Regular/Back/Scholarship Exam – 2081/2082 Chaitra/Baishakh
Diploma in Information Technology

1. What are deliverables and milestones? Explain the steps in software process.

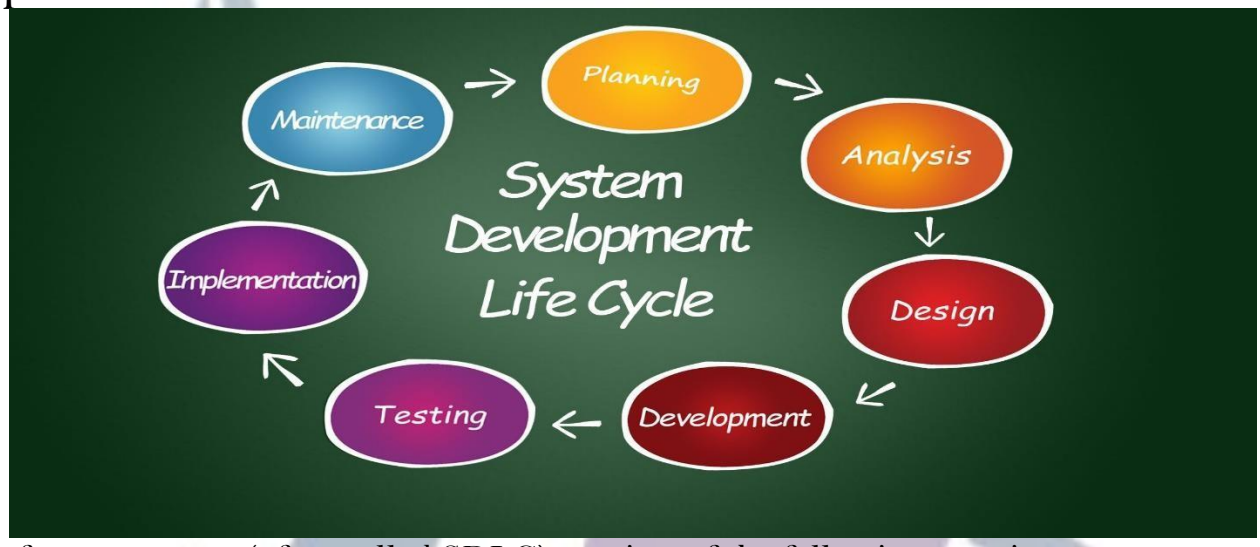
Deliverables

Deliverables are the final, measurable outputs of a software project which prove progress. They include documents, design reports, source code, test results, manuals etc.They are tangible items delivered to client at the end of each phase.

Milestones.

Milestones are important checkpoints or events in the project timeline that show completion of a stage.They help in tracking progress, controlling schedule and managing deadlines. They are intangible events, not physical items, like completion of design or testing phase.

2: Steps in Software Process.



The software process (often called SDLC) consists of the following 6 main steps:

1. Requirement Gathering & Analysis: This is the first step where the team communicates with the client to understand what the software needs to do. All requirements are documented in an **SRS (Software Requirement Specification)** document.

2. System Design: In this phase, the "blueprint" of the software is created. Developers decide the internal architecture, database structure, and how different parts of the system will interact with each other.

3. Implementation (Coding): This is where the actual programming happens. Developers write code using specific programming languages (like Java, Python, or C++) based on the design documents created in the previous step.

4. Testing: Once the code is written, it is checked for bugs or errors. Testers ensure that the software works correctly and meets all the requirements mentioned by the client.

5. Deployment: After the software is tested and found to be error-free, it is installed at the client's site or released to the public market for use.

6. Maintenance: The process doesn't end after release. Maintenance involves fixing any bugs that appear after use or updating the software to meet new requirements or technologies

.

Explain COCOMO Model

Definition

COCOMO (Constructive Cost Model) is a software cost estimation model. It predicts the effort and cost required to develop software based on the size in KLOC (thousends of lines of code).

Types of COCOMO

- Basic COCOMO: Simple model based on size only.
- Intermediate COCOMO: Includes additional cost drivers.
- Detailed COCOMO: Most detailed, includes phase-specific estimates.

Formula

$$\text{Effort (Person-Months)} = a \times (\text{KLOC})^b$$

Constants Table

Project Type	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embelded	3.6	1.20

Example

Given : Project size = 50 KLOC (Organic type).

Effort = $2.4 \times (50)1.05$

Step-bte : $1.05 = 1 + 0.05$, so $(50)^{1.05} = (50)^1 \times (50)^{0.05}$

Using $(50)^{0.05} \approx 1.3797$

Effort $\approx 2.4 \times 50 \times 1.3797 = 165.56$ Person-Months.

Result

165.56 Person-Months

3.Explain different levels of DFD. Draw DFD of library management system up to level 2.

Levels of DFD

1. Level 0 (Context Diagram):

- This is the highest level of DFD.
- The entire system is shown as one single process (a circle).
- Only external entities (like users, librarian, admin) and their data flows are shown.
- Purpose: To give a big picture view of the system.
- Example: In Library System, Student and Librarian interact with the "Library Management System" process.

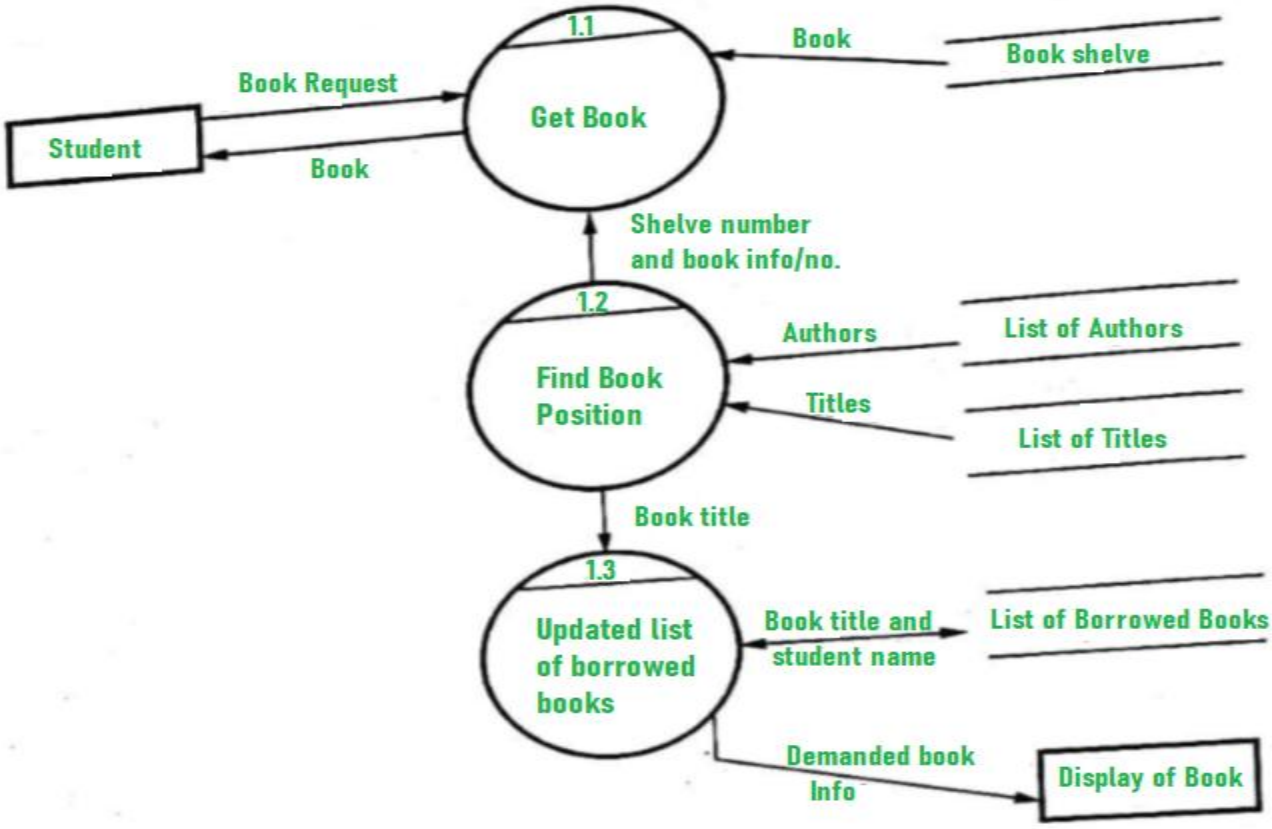
2. Level 1:

- Breaks the single process of Level 0 into major sub-processes.
- Shows data stores (like Book Database, Student Records).
- Explains how inputs are transformed into outputs through different processes.
- Example: Library System divided into Book Issue, Book Return, Catalog Management.

3. Level 2:

- Provides detailed breakdown of Level 1 processes.
- Each sub-process is further divided into smaller steps.
- Shows internal working of the system clearly.
- Example: Book Issue → Check availability → Update Student Record → Issue Book.

1. **Draw DFD of library management system up to level 2.**



Level 2 DFD

4. What is feasibility analysis? Explain cost benefit analysis technique with example.

Definition:

Feasibility analysis is the study to check whether a **software project is practical and possible**.It helps to decide **if the project should be undertaken**.It ensures resources are used efficiently and risks are reduced.

Cost-Benefit Analysis (CBA) [8 Marks]

Cost-Benefit Analysis is the most common technique used for **Economic Feasibility**. It compares the total expected **costs** of developing the software against the total expected **benefits** it will provide over its lifetime.

1. Types of Costs and Benefits

To do a CBA, we categorize items as:

- **Tangible:** Things we can measure in money (e.g., hardware cost, salary, increased sales).
- **Intangible:** Things hard to measure in money (e.g., brand image, employee morale).

2. Steps in CBA

1. **Identify Costs:** List all expenses (Development, Licenses, Training, Maintenance).
2. **Identify Benefits:** List all gains (Cost savings, increased revenue, faster processing).
3. **Assign Monetary Value:** Put a dollar/rupee value on every item.
4. **Compare:** Use formulas like **ROI (Return on Investment)** or **Payback Period**.

3. Example (Practical Case)

Imagine a company wants to replace its manual billing system with **Automated Billing Software**.

A. Estimated Costs (Initial Year):

- Software Development: ₹5,00,000
- Hardware/Servers: ₹1,00,000
- Employee Training: ₹50,000
- **Total Cost = ₹6,50,000**

B. Estimated Benefits (Per Year):

- Reduction in manual labor (2 staff members saved): ₹4,00,000
- Reduced errors/fines: ₹1,00,000
- Increased speed (more customers handled): ₹1,00,000
- **Total Benefits = ₹6,00,000 per year**

5. Explain about size estimation and software risk management.

Software Risk Management

. **Software Size Estimation**.

Size estimation is the process of predicting how big the software will be. This is crucial because size determines the effort, time, and cost of the project.

Main Techniques:

1. Lines of Code (LOC): The simplest method where we count the total number of lines of source code. However, it depends heavily on the programming language used.
2. Function Points (FP): A more modern method that measures size based on the functionality provided to the user (e.g., number of inputs, outputs, queries, and files). It is independent of the programming language.

Software Risk Management

Risk management is the process of identifying, analyzing, and responding to potential problems before they happen.

The 4 Main Steps:

1. Risk Identification: Listing all possible risks (e.g., staff quitting, budget cuts, or technical failure).
2. Risk Analysis: Determining how likely the risk is to happen and how much damage it will cause.
3. Risk Planning: Deciding what to do if the risk occurs (e.g., keeping a backup of data or hiring extra freelancers).
4. Risk Monitoring: Keeping an eye on the project to see if any new risks appear or if old ones are becoming more serious.

6. What are the objectives for input design? Explain COCOMO..

Objectives of Input Design

Definition: Input design is the link between the user and the information system. It focuses on how data enters the system for processing.

The main objectives are:

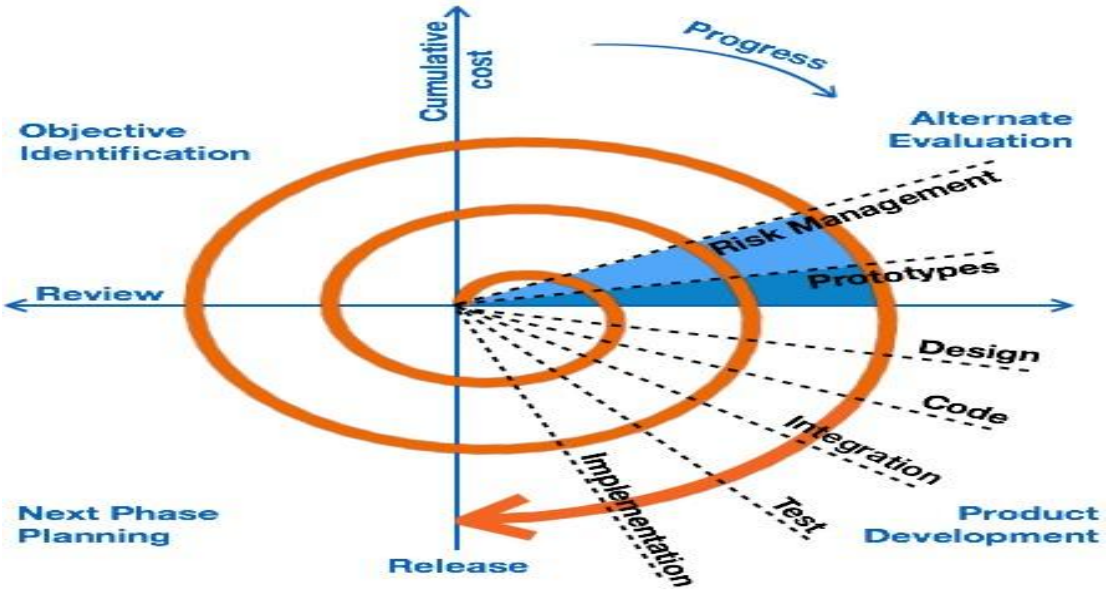
1. **Accuracy:** The design should ensure that data is entered correctly. Using "Validation Rules" (like checking if a phone number has 10 digits) helps prevent errors.
2. **Simplicity:** Input screens should be easy to understand. If a form is too complex, users will make mistakes.
3. **Efficiency:** It should minimize the effort of the user. For example, using **Radio Buttons** for Gender or **Date Pickers** instead of typing the full date.
4. **Consistency:** Every page of the software should have a similar input style. This builds user confidence and reduces training time.
5. **Minimize Volume:** Only necessary data should be asked. If the system can calculate a value (like "Age" from "Date of Birth"), the user shouldn't have to type.

Explain COCOMO..

COCOMO (Constructive Cost Model) is a software cost estimation model that predicts effort, time, and cost based on project size measured in KLOC (thousands of lines of code). It was developed by Barry Boehm in 1981 and remains one of the most widely used models in software engineering.

2. What is SDLC? Explain about the spiral model in software development.

SDLC is a step-by-step process to develop software systematically.It covers phases like requirement, design, coding, testing, deployment, maintenance.It ensures quality, cost-effectiveness and timely delivery of software.



Spiral Model

The Spiral Model is a risk-driven and iterative software development model. Development progresses in a spiral shape consisting of multiple loops, where each loop represents a complete development cycle.
The number of loops depends on project size, complexity, and risk.
Each loop includes:

- Planning
- Risk analysis
- Development
- Evaluation

Phases of the Spiral Model

Focus is on managing risk through multiple iterations of the software development process.
Each phase of the Spiral Model is divided into four Quadrants.

1. Objectives Definition

- Project goals and requirements are identified
- Functional and non-functional requirements are analyzed
- Possible solutions are explored

2. Risk Analysis and Resolution

- Risks related to cost, schedule, performance, and technology are identified
- Best solution is selected
- Prototypes are built to reduce risks

3. Development and Testing

- Selected features are designed, developed, and tested
- The next working version of the software is created

4. Review and Planning

- Customers evaluate the current version
- Feedback is collected
- Planning for the next iteration begins\

7. Define software quality. Explain about capability maturity model (CMM).

Software quality is the **degree to which a software product meets user requirements and expectations**.It shows how well the software performs **correctly, reliably, and efficiently** A quality software is **easy to use, maintain, and modify**..

Explain Capability Maturty Model (CMM).

Capability Maturity Model (CMM) is a process improvement model developeed by SEI.
It provides a framework to assess and soines development process maturity.

Purpose of CMM

Why CMM is used:

- To standardize and optimize development processes.
- To improve software quality and management.
- To reduce risks and improve predictability.

Levels of CMM

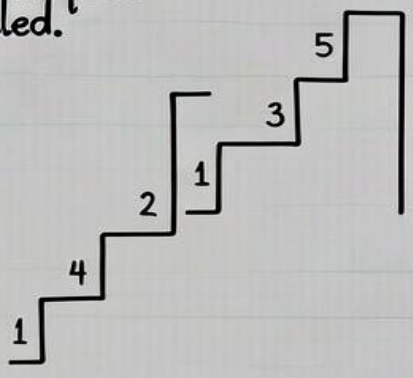
Level 1 – Initial: Ad chaatic process.
- Depends on individual effort.
- High risk and unpredictable results.

Level 2 – Repeatable:
- Basic projett management practices.
- Past success can be repeated.
- Requirements and schedules are controlled.

Level 3 – Defined:
- Standard and documenent processes
- Organization-wide process standards.
- Training programss are available.

Level 4 – Managed:
- Process performance is measured techniques.
- Quality and productiivity are controlled.

Level 5 – Optimizing:
- Continuous process improvement.
- Focus on defect prevention.
- Use of new tools and technologies.



Advantages of CMM
- Improved software quality.
- Better project control.
- Reduced cost and rework.
- Increased customer satisfaction.

Conclusion: CMM helps organizations develop reliable and high-quality software by standardizing and optimizing development processes.

- Milestones include Completion of Requirement Analysis, Completion of Design Phase, Completion of Testing Phase, and System Deployment.

2. Explain any four role performed by system analyst. What are the role of management in software development?

our Roles Performed by a System Analyst.

A System Analyst is like a "Project Architect." They act as a bridge between the business users and the technical team.

1. **Requirement Gatherer & Investigator:** The analyst's first job is to meet with clients and users to understand their problems. They investigate the current system and document exactly what the new software should do.
2. **Problem Solver (Designer):** After understanding the requirements, they design a logical solution. They create "blueprints" like Data Flow Diagrams (DFDs) and Entity-Relationship Diagrams (ERDs) so that developers know what to build.
3. **Communication Manager (The Bridge):** They translate "Business Talk" into "Technical Talk." They explain the client's needs to the programmers and explain technical limitations back to the client.
4. **Change Agent:** When a new system is introduced, people are often afraid of change. The analyst acts as a motivator, showing the users how the new system will make their work easier and providing them with training.

Roles of Management:

1. Planning – Define objectives, scope, and schedule of the project.
2. Resource Allocation – Provide manpower, budget, and tools.
3. Monitoring & Control – Track progress, set milestones, and ensure deadlines are met.
4. Quality Assurance – Ensure standards, testing, and reviews are followed.
5. Risk Management – Identify risks (cost, schedule, technical) and prepare backup plans.
6. Decision Making – Approve deliverables, resolve conflicts, and guide the project direction.

9. What are the advantages of software maintenance? Explain any one software maintenance model.

Advantages:

1. Error correction – Fix bugs and problems found after release.
2. Performance improvement – Optimize speed and efficiency.
3. Adaptation – Adjust software to new hardware, OS, or environment.
4. User satisfaction – Keep system reliable and useful for users.
5. Extended life – Increases the lifespan of the software product.

software Maintenance Model – Iterative Enhancement Model

Explanation:

- In this model, software is improved step by step through repeated cycles (iterations).
- Each cycle includes analysis, design, implementation, and testing.
- Maintenance is not done in one big step, but in small increments.
- New features, bug fixes, and performance improvements are added gradually.

Steps in Iterative Enhancement Model:

1. Analyze – Identify problems or new requirements.
2. Design – Plan changes or enhancements.
3. Implement – Apply modifications (coding).
4. Test – Verify correctness and performance.
5. Release – Deliver updated version to users.
6. Repeat – Continue with next iteration when new needs arise.

Advantages:

- Continuous improvement of software.
- Users get updates regularly.
- Easier to manage risks and errors.
- Keeps system up-to-date with changing environment.

10. Write short notes on.

(a) System Analyst – Deep Note

A **System Analyst** is a professional who studies business problems and designs information systems to solve them.

- Acts as a **bridge between users and developers**: understands user requirements and translates them into technical specifications.
- **Key Responsibilities:**
    1. **Requirement Gathering** – Collects needs from users and stakeholders.
    2. **System Study** – Analyzes existing processes and identifies problems.
    3. **Design Solutions** – Prepares models, diagrams, and specifications for developers.
    4. **Communication** – Explains technical details to management and users in simple language.
- **Importance:** Ensures the final system is **efficient, user-friendly, cost-effective, and meets business goals**.

(b) Testing Tool – Deep Note

- **Testing tools** are software applications used to test other software for quality, performance, and reliability.
- They help in **automating the testing process**, saving time and reducing human error.
- **Types of Testing Tools:**
    1. **Functional Testing Tools** – Check correctness of functions (e.g., Selenium).

2. **Performance Testing Tools** – Measure speed, load, and stress (e.g., JMeter).
3. **Automation Tools** – Execute repetitive test cases automatically.

- **Advantages:**
  - Faster testing process.
  - More accurate results.
  - Detects bugs early → reduces cost of fixing.
- **Example:** Using Selenium to test a login page by automatically entering username and password and checking output.

(c) Entity Relationship Diagram (ERD) – Deep Note

- ERD is a **graphical representation of entities, attributes, and relationships** in a database system.
- **Components:**
  1. **Entities** – Objects like Student, Book, Course.
  2. **Attributes** – Properties like Name, Roll No, Book Title.
  3. **Relationships** – Connections like "Student borrows Book".
- **Importance:**
  - Helps in **database design**.
  - Provides clear visualization of data flow.
  - Reduces redundancy and improves efficiency.
- **Example:** In a Library System ERD: *Student* entity connected to *Book* entity through "Borrows" relationship.

(d) Return on Investment (ROI) – Deep Note

- ROI is a **financial measure** used to check profitability of an investment.
- It compares the **gain from investment** with the **cost of investment**.
- **Formula:**

$$\text{ROI} = \frac{\text{Total Benefits} - \text{Total Costs}}{\text{Total Costs}} \times 100$$

- **Importance:**
  - Helps management decide whether a project is worth investing.
  - Used in feasibility analysis and project evaluation.
- **Why it's used:**
- To compare two different projects and see which one is more profitable.
- To convince stakeholders that the software will save or earn them money in the long run.
- **Result:** A higher ROI percentage means the project is a very good investment.
-

Regular Exam- 2081 Jestha/Ashadh

Diploma in Information Technology

1. Explain deliverables and milestones with an example. Describe the term measures, metrics and measurement.

Deliverables

Deep Explanation.

Deliverables are the tangible outputs or products created during different phases of a project. They are measurable, verifiable, and handed over to the client or stakeholders as proof of progress. Deliverables can be documents, reports, software modules, test results, or manuals.

Key Points:

1. Tangible outputs – actual items produced during the project.
2. Measurable and verifiable – can be checked against requirements.
3. Client-oriented – delivered to stakeholders for approval.
4. Phase-wise – produced at the end of each stage (analysis, design, coding, testing).
5. Ensures progress – shows that the project is moving forward.

Example:

In a Library Management System project:

• Deliverables include Requirement Specification Document, Design Report, Tested Software, and User Manual.

Milestones

Explanation.

Milestones are intangible checkpoints or events in the project timeline. They do not produce physical items but mark the completion of a major activity or phase. Milestones help in tracking progress, controlling schedules, and managing deadlines.

5 Main Points:

1. Intangible events – represent completion, not physical products.
2. Time-based – linked to project schedule and deadlines.
3. Progress tracking – used to monitor project advancement.
4. Management tool – helps managers control and plan resources.
5. Risk control – ensures project stays on track.

Example:

In a Library Management System project:

8. What is software testing? Differentiate between black box testing and white box testing..

Definition:

Software testing is the process of checking and verifying software to ensure it works correctly.

It helps to find errors, defects, and gaps between expected and actual results.Main aim: deliver quality software to the user.

Difference between Black Box and White Box Testing

S.No	Feature	Black Box Testing	White Box Testing
1	**Definition**	Testing based on external specifications and requirements.	Testing based on internal logic, code structure, and design.
2	**Knowledge**	Tester does **not** need to know the internal code or logic.	Tester **must** have full knowledge of the internal code.
3	**Main Goal**	To check if the software meets user requirements (Functional check).	To check the quality of code, loops, and logic (Structural check).
4	**Performed By**	Usually done by independent **Testers**.	Usually done by **Software Developers**.
5	**Levels**	Suitable for **System Testing** and **Acceptance Testing**.	Suitable for **Unit Testing** and **Integration Testing**.
6	**Programming**	No programming knowledge is required.	Strong programming and logic skills are required.
7	**Techniques**	Boundary Value Analysis (BVA), Equivalence Partitioning.	Statement Coverage, Branch Coverage, Path Testing.
8	**Visibility**	It is like looking at a "closed box" (Only inputs/outputs).	It is like looking at a "glass box" (Internal paths are visible).
9	**Time**	Less time-consuming; can start after development is done.	Very time-consuming; requires detailed analysis of every line of code.

3. What do you mean by DFD? Explain different levels with suitable example.

A Data Flow Diagram (DFD) is a graphical tool used to represent how data flows inside a system. It shows where data comes from, how it is processed, where it is stored, and where it goes. DFD is widely used in software engineering because it gives a clear picture of the system without technical coding details.

Explain Different Levels with Suitable Example

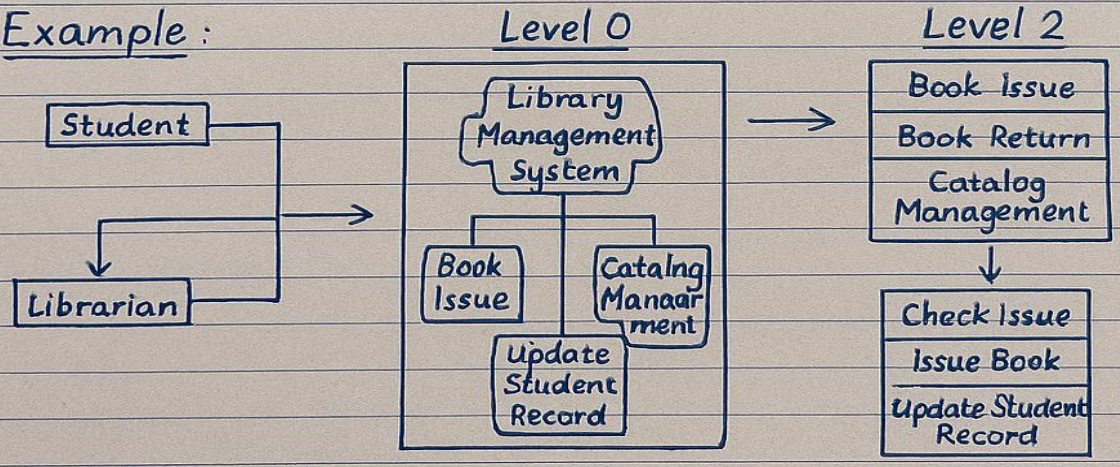
Definition: Levels represent progressive detail from general to specific, in software process or abstraction. They help in understanding complexity step-by-step.

Level 0: Single process represents overall system, with data flows between system and external entities. It provides high-level understanding of system scope.

Level 1: Main process is broken into sub-processes, showing data flow between them and entities. Adds more clarity and detail for analysis.

Level 2: Detailed breakdown of Level 1 processes, with sub-process interactions and data stores. Required for implementation or detailed analysis.

Example:



Conclusion: Multiple levels are necessary to progressively reveal details for effective system analysis and design.

Feature	Reverse Engineering	Re-engineering
Direction of Work	**Backward process**: code → design → specifications.	**Forward process**: modify or rebuild to create **improved version**.
Focus Area	Understanding **what the software does and how it works internally**.	Focused on **better efficiency, usability, maintainability**, and sometimes adding features.
Output	Documentation: **design diagrams, architecture, specifications**.	Updated software: **new version, optimized code, improved functionality**.
Dependency	Requires **existing code** to analyze.	Requires **existing design/code**, but may produce **new code or improved system**.
Example	Studying a **legacy payroll system** to extract design information.	Updating the payroll system to **work on modern platforms** with better reporting.
Benefits	Helps in **knowledge retention**, documentation, and understanding legacy systems.	Improves **performance, reduces maintenance cost, extends software life**, and makes it compatible with new technology.
Risk/Challenge	May **not produce working code**, only understanding.	Requires careful **planning, testing, and validation** to avoid introducing errors.

Why Do We Need Software Maintenance?,

Software maintenance is essential because software is never static—it must evolve to remain useful. Key reasons include:

- Corrective Maintenance 🩹

Fixing bugs and errors discovered after deployment.

- Adaptive Maintenance 🔧

Updating software to work with new hardware, operating systems, or environments.

- Perfective Maintenance 🚀

Enhancing performance, usability, or adding new features based on user needs.

- Preventive Maintenance 🛡

Restructuring code, cleaning up design, and preventing future problems.

6. Define software quality. Explain about Capability Maturity Model (CMM).

Software Quality is the degree to which a software system meets its specified requirements and satisfies the expectations of the end-user. In simpler terms, if a software does exactly what it is supposed to do, without crashing, and is easy to maintain, it is considered high-quality

Explain Capability Maturty Model (CMM).

Capability Maturity Model (CMM) is a process improvement model developeed by SEI.
It provides a framework to assess and soines development process maturity.

Purpose of CMM

Why CMM is used:
- To standardize and optimize development processes.
- To improve software quality and management.
- To reduce risks and improve predictability.

Levels of CMM

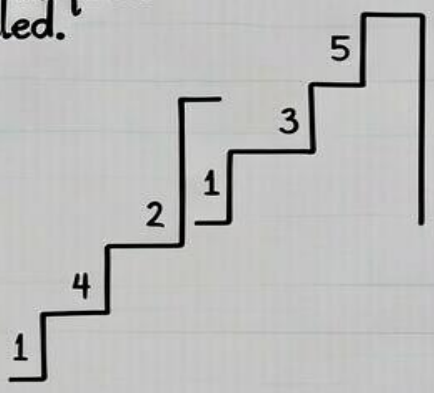
Level 1 – Initial: Ad chaatic process.
- Depends on individual effort.
- High risk and unpredictable results.

Level 2 – Repeatable:
- Basic project management practices.
- Past success can be repeated.
- Requirements and schedules are controlled.

Level 3 – Defined:
- Standard and documenent processes.
- Orqanization-wide process standards.
- Training programss are available.

Level 4 – Managed:
- Process performance is measured techniques.
- Quality and productiivity are controlled.

Level 5 – Optimizing:
- Continuous process improvement.
- Focus on defect prevention.
- Use of new tools and technologies.



Advantages of CMM
- Improved software quality.
- Better project control.
- Reduced cost and rework.
- Increased customer satisfaction.

Conclusion: CMM helps organizations develop reliable and high-quality software by standardizing and optimizing development processes.

7. What are the objectives for output design? Explain software risk management.

Output design focuses on how system results are presented to users. The goal is to make outputs clear, accurate, useful, and easy to understand. Good output design improves decision-making, user satisfaction, and system efficiency.

Objectives.:

1. Clarity – Outputs should be easy to read and interpret.
2. Accuracy – Data must be correct and reliable.
3. Relevance – Only useful and necessary information should be shown.
4. Timeliness – Outputs must be delivered at the right time.
5. Format & Layout – Use proper structure (tables, charts, forms) for better understanding.

Software Risk Management

Software Risk Management is the process of identifying, analyzing, and responding to potential problems before they happen. In software engineering, a "Risk" is an uncertain event that can negatively impact the project's schedule, cost, or quality.

The Risk Management Process:

1. **Risk Identification:**

Listing all possible things that could go wrong.

- *Example:* A key developer leaving the company, or the client changing requirements halfway through.

2. **Risk Analysis (Assessment):**

Evaluating the "Probability" (chance of happening) and "Impact" (how much damage it will cause) of each risk.

- *Formula:* $Risk\ Exposure = Probability \times Impact$

3. **Risk Planning:**

Creating a strategy to handle the risks. You can:

- **Avoid it:** Change the plan to remove the risk.
- **Transfer it:** Make it someone else's problem (like buying insurance).
- **Mitigate it:** Take steps to reduce the impact if it happens.

4. **Risk Monitoring:**

Continuously checking the project to see if new risks appear or if existing risks are becoming more dangerous.

8. What is software verification? Describe the different software testing process.

Software verification is the process of checking whether the software meets its specified requirements. It ensures that the product is being built correctly according to design documents, standards, and specifications. Verification answers the question: "Are we building the product right?"

Different Testing Processes

1. Unit Testing
- Tests individual modules or components.
- Ensures each unit works correctly in isolation.
- Example: Testing a login function separately.

2. Integration Testing
- Combines modules and tests them together.
- Checks data flow and interaction between components.
- Example: Testing login + dashboard together.

3. System Testing
- Tests the complete system as a whole.
- Ensures overall functionality meets requirements.
- Example: Testing the entire Library Management System.

4. Acceptance Testing
- Done by end users or clients.
- Confirms the system meets business needs.
- Example: Client testing payroll system before final approval.

5. Regression Testing
- Re-testing after changes or bug fixes.
- Ensures new updates don't break existing features.
- Example: After fixing a bug in book issue module, re-test other modules.

9. Differentiate between re-engineering and reverse engineering.Why do we need software maintenance.

Differentiate Between Re-engineering and Reverse Engineering (Detailed Version)

Feature	**Reverse Engineering**	**Re-engineering**
Definition	Process of **analyzing existing software** to understand **its structure, components, and functionality**.	Process of **improving or redesigning existing software** to enhance **performance, maintainability, or features**.
Purpose	To **extract knowledge** or design from existing software.	To **upgrade, optimize, or modernize** software.

4. What is feasibility analysis? Explain cost-benefits analysis technique with example.

Feasibility analysis is the process of evaluating whether a proposed software project is practical, possible, and worth doing. It checks technical, economic, operational, legal, and schedule aspects before development starts. The goal is to avoid wasting time, money, and effort on projects that cannot succeed

Benefit Analysis Technique

Explanation .

Cost–Benefit Analysis (CBA) is an economic feasibility technique. It compares the expected costs of the project with the expected benefits. If benefits are greater than costs, the project is considered feasible and profitable.

Key Points:

1. Identify all costs (development, hardware, software, training, maintenance).
2. Identify all benefits (time saving, error reduction, better service, profit increase).
3. Convert both costs and benefits into monetary values.
4. Compare total benefits with total costs.
5. Decide whether to accept or reject the project.

Example:

Suppose a library automation project costs $50,000 (software, hardware, training).

- Expected benefits: saving staff time, reducing errors, faster service → worth $80,000.
- Since Benefits ($80,000) > Costs ($50,000), the project is economically feasible.

5. What do you mean by project planning? Explain COCOMO model.

Project planning is the process of defining objectives, scope, schedule, resources, and risks for a software project. It acts as a roadmap to ensure the project is completed on time, within budget, and meets quality standards.

Explain COCOMO Model

Definition

COCOMO (Constructive Cost Model) is a software cost estimation model. It predicts the effort and cost required to develop software based on the size in KLOC (thousands of lines of code).

Types of COCOMO

- Basic COCOMO: Simple model based on size only.
- Intermediate COCOMO: Includes additional cost drivers.
- Detailed COCOMO: Most detailed, includes phase-specific estimates.

Formula

$$\text{Effort (Person-Months)} = a \times (\text{KLOC})^b$$

Constants Table

Project Type	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embelded	3.6	1.20

Example

Given : Project size = 50 KLOC (Organic type).

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

Step-bte : $1.05 = 1 + 0.05$, so $(50)^{1.05} = (50)^1 \times (50)^{0.05}$

Using $(50)^{0.05} \approx 1.3797$

Effort $\approx 2.4 \times 50 \times 1.3797 = 165.56$ Person-Months.

Result

165.56 Person-Months

10. Write short notes on:

a. E-R Diagram (Entity-Relationship Diagram)

An E-R Diagram is a visual representation of the logical structure of a database. It shows how "entities" (like people or objects) relate to one another.

- **Entities:** Represented by **Rectangles**. These are the "nouns" (e.g., Student, Teacher).
- **Attributes:** Represented by **Ellipses**. These are the properties of an entity (e.g., Student_Name, Roll_No).
- **Relationships:** Represented by **Diamonds**. These show how entities interact (e.g., Student "Enrolls" in Course).
- **Cardinality:** Defines the numerical relationship (One-to-One, One-to-Many, or Many-to-Many).

b. Payback Period

Payback Period is a financial metric used during **Cost-Benefit Analysis** to determine the time it takes for a project to "pay for itself."

- **Definition:** It is the time required to recover the initial investment made in a software project.
- **Decision Rule:** Usually, a project with a **shorter** payback period is preferred because it is less risky.
- **Formula:**

$$\text{Payback Period} = Initial Investment\text{Annual Cash Inflow**Example:** If a project costs **$10,000** and saves the company **$2,500** per year, the payback period is **4 years**.

c. Documentation

Documentation in software engineering is the written text or illustration that accompanies computer software. It is the "memory" of the project.

- **Types:**
  1. **System Documentation:** Explains the inner workings (Source code, DFDs, E-R diagrams) for future developers.
  2. **User Documentation:** Explains how to use the software (User Manuals, Installation Guides) for the end-user.
- **Importance:**
  - It helps in **Software Maintenance**.
  - It acts as a communication tool between the team and the client.
  - It ensures that if a developer leaves, the project can still continue.

d. Agile Model

The Agile model is a modern approach to software development that focuses on **iterative development** and **customer feedback**.

- **Iterative Process:** Instead of building the whole system at once, it is broken into small parts called "Sprints" (usually 2–4 weeks long).
- **Flexibility:** It welcomes changes in requirements even late in the development process.
- **Customer Involvement:** The client sees the progress every few weeks and provides feedback immediately..
- **Key Advantage:** It reduces the risk of project failure because errors are caught early in the small iterations.

b. Describe about the software design model and design strategies.

A software design model is a representation of the software system's structure and behavior, while design strategies are the approaches used to create this model.

Explanation

A software design model serves as a blueprint for implementation. It typically includes different views or diagrams, such as:

- **Architectural design:** Defines the overall structure and organization of the system, including major components and their relationships.
- **Data design:** Focuses on the structure and properties of the data that the system will manage.
- **Interface design:** Describes how the system interacts with users and other systems.
- **Component-level design:** Details the internal logic and structure of individual software components.

Design strategies are the methods and philosophies applied to develop these models. Common strategies include:

- **Top-down design:** Breaking the system down into smaller, manageable modules starting from a high-level view.
- **Bottom-up design:** Starting with the creation of basic components and then integrating them into larger systems.
- **Object-oriented design (OOD):** Structuring the software around objects rather than actions, focusing on data and its behavior.
- **Structured design:** Emphasizes modularity, logical structure, and control flow within the system.

4.a. Explain coupling and cohesion.

Cohesion

Cohesion refers to the degree to which elements inside a module belong together. A highly cohesive module performs a single, well-defined task. High cohesion improves readability, maintainability, and reliability.

Types of Cohesion (from worst to best):

- Coincidental Cohesion → Random grouping of tasks.
- Logical Cohesion → Grouped by logic but not tightly related.
- Temporal Cohesion → Tasks executed at the same time.
- Procedural Cohesion → Tasks executed in sequence.
- Communicational Cohesion → Tasks use same data.
- Functional Cohesion → Module performs one single function (best).

Example:

A "Login Module" that only handles user authentication has high cohesion.

Coupling

Coupling refers to the degree of dependency between modules. Low coupling means modules are independent, while high coupling means modules are tightly connected. Low coupling is desirable because it makes systems easier to maintain and modify.

Types of Coupling (from worst to best):

- Content Coupling → One module modifies another's data directly (worst).
- Common Coupling → Modules share global data.
- Control Coupling → One module controls another's logic.
- Stamp Coupling → Modules share composite data structures.
- Data Coupling → Modules share only necessary data (best).

Example:

A "Payment Module" passing only transaction ID to "Invoice Module" shows low coupling.

b. Define SDLC. Explain prototyping model with its advantages and disadvantages.

Definition of SDLC (Software Development Life Cycle)

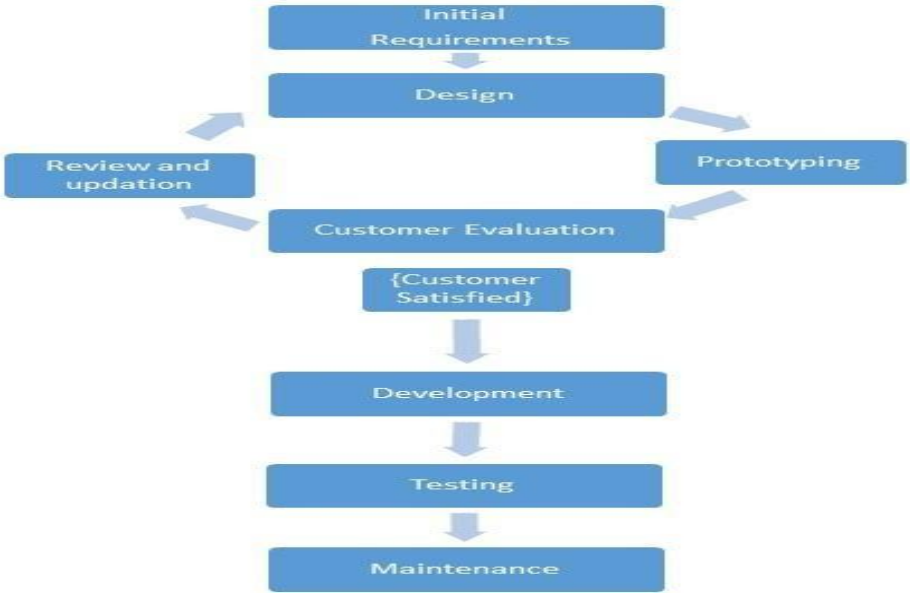
The Software Development Life Cycle (SDLC) is a structured framework that defines the step-by-step process of developing, maintaining, and replacing software systems. It provides a systematic approach to ensure that software is high-quality, cost-effective, and meets user requirements.

Explain prototyping model.

The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested, and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product

Why to use prototyping model

While developing software, there are cases wherein the initial stages, we do not know what the overall requirements of the software are. This happens mostly in cases where the customer is not completely sure what he wants his software to look like.



Advantages of Prototyping Model

1. Users get a clear idea of the system early.
2. Requirements can be refined and clarified through feedback.
3. Reduces risk of requirement misunderstanding.
4. Encourages user involvement in development.
5. Helps in building better user satisfaction.

❌ Disadvantages of Prototyping Model

1. Can lead to scope creep (continuous changes).
2. Prototype may be mistaken as the final product.
3. Frequent changes increase development cost and time.

4. Not suitable for large, complex projects.
5. Requires skilled developers to manage iterations.

2 .a. Explain software project scheduling. Write activities in project management.

Software Project Scheduling

- **Definition:** Software project scheduling is the process of **planning and allocating time** for all tasks and activities in a software project.
- **Purpose:** It ensures that the project is completed **on time**, within **budget**, and meets quality standards.

Steps in Software Project Scheduling:

1. **Identify tasks** – Break the project into smaller activities.
2. **Estimate time and resources** – Determine how long each task will take and what resources are needed.
3. **Set dependencies** – Identify which tasks depend on others.
4. **Allocate time and resources** – Assign tasks to team members and schedule start/end dates.
5. **Monitor progress** – Track task completion and adjust schedule if needed.

Activities in Project Management

1. **Planning** – Define objectives, scope, and resources.
2. **Scheduling** – Allocate tasks and time to each activity.
3. **Budgeting** – Estimate costs for tasks and resources.
4. **Risk Management** – Identify potential risks and plan mitigation.
5. **Monitoring & Controlling** – Track progress and make adjustments.
6. **Reporting** – Communicate progress to stakeholders.
7. **Closure** – Complete the project and review performance

b. Define requirement elicitation. What are the characteristics of good SRS?

Requirement elicitation is the process of gathering requirements from stakeholders to understand what the software should do. It uses techniques like interviews, questionnaires, observations, and prototyping to capture user needs clearly.

Characteristics of a Good SRS (Software Requirement Specification)

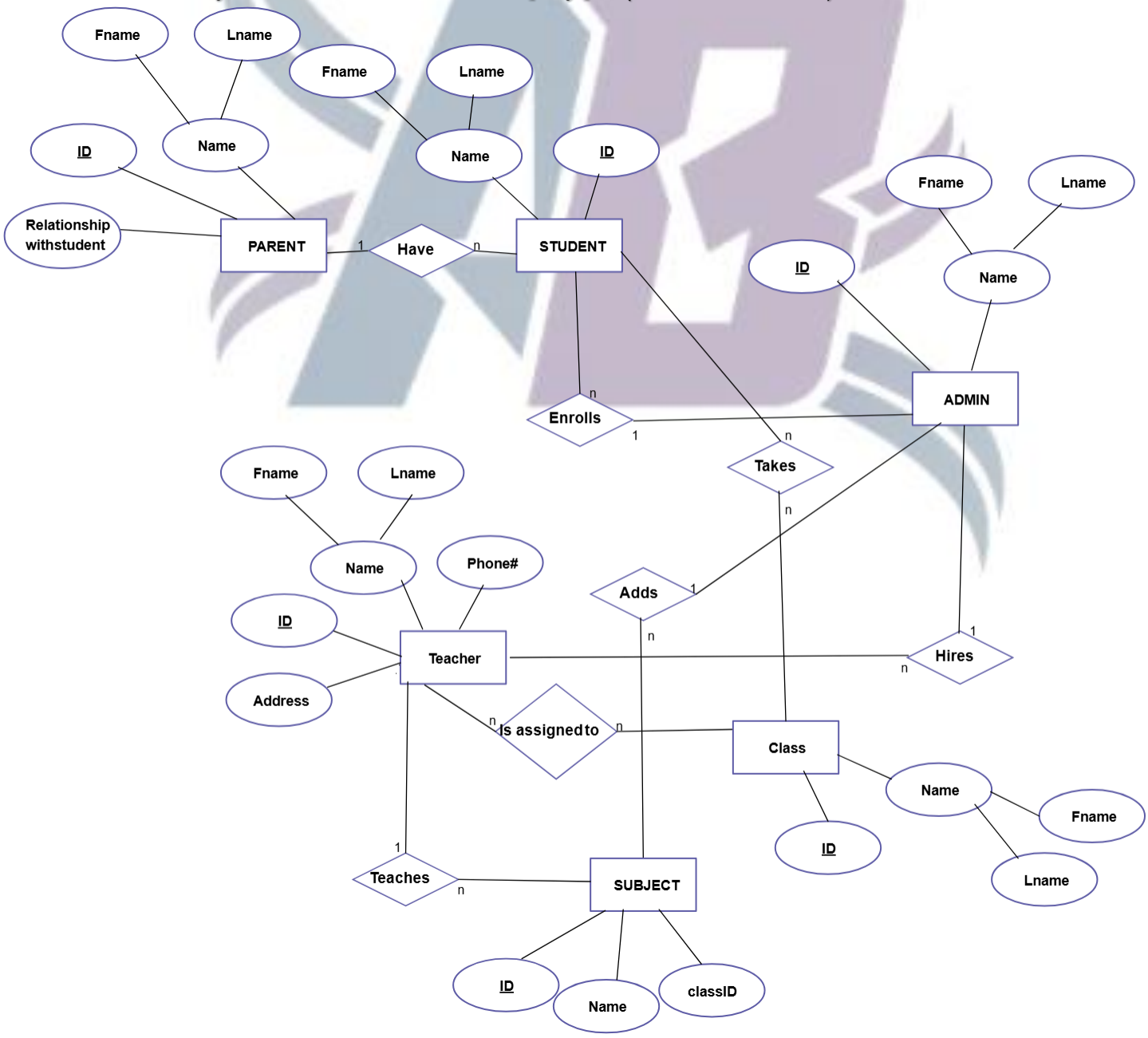
1. Correctness – All requirements must be accurate and valid.
2. Completeness – Covers all functional and non-functional requirements.
3. Consistency – No conflicting requirements.
4. Clarity – Requirements should be easy to understand, no ambiguity.
5. Verifiability – Each requirement must be testable.
6. Modifiability – Easy to update when changes occur.
7. Traceability – Each requirement should be linked to its source.

3. a. Describe ER diagram with example of school management system.

An Entity–Relationship (ER) diagram is a graphical representation of entities, attributes, and relationships in a database system. It helps in designing the logical structure of a database by showing how data is connected. ER diagrams are widely used in system analysis and design because they provide a clear, visual model of data flow and relationships.

📌 Key Components of ER Diagram

1. Entities – Objects in the system (e.g., Student, Teacher, Class).
2. Attributes – Properties of entities (e.g., Student_ID, Name, Age).
3. Relationships – Connections between entities (e.g., Student enrolled in Class).
4. Primary Key – Unique identifier for each entity.
5. Cardinality – Defines relationship type (1:1, 1:M, M:N).



BACK EXAM - 2082 SHRAWAN/BHADRA

DIPLOMA IN COMPUTER ENGINEERING

a. Define program and software. List characteristics and types of software.

Definition of Program.

A program is a finite set of instructions written in a programming language that directs the computer to perform a specific task or solve a particular problem. It is designed to be executed by the computer's processor, where each instruction tells the system what operation to perform

Definition of Software

Software is a comprehensive collection of programs, procedures, and documentation that together enable a computer system to perform tasks and provide solutions to user needs. Unlike a single program, software is broader in scope — it may consist of multiple interconnected programs, configuration files, libraries, and manualCharacteristics of Software

1. Intangible – Software cannot be touched physically.
2. Engineered product – Developed systematically using processes.
3. Does not wear out – Unlike hardware, software doesn't degrade physically, but may need updates.
4. Customizable – Can be modified to meet user needs.
5. Complex – Large software systems are highly complex and require proper design/testing.

🖥️ Types of Software

1. System Software

- Controls hardware and provides platform for application software.
- Example: Operating Systems (Windows, Linux), Utility Programs.

2. Application Software

- Designed to perform specific user tasks.
- Example: MS Word, Photoshop, Library Management System.

3. Programming Software

- Tools used by developers to create programs.
- Example: Compilers, Interpreters, IDEs.

b. What are software metries? Why do we need it?Explain its various categories.

Software metrics are quantitative measures used to assess different aspects of software development and performance. They provide objective data about software quality, productivity, complexity, and reliability.

Why Do We Need Software Metrics?

1. To measure software quality.
2. To track productivity and progress.
3. To identify complexity and risks.
4. To improve decision-making in project management.
5. To ensure maintainability and reliability

Categories of Software Metrics.

Software metrics are quantitative measures used to evaluate software quality, productivity, performance, and project success. They are broadly classified into three main categories: Product Metrics, Process Metrics, and Project Metrics.

1 Product Metrics

Definition:

Product metrics measure the characteristics of the software product itself. They focus on the internal attributes (size, complexity, performance, reliability) and external attributes (quality, usability).

Why Important:

- Helps in assessing software quality.
- Identifies areas of high complexity that may cause errors.
- Ensures the product meets performance and reliability standards.

Example:

In a Banking System, measuring response time of fund transfer ensures performance quality.

2 Process Metrics

Definition:

Process metrics measure the efficiency and effectiveness of the software development process. They focus on how well the team follows procedures, manages resources, and removes defects..

Why Important:

- Helps in process improvement.
- Identifies bottlenecks in development.
- Ensures better quality control and defect prevention.

Example:

Tracking defect removal rate during testing shows how effective the QA process is.

3 Project Metrics

Definition:

Project metrics measure the management aspects of a software project. They focus on cost, schedule, resources, and risk management.

Why Important:

- Helps managers control budget and timeline.
- Improves resource allocation.
- Reduces risk of project failure.

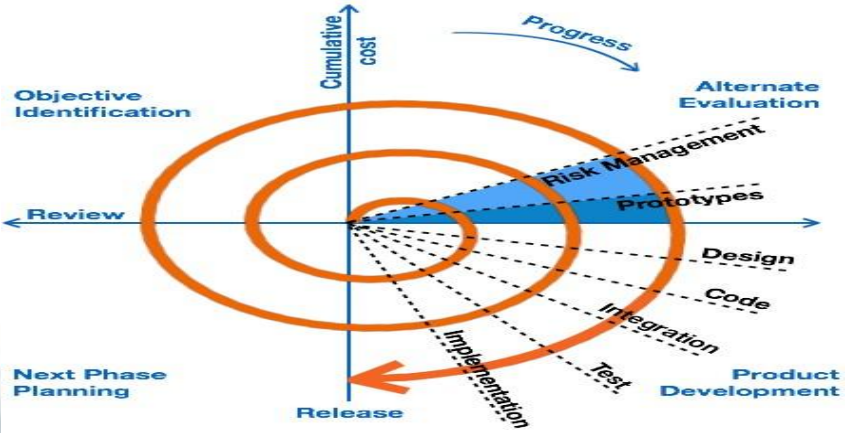
Example:

In a School Management System project, tracking schedule variance ensures modules (Student, Teacher, Course) are delivered on time.

2.a. Describe spiral model with its advantages and disadvantages.

The Spiral Model, introduced by Barry Boehm in 1986, is a risk-driven software development process model. It combines the linear structure of the Waterfall model with the iterative refinement of Prototyping.

Development is visualized as a spiral (loop), where each loop represents a phase of the software process.



Spiral Model

The Spiral Model is a risk-driven and iterative software development model. Development progresses in a spiral shape consisting of multiple loops, where each loop represents a complete development cycle.

The number of loops depends on project size, complexity, and risk.

Each loop includes:

- Planning
- Risk analysis
- Development
- Evaluation

Phases of the Spiral Model

Focus is on managing risk through multiple iterations of the software development process. Each phase of the Spiral Model is divided into four Quadrants.

1. Objectives Definition

- Project goals and requirements are identified
- Functional and non-functional requirements are analyzed
- Possible solutions are explored

2. Risk Analysis and Resolution

- Risks related to cost, schedule, performance, and technology are identified
- Best solution is selected
- Prototypes are built to reduce risks

3. Development and Testing

- Selected features are designed, developed, and tested
- The next working version of the software is created

4. Review and Planning

- Customers evaluate the current version
- Feedback is collected
- Planning for the next iteration begins\

Advantages

- • Strong risk management at every stage.
- • Supports customer feedback and requirement changes.
- • Combines structured approach (Waterfall) with flexibility (Prototyping).
- • Early detection of errors reduces overall cost.
- • Suitable for large, mission-critical projects (banking, defense, aerospace).
- 📌 Disadvantages
- • Complex to manage compared to linear models.
- • Requires expertise in risk analysis.
- • Can be costly and time-consuming for small projects.

6.a. Define software assurance. Describe software maintenance.

Software assurance is the confidence that software functions as intended, is free from vulnerabilities, and meets quality standards throughout its lifecycle. It ensures that software is secure, reliable, and dependable in its operation.

Software Maintenance

Definition:

- Software maintenance is the **process of modifying and updating software after delivery** to correct faults, improve performance, or adapt to new environments.

Types of Software Maintenance:

1. **Corrective Maintenance:** Fixing **errors and bugs** after release.
2. **Adaptive Maintenance:** Modifying software to **work with new hardware or OS**.
3. **Perfective Maintenance: Enhancing performance or adding new features**.
4. **Preventive Maintenance: Preventing future issues** by improving software quality.

Importance:

- Reduces **system failures**
- Extends **software life**
- Ensures **user satisfaction**
- Keeps software **relevant and secure**

b. Write short notes on:

Quality Control

Definition:

Quality control in software engineering is the process of ensuring that the developed software meets defined standards and requirements. It focuses on detecting defects in the product before release.

Key Points:

- Involves reviews, inspections, audits, and testing.
- Ensures conformance to standards (ISO, IEEE).
- Detects errors early → reduces cost of fixing later.
- Improves customer satisfaction and reliability.

Example:

Running test cases to check if a payroll system calculates salaries correctly before deployment.

✍️ **Risk Analysis**

Definition:

Risk analysis is the systematic process of identifying, assessing, and prioritizing risks in software development. Risks can be technical, managerial, or operational.

Steps in Risk Analysis:

1. Identify Risks → e.g., cost overruns, schedule delays, security threats.
2. Analyze Risks → Estimate probability and impact.
3. Prioritize Risks → Rank based on severity.
4. Mitigate Risks → Prepare strategies to reduce or avoid risks.
5. Monitor Risks → Continuously track during project lifecycle.

Example:

In an e-commerce system, risk analysis may highlight risks like payment gateway failure or cyber-attacks.

✍️ **Requirement Engineering**

Definition:

Requirement engineering is the systematic process of gathering, analyzing, documenting, and managing software requirements. It ensures that the final product meets user needs and expectations.

Activities in Requirement Engineering:

1. Requirement Elicitation → Collect requirements via interviews, surveys, prototyping.
2. Requirement Analysis → Resolve conflicts, check feasibility.
3. Requirement Specification → Document requirements in SRS (Software Requirement Specification).
4. Requirement Validation → Verify correctness and completeness.
5. Requirement Management → Handle changes during project lifecycle.

Importance:

- Prevents misunderstandings between users and developers.
- Provides a clear blueprint for design and testing.
- Ensures traceability of requirements.

Example:

In a hospital management system, requirement engineering ensures modules like patient records, billing, and appointments are clearly defined in the SRS.

REGULAR/BACK EXAM- 2081/2082CHAITRA/BAISHAKM:

DIPLOMA IN COMPUTER ENGINEERING

1.a. Define software engineering. Differentiate between software and program.

Software engineering is the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. It applies engineering principles to software to ensure quality, reliability, efficiency, and maintainability.

Difference Between Software and Program

S.N.	Feature	Program	Software
1	**Definition**	A set of instructions for a specific task.	A complete product with code, docs, and data.
2	**Scope**	Small in size and limited in scope.	Large, complex, and covers a whole domain.
3	**Components**	Contains only the **Source Code**.	Contains **Code + Manuals + Design + Test Cases**.
4	**Development**	Usually built by an **Individual**.	Developed by a **Team** of engineers.
5	**Approach**	Ad-hoc (informal) programming.	Systematic Engineering approach (SDLC).
6	**User Interface**	Often has a simple or no UI (CLI).	Has a professional and user-friendly GUI.
7	**Maintenance**	Not usually maintained or updated.	Regularly updated and maintained for years.
8	**Example**	A 'Hello World' code or a sorting script.	Windows OS, MS Office, or Photosho

b. What is data dictionary? Explain E-R diagram with an example.

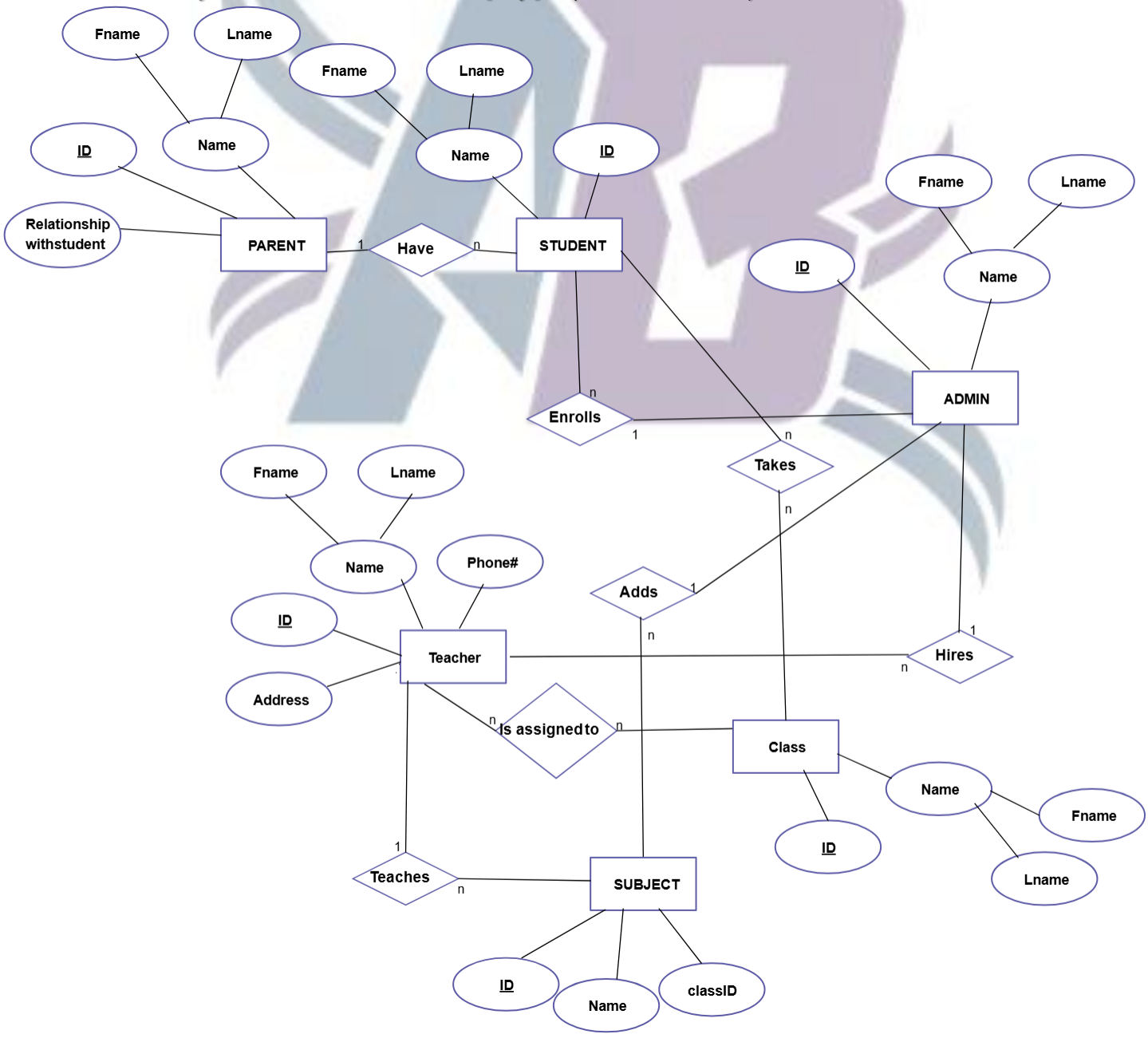
A data dictionary is a central repository of information about data in a database. It defines data elements, their types, sizes, relationships, and constraints. It acts as a reference for developers and users to understand the structure and meaning of data..

Explain E-R diagram with an example

An Entity–Relationship (ER) diagram is a graphical representation of entities, attributes, and relationships in a database system. It helps in designing the logical structure of a database.

Components of E-R Diagram:

- Entity → A real-world object (e.g., Student, Teacher).
- Attribute → Properties of an entity (e.g., StudentID, Name).
- Relationship → Logical association between entities (e.g., Student enrolls in Course).
- Primary Key → Unique identifier for each entity.
- Cardinality → Defines relationship type (1:1, 1:M, M:N).



5.a. Define software reliability. Explain different levels of CMM.

Software reliability is the probability that software will function correctly without failure under specified conditions for a given period of time. It is a key quality attribute that measures the dependability of software. Reliable software produces consistent, accurate results and handles errors gracefully.

1. Level 1 – Initial
- Processes are ad hoc, chaotic, and unpredictable.
- Success depends on individual effort.
- Example: A startup with no formal process.

2. Level 2 – Repeatable
- Basic project management practices are established.
- Past successes can be repeated.
- Example: Scheduling, cost tracking introduced.

3. Level 3 – Defined
- Processes are documented, standardized, and integrated across the organization.
- Example: Organization-wide development methodology.

4. Level 4 – Managed
- Processes are measured and controlled using quantitative metrics.
- Example: Defect density, productivity tracked systematically.

5. Level 5 – Optimizing
- Continuous process improvement through innovation and feedback.
- Example: Organization uses advanced tools and defect prevention strategies.

b. What is software testing? Explain different types of software testing.

Definition

Software testing is the systematic process of executing a program with the intent of finding errors and verifying that it meets requirements. It ensures that the software is functional, reliable, secure, and user-friendly. Testing is both verification (building the product right) and validation (building the right product).

Types of Software Testing

1. Based on Functionality:

- **Unit Testing:** Testing the smallest individual parts of code (functions/classes).
- **Integration Testing:** Testing how different modules work together.
- **System Testing:** Testing the complete, integrated system as a whole.
- **Acceptance Testing:** Final testing done by the user to decide if the software is ready for use.

2. Based on Methodology:

- **White Box Testing:** The tester knows the internal logic/code of the software. It is usually done by developers.
- **Black Box Testing:** The tester does not know the internal code. They only test the inputs and outputs. It is usually done by independent testers
-
-
-

b. Explain about software project estimation techniques.

Software project estimation techniques are methods used to predict effort, cost, time, and resources required for software development. Accurate estimation is critical for project planning, budgeting, and scheduling.

Empirical Estimation (Expert Judgment)

This relies on the experience of experts who have done similar projects in the past.

- **Delphi Technique:** A group of experts provides estimates anonymously. They discuss until they reach a common agreement.

2. Heuristic Techniques (Mathematical Models)

These use formulas to calculate effort and time.

- **COCOMO (Constructive Cost Model):** This is the most famous model. It uses the size of the software (Lines of Code) to estimate effort.
  - *Formula:* Effort = a* (KLOC)^b
- **Function Point (FP) Analysis:** Instead of lines of code, this measures the "functionality" provided to the user (e.g., number of inputs, outputs, and files).

3. Analytical Techniques

This involves breaking the project into smaller tasks (Work Breakdown Structure) and estimating the cost for each small task, then adding them all up.

3.a. Describe the various activities involved in software project management.

Definition.

Software project management is the discipline of planning, organizing, monitoring, and controlling software projects to ensure they are completed on time, within budget, and meet quality standards. It involves a set of activities that guide the project from initiation to closure.

📌 Key Activities

1. Project Planning
- Define objectives, scope, deliverables, and constraints.
- Prepare schedules, budgets, and resource allocation.
- Example: Planning modules of a School Management System.

2. Project Scheduling
- Break down tasks into timelines using tools like Gantt charts, PERT charts.
- Assign responsibilities to team members.
- Example: Assigning login module to one developer, database design to another.

3. Effort and Cost Estimation
- Estimate resources, time, and cost using techniques like LOC, Function Points, COCOMO.
- Example: Predicting 6 months and 10 developers for a hospital system.

4. Risk Management
- Identify, analyze, and mitigate risks (technical, financial, operational).
- Example: Risk of data breach in online banking system.

5. Quality Management
- Ensure software meets standards (ISO, IEEE).

6. Write short notes on:

a. Requirement Elicitation

1. Requirement elicitation ka matlab users se **system ki requirements collect karna** hota hai.
2. Is process me user ki **needs, expectations aur problems** samjhi jaati hain.
3. Ye SDLC ka **initial aur important step** hota hai.
4. Common techniques: **Interview, questionnaire, observation, meetings**.
5. Galat elicitation se **wrong software develop** ho sakta hai.
6. Ye developer aur user ke beech **clear communication** banata hai.
7. Proper elicitation se **rework aur cost kam** hoti hai.
8. Achhi requirements se **high quality software** banta hai.

b. Verification and Validation

1. Verification ensure karta hai ki software **specification ke according bana hai**.
2. Iska question hota hai: *"Are we building the product right?"*
3. Validation check karta hai ki software **user ki needs satisfy karta hai ya nahi**.
4. Iska question hota hai: *"Are we building the right product?"*
5. Verification activities: **reviews, walkthroughs, inspections**.
6. Validation activities: **testing, user acceptance testing**.
7. Dono software quality ke liye **bahut important** hain.
8. Ye errors ko **early stage me pakadne** me help karta hai.

c. RAD Model (Rapid Application Development)

1. RAD model ka focus **fast development** par hota hai.
2. Isme **prototyping** ka use hota hai.
3. User involvement **bahut zyada** hota hai.
4. Development **short time me complete** hota hai.
5. Reusable components ka use kiya jata hai.
6. Small aur medium size projects ke liye **suitable** hai.
7. Skilled developers ki **requirement hoti hai**.
8. Large projects ke liye ye model **effective nahi** hota.

d. Software Reliability

1. Software reliability ka matlab software ka **failure-free kaam karna** hota hai.
2. Ye software ki **dependability** show karta hai.
3. High reliability ka matlab **kam bugs aur crashes**.
4. Reliability time period ke saath **measure ki jaati hai**.
5. Achhi testing se reliability **increase hoti hai**.
6. Reliability user satisfaction ko **directly affect** karti hai.
7. Safety-critical systems me reliability **bahut important** hoti hai.
8. Maintenance se reliability aur **improve** hoti hai.

e. COCOMO Model

1. COCOMO ka full form **Constructive Cost Model** hai.
2. Ye software project ka **cost aur effort estimate** karta hai.
3. Is model ko **Barry Boehm** ne develop kiya.
4. Ye software ke **size (LOC)** par based hota hai.
5. Isse manpower aur development time estimate hota hai.
6. Types: **Basic, Intermediate, Detailed COCOMO**.
7. Planning aur budgeting me **helpful** hota hai.
8. Large projects me ye model **useful** hota hai.

c. What are the key attributes of high quality software? Explain in brief.

High-quality software is not just about code that works; it must meet several non-functional requirements. These are often called **"Quality Attributes"** or **"McCall's Quality Factors."**

1. **Functionality:** The software must provide all the functions it was built for.
2. **Reliability:** The software should perform its intended functions without failure for a specific time. It should be "bug-free" under normal use.
3. **Usability:** It should be easy to learn and operate for the end-user. A good UI/UX is essential.
4. **Efficiency:** The software should use minimum system resources (CPU, Memory, Disk space) and provide fast response times.
5. **Maintainability:** It should be easy to modify, fix errors, or update features without breaking the existing system.
6. **Portability:** The software should be able to run on different environments (e.g., Windows, Linux, Mobile) with little or no change.

5. a. Differentiate between function oriented design and object oriented design.

S.N.	Feature	Function-Oriented Design (FOD)	Object-Oriented Design (OOD)
1	Core Unit	Focuses on Functions (Processes).	Focuses on Objects (Data + Functions).
2	Approach	Follows a Top-Down approach.	Follows a Bottom-Up approach.
3	Data Movement	Data moves freely from function to function.	Data is hidden inside objects (Encapsulation).
4	System State	State is centralized (Global data).	State is distributed among various objects.
5	Scalability	Hard to manage as system size grows.	Highly scalable and easier to manage.
6	Reusability	Lower reusability of code.	High reusability (via Inheritance).
7	Example	Developed using C or Pascal.	Developed using Java, C++, or Python.

b. What do you mean by software design? Explain the software design strategies.

Software design is the process of transforming requirements into a structured plan that defines system architecture, components, interfaces, and data. It acts as a blueprint for implementation, ensuring the software meets user needs effectively.

Software Design Strategies.

Design strategies are the approaches or methodologies used to break down a complex system into manageable parts. There are two primary strategies:

A. Top-Down Design Strategy

This strategy follows the principle of **"Divide and Conquer."** You start with the system as a single high-level entity and gradually break it down into smaller sub-systems.

- **How it works:** 1. The main system is defined first. 2. It is decomposed into major modules. 3. Each major module is further broken down into smaller sub-modules. 4. This continues until the modules are simple enough to be coded as individual functions.
- **Characteristics:**
  - It uses **Stepwise Refinement**.
  - It focuses on the "Big Picture" first.
  - In testing, it uses **Stubs** (dummy modules) to represent lower-level functions that aren't built yet.

B. Bottom-Up Design Strategy

This strategy starts with the most basic, fundamental components and combines them to form larger, more complex systems.

- **How it works:**
  1. Designers start by identifying and building the smallest, lowest-level modules (like basic data processing tools).
  2. These modules are tested and grouped together to create a higher-level module.
  3. This process continues until the entire system is built from these pre-tested pieces.
- **Characteristics:**
  - It focuses on **Reusability** (using existing libraries or tools).
  - It is often used in **Object-Oriented Design**, where you build classes first and then assemble the system.
  - In testing, it uses **Drivers** to simulate the higher-level modules that will eventually call these small components.

5.a. Why do we need software testing? Explain the different levels of software testing.

Software Testing: Why and How?

Why do we need Software Testing?

Software testing is essential because "to err is human." Even the best developers make mistakes. We need testing to:

- **Identify Defects:** To find bugs and errors made during the coding phase.
- **Ensure Security:** To make sure the software is safe from hackers and data leaks.
- **Quality Assurance:** To ensure the product meets the user's requirements and is reliable.
- **Cost-Effectiveness:** Finding a bug during testing is much cheaper than fixing it after the product has been delivered to the customer.
- **Performance:** To ensure the software doesn't crash under heavy load.

Levels of Software Testing

Testing is performed in a specific sequence. These are known as the levels of testing:

1. **Unit Testing:** The smallest level of testing. It focuses on individual components or "units" of code (like a single function or class). Usually done by **developers**.
2. **Integration Testing:** After unit testing, different modules are combined and tested together to see if they interact correctly.
3. **System Testing:** The entire software is tested as a whole. The goal is to verify that the complete system meets the functional and non-functional requirements.
4. **Acceptance Testing:** This is the final level. It is performed by the **end-user** or client to decide whether the software is ready for "Go-Live" or deployment.

b. What is software development life cycle? Explain different types of software maintenance.

SDLC is a structured process or framework used by the software industry to design, develop, and test high-quality software. It provides a sequence of steps that a project team must follow to build a software product from scratch to finish.

Different levels of software testing

Unit Testing

- Tests the smallest testable parts (individual functions, methods, classes, or modules) in isolation.
- Usually done by developers using frameworks (JUnit, pytest, etc.).
- Fast, cheap, and catches bugs very early.

2. **Integration Testing**
   - Tests the interaction between integrated units/modules/components (e.g., how two modules communicate, data flow between them).
   - Approaches: Big Bang, Top-Down, Bottom-Up, Sandwich.
     Reveals problems that unit tests miss (e.g., mismatched data formats).
3. **System Testing**
   - Usually performed by independent QA/testers.
   - Focus: End-to-end functionality, non-functional aspects (performance, security, usability, reliability), compliance with overall requirements.
   - Black-box testing (no code knowledge needed).
4. **Acceptance Testing** (User Acceptance Testing – UAT)
   - Final level, validates the system meets business requirements and is ready for deployment.
   - Performed by end-users, clients, or stakeholders (sometimes Alpha/Beta testing).
   - Types:
     - **Alpha testing** — in-house by QA/developers.
     - **Beta testing** — real users in real environment.

- Includes reviews, inspections, testing.
- Example: Testing payroll system for accuracy.

6. Team Management
- Organize team roles, responsibilities, and communication.
- Motivate and resolve conflicts.
- Example: Assigning project manager, developers, testers.

7. Monitoring and Control
- Track progress against plan.
- Use metrics (defect rate, productivity) to control deviations.
- Example: Weekly status meetings to check progress.

8. Configuration Management
- Control versions of software, documents, and code.
- Example: Using GitHub for version control.

9. Project Closure
- Deliver final product, prepare documentation, and conduct post-mortem analysis.
- Example: Delivering ERP system and analyzing lessons learned.

b. Why do we need software metrics? Explain.

Definition

Software metrics are quantitative measures used to assess different aspects of software development and performance. They provide objective data about software quality, productivity, complexity, and reliability.

📌 Need for Software Metrics

1. Measure Software Quality
- Metrics like defect density, reliability, maintainability help ensure high quality.

2. Track Productivity and Progress
- LOC, Function Points, and effort metrics measure developer productivity.

3. Identify Complexity and Risks
- Cyclomatic complexity highlights modules that are error-prone.

4. Improve Decision-Making
- Managers use metrics for scheduling, budgeting, and resource allocation.

5. Ensure Maintainability and Reliability
- Metrics like MTBF (Mean Time Between Failures) ensure dependable software.

6. Customer Confidence
- Objective measures reassure clients about project progress and quality.

f. Types of Software

1. **System Software** hardware ko control karta hai (Operating System).
2. **Application Software** user ke kaam ke liye hota hai (MS Word).
3. **Utility Software** system maintenance ke liye hota hai (Antivirus).
4. **Embedded Software** devices me use hota hai (Washing machine).
5. System software ke bina computer **work nahi karta**.
6. Application software user-oriented hota hai.
7. Embedded software specific task ke liye hota hai.
8. Har software ka **different purpose** hota hai.

BACK EXAM - 2081 KARTIK/MANGSIN

DIPLOMA IN COMPUTER ENGINEERING

1a. Define software and software engineering. Differentiate between program and software.

Software Software is a set of instructions, programs, data, and documents that tell a computer what to do. It is the non-physical part of a computer system (opposite of hardware). Examples: apps like WhatsApp, games, operating systems like Windows, or even the software inside your microwave.

Software Engineering Software Engineering is the proper, organized way of building software using engineering principles. It includes planning, designing, coding, testing, delivering, and maintaining software so that it is good quality, works correctly, is delivered on time, and doesn't cost too much. It's not just writing code — it's the full science and discipline of making reliable software.

rogram vs Software (अंतर)

Parameter	Program	Software
Definition	Set of instructions for specific task	Programs + Documentation + Data + Procedures
Size	Small (individual task)	Large (complete system)
Development	Single programmer	Team of engineers
Documentation	Optional/Minimal	Mandatory & Detailed
Examples	Calculator program, Sort function	MS Office, Banking System
Testing	Basic unit testing	Comprehensive testing (unit, integration, system)
Maintenance	Rarely required	Regular updates needed
Cost	Low	High
User Base	Limited/Developer	Multiple end-use

3. **Applicability:** Metrics should be applicable in the initial phases of the development of the software.
4. **Repeatable:** When measured repeatedly, the metric values should be the same and consistent.
5. **Economical:** The computation of metrics should be economical.
6. **Language Independent:** Metrics should not depend on any programming language.

Example: Cyclomatic Complexity V(G)

This measures the number of independent paths through a program's source code.

V(G) = P + 1

(Where P is the number of decision points like if, while, or for loops).

- **V(G) = 1-10:** Simple code, easy to test.
- **V(G) > 50:** Extremely complex, almost impossible to test.

5. a) Define software quality. Explain Capability Maturity Model (CMM).

Software Quality: Software quality means how good, correct and useful the software is for its users. It shows that the software meets all user needs, works without many errors, is easy to use, reliable, safe and can be maintained easily over time.

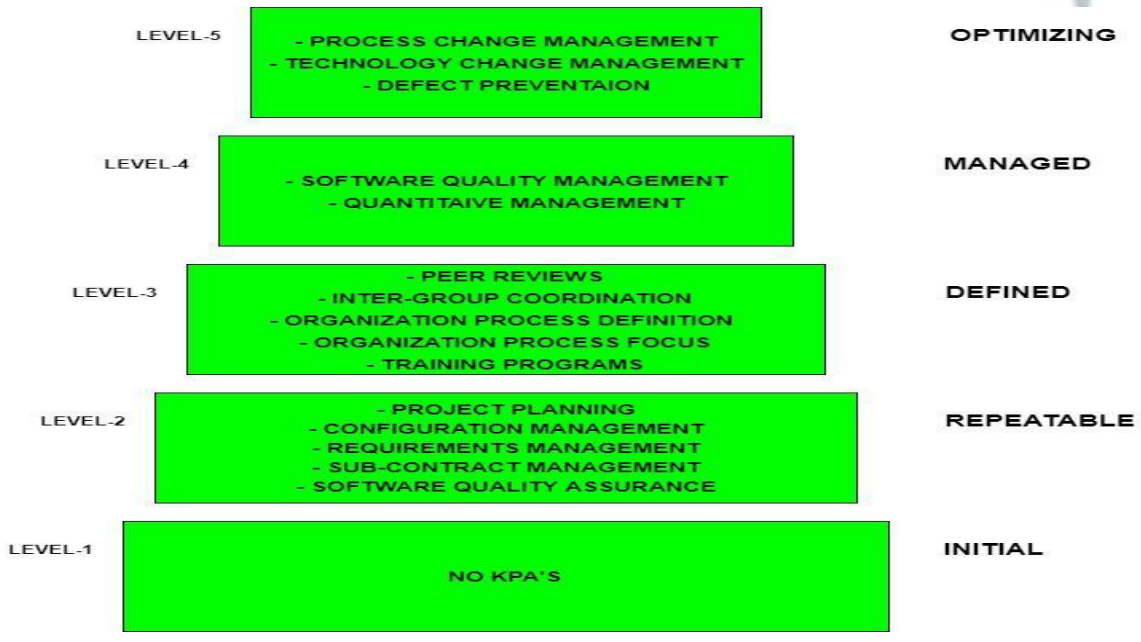
Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987. It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products. It also provides guidelines to enhance further the maturity of the process used to develop those software products

Importance of Capability Maturity Model

- **Optimization of Resources:** CMM helps businesses make the best use of all of their resources, including money, labor, and time. Organizations can improve the effectiveness of resource allocation by recognizing and getting rid of unproductive practices.
- **Comparing and Evaluating:** A formal framework for benchmarking and self-evaluation is offered by CMM. Businesses can assess their maturity levels, pinpoint their advantages and disadvantages, and compare their performance to industry best practices.
- **Management of Quality**: CMM emphasizes quality management heavily. The framework helps businesses apply best practices for quality assurance and control, which raises the quality of their goods and services.
- **Enhancement of Process**: CMM gives businesses a methodical approach to evaluate and enhance their operations. It provides a road map for gradually improving processes, which raises productivity and usefulness.
- **Increased Output:** CMM seeks to boost productivity by simplifying and optimizing processes. Organizations can increase output and efficiency without compromising quality as they go through the CMM levels.

Levels of Capability Maturity Model (CMM)

There are 5 levels of Capability Maturity Models. We will discuss each one of them in detail.



3. a) What are activities in project management? Explain COCOMO model

Activities in Project Management

Software Project Management (SPM) involves the following key activities to ensure a project is successful:

- **Project Planning:** Defining the scope, goals, and roadmap.
- **Estimation:** Determining the cost, time, and effort required.
- **Scheduling:** Creating a timeline and assigning tasks (using Gantt or PERT charts).
- **Risk Management:** Identifying and mitigating potential failures.
- **Monitoring and Control:** Tracking progress and taking corrective action if the project deviates from the plan.

Explain COCOMO model.

COCOMO Model is a procedural cost estimate model for **Software Projects** and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.

The key parameters that define the quality of any **Software Product**, which are also an outcome of COCOMO, are primarily effort and schedule.

ypes of Projects in COCOMO Model

In the COCOMO model, software projects are categorized into three types based on their complexity, size, and the development environment. These types are:

1. Organic

A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

2. Semi-detached

A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and embedded.

The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.

3. Embedded

A software project requiring the highest level of complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Structure of COCOMO Model

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the **Software Engineering Process**.

In detailed COCOMO, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

b) Define requirement elicitation technique. Explain requirement elicitation techniques in detail.

Requirement Elicitation is the process of gathering, discovering, and uncovering the requirements for a software system from stakeholders (customers, users, and managers). It is the first and most critical step in Requirement Engineering.

Requirement Elicitation Techniques in Detail

1. Interviews

This is the most common technique where the analyst talks directly to the stakeholders.

- **Structured Interviews:** The analyst has a fixed list of questions. It is good for getting specific data.
- **Unstructured Interviews:** A free-flowing conversation that allows for exploring new ideas and uncovering unexpected requirements.

2. Surveys and Questionnaires

When there are a large number of users located in different places, surveys are used.

- **Pros:** Can gather a lot of data quickly.
- **Cons:** You cannot ask follow-up questions if a user's answer is unclear.

3. Brainstorming

A group session where stakeholders and developers meet to "generate" ideas without criticism.

- It is excellent for identifying new features and resolving conflicting views between different departments.

4. Observation (Ethnography)

Sometimes users cannot explain what they do. In this technique, the analyst watches the user perform their actual job in the real workplace.

- **Why it works:** It helps find "hidden" requirements or steps that the user considers so "obvious" that they forget to mention them in an interview.

5. Prototyping

Building a simple, semi-functional version (mock-up) of the software.

- The user interacts with the prototype and says, "I like this, but I don't like that." This is the best way to clarify requirements when the user is unsure of what they want.

6. Facilitated Application Specification Techniques (FAST)

A team-oriented approach where a joint team of users and developers works together in a specialized workshop.

- The goal is to identify the problem, propose elements of the solution, and negotiate different approaches in a collaborative environment.

7. **Use Case Analysis:**A technique used to describe how a user (Actor) interacts with the system to achieve a specific goal. It focuses on the "behavior" of the system.

4. **a) What is software prototyping? Differentiate between function oriented design and object oriented design.**
   Software prototyping is the process of creating incomplete but working models (prototypes) of a software system during the early stages of development. These prototypes simulate key aspects of the final product — such as user interface, functionality, navigation, or core behavior — without implementing the complete system.
5. **Differentiate between function oriented design and object oriented design.**

S.N.	Feature	Function-Oriented Design (FOD)	Object-Oriented Design (OOD)
1	Central Focus	Focuses on Functions (Processes).	Focuses on Objects (Data + Action).
2	Approach	Top-Down approach.	Bottom-Up approach.
3	Basic Building Block	Sub-programs or modules.	Classes and Objects.
4	Data Management	Data is global or passed around.	Data is hidden (Encapsulation).
5	Real-world Mapping	Maps to a flow of events.	Maps to real-world entities.
6	Reusability	Low (hard to reuse functions).	High (via Inheritance).
7	Complexity	Becomes messy in large systems.	Manages large-scale complexity well.

b) What is risk? How can you manage it? Explain software matrices with example.
In software engineering, **risk** is an **uncertain event or condition** that, if it occurs, has a **negative impact** on one or more project objectives (cost, schedule, quality, scope, customer satisfaction).
Risk = Probability of occurrence × Impact (severity if it occurs)
How to Manage Risk (RPMM Plan):

1. **Risk Identification:** List all possible things that could go wrong (Technical, Financial, or Schedule risks).
2. **Risk Analysis:** Estimate the **Probability** (how likely) and **Impact** (how bad) of each risk.
3. **Risk Mitigation:** Take steps to prevent the risk (e.g., train a backup developer).
4. **Risk Monitoring:** Keep checking the project status to see if a risk is becoming a reality.

Software Metrics
A metric is a measurement of the level at which any impute belongs to a system product or process.
Software metrics are a quantifiable or countable assessment of the attributes of a software product. There are 4 functions related to software metrics:

1. **Planning**
2. **Organizing**
3. **Controlling**
4. **Improving**

Characteristics of software Metrics

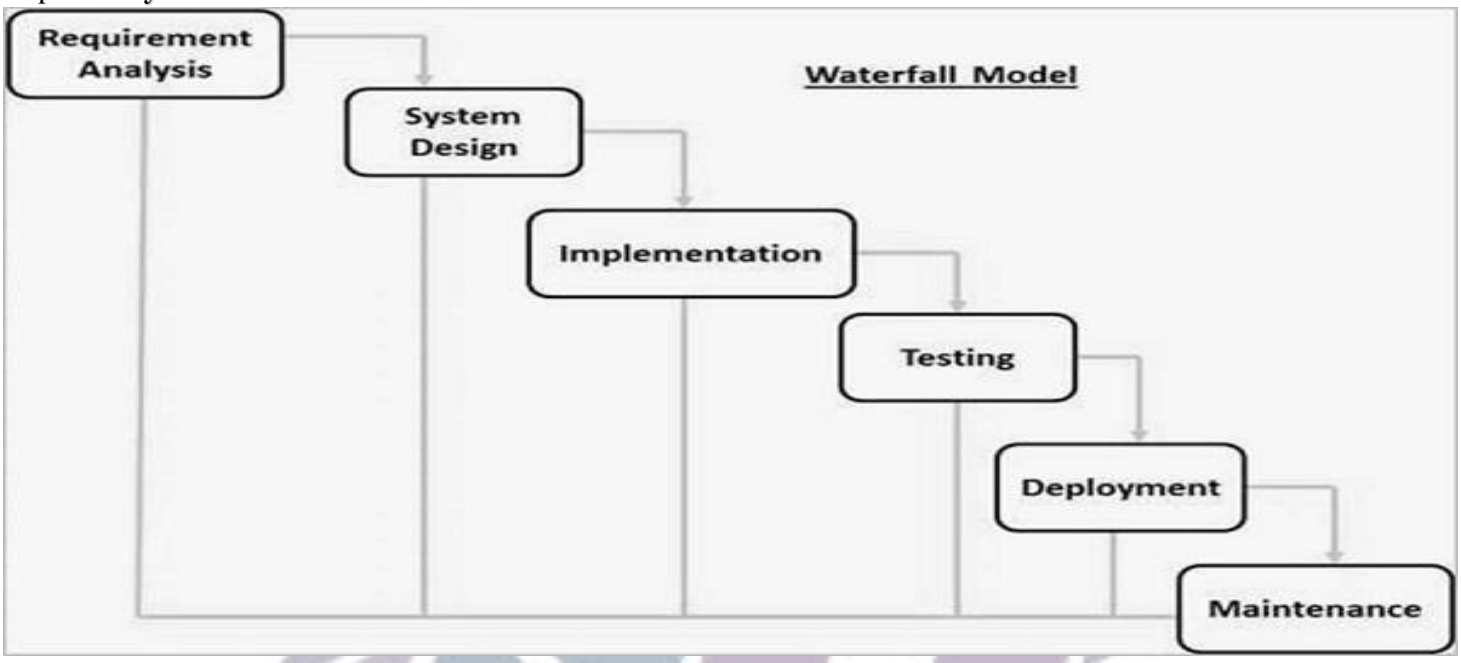
1. **Quantitative:** Metrics must possess a quantitative nature. It means metrics can be expressed in numerical values.
2. **Understandable:** Metric computation should be easily understood, and the method of computing metrics should be clearly defined.

b. Define SDLC. Explain waterfall model with its advantages and disadvantages.

SDLC is a structured process used by the software industry to design, develop, and test high-quality software. It consists of a series of steps that ensure the software meets user requirements and is delivered efficiently.

waterfall model

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.



Phases of Waterfall Model.

1. **Requirements Analysis & Specification** — Gather and document all requirements (SRS — Software Requirements Specification).
2. **System & Software Design** — Create high-level (architecture) and low-level (detailed) design.
3. **Implementation / Coding** — Write actual code based on design.
4. **Testing** — Integrate and test the system (unit, integration, system testing).
5. **Deployment** — Install and release the software to users.
6. **Maintenance** — Fix issues and make changes after deployment

Advantages of Waterfall Model

- Simple and easy to understand.
- Well-structured with clear documentation.
- Works well for small projects with stable requirements.
- Each phase has specific deliverables → easy to manage.

👎 Disadvantages of Waterfall Model

- Rigid – difficult to go back to previous phases.
- Poor model for projects with changing requirements.
- Late testing → errors found at the end, costly to fix.
- Not suitable for complex or long-term projects.

b) What is software testing? Explain different types of software testing.

Software Testing: Software testing is the process of checking and finding errors in a software program to make sure it works correctly as per requirements. It helps to verify that the software does what it is supposed to do and does not have bugs before giving it to users.

Common types of software testing include:

- **Black-box testing**: A method where the internal structure, design, and implementation of the item being tested are not known to the tester. It focuses solely on the functionality and user interface.
- **White-box testing**: A method where the internal structure, design, and implementation of the item being tested are known to the tester. It focuses on testing the internal logic and code of the software.
- **Unit testing**: Testing individual components or sections of code to ensure they work correctly in isolation.
- **Integration testing**: Testing the combination of components to ensure they work together as a group.
- **System testing**: Testing the complete and integrated software product to evaluate the system's compliance with its specified requirements.
- **Acceptance testing**: Formal testing conducted to determine whether a system satisfies its acceptance criteria and to enable the customer to determine whether to accept the system.

6. a) Define software maintenance? Explain types of software maintenance.

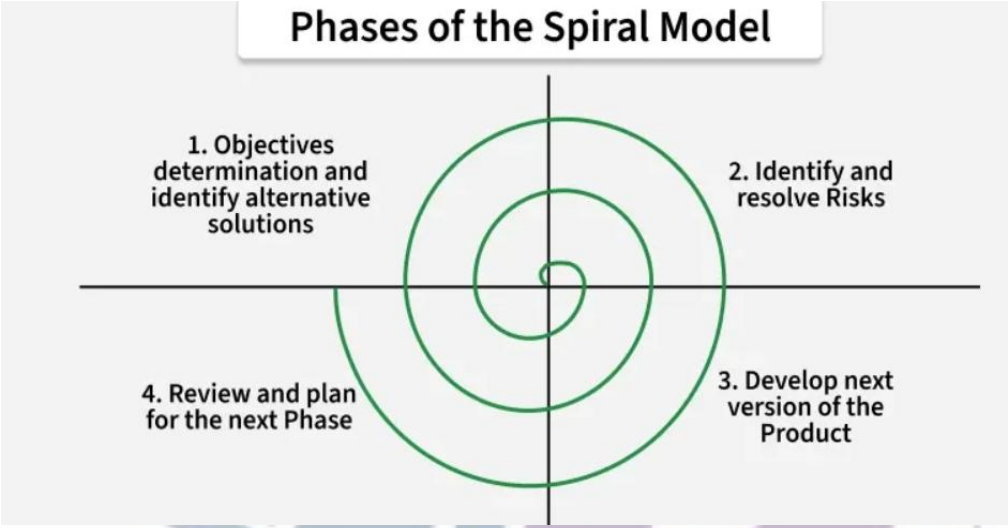
Software Maintenance: Software maintenance is the process of changing and updating a software after it has been delivered to the customer and is in use. It includes fixing errors, improving performance, adding new features or adapting the software to new environment so that it keeps working well for long time.

Types of Software Maintenance: Software maintenance is commonly classified into **four main types** (as per IEEE/ISO standards and widely accepted in software engineering):

1. **Corrective Maintenance** This involves fixing defects, bugs, or errors discovered after delivery. It is **reactive** and aims to restore the software to correct functioning when it does not perform as intended.
2. **Adaptive Maintenance** This type modifies the software to adapt to **changes in the environment**, such as new hardware, operating systems, databases, regulations, or business rules. It ensures the software remains compatible and functional in evolving contexts.
3. **Perfective Maintenance** This focuses on **improving** performance, maintainability, usability, efficiency, or other attributes without changing the core functionality. It includes enhancements like adding new features requested by users or optimizing code for better speed/resource usage.
4. **Preventive Maintenance** This is proactive work done to **prevent future problems**. It includes restructuring code, refactoring, updating documentation, or applying best practices to reduce the risk of faults, improve maintainability, and make future changes easier.

b. Explain spiral model with diagram.

The Spiral Model is one of the most important SDLC models, combining the structured approach of the Waterfall Model with the flexibility of the Iterative Model. It is mainly used for risk handling in large and complex projects.The Spiral Model was proposed by Barry Boehm.



Spiral Model

The Spiral Model is a risk-driven and iterative software development model. Development progresses in a spiral shape consisting of multiple loops, where each loop represents a complete development cycle.

The number of loops depends on project size, complexity, and risk.

Each loop includes:

- Planning
- Risk analysis
- Development
- Evaluation

Phases of the Spiral Model

Focus is on managing risk through multiple iterations of the software development process. Each phase of the Spiral Model is divided into four Quadrants.

1. Objectives Definition

- Project goals and requirements are identified
- Functional and non-functional requirements are analyzed
- Possible solutions are explored

2. Risk Analysis and Resolution

- Risks related to cost, schedule, performance, and technology are identified
- Best solution is selected
- Prototypes are built to reduce risks

3. Development and Testing

- Selected features are designed, developed, and tested
- The next working version of the software is created

4. Review and Planning

- Customers evaluate the current version
- Feedback is collected
- Planning for the next iteration begins

REGULAR/BACK EXAM – 2080/2081, CHAITRA/BAISHAKH

DIPLOMA IN COMPUTER ENGINEERING

1.a. Describe SDLC. Explain in brief about different phases of SDLC.

The **Software Development Life Cycle (SDLC)** is a structured process used by the software industry to design, develop, and test high-quality software. It provides a systematic flow of phases that help a team produce software that meets customer expectations within reach of time and cost estimates.

Phases of SDLC

1. **Requirement Collection and Analysis:** This is the most crucial phase. Senior members of the team gather detailed requirements from stakeholders and customers. The outcome is a **Software Requirement Specification (SRS)** document that serves as a guide for the entire project.
2. **Feasibility Study:** Once requirements are clear, the team analyzes the project's viability. This includes checking technical feasibility (can we build it?), operational feasibility (will it work in the environment?), and financial feasibility (is it within budget?).
3. **Design:** Based on the SRS, the system design is prepared. This includes the **High-Level Design (HLD)** (architecture and modules) and the **Low-Level Design (LLD)** (database tables, interface details, and dependency charts).
4. **Implementation / Coding:** This is where the actual development starts. Developers write code using the chosen programming languages (like Java, Python, or C++) based on the design documents.
5. **Testing:** Once the code is developed, it is tested against the requirements. This phase includes unit testing, integration testing, system testing, and **User Acceptance Testing (UAT)** to ensure the software is bug-free.
6. **Deployment:** After the software is tested and approved, it is released into the production environment for the end-users to use.
7. **Maintenance:** After the software is "live," it requires ongoing support. This involves fixing bugs that appear in real-world use or updating the software to work with new hardware or operating systems

b. Explain Rapid Application Development model (RAD) with its advantages.

Rapid Application Development (RAD) is an **iterative and incremental** software development methodology introduced by James Martin in the 1990s. It focuses on **quick delivery** of functional software through rapid prototyping, strong user involvement, and reusable components, rather than rigid, lengthy planning. RAD prioritizes speed and adaptability over comprehensive upfront documentation.

Key Phases of RAD Model (typically 4 main phases):

1. **Requirements Planning** High-level requirements are quickly gathered through joint workshops with users and stakeholders. Scope and objectives are defined rapidly.
2. **User Design / Prototyping** The most time-intensive phase. Developers create iterative prototypes (working models) of the system. Users review, provide feedback, and suggest changes. Prototypes are refined repeatedly until they meet user needs. Tools like visual builders, low-code platforms, or 4GLs speed up this step.
3. **Construction / Rapid Construction** Using the approved prototype as a blueprint, the final system is built quickly. Coding, testing, and integration happen in short cycles with heavy reuse of components and automation tools.
4. **Cutover / Deployment & Transition** The completed system is deployed, users are trained, data is migrated, and the system goes live. Final adjustments are made based on real-world use.

Advantages of RAD Model:

- **Faster development and delivery** — Projects can be completed in weeks or months instead of years due to prototyping and reuse.
- **High user involvement and satisfaction** — Continuous feedback ensures the final product closely matches user expectations and reduces rework.
- **Greater flexibility and adaptability** — Easy to incorporate changes during development without major cost/time overruns.
- **Reduced development cost** (in suitable projects) — Less time spent on manual coding, more use of tools, prototypes, and reusable modules.
- **Early visibility of the product** — Users see working versions early, allowing better risk identification and mitigation.
- **Improved quality through iteration** — Continuous refinement and testing lead to fewer defects at release.

b) Write short notes on on

i) Quality Assurance

Quality assurance is the planned and systematic process to ensure that software meets the required standards and user expectations. It focuses on preventing defects rather than just finding them. QA involves activities like reviews, audits and testing to check that proper methods are followed during development. The aim is to build confidence in the product and improve reliability. By applying QA, organizations can reduce errors, save cost and deliver better software to customers.

ii) Generic View of Software Re-engineering

Software re-engineering means improving existing software without changing its basic functionality. The generic view includes activities like inventory analysis, restructuring of code, reverse engineering, forward engineering and data re-engineering. It is used when old software becomes difficult to maintain but is still important for the organization. Re-engineering helps in reducing maintenance cost, improving quality and extending the life of the system. It is a cost-effective way compared to developing new software from scratch.

iii) RAD (Rapid Application Development)

Rapid Application Development is a software development model that emphasizes quick development and delivery of software with user involvement. It uses prototyping and iterative development to produce results faster. RAD divides the project into small modules which are developed in parallel and integrated later. Users give continuous feedback which helps in refining requirements. The main advantage is speed and flexibility, but it may not be suitable for very large or complex projects.

2. a. Define software with its characteristics. Explain types of software.

Definition of Software: Software is a set of instructions, programs, data and documents that tell the computer how to perform tasks. It is the invisible part of the computer (opposite of hardware which we can touch).

Characteristics of Software

- Software is **intangible** (cannot be touched or seen physically).
- It does **not wear out** or get damaged with use (unlike hardware).
- It is **developed and engineered**, not manufactured like hardware.
- It can be easily **modified, updated or improved**.
- Once created, it can be **copied** many times at very low or zero cost.
- Even a small mistake can make the whole software fail.
- It is **reusable** — same code or module can be used in different programs.

Types of Software

1. System Software

👉 System software acts as a bridge between hardware and user applications. It manages the computer's basic functions and ensures smooth operation.

Main Categories of System Software

- **Operating System** (OS)
- Controls hardware and provides a platform for applications.
- Examples: Windows, Linux, macOS, Android.
- Functions: File management, memory management, process scheduling, security.
- **Language Processors**
- Convert human-written code into machine language.
- Types:
- Compiler (translates whole program at once).
- Interpreter (translates line by line).
- Assembler (converts assembly language to machine code).
- **Device Drivers**
- Specialized programs that allow OS to communicate with hardware.
- Example: Printer driver, graphics card driver.
- **Utility Software**
- Helps in system maintenance and optimization.
- Examples: Antivirus, Disk Cleanup, Backup tools.

📌 Exam Tip: System software = "foundation" of computer operations.

2. Application Software

👉 Application software is designed to help users perform specific tasks. It runs on top of system software.

Types of Application Software

- **General Purpose Software**
- Used for common tasks.
- Examples: MS Word (document writing), Excel (spreadsheets), PowerPoint (presentations), Web browsers.
- **Specialized/Customized Software**
- Tailored for specific industries or organizations.
- Examples: Hospital Management System, Banking Software, ERP systems.
- **Utility Applications** (User-level)
- Small programs for specific tasks.
- Example: Media players, photo editors, calculators.

2. a. Define risk. Explain the process of risk management.

a. Risk and Risk Management

Definition of Risk (2 lines): A risk is a potential problem that might happen in the future and cause a loss or hinder the success of a software project. It is characterized by **Uncertainty** (it may or may not happen) and **Loss** (it results in unwanted consequences if it occurs).

The Risk Management Process

This is a systematic process to identify and control risks. It generally follows these steps:

1. **Risk Identification:**
   - The team brainstorms to find all potential threats (e.g., technology changes, team members leaving, or unclear requirements).
   - A "Risk Register" or list is created to keep track of these.
2. **Risk Analysis & Assessment:**
   - Each risk is evaluated based on two factors: **Probability** (How likely is it?) and **Impact** (How bad will it be?).
   - **Risk Exposure (RE)** is calculated as: RE = P* C (where P is Probability and C is Cost/Impact).
3. **Risk Planning (Mitigation):**
   - Developing strategies to handle the risk. Common strategies include:
     - **Avoidance:** Changing the plan to eliminate the risk.
     - **Mitigation:** Taking steps to reduce the impact (e.g., training staff to avoid technical errors).
     - **Transfer:** Shifting the risk to a third party (like buying insurance).
4. **Risk Monitoring:**
   - Continuously checking the project to see if old risks have disappeared or if new ones have emerged.

b. Explain software project estimation and write briefly on COCOMO model

Software Project Estimation Software project estimation means predicting (guessing) how much effort, time, cost, and resources will be needed to complete the project successfully. It is done early to make plans, set budget, and decide schedule

COCOMO Model

COCOMO Model is a procedural cost estimate model for **Software Projects** and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.

The key parameters that define the quality of any **Software Product**, which are also an outcome of COCOMO, are primarily effort and schedule.

Types of Projects in COCOMO Model

In the COCOMO model, software projects are categorized into three types based on their complexity, size, and the development environment. These types are:

1. Organic

A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

2. Semi-detached:A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and embedded.The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and

b. Explain software metrics with its example.

Software Metrics Software metrics are **measurable quantities** used to measure different aspects of software like size, complexity, quality, productivity, and performance. They help developers, managers, and testers understand, control, and improve the software development process and product.

Purpose of Software Metrics

- Measure progress and quality.
- Find problems early (e.g., too complex code).
- Compare projects or teams.
- Predict effort, cost, and defects.
- Improve processes.

Types of Software Metrics (with examples)

1. **Size Metrics**
   - Measure how big the software is.
   - **Lines of Code (LOC)** → Count total lines written. Example: A program with 5000 LOC is bigger than one with 1000 LOC.
   - **Function Points (FP)** → Measures functionality based on inputs, outputs, inquiries, files, interfaces. Example: A simple login system may have 20 FP.
2. **Complexity Metrics**
   - Measure how difficult the code is to understand or maintain.
   - **Cyclomatic Complexity (McCabe's)** → Number of independent paths in code = Edges – Nodes + 2. Example: If a function has cyclomatic complexity 15, it is very complex and risky (should be <10 ideally).
3. **Quality Metrics**
   - Measure how good the software is.
   - **Defect Density** = Total defects / Size (e.g., per 1000 LOC). Example: 5 defects per 1000 LOC means good quality.
   - **Maintainability Index** → A score (0–100) based on complexity, LOC, comments, etc. Higher is better.
4. **Productivity Metrics**
   - Measure how fast team is working.
   - **Effort per Function Point** = Person-hours / FP. Example: If team takes 10 hours per FP, productivity is low.
5. **Process Metrics**
   - Measure development process.
   - **Defect Removal Efficiency (DRE)** = (Defects found before release / Total defects) × 100. Example: 90% DRE means most bugs were caught before delivery.

b. What do you mean by requirement elicitation? Explain the methods of information gathering in brief.

Requirement Elicitation is the process of "extracting" or discovering the requirements of a system from users, customers, and other stakeholders. It is more than just "gathering"; it involves active discussion to find hidden needs.

Methods of Information Gathering (Brief)

In an exam, you can list these 5 common methods:

1. **Interviews:**
   - The most direct method. The analyst talks to the users face-to-face.
   - **Structured:** Pre-planned questions.
   - **Unstructured:** Open discussion to explore new ideas.
2. **Questionnaires (Surveys):**
   - Used when there are many users (e.g., thousands of people) or when they are far away.
   - It is a low-cost way to get data, but the answers might not be very detailed.
3. **Brainstorming:**
   - A group of stakeholders sits together to generate new and creative ideas.
   - It helps in resolving conflicts between different departments.
4. **Observation (Social Analysis):**
   - The analyst watches how the users do their current job in their real environment.
   - This helps find "hidden" requirements that users might forget to mention in an interview.
5. **Prototyping:**
   - A "dummy" or sample version of the software is built and shown to the user.
   - Users find it easier to give feedback when they have something physical to look at.

6. a. What id DFD? Explain with example in brief.

It is a simple graphical way to show how data moves through a system. It shows where data comes from, where it goes, how it is processed, and where it is stored. DFD is used in the design phase to understand the flow of information clearly

The 4 Essential Components

To score well, you must describe these symbols:

1. **External Entities (Squares/Rectangles):** These are the sources or destinations of data outside the system (e.g., a Customer or a Manager).
2. **Processes (Circles/Bubbles):** These represent the actions happening to the data (e.g., "Calculate Tax" or "Verify User").
3. **Data Stores (Open Rectangles/Parallel Lines):** These represent data at rest, like a database table or a physical file.
4. **Data Flow (Arrows):** These show the movement of data between the other three components.

Levels of DFD

A DFD is drawn in layers to avoid clutter:

- **Level 0 (Context Diagram):** This is the highest level. It shows the entire system as a single process. It defines the scope of the project and its interaction with outside entities.
- **Level 1 (Factored DFD):** The single process from Level 0 is broken down into major functional subsystems.
- **Level 2 (Detailed DFD):** This level zooms into a specific process from Level 1 to show its inner workings.

b. Explain briefly about software design model and design strategies.

Software Design Model A software design model is like a blueprint or plan of the complete software system. It shows how the system will be built, what its parts (modules) are, how they connect, and how data moves between them. Design is done in two levels:

- **Architectural Design** (high-level) – overall structure, major modules, database, user interface.
- **Detailed Design** (low-level) – inside each module: algorithms, data structures, functions.

Common Software Design Models

1. **Modular Design** – Break system into small independent modules.
2. **Layered Architecture** – Divide into layers (e.g., Presentation Layer → Business Logic Layer → Data Layer).
3. **Client-Server Model** – Client asks for data, Server gives response (used in web applications).
4. **MVC Model** – Model (data), View (user interface), Controller (handles logic).
5. **Object-Oriented Design** – Uses classes, objects, inheritance, and encapsulation.

Design Strategies These are approaches used to create a good design:

1. **Top-Down Design**
    - Start with the big picture of the whole system.
    - Then break it into smaller and smaller parts.
    - Example: First "Online Shopping System" → then "User Module", "Product Module", "Payment Module".
2. **Bottom-Up Design**
    - First build small ready-made parts or components.
    - Then join them to make the full system.
    - Useful when reusing existing code or libraries.
3. **Modular Design (Divide and Conquer)**
    - Divide system into small modules.
    - Each module does one main job (high cohesion).
    - Modules have minimum connection with each other (low coupling).
4. **Structured Design**
    - Use Data Flow Diagrams to create Structure Charts.
    - Focus on functions and how data passes between them.
5. **Object-Oriented Design**
    - Model real-world things as objects and classes.
    - Use concepts like abstraction, encapsulation, inheritance, and polymorphism.

6. a. Differentiate between function oriented design and object oriented design.

S.No	Feature	Function Oriented Design (FOD)	Object Oriented Design (OOD)
1	Primary Focus	Focuses on Functions or processes (what the system does).	Focuses on Objects or entities (who is involved in the system).
2	Approach	Follows a Top-Down design approach.	Follows a Bottom-Up design approach.
3	Data & Logic	Data and functions are separate entities.	Data and functions are tied together in a single unit (Object).
4	Abstraction	Focuses on Functional Abstraction (hiding how a task is done).	Focuses on Data Abstraction (hiding how data is stored).
5	Security	Less secure because data is often global and accessible to all functions.	Highly secure due to Encapsulation (data is hidden inside objects).
6	Reusability	Low reusability; functions are usually specific to one project.	High reusability; objects/classes can be used in many projects.
7	Maintenance	Difficult to maintain; changing one function can affect the whole system.	Easier to maintain; changes in one object don't usually break others.
8	Mapping	It is an abstract way of looking at a problem; doesn't feel "real-world."	It maps very naturally to real-world things (e.g., Car, Bank Account).
9	Key Concepts	Uses Structure Charts and Decision Tables.	Uses Inheritance, Polymorphism, and Encapsulation.
10	Complexity	Becomes very messy and "spaghetti-like" for large systems.	Manages large, complex systems very efficiently.

require more experience better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.

3. Embedded

A software project requiring the highest level of complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models

4.a. Describe software project scheduling and time line charts

oftware Project Scheduling Software project scheduling means deciding **when** each task will start and finish, **who** will do it, and in **what order** tasks will be done so that the whole project gets completed on time. It is part of project planning. Good scheduling helps avoid delays, use resources properly, and finish within budget and deadline.

Main purposes of scheduling

- Find the total time needed for the project.
- Identify critical tasks (if delayed, whole project gets delayed).
- Balance workload among team members.
- Track progress during development.

Key elements in scheduling

- List all tasks/activities (from work breakdown structure).
- Find dependencies (which task must finish before another starts).
- Estimate duration of each task.
- Assign resources (people, tools).
- Create a schedule.

Timeline Charts Timeline charts show project tasks on a time scale (usually horizontal axis = time in days/weeks/months). They help everyone see the plan visually.

Most common timeline charts

1. **Gantt Chart**
    - Horizontal bars show tasks.
    - Length of bar = duration of task.
    - Start and end dates are marked.
    - Shows dependencies with arrows (sometimes).
    - Easy to understand and update. Example:
    - Task 1: Requirements (Week 1–2)
    - Task 2: Design (Week 3–5)
    - Bars drawn side by side or overlapping if parallel.
2. **PERT Chart (Program Evaluation and Review Technique)**
    - Uses circles (nodes) for tasks/events and arrows for dependencies.
    - Shows earliest start, latest finish, slack time.
    - Helps find **critical path** (longest path = minimum project time).
    - More detailed for complex projects.

7. a. What is software reliability? Explain different types of software reliability models.

Software Reliability Software reliability is the **probability** that software will work **without failure** for a **specified period of time** under given conditions (environment and usage). It measures how dependable the software is — it does not wear out like hardware, but fails due to bugs, design errors, or wrong logic. High reliability means fewer crashes or wrong outputs in real use.

Types of Software Reliability Models Software reliability models help **predict**, **estimate**, or **measure** reliability using math and data (like failure times or bug counts). They are mainly divided into two big groups:

1. **Prediction Models** These models predict reliability **before** or **early in** development using factors like project size, team experience, tools, and past data. They do not need actual failure data from testing.
    - Example: **Rome Laboratory Model (TR-92-52 or its updated versions)** — Uses questionnaires about project factors to predict reliability for systems like avionics.
    - Used in early planning to decide if the software will meet reliability goals.
2. **Estimation / Reliability Growth Models (SRGMs)** These models use **actual failure data** collected during testing to estimate current reliability and predict future improvement as bugs are fixed. Reliability "grows" as more bugs are removed. Common examples:
    - **Jelinski-Moranda Model** (1972) — Assumes each bug fixed increases reliability by a constant amount. It models time between failures.
    - **Musa-Okumoto Model** (Logarithmic)** — Assumes failure rate decreases slowly (logarithmic way) as testing continues. Good when later fixes have smaller effect.
    - **Goel-Okumoto Model** — Exponential model where failure detection rate reduces over time.
    - **Non-Homogeneous Poisson Process (NHPP) Models** — Many models like Yamada models fall here; they assume failures follow Poisson distribution and rate changes with time.

Other classifications:

- **Error Counting Models** → Count bugs and assume fixes improve reliability (e.g., Jelinski-Moranda).
- **Input Domain Models** → Divide inputs into classes and test to estimate failure probability (e.g., Nelson model).
- **Markov Structure Models** → Use states (working/failing) to model transitions

S.No	Feature	Program	Software
7	Maintenance	Hard to maintain as it's often written without standards.	Systematically designed to be easily maintained and updated.
8	Testing	Tested casually by the programmer (Unit Testing).	Undergoes rigorous, multi-level testing (Beta, System, Regression).
9	Reliability	Not guaranteed; usually meant for personal use.	Built to be highly reliable and robust for commercial use.
10	Examples	A simple 'C' program to add two numbers.	Microsoft Word, Photoshop, or an Operating System.

b. Explain iterative enhancement model in brief.

Iterative Enhancement Model

Definition : The Iterative Enhancement Model is a software development approach where the system is developed through repeated cycles (iterations). Instead of building the whole thing at once, you start with a small part and keep adding features in each step.

How it Works

The project is broken down into various versions (increments).

1. **Initial Step:** The most important requirements are identified and a "core" version of the software is built.
2. **Iteration:** In each cycle, the software is designed, implemented, and tested.
3. **Enhancement:** With every cycle, new features are added ("enhanced") until the full system is ready.

Steps in each Iteration:

- **Analysis:** Understanding the requirements for the current version.
- **Design:** Creating the technical blueprint for new features.
- **Implementation:** Writing the code.
- **Evaluation:** Getting feedback to plan the next version.

Advantages:

- **Early Delivery:** The customer gets a working version of the software very early.
- **Easy Changes:** It is easier to change requirements because you only work on one small part at a time.
- **Risk Management:** Problems are found in the early iterations before the whole project is finished.

9. a. Define white box and black box testing.

White Box Testing White box testing (also called structural testing, glass box testing, or clear box testing) is a testing method where the tester has full knowledge of the **internal structure**, **code**, **logic**, and **implementation** of the software. The tester looks "inside the box" to design test cases based on code paths, branches, loops, conditions, and data structures.

Main features

- Tester needs programming knowledge and access to source code.
- Focuses on **how** the software works (internal logic).
- Finds bugs like dead code, unused variables, wrong logic, boundary errors.
- Done mainly by developers.
- Techniques: Statement coverage, Branch coverage, Path coverage, Condition coverage.

Example Testing a function that calculates discount:

- Check if all if-else branches are tested.
- Check if loop runs 0, 1, and many times.
- Check boundary values like discount = 0%, 100%.

Black Box Testing Black box testing (also called functional testing or behavior testing) is a method where the tester **does not know** the internal code or structure of the software. The tester treats the software as a "black box" and only checks **inputs** and **outputs** to see if it meets the requirements and works correctly from the user's view.

Main features

- No need for code access or programming knowledge.
- Focuses on **what** the software does (functionality).
- Done mainly by testers, QA team, or end-users.
- Finds missing functions, wrong outputs, usability issues.
- Techniques: Equivalence partitioning, Boundary value analysis, Decision table, Use case testing.

Example For a login page:

- Input: Correct username + password → Expected: Login success.
- Input: Wrong password → Expected: Error message.
- No need to see how password is checked in code

b. Explain different types of software maintenance. How can we determine the software maintenance cost?

Types of Software Maintenance After software is delivered and used, it needs changes to keep working well. Software maintenance is divided into four main types (as per IEEE/ISO standards):

1. **Corrective Maintenance** Fixing bugs or errors found after delivery.
    - Reactive (done when problem occurs).
    - Example: User reports app crashes on login → fix the bug.
2. **Adaptive Maintenance** Changing software to work in a new or changed environment.
    - Due to new OS, hardware, laws, or database.
    - Example: Update app to work on new Android version or new tax rules.
3. **Perfective Maintenance** Improving or adding new features to make software better (even if it already works).
    - Improves performance, usability, efficiency.
    - Example: Add dark mode, faster search, or new report feature.
4. **Preventive Maintenance** Making changes to prevent future problems (proactive).

- Refactoring code, updating documentation, improving structure.
- Example: Rewrite old messy code so future changes are easier.

Determining Software Maintenance Cost.

Maintenance cost depends on several factors:

- Complexity of software – More complex → higher cost.
- Size of software – Larger systems need more updates.
- Quality of initial design/code – Poor design → more fixes.
- Frequency of changes – Frequent environment/requirement changes increase cost.
- Manpower & tools used – Skilled staff and advanced tools may increase cost but reduce long-term issues.
- Metrics used:
- % of effort spent on maintenance vs development.
- Number of defects reported per month.
- Average cost per change request.

Formula

$$M = p + K^{(c - d)}$$

Write short notes on.

a) Selection Criteria of Lifecycle Model

1. **Consider project size – small, medium, or large.**
2. **Check requirement stability – stable or changing.**
3. **Evaluate risk level – high-risk projects need iterative models.**
4. **Look at time constraints – short deadlines → RAD/Prototyping.**
5. **Assess budget/cost – simpler models for limited budget.**
6. **Consider team expertise – experienced → Agile, novice → Waterfall.**
7. **Match project complexity – complex → Spiral/Iterative.**
8. **Ensure client involvement level – high → RAD/Agile, low → Waterfall.**

---

b) Verification vs Validation

1. **Verification ensures software meets design specifications.**
2. **Validation ensures software meets user requirements.**
3. **Verification asks: "Are we building the product right?"**
4. **Validation asks: "Are we building the right product?"**
5. **Verification is usually done by developers.**
6. **Validation is usually done by users/testers.**
7. **Verification examples: code review, walkthrough.**
8. **Validation examples: user acceptance testing (UAT).**

---

c) Regression Testing

1. **Done to check new changes do not break existing features.**
2. **Ensures software stability after updates.**
3. **Usually performed after bug fixes or feature additions.**
4. **Can be manual or automated.**
5. **Example: Adding a new report → old reports must still work.**
6. **Helps in early detection of unexpected errors.**
7. **Maintains overall software reliability.**

8. Part of maintenance testing activities.

---

d) Data Dictionary

1. Central repository of all data elements used in software.
2. Contains data name, type, size, and description.
3. Stores valid values or constraints for data.
4. Helps developers, testers, and maintainers understand data.
5. Reduces ambiguity and errors in software.
6. Supports software maintenance and updates.
7. Example: Student_ID: integer, 6 digits, unique identifier.
8. Improves standardization and documentation of the system.

---

e) Quality Assurance (QA)

1. QA is a planned process to ensure software quality.
2. Focuses on preventing defects rather than finding them.
3. Includes reviews, audits, and testing.
4. Ensures software reliability, efficiency, and correctness.
5. Helps in meeting user requirements and satisfaction.
6. Can be process-oriented or product-oriented.
7. Example activities: code review, automated testing, process audits.
8. Goal: improve software quality and maintain standards.

Regular/Back Exam – 2080 Magh/Phagun - Diploma in Computer Engineering

1.a. Differentiate between program and software.

S.No	Feature	Program	Software
1	Definition	A set of instructions written in a specific language to solve a problem.	A collection of programs, documentation, and configuration data.
2	Size	Typically small and limited in scope.	Usually large and highly complex.
3	Development	Created by an individual (Programmer).	Developed by a team of Engineers/Developers.
4	Documentation	Documentation is rarely provided or very brief.	Comprehensive documentation (User Manual, Design Doc) is mandatory.
5	User Interface	Often lacks a professional UI; may be command-line based.	Must have a user-friendly and tested Interface (UI/UX).
6	Life Cycle	Ends once the task is performed or logic is finished.	Follows a strict life cycle (SDLC) from planning to retirement.

b. Explain the types of testing carried out during software testing.

Software Testing Software testing is the process of checking if the software works correctly, meets requirements, and has no major defects. Different types of testing are done at different stages to find bugs early and ensure quality.

Main Types of Software Testing

1. **Unit Testing**
   - Tests individual components or modules (functions, classes) in isolation.
   - Done by developers using tools like JUnit.
   - Finds bugs in small code parts early.
   - Example: Test if a "calculate total" function gives correct sum.
2. **Integration Testing**
   - Tests how different modules work together after combining them.
   - Checks interfaces, data flow between units.
   - Types: Big Bang, Top-Down, Bottom-Up, Sandwich.
   - Example: Test if login module connects properly to database module.
3. **System Testing**
   - Tests the complete, integrated system as a whole in a real-like environment.
   - Verifies if it meets all functional and non-functional requirements.
   - Includes functional and non-functional checks.
   - Done after integration.
4. **Acceptance Testing**
   - Final testing to decide if software is ready for delivery.
   - Done by end-users or clients.
   - Types: Alpha (internal), Beta (external users), UAT (User Acceptance Testing).
   - Ensures software meets business needs.
   - Example: Client checks if banking app handles real transactions correctly.

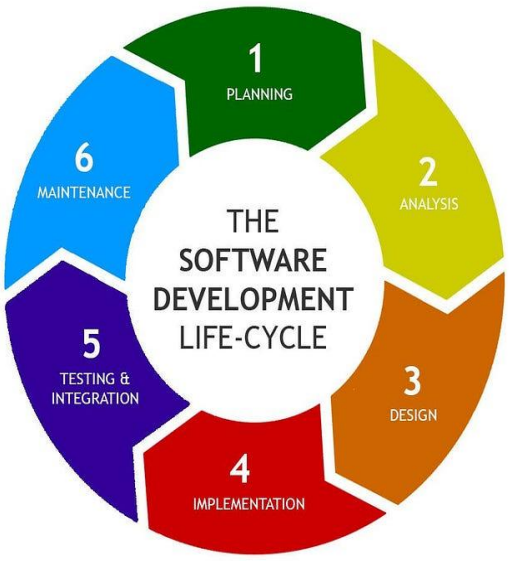
Other Important Types (Mostly Non-Functional or Special)

5. **Functional Testing**
   - Checks if software functions work as per requirements (what it does).
   - Includes unit, integration, system, acceptance.
   - Black-box mostly (no code knowledge needed).
6. **Non-Functional Testing**
   - Checks quality attributes (how well it works).
   - Examples:
     - **Performance Testing** — Speed, load handling (e.g., Load, Stress, Scalability).
     - **Security Testing** — Finds vulnerabilities (e.g., SQL injection, hacking).
     - **Usability Testing** — Easy to use, user-friendly interface.
     - **Compatibility Testing** — Works on different browsers, OS, devices.
     - **Reliability / Recovery Testing** — Works after crash or power failure.
7. **Regression Testing**
   - Re-tests old features after changes (new code, bug fixes) to ensure nothing breaks.
   - Done manually or automated (very common with tools like Selenium).
8. **Smoke / Sanity Testing**
   - Quick check if major functions work (after build or fix).
   - Smoke: Basic build test. Sanity: Focused on specific changes.

2.a. Explain briefly about software development life cycle phases.

Software Development Life Cycle (SDLC) Phases

Definition :SDLC is a structured process used by the software industry to design, develop, and test high-quality software. It ensures that the software meets user requirements within budget and time.



Phases of SDLC:

1. **Requirement Analysis:** This is the most important stage. Developers and analysts gather detailed requirements from the customer to understand what needs to be built.
2. **Feasibility Study:** The team checks if the project is possible. They look at technical (tools), financial (budget), and time feasibility.
3. **Design:** Based on the requirements, a technical blueprint is created. This includes **High-Level Design (HLD)** (architecture) and **Low-Level Design (LLD)** (database, modules).
4. **Coding (Implementation):** The actual development happens here. Programmers write code in the chosen language (Java, Python, etc.) based on the design documents.
5. **Testing:** Once the code is ready, it is checked for bugs and errors. Testing ensures the software works exactly as the customer expected.
6. **Deployment:** After testing, the software is released to the customer or installed on their servers.
7. **Maintenance:** After the software is live, it needs support. This includes fixing new bugs or updating the software for new hardware/OS.

b. What is SRS? Explain characteristics of a good SRS.

Software Requirement Specification (SRS)

Definition : SRS is a formal document that describes what the software is expected to do. it acts as a legal agreement between the customer and the developers.

Characteristics of a Good SRS

To score well, list these 7–8 points clearly:

1. **Correctness:** The SRS should accurately reflect what the customer wants. Every requirement must be true and necessary.
2. **Unambiguous:** Every sentence should have only **one** meaning. There should be no confusion between the developer and the user.

- Checks interfaces, data flow, and communication between parts.
- Approaches: Big Bang, Top-Down, Bottom-Up.
- Finds issues like wrong data passing or module mismatches.
- Example: Test if the "login module" correctly connects to the "database module" and fetches user data.

3. **System Testing**
   - Tests the complete, fully integrated software system as a whole in an environment similar to real use.
   - Verifies both functional (what it does) and non-functional (how well it does) requirements.
   - Done after integration by independent testers.
   - Includes performance, security, compatibility checks.
   - Example: Test the entire banking app to see if it handles login, transfer money, generate statements correctly under load.
4. **Acceptance Testing**
   - Final testing to decide if the software is ready to be delivered and used by the customer.
   - Done by end-users, clients, or their representatives.
   - Types: Alpha (internal team), Beta (real users), UAT (User Acceptance Testing).
   - Confirms the software meets business needs and is acceptable.
   - Example: Client tests an e-commerce website to make sure checkout, payment, and order tracking work as expected in real scenarios.

b. Differentiate between quality assurance and quality control.

S.No	Feature	Quality Assurance (QA)	Quality Control (QC)
1	Primary Goal	To prevent defects from occurring in the first place.	To detect and fix defects in the finished product.
2	Focus	Focuses on the Process (how it's made).	Focuses on the Product (what is made).
3	Nature	It is Proactive (acting before a bug exists).	It is Reactive (acting after a bug is found).
4	Objective	To improve the development and testing processes.	To identify bugs before the software is released.
5	Orientation	It is Process-Oriented.	It is Product-Oriented.
6	Technique	Uses "Managerial" tools (Audits, Training, Standards).	Uses "Testing" tools (Unit testing, System testing).
7	Timeline	Carried out throughout the entire SDLC.	Carried out during the specific Testing phase.
8	Involvement	Involves the entire team (Managers, Devs, Testers).	Usually involves a specific Testing/QC team.
9	Example	Creating a standard "Coding Guideline" for developers.	Running a test script to check if the Login button works.
10	Statistical Tool	Uses SPC (Statistical Process Control).	Uses SQC (Statistical Quality Control).

4.a. What are software metrics? Why do we need software metrics? Explain.

Software metrics are measurable quantities or numbers that help us evaluate different aspects of a software product, the development process, or the project itself. They give us objective data about size, complexity, quality, productivity, effort, and performance of software.

Why do we need Software Metrics? Software metrics are needed because software development is complex and invisible. We cannot "see" quality or progress like in building a bridge. Metrics help us:

1. **Measure and Quantify** Turn subjective things (e.g., "this code is messy") into numbers so we can compare and track.
2. **Improve Quality** Find problem areas early (e.g., high complexity code is likely to have more bugs).
3. **Control and Manage Projects** Predict effort, time, and cost more accurately. Decide when to stop testing (e.g., when defect rate becomes very low).
4. **Increase Productivity** See which teams or methods work better (e.g., compare productivity before and after using new tools).
5. **Make Better Decisions** Management can use metrics to decide resource allocation, risk levels, or whether to release the product.
6. **Reduce Risks and Costs** Early detection of issues saves money (fixing a bug in design is 10–100 times cheaper than after release).
7. **Support Process Improvement** Organizations use metrics to reach higher maturity levels (e.g., CMMI Level 3–5).

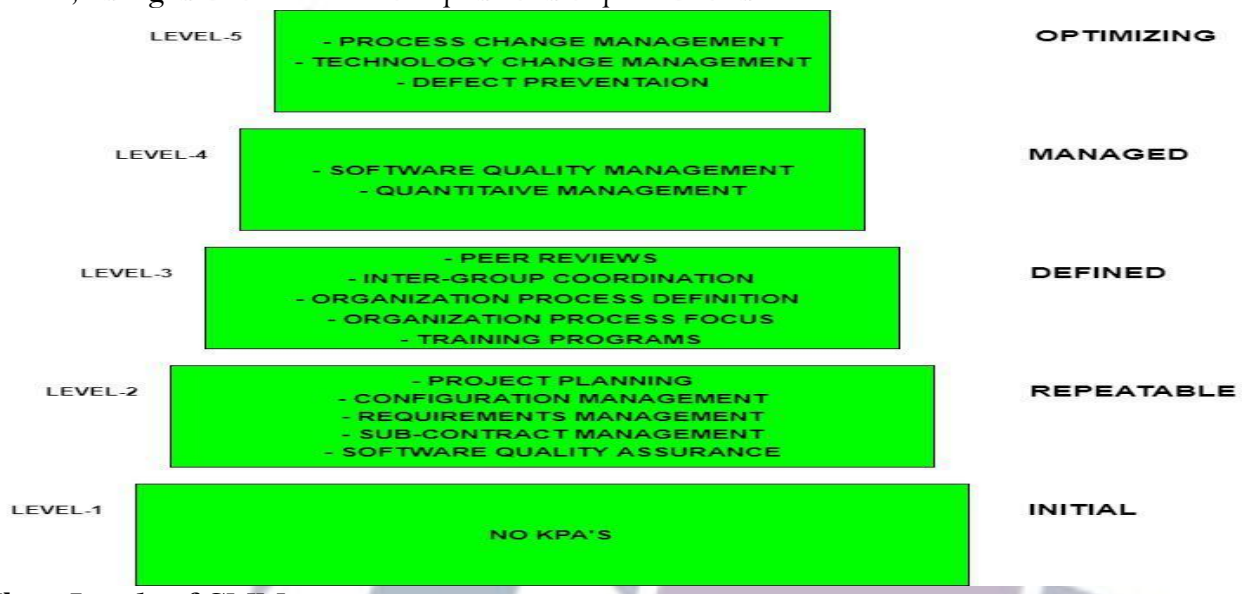
b. Differentiate between function oriented design and object oriented design.

S.No	Feature	Function Oriented Design (FOD)	Object Oriented Design (OOD)
1	Primary Unit	The basic building block is the Function or process.	The basic building block is the Object (Data + Behavior).
2	Design Approach	Follows a Top-Down strategy (Stepwise Refinement).	Follows a Bottom-Up strategy (Entity First).
3	Data Handling	Data is separate from functions and often global.	Data and functions are bundled together (Encapsulation).
4	Abstraction	Focuses on Functional Abstraction.	Focuses on Data Abstraction.
5	State Management	State is often represented in a centralized shared memory.	State is distributed among individual objects.
6	Security	Less secure because data is accessible to all functions.	Highly secure; internal data is hidden from outside world.
7	Reusability	Generally low; functions are specific to one system.	High reusability through Inheritance and Classes.
8	Maintenance	Difficult; a change in data structure affects all functions.	Easier; changes in an object don't usually break others.

S.No	Feature	Function Oriented Design (FOD)	Object Oriented Design (OOD)
9	Real-world Mapping	Maps to mathematical processes or computations.	Maps naturally to real-world entities (Student, Car).
10	Modeling Tools	Uses Data Flow Diagrams (DFD) and Structure Charts.	Uses UML Diagrams (Class, Sequence, Use Case).

5.a. Explain Capability Maturity Model (CMM) and its different levels.

Definition: The **Capability Maturity Model (CMM)** is a framework used to analyze the "maturity" of an organization's software development process. It helps companies improve their processes by moving from a chaotic, unorganized state to a disciplined and optimized one.



The 5 Levels of CMM

You should describe the levels in order, as they represent a path to improvement:

1. **Level 1: Initial (Chaotic):**
   - The process is unorganized and "ad-hoc."
   - Success depends on individual heroes rather than a set process.
   - Projects often go over budget and miss deadlines.
2. **Level 2: Repeatable:**
   - Basic project management processes are established.
   - Success can be repeated because the company tracks costs, schedules, and functionality.
   - Key Focus: Project planning and configuration management.
3. **Level 3: Defined:**
   - The process is documented, standardized, and integrated into the whole organization.
   - All projects use an approved, tailored version of the organization's standard process.
   - Key Focus: Training and technical standards.
4. **Level 4: Managed (Quantitative):**
   - The organization sets quantitative (numerical) goals for both process and product quality.
   - Performance is measured using detailed metrics.
   - Key Focus: Software quality management.
5. **Level 5: Optimizing:**
   - The focus is on **continuous process improvement**.
   - The organization uses feedback from the process and from piloting innovative ideas to prevent defects.
   - Key Focus: Innovation and constant refinement.

b. Explain briefly the process of software maintenance.

Software Maintenance Software maintenance is the process of modifying and updating software **after delivery** to the customer to keep it useful, correct, and efficient over its lifetime. It is the longest and most expensive phase (often 60–80% of total life-cycle cost).

Main Steps in the Software Maintenance Process

1. **Problem/Change Identification**
   - Receive requests from users, customers, or internal team (bug reports, new needs, environment changes).
   - Classify the request (bug, enhancement, adaptation, etc.).
2. **Analysis and Feasibility Study**
   - Understand the change: What is the impact?
   - Check existing code, design documents, and requirements.
   - Assess cost, time, risk, and benefits.
   - Decide whether to accept, reject, or postpone the change.
3. **Design the Change**
   - Plan how to implement the change (modify modules, add new ones).
   - Update design documents if needed.
   - Ensure it does not break existing features.
4. **Implementation / Coding**
   - Make the actual code changes.
   - Follow coding standards.
   - Do unit testing on changed parts.
5. **Testing**
   - Test the modified software thoroughly.
   - Especially **regression testing** to ensure old features still work.
   - Types: Unit, integration, system, acceptance testing (if major change).
6. **Deployment / Release**
   - Install the updated version at user site (patch, new release).
   - May include data migration or user training.
   - Use controlled rollout (pilot, phased).
7. **Review and Documentation Update**
   - Record what was changed and why.
   - Update user manuals, technical documents, and knowledge base.
   - Collect feedback for future improvements.

6.a. Why do we need software testing? Explain any four types of software testing.

6. a. Importance and Types of Software Testing

Why do we need Software Testing? Testing is not just about finding bugs; it is a critical part of the development process for the following reasons:

1. **Reliability:** It ensures that the software performs its intended functions consistently without crashing.
2. **Cost-Effectiveness:** Finding a bug during the testing phase is much cheaper than fixing it after the software has been released to thousands of users.
3. **Security:** Testing helps identify vulnerabilities and "loopholes" that hackers could exploit to steal data.
4. **Customer Satisfaction:** It ensures that the final product meets the user's requirements and provides a smooth experience

Four Types of Software Testing (explained briefly):

1. **Unit Testing**
   - Tests the smallest part of the code (single function, method, or class) in isolation.
   - Done by developers during coding.
   - Uses tools like JUnit, NUnit.
   - Goal: Catch bugs early in individual pieces.
   - Example: Test if a "addTwoNumbers(3, 5)" function returns 8 correctly.
2. **Integration Testing**
   - Tests how different modules/units work together after combining them.

3. **Complete:** It must include all requirements—functional, non-functional, and safety requirements—without leaving anything out.
4. **Consistent:** Requirements should not contradict each other. For example, one page shouldn't say "Red button" and another page say "Blue button."
5. **Verifiable (Testable):** You should be able to test every requirement. Instead of saying "The app should be fast," say "The app should load in under 2 seconds."
6. **Modifiable:** The structure should be easy to change if the customer wants to add or update a requirement later.
7. **Traceable:** You should be able to track each requirement back to its source and forward to the specific code that implements it.
8. **Understandable by Customer:** It should be written in simple language so that even a non-technical customer can understand what is being built

3.a. Describe briefly project estimation techniques.

a. Project Estimation Techniques

Definition : Project estimation is the process of predicting the amount of time, money, and effort required to build a software system. It helps managers set deadlines and budgets.

Common Estimation Techniques:

1. **Expert Judgment:** Experienced experts look at the project and give their "best guess" based on similar projects they have done in the past. It is fast but can be biased.
2. **Analogy-based Estimation:** We compare the new project with a similar completed project from the past. If the old project cost $10,000, we estimate the new one similarly.
3. **Bottom-Up Estimation:** The project is broken down into very small tasks. We estimate the time/cost for each tiny task and then add them all up to get the total. It is very accurate but takes a lot of time.
4. **Top-Down Estimation:** We estimate the total cost of the whole system first and then divide that budget among the various departments or phases.
5. **Parametric Estimation (COCOMO):** This uses mathematical formulas to calculate effort. For example, the COCOMO model uses "Lines of Code" (LOC) as a parameter to calculate the total cost.

b. What is risk and risk analysis? Explain briefly about how we can manage risk.

What is Risk?

Risk is any uncertain event or condition that, if it happens, can have a **negative impact** on the project — like delay, extra cost, poor quality, or complete failure. Examples: Team member leaves, requirements change suddenly, new technology fails, budget cut.

What is Risk Analysis?

Risk analysis is the step where we study identified risks to understand:How likely is it to happen? (probability)How bad will the effect be? (impact / severity) We prioritize risks (high probability + high impact = most dangerous)

How to Manage Risk (Risk Management Process)

To score high marks, explain these 4 steps:

1. **Risk Identification:** Brainstorming to find all possible things that could go wrong (e.g., "The lead developer might quit" or "The budget might run out").
2. **Risk Assessment:** Ranking the risks based on their probability and impact so the team knows which ones to fix first.
3. **Risk Planning (Mitigation):** Creating a "Plan B."
    - *Avoidance:* Change the plan so the risk is gone.
    - *Transference:* Buy insurance or outsource the risky part to another company.
    - *Mitigation:* Take steps to reduce the impact (e.g., keep extra backups in case of data loss).
4. **Risk Monitoring:** Keeping an eye on the project to see if old risks have disappeared or if new ones have come up.

.

B .Write short notes on:

a. ER Diagram (Entity-Relationship Diagram)

Definition: An ER Diagram is a visual representation of a database's structure. It shows how "entities" (like people, objects, or concepts) relate to one another within a system.

Key Components:

1. **Entities (Rectangle):** Represents a real-world object, such as a *Student* or *Course*.
2. **Attributes (Ellipse/Oval):** Represents the properties of an entity, such as *Student_Name* or *Roll_No*.
3. **Relationships (Diamond):** Represents how two entities interact, such as a Student "Enrolls" in a Course.
4. **Lines:** Connect attributes to entities and entities to relationships.

Importance:

- It helps designers understand the data requirements clearly before creating the actual database tables.
- It serves as a blueprint for database administrators to implement the system in SQL.
- It defines "Cardinality" (e.g., One-to-One, One-to-Many, or Many-to-Many relationships).

b. Version Control System (VCS)

Definition: A Version Control System is a software tool that helps software teams manage changes to source code over time. It keeps a "history" of every modification made to the code.

How it Works:

- **Tracking Changes:** It saves a "snapshot" of the code every time a developer commits a change.
- **Branching:** It allows developers to work on a separate copy of the code (a branch) without affecting the main project.
- **Merging:** Once a feature is ready, the branch can be merged back into the main code.

Types:

1. **Centralized VCS:** A single server holds all versioned files (e.g., SVN).
2. **Distributed VCS:** Every developer has a full copy of the project history on their own machine (e.g., **Git**, which is the most popular).

Benefits:

- **Collaboration:** Multiple people can work on the same file simultaneously.
- **Reversibility:** If a new update breaks the system, you can easily "roll back" to a previous working version.

c. COCOMO Model

Definition: COCOMO stands for **COnstructive COst MOdel**. It is a mathematical model used to estimate the cost, effort, and schedule required for a software project based on the size of the code (measured in KLOC - Kilo Lines of Code).

The Three Levels of COCOMO:

1. **Basic COCOMO:** Uses a simple formula to give a rough estimate. It is good for quick, early-stage guesses.
2. **Intermediate COCOMO:** Factors in "Cost Drivers" such as hardware constraints, personnel capability, and experience.
3. **Detailed COCOMO:** Accounts for the influence of individual project phases (design, testing, etc.) on the total cost.

Project Categories:

- **Organic:** Small teams, simple projects, flexible requirements.
- **Semi-Detached:** Medium teams, average complexity, mixed experience.
- **Embedded:** Large projects with very strict constraints (e.g., Air Traffic Control or Banking systems).

4.a) What do you understand by software design strategies?Differentiate between function and object oriented design.

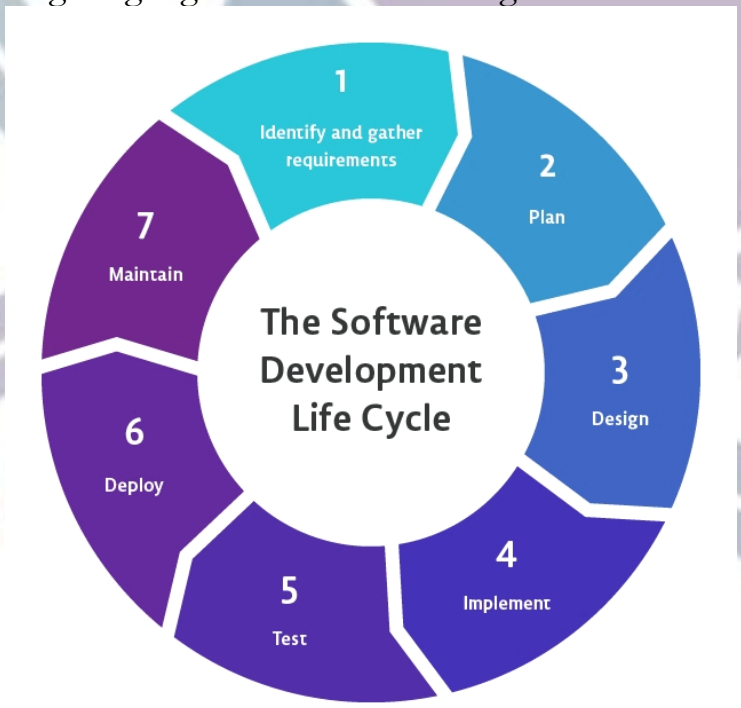
oftware Design Strategies Software design strategies are the different approaches or techniques used by designers to create the structure (architecture) and detailed organization of a software system. They guide how we divide the system into modules/components, how we arrange data and functions, and how we make the system easy to understand, change, test, and maintain.

S.No	Point of Comparison	Function-Oriented Design (FOD)	Object-Oriented Design (OOD)
1	**Basic Unit**	The primary unit is the **Function** or Procedure.	The primary unit is the **Object** or Class.
2	**Data & Functions**	Separate; data is usually passed as parameters.	Combined; data and methods are bundled together (**Encapsulation**).
3	**Main Focus**	Focuses on **What the system does** (the process).	Focuses on **What the system is** (entities and behavior).
4	**Design Approach**	Follows a **Top-Down** approach (Functional Decomposition).	Follows a **Bottom-Up** approach (starting with entities).
5	**Core Principles**	High cohesion, low coupling, and modularity.	**Encapsulation, Inheritance, Polymorphism,** and **Abstraction**.
6	**Data Protection**	**Low**; data is often global and easily accessible.	**High**; data is hidden (private/protected) and accessed via methods.
7	**Reusability**	**Moderate**; specific functions can be reused.	**High**; classes can be reused via inheritance and composition.
8	**Real-world Modeling**	Not very natural; treats systems as tasks.	Very natural; objects match real-world entities (e.g., Student, Car).
9	**Change Impact**	High; a change in a function may affect many parts.	Low; changes are mostly local (contained within the class).
10	**Typical Diagrams**	Uses **Data Flow Diagrams (DFD)** and Structure Charts.	Uses **UML Diagrams** (Class, Use Case, Sequence Diagrams).

b) **What is Software Development Life Cycle (SDLC)? Explain different phases of SDLC**.

Definition: SDLC is a structured "roadmap" or framework followed by software organizations to build high-quality software in a systematic way. It ensures the product is delivered on time, within budget, and meets all user needs.

- **Planning:** The base of the project. Here, we define the scope, estimate the cost, and check if the project is feasible (technically and financially).
- **Requirement Analysis:** This is the most critical phase. Senior team members gather detailed requirements from the customer to understand exactly "what" the software must do. The result is the **SRS (Software Requirement Specification)** document.
- **Design:** Architects create the "blueprint." This includes **High-Level Design (HLD)** (the overall system architecture) and **Low-Level Design (LLD)** (how specific modules and databases will look).
- **Implementation (Coding):** This is the longest phase. Developers write the actual code using chosen programming languages based on the design documents.



- **Testing:** Once the code is ready, the testing team looks for "bugs" or errors. They ensure the software works exactly as defined in the SRS.
- **Deployment:** After the software is bug-free, it is released to the customer. This can be a "Beta" release for a few users or a full launch.
- **Maintenance:** The work doesn't end at launch. This phase involves fixing new bugs that appear in the real world and updating the software to keep it working with new hardware or OS updates.

3. a) Explain briefly iterative enhancement model.

The Iterative Enhancement Model (also widely known as the Incremental Model or Iterative & Incremental Model) is a practical software development approach where the complete system is not built all at once. Instead, the product is developed and delivered in small, functional portions called increments or iterations.

How it Works:

1. **First Iteration:** You pick the most critical requirements and build a "core" version of the software.

2. **Feedback:** The customer uses this version and provides feedback.
3. **Enhancement:** In the next iteration, you fix any issues from the first version and add new features.
4. **Completion:** This cycle repeats until the final, full-featured product is ready.

Key Benefits:

- **Early Delivery:** The customer gets to see and use a working product very quickly.
- **Reduced Risk:** If there is a major error in the logic, it is found in the first iteration rather than at the end of the project.
- **Flexibility:** It is easy to change requirements even halfway through the project.

Typical flow (example – building an e-commerce mobile app)

1. Increment 1: User registration + login + view product catalog → Delivered in 4–6 weeks, users start using it
2. Increment 2: Add search + filters + add to cart
3. Increment 3: Checkout + payment gateway integration
4. Increment 4: Order tracking + reviews + wishlist Each increment is a **working, tested, usable mini-version**

b) What is software project scheduling? Explain time line charts with its benefits.

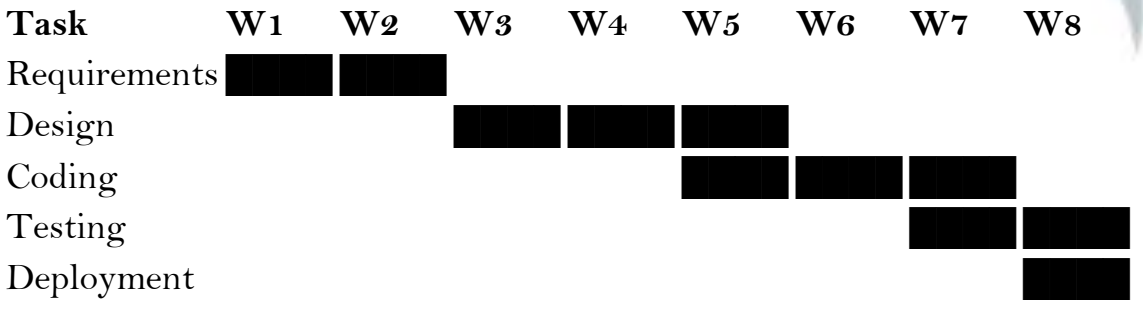
Scheduling is the process of distributing the total estimated effort (work) across a specific project duration. It involves identifying all the tasks that need to be done, deciding who will do them, and setting deadlines for each.

Timeline Charts Timeline charts are **visual diagrams** that show project tasks along a horizontal time axis (days, weeks, months). They make the schedule easy to read and understand at a glance.

Two most important types used in software projects

1. **Gantt Chart** (most popular)
   - Horizontal bars represent tasks
   - Length of bar = duration of the task
   - Position shows start and end date
   - Can show progress (shaded portion), dependencies (arrows), milestones (diamonds)

Simple text example of Gantt chart



Benefits of Timeline Charts:

1. **Visual Clarity:** It provides a clear, bird's-eye view of the entire project progress.
2. **Tracking Progress:** Project managers can easily see if a task is running late by looking at the current date on the chart.
3. **Dependency Management:** It shows which tasks must be finished before the next one can start (e.g., you cannot start Testing until Coding is done).
4. **Resource Allocation:** It helps in identifying if a team member is overloaded with too many tasks at the same time.
5. **Motivation:** Seeing the bars "fill up" as tasks are completed keeps the team motivated and on track.

2. **a) Define requirement elicitation. What characteristics of good SRS?**
3. **3. a) Define requirement elicitation. What are the characteristics of a good SRS?**
4. **Requirement Elicitation** Requirement elicitation (also called requirements gathering or requirements capture) is the process of **identifying**, **understanding**, **collecting**, and **documenting** the needs and expectations of the stakeholders (users, customers, business people, end-users) for the software system.

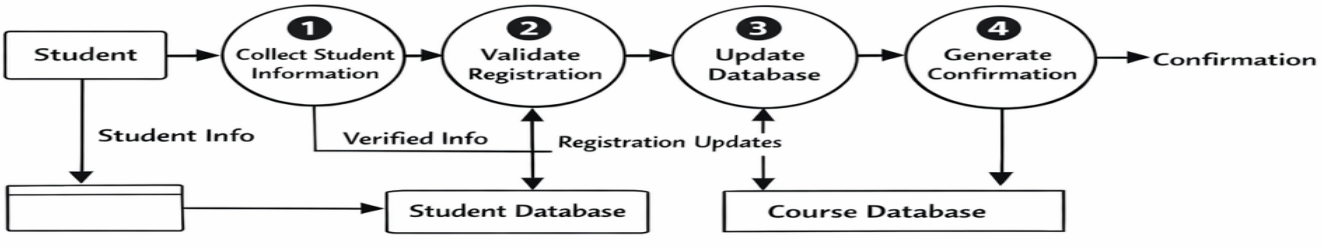
Characteristics of a Good SRS (Software Requirement Specification)

1. **Correct:** Every requirement should accurately represent a feature the system must have.
2. **Unambiguous:** Every requirement should have only **one** interpretation. No "confusing" language.
3. **Complete:** It should include all functional and non-functional requirements.
4. **Consistent:** One part of the SRS should not contradict another part.
5. **Verifiable (Testable):** You should be able to write a test case to check if the requirement is met (e.g., use "Response time < 2s" instead of "The system should be fast").
6. **Modifiable:** The document structure should allow for changes without rewriting the whole thing.
7. **Traceable:** You should be able to track a requirement back to its origin and forward to the code/test case.
8. **Prioritized:** Requirements should be ranked (e.g., Essential, Desirable, Optional).

b) Make Level 0 and Level 1 data flow diagram of student registration system.

Level 0 DFD: Student Registration System

Level 1 DFD: Student Registration System



Regular Exam-2080 Bhadra

Diploma in Computer Engineering

1. a) Define program and software. Explain briefly types of software.

Program A program is a set of instructions written in a programming language that tells the computer to perform a specific task or solve a particular problem. It is basically just the code (source code or compiled executable) designed for one job.

Software Software is a complete collection that includes one or more programs, along with associated data, configuration files, documentation, user manuals, and installation procedures. It is the full ready-to-use product that helps users perform tasks.

1. System Software

This is the foundational layer that manages hardware resources, provides services to other software, and acts as an intermediary between hardware and users/applications. Without system software, hardware is useless.

characteristics

- Runs in privileged mode (close to hardware)
- Usually loaded at boot time
- Low-level, efficient, and resource-intensive
- Rarely changed by end-users
- Focuses on **resource management** (CPU scheduling, memory allocation, I/O control, file systems, security)

Application Software

This layer is built for end-users to perform specific real-world tasks. It runs **on top of** system software and uses OS services.

Deep characteristics

- User-facing (GUI or command-line)
- Solves domain-specific problems
- Frequently updated
- Can be general-purpose or highly specialized
- Often customized or personalized

b) What is software metrics and why do we need them?Explain briefly various categories of software metrics.

Software metrics are quantitative measures that help us evaluate different aspects of a software product, the development process, or the project team. They turn subjective qualities (e.g., "this code is complex") into objective numbers so we can compare, predict, control, and improve.

Why do we need software metrics?

1. To **measure quality** objectively (instead of guessing)
2. To **predict effort, cost, time**, and defects early
3. To **find problems** early (high complexity → more bugs likely)
4. To **improve processes** and reach higher maturity (CMMI levels)
5. To **compare projects/teams** and track productivity
6. To **decide when to stop testing** (reliability goals met)
7. To **reduce risk** and **control costs** (fixing early is cheape

Various Categories of Software Metrics (with brief examples)

1. **Size Metrics** Measure how big the software is.
    - Lines of Code (LOC / KLOC)
    - Function Points (FP) – counts inputs, outputs, inquiries, files
    - Object Points, Use Case Points
2. **Complexity Metrics** Measure how difficult the code is to understand/maintain.
    - Cyclomatic Complexity (McCabe) = Edges – Nodes + 2 (independent paths)
    - Halstead Metrics (difficulty, effort, volume)
    - Nesting Depth, Fan-in/Fan-out
3. **Quality Metrics** Measure how good the software is.
    - Defect Density = Defects / Size (e.g., per KLOC)
    - Maintainability Index (MI) – score 0–100 based on complexity, LOC, comments
    - Reliability metrics (MTBF – Mean Time Between Failures)
    - Test coverage (statement, branch, path coverage)
4. **Productivity Metrics** Measure how efficiently the team works.
    - Function Points per person-month
    - LOC per person-day
    - Defects fixed per day
5. **Process Metrics** Measure the development process itself.
    - Defect Removal Efficiency (DRE) = (Defects found before release / Total defects) × 100
    - Schedule Variance, Cost Variance
    - Requirements stability index
6. **Resource / Project Metrics**
    - Effort (person-hours / person-months)
    - Schedule performance (actual vs planned time)
    - Cost performance index

b) List out types of software testing. Differentiate between white box testing and black box testing.

Types of Software Testing

Software testing is generally categorized into:

1. **Unit Testing:** Testing individual components or modules.
2. **Integration Testing:** Testing the interaction between integrated modules.
3. **System Testing:** Testing the complete, integrated system as a whole.
4. **Acceptance Testing:** Final testing by the user (Alpha/Beta testing).
5. **Regression Testing:** Re-testing after changes to ensure no new bugs were introduced.

S.No	Point of Comparison	Black Box Testing	White Box Testing
1	**Basic Definition**	Testing based on external specifications and requirements.	Testing based on internal logic and code structure.
2	**Full Knowledge**	Tester has no knowledge of the internal code or logic.	Tester has full knowledge of the source code and design.
3	**Primary Goal**	To check if the system performs what it is supposed to do.	To check how the system performs its internal operations.
4	**Who Performs It?**	Usually done by independent Software Testers.	Usually done by Software Developers.
5	**Testing Level**	Mainly used for System and Acceptance testing.	Mainly used for Unit and Integration testing.
6	**Techniques Used**	Boundary Value Analysis, Equivalence Partitioning.	Statement Coverage, Branch Coverage, Path Testing.
7	**Programming Skill**	No programming or coding knowledge is needed.	High-level programming and implementation knowledge is required.
8	**Visibility**	Internal workings are "hidden" (Opaque box).	Internal workings are "visible" (Transparent/Glass box).
9	**Time Consumption**	Less time-consuming and can be done quickly.	More time-consuming as it involves detailed code analysis.
10	**Granularity**	Focuses on end-to-end functionality of the system.	Focuses on specific blocks, loops, and statements in code.

6. a) Explain briefly software quality attributes.

oftware quality attributes (also called **non-functional requirements** or **quality factors**) are the characteristics that define **how well** a software system performs its functions, rather than **what** it does. They describe the overall goodness, usability, and long-term value of the software.

These attributes are defined in standards like **ISO/IEC 25010** (the most widely used modern model). The main quality attributes are:

1. **Functionality** The degree to which the software provides the functions that meet stated and implied needs (accuracy, completeness, suitability).
2. **Performance Efficiency** How fast and resource-efficient the software is (time behavior, resource utilization, capacity).
3. **Compatibility** Ability to work with other systems/products (co-existence, interoperability).
4. **Usability** Ease of use and satisfaction for users (learnability, operability, user error protection, aesthetics, accessibility).
5. **Reliability** The capability to perform under stated conditions for a specified period (maturity, availability, fault tolerance, recoverability).
6. **Security** Protection of information and data (confidentiality, integrity, non-repudiation, accountability, authenticity).
7. **Maintainability** Ease of modifying, fixing, or improving the software (modularity, reusability, analyzability, modifiability, testability).
8. **Portability** Ability to transfer the software from one environment to another (adaptability, installability, replaceability).

b) What is software maintenance? Why do we need software maintenance? Explain different types of software maintenance.

Software maintenance is the process of modifying, updating, correcting, and enhancing a software system after it has been delivered to the customer and is in operation (live use). It includes all activities to keep the software correct, useful, efficient, and compatible with changing needs and environments throughout its lifetime

Why do we need software maintenance?

- Software never "wears out" physically like hardware, but it becomes outdated or defective due to:
  - Bugs discovered after release
  - Changes in user requirements or business rules
  - New hardware, OS, browsers, databases, or laws
  - Performance or security issues over time
- Without maintenance, software becomes unusable, insecure, non-compliant, or too expensive to operate.
- **Statistics**: Maintenance often accounts for **60–80%** of the total software life-cycle cost — far more than development.
- Maintenance keeps software valuable, safe, and competitive for many years (sometimes 10–20+ years).

Different Types of Software Maintenance (As per IEEE/ISO standards — four main categories)

1. **Corrective Maintenance** Fixing defects, bugs, or errors found after delivery.
   - Reactive — done when something breaks.
   - Example: User reports "app crashes on login" → fix the login bug.
2. **Adaptive Maintenance** Modifying the software to work in a changed or new environment.
   - Caused by external changes (not bugs).
   - Example:
     - Update app to support new Android/iOS version
3. **Perfective Maintenance** Improving the software even when it is working correctly — to make better.
   - Adds new features, improves performance, usability, or efficiency.
   - Example:
     - Add dark mode or faster search
4. **Preventive Maintenance** Making changes proactively to **prevent future problems**.
   - Improves maintainability for future updates.
   - Example:

- Refactor messy/old code

7. Write short notes on:

.

(a) COCOMO Model –

1. COCOMO stands for **Constructive Cost Model**.
2. It is used to **estimate cost, effort, and development time** of software.
3. Estimation is mainly based on **size of software (KLOC)**.
4. It helps project managers in **planning manpower and schedule**.
5. The model is used in the **early stage of software development**.
6. **Basic COCOMO** gives rough estimation using simple formulas.
7. **Intermediate COCOMO** considers cost drivers like hardware and team experience.
8. **Detailed COCOMO** estimates effort for each phase of development.

(b) Verification vs Validation –

1. **Verification** checks whether the software is built **correctly as per design**.
2. **Validation** checks whether the software **meets user requirements**.
3. Verification is a **static process** (no execution of code).
4. Validation is a **dynamic process** (code is executed).
5. Verification includes **reviews, inspections, and walkthroughs**.
6. Validation includes **testing activities**.
7. Verification answers: *"Are we building the product right?"*
8. Validation answers: *"Are we building the right product?"*

(c) Activities in Project Management –

1. **Project planning** defines project goals, scope, and resources.
2. **Project scheduling** decides time and order of activities.
3. **Cost estimation** estimates total project budget.
4. **Resource management** assigns manpower, tools, and equipment.
5. **Risk management** identifies and reduces possible project risks.
6. **Monitoring and control** tracks progress of the project.
7. **Quality management** ensures software meets quality standards.
8. **Project closure** completes documentation and delivers the project.

5. a) What is software reliability? Differentiate between software reliability prediction model and estimation model.

Definition: Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Unlike hardware, software does not "wear out"; reliability issues in software are usually due to hidden design or coding defects.

S.No	Point of Comparison	Reliability Prediction Model	Reliability Estimation Model
1	**Timing**	Used during the **early stages** (Requirement/Design).	Used during the **later stages** (Testing/Operation).
2	**Data Source**	Based on **historical data** from similar past projects.	Based on **actual data** from the current project's testing.
3	**Primary Objective**	To predict reliability before the code is even written.	To assess current reliability based on observed failures.
4	**Input Parameters**	Uses project size, complexity, and team experience.	Uses failure intervals and time between failures (TBF).
5	**Application Phase**	Mostly applied in the **Development Phase**.	Mostly applied in the **Testing and Maintenance Phase**.
6	**Accuracy**	Generally **less accurate** as it relies on assumptions.	Highly **accurate** as it uses real-time execution data.
7	**Failure Data**	Does not require actual failure data to start.	Requires a significant amount of **actual failure data**.
8	**Dependency**	Dependent on the quality of historical databases.	Dependent on the thoroughness of the testing process.
9	**Usage for Managers**	Helps in early budgeting and resource planning.	Helps in deciding if the software is "Ready to Release."
10	**Key Examples**	Musa's Execution Time Model (early stage), Phase-based.	Logarithmic Poisson Model, Jelinski-Moranda Model.