USN | | | | | | | | | |

Internal Assessment Test 1 Answer scheme & Solutions – October 2024

Sub:	**SOFTWARE ENGINEERING AND PROJECT MANAGEMENT**				Sub Code:	BCS501	Branch:	**AInDS**		
Date:	**14/12/2024**	Duration:	**90 minutes**	Max Marks:	**50**	Sem	**V**		**OBE**	
Answer any FIVE Questions								**MARKS**	**CO**	**RBT**

1		Effective communication is among the most challenging activities that you will confront. Justify this statement by discussing about the principles that apply for communication within a software project	10	CO3	L2

Effective communication is indeed one of the most challenging activities in any context, particularly within a software project. This complexity arises due to diverse factors such as varying perspectives, technical complexities, team dynamics, and the need for precision and clarity. Below are the principles of communication within a software project and how they justify the challenges of effective communication:

1. Clarity and Precision

- Software projects require precise communication to avoid misunderstandings. Ambiguity in requirements, design, or implementation details can lead to errors, rework, and delays.
- For example, if a requirement is vaguely communicated, developers might interpret it differently, leading to mismatched expectations.

2. Audience Awareness

- Communication must be tailored to the audience's technical expertise. Developers, clients, and stakeholders often have different levels of understanding.
- Explaining technical details to a non-technical client demands simplification, while communicating with developers requires in-depth technical accuracy.

3. Timeliness

- Communication must occur at the right time. Delayed communication about changes in requirements or deadlines can disrupt the entire project's progress.
- For example, if a change in a project's scope is not promptly conveyed, developers might work on outdated features.

4. Consistency

- Consistent messaging ensures that all team members are aligned on project goals and progress.
- Inconsistent updates can lead to confusion, conflicting tasks, and inefficiencies.

5. Feedback and Active Listening

- Effective communication is two-way. Teams need to provide and accept feedback actively to address misunderstandings or concerns.
- For instance, during code reviews, developers must articulate feedback constructively, and recipients must understand and act on it.

6. Use of Appropriate Tools

- Software projects often leverage tools for documentation, version control, and task management (e.g., JIRA, Slack, Git). Misuse or lack of familiarity with these tools can hinder communication.
- For example, failing to update a project management tool can lead to team members working on outdated tasks.

7. Cultural and Linguistic Differences

- In global teams, cultural and linguistic differences can lead to misinterpretations.
- For example, indirect communication styles in one culture might be misinterpreted as a lack of clarity by another.

8. **Conflict Resolution**

- Software projects can involve disagreements over approaches or priorities. Resolving such conflicts requires diplomatic communication.
- A poorly handled conflict can escalate tensions, while effective communication can turn it into a constructive discussion.

2		Briefly Explain the deployment design modeling principles. **1. Scalability** • The design should support scaling the system to handle increased loads, whether by adding resources (vertical scaling) or distributing the load across multiple nodes (horizontal scaling). **2. Performance Optimization** • Deployment should be planned to minimize latency and maximize throughput. This includes optimizing resource allocation, such as CPU, memory, and storage, and considering proximity to end-users. **3. Reliability and Availability** • The system must be designed to ensure high availability through redundancy, failover mechanisms, and clustering. Reliable deployment minimizes downtime and maintains service continuity. **4. Security** • Security measures such as firewalls, encryption, and secure access controls should be incorporated to protect data and applications from vulnerabilities and attacks. **5. Modularity** • Components should be deployed in a modular manner, allowing independent updates, maintenance, and scaling of specific parts of the system without impacting others. **6. Maintainability** • The design should facilitate easy updates, monitoring, and debugging. Use of containers and orchestration tools like Docker and Kubernetes can enhance maintainability. **7. Cost-Effectiveness** • The deployment model should balance performance and cost. This involves selecting appropriate infrastructure (on-premise, cloud, or hybrid) and optimizing resource usage. **8. Interoperability** • The deployment model should ensure compatibility and seamless communication between different components and systems. **9. Compliance and Standards** • Adherence to regulatory requirements, industry standards, and organizational policies must be ensured in the deployment design. **10. Documentation** • Clear documentation of the deployment model is crucial for team understanding, onboarding, troubleshooting, and future modifications.	10	CO3	L1

3		Explain the activities of management in doing management control. 1. Setting Objectives and Standards • Define clear, measurable objectives that align with the organization's strategic goals. • Establish performance standards or benchmarks to measure progress and success. 2. **Planning** • Develop detailed plans that outline how the objectives will be achieved. • Allocate resources (financial, human, technological) required to implement the plans effectively. **3. Monitoring and Measuring Performance** • Continuously track the progress of activities against the defined standards. • Use tools like key performance indicators (KPIs), dashboards, and reports to gather performance data. **4. Evaluating Performance** • Compare actual results with planned objectives and performance standards. • Identify variances and analyze their causes to understand areas of improvement. **5. Taking Corrective Action** • Implement corrective measures to address deviations from the plan. • This may involve reallocating resources, adjusting strategies, or revising goals if necessary. **6. Communicating and Reporting** • Ensure regular communication of performance status to stakeholders, including team members and upper management. • Provide transparent and accurate reports to facilitate informed decision-making. **7. Motivating and Guiding Employees** • Encourage employees to meet performance standards through motivation, training, and support. • Provide feedback and recognition to maintain morale and productivity. **8. Ensuring Accountability** • Assign clear responsibilities and accountabilities to team members or departments. • Establish mechanisms for regular reviews and audits to ensure compliance with standards. **9. Adapting to Changes** • Adjust management control processes in response to changes in the internal or external environment, such as market trends, technology advancements, or organizational restructuring. **10. Risk Management** • Identify potential risks that may affect the achievement of objectives. • Develop and implement strategies to mitigate these risks.	10	CO	L1

4		Explain traditional v/s Modern Project management Practices	10	CO4	L2

Aspect	Traditional Project Management	Modern Project Management
Approach	Linear and sequential (Waterfall model).	Iterative, flexible, and adaptive (Agile, Scrum, Kanban).
Planning	Detailed upfront planning with fixed scope, timelines, and budget.	Incremental planning, often revisited and adjusted throughout the project.
Focus	Deliverables and milestones.	Value delivery, customer satisfaction, and continuous improvement.
Project Structure	Rigid hierarchies and defined roles.	Collaborative teams with cross-functional roles.
Documentation	Extensive and detailed documentation is critical.	Minimal documentation, focusing on what is essential for execution.
Change Management	Resistant to change; changes require formal processes.	Embraces change as a way to improve and adapt to evolving needs.
Team Dynamics	Authority-driven, manager-centric decision-making.	Empowered teams with shared decision-making responsibilities.
Technology Usage	Limited reliance on tools; manual tracking and reporting.	Heavy use of project management software and automation tools (e.g., Jira, Trello).
Risk Management	Risk is analyzed and mitigated primarily at the start of the project.	Continuous risk identification and management throughout the project.
Customer Involvement	Customers are involved mainly during requirement gathering and delivery.	Continuous customer involvement and feedback throughout the project lifecycle.
Delivery Style	One-time delivery at the end of the project.	Frequent, smaller deliveries or increments of functionality.
Performance Measurement	Based on adherence to schedule, budget, and scope.	Based on value delivered, team performance, and customer satisfaction.
Tools and Techniques	Gantt charts, Work Breakdown Structures (WBS), Critical Path Method (CPM).	Agile boards, burndown charts, continuous integration/continuous delivery (CI/CD).
Flexibility	Low flexibility, making it difficult to adapt to changes.	High flexibility, allowing rapid response to changing priorities.
Industries Suited	Construction, manufacturing, or industries with fixed requirements.	Software development, IT, or industries with dynamic requirements.

5		Why do we need software quality models? Explain Garvin's quality dimension Explain McCall's Model	10	CO5	L2

Software quality models provide a structured framework to evaluate, manage, and improve software quality. These models help stakeholders understand the critical aspects of quality, establish measurable criteria, and ensure the software meets user expectations and industry standards.

Key Reasons for Software Quality Models:

1. **Objective Assessment:** Define measurable attributes for evaluating software quality.
2. **Improved Development:** Identify areas of improvement during development and maintenance.
3. **Stakeholder Communication:** Bridge gaps between developers, clients, and users regarding quality expectations.
4. **Consistency:** Ensure consistent quality standards across projects.
5. **Risk Mitigation:** Identify and address potential quality risks early in the lifecycle.

Garvin's Quality Dimensions

David Garvin introduced eight dimensions of quality, providing a comprehensive view of quality in products, including software.

1. **Performance:** How well the software performs its intended function.
   Example: Response time of a search query in an application.
2. **Features:** Additional functionalities that enhance the core operations.
   Example: Auto-save in a text editor.
3. **Reliability:** The likelihood of the software functioning without failure over a specified period.
   Example: A server maintaining uptime for 99.9% of the time.
4. **Conformance:** The degree to which the software adheres to specifications and standards.
   Example: Compliance with ISO or industry-specific standards.
5. **Durability:** The software's ability to withstand changes and maintain performance over time.
   Example: Legacy systems that still operate efficiently after decades.
6. **Serviceability:** The ease of maintaining and repairing the software.
   Example: Providing timely patches and updates.
7. **Aesthetics:** The user experience, including interface design and intuitiveness.
   Example: Clean and user-friendly UI design.
8. **Perceived Quality:** The user's subjective judgment of quality, often influenced by branding or reputation.
   Example: Trust in software from well-known developers.

McCall's Model

McCall's model, one of the earliest software quality models, focuses on defining quality attributes and their impact on software from the developer's and user's perspectives. It categorizes quality into three main categories:

1. **Product Operation:** Attributes affecting the software's functionality during use.
   - **Correctness:** Adherence to specified requirements.
   - **Reliability:** Continuity of service under specified conditions.
   - **Efficiency:** Optimal use of resources like memory, CPU, and bandwidth.
   - **Integrity:** Protection against unauthorized access and data corruption.
   - **Usability:** Ease of learning and operating the software.
2. **Product Revision:** Attributes affecting software maintenance and updates.
   - **Maintainability:** Ease of fixing defects or making enhancements.
   - **Flexibility:** Adaptability to changing requirements.
   - **Testability:** Ease of testing the software to ensure quality.
3. **Product Transition:** Attributes affecting software portability and adaptability to new environments.
   - **Portability:** Ability to function across different platforms.
   - **Reusability:** Use of software components in other applications.
   - **Interoperability:** Ability to integrate with other systems.

Representation:

6		Briefly explain the cocomo II model	10	CO5	L2

The **COCOMO II (Constructive Cost Model II)** is a comprehensive framework used to estimate the cost, effort, and schedule required for software development projects. It is an updated version of the original COCOMO model, designed to address modern software development practices and technologies.

Key Features of COCOMO II:

1. **Three Submodels**:
    - **Application Composition Model**: Used for projects involving rapid application development (RAD) or prototyping.
    - **Early Design Model**: Provides rough estimates based on high-level system characteristics when detailed information is unavailable.
    - **Post-Architecture Model**: Provides more accurate estimates during the detailed design and coding phases when more information is available.
2. **Effort Estimation**: The model predicts the effort (in person-months) using the formula:

    $$\text{Effort} = A \times \text{(Size)}^E \times \prod \text{(EM)}$$

    - **A**: Constant scaling factor
    - **Size**: Estimated size of the software (usually in KSLOC – thousands of source lines of code)
    - **E**: Exponent derived from project scale factors
    - **EM**: Effort multipliers based on project-specific attributes (e.g., team experience, tools, etc.)
3. **Cost Drivers**: The model considers various cost drivers grouped into categories like product, hardware, personnel, and project factors, each influencing the effort.
4. **Scalability**: COCOMO II adjusts for project size, complexity, and development methodology, making it suitable for small to large projects.
5. **Flexibility**: It supports different stages of software development, allowing estimates to evolve as project details become clearer.

Advantages:

- Tailored to modern software engineering practices.
- Provides more detailed and customizable estimation compared to its predecessor.
- Supports iterative and incremental development methodologies.

Limitations:

- Requires accurate inputs and historical data for reliable estimates.
- Complex to implement compared to simpler estimation models.