


Solution with scheme-Model Answer

Prof.Lynsha Helena Pratheeba HP and Prof.Rajeshwari R										
Internal Assessment Test 1 – September 2025										
Sub	Software Engineering & Project Management					Sub code	BCS501	Branch	ISE, AIML, CSE(AIML), AIDS	
Date	29.09.2025	Duration	90 mins	Max Marks	50	Sem /Sec	V / A, B, C		OBE	
Answer any FIVE FULL Questions								MARKS	CO	RBT
1. a)	Define Process. What are the key framework activities included in a generic software process model?						[6]	CO1	L1	
b)	Discuss David Hooker’s seven principles of software engineering practice.						[4]	CO1	L1	
2. a)	Explain and demonstrate the waterfall model spiral and model with real time examples. State the Limitations of Waterfall model and Spiral Model.						[10]	CO1	L1	
3. a)	How can Class-Responsibility-Collaborator (CRC) modeling be used to identify classes, their roles, and interactions in object-oriented design?						[6]	CO1	L2	
b)	How can requirement negotiation and validation be used to ensure that software requirements are agreed upon and correct?						[4]	CO1	L2	
4. a)	Illustrate the UML use case diagram for a safe home system.						[6]	CO2	L2	
b)	Explain the distinct tasks of requirement engineering.						[4]	CO2	L2	
5. a)	Explain the following: 1. Adaptive Software development 2. SCRUM						[7]	CO3	L2	
b)	What is Feature Driven Development (FDD) and what is its main purpose in software engineering?						[3]	CO3	L1	
6. a)	Define Extreme Programming. What are the different key strategies used for Extreme Programming approaches to software development?						[7]	CO3	L1	
b)	What is agility in software development, and how does it relate to the cost of change? (include a diagram).						[3]	CO3	L2	

1. a) **Define Process. What are the key framework activities included in a generic software process model?**

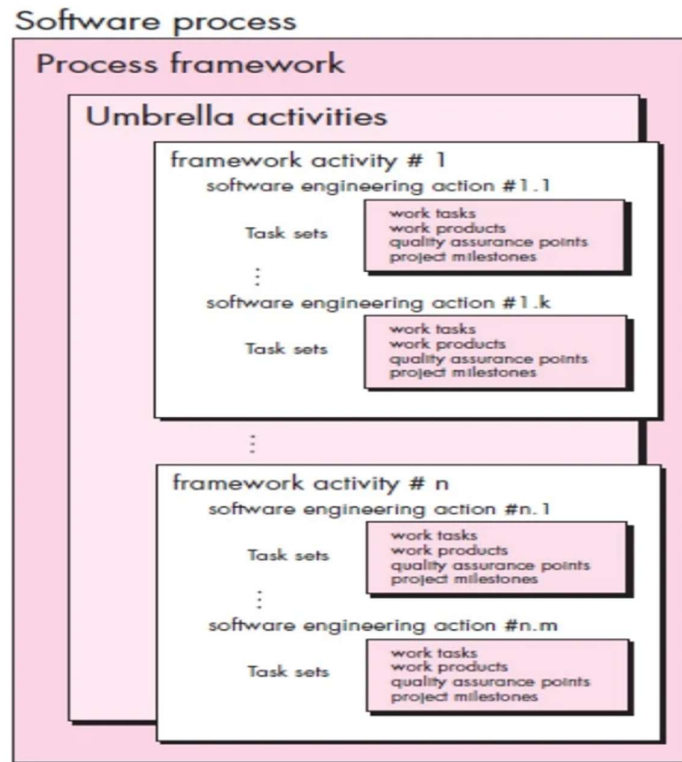
[6]

Diagram with Explanation [each 1M]

Process:

A process is a **collection of activities, actions, and tasks** that are performed when some work product is to be created.

Software Process Framework



Software Process

When you work to build a product or system, it's important to go through a series of predictable steps—a **road map** that helps you create a timely, high-quality result. The road map that you follow is called a “software process.”

Generic Process Model

The generic process model is **an abstraction of the software development process**. It is used in most software since it provides a base for them.

Process Framework

A generic process framework for software engineering defines five framework activities - **communication, planning, modeling, construction, and deployment**.

Umbrella activities

Software engineering process framework activities are complemented by a number of umbrella activities include:

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

b) **Discuss David Hooker's seven principles of software engineering practice. (each 0.5 M with exp)**

[4]

The dictionary defines the word principle as “an important underlying law or assumption required in a system of thought

1. The Reason It All Exists
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think!

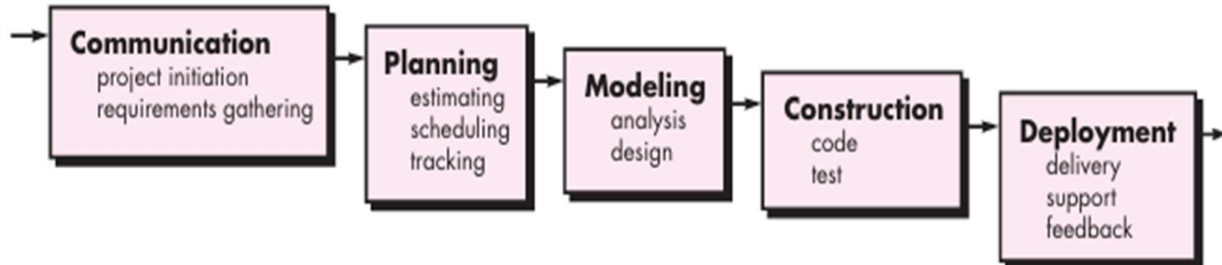
2. a) **Explain and demonstrate the waterfall model and spiral model with real time examples. State the Limitations of Waterfall model and Spiral Model.**

[10]

Waterfall [4] + Spiral [4] + Example [2]

Waterfall Model

The waterfall model, also 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.

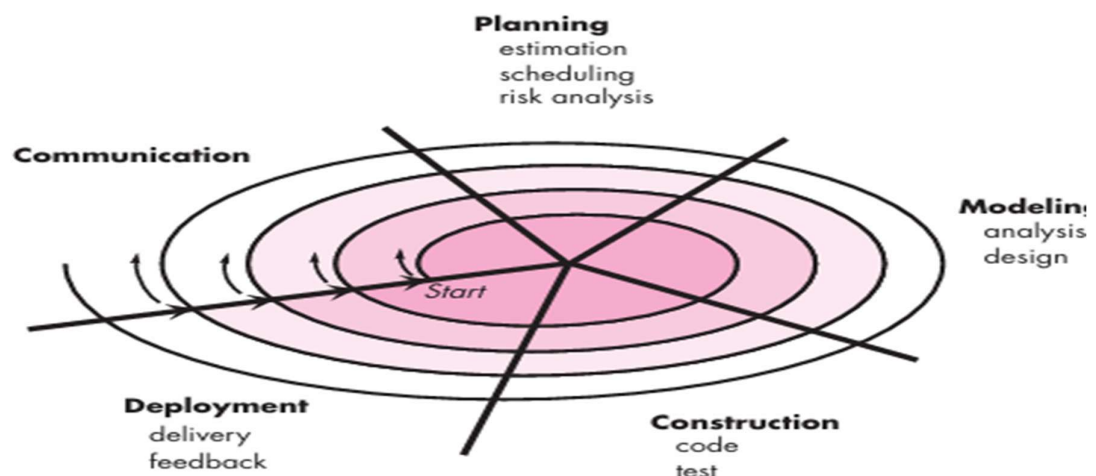


Drawbacks:

- No reverse direction
- Only requirements are clear
- Low budget projects
- No customer involvement
- Will not allow any changes

Spiral Model:

The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems. It has two main distinguishing features. One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.



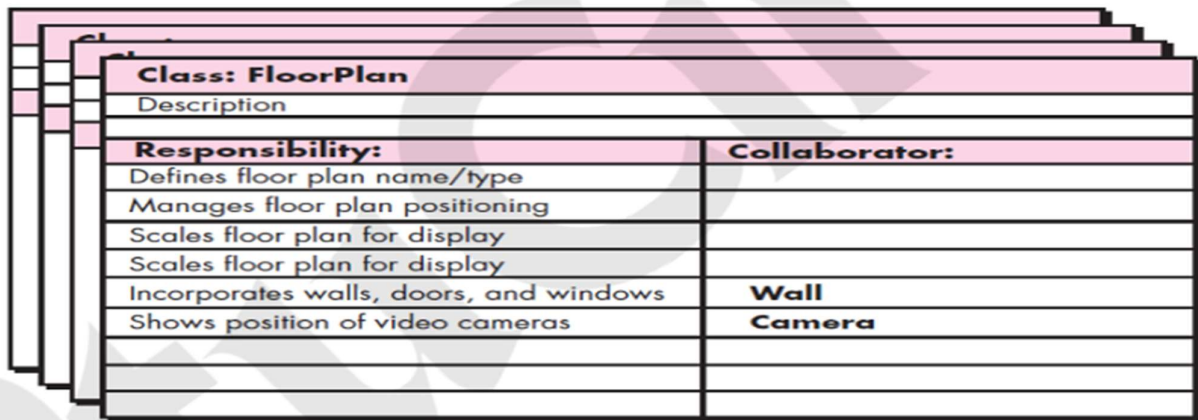
3 a) **How can Class-Responsibility-Collaborator (CRC) modeling be used to identify classes, their roles, and interactions in object-oriented design? CRC (3 marks) + diagram (3 marks)**

[6]

Class-responsibility-collaborator (CRC) modeling provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.

A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card, you write the name of the class. In the body of the card, you list the class responsibilities on the left and the collaborators on the right.

The CRC model may make use of actual or virtual index cards. The intent is to develop an organized representation of classes. Responsibilities are the attributes and operations that are relevant for the class and Collaborators are those classes that are required to provide a class with the information needed to complete a responsibility.



Class: FloorPlan	
Description	
Responsibility:	Collaborator:
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Scales floor plan for display	
Incorporates walls, doors, and windows	Wall
Shows position of video cameras	Camera

Fig : A CRC model index card

It consists of three sections: Class name – Responsibilities - Collaborators

Classes: The taxonomy of class types can be extended by considering the following categories:

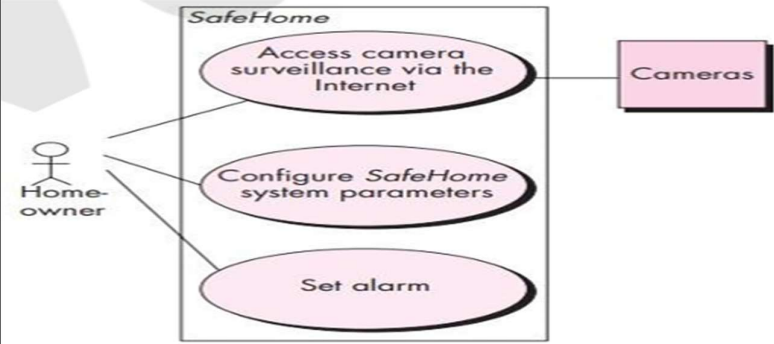
- Entity classes
 - * model or business classes
 - * extracted directly from the statement of the problem
- Boundary classes
 - * used to create the interface that the user sees and interacts with as the software is used
 - * designed with the responsibility of managing the way entity objects are represented to users
- Controller
 - * manage a “unit of work” from start to finish
 - * designed to manage
 - (1) the creation or update of entity objects,
 - (2) the instantiation of boundary objects as they obtain information from entity objects,
 - (3) complex communication between sets of objects,
 - (4) validation of data communicated between objects or between the user and the application

Responsibilities: Wirfs-Brock and her colleagues suggest five guidelines

- System intelligence should be distributed across classes to best address the needs of the problem
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class.
- Information about one thing should be localized with a single class, not distributed across multiple classes
- Responsibilities should be shared among related classes, when appropriate

Collaborations: Classes fulfill their responsibilities in one of two ways:

- A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
- A class can collaborate with other classes
 - Information and the behavior related to it should reside within the same class.
 - Information about one thing should be localized with a single class, not distributed across multiple classes
 - Responsibilities should be shared among related classes, when appropriate

b)	<p>How can requirement negotiation and validation be used to ensure that software requirements are agreed upon are correct? Negotiation [2 marks] + Validation [2 marks]</p>	[4]
	<p>Requirement negotiation and validation are essential processes that ensure software requirements are both agreed upon by all stakeholders and are correct, reducing the risk of costly errors later in development.</p> <p>Requirement Negotiation</p> <ul style="list-style-type: none"> • The negotiation process involves stakeholders such as developers, users, customers, and managers discussing their goals, priorities, and constraint. • Major activities include identifying stakeholders, establishing their “win conditions,” and reconciling conflicting requirements into a set of mutually acceptable agreements. • Techniques like structured workshops and methods such as WinWin and EasyWinWin ensure consensus and document the rationale for each requirement. • This process helps balance functionality, cost, schedules, and performance, solution space is iteratively narrowed down until all critical parties agree, and formal documentation of agreed requirements is produced. <p>Requirement Validation</p> <ul style="list-style-type: none"> • Validation checks requirements for accuracy, completeness, consistency, and alignment with overall objectives. • Steps involve inspections, stakeholder reviews, prototyping, and analysis to find issues like ambiguity, incomplete requirements, or conflicts. • Validation ensures each requirement is testable, feasible, well-defined, and attributed to its source. • Refinement follows validation, where stakeholders collaborate to correct errors and improve clarity or completeness, ultimately leading to requirements set that is both achievable and meets business and technical needs 	
4 a)	<p>Illustrate the UML use case diagram for a safe home system. Explanation [3marks] + Fig [3 marks]</p>	[6]
	<p>When a use case involves a critical activity or describes a complex set of steps with a significant number of exceptions, a more formal approach may be desirable. The typical outline for formal use cases can be in following manner:</p> <ul style="list-style-type: none"> • The goal in context identifies the overall scope of the use case. • The precondition describes what is known to be true before the use case is initiated. • The trigger identifies the event or condition that “gets the use case started” • The scenario lists the specific actions that are required by the actor and the appropriate system responses. • Exceptions identify the situations uncovered as the preliminary use case is refined  <p>The typical outline for formal use cases can be in following manner: Primary actor – House owner</p> <ul style="list-style-type: none"> • The goal in context identifies the overall scope of the use case. – view camera remotely • The precondition describes what is known to be true before the use case is initiated. – sys config • The trigger identifies the event or condition that “gets the use case started” – selects camera • The scenario lists the specific actions that are required by the actor and the appropriate system responses. • Exceptions identify the situations uncovered as the preliminary use case is refined 	

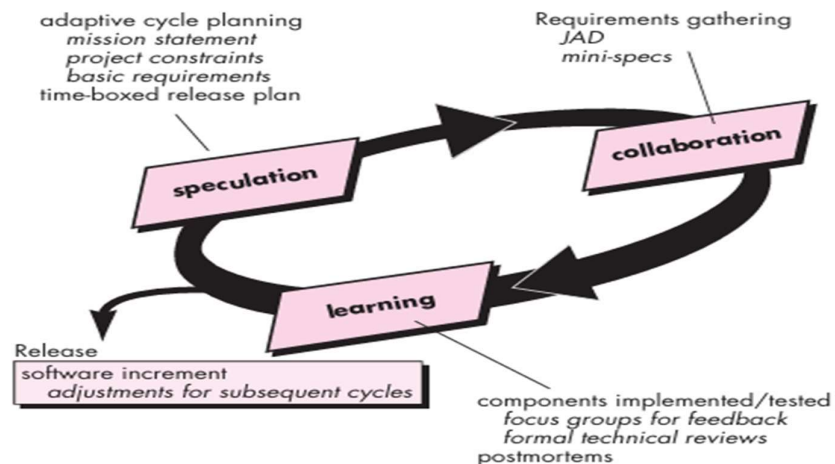
b)	<p>Explain the distinct tasks of requirement engineering.</p> <p>Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system.</p> <p>It encompasses seven distinct tasks: inception, elicitation, elaboration, negotiation, specification, validation, and management.</p> <p>1. Inception: It establishes 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.</p> <p>2. Elicitation: In this stage, proper information is extracted to prepare and document the requirements. It certainly seems simple enough ask the customer, the users, and others 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.</p> <p>3. 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.</p> <p>4. Negotiation: To negotiate the requirements of a system to be developed, it is necessary to identify conflicts and to resolve those conflicts. 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.</p> <p>5. Specification: The term 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.</p> <p>6. Validation: Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product. The primary requirements validation mechanism is the technical review. The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking for errors in content or interpretation, areas where clarification may be required, missing information, inconsistencies, conflicting requirements, or unrealistic requirements.</p> <p>7. Requirements management. Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system. It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project. Many of these activities are identical to the software configuration management (SCM) techniques. These tasks start with the identification and assign a unique identifier to each of the requirement. After finalizing the requirement traceability table is developed.</p>	[4]
5a)	<p>Explain the following:</p> <ol style="list-style-type: none"> 1. Adaptive Software development [3 marks] 2. SCRUM [4 marks] <p>Adaptive Software development:</p> <p>Adaptive Software Development (ASD) has been proposed by Jim Highsmith as a technique for building complex software and systems. The philosophical underpinnings of ASD focus on human collaboration and team self-organization.</p> <p>During speculation, the project is initiated and adaptive cycle planning is conducted.</p> <p>Adaptive cycle planning uses project initiation information the customer's mission statement, project constraints, and basic requirements to define the set of release cycles (software increments) that will be required for the project.</p> <p>Motivated people use collaboration in a way that multiplies their talent and creative output beyond their absolute numbers. This approach is a recurring theme in all agile methods. But collaboration is not easy. It encompasses communication and teamwork, but it also emphasizes individualism, because individual creativity plays an important role in collaborative thinking. It is, above all, a</p>	[7]

matter of trust. People working together must trust one another to

- (1) criticize without animosity,
- (2) assist without resentment,
- (3) work as hard as or harder than they do,
- (4) have the skill set to contribute to the work at hand, and
- (5) communicate problems or concerns in a way that leads to effective action. As members of an ASD team begin to develop the components that are part of an adaptive cycle, the emphasis is on “learning” as much as it is on progress toward a completed cycle.

FIGURE 3.3

Adaptive software development



SCRUM:

Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements, and business criticality. Each of these process patterns defines a set of development actions:

Backlog - a prioritized list of project requirements or features that provide business value for the customer. Items can be added to the backlog at any time. The product manager assesses the backlog and updates priorities as required.

Sprints - consist of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time-box

Scrum meetings - are short meetings held daily by the Scrum team. Three key questions are asked and answered by all team members

o What did you do since the last team meeting?

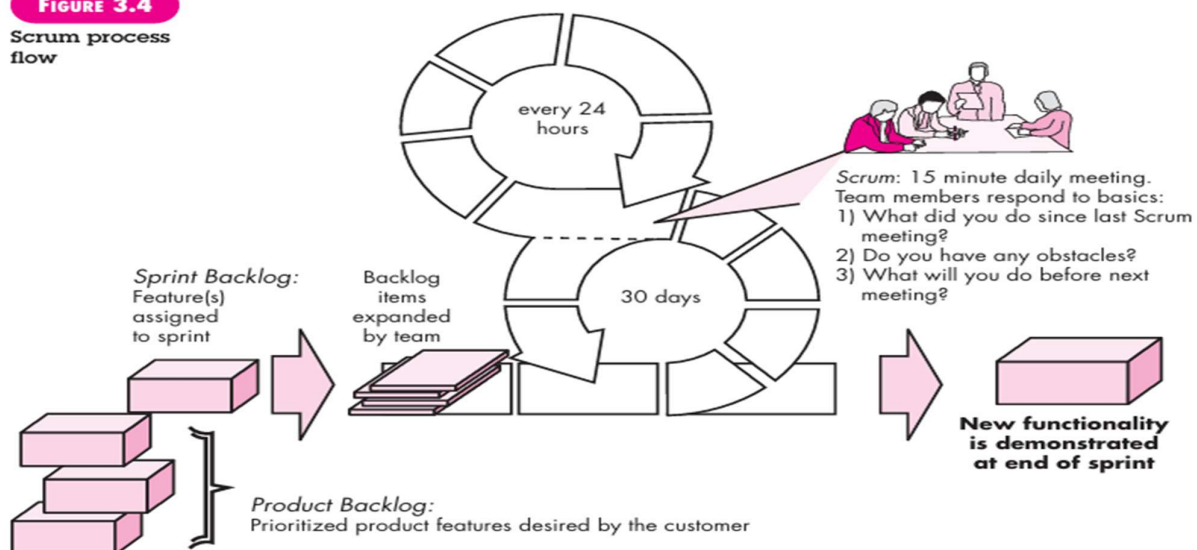
o What obstacles are you encountering?

o What do you plan to accomplish by the next team meeting?

Demos - deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer

FIGURE 3.4

Scrum process flow



b) **What is Feature Driven Development (FDD) and what is its main purpose in software engineering? [Explanation with diagram – 3 marks]** [3]

FDD emphasizes software quality assurance activities by

- encouraging an incremental development strategy
- the use of design and code inspections
- the application of software quality assurance audits, the collection of metrics, and the use of patterns

The emphasis on the definition of features provides the following benefits:

- features are small blocks of deliverable functionality, users can describe them more easily
- features can be organized grouping
- deliverable software increment
- design and code representations are easy to inspect as the feature is small

Project planning, scheduling, and tracking Hierarchy

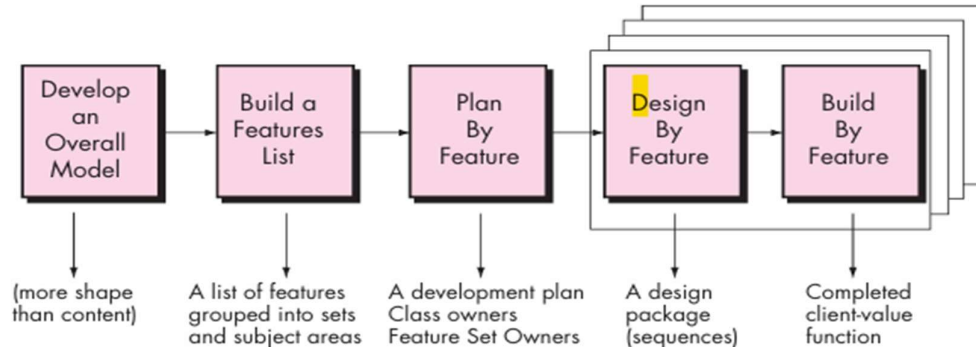
Coad and his colleagues suggest the following template for defining a feature:

<action> the <result> <by|for| of |to> a(n) <object>

Where an <object> is “a person, place or thing.

FIGURE 3.5

Feature Driven Development [Coad99] (with permission)



6 a) **Define Extreme Programming. What are the different key strategies used for Extreme Programming approaches to software development?** [7]

Extreme Programming (XP), the most widely used approach to agile software development, emphasizes business results first and takes an incremental, get-something-started approach to building the product, using continual testing and revision.

XP Values:

Beck defines a set of five values that establish a foundation for all work performed as part of XP

- Communication - communication between software engineers and other stakeholders
- Simplicity - create a simple design that can be easily implemented in code
- Feedback - derived from customer, software or team members
- Courage - discipline among team members to maintain sustainable team work
- Respect - mutual respect among teammates for the supportive and productive environment

The XP Process:

FIGURE 3.2
The Extreme Programming process

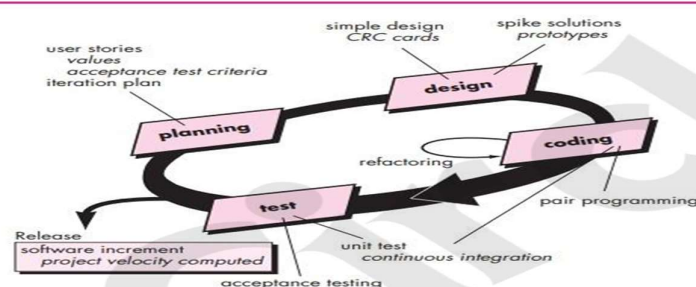


Fig : The Extreme Programming process

Extreme Programming uses an object-oriented approach as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing.

1. XP Planning
2. XP Design
3. XP Design
4. XP Testing

Industrial XP:

IXP is an organic evolution of XP. It is imbued with XP's minimalist, customer-centric, test-driven spirit. IXP differs most from the original XP in its greater inclusion of management customers, and its upgraded technical practices.

IXP incorporates six new practices

1. Readiness assessment
2. Project Community
3. Project Chartering
4. Test Driven Management
5. Retrospectives
6. Continuous learning

The XP Debate:

All new process models and methods spur worthwhile discussion and, in some instances, heated debate. Among the issues that continue to trouble some critics of XP are:

- Requirements volatility: customer is an active member of XP team, changes to requirements are requested informally and frequently.
- Conflicting customer needs: different customers' needs need to be assimilated by the team.
- Requirements are expressed informally: Use stories and acceptance tests are the only explicit manifestation of requirements.
- Lack of formal design: XP deemphasizes the need for architectural design.

b) **What is agility in software development, and how does it relate to the cost of change? (include a diagram).**

[3]

- In conventional software development, the cost of change increases non linearly as a project progress
- An agile process reduces the cost of change because software is released in increments and change can be better controlled within increment
- A well-designed agile process may “flatten” the cost of change curve by coupling incremental delivery with agile practices such as continuous unit testing and pair programming

FIGURE 3.1
Change costs
as a function
of time in
development

note:
“Agility is dynamic,
content specific,
aggressively
change embracing

