USN | | | | | | | | | |

Internal Assessment Test 2 – November 2025

Sub:	Software Engineering and Project Management				Sub Code:	BCS501	Branch:	AIML/CSE-AIML		
Date:	02/12/2025	Duration:	90 mins	Max Marks:	50	Sem/Sec:	V – A/B/C		OBE	

Answer any FIVE FULL Questions		MARKS	CO	RBT
1a	Analyze the software engineering principles that guide any one framework activity.	5	CO3	L4
1b	Differentiate the components of SMART objectives and examine how each element influences the success of a project plan.	5	CO5	L4
2	Explain the importance of project management in software development.	10	CO3	L2
3	Describe the major activities in software project management.	10	CO4	L2
4	Apply the phases of the project management life cycle to a suitable project scenario.	10	CO3	L3
5	Apply function point–based decomposition to estimate the size of a software system.	10	CO5	L3
6	Explain the components of the COCOMO II model.	10	CO4	L2

Internal Assessment Test 2 – November 2025

Sub: Software Engineering and Project Management
Sub Code: BCS501 Branch: AIML/CSE-AIML
Date: 02/12/2025 Duration: 90 mins Max Marks: 50 Sem/Sec: V – A/B/C

Answer any FIVE FULL Questions

1a Analyze the software engineering principles that guide any one framework activity.

Design Modeling Principles

Principle 1. Design should be traceable to the requirements model.

The requirements model describes the information domain of the problem, user-visible functions, system behavior, and a set of requirements classes that package business objects with the methods that service them. The design model translates this information into an architecture, a set of subsystems that implement major functions, and a set of components that are the realization of requirements classes.The elements of the design model should be traceable to the requirements model.

Principle 2. Always consider the architecture of the system to be built.

Software architecture is the skeleton of the system to be built. It affects interfaces, data structures, program control flow and behavior, the manner in which testing can be conducted, the maintainability of the resultant system, and much more. For all of these reasons, design should start with architectural considerations. Only after the architecture has been established should component-level issues be considered.

Principle 3. Design of data is as important as design of processing functions.

Data design is an essential element of architectural design. The manner in which data objects are realized within the design cannot be left to chance. A well- structured data design helps to simplify program flow, makes the design and implementation of software components easier, and makes overall processing more efficient.

Principle 4. Interfaces (both internal and external) must be designed with care.

The manner in which data flows between the components of a system has much to do with processing efficiency, error propagation, and design simplicity. A well- designed interface makes integration easier and assists the tester in validating component functions.

Principle 5. User interface design should be tuned to the needs of the end user.

However, in every case, it should stress ease of use. The user interface is the visible manifestation of the software. No matter how sophisticated its internal functions, no matter how comprehensive its data structures, no matter how well designed its architecture, a poor interface design often leads to the perception that the software is "bad."

Principle 6. Component-level design should be functionally independent.

Functional independence is a measure of the "single-mindedness" of a software component. The functionality that is delivered by a component should be cohesive—that is, it should focus on one and only one function or subfunction.

Principle 7. Components should be loosely coupled to one another and to the external environment.

Coupling is achieved in many ways— via a component interface, by messaging, through global data. As the level of coupling increases, the likelihood of error propagation also increases and the overall maintainability of the software decreases. Therefore,component coupling should be kept as low as is reasonable.

Principle 8. Design representations (models) should be easily understandable.

The purpose of design is to communicate information to practitioners who will generate code, to those who will test the software, and to others who may maintain the software in the future. If the design is difficult to understand, it will not serve as an effective communication medium.

Principle 9. The design should be developed iteratively

. With each iteration, the designer should strive for greater simplicity. Like almost all creative activities, design occurs iteratively. The first iterations work to refine the design and
correct errors,

1 b. Differentiate the components of SMART objectives and examine how each element influences the success of a project plan.

ANS. The mnemonic SMART is sometimes used to describe well defined objectives:

Specific: Effective objectives are concrete and well defined. Vague aspirations such as 'to improve customer relations' are unsatisfactory. Objectives should be defined in such a way that it is obvious to all whether the project has been successful or not.

Measurable: Ideally there should be measures of effectiveness which tell us how successful the project has been. For example, 'to reduce customer complaints' would be more satisfactory as an objective than 'to improve customer relations'. The measure can, in some cases, be an answer to simple yes/no questions, e.g. 'Can we install the new software by 1 November 2011?'

Achievable: It must be within the power of the individual or group to achieve the objective.

Relevant: The objective must be relevant to the true purpose of the project.

Time constrained: There should be a defined point in time by which the objective should have been achieved.

2 Explain the importance of project management in software development.

ANS. **Importance of Project Management in Software Development**

1. **Ensures Clear Goals and Direction**
   Project management helps define the project's scope, objectives, and requirements. This prevents misunderstandings and keeps the entire team aligned on what needs to be built.

2. **Improves Planning and Organization**
   It provides a structured approach for scheduling tasks, allocating resources, estimating timelines, and identifying dependencies. This reduces chaos and increases efficiency.

3. **Manages Risks Effectively**
   Every software project faces risks—technical failures, delays, budget overruns, changing requirements. Project management identifies, analyzes, and mitigates these risks before they become major issues.

4. **Enhances Communication and Collaboration**
   It establishes communication channels among developers, testers, stakeholders, and managers. Clear communication ensures transparency, quicker issue resolution, and smoother teamwork.

5. **Controls Budget and Time**
   Project management monitors progress against plans, ensuring that the project stays within

budget and deadlines. It also enables early detection of deviations so corrective actions can be taken.

6. **Ensures Quality Deliverables**
   Through planning, reviews, testing strategies, and continuous monitoring, project management ensures that the final software meets quality standards and user expectations.

7. **Facilitates Change Management**
   Requirements often change. Project management provides processes to handle changes without derailing the entire project, ensuring adaptability and control.

8. **Supports Continuous Improvement**
   After project completion, project managers conduct evaluations to identify lessons learned. This help

3. Describe the major activities in software project management.

ANS The Feasibility Study:

This is an investigation into whether a prospective project is worth starting that it has
a valid business case. Information is gathered about the requirements of the proposed
application. The probable developmental and operational costs, along with the value of the
benefits of the new system, are estimated. The study could be part of a strategic planning
exercise examining and prioritizing a range of potential software developments.

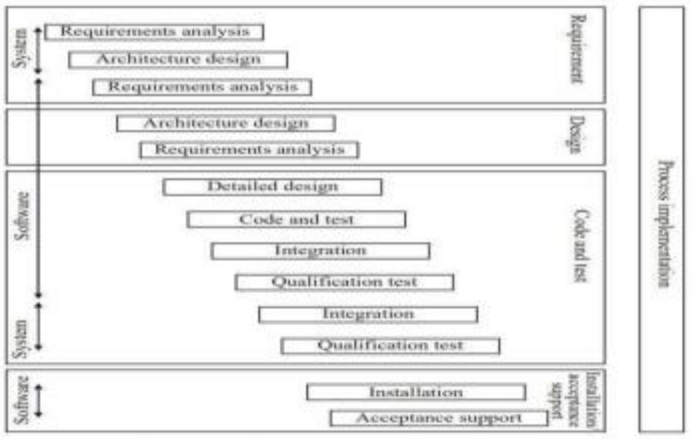
Planning:
If the feasibility study produces results which indicate that the prospective project
appears viable, planning of the project can take place. However, for a large project, we would
not do all our detailed planning right at the beginning. We would formulate an outline plan
for the whole project and a detailed one for the first stage. More detailed planning of the
later stages would be done asthey approached. Thisis because we would havemore detailed
and accurate information upon which to base our plans nearer to the start of the later stages.

Project Execution:

The project can now be executed. The execution of a project often contains design
and implementation subphases. The same is illustrated in Figure 1.2 which shows the
typical sequence of software development activities recommended in the international
standard ISO 12207.

Requirements Analysis:

This starts with requirement elicitation or requirement gathering which establishes
what the users require of the system that the project is to implement. Some work along
these lines will almost certainly have been carried out when the project was evaluated,
but now the original information obtained needs to be updated and supplemented.



.1.1 **Figure 1.2: Sub Phases in Project Execution**

Specification:

Detailed documentation of what the proposed system is to do.

Design:
A design has to be drawn up which meets the specification. This design will be in two stages. One will be the external or user design concerned with the external appearance of the application. The other produces the physical design which tackles the way that the data and software procedures are to be structured internally.
Architecture Design: This maps the requirements to the components of the system that is to be built. At the system level, decisions will need to be made about which processes in the new system will be carried out by the user and which can be computerized. This design of the system architecture thus forms an input to the development of the software requirements. A second architecture design process then takes place which maps the software requirements to software components.
Detailed Design: Each software component is made up of a number ofsoftware units that can be separately coded and tested. The detailed design of these units is carried out separately.

Coding:
This may refer to writing code in a procedural language or an object-oriented language or could refer to the use of an application-builder. Even where software is not being built from scratch, some modification to the base package could be required to Meet the needs of the new application.

Testing (Verification and Validation):
Whether software is developed specially for the current application or not, careful testing will be needed to check that the proposed system meets its requirements.
Integration: The individual components are collected together and tested to see if they meet the overall requirements. Integration could be at the level of software where Different software components are combined, or at the level of the system as a whole where the software and other components of the system such as the hardware platforms and networks and the user procedures are brought together.
Qualification Testing: The system, including the software components, has to be tested carefully to ensure that all the requirements have been fulfilled.

Implementation/Installation:
Some system development practitioners refer to the whole of the project after design as ‘implementation’ (that is, the implementation of the design) while others insist that the term refers to the installation of the system after the software has been developed.

Acceptance Support:
Once the system has been implemented there is a continuing need for the correction of any errors that may have crept into the system and for extensions and improvements to the system. Maintenance and support activities may be seen as a series of minor software projects.

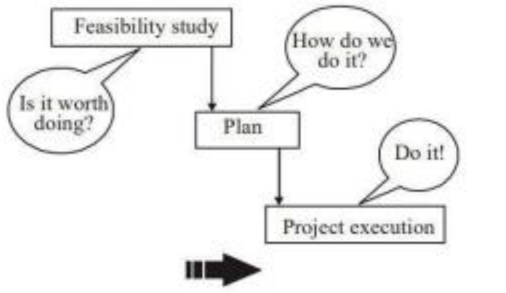


Figure 1.1: The Feasibility Study / Plan / Execution Cycle

4 Apply the phases of the project management life cycle to a suitable project scenario.

ANS

1. Project Initiation: The project initiation phase starts with project concept development. During concept development the different characteristics of the software to be developed are thoroughly understood,which includes,the scopeofthe project,the project constraints, the cost that would be incurred and the benefits that would accrue. Based on this understanding, a feasibility study is undertaken to determine the project would be financially and technically feasible.

Based on feasibility study, the business case is developed. Once the top management agrees to the business case, the project manager is appointed, the project charter is written and finally project team is formed. This sets the ground for the manager to start the project planning phase.
W5HHPrinciple: Barry Boehm,summarized the questionsthat need to be asked and answered in order to have an understanding of these project characteristics.
Why is the software being built?
What will be done?
When will it be done?
Who is responsible for a function?

Where are they organizationally located?
How will the job be done technically and managerially?

How much of these each resource is needed.

2. Project Bidding: Once the top management is convinced by the business case, the project charter is developed. For some categories of projects, it may be necessary to

have formal bidding process to select suitable vendor based on some cost-performance criteria. The different types of bidding techniques are:

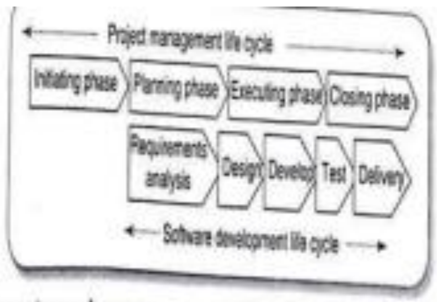
Request for quotation(RFQ) : An organization advertises an RFQ if it has good understanding of the project and the possible solutions.

Request for Proposal(RFP) : An organization had reasonable understanding of the problem to be solved, however, it does not have good grasp of the solution aspects. i.e. may not have sufficient knowledge about different features to be implemented. The purpose of RFP is to get an understanding of the alternative solutions possible that can be deployed and not vendor selection. Based on the RFP process, the requesting organization can form a clear idea of the project solutions required, based on which it can form a statement work (SOW) for requesting RFQ for the vendors.
Request for Information (RFI): An organization soliciting bids may publish an RFI. Based on the vendor response to the RFI, the organization can assess the competencies of the vendors and shortlist the vendors who can bid for the work.

3. Project Planning:

An importance of the project initiation phase is the project charter.
During the project planning the project manger carries out several processes and creates the following documents:
Project plan: This document identifiesthe project the project tasks and a schedule for the project tasks that assigns project resources and time frames to the tasks.
Resource Plan: It lists the resources , manpower and equipment that would be required to execute the project.
Functional Plan: It documentsthe plan for manpower, equipment and other costs.
Quality Plan: Plan of quality targets and control plans are included in this document.
Risk Plan: This document lists the identification of the potential risks, their prioritization and a plan for the actions that would be taken to contain the different risks.

4. Project Execution: In this phase the tasks are executed as per the project plan developed during the planning phase. Quality of the deliverables is ensured through execution of proper processes. Once all the deliverables are produced and accepted by the customer, the project execution phase completes and the project closure phase starts.
5. Project Closure: Project closure involves completing the release of all the required deliverables to the customer along with the necessary documentation. All the Project resources are released and supply agreements with the vendors are terminated and



8 Different phases of project management life cycle and software development life cycle

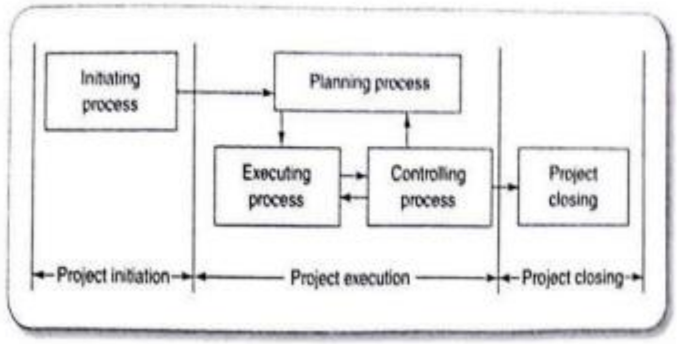


FIGURE 1.5 Principal project management processes

5 Apply function point–based decomposition to estimate the size of a software system.

ANS **Function Point–Based Decomposition**

1. Identify the System's Functional Components

Break the system into the five standard function types:

- **External Inputs (EI)** – data entering the system
- **External Outputs (EO)** – data leaving the system
- **External Inquiries (EQ)** – data retrieval without update
- **Internal Logical Files (ILF)** – data groups maintained within the system
- **External Interface Files (EIF)** – data used by the system but maintained externally

This step ensures you understand what the system *does* from the user's perspective.

2. Determine the Complexity of Each Function

For each identified function:

- Evaluate its **data element types (DETs)** and **file types referenced (FTRs)**.
- Assign a complexity level: **Low**, **Average**, or **High**.

These complexity levels correspond to standard weights defined in Function Point Analysis.

3. Calculate the Unadjusted Function Points (UFP)

- Multiply the **number of functions** in each category by their **assigned weights**.
- Sum the results across all categories.

This produces the **Unadjusted Function Point (UFP)** total, representing the system's logical size.

4. Apply the Value Adjustment Factor (VAF)

Evaluate 14 general system characteristics (such as performance, reusability, security, maintainability).
Each characteristic is rated from **0 to 5** based on its influence on the system.

Compute the VAF using:

VAF = 0.65 + (0.01 × ΣFi)
where **Fi** are the ratings of the 14 factors.

5. Compute the Adjusted Function Points (AFP)

Calculate the final estimate:

Adjusted Function Points = UFP × VAF

This produces the final function point size of the system.

6 Explain the components of the COCOMO II model.

ANS

Components of the COCOMO II Model

COCOMO II (Constructive Cost Model II) is a software cost-estimation model used to predict the effort, time, and cost required to develop software. It consists of several key components:

1. Application Composition Model

Used in early prototyping stages.

- Based on **Object Points** (screens, reports, 3GL components).
- Estimates effort using UI requirements, prototypes, and high-level features.
- Useful when the system is still evolving and size is uncertain.

2. Early Design Model

Used when the basic architecture is defined.

- Uses **Unadjusted Function Points (UFP)** or **Lines of Code (LOC)** as size measures.
- Applies **seven cost drivers** (e.g., reliability, complexity, platform difficulty).
- Provides rough effort estimates early in the design process.

3. Post-Architecture Model

Used when full system architecture and detailed requirements are known.

- The most detailed and commonly used level of COCOMO II.
- Incorporates:
    - **Scale Factors (5 factors) –** determine the exponent in the effort equation and reflect project “economies or diseconomies of scale.”
        - Precedentedness
        - Development flexibility
        - Architecture/risk resolution
        - Team cohesion
        - Process maturity
    - **Effort Multipliers (17 cost drivers) –** adjust effort based on product, platform, personnel, and project attributes.
      Examples:

        - Product complexity
        - Required reliability
        - Personnel capability
        - Platform volatility

4. Effort Estimation Formula

COCOMO II uses the general equation:

Effort = A × (Size)^E × $\Pi(EM_i)$
Where:

- **A** = calibration constant
- **Size** = estimated size in KSLOC or function points
- **E** = exponent derived from scale factors
- **EM_i** = effort multipliers from cost drivers