CBCS SCHEME

USN

BCS501

Fifth Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Software Engineering and Project Management

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	a.	Explain the domains of software applications.	08	L2	CO1
	b.	Outline the unique nature of WebApps.	08	L2	CO1
	c.	Explain various software myths. Discuss.	04	L2	CO1
		OR			
Q.2	a.	Explain the activities performed in a software process framework?	06	L2	CO1
	b.	Explain the waterfall model along with its pros and cons.	08	L2	CO1
	c.	Explain specialized process models.	06	L2	CO1
		Module – 2			
Q.3	a.	Explain how groundwork parameters are established in requirements engineering.	08	L2	CO2
	b.	What is the importance of quality function deployment in eliciting requirements?	06	L1	CO2
	c.	How can we validate requirements?	06	L1	CO2
		OR			
Q.4	a.	Explain about scenario based modelling.	10	L2	CO2
	b.	Illustrate regarding how can we create a Behavioral Model.	10	L2	CO2
		Module – 3			
Q.5	a.	Explain Agility along with the principles of Agility.	10	L2	CO3
	b.	Explain the Extreme Programming Process.	06	L2	CO3
	c.	Explain about the critics of XP.	04	L2	CO3
		OR			
Q.6	a.	Explain the scrum flow process.	08	L2	CO3
	b.	Explain the communication principles guiding framework activity.	08	L2	CO3
	c.	How can we validate and test principles in coding.	04	L1	CO3
		Module – 4			
Q.7	a.	Define Project. Show the contrast of software projects with other types of projects.	06	L2	CO4
	b.	Explain the ISO 12207 software development life cycle with a neat diagram.	10	L2	CO4
	c.	What are outsourced projects?	04	L1	CO4
		OR			
Q.8	a.	Illustrate the cost benefit evaluation techniques.	10	L2	CO4
	b.	Illustrate the concept of Risk evaluation.	10	L2	CO4
		Module – 5			
Q.9	a.	Explain the details to be drafted for achieving quality in software.	06	L2	CO5
	b.	Explain the software quality characteristics of ISO 9126.	08	L2	CO5
	c.	Explain process requirements for the process quality management.	06	L2	CO5
		OR			
Q.10	a.	Explain about the decomposition techniques.	10	L2	CO5
	b.	Explain the COCOMO II model.	10	L2	CO5

* * * * *

Software Engineering & Project Management (BCS501) SEM-5
VTU Question & Solution (December 2025)
Prof. Lynsha Pratheeba HP and Prof. Rajeshwari R
Department of CSE

MODULE-1

Q1 a. Explain the domains of Software Applications [8 Marks]

1. System Software
2. Application Software
3. Engineering & Scientific Software
4. Embedded Software
5. Product-line Software
6. Web Application Software
7. AI Software

Q.1.b) Outline the unique nature of WebApps. [8 Marks]

1. Network intensiveness – Internet and browsers
2. Concurrency – a greater number of users access the webapp at a time.
3. Unpredictable load – day to day users may increase
4. Performance – wait too long
5. Availability – 24/7 demand access
6. Content sensitive
7. Continuous evolution
8. Immediacy
9. Security – protect sensitive contents
10. Aesthetics – look and marketing

8 point with description carries 8 marks

Q.1.c) Explain various software myths. [4 Marks]

1. Management Myths
2. Customer Myths
3. Practitioner's Myths

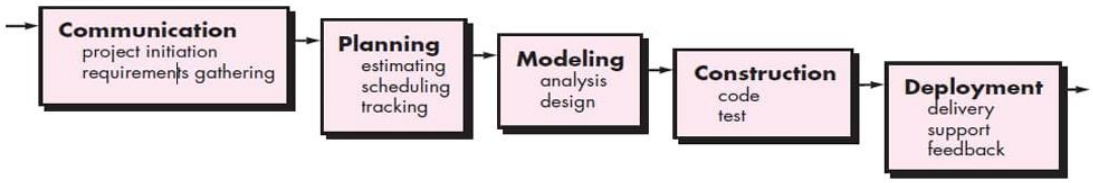
Any 4 myth and reality under each heading carries 4 Marks.

Q.2.a) Explain how groundwork parameters are established in Requirements Engineering [8 Marks]

1. Defining s framework activity
2. Identify a task
3. Process Patterns

Dexcription: 3*2 = 6 marks

Q.2.b) Explain the waterfall model along with its pros and cons. [8 Marks]



- The waterfall model, sometimes called the classic life cycle, suggests

a systematic sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment

Pros:
The waterfall model is simple and easy to understand, implement, and use.
All the requirements are known at the beginning of the project, hence it is easy to manage
Cons:
This model is not good for complex and object-oriented projects
It is a poor model for long projects

Diagram with description (4M) and Pros snd cons (4M)

Q.2.b) Explain Specialized Process Model. [6 Marks]

1. Component-Based Development
2. The Formal Methods Model
3. Aspect-Oriented Software Development

Description for each carries 2M.

MODULE-2

Q.3.a) Explain how groundwork parameters are established in Requirements Engineering. [8 Marks]

1. Identifying stakeholders
2. Identifying multiple viewpoints
3. Working towards collaboration
4. Asking the first question

Description: 4*2=8M

Q.3.b) What is the importance of quality function deployment in eliciting requirements. [6 Marks]

1. Normal Requirements
2. Expected Requirements
3. Exciting Requirements

Description: 3*2=6M

Q.3.c) How can we validate requirements. [6 Marks]

1. **If each requirement consistent with the overall objectives for the systems**
2. **Have all requirements been specified at the project level of abstraction.**

Description: 3*2=6M

Q.4.a) Explain about scenario-based modeling. [10 Marks]

1. Creating a Preliminary use case
2. Refining a preliminary use case
3. Writing a formal use case

Description: 4+2+2=10M

Q.4.b) Illustrate regarding how can we create a Behavioral model [10 Marks]

1. Identifying events with the use case
2. State Representations
    a. State diagrams for analysis usecase
    b. Sequence diagram

Description" 4+6=10M

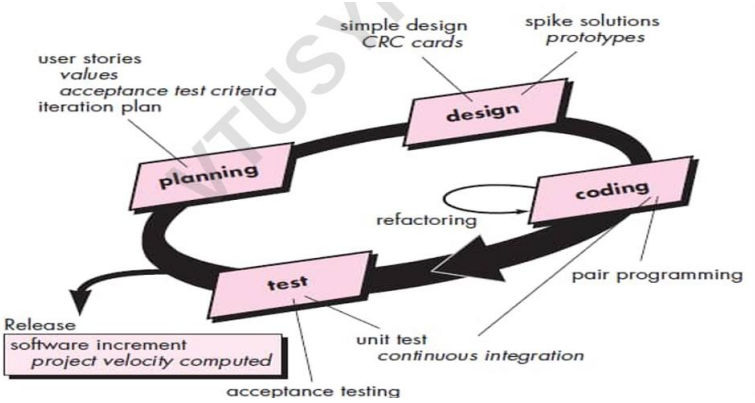
MODULE-3

Q.5.a) Explain Agility along with the principles of agility. [10 Marks]

1. Satisfy the customer
2. Welcome changing requirements
3. Deliver working software frequently
4. motivated individuals
5. face–to–face conversation
6. Working software is the primary measure of progress
7. sustainable development
8. technical excellence and good design
9. Simplicity
10. self–organizing teams

Description: each carries 1M with explanation = 10M

Q.5.b) Explain the Extreme Programming Process. [6 Marks]



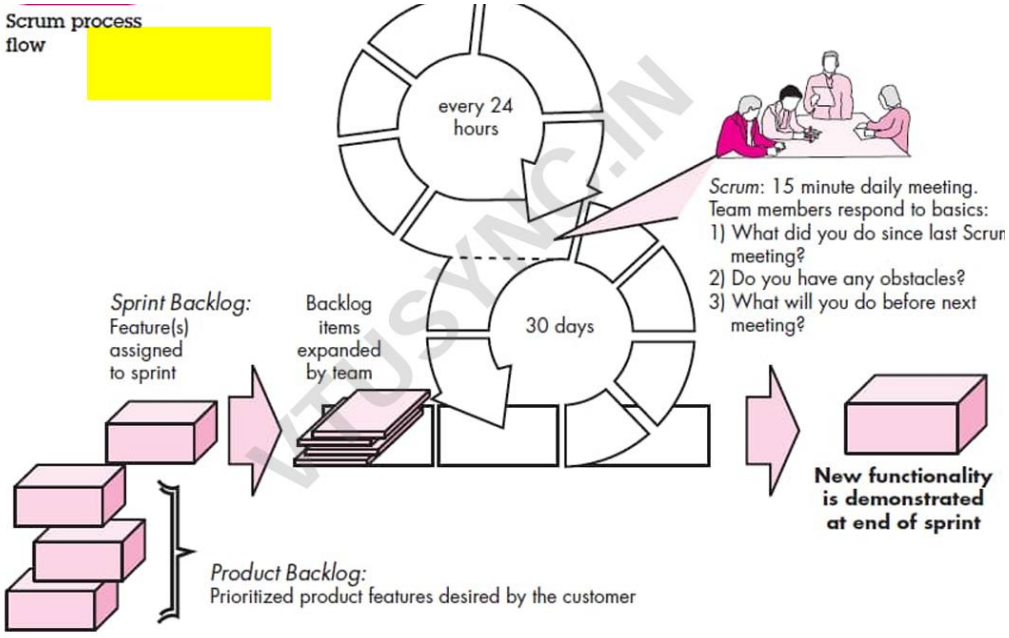
Description: Diagram (1M)+Explanation (5M)=6M

Q.5.c) Explain about the critics of XP. [4 Marks]

1. Requirements volatility
2. Conflicting customer needs
3. Requirements are expressed informally
4. Lack of formal design

Description: Explain 4 points with explanation = 4M

Q.6.a) Explain Scrum flow Process. [8 Marks]



1. Backlog
2. Sprints
3. Scrum meetings
4. Demos

Description: Diagram (2M) + Explanation (1M+1M+2M+2M) = 8M

Q.6.b) Explain the communication principles guiding framework activity [8 Marks]

1. Listen
2. Prepare before you communicate
3. Someone should facilitate the activity
4. Face to face communication
5. Take notes
6. Document the decisions
7. Strive for collaboration
8. Stay focused

Description: Each explanation carries 1M= 8M

Q.6.c) How can we validate and test principles in coding. [4 Marks]

1. Validate
    a. Conduct code walk-through when appropriate
    b. Perform unit tests
    c. Refactor the code

2. Testing
    a. Process of executing a program with the intent of finding an error
    b. A good test case finds a unidirectional error

    Description: 2M+2M=4M

MODULE-4

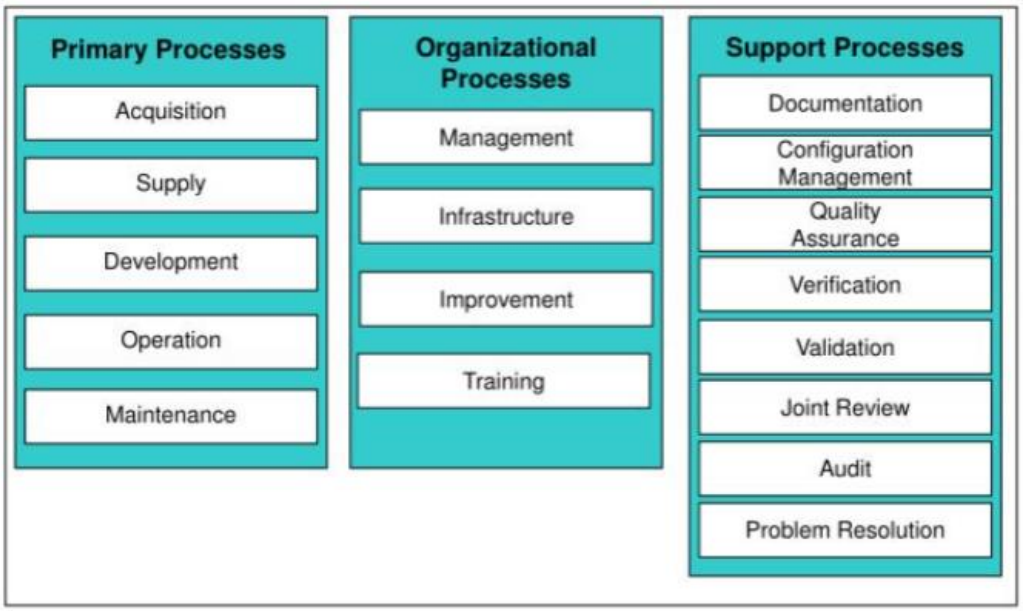
Q.7.a) Define Project. Show the contrast of software project with other types of projects. [6 Marks]

A non-routine task.

1. Invisibility
2. Complexity
3. Conformity
4. Flexibility

Description: Definition (2M) + Explanation (4M) = 6M

Q.7.b) Explain the ISO 12207 software development life cycle with a neat diagram. [10 Marks]



1. Primary Life Cycle Processes
2. Supporting Life Cycle Processes
3. Organizational Life Cycle Processes

Description: Diagram (3M) + Explanation (2M+2M+3M) = 10M

Q.7.c) What are the outsourced projects. [4 Marks]

1. Lack of expertise, cost-effectively by others
2. Impact on profitability

Description – 4M

Q.8.a) Illustrate the cost-benefit evaluation techniques. [10 Marks]

1. Net Profit (NP)
2. Payback Period
3. Return on Investment (ROI)
4. Net Present Value (NPV)
5. Internal Rate of Return (IRR)

Description – Each explanation with formula 2M = 10M

Q.8.b) Explain the concept of risk evaluation. [10 Marks]

Risk evaluation is the process of comparing estimated risks against predefined risk criteria or tolerance levels to prioritize them and decide on appropriate responses, such as acceptance, avoidance, transfer, or mitigation.

Components

- Likelihood Assessment
- Impact Assessment

- Risk Criteria

Main Techniques

- Qualitative
- Quantitative
- Semi-Quantitative
- Specialized

Process Steps

1. Analyze risk level using data or expert judgment.
2. Compare against tolerance
3. Prioritize and recommend actions

Descriprion: Definition is 2M + Components is 2M + Techniques is 4M + Process is 2M = 10 M

Q.9.a) Explain the details to be drafted for achieving quality in software [6 Marks]

1. Introduction to Software Quality (**1 mark**)
Briefly define what software quality is: Software quality refers to the degree to which software meets the requirements of functionality, performance, reliability, maintainability, and security. It involves ensuring that software is defect-free and satisfies user needs.

2. Key Practices for Achieving Quality (**2 marks**)
Requirements Analysis: Ensuring clear and testable requirements.
Design and Architecture: Applying best practices in design to ensure scalability and maintainability.
Testing: Implementing various levels of testing (unit, integration, system, acceptance) to detect defects.
Continuous Integration and Delivery (CI/CD): Automating processes for rapid testing and deployment to detect defects early.

3. Tools and Techniques for Quality Assurance (**1 mark**)
Test Automation: Using automated testing tools to improve efficiency.
Static Code Analysis: Tools that automatically detect coding issues and potential defects.
Code Reviews: Peer reviews to catch errors early and ensure code quality.

4. Metrics for Measuring Quality (**1 mark)**
Defect Density: Number of defects per unit of code.
Code Coverage: Percentage of code tested by automated tests.
Customer Satisfaction: Feedback from users and stakeholders.

5. Continuous Improvement and Team Collaboration (**1 mark)**
Continuous Improvement: Iteratively refining processes based on feedback and metrics.
Collaboration: Fostering collaboration between developers, testers, and stakeholders to address quality concerns at all stages.

Q.9.b) Explain the software quality characteristics of ISO9126
[8 Marks]

1. Functionality (1.5 marks)

Functionality refers to the degree to which the software meets the specified requirements and performs the intended tasks.

- **Suitability**: The software's ability to meet the functional requirements and provide necessary features.
- **Accuracy**: The correctness of the software's output, ensuring it meets expected results.
- **Interoperability**: The software's ability to interact with other systems, databases, or software components.
- **Security**: The protection of software from unauthorized access or malicious attacks.

Example: An email application must be able to send and receive messages (suitability), ensure no errors in sending/receiving (accuracy), and interact with external email servers securely (security).

2. Reliability (1.5 marks)

Reliability measures the software's ability to perform under specified conditions without failure.

- **Maturity**: The frequency of defects or failures. Mature software experiences fewer failures.
- **Fault Tolerance**: The ability of the software to function correctly even in the event of unexpected conditions or failures.
- **Recoverability**: The software's ability to recover from failures with minimal disruption to operations.
  Example: A banking system must continue processing transactions smoothly, even if a server goes down temporarily (fault tolerance), and quickly recover without data loss (recoverability).

3. Usability (1.5 marks)

Usability evaluates how easy and user-friendly the software is for its target audience.

- **Understandability**: The ease with which users can understand the software's functionality.
- **Learnability**: How easy it is for new users to become proficient in using the software.
- **Operability**: The ease with which users can interact with the software to achieve their goals.

Example: A mobile app with a simple, intuitive interface that users can navigate without much effort demonstrates high usability.

4. Efficiency (1 mark)

Efficiency refers to how well the software uses system resources while performing its tasks.

- **Time Behavior**: The software's responsiveness and the time it takes to complete a given task or process.
- **Resource Utilization**: The amount of system resources (CPU, memory, disk space) the software requires for operation.

Example: A video game should run smoothly without excessive lag (time behavior), even on devices with limited resources (resource utilization).

5. Maintainability (1 mark)

Maintainability assesses how easily the software can be modified after release.

- **Analyzability**: The ease with which software defects or issues can be identified.
- **Changeability**: The ease with which the software can be modified or updated.
- **Stability**: The software's ability to undergo changes without introducing new defects.
- **Testability**: The ease with which the software can be tested to ensure modifications do not introduce new issues.

Example: A system with modular design and clear documentation will be easier to maintain and update over time.

6. Portability (1 mark)

Portability measures the software's ability to operate in different environments or systems.

- **Adaptability**: The ability of the software to be adapted to different environments (e.g., different operating systems).
- **Installability**: How easy it is to install the software in different environments.
- **Co-existence**: The software's ability to work alongside other software without conflicts.

Example: A web-based application that works on both Windows and macOS platforms with minimal changes demonstrates good portability.

Q.9.c) Explain process requirements for the process quality management [6 Marks]

1. Definition of Process Quality Management (**1 mark**)

Process Quality Management (PQM) is a set of activities and techniques used to ensure that software processes meet defined standards and achieve the desired outcomes. It involves planning, controlling, measuring, and improving processes to ensure that software development is efficient, effective, and aligned with the organization's objectives.

2. Key Process Requirements for Process Quality Management (**4.5 marks**)

a) Process Definition (**1 mark**)

- To effectively manage and improve a process, the process must first be well-defined. This includes clearly identifying the process steps, inputs, outputs, roles, responsibilities, and tools required.
- Each process should have specific objectives and measurable performance indicators.

  Example: A "Code Review" process should have clearly defined steps such as: selecting the code to review, assigning reviewers, establishing review criteria, and tracking defects.

b) Process Measurement (**1 mark**)

- To monitor the performance of a process, it is essential to define metrics that help measure the process's effectiveness, efficiency, and compliance with quality standards.
- Common process metrics might include defect rates, process cycle time, or compliance with process steps.

Example: In the software testing process, metrics like defect density (defects per thousand lines of code) or test coverage (percentage of code covered by tests) can be tracked.

c) Process Control (**1 mark**)

- Process control involves ensuring that the process is followed consistently and effectively. This includes regular monitoring of the process, taking corrective actions if performance deviates from expectations, and addressing any non-conformance.
- Control mechanisms can include checklists, audits, reviews, and quality gates (go/no-go decision points).

Example: A code quality gate that checks if the code meets the required coding standards before being integrated into the main codebase.

d) Process Improvement (**1.5 marks**)

- Continuous improvement is a key requirement for process quality management. After measuring and controlling processes, organizations should focus on identifying opportunities for process optimization and refinement.

- Improvement initiatives may involve techniques such as Root Cause Analysis (RCA), Six Sigma, or Lean principles to remove inefficiencies and defects.

- PDCA (Plan-Do-Check-Act) or CMMI (Capability Maturity Model Integration) can be adopted as frameworks for continuous improvement.

Example: If a process reveals frequent delays in code deployment, root cause analysis could reveal inadequate testing times. In response, the team might adjust the process to allow more testing time or automate parts of the testing.

3. Integration with Organizational Goals (**0.5 marks**)

- Process Quality Management should be aligned with the broader organizational goals. This means that the processes should not only be optimized for efficiency but also deliver value that supports the organization's mission, such as customer satisfaction, cost reduction, or market competitiveness.

Example: A software company may focus on process improvements that shorten the software delivery cycle to meet customer demands for quicker feature releases.

Q.10.a) Explain about the decomposition techniques. **[10 Marks]**

Definition of Decomposition (1 mark)

- **Decomposition** refers to the process of dividing a complex problem or system into smaller, easier-to-understand sub-problems or components. This simplifies problem-solving, design, and project execution.

2. Types of Decomposition Techniques (7 marks)

1. **Functional Decomposition (2 marks)**
    - **Description**: Breaks down a system or process into smaller functions or tasks. This technique is primarily used to focus on specific functionalities and how they relate to the overall system.
    - **Purpose**: To divide the system into manageable components, each of which performs a specific function.
    - **Example**: In software development, a system might be decomposed into modules like "User Authentication," "Data Management," and "Report Generation."

2. **Short Key**:
    - **Focus**: Functions
    - **Example**: Banking system (Login, Transfer, Balance Check).

3. **Object-Oriented Decomposition (2 marks)**

    - **Description**: Decomposes a system based on objects and their interactions, as per **Object-Oriented Design (OOD)** principles. It focuses on identifying objects (classes) and how they interact with each other.
    - **Purpose**: To create classes and objects that can be reused, modified, and tested independently.
    - **Example**: A library management system might be decomposed into objects like "Book," "Member," and "Librarian."

4. **Short Key**:

    - **Focus**: Objects and classes
    - **Example**: Library system (Book, Member, Librarian).

5. **Data Decomposition (2 marks)**

    - **Description**: Focuses on dividing the data structure into smaller, manageable parts. It is often used in database design, where data entities are broken down into smaller tables or records.
    - **Purpose**: To simplify data handling and improve performance.
    - **Example**: A product database might be decomposed into separate tables for "Products," "Categories," "Suppliers," and "Sales."

6. **Short Key**:

    - **Focus**: Data structures
    - **Example**: Database (Products, Categories, Sales).

7. **Top-Down Decomposition (1 mark)**

    - **Description**: Begins with a high-level overview of the system and then progressively breaks it down into more detailed components. It is a hierarchical approach to decomposition.
    - **Purpose**: To provide an overall structure and clarity, starting from abstract concepts to concrete details.
    - **Example**: For a software project, the system can be decomposed from the entire system to the application components, and then to specific modules.

8. **Short Key**:

    - **Focus**: High-level to detailed breakdown
    - **Example**: Software system -> Application -> Module -> Class.

9. **Bottom-Up Decomposition (1 mark)**

    - **Description**: Starts with the identification of smaller, simpler components or tasks and works up towards creating the entire system.

Often used for systems that are composed of already existing components.
    - **Purpose**: To focus on reusability and integration of small components into a larger system.
    - **Example**: In building a software system, developers may begin by creating and testing individual modules before combining them into a full-fledged application.

10. **Short Key**:

    - **Focus**: Small components to large system
    - **Example**: Individual module testing -> Integration -> Full System.

3. Advantages of Decomposition (1 mark)

- **Simplification**: Breaks down complex problems or systems into smaller, more manageable parts.
- **Improved Collaboration**: Teams can work on different components independently, speeding up the development process.
- **Reusability**: Allows for the reuse of components or modules in different parts of the system or in future projects.
- **Better Management**: Helps in better tracking, testing, and maintaining smaller parts of the system.
- **Scalability**: Makes it easier to scale the system by modifying or adding components without affecting the whole system.

4. Challenges of Decomposition (1 mark)

- **Complexity in Integration**: While individual components may be easier to manage, integrating them into a cohesive whole can be complex.
- **Overhead**: Decomposing a system into too many parts can lead to unnecessary complexity and overhead in managing the interactions.
- **Dependency Management**: Ensuring that the decomposed components work together without introducing dependencies that complicate future changes can be challenging.

Q.10.b) Explain the COCOMO II Model **[10 Marks]**

The COCOMO II (Constructive Cost Model II) is an estimation model used to predict the cost, effort, and schedule required for a software project based on the project's size and other attributes. It is an enhancement of the original COCOMO model (Constructive Cost Model), and it is designed to provide more accurate predictions for modern software development practices.

COCOMO II was developed by Barry Boehm and is widely used in software engineering for project estimation. It applies to various software development

environments and includes parameters for the evolving nature of software projects, including newer paradigms like rapid prototyping and software reuse.

1. Overview of COCOMO II Model (**1 mark**)

COCOMO II is a parametric estimation model that calculates the cost, effort, and schedule of a software project by considering factors like project size, complexity, development environment, and team capability. It includes three primary stages:

- Early Design Stage
- Post-Architecture Stage
- Application Composition Stage

These stages provide flexibility in estimating software projects at different stages of development.

2. Phases of COCOMO II (**6 marks**)

a) Early Design Stage (**1.5 marks**)

- Purpose: The Early Design stage is used to provide a rough estimate of the project during the initial stages of development, often based on limited information.
- Model Type: Uses the basic COCOMO II equation to estimate the cost based on the size of the software (in KLOC, i.e., thousands of lines of code).
- Key Equation:
  Effort Applied (person-months)=A×(KLOC)E\text{Effort Applied (person-months)} = A \times (KLOC)^EEffort Applied (person-months)=A×(KLOC)E
  Where:
  - A and E are constants derived from historical project data, and
  - KLOC is the size of the software in thousands of lines of code.
- Factors Considered:
  - Software size (KLOC)
  - Cost drivers like complexity, experience, and tools.

Short Key:

- Use: Initial estimation
- Key Output: Effort (person-months), Schedule (months)

b) Post-Architecture Stage (**2 marks**)

- Purpose: This stage provides more accurate estimates after the system architecture has been defined and more information is available about the project.
- Model Type: The Post-Architecture stage uses a more refined version of the COCOMO II model with adjustments for factors such as reuse, team capability, and tool support.

- Key Equation:
  Effort Applied (person-months)=A×(KLOC)E×∏(Cost Drivers)\text{Effort

Applied (person-months)} = A \times (KLOC)^E \times \prod (\text{Cost Drivers})Effort Applied (person-months)=A×(KLOC)E×∏(Cost Drivers)
Where:
    - Cost Drivers are a set of parameters that reflect the environment and team factors (e.g., team experience, software tools, etc.)
- Cost Drivers (Factors) include:
    - Product Reliability
    - Data Complexity
    - Team Capability
    - Process Maturity
    - Software Tools
- Output: Provides estimates for effort, schedule, and personnel required.

Short Key:

- Use: After architecture is defined
- Factors Considered: Team, tools, product reliability

c) Application Composition Stage (**2.5 marks**)

- Purpose: This stage applies to projects where a significant amount of software is being developed through the use of reusable components or in an environment where software reuse is prevalent.
- Model Type: It involves a more detailed estimation process, focusing on development time, personnel required, and software components reused.
- Key Equation:
  Effort Applied (person-months)=A×(KLOC)E×∏(Cost Drivers)\text{Effort Applied (person-months)} = A \times (KLOC)^E \times \prod (\text{Cost Drivers})Effort Applied (person-months)=A×(KLOC)E×∏(Cost Drivers)
  This is similar to the Post-Architecture stage but places more emphasis on the reuse factor.
- Output: Includes estimates for effort, schedule, and resources, with a focus on reusability.

Short Key:

- Use: Reuse-based development
- Focus: Effort and personnel required

3. Cost Drivers in COCOMO II (**2 marks**)

Cost drivers are the factors that affect the estimation of effort, schedule, and resources in the COCOMO II model. These drivers adjust the basic model equation based on the specific characteristics of the project.

Key Cost Drivers:

- Product Reliability: The degree of reliability required by the system (e.g., mission-critical systems have higher effort estimates).
- Data Complexity: The complexity of the data being processed, which can increase the effort required for the system.

- Personnel Capability: The skill and experience level of the project team.
- Use of Software Tools: The impact of using tools (e.g., CASE tools, version control) on efficiency.
- Development Flexibility: The flexibility of the development process, which can affect time and effort (e.g., a highly flexible approach may lead to more effort).
- Required Software Reusability: The extent to which the project depends on reusable components.

Each of these drivers has a rating scale that modifies the effort estimate based on the specific conditions of the project.

4. COCOMO II Equations and Parameters (**1.5 marks**)

The general form of the COCOMO II equation is:

Effort Applied (person-months)=A×(KLOC)E×∏(Cost Drivers)\text{Effort Applied (person-months)} = A \times (KLOC)^E \times \prod (\text{Cost Drivers})Effort Applied (person-months)=A×(KLOC)E×∏(Cost Drivers)

Where:

- A = scaling constant (specific to the project type)
- KLOC = estimated size of the project in thousands of lines of code
- E = exponent derived from historical data
- Cost Drivers = multipliers based on project attributes (product reliability, team capability, etc.)

For example, if the cost driver for team capability is rated as “high,” the corresponding multiplier would increase the total effort estimate.

5. Advantages of COCOMO II (**1 mark**)

- Flexibility: COCOMO II can be used at different stages of the project (early design, post-architecture, and application composition).
- Accuracy: It improves over its predecessor, COCOMO, by accounting for modern software development practices like rapid prototyping and reuse.
- Customizability: Adjusts for a wide range of factors affecting software projects, including team skill, process maturity, and tool usage.