

Model Question Paper-1 with effect from 2021(CBCS Scheme)

US
N

--	--	--	--	--	--	--	--	--	--

**Sixth Semester B.E. Degree Examination SOFTWARE
ENGINEERING & PROJECT MANAGEMENT
21CS61**

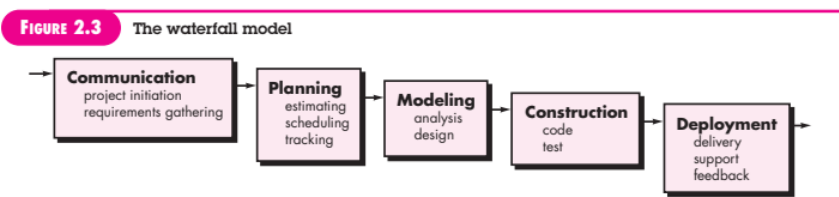
**TIME: 03 Hours
100**

SET - 1

Max. Marks:

Note: Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE. THESE ANSWERS FROM TEXTBOOK**

Module -1 DOWNLOAD			Bloom 's Taxonomy Level	COs	Marks
Q.01	a	<p>Define software engineering and explain its process.</p> <p>Software Engineering and Its Process</p> <p>Definition of Software Engineering Software engineering is a framework for the activities, actions, and tasks required to build high-quality software. It encompasses technologies, technical methods, automated tools, and the creative adaptation by knowledgeable individuals to tailor mature processes to the needs of the products and market demands.</p> <p>Process of Software Engineering Generic Process Model</p> <ul style="list-style-type: none"> - A software process is a collection of work activities, actions, and tasks within a framework that defines their relationships. - The software process involves five framework activities: communication, planning, modeling, construction, and deployment. - Umbrella activities like project tracking, risk management, quality assurance, configuration management, and technical reviews are applied throughout the process. - Process flow describes how activities are organized with respect to sequence and time, such as linear, iterative, evolutionary, or parallel flows. <p>Actions and Tasks in Software Engineering</p>	L1	1	10

	<ul style="list-style-type: none"> - Each software engineering action is defined by a task set that includes work tasks, work products, quality assurance points, and milestones. - The software process is represented by a schematic framework where each framework activity is populated by software engineering actions. <p>Importance of Software Engineering</p> <ul style="list-style-type: none"> - Software engineering is a layered technology resting on an organizational commitment to quality. - It serves as the foundation for developing computer software, enabling rational and timely development through a focus on process improvement. - It encompasses a process, methods for managing and engineering software, and various tools for effective software development. <p>Conclusion</p> <p>Software engineering is a structured approach that guides the development, operation, and maintenance of software through defined processes, activities, and tasks, involving a combination of technology, methodologies, and human expertise.</p>			
b	<p>With a neat diagram explain Prescriptive and Waterfall model.</p> <p>Prescriptive and Waterfall Model Explanation</p>  <p>FIGURE 2.3 The waterfall model</p> <pre> graph LR A[Communication
project initiation
requirements gathering] --> B[Planning
estimating
scheduling
tracking] B --> C[Modeling
analysis
design] C --> D[Construction
code
test] D --> E[Deployment
delivery
support
feedback] </pre> <p><small>6 Although the original waterfall model proposed by Winston Royce [Roy70] made provision for "feedback loops," the vast majority of organizations that apply this process model treat it as if it were strictly linear.</small></p> <p>Prescriptive Process Models</p> <ul style="list-style-type: none"> - Prescriptive process models prescribe a set of process elements for each project, including framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms. - They focus on detailed definition, identification, and application of process activities and tasks to improve system quality, manage projects effectively, predict delivery dates and costs, and guide software engineering 	L2	1	10

teams.

- If applied dogmatically without adaptation, prescriptive models can increase bureaucracy and create difficulties for stakeholders.
- Prescriptive models stress order and project consistency as dominant issues, defining a process flow for interrelated process elements.

Waterfall Model

- The waterfall model is the oldest paradigm for software engineering, suggesting a systematic, sequential approach to software development.
- It follows a linear progression from customer requirements specification through planning, modeling, construction, deployment, and ongoing support of the completed software.
- It can cause issues due to the sequential flow not aligning with real project progress, difficulty in explicitly stating all requirements upfront, and the lengthy time span until a working program is available.
- Projects rarely follow the strict sequential flow proposed by the model, making it challenging to accommodate changes and uncertainties.
- The waterfall model is more suitable for situations where requirements are fixed and work can proceed linearly, such as in well-defined adaptations or enhancements to existing systems.

Diagram Explanation

- The waterfall model diagram typically depicts a flow from project initiation through requirements gathering, planning, modeling, construction, deployment, and support in a linear fashion.
- It shows a sequential progression of software development stages where each phase is completed before moving to the next.

Key Contrasts

- Prescriptive models emphasize detailed process elements and order, while the waterfall model follows a sequential, linear approach to software development.
- Prescriptive models focus on project consistency and system quality improvement, while the waterfall model may face challenges with changing requirements and project uncertainties.

Additional Insights

- The V-model is a variation of the waterfall model that

		illustrates the relationship of quality assurance actions with earlier engineering work. - Incremental process models are suitable when initial software requirements are well defined, but a purely linear process is not feasible due to the development effort's overall scope.			
OR					
Q.02	a	<p>Explain Software Myths with examples.</p> <p>Understanding Software Myths with Examples</p> <p>Software Myths Explained</p> <p>1. Myth : Software requirements continually change, but change can be easily accommodated because software is flexible.</p> <ul style="list-style-type: none"> - Reality : Early requirement changes have a small cost impact, but as time passes, the impact grows rapidly due to committed resources and established design frameworks. <p>2. Myth : Once a program is written and functional, the job is done.</p> <ul style="list-style-type: none"> - Reality : Significant effort is expended post-delivery, indicating that the job extends beyond initial development. <p>3. Myth : Outsourcing a software project allows relaxation during development.</p> <ul style="list-style-type: none"> - Reality : Lack of internal project management understanding can lead to struggles when outsourcing software projects. <p>4. Myth : A general statement of objectives is sufficient to start writing programs.</p> <ul style="list-style-type: none"> - Reality : Ambiguous objectives can lead to disaster, emphasizing the need for clear and stable requirements. <p>5. Myth : Adding more programmers can help catch up if behind schedule.</p> <ul style="list-style-type: none"> - Reality : Software development is not mechanistic; adding people late can further delay the project. <p>6. Myth : Having a book of standards and procedures is sufficient for software development.</p> <ul style="list-style-type: none"> - Reality : Mere existence of standards may not ensure their effectiveness or use by practitioners. 	L2	1	10

7. Myth : Software engineering involves excessive documentation and slows down the process.
 - Reality : Software engineering focuses on quality product creation, reducing rework and improving delivery times.

8. Myth : Assessing software quality is only possible once the program is running.
 - Reality : Technical reviews from project inception provide effective quality assurance mechanisms.

Conclusion

Software myths often stem from long-standing practices and misconceptions in the industry. Understanding the realities behind these myths is crucial for effective software development and project management.

b

With a neat diagram explain Incremental process models and Evolutionary process models.

L2

1

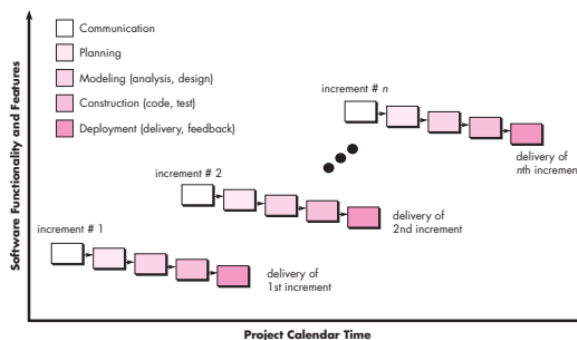
10

42

PART ONE THE SOFTWARE PROCESS

FIGURE 2.5

The incremental model



Refer diagram for Evolutionary process model

Incremental Process Model

The incremental process model focuses on delivering an operational product in increments. Early increments are simplified versions of the final product that provide basic functionality for user evaluation and serve as a platform for future development. It is particularly useful when staffing is limited, allowing for gradual implementation based on user feedback and technical considerations. The process involves iterative cycles of planning, modification, and delivery of new features until the complete product is achieved.

Key Points:

- Each increment delivers additional features and functionality.
- Early increments serve as a basis for user evaluation and feedback.
- Allows for gradual implementation based on staffing availability.
- Helps manage technical risks and adapt to evolving requirements.

Evolutionary Process Model

Evolutionary process models are designed to accommodate software evolution over time. These models enable the development of increasingly complete versions of the software through iterative cycles of refinement and expansion. They address the need for software to evolve in response to changing business and product requirements, tight market deadlines, and the necessity for quick delivery of software functionality to users.

Key Points:

- Software evolves over time to adapt to changing requirements.
- Iterative development approach to produce more complete software versions.
- Designed to meet tight market deadlines and evolving customer needs.
- Allows for the introduction of limited versions of software quickly and refinement over time.

Diagram Explanation

The incremental model delivers a series of releases, called increments, providing progressively more functionality for the customer with each iteration. Early increments serve as core products, addressing basic requirements while leaving room for additional features in subsequent increments. The evolutionary model, on the other hand, focuses on evolving the software over time to meet changing requirements and market demands through iterative cycles of refinement and expansion.

Diagram Components:

1. Incremental Model:
 - Delivery of increments with increasing functionality.
 - Core product addressed in early increments.
 - User evaluation and feedback incorporated.

		<p>2. Evolutionary Model:</p> <ul style="list-style-type: none"> - Iterative development for evolving software. - Adaptation to changing business and product requirements. - Incremental refinement and expansion over time. 			
<p align="center">Module-2 <u>DOWNLOAD</u></p>					
Q. 03	a	<p>What are the nature of Software system and explain its characteristics</p> <p>. Characteristics of Software Systems</p> <p>Nature of Software Systems</p> <ul style="list-style-type: none"> - Software systems involve various modeling elements that provide different viewpoints on requirements, including class models, flow models, scenario-based models, and behavioral models. - These elements help depict how users interact with the system, the objects manipulated, system behavior, and information transformation. - Analysis modeling leads to the derivation of these modeling elements, with each element presenting the problem from a unique perspective. - The software team must work to keep the modeling simple by including only elements that add value to the model. <p>Characteristics of Software Systems</p> <ol style="list-style-type: none"> 1. User -Centric Design : Software systems focus on user satisfaction by understanding how end users want to interact with the system. 2. Flexibility : Software systems need to adapt to changing requirements throughout the project. 3. Requirement Variability : The nature of software requirements varies with each project , from simple user scenarios to detailed documents containing models and descriptions. 4. Modeling Elements : Software systems encompass different modeling elements like class-based, scenario-based, behavioral, and flow-oriented elements to represent various aspects of the system. 5. Stakeholder Engagement : Involving stakeholders in the modeling process, such as having them write use cases , enhances the understanding of software usage. 6. Effective Communication : Models are constructed to highlight critical features of the system and de-emphasize less essential aspects, aiding in effective communication among project stakeholders. 	L2	2	10

		<p>7. Iterative Development : Software systems often undergo iterative development cycles to refine requirements and design based on feedback and validation.</p> <p>8. Continuous Improvement : Software systems evolve over time, requiring continuous improvement to meet changing user needs and technological advancements.</p>			
	b	<p>How Requirement engineering it works explain in detail.</p> <p>Requirement Engineering: Detailed Explanation</p> <p>What is Requirements Engineering?</p> <p>Requirements engineering is a major software engineering action that involves a broad spectrum of tasks and techniques aimed at understanding and defining the needs of a software project. It is crucial for ensuring that the software developed meets the expectations and requirements of stakeholders.</p> <p>Tasks Involved in Requirements Engineering</p> <ol style="list-style-type: none"> 1. Inception: Defining the scope and nature of the problem to be solved. 2. Elicitation: Helping stakeholders define what is required. 3. Elaboration: Refining and modifying basic requirements. 4. Negotiation: Determining priorities and essential elements. 5. Specification: Clearly defining the problem and solution. 6. Validation: Ensuring the understanding of the problem aligns with stakeholders' understanding. 7. Management: Handling requirements as they are transformed into an operational system. <p>Importance of Requirements Engineering</p> <ul style="list-style-type: none"> - Understanding what the customer wants before designing and building a computer-based system is crucial. - It provides a mechanism to analyze needs, assess feasibility, specify solutions unambiguously, validate specifications, and manage requirements effectively. <p>Key Considerations</p> <ul style="list-style-type: none"> - Requirements engineering begins with defining the problem scope and progresses through various stages of refinement and validation. - The process involves collaboration among system 	L2	2	10

		<p>engineers, stakeholders, and project team members to ensure a clear understanding of project needs.</p> <ul style="list-style-type: none"> - Effective requirements engineering mitigates the risk of project failure by aligning software development with stakeholder expectations. <p>Validation Checklist</p> <ul style="list-style-type: none"> - Are requirements stated clearly and unambiguously? - Is the source of each requirement identified and verified? - Are requirements testable, traceable, and structured for easy understanding? - Is consistency maintained among different requirements? - Are performance, behavior, and operational characteristics clearly specified? <p>Challenges and Solutions</p> <ul style="list-style-type: none"> - Many software developers overlook requirements engineering, leading to failed projects. - Challenges arise when eliciting requirements from multiple customers, requiring effective communication and coordination. - The requirements model represents a snapshot of the system at a specific time, emphasizing the dynamic nature of software development. 			
		OR			
Q.04	a	<p>Explain three types of QFD with examples.</p> <p>Three Types of QFD with Examples</p> <p>Normal Requirements</p> <ul style="list-style-type: none"> - Normal requirements are objectives and goals stated for a product or system during meetings with the customer. - These requirements lead to customer satisfaction when met. - Examples include requested types of graphical displays, specific system functions, and defined levels of performance. <p>Expected Requirements</p> <ul style="list-style-type: none"> - Expected requirements are implicit to the product or system and are fundamental for customer satisfaction. - Their absence can cause significant dissatisfaction. - Examples of expected requirements are ease of human/machine interaction, operational correctness and reliability, and ease of software installation. <p>Exciting Requirements</p>	L2	2	10

- Exciting requirements go beyond customer expectations and provide high satisfaction.
- They can lead to breakthrough products and delight users.
- For instance, software for a new mobile phone with unexpected capabilities like multitouch screen and visual voicemail.

b

Explain scenario based model with example.

CHAPTER 6 REQUIREMENTS MODELING: SCENARIOS, INFORMATION, AND ANALYSIS CLASSES 181

FIGURE 6.13
Multiplicity

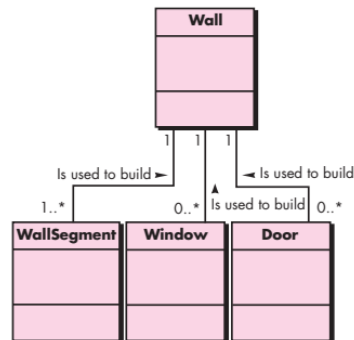
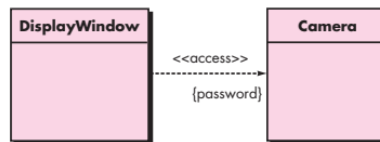


FIGURE 6.14
Dependencies



Scenario-Based Model Explanation with Example

Definition of Scenario-Based Model

A scenario-based model involves creating scenarios or use cases to describe how a system will be utilized. These scenarios help in understanding the interactions and functionalities of the system from the end user's perspective. Scenarios can be informal descriptions of how individuals interact with the system, providing a narrative of specific tasks or interactions.

Example Scenario-Based Model Discussion

In a meeting discussing the SafeHome system, a usage scenario for accessing the home security function was developed. The scenario involved a marketing person describing accessing the system to let someone into the house when away. This informal scenario highlighted the steps involved, such as verifying identity and performing security functions like arming or disarming the system. The scenario captures a typical user interaction with the home security feature, showcasing the practical application of a

L1

2

10

scenario-based model in system development.

Key Components of Scenario-Based Models

1. Use Cases (Scenarios): Descriptions of how the system will be used, often depicted as narratives or outlines of interactions.
2. Primary and Secondary Scenarios: Primary scenarios outline main interactions, while secondary scenarios represent alternative or error conditions.
3. Actor Actions and System Responses: Detailing user actions and system behaviors at each step of the scenario.
4. Exception Handling: Identifying failure conditions or alternative user choices that lead to different system behaviors.
5. Narrative and Interaction Outlines : Different forms of presenting scenarios , such as narrative text, diagrams, or task outlines.

Benefits of Scenario-Based Modeling

1. User-Centered Design: Focuses on understanding user interactions and needs.
2. Visualizing System Functionality: Helps stakeholders visualize and comprehend system operations.
3. Error Identification: Facilitates the identification of potential errors and alternative user paths.
4. Enhanced Communication: Provides a structured way to communicate system functionalities to diverse stakeholders.

Example Use Case Scenario (SafeHome Home Management Function)

- Use Case: SafeHome home management function.
- Narrative: Control electronic devices via a PC or Internet connection, including lights, appliances, and HVAC settings.
- Features : Select devices from a house floor plan, control audiovisual devices, set different house modes (home, away, overnight travel).

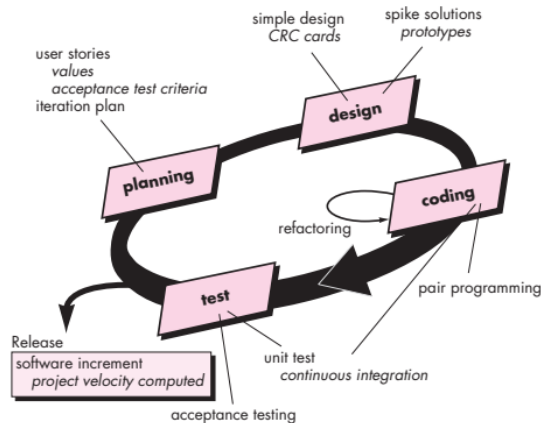
Conclusion

Scenario-based modeling offers a practical approach to understanding system functionalities through user-centric scenarios. By detailing user interactions, system responses, and potential exceptions, scenario-based models aid in the effective development and communication of system requirements.

Module-3 DOWNLOAD					
Q. 05	a	<p>Define Agile Process and explain agility principle.</p> <p>Define Agile Process and Explain Agility Principle</p> <p>Agile Process Definition: An agile process in software engineering refers to a methodology that emphasizes flexibility, adaptability, collaboration, and iterative development. It prioritizes customer satisfaction, responding to change, delivering working software frequently, and fostering effective team communication.</p> <p>Agility Principle Explanation: The agility principles guide software development processes towards agility. One key agility principle is to "Be agile," emphasizing simplicity, economy of action, and local decision-making. This principle underscores the importance of maintaining an agile mindset regardless of the chosen process model. Agility also involves focusing on quality, being ready to adapt to constraints, and building effective teams. These principles help teams navigate the dynamic nature of software development and encourage continuous improvement and customer satisfaction.</p> <p>Principles That Guide Agility:</p> <ol style="list-style-type: none"> 1. Be agile by emphasizing simplicity and local decision-making. 2. Focus on quality at every step of the process. 3. Be ready to adapt to constraints and changes. 4. Build an effective team to foster collaboration and productivity. 5. Continuously strive for technical excellence and good design. 6. Maximize work efficiency by simplifying processes. 7. Encourage self-organizing teams to drive innovation. 8. Regularly reflect on performance to enhance effectiveness and make necessary adjustments. 	L2	3	10
	b	With a Neat Diagram explain Extreme Programming (XP).	L2	3	10

FIGURE 3.2

The Extreme Programming process



Overview of Extreme Programming (XP)

Explanation with Diagram

Extreme Programming (XP) is a software development methodology that emphasizes flexibility, adaptability, and customer satisfaction through iterative development cycles. Here is an explanation of XP with a neat diagram:

Key Principles of Extreme Programming:

1. **Customer Involvement** : Customers are actively involved throughout the development process to ensure alignment with their needs.
2. **Iterative Development**: Software is developed and delivered in short iterations, allowing for frequent feedback and adjustments.
3. **Pair Programming**: Two programmers work together at one workstation, enhancing code quality and knowledge sharing.
4. **Test-Driven Development** : Tests are written before code implementation to ensure functionality and facilitate continuous testing.
5. **Continuous Integration**: Code changes are integrated and tested frequently to detect issues early.
6. **Sustainable Pace**: Encourages a sustainable work pace to maintain productivity and quality over time.
7. **Simple Design**: Promotes simplicity in design to avoid unnecessary complexity and facilitate easier maintenance.
8. **Refactoring**: Code is continuously improved without changing its external behavior to enhance readability and maintainability.

Diagram of Extreme Programming (XP): !
 XP Diagram](insert_diagram_here)

In the diagram, you can visualize the iterative nature of XP, starting from customer involvement and requirements gathering, moving through pair programming and test-driven development, and culminating in continuous integration and refactoring for sustainable software development.

OR

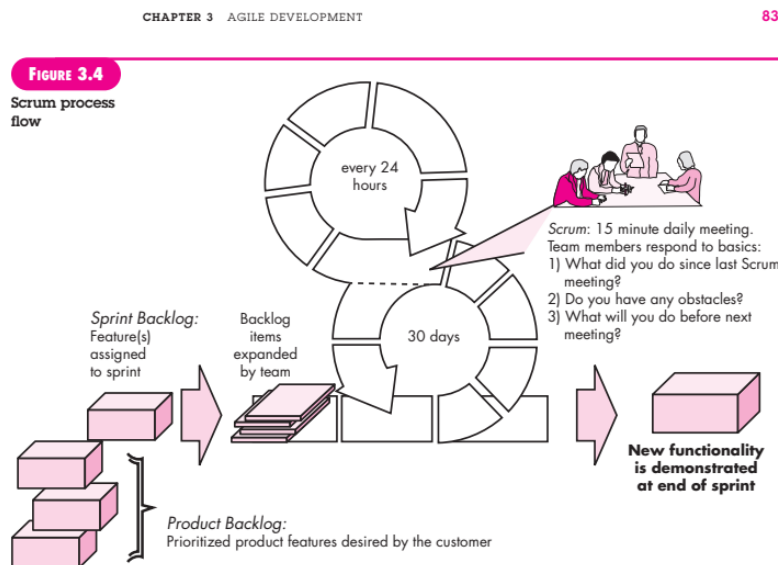
Q. 06 a

Write a short note on Scrum and Crystal.

L2

3

10



Scrum and Crystal Overview

Scrum

- Definition : Scrum is an agile software development method emphasizing project priorities, compartmentalized work units, communication, and frequent customer feedback.
- Origin : Conceived by Jeff Sutherland and his team in the early 1990s.
- Activities : Guides development through requirements, analysis, design, evolution, and delivery within a structured framework.
- Principles : Consistent with the agile manifesto , promoting agility and adaptability in the development process.
- Process Patterns : Utilizes software process patterns for effective project management, including Backlog and Sprints.
- Collaboration : Emphasizes collaboration within self-organizing teams for increased project success.

		<p>Crystal</p> <ul style="list-style-type: none"> - Definition : Crystal is a family of agile process models tailored to project-specific characteristics. - Creators : Developed by Alistair Cockburn and Jim Highsmith to prioritize maneuverability in software development. - Methodologies : Offers a set of methodologies with core elements common to all and unique characteristics tailored to each project. - Adaptability : Agile teams can select the most suitable Crystal family member for their specific project and environment. - Effectiveness : Proven to be effective for different types of projects , focusing on delivering useful , working software. 			
	b	<p>Explain Communication Practice principle</p> <p>. Explain Communication Practice Principle</p> <p>Principle Overview</p> <p>Communication principles in software engineering focus on facilitating effective communication between developers, customers, and stakeholders throughout the project lifecycle. These principles aim to reduce noise, improve bandwidth, and enhance collaboration to ensure clear understanding and successful outcomes.</p> <p>Core Communication Principles:</p> <ol style="list-style-type: none"> 1. Listen : Focus on the speaker 's words , ask for clarification when needed, and avoid interruptions or contentious behavior during conversations. 2. Prepare before you communicate : Understand the problem, do necessary research, and have an agenda for meetings to ensure productive discussions. 3. Facilitate the activity : Assign a leader (facilitator) to guide discussions, mediate conflicts, and ensure adherence to communication principles. 4. Use face-to-face communication with supporting materials : Face-to-face interactions work best when supplemented with visual aids or documents to aid discussions. 5. Take notes and document decisions : Assign a recorder to document important points and decisions made during communication sessions. 6. Strive for collaboration : Utilize collective knowledge to describe product or system functions, fostering trust and a 	L2	3	10

common goal among team members.

7. Stay focused and modularize discussions : Keep the conversation focused on one topic at a time to enhance clarity and resolution.

8. Negotiate for mutual benefit : Approach negotiations with the goal of ensuring that both parties win, promoting collaboration and compromise.

Additional Details:

- Effective communication principles are crucial for successful software engineering projects.
- Communication principles apply to interactions with customers, stakeholders, technical peers, and project managers.
- These principles aim to establish clear channels of communication, foster understanding, and promote collaboration for project success.

Module-4
DOWNLOAD

Q. 07 a

What is software project management explain project management life cycle.

Software Project Management and Project Management Life Cycle

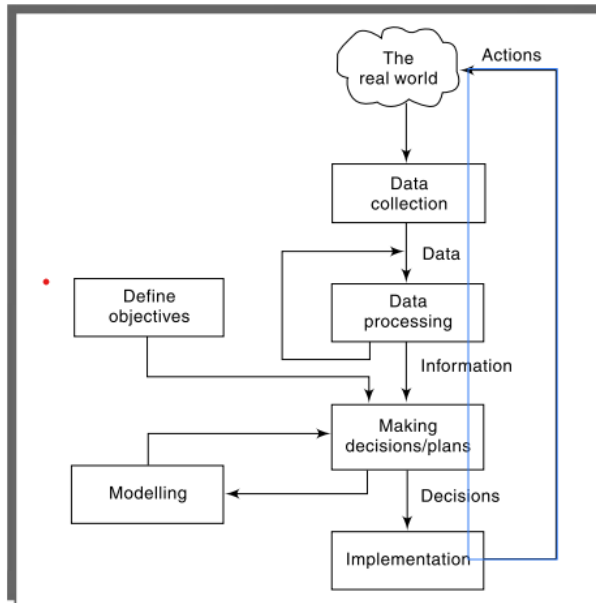


FIGURE 1.5 The project control cycle

Definition of Software Project Management

- Software project management involves planning, monitoring, and controlling software projects to meet objectives.
- It focuses on satisfying real needs by identifying stakeholders and their objectives.
- The primary goal is to ensure that project objectives are met through careful planning, monitoring, and control.

Importance of Software Project Management

- Essential for students of software engineering, computer science, and business information systems.
- Vital for understanding project management principles and practices in software development.
- Provides a structured approach to managing software projects efficiently and effectively.

Project Management Life Cycle Stages

1. Feasibility Study: Assessing the project's viability and business case.
2. Planning: Involves estimating project attributes like cost, duration, and effort.
3. Staffing: Organizing staff and resource plans.

L2

4

10

	<p>4. Risk Management: Identify, analyze, and plan for project risks.</p> <p>5. Miscellaneous Plans: Developing quality assurance and configuration management plans.</p> <p>Project Monitoring and Control</p> <ul style="list-style-type: none"> - Activities undertaken after initiating development to ensure project progress aligns with the plan. - Involves adapting to specific situations and updating plans as necessary. - Parameters are periodically re-estimated based on evolving project understanding. <p>Modern Project Management Practices</p> <ul style="list-style-type: none"> - Focus on code reuse, compression of project durations, and incremental delivery. - Encourage client feedback, customer participation, and accommodating change requests. - Emphasize rapid application development, deployment, and adaptive planning over long-term planning. <p>Success Criteria for a Project</p> <ul style="list-style-type: none"> - Meeting stakeholder objectives and satisfying real needs. - Ensuring project completion within estimated cost, duration, and effort. - Adaptation to changing project requirements and risks. <p>Stakeholders in a Project</p> <ul style="list-style-type: none"> - Individuals or groups with an interest in the project's outcome. - Their objectives must be identified, understood, and met for project success. <p>Need for Careful Planning, Monitoring, and Control</p> <ul style="list-style-type: none"> - Crucial for ensuring project success and meeting stakeholder expectations. - Involves detailed estimation, scheduling, staffing, and risk management activities. - Key to adapting to evolving project requirements and maintaining project alignment. 			
b	<p>List and explain categorizing software projects.</p> <p>1. Compulsory vs. Voluntary Users:</p> <ul style="list-style-type: none"> - Systems that staff must use vs. voluntary systems like computer games - Eliciting precise requirements differs between the two types 	L1	4	10

		<p>2. Information Systems vs. Embedded Systems:</p> <ul style="list-style-type: none"> - Information systems enable office processes; embedded systems control machines - Examples: Stock control system vs. process control system for air conditioning <p>Managing Software Projects:</p> <ul style="list-style-type: none"> - Outsourcing vs. Product Development: <ul style="list-style-type: none"> - Outsourced projects usually smaller, completed in few months - Managing outsourced projects presents special challenges - Modern vs. Traditional Software Project Management: <ul style="list-style-type: none"> - Key aspects where modern practices differ from traditional ones - Challenges in Software Development Projects: <ul style="list-style-type: none"> - Software projects harder to manage due to specific characteristics - Differences from managing building construction projects <p>Essential Planning in Software Projects:</p> <ul style="list-style-type: none"> - Method vs. Methodology: <ul style="list-style-type: none"> - Difference between a method and a methodology - Essential items to plan before carrying out a method or methodology <p>Additional Aspects of Software Projects: -</p> <ul style="list-style-type: none"> - Timing involved in project completion - Resources allocation and coordination - Impact of project size on difficulty level 			
OR					
Q. 08	a	<p>Explain traditional v/s project management practices</p> <p>Traditional vs. Project Management Practices</p> <p>Key Differences:</p> <p>1. Planning Approach :</p> <ul style="list-style-type: none"> - Traditional: Projects were planned with sufficient detail before execution. - Modern: Focus on rapid application development and deployment, replacing long-term planning with adaptive strategies. <p>2. Project Duration :</p>	L2	4	10

	<ul style="list-style-type: none"> - Traditional: Projects had longer durations. - Modern: Projects are completed over shorter durations with a focus on quick delivery. <p>3. Code Reuse and Project Compression :</p> <ul style="list-style-type: none"> - Modern projects emphasize maximizing code reuse and compressing project durations. <p>4. Client Involvement :</p> <ul style="list-style-type: none"> - Modern practices encourage client feedback, participation, and incremental delivery of evolving functionalities. <p>5. Change Requests :</p> <ul style="list-style-type: none"> - Modern practices encourage change requests from customers rather than avoiding them. <p>6. Cost and Time Pressures :</p> <ul style="list-style-type: none"> - Clients demand further reductions in product delivery times and costs in modern project management. <p>7. Monitoring and Control :</p> <ul style="list-style-type: none"> - In the modern approach, monitoring and control activities occur during project execution to adapt to changing circumstances. <p>Impact on Project Management:</p> <ul style="list-style-type: none"> - Recent developments have significantly impacted project management practices, shifting from traditional to modern methodologies. - The software industry has transitioned from developing software from scratch to tailoring existing products or reusing pre-built libraries. - Project managers now need to adapt to shorter project durations and prioritize adaptive strategies over long-term planning. <p>Conclusion:</p> <p>The shift from traditional to modern project management practices has revolutionized how projects are planned and executed, emphasizing rapid delivery, client involvement, code reuse, and adaptability to change. This evolution reflects the dynamic nature of project management in response to industry demands and technological advancements.</p>			
b	<p>How to assess Project Success and Failure in SPM.</p> <p>Assessing Project Success and Failure in Software</p>	L2	4	10

Project Management (SPM)

Key Aspects of Project Success and Failure

- Project objectives vs. business objectives: Project objectives include delivering agreed functionality, quality, on-time delivery, and within budget. Meeting these targets may not guarantee overall business success if external factors like customer adoption are not considered.
- Value of benefits exceeding costs: A project is generally considered successful if the benefits outweigh the costs, emphasizing the importance of cost control and realizing value from project deliverables.
- Impact of development costs and completion delays: Increasing development costs reduce profitability, while delays in completion limit the time available for generating benefits and diminish the project's value.

Factors Influencing Project Success

- Clear objectives: Having clear project objectives is crucial for project success, facilitating effective communication and ensuring all stakeholders are aligned.
- Testing objectives: Practical ways of testing whether project objectives have been met are essential for evaluating success.
- Effective communication channels: Projects involving multiple stakeholders require clear, objective measures of success to facilitate unambiguous communication among parties.

Modern vs. Traditional Software Project Management Practices

- Modern software project management practices differ from traditional ones in various key aspects, such as the approach to project planning, communication strategies, stakeholder involvement, and the use of technology for project monitoring and control.

Major Activities of a Software Project Manager

1. Defining the scope of software project management.
2. Addressing problems and concerns specific to software project management.
3. Identifying the usual stages of a software project.
4. Understanding the main elements of the management role.
5. Emphasizing the need for meticulous planning, monitoring, and control.
6. Identifying project stakeholders and their objectives.
7. Defining success criteria for a project.

	<p>Order of Major Activities for a Software Project Manager</p> <ol style="list-style-type: none">1. Defining the scope of software project management.2. Addressing problems and concerns specific to software project management.3. Identifying the usual stages of a software project.4. Understanding the main elements of the management role.5. Emphasizing the need for meticulous planning, monitoring, and control.6. Identifying project stakeholders and their objectives.7. Defining success criteria for a project.			
--	--	--	--	--

Module-5 <u>DOWNLOAD</u>					
Q. 09	a	<p>Define software quality and explain place of software quality in project management.</p> <p>Software Quality and Its Importance in Project Management</p> <p>Definition of Software Quality Software quality refers to how well a system operates and meets user expectations. It includes the reliability, usability, maintainability, efficiency, and other characteristics of software products. Quality requirements specify how well a system should operate, alongside functional and resource requirements.</p> <p>Software Quality in Project Management</p> <ol style="list-style-type: none"> 1. Essential Component : Quality management is crucial for effective project management. 2. Special Demands of Software : Software quality is critical due to the intangibility of software, accumulating errors during development, and the increasing criticality of software in various applications. 3. Quality Planning : Quality plans in project management include scope, management arrangements , documentation, standards, reviews, testing, problem reporting, tools, training, risk management, and more. 4. Testing and Verification : Practical ways of testing for quality presence and absence are necessary , with most qualities apparent to users only testable when the system is completed. 5. Quality Assurance Plans : Quality assurance plans aim to ensure that all quality issues are addressed during the planning process, with detailed entries covering various aspects of quality management. 6. CMM Model : The Capability Maturity Model (CMM) is a reference model for evaluating software development organizations into five process maturity levels, aiding in predicting project outcomes based on the organization's development processes. <p>Importance of Software Quality</p> <ol style="list-style-type: none"> 1. Criticality : Quality is a concern for all producers due to the increasing criticality of software in various applications, especially in safety-critical systems like aircraft control. 2. Intangibility : The intangible nature of software 	L2	5	10

	<p>makes it challenging to assess task outcomes satisfactorily, emphasizing the need for tangible deliverables to examine for quality.</p> <p>3. Error Accumulation : Errors in software development accumulate throughout the process, leading to cost escalations and quality issues, highlighting the importance of managing quality throughout the project lifecycle.</p> <p>Summary Software quality is essential in project management to ensure that software products meet user expectations, operate effectively, and maintain reliability. Quality planning, testing, assurance plans, and the CMM model play vital roles in managing software quality throughout the development process.</p>			
b	<p>Explain capability process model and CMM key areas.</p> <p>Capability Process Model and CMM Key Areas</p> <p>Capability Process Model (CMM)</p> <ul style="list-style-type: none"> - The Capability Maturity Model (CMM) is a reference model for appraising a software development organization into one of five process maturity levels. - It assists in predicting the likely outcome of a project based on the quality of the development process used by the organization. - CMM was developed in collaboration with the Software Engineering Institute (SEI) of Carnegie Mellon University to improve software acquisition methods and contractor performance evaluation. - CMM has been widely adopted by organizations for process improvement initiatives, leading to improved software quality and significant business benefits. - CMM has evolved into the Capability Maturity Model Integration (CMMI) to provide a generalized framework applicable to various domains using a single framework. <p>CMM Key Process Areas</p> <ul style="list-style-type: none"> - CMMI identifies key process areas (KPAs) at different maturity levels to guide organizations in improving their process quality. - Each maturity level, except for Level 1, is characterized by several KPAs that an organization needs to focus on to advance to the next level. - The key process areas include requirements management, 	L2	5	10

		<p>project planning and monitoring, supplier agreement management, measurement and analysis, process and product quality assurance, and configuration management.</p> <ul style="list-style-type: none"> - KPAs serve as focal points for organizations to enhance their quality gradually over several stages. - Focusing on higher-level KPAs before achieving lower-level ones can be counterproductive as it may become difficult to follow the established processes due to schedule and budget pressures. <p>Example of KPA Implementation</p> <ul style="list-style-type: none"> - Trying to implement a defined process (Level 3) before having a repeatable process (Level 2) could be counterproductive. - For instance, attempting to enforce detailed requirements management without having a stable project planning and monitoring process in place may lead to inefficiencies and difficulties in meeting project goals effectively. <p>Importance of Process Maturity Levels</p> <ul style="list-style-type: none"> - Process maturity levels in CMM/CMMI provide organizations with a structured approach to gradually enhance their process capabilities. - Higher maturity levels signify better process control, predictability, and effectiveness in delivering quality products or services. - Achieving higher levels requires a systematic improvement approach across key process areas to ensure sustained and continuous enhancement in organizational processes. <p>Relevance of CMM in Organizations</p> <ul style="list-style-type: none"> - CMM and CMMI have become valuable tools for organizations aiming to enhance their software development processes and overall performance. - These models offer a framework for organizations to assess, improve, and optimize their processes, ultimately leading to better quality outcomes and increased efficiency. 			
OR					
Q. 10	a	<p>Explain product v/s process quality management.</p> <p>Product vs Process Quality Management</p> <p>Product Quality Management</p> <ul style="list-style-type: none"> - Definition : Focuses on measuring characteristics of the final product being developed. - Examples of Product Metrics : <ul style="list-style-type: none"> - LOC and function point metrics measure size. 	L2	5	10

- PM (person-month) metric measures effort required.
- Time required for development is measured in months.
- Purpose : Evaluates the product's quality based on specific attributes like size, effort, and development time.

Process Quality Management

- Definition : Involves measuring how the development process is performing.
- Examples of Process Metrics :
 - Review effectiveness, average defects found per hour of inspection.
 - Average defect correction time, productivity, failures detected during testing per LOC, latent defects per line of code.
- Purpose : Analyzes the efficiency and effectiveness of the development process itself.

Comparison

- Product Quality Focus : Centers on the characteristics and attributes of the final product.
- Process Quality Focus : Centers on evaluating and improving the efficiency and effectiveness of the development process.
- Benefits :
 - Product Quality Management ensures the final product meets specified standards.
 - Process Quality Management enhances the development process to produce better products in the long run.

Importance

- Balanced Approach : Both product and process quality management are crucial for ensuring overall quality in software development.
- Integration : Integrating both approaches can lead to improved products and more efficient development processes.

Significance

- Efficiency : Process quality management can lead to smoother development processes and reduced errors.
- Effectiveness : Product quality management ensures that the final product meets user requirements and expectations.

Challenges

- Measurement : Process metrics can be more challenging to quantify compared to product metrics.
- Implementation : Balancing both aspects effectively

		can be a significant challenge for organizations.			
		<p>Conclusion Product quality management focuses on the final product characteristics, while process quality management evaluates the efficiency and effectiveness of the development process. Both are essential for ensuring high-quality software output and continuous improvement in software development practices.</p>			
	b	<p>Explain in detail the techniques to enhance software quality.</p> <p>Techniques to Enhance Software Quality</p> <p>Quality Specifications</p> <ul style="list-style-type: none"> - Definition/description of the quality characteristic. - Scale: unit of measurement. - Test: practical test of the extent to which the attribute quality exists. - Minimally acceptable value and target range. - Current value for assessment. <p>Measurement for Reliability</p> <ul style="list-style-type: none"> - Availability: percentage of time a system is usable. - Mean time between failures: total service time divided by failures. <p>Peer Review Process</p> <ul style="list-style-type: none"> - Inspection meetings limited to two hours. - Led by a moderator with specific training. - Defined roles for participants. - Use of checklists for fault-finding. - Material inspected at an optimal rate. - Maintenance of statistics for monitoring effectiveness. <p>Fagan Inspection Method</p> <ul style="list-style-type: none"> - Inspections on all major deliverables. - All types of defects noted. - Inspections at all levels except the top. - Use of predefined steps for inspections. <p>Debugging and Error Correction</p> <ul style="list-style-type: none"> - Debugging to identify error statements. - Correction of errors in the code. - Defect retesting after correction. 	L1	5	10

	<p>Benefits of Review Techniques</p> <ul style="list-style-type: none"> - Effective in removing superficial errors. - Motivates developers to produce better software. - Enhances team spirit and promotes good programming practices. <p>Software Testing Process</p> <ul style="list-style-type: none"> - Identification and correction of faults. - Dealing with incorrect specifications. - Informal communication of test failures to the development team. <p>Improving Software Delivery</p> <ul style="list-style-type: none"> - Proper planning and control to assess and address issues. - Identifying and addressing problems systematically. - Implementing formal processes for quality assurance. <p>SEI CMM Maturity Levels</p> <ul style="list-style-type: none"> - Steps to achieve repeatable software development. - Importance of implementing lower-level KPAs before higher-level ones. 			
--	--	--	--	--

Bloom's Taxonomy Level: Indicate as L1, L2, L3, L4, etc. It is also desirable to indicate the COs and POs to be attained by every bit of questions.