

**Model Question Paper (CBCS Scheme)**  
**SIXTH Semester B.E. Degree Examination**

**SOFTWARE ENGINEERING AND PROJECT  
MANAGEMENT**

**TIME: 03 Hours**  
**100**

**Max. Marks:**

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

**These Answers are from TEXTBOOK**

<p align="center"><b>Module -1</b>  <a href="#">Download</a></p>			<b>Bloom's Taxonomy Level</b>	<b>Marks</b>
Q.01	a	<p>Define process and explain generic process framework for software engineering.  Define Process and Generic Process Framework for Software Engineering</p> <p align="center"><b>Figure 2.2 Process flow</b></p> <pre> graph LR     subgraph (a) Linear process flow         C1[Communication] --&gt; P1[Planning] --&gt; M1[Modeling] --&gt; Co1[Construction] --&gt; D1[Deployment]     end     subgraph (b) Iterative process flow         C2[Communication] --&gt; P2[Planning] --&gt; M2[Modeling] --&gt; Co2[Construction] --&gt; D2[Deployment]         P2 --&gt; C2         M2 --&gt; P2         Co2 --&gt; M2     end     subgraph (c) Evolutionary process flow         C3[Communication] --&gt; P3[Planning] --&gt; M3[Modeling] --&gt; Co3[Construction] --&gt; D3[Deployment]         D3 -- Increment released --&gt; C3     end     subgraph (d) Parallel process flow         C4[Communication] --&gt; P4[Planning]         P4 --&gt; M4[Modeling]         M4 --&gt; Co4[Construction] --&gt; D4[Deployment]         Time[Time] --&gt; Co4     end </pre> <p>(a) Linear process flow</p> <p>(b) Iterative process flow</p> <p>(c) Evolutionary process flow</p> <p>(d) Parallel process flow</p> <p>Software Process Definition -  A software process is a framework for the activities, actions, and</p>	L2	10

tasks necessary to build high-quality software. It defines the approach taken in software engineering, encompassing both the process itself and the technologies utilized. Software engineering involves creative individuals who adapt mature processes to suit their product requirements and market demands.

#### Generic Process Framework for Software Engineering -

- A software process model consists of framework activities, actions, and tasks that guide the creation of work products.
- The generic process framework for software engineering includes five core framework activities: communication, planning, modeling, construction, and deployment.
- Additionally, umbrella activities like project tracking, risk management, quality assurance, configuration management, and technical reviews are integrated throughout the process.
- The process flow defines how framework activities, actions, and tasks are organized in terms of sequence and time, with variations such as linear, iterative, evolutionary, and parallel flows.

#### Example Process Models -

- The Waterfall Model: Suitable when requirements are well understood, and work progresses linearly from communication to deployment.
- Agile Process Models: Emphasize adaptability and are suitable for various projects, especially web applications.

#### Benefits of Generic Process Framework

- Provides a structured approach for software development.
- Ensures coordination and coherence in software work.
- Allows for adaptability to different project requirements and market dynamics.

#### Importance of Process Models

- Process models aid in achieving project consistency and order.
- 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.

#### Agility in Software Engineering

- Agility is crucial in modern software engineering to respond effectively to changing requirements and market conditions.

		- Agile software development differs from traditional models by emphasizing flexibility and iterative development approaches.		
b	Demonstrate the waterfall model spiral and model with real time example.	<p>Waterfall Model vs. Spiral Model: A Comparison with Real-Time Examples</p> <p><b>FIGURE 2.3</b> The waterfall model</p> <p><b>FIGURE 2.7</b> A typical spiral model</p>	L3	10
		<p>Waterfall Model Overview:</p> <ul style="list-style-type: none"><li>- The waterfall model is a systematic, sequential approach to software development.</li><li>- It begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.</li><li>- The V-Model is a variation of the waterfall model that depicts the relationship of quality.</li></ul> <p>Issues with Waterfall Model:</p> <ul style="list-style-type: none"><li>- It requires explicit customer requirements which can be challenging.</li><li>- Difficulty accommodating natural uncertainties at the project's beginning.</li><li>- Blocking states can occur, leading to wait times exceeding productive work time.</li></ul> <p>Spiral Model Overview:</p>		

- Proposed by Barry Boehm, the spiral model combines iterative prototyping with controlled aspects of the waterfall model.
- It is an evolutionary process model that allows for rapid development of increasingly complete software versions.

#### Spiral Model Characteristics:

- It is a risk-driven model that guides multi-stakeholder concurrent engineering.
- Involves incremental growth of system definition and implementation while reducing risk.
- Employs anchor point milestones for stakeholder commitment to feasible solutions.

#### Real-Time Example of Spiral Model:

- The spiral model involves a series of evolutionary releases.
- Early iterations may result in a model or prototype, while later iterations produce more sophisticated versions.
- Adjustments to project plans, costs, and schedules are made based on customer feedback.

#### Concurrent Models:

- Concurrent development models allow for iterative and concurrent elements from various process models.
- Modeling activities in the spiral model can incorporate prototyping, analysis, and design simultaneously.

#### Advantages of Spiral Model:

- Adaptable throughout the life cycle of software.
- Supports rapid development of software in evolutionary releases.
- Allows for risk reduction and stakeholder commitment through anchor point milestones.

#### Real-World Application:

- The spiral model can be suitable for projects where requirements are not fixed and changes are frequent.
- It provides flexibility and adaptability in scenarios where a linear approach like the waterfall model may not be suitable.

#### Conclusion:

In conclusion, the waterfall model follows a linear sequential approach, whereas the spiral model combines iterative prototyping with controlled aspects, allowing for rapid and

		adaptive software development. The spiral model's risk-driven nature, incremental growth, and stakeholder commitment through milestones make it a valuable alternative to the waterfall model in dynamic software development environments.		
OR				
Q.02	a	<p>Explain characteristics that differentiate WebApps from other software.</p> <p>Characteristics Differentiating WebApps from Other Software</p> <p>Network Intensiveness</p> <ul style="list-style-type: none"> <li>- WebApps reside on networks and serve diverse client needs.</li> <li>- They may enable worldwide or limited access and communication.</li> </ul> <p>Concurrency</p> <ul style="list-style-type: none"> <li>- Large numbers of users can access WebApps simultaneously.</li> <li>- Usage patterns among users vary greatly.</li> </ul> <p>Unpredictable Load</p> <ul style="list-style-type: none"> <li>- The number of users accessing WebApps can vary significantly day by day.</li> </ul> <p>Performance</p> <ul style="list-style-type: none"> <li>- Users may leave if they experience long wait times for server-side processing or content display.</li> </ul> <p>Availability</p> <ul style="list-style-type: none"> <li>- Users often expect 24/7 access to popular WebApps.</li> </ul> <p>Data-Driven</p> <ul style="list-style-type: none"> <li>- WebApps primarily use hypermedia to present various content types to users.</li> </ul> <p>Content Sensitivity</p> <ul style="list-style-type: none"> <li>- Quality and aesthetics of content impact the overall quality of a WebApp.</li> </ul> <p>Continuous Evolution</p> <ul style="list-style-type: none"> <li>- WebApps evolve continuously, with content updates sometimes happening on a minute-by-minute basis.</li> </ul>	L2	10

		<p>Immediacy</p> <ul style="list-style-type: none"> <li>- WebApps often have a quick time-to-market, sometimes launching in a matter of days or weeks.</li> </ul> <p>Security</p> <ul style="list-style-type: none"> <li>- Due to network access, WebApps face challenges in limiting user access and ensuring content security.</li> </ul>		
	b	<p>Discuss the David Hooker's seven principles of software engineering practice.</p> <p>David Hooker's Seven Principles of Software Engineering Practice</p> <p>The First Principle: The Reason It All Exists</p> <ul style="list-style-type: none"> <li>- A software system exists to provide value to its users.</li> <li>- All decisions should prioritize adding real value to the system.</li> </ul> <p>The Second Principle: KISS (Keep It Simple, Stupid!)</p> <ul style="list-style-type: none"> <li>- Software design should be as simple as possible but not oversimplified.</li> <li>- Simplicity facilitates understanding and maintenance of the system.</li> </ul> <p>The Third Principle: Plan Ahead for Reuse</p> <ul style="list-style-type: none"> <li>- Reuse of code and designs saves time and effort.</li> <li>- Achieving a high level of reuse requires forethought and planning.</li> </ul> <p>The Fourth Principle: Strive for a Balance</p> <ul style="list-style-type: none"> <li>- Balance is essential in software engineering.</li> <li>- Balancing conflicting forces like time, effort, functionality, and quality is crucial.</li> </ul> <p>The Fifth Principle: Listen to the Cavalry</p> <ul style="list-style-type: none"> <li>- Input from stakeholders, users, and other relevant parties is valuable.</li> <li>- Listening to different perspectives aids in developing successful software systems.</li> </ul>	L2	10

The Sixth Principle: Prototype, but Be Careful

- |  |   |  |  |
|--|---|--|--|
|  | <ul style="list-style-type: none"><li>- Prototyping can be beneficial but comes with risks.</li><li>- Careful consideration and planning are necessary to avoid pitfalls in the prototyping process.</li></ul> <p>The Seventh Principle: Think!</p> <ul style="list-style-type: none"><li>- Clear and complete thought before action leads to better results.</li><li>- Thinking before acting helps in doing things correctly and gaining valuable experience.</li></ul> <p>Importance of Following the Principles</p> <ul style="list-style-type: none"><li>- Following these principles can eliminate difficulties in building complex computer-based systems.</li><li>- Engaging intense thought and consideration in software engineering practices leads to significant rewards.</li></ul> <p>Application of Principles</p> <ul style="list-style-type: none"><li>- Implementing these principles can guide software engineers towards effective software engineering practices.</li><li>- Each principle serves as a foundational concept for solid software engineering practice.</li></ul> |  |  |
|--|---|--|--|

## Module-2

### [Download](#)

Q. 03

a Develop a UML use case diagram for home security function.

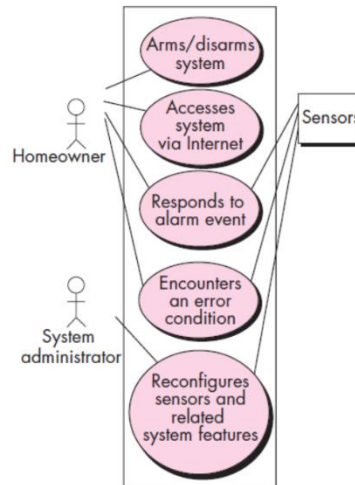
L3

10

### UML Use Case Diagram for Home Security Function

**FIGURE 5.2**

UML use case diagram for SafeHome home security function



#### Overview

The UML use case diagram for the SafeHome home security function includes various actors and use cases related to the system's functionality and interactions. The diagram visualizes the primary actors and their interactions with the system, highlighting essential scenarios and requirements.

#### Actors

1. Homeowner
2. System administrator

#### Use Cases

1. Arms/disarms system
2. Responds to alarm event
3. Accesses system via Internet
4. Encounters an error condition
5. Reconfigures sensors and related system features

#### Additional Information

- Use cases depict the software or system from the end user's point of view.
- The diagram includes actors represented by stick figures and labeled squares to represent non-human actors like sensors.
- Use case narratives are developed for each oval in the diagram to detail specific scenarios and interactions.
- The system must be fully configured, and appropriate user IDs



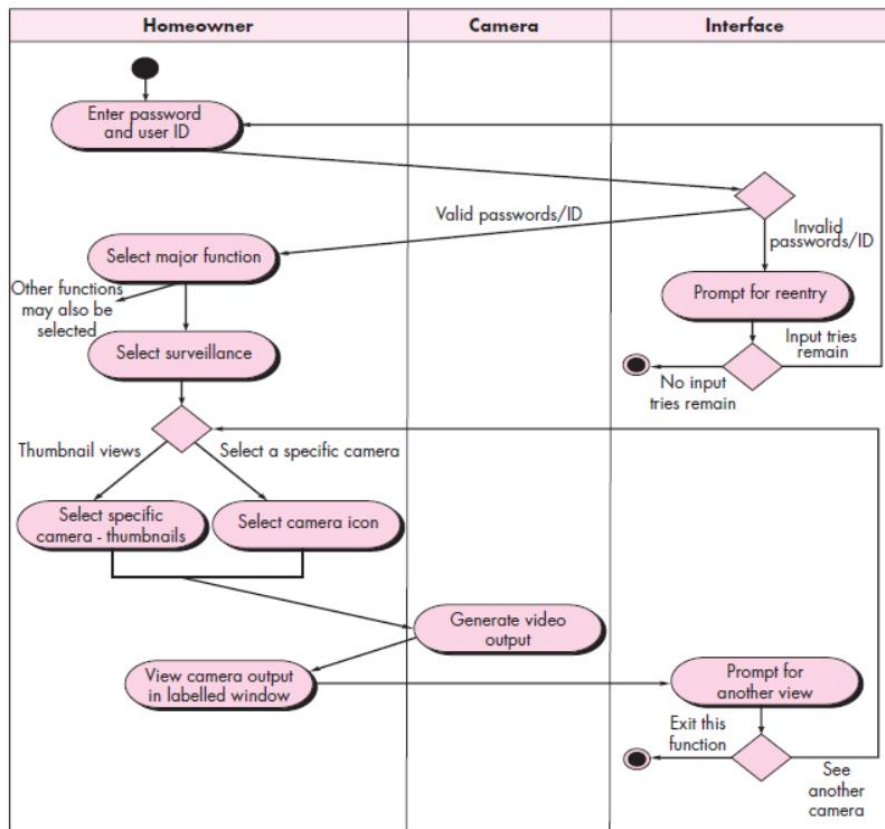
	<p>and passwords must be obtained to trigger specific actions like accessing camera surveillance via the Internet.</p> <p><b>Relevant Considerations</b></p> <ul style="list-style-type: none"> <li>- Ensuring system security against unauthorized use and potential hacking threats.</li> <li>- Addressing system response via the Internet in terms of bandwidth requirements and video quality.</li> <li>- Providing detailed scenario steps for accessing and using specific system functions.</li> <li>- Considering different states of the system such as home, away, overnight travel, and extended travel with corresponding settings and controls.</li> </ul>		
b	<p>Narrate the steps to be followed for building the requirement model.</p> <p><b>Steps for Building a Requirement Model</b></p> <p><b>Overview of Requirements Engineering Tasks</b></p> <ul style="list-style-type: none"> <li>- Requirements engineering tasks establish a foundation for design and construction.</li> <li>- Seven distinct requirements engineering functions are conducted: inception, elicitation, elaboration, negotiation, specification, validation, and management.</li> <li>- Stakeholders establish basic problem requirements and project constraints at project inception.</li> <li>- Elicitation involves gathering requirements through facilitated meetings, QFD, and usage scenarios.</li> </ul> <p><b>Requirements Modeling Objectives and Philosophy</b></p> <ul style="list-style-type: none"> <li>- Focus on "what" rather than "how" during requirements modeling.</li> <li>- Use an iterative approach due to uncertainty in specific requirements.</li> <li>- The requirements model must describe customer needs, aid in software design creation, and provide a basis for validation.</li> </ul> <p><b>Steps for Building a Requirement Model</b></p> <p>1. Scenario-Based Modeling :</p> <ul style="list-style-type: none"> <li>- Represent the system from the user's perspective.</li> <li>- Develop basic use cases and scenarios to understand user interactions.</li> </ul>	L2	10

		<p>2. Data Modeling :</p> <ul style="list-style-type: none"> <li>- Depict the information space and data objects manipulated by the software.</li> <li>- Illustrate relationships among data objects.</li> </ul> <p>3. Class-Based Modeling :</p> <ul style="list-style-type: none"> <li>- Define objects, attributes, and relationships within the system.</li> <li>- Identify candidate classes based on descriptive nouns in use-case scripts.</li> </ul> <p>4. Refinement and Analysis :</p> <ul style="list-style-type: none"> <li>- Create preliminary models and refine them for clarity, completeness, and consistency.</li> <li>- Analyze models to ensure accuracy and address any issues.</li> </ul> <p>5. Review and Validation :</p> <ul style="list-style-type: none"> <li>- Prioritize requirements, group them into packages for implementation.</li> <li>- Validate requirements model by asking specific questions related to consistency, completeness, ambiguity, achievability, and testability.</li> </ul> <p>Importance of Requirements Model</p> <ul style="list-style-type: none"> <li>- The model bridges the gap between system-level description and software design.</li> <li>- It helps in understanding stakeholder needs, guiding design tasks, and providing a basis for validation.</li> </ul> <p>Key Considerations</p> <ul style="list-style-type: none"> <li>- Use multiple modes of representation to uncover omissions, inconsistencies, and ambiguities.</li> <li>- Engage stakeholders throughout the modeling process to ensure alignment with customer requirements.</li> </ul>		
OR				
Q.04	a	<p>Explain the steps to be followed for validating requirements in detail.</p> <p>Steps for Validating Requirements</p> <p>1. Review for Consistency, Omissions, and Ambiguity - Examine each element of the requirements model for</p>	L2	10

		<p>inconsistencies, omissions, and ambiguity.</p> <ul style="list-style-type: none"> <li>- Prioritize requirements by stakeholders and group them within packages for implementation as software increments.</li> </ul> <p>2. Ask Key Questions during Review</p> <ul style="list-style-type: none"> <li>- Ensure each requirement aligns with the overall system objectives.</li> <li>- Verify that requirements are specified at the appropriate level of abstraction.</li> <li>- Determine if each requirement is necessary or an add-on feature.</li> <li>- Confirm that each requirement is bounded, unambiguous, and attributed to a specific source.</li> <li>- Check for conflicts between requirements and their achievability in the technical environment.</li> <li>- Assess if each requirement is testable once implemented and reflects the system's information, function, and behavior.</li> </ul> <p>3. Utilize a Requirements Validation Checklist</p> <ul style="list-style-type: none"> <li>- Examine requirements against a set of checklist questions for clarity, interpretability, and testability.</li> <li>- Verify if requirements are clearly stated, bounded quantitatively, and traceable to system objectives.</li> <li>- Ensure requirements have been consistently stated and structured for easy understanding and translation into technical work products.</li> </ul> <p>4. Review Work Products and Ensure Correctness</p> <ul style="list-style-type: none"> <li>- Review requirements modeling work products for correctness, completeness, and consistency.</li> <li>- Ensure that the work products reflect the needs of all stakeholders and establish a foundation for design.</li> </ul> <p>5. Focus on Consistency and Traceability</p> <ul style="list-style-type: none"> <li>- Use the analysis model to ensure requirements have been consistently stated and traceable to system models.</li> <li>- Pay attention to ensuring that requirements are clear, testable, and aligned with overall system/product objectives.</li> </ul>		
	b	<p>Draw the Swimlane diagram for access camera surveillance via the Internet and explain its functions.</p> <p>Swimlane Diagram for Access Camera Surveillance via the</p>	L2	10

## Internet

**FIGURE 6.6** Swimlane diagram for Access camera surveillance via the Internet—display camera views function



### Swimlane Diagram

- The swimlane diagram for accessing camera surveillance via the Internet involves activities associated with different analysis classes like Homeowner, Camera, and Interface.
- Responsibilities are divided among these classes, with actions like entering passwords, selecting major functions, and viewing camera outputs being allocated to specific lanes.
- The diagram represents the flow of actions and decisions, indicating which actors perform each task within the surveillance function.

### Functions of Accessing Camera Surveillance via the Internet

1. User Authentication:
  - Entering user ID and passwords for validation to access the SafeHome system.
  - Handling incorrect or unrecognized passwords and IDs.
2. Camera Selection and Display:
  - Selecting the "surveillance" major function and picking a

	<p>specific camera for viewing.</p> <ul style="list-style-type: none"><li>- Displaying floor plans of the house and camera icons for selection.</li><li>- Viewing camera outputs in labeled windows identified by camera IDs.</li></ul> <p>3. Video Output and Control:</p> <ul style="list-style-type: none"><li>- Generating video output within viewing windows at a frame rate of one frame per second.</li><li>- Controlling functions like pan and zoom for specific cameras.</li></ul> <p>4. Additional Features:</p> <ul style="list-style-type: none"><li>- Offering thumbnail views of all cameras simultaneously.</li><li>- Allowing selective recording and replaying of camera outputs.</li><li>- Providing the option to block access to certain cameras with specific passwords.</li><li>- Maintaining a consistent interface look and feel across all SafeHome interfaces.</li></ul> <p>Implementation Details</p> <ul style="list-style-type: none"><li>- Priority: Moderate priority for implementation after basic functions.</li><li>- Frequency of Use: Moderately frequent.</li><li>- Channels to Actor: Accessible via PC-based browser and Internet connection.</li><li>- Secondary Actors: System administrator and cameras with respective channels for interaction.</li><li>- Open Issues: Addressing concerns about unauthorized use by employees of SafeHome Products.</li></ul>		
--	---	--	--

## Module-3

### [Download](#)

Q. 05

a

Explain Adaptive Software Development (ASD) Model with sketch.

L2

10

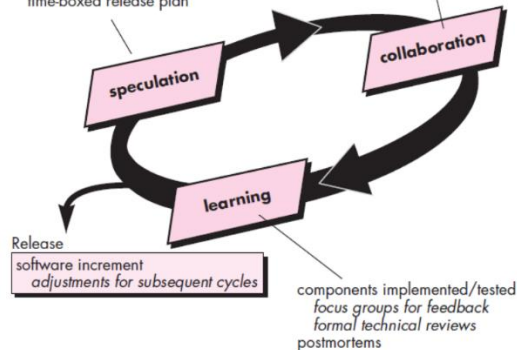
### Adaptive Software Development (ASD) Model Overview

FIGURE 3.3

Adaptive  
software  
development

adaptive cycle planning  
mission statement  
project constraints  
basic requirements  
time-boxed release plan

Requirements gathering  
JAD  
mini-specs



#### Philosophy and Approach

- Proposed by Jim Highsmith for building complex software and systems.
- Focuses on human collaboration and team self-organization.
- Emphasizes an agile, adaptive development approach based on collaboration.
- Incorporates three phases: speculation, collaboration, and learning.

#### Key Elements of ASD

1. Learning Emphasis
  - Teams prioritize learning alongside progress.
  - Learning improves understanding of technology, processes, and projects.
2. Modes of Learning
  - Focus groups, technical reviews, and project postmortems.
3. Team Dynamics
  - ASD highlights self-organizing teams, interpersonal collaboration, and individual/team learning for project success.

#### Adaptive Cycle Planning

- Utilizes project initiation information for planning.
- Involves customer's mission statement, project constraints, and basic requirements.

		<ul style="list-style-type: none"> <li>- Defines release cycles based on project needs, subject to adjustments.</li> </ul> <p>Collaboration and Communication</p> <ul style="list-style-type: none"> <li>- Motivation and Collaboration <ul style="list-style-type: none"> <li>- Emphasizes collaboration to enhance talent and creative output.</li> <li>- Stresses the importance of trust and effective communication within teams.</li> </ul> </li> <li>- Software Development Process <ul style="list-style-type: none"> <li>- Involves speculation, collaboration, and learning phases for effective project management.</li> </ul> </li> </ul> <p>ASD Resources</p> <ul style="list-style-type: none"> <li>- Website: <a href="http://www.adaptivesd.com">www.adaptivesd.com</a></li> <li>- Useful resources: adaptive cycle planning, mission statement, project constraints, basic requirements, time-boxed release plan, components implemented/tested, focus groups for feedback.</li> </ul>		
	b	<p>Narrate the core principles can be applied to the framework in all software process.</p> <p>Core Principles Applicable to Framework in All Software Processes</p> <p>Principles That Guide Process:</p> <ol style="list-style-type: none"> <li>1. Be Agile: Emphasize economy of action and simplicity in your technical approach.</li> <li>2. Focus on Quality at Every Step: Ensure the quality of work products is a primary focus in all activities.</li> <li>3. Be Ready to Adapt: Flexibility is essential in adapting the approach to constraints imposed by the problem, people, and project.</li> <li>4. Build an Effective Team: Establish and maintain an effective team for successful software engineering processes.</li> </ol> <p>Principles That Guide Practice:</p> <ol style="list-style-type: none"> <li>1. Divide and Conquer: Emphasize separation of concerns in analysis and design for easier problem-solving.</li> <li>2. Understand the Use of Abstraction: Utilize abstraction to simplify complex elements of a system for effective communication.</li> </ol>	L2	10

- |  |   |  |  |
|--|---|--|--|
|  | <p>3. Represent Problems and Solutions from Different Perspectives: Gain insights and uncover errors by examining issues from various viewpoints.</p> <p>4. Remember Someone Will Maintain the Software : Apply solid software engineering practices for long-term maintenance and enhancement.</p> <p>These core principles provide a foundation for software engineering methods, guiding both process and practice in software development.</p> <p>Communication Principles:</p> <ul style="list-style-type: none"><li>- Listen : Focus on the speaker's words, seek clarification when needed , and avoid interruptions or contentious behavior during communication.</li></ul> <p>Principles That Guide Each Framework Activity:</p> <ul style="list-style-type: none"><li>- Communication : Effective communication is essential among technical peers, customers, stakeholders, and project managers.</li><li>- Representation of Problems and Solutions: Viewing issues from different perspectives enhances understanding and uncovers insights.</li><li>- Software Maintenance: Emphasize the importance of maintaining software quality for long-term viability and adaptability.</li></ul> <p>These principles contribute to successful software engineering practices, emphasizing effective communication, problem representation, and software maintenance.</p> <p>Planning Principles:</p> <ul style="list-style-type: none"><li>- Constructing a Comprehensive Plan : Design a detailed plan outlining tasks , responsible individuals , and completion timelines for successful project execution.</li></ul> <p>Modeling Principles:</p> <ul style="list-style-type: none"><li>- Incremental Representation: Progressively detailed models solidify understanding and provide guidance for software implementation.</li></ul> <p>Construction Principles:</p> <ul style="list-style-type: none"><li>- Coding and Testing Cycles: Implement coding and testing</li></ul> |  |  |
|--|---|--|--|



		<p>procedures following generic actions to ensure software functionality and quality.</p> <p>These principles collectively guide software development processes, from planning and modeling to construction and maintenance.</p>		
OR				
Q. 06	a	<p>Elucidate the concepts of extreme programming (XP) with its functional diagram</p> <p><b>FIGURE 3.2</b> The Extreme Programming process</p> <p>Elucidating Extreme Programming (XP) Concepts with Functional Diagram</p> <p>Overview of Extreme Programming (XP)</p> <p>Extreme Programming (XP) is an agile software development approach organized around four framework activities: planning, design, coding, and testing. XP emphasizes rapid delivery of software that satisfies customer needs. Key aspects of XP include human collaboration, team self-organization, and creating frequent software releases based on stakeholder priorities.</p> <p>XP Values</p> <ul style="list-style-type: none"><li>- XP is driven by five core values: communication, simplicity, feedback, courage, and respect.</li><li>- Effective communication, simplicity in design, continuous feedback, courage to embrace changes, and respect among team members and stakeholders are central to XP.</li></ul> <p>XP Process</p>	L2	10

	<p>1. Planning: The planning game involves requirements gathering to understand business context and prioritize features</p> <p>2. Design: Emphasizes an object-oriented approach and involves creating solutions within the planning framework.</p> <p>3. Coding: Involves writing code based on the planned design and requirements.</p> <p>4. Testing: Focuses on ensuring software quality through real-time quality checks and feedback mechanisms.</p> <p>XP Concerns and Debate</p> <ul style="list-style-type: none"> <li>- Criticism: XP's incremental nature limits complexity, potentially leading to inadequate design.</li> <li>- Debate: The debate around XP includes concerns about downplaying analysis and design in favor of coding, the need for discipline in adapting XP to organizational needs, and the balance between agility and structured development.</li> </ul> <p>XP Implementation</p> <ul style="list-style-type: none"> <li>- XP recommends practices like pair programming and stakeholder involvement.</li> <li>- Incorporating XP concepts into development approaches can enhance agility while retaining aspects of disciplined design and analysis.</li> </ul>		
b	<p>What is design modelling? Explain design modelling principles.</p> <p>Design Modelling Principles</p> <p>Design Modelling Overview</p> <p>Design modelling in software engineering involves creating high-quality requirements and design models to guide the software development process. Design models provide a concrete specification for constructing software based on customer requirements.</p> <p>Principles of Design Modelling</p> <ol style="list-style-type: none"> <li>1. Explicit Purpose for Each Model : Ensure each model has a clear justification for its existence to avoid missing important functions and features during design.</li> <li>2. Adapt Models to the System : Modify model notation or rules to suit the specific application being developed, considering differences in requirements and constraints.</li> </ol>	L2	7

3. Build Useful Models, Not Perfect Ones : Focus on creating useful models rather than striving for perfection, as excessive effort in perfecting models may not yield significant benefits.

4. Communication through Models : Models should effectively communicate information, using a consistent format and standing on their own without the need for additional explanations.

5. Traceability to Requirements Model : Design should be traceable back to the requirements model to ensure alignment with customer needs and specifications.

6. Consider System Architecture : Design should start with the system architecture, as it influences interfaces, data structures, program control flow, testing, and system maintainability.

7. Data Design Importance : Emphasize the design of data structures as much as processing functions, as well-structured data design simplifies program flow and enhances software component design and implementation.

8. Iteration and Refinement : Design models progress from representing the entirety of the system to refining details, similar to an architect's plans, with different views of the system to guide construction.

#### Applying Design Modelling Principles

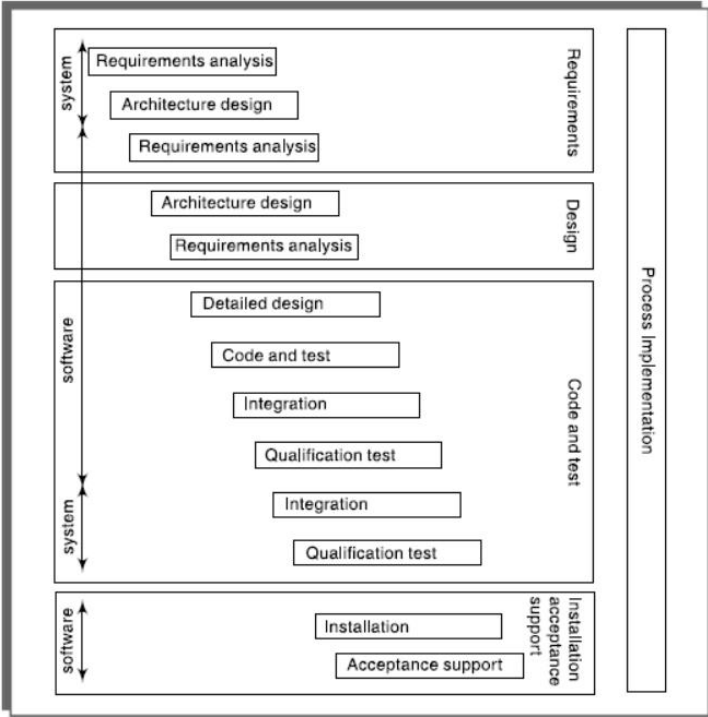
- The software design model acts as a blueprint for the software development process, evolving from high-level representations to detailed specifications.
- Various methods exist for deriving design elements, such as data-driven, pattern-driven, and object-oriented approaches, all guided by common design principles.
- These principles ensure consistency and effectiveness in software design, fostering traceability, adaptability, and communication in the development process.

#### Conclusion

Design modelling principles are essential in software

		engineering to create robust and effective design models that align with customer requirements and facilitate the development of high-quality software systems.		

vtuccloud.in

Module-4 <a href="#">Download</a>		Bloom's Taxonomy Level	Marks
Q. 07	<div>a</div> <div>Explain the software development life cycle with block diagram</div> <div>Software Development Life Cycle Explanation</div> <div>6 Software Project Management</div> <div></div> <div>FIGURE 1.3 The ISO 12207 software development life cycle</div> <div>Overview of Software Development Life Cycle</div> <div>The software development life cycle involves several key stages that guide the process from project initiation to completion. These stages include feasibility study, requirements analysis, architecture design, detailed design, planning, and incremental delivery. The life cycle emphasizes the importance of understanding project requirements, designing system architecture, and incrementally delivering the software product.</div> <div>Components of Software Development Life Cycle</div> <div>1. Feasibility Study: Assessing the project's viability and business case before initiation.</div> <div>2. Requirements Analysis: Gathering and understanding user and manager requirements for the new system.</div> <div>3. Architecture Design: Identifying components of the system that fulfill requirements and mapping software requirements to components.</div> <div>4. Detailed Design: Creating software components consisting of</div>	L2	10

separate units that are individually coded and tested.

5. Planning Incremental Delivery: Emphasizing rapid application development, project compression, and adaptive planning strategies.

#### Key Activities in the Software Development Life Cycle

1. Requirements Elicitation: Establishing user and manager needs for the system, including functional and quality requirements.
2. System and Software Development: Involves creating software components, designing user interfaces, internal architectures, and ICT infrastructure.
3. Training Course Organization: Involves designing training materials, drafting timetables, arranging course dates, identifying attendees, and providing necessary facilities.
4. Objective-Driven Development: Distinguishing projects based on product creation or meeting specific objectives, guiding the project's purpose and execution.

#### Modern Trends in Software Project Management

- Emphasis on code reuse and project duration compression.
- Facilitation of client feedback and incremental product delivery.
- Shift from long-term planning to adaptive strategies for rapid project completion.
- Increasing importance of service level agreements in outsourced projects.

#### Project Management Practices Evolution

- Projects now focus on tailoring existing products or reusing pre-built libraries.
- Clients demand further reductions in project delivery times and costs.
- Encouragement of change requests from customers for incremental product evolution.
- Adoption of modern project management practices for effective project development.

#### Software Development Life Cycle Block Diagram

The software development life cycle typically follows the sequence recommended in the ISO 12207 standard, involving various activities related to system and software development, infrastructure installation, user job design, and user training.

#### Conclusion

		The software development life cycle encompasses critical stages from feasibility study to incremental delivery, emphasizing requirements analysis, system architecture design, and adaptive planning for successful project completion.		
	b	<p>List the characteristics of projects and show the differences between Contract management and project management</p> <p>Characteristics of Projects</p> <ul style="list-style-type: none"> <li>- Non-routine tasks are involved.</li> <li>- Planning is required.</li> <li>- Specific objectives are to be met or a specified product is to be created.</li> <li>- The project has a predetermined time span.</li> <li>- Work is carried out for someone other than yourself.</li> <li>- Work involves several specialisms.</li> <li>- People are formed into a temporary work group to carry out the task.</li> <li>- Work is carried out in several phases.</li> <li>- The resources available for use on the project are constrained.</li> <li>- The project is large or complex.</li> </ul> <p>Differences Between Contract Management and Project Management</p> <ul style="list-style-type: none"> <li>- In contract management, organizations contract out ICT development to outside developers, while in project management, in-house projects involve users and developers working for the same organization.</li> <li>- Contract management focuses on supervising the contract and delegating technically oriented decisions to contractors, whereas project management involves control over development costs and project deliverables.</li> <li>- Contract managers ensure the overall project is within budget and on time, delegating technical decisions, whereas project managers have control over development costs and project benefits.</li> <li>- Project managers have considerable control over development costs and the value of project deliverables, while contract managers focus on technical issues and ensuring the project meets contractual obligations.</li> <li>- Project managers are concerned with achieving project objectives such as delivering agreed functionality, quality, timeliness, and budget adherence, while contract managers oversee compliance with contractual terms and conditions.</li> </ul>	L2	10

		<ul style="list-style-type: none"> <li>- Contract management involves appointing a project manager to supervise the contract, while project management involves overseeing the entire project lifecycle from initiation to closure.</li> <li>- Contract managers focus on the contractual aspects of the project, while project managers are responsible for overall project success and stakeholder satisfaction.</li> <li>- Contract management may involve multiple technical project managers dealing with specific technical issues, while project management requires a holistic approach to meet project objectives and deliver business value.</li> </ul>		
OR				
Q. 08	a	<p>Discuss the ways of categorizing the software projects with real time examples.</p> <p>Categorizing Software Projects with Real-Time Examples</p> <p>Characteristics of Software Projects:</p> <ul style="list-style-type: none"> <li>- Non-routine tasks are involved.</li> <li>- Specific objectives are to be met or a specified product is to be created.</li> <li>- The project has a predetermined time span.</li> <li>- Work is carried out for someone other than yourself.</li> <li>- Work involves several specialisms.</li> <li>- People are formed into a temporary work group to carry out the task.</li> <li>- Work is carried out in several phases.</li> <li>- The resources available for use on the project are constrained.</li> </ul> <p>Examples of Software Projects:</p> <ol style="list-style-type: none"> <li>1. Getting married.</li> <li>2. Amending a financial computer system to deal with a common European currency.</li> <li>3. A research project into what makes a good human-computer interface.</li> <li>4. An investigation into the reason why a user has a problem with a computer system.</li> <li>5. Writing an operating system for a new computer.</li> <li>6. Installing a new version of a word processing package in an organization.</li> <li>7. Developing a customized order processing system for a client.</li> <li>8. Organizing a training course for end-users of a software system.</li> </ol>	L2	10



	<p>Factors Affecting Software Projects:</p> <ul style="list-style-type: none"> <li>- Compulsory versus voluntary users.</li> <li>- Information systems versus embedded systems.</li> </ul> <p>Success Measurement:</p> <ul style="list-style-type: none"> <li>- Projects can be successful on delivery but may fail as a business.</li> <li>- Projects can be late and over budget but still generate benefits over time.</li> </ul> <p>Project Management Practices:</p> <ul style="list-style-type: none"> <li>- Projects are temporary sub-organizations involving specialists.</li> <li>- Projects may be seen as disruptive but are focused on important tasks.</li> </ul> <p>Outsourcing vs. Product Development Projects:</p> <ul style="list-style-type: none"> <li>- Managing outsourced projects differs in terms of control and communication.</li> <li>- Benefits of technical skills learned in earlier projects impact later projects positively.</li> </ul> <p>Differences in Modern Software Project Management:</p> <ul style="list-style-type: none"> <li>- Modern practices focus on reducing business risks.</li> <li>- Activities such as market surveys, competitor analysis, and prototyping are emphasized.</li> </ul> <p>Method vs. Methodology:</p> <ul style="list-style-type: none"> <li>- A method is a way of doing something, while a methodology is a system of methods used in a particular area.</li> <li>- Planning essential items before implementing a method or methodology is crucial for success.</li> </ul>		
	<p>b Elucidate the concepts in activity planning in software project management.</p> <p>Elucidating Concepts in Activity Planning in Software Project Management</p> <p>Key Concepts in Activity Planning:</p> <ol style="list-style-type: none"> <li>1. Software Project Management vs. Other Projects: <ul style="list-style-type: none"> <li>- Understanding the similarities and differences in managing software projects compared to other types of projects.</li> </ul> </li> </ol>	L3	10

	<ul style="list-style-type: none"> <li>- Emphasizing the importance of meeting objectives and satisfying real needs in software project management.</li> </ul> <p>2. Stakeholders and Objectives:</p> <ul style="list-style-type: none"> <li>- Identifying project stakeholders and their objectives as crucial components in project planning.</li> <li>- Ensuring that stakeholder objectives are met through effective project management practices.</li> </ul> <p>3. Elements of Project Management:</p> <ul style="list-style-type: none"> <li>- Outlining key elements of the management role, including planning, organizing, staffing, directing, monitoring, controlling, innovating, and representing.</li> <li>- Highlighting the significance of careful planning, monitoring, and control in software project management.</li> </ul> <p>4. Project Stages and Activities:</p> <ul style="list-style-type: none"> <li>- Defining the usual stages of a software project, emphasizing the need for detailed planning, monitoring, and control throughout.</li> <li>- Discussing the importance of creating test scripts, comparing actual vs. expected results, and identifying discrepancies as part of project activities.</li> </ul> <p>5. Activity Planning Details:</p> <ul style="list-style-type: none"> <li>- Explaining that a plan in software project management converts methods into real activities, specifying start and end dates, responsible individuals, and required tools/materials.</li> <li>- Addressing the interrelationships between different methods and techniques within methodologies like object-oriented design.</li> </ul> <p>Planning and Management Activities:</p> <p>1. Project Planning Responsibilities:</p> <ul style="list-style-type: none"> <li>- Highlighting the importance of project planning in software project management.</li> <li>- Mentioning key activities during project planning, such as estimation of cost, duration, and effort.</li> </ul> <p>2. Scheduling and Staffing:</p> <ul style="list-style-type: none"> <li>- Describing how schedules for manpower and resources are developed based on estimations of effort and duration.</li> <li>- Emphasizing the significance of staff organization and</li> </ul>		
--	---	--	--

staffing plans in project management.

3. Risk Management and Quality Assurance:

- Discussing the activities involved in risk management, including risk identification, analysis, and abatement planning.
- Addressing the increasing focus on quality management responsibilities of project managers, tracking project progress and ensuring quality in intermediate artifacts.

4. Change Management and Miscellaneous Plans:

- Explaining the shift towards actively soliciting customer feedback and incorporating changes throughout the development process.
- Mentioning the necessity of various plans like quality assurance and configuration management plans in software project management.

**Module-5**  
**[Download](#)**

Q. 09	<p>a Explain Quality Management Systems with Principles of BS EN ISO 9001:2000</p> <p>Quality Management Systems with Principles of BS EN ISO 9001:2000</p> <p>Quality Management Systems in BS EN ISO 9001:2000</p> <ul style="list-style-type: none"><li>- Purpose: Ensuring effective quality control and assurance in software projects.</li><li>- Standard: BS EN ISO 9001:2000 (identical to ISO 9001:2000).</li><li>- Key Element: Contractor's quality management system validation.</li><li>- Standards: ISO 9000 series focus on monitoring and control systems to check quality.</li><li>- Certification: Ensures development process quality rather than end-product quality.</li></ul> <p>Principles of BS EN ISO 9001:2000</p> <ul style="list-style-type: none"><li>- Continuous Improvement: Emphasizes ongoing enhancement of processes.</li><li>- Factual Evidence: Decision-making based on factual data. -</li><li>Relationships with Suppliers: Building mutually beneficial relationships for quality.</li></ul> <p>Activities to Implement Principles</p> <ol style="list-style-type: none"><li>1. Determine customer needs and expectations.</li><li>2. Establish a quality policy defining quality objectives.</li><li>3. Design processes to meet quality objectives.</li><li>4. Allocate responsibilities for process compliance.</li><li>5. Ensure resources availability for proper process execution.</li><li>6. Measure process effectiveness and efficiency.</li><li>7. Gather measurements.</li><li>8. Identify and address discrepancies between actual and target values.</li><li>9. Analyze and eliminate causes of discrepancies for continual improvement.</li></ol> <p>Detailed ISO 9001 Requirements</p> <ul style="list-style-type: none"><li>- Documentation: Objectives, procedures (quality manual), plans, and records essential.</li></ul>	L1	10
-------	--	----	----

	<p>BS EN ISO 9001:2000 QMS Requirements Overview</p> <ul style="list-style-type: none"> <li>- Foundation Principles: Understanding customer requirements, leadership, staff involvement, process focus, system focus.</li> <li>- Applicability: Standards relevant to all types of production, not limited to software development.</li> </ul> <p>Controversies and Criticisms</p> <ul style="list-style-type: none"> <li>- Controversies: Concerns over perceived product quality, expense, and distraction from real quality issues.</li> </ul>		
	<p>b Explain Structured programming and clean-room software development.</p> <p>Structured Programming and Clean-Room Software Development</p> <p>Structured Programming</p> <p>Structured programming emphasizes a procedural structure in software development, involving carefully laid down steps for each process in the software development cycle. It focuses on breaking down complex systems into comprehensible components and sub-components with clear entry and exit points. This method aims to enhance the quality and reliability of software by promoting systematic development practices.</p> <p>Key Points:</p> <ul style="list-style-type: none"> <li>- Structured programming techniques advocate a methodical approach to software development.</li> <li>- It involves breaking down complex systems into manageable components.</li> <li>- The emphasis is on maintaining a clear procedural structure throughout the development cycle.</li> <li>- The approach aims to improve software quality by ensuring systematic and structured development processes.</li> </ul> <p>Clean-Room Software Development</p> <p>Clean-room software development is an advanced methodology that utilizes mathematical verification techniques to ensure software correctness and reliability. It involves three distinct teams – specification, development, and certification – each with specific roles in the software creation process. Clean-room development focuses on incremental development, rigorous</p>	L2	10

		<p>testing, and verification using formal mathematical techniques to enhance software quality and reliability.</p> <p>Key Points:</p> <ul style="list-style-type: none"> <li>- Clean-room development employs mathematical verification techniques to ensure software correctness.</li> <li>- It consists of three specialized teams, each with defined roles in the software development process.</li> <li>- Incremental development is a fundamental aspect of clean-room software development.</li> <li>- Rigorous testing and verification processes are conducted to enhance software quality and reliability.</li> </ul> <p>Comparison</p> <p>Structured programming provides a systematic approach to software development, emphasizing procedural clarity and component breakdown. Clean-room software development takes this further by incorporating formal mathematical verification techniques and specialized teams to ensure software correctness and reliability through incremental development.</p> <p>Overall, both methodologies aim to enhance software quality and reliability through structured processes and rigorous testing, with clean-room development adding a layer of formal mathematical verification to further improve software correctness.</p>		
OR				
Q. 10	a	<p>Identify how Automation testing is preferred over manual testing, with different tools used for Automation Testing.</p> <p>Automation Testing vs. Manual Testing: Advantages and Tools</p> <p>Advantages of Automation Testing over Manual Testing:</p> <ol style="list-style-type: none"> <li>1. Sophisticated Test Case Design : Automation allows for the deployment of more sophisticated test case design techniques.</li> <li>2. Time and Cost Efficiency : Automation significantly reduces time and effort in testing large and complex software products.</li> <li>3. Reliable Results : Automated test results are more reliable and eliminate human errors during testing.</li> <li>4. Regression Testing Simplification : Automation simplifies</li> </ol>	L2	10

the repeated running of test cases, especially for regression testing after changes or error corrections.

5. **Cost and Time Reduction** : Automation tools promise substantial cost and time reduction in testing and maintenance phases.

#### Tools for Automation Testing:

1. **Capture and Playback Tools** : Record test cases during manual execution for easy replay in subsequent tests.
2. **Model-based Testing** : Utilizes models, such as state models and activity models, to generate tests covering the program's state space.
3. **Automated Test Script Tools** : Drive automated tests using scripts, providing input and recording output for easy rerunning.
4. **Random Input Testing Tools** : Generate random test values to cover the input space of the unit under test, focusing on crashing the unit.
5. **Test Automation** : Generic term for automating test processes, reducing human effort and improving testing thoroughness.

#### Automation Testing Strategies:

- **Test Case Design** : The effectiveness of testing depends on the exact test case design strategy used.
- **Maintenance\*\***: Effort required to determine and remove invalid test cases or modify test input and output data when changes occur.
- **Thoroughness\*\***: Automation allows more testing with a large number of test cases within a short time without significant cost overhead.

#### Challenges and Considerations:

- **Test Maintenance** : Updating tests when the unit under test changes can be costly.
- **Test Script Accuracy** : Ensuring test script accuracy requires significant effort.
- **Defect Detection** : Random input testing tools may find defects that crash the unit under test but not all incorrect results.

#### Conclusion:

Automation testing offers several advantages over manual testing, including improved efficiency, reliability, and cost reduction. Various tools cater to different automation testing needs, from





	<p>addressed effectively.</p> <ul style="list-style-type: none"> <li>- Quality plans may be necessary for external client projects to ensure that all quality issues are adequately addressed.</li> </ul> <p>Quality Plans</p> <ul style="list-style-type: none"> <li>- Quality plans outline how standard quality procedures and standards are applied to a project.</li> <li>- They serve as checklists to ensure that all quality issues have been addressed during the planning process.</li> <li>- Quality plan contents may include purpose, references to other documents, management arrangements, documentation, standards, practices, reviews, testing, problem reporting, tools, code control, training, risk management methods.</li> </ul> <p>Testing and Evaluation</p> <ul style="list-style-type: none"> <li>- Methods are needed to assess the likely quality of the final software system during development.</li> <li>- Quality-enhancing techniques focus on testing product deliverables and evaluating the quality of the development processes used.</li> </ul> <p>Further Exercises</p> <ol style="list-style-type: none"> <li>1. Draft quality specifications for a project planning software tool, focusing on usability, reliability, and recoverability features.</li> <li>2. Assess the maintainability of a software module from user and developer management perspectives.</li> <li>3. Discuss the meaningfulness of measurements like the number of error messages produced on the first compilation of a program.</li> </ol>		
--	--	--	--