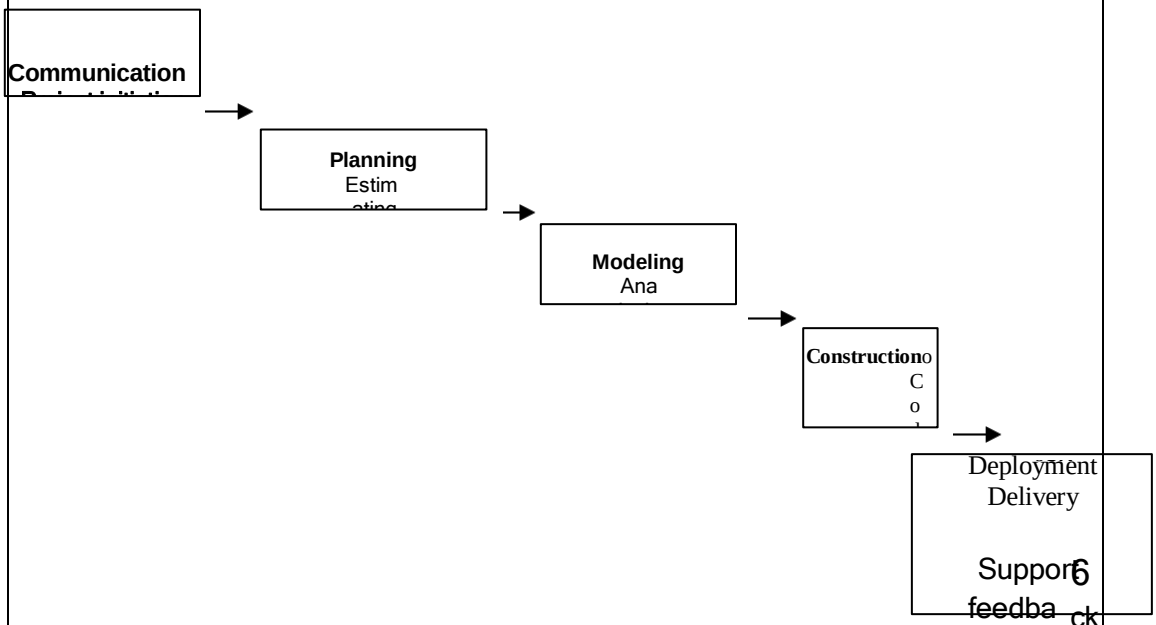
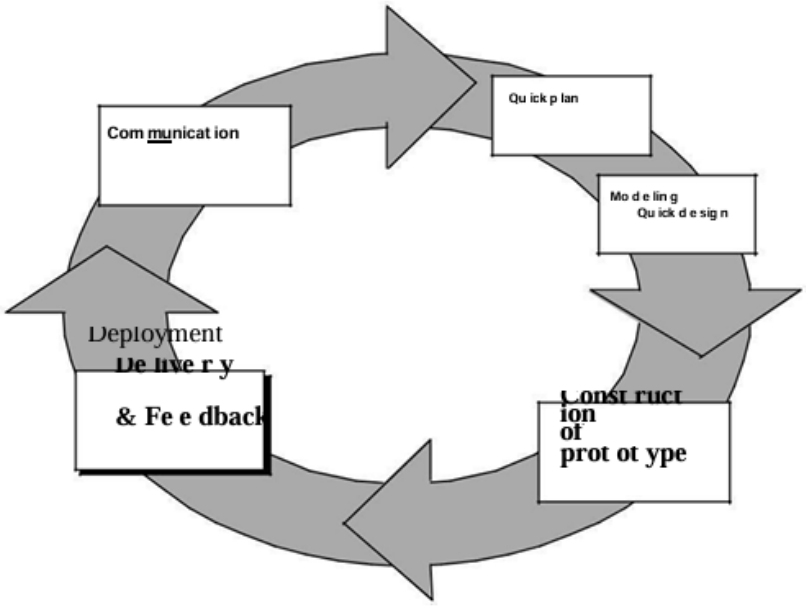
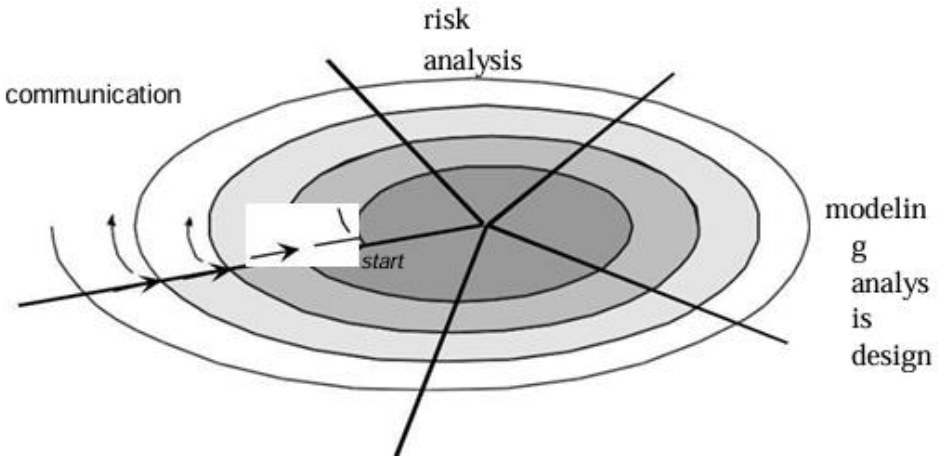
Internal Assessment Test 2 January 2024									
Sub:	**Software Engineering and Project Management**				Sub Code:	**BCS501**	Branch:	AIML/CSE-AIML	
Date :	**7/11/24**	Duration:90 m	Max Marks:	50	Sem :	**V**		OBE	

	Answer any FIVE FULL Questions	Marks	CO	RBT
1	**What is waterfall model?** THE WATERFALL MODEL: The waterfall model, sometimes called the classic life cycle, suggests a systematic sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment. Context: Used when requirements are reasonably well understood. Advantage: It can serve as a useful process model in situations where requirements are fixed and work is to proceed to complete in a linear manner. The problems that are sometimes encountered when the waterfall model is applied are: Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds. It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exist at the beginning of many projects. The customer must have patience. A working version of the programs will not be available until late in the project time-span. If a major blunder is undetected then it can be disastrous until the program is reviewed. 	**10**	**CO1**	**L2**

2.	**List & explain different types of evolutionary process models** Prototype model Spiral model PROTOTYPING: Prototyping is more commonly used as a technique that can be implemented within the context of anyone of the process model. The prototyping paradigm begins with communication. The software engineer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. Prototyping iteration is planned quickly and modeling occurs. The quick design leads to the construction of a prototype. The prototype is deployed and then evaluated by the customer/user. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done. Qu ick p lan Com municat ion Deployment De live r y & Fe e dback Context: Mo d e lin g Qu ick d e sig n Const ruct ion of prot ot ype If a customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements, in such situation prototyping paradigm is best approach. If a developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system then he can go for this prototyping method. Advantages: The prototyping paradigm assists the software engineer and the customer to better understand what is to be built when requirements are fuzzy. The prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to make use of existing program fragments or applies tools. Prototyping can be problematic for the following reasons: The customer sees what appears to be a working version of the software, unaware that the prototype is held together ―with chewing gum and baling wire‖, unaware that in the rush to get it working we haven‘t considered overall software quality or long-term maintainability. When informed that the product must be rebuilt so that high-levels of quality can be maintained, the customer cries foul and demands that ―a few fixes‖ be applied to make the prototype a working product. Too often, software development relents. The developer often makes implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability. After a time, the developer may become comfortable with these choices and forget all the reasons why they were inappropriate. The less-than-ideal choice has now become an integral part of the system.	**10**	**CO1**	**L3**



The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. The spiral model can be adapted to apply throughout the entire life cycle of an application, from concept development to maintenance. Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are planning scheduling risk analysis communication start deployment delivery produced. feedback modeling analys is design construction code test



Anchor point milestones- a combination of work products and conditions that are attained along the path of the spiral- are noted for each evolutionary pass. The first circuit around the spiral might result in the development of product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the

	planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software. It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world. The first circuit around the spiral might represent a ―concept development project‖ which starts at the core of the spiral and continues for multiple iterations until concept development is complete. If the concept is to be developed into an actual product, the process proceeds outward on the spiral and a ―new product development project‖ commences. Later, a circuit around the spiral might be used to represent a ―product enhancement project.‖ In essence, the spiral, when characterized in this way, remains operative until the software is retired. Context: The spiral model can be adopted to apply throughout the entire life cycle of an application, from concept development to maintenance. Advantages: It provides the potential for rapid development of increasingly more complete versions of the software. The spiral model is a realistic approach to the development of large-scale systems and software. The spiral model uses prototyping as a risk reduction mechanism but, more importantly enables the developer to apply the prototyping approach at any stage in the evolution of the product. Draw Backs: The spiral model is not a panacea. It may be difficult to convince customers that the evolutionary approach is controllable. It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.			
3	**Define Requirement Engineering. Explain its Distinct tasks.** Requirements engineering builds a bridge to design and construction. Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system. It encompasses seven distinct tasks: inception, elicitation, elaboration, negotiation, specification, validation, and management. a) Inception. In general, most projects begin when a business need is identified or a potential new market or service is discovered. Stakeholders from the business community define a business case for the idea, try to identify the breadth and depth of the market, do a rough feasibility analysis, and identify a working description of the project's scope. At project inception, you establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team. b)Elicitation. Ask the customer, what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis. A number of problems that are encountered as elicitation occurs. • Problems of scope. The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives. • Problems of understanding. The customers/users are not completely sure of what is	10	**CO2**	**L2**

	needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be "obvious," specify requirements that conflict with the needs of other customers/users, or specify requirements that are ambiguous or un testable. • Problems of volatility. The requirements change over time. To help overcome these problems, you must approach requirements gathering in an organized manner. c) Elaboration. The information obtained from the customer during inception and elicitation is expanded and refined during elaboration. This task focuses on developing a refined requirements model that identifies various aspects of software function, behavior, and information. Elaboration is driven by the creation and refinement of user scenarios that describe how the end user (and other actors) will interact with the system. Each user scenario is parsed to extract analysis classes—business domain entities that are visible to the end user. The attributes of each analysis class are defined, and the services that are required by each class are identified. The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced. d)Negotiation. It usual for customers, to given limited business resources. It's also relatively common for different customers or users to propose conflicting requirements, arguing that their version is "essential for our special needs." You have to reconcile these conflicts through a process of negotiation. Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority. Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction. e) Specification. Specification means different things to different people. A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these. Some suggest that a "standard template" should be developed and used for a specifcation, arguing that this leads to requirements that are presented in a consistent and therefore more understandable manner. However, it is sometimes necessary to remain flexible when a specification is to be developed. For large systems, a written document, combining natural language descriptions and graphical models may be the best approach.			
4	**Analyze the various approaches used in requirement modelling** Elements of the Requirements Model: There are many different ways to look at the requirements for a computer-based system. Different modes of representation force you to consider requirements from different viewpoints—an approach that has a higher probability of uncovering omissions, inconsistencies, and ambiguity. **Scenario-based elements:** The system is described from the user's point of view using a scenario-based approach. For example, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases. Scenario-based elements of the requirements model are often the first part of the model that is developed. Three levels of elaboration are shown, culminating in a scenario-based representation. **Class-based elements:** Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system. These objects are categorized into classes—a collection of things that have similar attributes and common behaviors.	**10**	**CO2**	**L2**

	Behavioral elements: The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied. Therefore, the requirements model must provide modeling elements that depict behavior. The state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state. A state is any externally observable mode of behavior. In addition, the state diagram indicates actions taken as a consequence of a particular event. **Flow-oriented elements:** Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.			
5	**Construct an explanation of the principles of agility.** Agility Principles: The Agile Alliance defines 12 agility principles for those who want to achieve agility: 1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. 2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. 3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. 4. Business people and developers must work together daily throughout the project. 5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. 6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. 7. Working software is the primary measure of progress. 8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. 9. Continuous attention to technical excellence and good design enhances agility. 10. Simplicity—the art of maximizing the amount of work not done—is essential. 11. The best architectures, requirements, and designs emerge from self–organizing teams. 12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	**10**	**CO3**	**L3**
6	Explain the process of Extreme Programming XP Planning ● Begins with the creation of "user stories" ● Agile team assesses each story and assigns a cost ● Stories are grouped to for a deliverable increment ● A commitment is made on delivery date	**10**	**CO3**	**L2**

- After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

XP Design

- Follows the KIS principle
- Encourage the use of CRC cards (see Chapter 8)
- For difficult design problems, suggests the creation of "spike solutions"—a design prototype
- Encourages "refactoring"—an iterative refinement of the internal program design

XP Coding

- Recommends the construction of a unit test for a store *before* coding commences
- Encourages "pair programming"

XP Testing

- All unit tests are executed daily
- "Acceptance tests" are defined by the customer and excuted to assess customer visible functionality