the software is built and tests each logic path.

Types of White Box Testing:

- **Unit Testing** – individual functions or methods
- **Integration Testing** – testing interaction between units
- **Code Coverage Testing** – condition, branch, and path coverage

Advantages:

- Helps in **optimizing code and improving quality**
- Ensures **maximum code coverage**

- Detects **hidden errors**, logical flaws, dead code
- Useful for **security and loop testing**

Disadvantages:

- Requires **knowledge of programming**
- Time-consuming to test all code paths
- Not suitable for **large-scale system testing** alone

Example:

Testing a login function by:

- Checking code conditions (if, else)
- Validating hash logic for password matching
- Ensuring error messages are handled properly

Black Box vs White Box Testing – Comparison Table

Feature	Black Box Testing	White Box Testing
Knowledge of Code	Not required	Required
Focus	Functionality & output	Internal logic & code paths
Tester Role	End-user or independent tester	Developer or technically skilled tester
Testing Basis	Requirements and specs	Code structure and logic
Detects	Missing functionality, interface issues	Logical errors, hidden bugs, code coverage

Feature	Black Box Testing	White Box Testing
Tools Used	Selenium, QTP, LoadRunner	JUnit, NUnit, SonarQube, Code coverage tools
Test Level	System, Acceptance	Unit, Integration
Time Required	Less (in general)	More (detailed code-level testing)

31. Explain static vs dynamic testing.

Ans: In Software Testing, we use different approaches to detect errors and ensure quality.

Two main testing types are:

Static Testing

Dynamic Testing

These differ in when and how the testing is performed — whether the software code is executed or not.

1. Static Testing

Static Testing is a verification process where the code, documents, and design are checked without executing the program.

It focuses on finding errors early in the development phase (before the program runs).

Focus:

- Reviews, inspections, and walkthroughs of code and documents.
- Detects errors in syntax, logic, design, and documentation.

Objective:

- To ensure the correctness and completeness of the design, code, and documentation before execution.
- To catch defects early, reducing cost and time.

Examples:

1. Code Review: Peers review the source code for errors or poor practices.
2. Walkthrough: Developer explains the code or design to the team for feedback.
3. Inspection: Formal examination of requirements or design documents.
4. Static Analysis Tools: Tools like *SonarQube* or *Lint* detect syntax or code quality issues.

Example Scenario:

A developer reviews another developer's code and finds:

```
if (a = b): # Error: Assignment instead of comparison
```

Advantages:

- Detects early defects before execution.
- Saves time and cost of fixing errors later.
- Improves code quality and documentation accuracy.
- Helps ensure standard coding practices.

Disadvantages:

- Cannot find runtime errors.
- Requires manual effort and experience.
- May not cover execution-based issues (like performance).

2. Dynamic Testing

Dynamic Testing is a validation process where the software is executed to verify that it behaves as expected.

It focuses on runtime behavior of the system.

Focus:

- Checks the functionality, performance, and output of the software by running it with different inputs.

Objective:

- To ensure that the software meets user requirements.
- To detect runtime errors, logical errors, and performance issues.

Examples:

- Unit Testing – Testing individual components.
- Integration Testing – Checking interaction between modules.
- System Testing – Testing the entire system.
- Acceptance Testing – Ensuring software meets user needs.

Example Scenario:

A tester runs a login page program with valid and invalid credentials:

Input	Expected Output
Correct username/password	Login successful
Wrong password	Error message

Advantages:

- Detects runtime and logical errors.
- Ensures software behavior meets the requirements.
- Validates performance and usability.

Disadvantages:

- Requires more time and resources.
- Can detect errors only after execution.
- Fixing late-stage errors is costlier.

Comparison: Static vs Dynamic Testing

Feature	Static Testing	Dynamic Testing
Definition	Testing without executing the code	Testing by executing the code
Phase	Verification phase	Validation phase
Goal	Find errors in documents, design, or code	Check software behavior and functionality
Performed By	Developers, reviewers, QA	Testers or QA engineers
Example Methods	Code review, walkthrough, inspection	Unit, Integration, System testing
Execution	Not required	Required
Type of Errors Found	Syntax, design, documentation errors	Logical, runtime, performance errors
Cost of Fixing Errors	Low (early stage)	High (later stage)
Tools	Lint, SonarQube	Selenium, JUnit, LoadRunner
Execution	Not executed	Executed
Purpose	Verification	Validation
When Done	Early in SDLC	After coding is complete
Error Type	Design, syntax	Logic, performance
Examples	Code reviews, walkthroughs	Unit, system, acceptance testing

32. Explain verification and validation in software testing.

Ans: In Software Engineering, both Verification and Validation are important parts of the software testing process.

They ensure that the software product is:

- Built correctly (Verification)
- Built as per user's needs (Validation)

1. Verification

Verification is the process of checking whether the software is being built correctly according to the design documents, plans, and requirements — without executing the code.

It answers the question:

“Are we building the product right?”

Purpose:

To ensure that the product follows the specifications and design standards during development.

Characteristics:

- A static process (code is not executed).
- Involves reviews, inspections, and walkthroughs.
- Conducted at each phase of development (requirement, design, coding).

Examples:

Activity	Description
Requirement Review	Checking if SRS (Software Requirement Specification) is correct and complete
Design Review	Ensuring that the design meets the requirements
Code Review	Checking for coding standards and syntax errors
Document Verification	Ensuring documentation is clear and accurate

Techniques Used:

- Inspections
- Walkthroughs
- Reviews
- Static analysis tools (e.g., SonarQube, Lint)

Example:

Before coding, developers review the SRS to confirm that all user requirements are clearly stated and no features are missing.

This ensures the system is being built correctly before ~~actual~~ development.

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

Advantages:

- Detects errors early in the development phase.
- Saves time and cost of fixing errors later.
- Ensures consistency and completeness of documentation.

2. Validation

Validation is the process of checking whether the developed software meets the user's actual needs and requirements — by executing the program.

It answers the question:

“Are we building the right product?”

Purpose:

To ensure that the final product works as intended and satisfies user expectations.

Characteristics:

- A dynamic process (code is executed).
- Focuses on the behavior and performance of the software.
- Conducted after coding is complete.

Examples:

Activity	Description
Unit Testing	Testing individual components
Integration Testing	Testing interaction between modules
System Testing	Testing the complete system
Acceptance Testing	Ensuring the system meets user needs

Example:

After developing a login system, testers run it using:

- Valid username/password → Should log in successfully
- Invalid credentials → Should show error message

Techniques Used:

- Functional testing
- System testing
- User acceptance testing (UAT)

30

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](#) |

Insta : [v2vedtech](#) | App Link | [v2vedtech.com](#)

- Performance testing

Advantages:

- Ensures software meets user requirements.
- Detects runtime and logical errors.
- Improves software reliability and user satisfaction.

Comparison: Verification vs Validation

Feature	Verification	Validation
Definition	Ensures the product is built correctly	Ensures the right product is built
Question Answered	Are we building the product right?	Are we building the right product?
Type of Process	Static (no code execution)	Dynamic (code execution)
Performed During	Development phases	After development or during testing
Main Objective	To check design, documents, and code	To check functionality and behavior
Performed By	Developers, QA, reviewers	Testers, end-users
Examples	Reviews, walkthroughs, inspections	Unit testing, system testing, UAT
Cost of Fixing Errors	Low (early detection)	High (late detection)

Simple Example

Activity	Type	Purpose
Checking if login page is designed as per SRS	Verification	Confirm design correctness
Testing login with valid/invalid credentials	Validation	Confirm system works correctly

33. Difference Between QC & QA.

Ans: Both QA (Quality Assurance) and QC (Quality Control) are key components of Software Quality Management, but they serve different purposes in ensuring software quality.

❖ Definition

Aspect	Quality Assurance (QA)	Quality Control (QC)
Definition	QA is a process-oriented activity that focuses on ensuring that proper processes and methods are followed during software development to prevent defects.	QC is a product-oriented activity that focuses on detecting defects in the final product through testing and inspection.
Goal	To prevent defects by improving the development process.	To identify and fix defects in the finished product.

❖ Key Differences

31

Mob No : 9326050669 / 9372072139 | Youtube : [@v2vedtechllp](https://www.youtube.com/@v2vedtechllp)

Insta : [v2vedtech](https://www.instagram.com/v2vedtech/) | App Link | v2vedtech.com

Basis	Quality Assurance (QA)	Quality Control (QC)
Nature	Preventive	Corrective
Focus	On the process used to make the product	On the actual product itself
Objective	To ensure that the process is defined, managed, and followed properly	To ensure the final product meets the required quality standards
Performed by	QA Team / Process Analysts	Testing Team / Test Engineers
Activities Involved	Process definition, audits, training, reviews	Testing, inspection, validation, verification
Timing	Done throughout the development process	Done after development (during testing phase)
Output	Process improvement reports	Defect reports, bug logs
Approach	Proactive (prevents defects)	Reactive (finds and fixes defects)

 Example

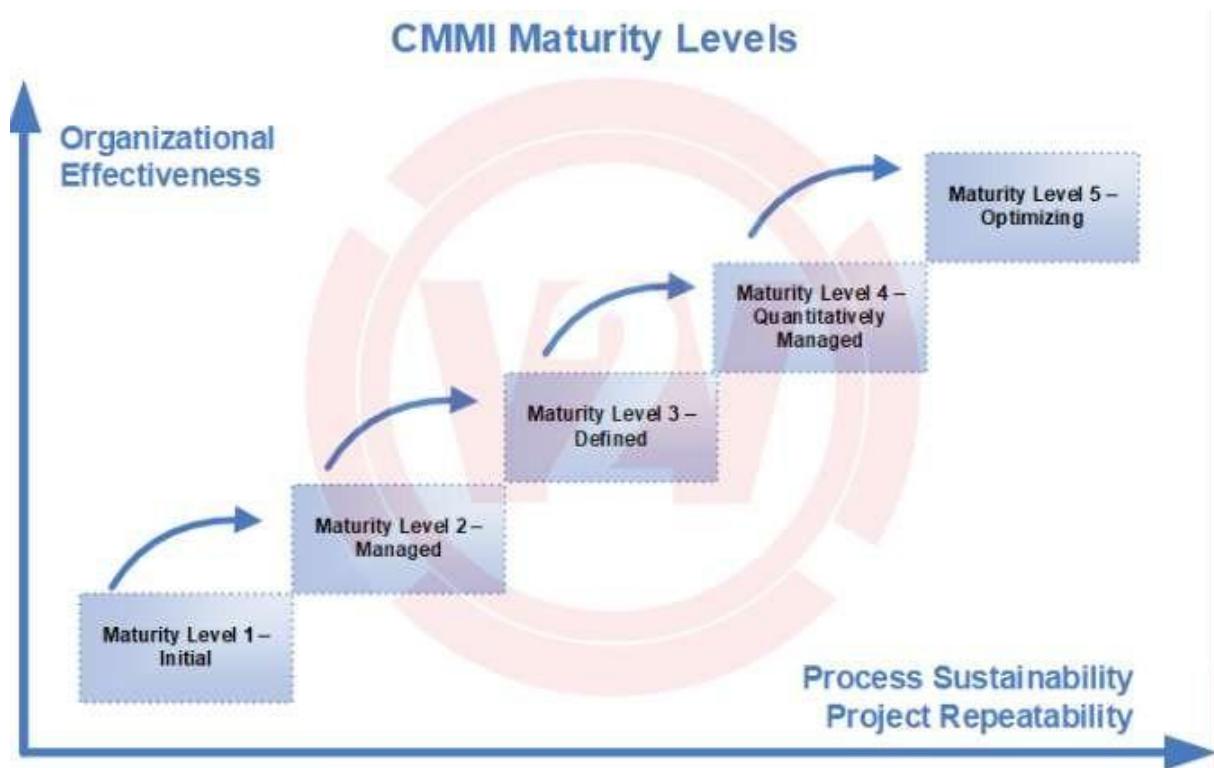
- QA Example:
Conducting process audits, defining coding standards, or reviewing design documents to ensure proper methodology is followed.
- QC Example:
Performing software testing, finding bugs, and verifying if the software works as expected.

34. Explain CMMI (Capability Maturity Model Integration) with its levels.

Ans:

CMMI (Capability Maturity Model Integration)

Definition: CMMI is a **process-level improvement training and appraisal program** that guides software organizations in improving their processes.



CMMI Levels:

Level	Description
1	Initial – Unpredictable process
2	Managed – Basic project management processes exist
3	Defined – Processes are documented and standardized
4	Quantitatively Managed – Processes are measured and controlled
5	Optimizing – Focus on continuous improvement

Benefits:

- Structured and measurable process improvement
- Better risk management
- Consistent quality in projects

2. Continuous CMMI Model:

Continuous maturity models do not classify an organization according to discrete levels. Rather they are finer-grained models that consider individuals or groups of practices. The maturity assessment is not, therefore, a single value but a set of values showing the organization's maturity for each process.

CMMI Levels

The six-point scales assign a level to a process as follows:

Level 0: Not performed: One or more specific goals associated with the process area is not satisfied.

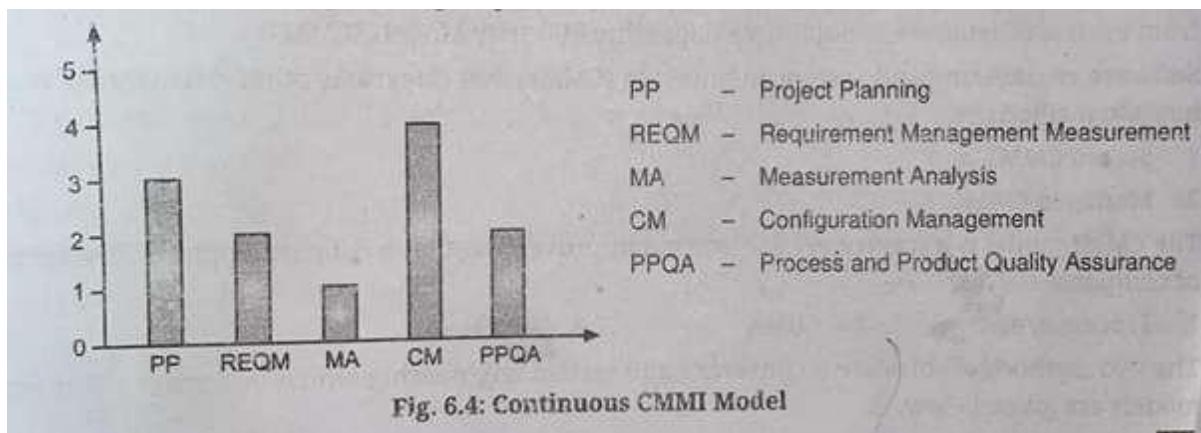
Level 1: Performed: The goals associated with the process area are satisfied and for all processes the scope of the work to be performed.

Level 2: Managed: At this level, the goals associated with the process areas are met and organizational policies are in place that define when each process should be used.

Level 3: Defined: This level focuses on organizational standardization and deployment of processes.

Level 4: Quantitatively Managed: At this level, there is an organizational responsibility to use statistical and other qualitative methods to control sub-processes.

Level 5: Optimizing: At this highest level, the organization must use the process and product measurements to drive process improvement.



Example: A company at **CMMI Level 3** has all project processes well documented and uses defined standards for development.

35. Explain software quality assurance (SQA) processes.

Ans: Software Quality Assurance (SQA) is a systematic set of activities designed to ensure that the software development process and the final product meet the required quality standards and satisfy customer requirements.

It focuses on preventing defects in the software throughout the development lifecycle, not just detecting them during testing.

Main Objective of SQA:

- To ensure that the software development process is well-defined, effective, and follows standards.
- To produce high-quality, reliable, and maintainable software.
- To prevent errors rather than just finding them after coding.

Who Performs SQA?

- The SQA team (independent from the development team).
- Includes QA engineers, auditors, and process managers.

Software Quality Assurance Process – Overview

The SQA process is carried out throughout the Software Development Life Cycle (SDLC) and involves the following key steps

1 Establish Quality Goals and Standards

- Define quality objectives and acceptance criteria for the project.
- Select quality models and standards such as ISO 9001, CMMI, or IEEE standards.
- Decide which metrics will be used to measure quality (e.g., defect density, reliability, maintainability).

2 Develop a Quality Assurance Plan

- Create a Software Quality Assurance Plan (SQAP) that defines:
 - QA activities and responsibilities
 - Schedule and tools
 - Review and audit methods
 - Testing procedures
- The plan acts as a roadmap for ensuring quality throughout the project.

3 Define and Implement Software Development Processes

- Define standard procedures for each SDLC phase (requirements, design, coding, testing, maintenance).
- Ensure all team members follow the same process.
- Provide templates, checklists, and guidelines to maintain consistency.

4 Conduct Reviews and Audits

- Perform regular technical reviews, inspections, and walkthroughs to identify errors early.
- Conduct process audits to verify compliance with defined standards and procedures.

5 Perform Testing Activities

- Testing is part of SQA but not the only activity.
- Conduct unit testing, integration testing, system testing, and acceptance testing to verify and validate the software.

6 Measure, Monitor, and Report Quality

- Use software metrics (like defect density, code coverage, reliability) to measure process and product quality.
- Track progress using QA reports and identify areas needing improvement.

7 Control and Manage Changes

- Implement a change control process to handle modifications in requirements, design, or code.
- Ensure every change is reviewed, tested, and approved.

8 Take Corrective and Preventive Actions

- When issues are found, identify root causes and take corrective actions to fix them.
- Implement preventive measures to avoid similar defects in the future.

9 Continuous Process Improvement

- Continuously improve the quality process using feedback, audits, and lessons learned.
- Adopt models like CMMI Level 5 or PDCA (Plan–Do–Check–Act) cycle.

36. Explain SIX Sigma with DMAIC & DMADV

Ans: Six Sigma is a data-driven quality management approach that aims to reduce defects, improve processes, and ensure near-perfect quality in software or product development.

It focuses on continuous improvement by identifying and eliminating the causes of errors or variations in processes.

Meaning of “Six Sigma”:

- “Sigma (σ)” is a statistical term that measures process variation (i.e., how much a process deviates from perfection).

- Six Sigma means only 3.4 defects per million opportunities (DPMO) — representing 99.99966% quality accuracy.

✍ In simple terms:

The higher the sigma level, the fewer the defects.

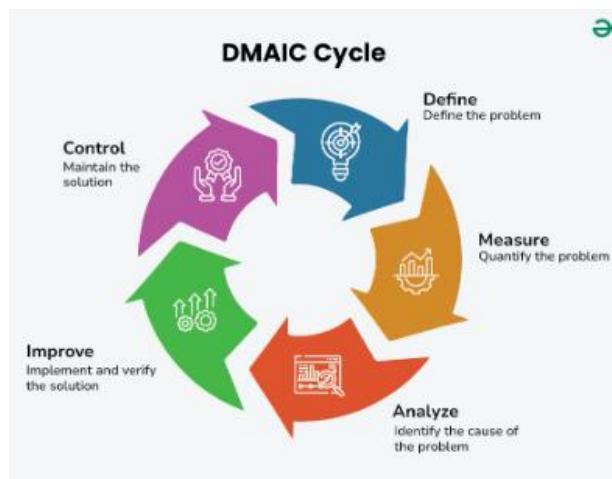
Objectives of Six Sigma:

1. To improve software quality by minimizing defects.
2. To increase customer satisfaction through reliable and consistent performance.
3. To reduce costs by improving process efficiency.
4. To enhance productivity and decision-making using data and statistical analysis.

Six Sigma Methodologies:

There are two main approaches used in Six Sigma:

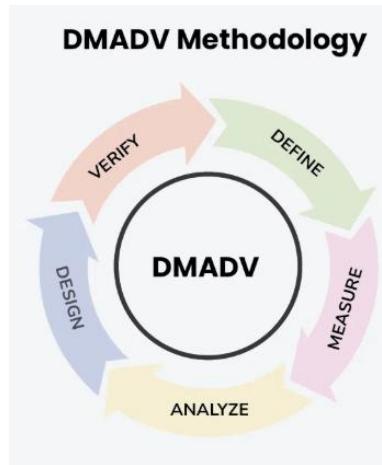
1. DMAIC (for improving existing processes)



Used when a process already exists but needs improvement.

Phase	Description
D – Define	Define the problem and customer requirements.
M – Measure	Measure current performance and collect data.
A – Analyze	Analyze data to find root causes of defects.
I – Improve	Implement solutions to remove causes of defects.
C – Control	Monitor the process to maintain improvements.

2. DMADV (for creating new processes/products)



Used when a new process or product is being developed

Phase	Description
D – Define	Define project goals and customer needs.
M – Measure	Measure and identify critical quality factors.
A – Analyze	Analyze different design options.
D – Design	Design the process or product.
V – Verify	Verify that the design meets customer expectations.

Key Principles of Six Sigma

1. Customer Focus:
The primary goal is to meet or exceed customer expectations.
2. Data-Driven Decision Making:
All decisions are based on statistical data, not assumptions.
3. Process Improvement:
Focus on improving business and software development processes.
4. Employee Involvement:
Encourages teamwork and involvement of every employee.
5. Defect Prevention:
Emphasizes preventing errors rather than fixing them later.
6. Continuous Improvement:
Promotes ongoing efforts to enhance processes and performance.

22. Define Unit Testing , Integration testing , alpa testing , beta testing , acceptance testing , bang bang , performance testing , stress , Software Engineering , defect , bug, error, fault, failure.

Ans:

 1. Software Engineering (Definition)

Software Engineering is the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software.

It applies engineering principles to produce reliable and efficient software systems.

 2. Unit Testing

Unit Testing is a testing technique where individual modules or components of software are tested in isolation to ensure that each part works correctly.

Example: Testing a single function that calculates the total bill in a billing system.

 3. Integration Testing

Integration Testing is the process of testing the interfaces and interactions between integrated modules to ensure they work together correctly.

Types:

- Top-Down: Testing starts from top modules to lower ones.
- Bottom-Up: Testing starts from lower modules to upper ones.
- Big Bang: All modules are combined and tested at once.
- Sandwich (Hybrid): Combines both top-down and bottom-up approaches.

 4. Alpha Testing

Alpha Testing is performed by internal employees/testers at the developer's site before the product is released to users.

Goal: Identify bugs before releasing the software to external users.

 5. Beta Testing

Beta Testing is performed by real users at their locations after Alpha Testing and before the final release.

Goal: Get user feedback and detect real-world issues.

6. Acceptance Testing

Acceptance Testing is done to verify whether the software meets customer requirements and is ready for delivery.

Types:

- User Acceptance Testing (UAT) – Done by end users.
- Business Acceptance Testing (BAT) – Done by business team.
- Contract Acceptance Testing (CAT) – Based on agreement terms.

Goal: Confirm that the software satisfies the business needs.

7. Big Bang Testing

Big Bang Testing is an integration testing approach in which all modules are combined at once and tested together as a complete system.

 *Disadvantage:* Difficult to identify which module caused the failure if an error occurs.

8. Performance Testing

Performance Testing checks how well the software performs under expected workloads, such as speed, scalability, and stability.

Goal: Ensure system works efficiently under normal conditions.

 *Example:* Measuring response time of a website.

9. Stress Testing

Stress Testing evaluates the system's behavior under extreme or beyond-limit conditions, such as heavy load or low memory.

Goal: Check how the system handles overload or crashes gracefully.

 *Example:* Simulating 10,000 users on an app designed for 5,000.