Question-1

In a Scrum-based software development project, the product Owner has defined the following user stories for an e-commerce application: As a user, I want to log in securely so that I can access my account. As a user, I want to search for products by category to find items easily.

(i) Create a product backlog for these user stories by breaking them into tasks.

(ii) Describe how the development team can prioritize these user stories during a Sprint planning meeting, considering value to the customer and technical feasibility.

(iii) Illustrate how these tasks will be tracked using a Scrum board. Use proper terms like "To Do", "In Progress", and "Done".

Ans to the Que : NO : 1

* Product Backlog (Tasks Breakdown)

(1) User login Feature.

- Design login UI
- Implement authentication (Email/Password, OAuth)
- Secure user data with encryption.
- Implement session management.
- Write unit and integration tests.

(2) Product search by Category.

- Design search UI
- Implement backend filtering logic.
- Optimize database queries for performance
- Implement search result pagination.
- Write unit and integration tests.

* Sprint Planning Prioritization.

• Value to customer: Login is critical for account access, search improves user experience.

• Technical Fesibility: Login involves security risks but is straightforward, search requires efficient indexing.

• Priority Order: Login first (essential security) then search (enhanced UX)

* Scrum Board Tracking.

• To Do: Tasks not started yet.

• In Progress: Active development tasks.

• Done: Completed, tested and approved tasks

Example-

Task	To Do	In Progress	Done
Design login UI	✓		
Implement authentication		✓	
Secure User Data			✓

Question - 2

A software development team is about to start a Project for a new innovative product. The project has several high-risk components due to its novelty and there's uncertainty regarding the client's future needs. The client is open to iterative changes, but the team must ensure that the software envolves in a manageable, cost-effective way. Considering the high risks and the evolving nature of the client's needs, discuss how the Spiral, Agile and Extreme methodologies address risk management and adaptability. Which methodologies would be the most suitable for a project with significant risk and evolving requirements, and why?

Ans to the Que: No: 02

Comparison of Spiral, Agile and Extreme Programming (XP) for Risk and Adaptability –

1. Spiral Model –

(i) Risk management – Explicity Focuses on risk assessment in each iteration.

(ii) Adaptability – Allows changes but follows a structured approch.

(iii) Suitability – Best for projects with high risks and evolving needs but requires extensive planning.

2. Agile Methodology –

(i) Risk management – Mitigates risks through continuous feedback and iterative releases.

(ii) Adaptability – Highly flexible, embraces changing requirements.

(iii) Suitability – Ideal for projects with evolving client needs and moderate risks.

3. Extreme Programming (XP) -

(i) Risk management - Handles risks through continuous testing and close client collaboration.

(ii) Adaptability - Rapid adaptability with Frequent releases and feedback.

(iii) Suitability - Best for small, fast-moving teams needing high adaptability.

Best choice: Agile.

Agile balances risk management and adaptability effectively, making it the best fit for high-risk evolving projects.

Question-3

A company is working on two different projects. Project A has well-defined requirements and a strict deadline, while project B has evolving requirements with an uncertain timeline and continuous customer feedback. Both project involve high stakes, and the team must decide which development methodology to use.

Compare and contrast the waterfall, Agile, Extrem and spiral development models. Based on the characteristics of both projects (Project A and Project B), which methodology would best suit each? Support your answer with a detailed analysis of how each methodology would addres the specific needs of the projects, considering factors such as predictability, customer collabora -tion, and risk management.

Ans to the Que : No: 3

Comparison of Development models –

Model	Characteristics	Best for	Predict-ability	Customer Collaboration	Risk management
Waterfall	Linear, sequential. Fixed scope	well-defined projects	High	Low	Low
Agile	Iterative, adaptive, customer-focused	Dynamic Projects	Low	High	High
Extreme Programming (XP)	continuous testing, rapid iteration, close collaboration	Fast-changing high-risk projects	Low	very high	Medium
Spiral	Risk-driven, iterative with risk assessment.	High risk complex projects	Medium	Medium	Very high

Best Fit for Each Project.

1) Project A (well-defined, strick deadline) → Waterfal
   - clear requirements suit a structured approch.
   - Predictability ensures timely delivery.
   - low need for continuous customer feedback.

2) Project B (Evolving requirements, uncertain timeline) ⟶ Agile or Spiral.
   - Agile – ~~Headtir~~ Handles evolving needs through iterations and customer feedback.
   - Spiral – Ideal if risk assesment is critical due to high stakes.

For flexibility and customer involvement, Agile is preferable, If risk management is a priority, Spir is better.

Question-4

Explain the principles of software engineering ethics, highlighting the issues related to professional responsibility. Discuss how the ACM/IEEE code of Ethics guides ethical decision-making in software engineering practices.

Ans to the Que: No: 04

Principles of Software Engineering Ethics and Professional Responsibility

Software engineering ethics focus on integrity, fairness and responsibility in developing reliable, safe and user-centric software, Key issues in professional responsibity include:

1) Public Interest - Prioritizing user safety, privacy and societal well-being.

2) Confidentiality - Protecting sensitive data.

3) Competence - Maintaining skills and delivering quality work.

4) Accountability - Taking responsibility for errors and decisions.

5) Honesty - Avoiding deception and misrepresentation

Role of ACM/IEEE Code of Ethics -

The ACM/IEEE Code of Ethics provides a framework for ethical decision-making:

1) Public Good First — Prioritize users and society
2) Honest and transparent — Communicate truthfully.
3) Competence and Professionalism - stay updated and deliver quality.
4) Fairness and Inclusion — Avoid discriminati
5) Accountability — Take responsibility for Actio

This code helps engineers navigate ethical dilemmas, ensuring integrity and professionali in software development.

Question-5

Given the story of the Airport Reservation System, identify at least five functional and five non-functional requirements for the system. In your answer, explain how each requirement contributes to the overall performance, usability and security of the system. Consider factors such as performance, user experience and system maintainance in your discussion.

Ans to the Que. No. 5

Functional Requirements

1. Flight Booking and Cancellation — Users can reserve and cancel tickets, ensuring flexibility and efficiency.

2. User Authentication — Secure login with credentials to prevent unauthorized access.

3. Payment Processing — Supports multiple payment methods for seamless transactions.

4. Seat Selection — Allows users to chose seats enchancing user experience.

5. Flight status Updates — Real-time flight status notifications improve reliability.

Non-Functional Requirements

1. Performance and Scalability — System must handle high traffic efficiently to avoid downtime.

2. Security and Data protection — Encryption and secure authentication prevent data breaches.

3. Usability and Accessibility — Intuitive UI with multi-language support enhances user experience.

4. System Availability - 24/7 uptime ensures reliability for global users.

5. Maintainability and Upgradability - Modular design allows easy updates and bug fixes.

Question - 6

Illustrate and explain the V-model of testing phases in a plan-driven software process, detailing the relationships between development activities and corresponding testing activities.

Ans to the Que: NO: 6

The V-model is a software development methodology where development and testing activities are closely linked. It emphasizes verification at every stage.

Phases and Corresponding Testing Activities -

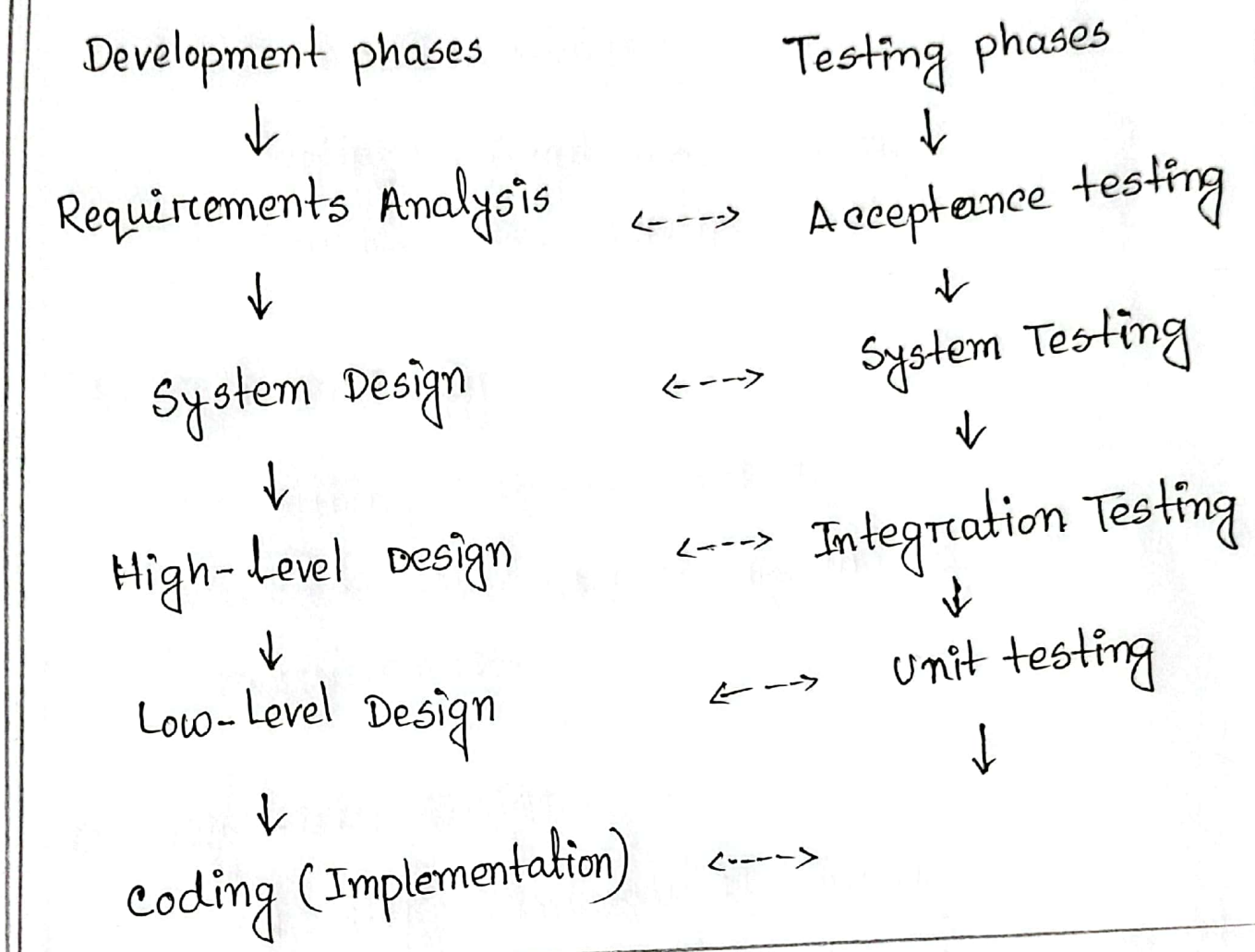
1. Requirements Analysis —
   - Testing :- Acceptance testing
     Ensures the system meets users requirements

2. System Design
   - Testing : System testing
     Verifies the complete system against system requirements

3. High-Level Design
   - Testing : Integration Testing
     Validates interaction between integrated modules.

4. Low-Level Design
   - Testing : Unit Testing
     Tests individual components/ modules for correctness.

Visual Overview –



Here each development phase has a corresponding testing phase to ensure the product meets requirements and works as intended.

The model promotes "early validation" and "verification" of each component as development progresses.

Question – 7

Explain the process of prototype development in software engeneering. Discuss the key stages involved in creating a prototype and how it helps in refining software requirements. Analyze the benefits of using the prototyping model, particularly in terms of user feedback, risk reduction and iterative development.

Ans to the Que: No: 7

Prototype Development in software engineering

Key stages of prototype development –

1. Requirement Gathering – Identify initial requirements based on user needs.

2. Quick Design – Create a basic UI/ functional outline.

3. Prototype Building – Develop a working model with limited functionality.

4. User Evaluation – Gather feedback from users and stakeholders.

5. Refinement – Modify the prototype based on feedback.

6. Finalization – Convert the refined prototype into the final system.

How prototyping Helps in Refining Requirements

1. Exposes missing, ambigous or unclear requirements
2. Allows real-time user feedback, leading to better-defined features.
3. Ensures alignment between user expectations and final software.

Benefits of the prototyping Model

1. User Feedback – Enhances usability and functionality through iterative input.
2. Risk Reduction – Identifies design flaws and requirement mismatches early.
3. Iterative Development – Allows gradual

improvements, ensuring a more refined final product

The prototyping model improves software quality by refining requirements, reducing risks, and integrating user feedback throughout development.

Question – 8

Explain the process improvement cycle in software engineering and describe its key stages. Name and explain some commonly used process metrics, highlighting how they help in monitoring and improving software processes.

Ans to the Que.: No: 8

Process Improvement cycle in software Engineering

The process Improvement cycle enhances software development efficiency and quality. It follows these key stages:

1. Plan – Identify process inefficiencies and

set improvement goals.

2. Do – Implement changes on a small scale.

3. check – Evaluate the impact using process metrics.

4. Act – standardize successful changes and refine processes further.

Common Process Metrics and their Importance

1. Defect Density – Measures defects per KLOC (thousand lines of code) to assess software quality

2. Cycle time – Time taken to complete a process, helps optimize efficiency.

3. Velocity – Amount of work completed in a sprint, tracks development speed.

4. Lead Time – Time from request to delivery, improv responsiveness.

5. Escaped Defects – Bugs found after release, indicates testing effectiveness.

Question-9

Explain the software Engineering Institute capability Maturity Model (SEI CMM) and its five levels of capability and maturity. Analyze how each level contributes to improving the software development process and organizational performance.

Ans to the Que: No: 9

#SEI Capability Maturity Model (CMM) -

The SEI CMM is a framework for improving software processes, defining five maturity levels to enhance development and organizational performance.

Five maturity levels and their contributions—

1) Initial (Level 1) — Ad hoc and chaotic, success depends on individuals, leading to inconsistent resu

2) Repeatable (Level 2) - Basic Project management practices in place; ensures repeatability of successful processes.

3. Defined (Level 3) - standardized processes across the organization, enhances consistency and efficiency.

4) Managed (Level-4) — Quantitative metrics used for monitoring and controlling processes, leads to predictable outcomes.

5) Optimizing (Level 5) — Focus on continuous improvement and innovation, ensures high-quality software and process agility.

Each level builds on the previous one, leading to better project control, higher quality and improved organizational performance.

Question-10

Describe the core principles of agile software development methods. Analyze how these principles are applied in different software development environments and assess the benefits and challenges of using agile methods in various project types and organizational settings.

Ans to the Que: NO: 10

Core Principles of Agile software Developmen

Agile follows the 12 Agile principles (from the Agile Manifesto) with key principles including —

1. Customer Collaboration — Continuous feedback ensures alignment with user needs.

2. Iterative Development — software is built in small, functional increments.

3. Embracing Change — Flexibility to adapt to evolving requirements.

4. Continuous Delivery – Frequent releases for rapid feedback and improvement.

5. Self-Organizing Teams – Empowered teams enhance efficiency and creativity.

6. Simplicity and lean Processes – Prioritizing essential work to maximize value.

Application in Different Environment

1. Startups – Agile fosters rapid prototyping and market responsiveness.

2. Enterprise IT – Scaled Agile (SAFe, LeSS) integrates Agile in large teams.

3. Regulated Industries – Agile combined with compliance frameworks ensures adaptability while maintaining strict controls.

Benefits and challenges of Agile.

Benefits – Faster delivery, higher adaptability improved stakeholder engagement, better product quality.

Challenges — Requires cultural shift, difficult in highly structured organizations, scalability issues in large projects.

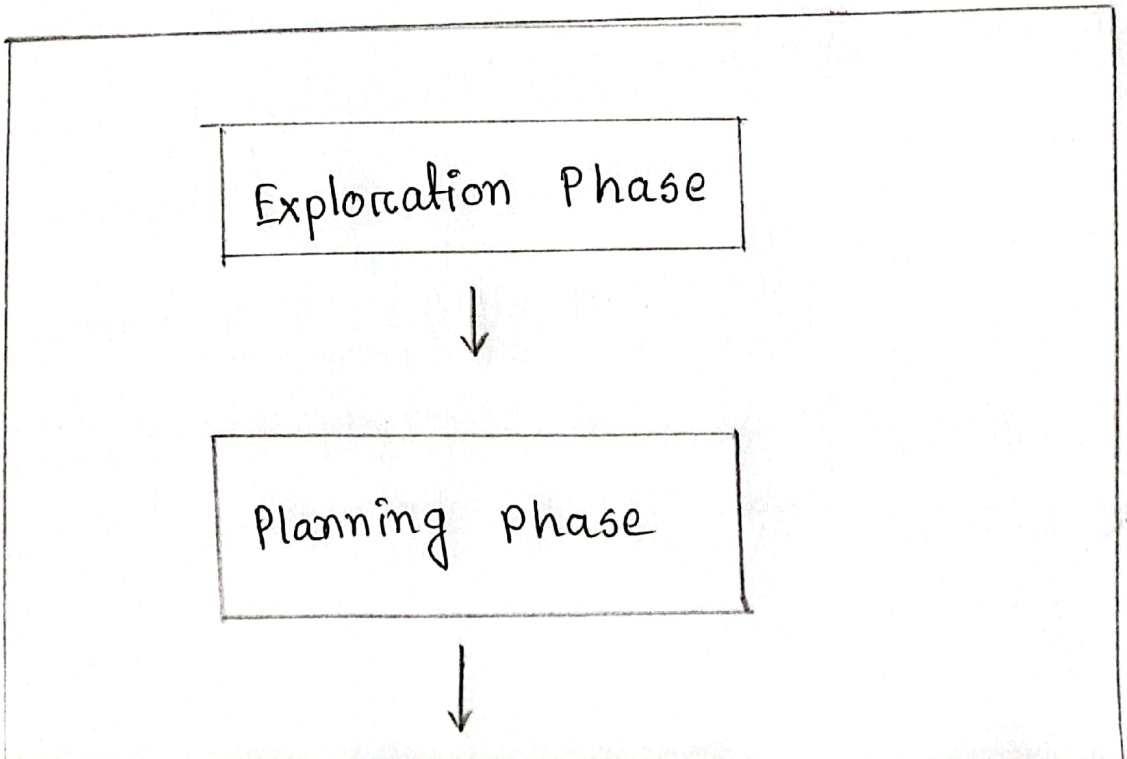
Agile excels in dynamic environments but may face resistance in rigid, highly regulated settings.

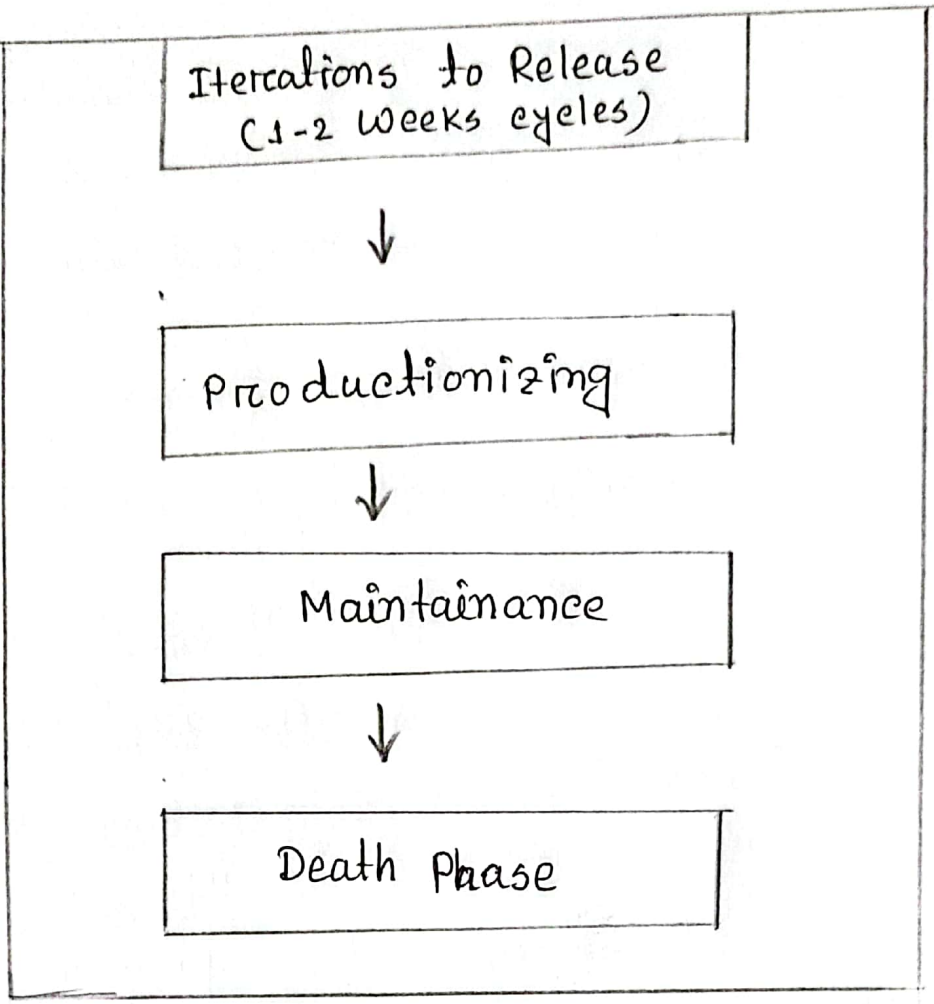
Question-11

Draw the release cycle of (Extreme Programming (XP)) and Explain the influential programming practices.

Ans to the Que: No: 11

Here is a diagram illustrating the Extreme Programming (XP) Release cycle:





Influential Programming Practices in XP

XP promotes several key programming practices to improve code quality and team collaboration:

1) Test-driven Development (TDD) — Writing tests before writing code ensures correctness and maintainability

2) Pair Programming — Two developers work

togathere at one workstation, improving code quality and knowledge sharing.

3) Continuous Integration (CI) – Frequent code integration issues and ensures a working product.

4) Refactoring – Continuous code improvement enhances readability, maintainability and performance.

Question-12

A local library wants to create a digital system to manage its operations, The system will track books, members, and borrowing activities. Each book has attributes like title, author, ISBN and genre. Members have attributes such as name, membership ID, and contact details. When a Member borrows a book, the system records the borrowing date, return due date and return status. The library also wants to maintain a catalog of overdue books and their respective fines Using the senario of a digital library management system, design an Entity-Relationship Diagram (ERD) to represent the entities (e.g. books, members, borrowing activities) and their relationships. Clearly explain the attributes of each entity and how they are interconnected.

Ans to the Que: No: 12

Entity - Relationship Diagram (ERD) for Library Management system :-

Entities and attributes:

1. Book:
   Book_ID(PK), Title, Author, ISBN, Genre.

2. Member:
   Member_ID(PK), Name, Contact_Details, Membership_ID

3. Borrowing:
   Borrowing_ID(PK), Book_ID(FK), Member_ID(FK), Borrow_Date, Due_Date, Return_Date, Return_status.

Relationships:

One-to-many:

1. Book to Borrowing (one book can be borrowed multiple times)
2. Member to Borrowing (One book Member can borrow multiple books)

Additional Considerations:

1. Overdue Books – This can be represented as a view or derived table based on the Borrowing table, filtering for records where Return_Date is later than Due_Date.
2. Fines – Could be a separate entity linked to Borrowing, storing information about fine amount payment status etc.

The ERD provides a basic framework. The actual design may need adjustments based on specific library requirements and the chosen database system.

Question -13

What is called Testing? Differentiate between Validation and Verification.

Ans to the Que: NO: 13

Testing — Testing is the process of evaluating a software system to identify defects, ensure functionality and validate that it meets the specified requirements. It includes various techniques like manual testing, automated testing, unit testing, integration testing and system testing to enhance software quality.

Difference Between Validation and Verification

Aspect	Verification	Validation
Definition	Ensures the product is being built correctly (meets specifications)	Ensures the right product is built (meets user needs)
Focus	Process-oriented (Design, documents, plans)	Product-oriented (actual software functionality)

Aspect	Verification	Validation
Techniques	Reviews, Walkthroughs, inspections.	Testing, UAT (User Acceptance Testing)
Performed in	Early development phases	After coding is complete, before deployment
Example	checking design documents before coding starts.	Running test cases to validate software behavior.

Question - 14

Design a layered architecture model for an online judge system, identifying the key layers (e.g. presentation, application, business logic and data). Explain the responsibilities of each layer and analyze how this architecture ensures scalability, maintainability and efficient performance.

Ans to the Que: No: 14

An online judge system can be designed using a layered architecture consisting of the following layers.—

1. Presentation layer — This layer handles user iteration, such as displaying problems, accepting code submissions, and showing results. It ensures a smoth user experience and communicates with the application layer to process requests.

2. Application layer — It manages the flow of information between the presentation and bussiness logic layers, handling user requests, such as submitting code and retrieving results. It orchestrate processes but doesn't handle complex business logic.

3. Business logic layer — This layer is responsible for processing the code submissions, running the test cases, evaluating the results and applying the rules of the competition. It ensures correctne and accuracy of operations.

4. Data layer — This layer stores persistent data like user profiles, problem sets, submissions and results in a database. It ensures data integrity and efficient querying.

- Scalability — By separating concerns, each layer can be scaled independently (e.g. more application servers for traffic or seperate resource for the evaluation process).

- Maintainability — changes in one layer (e.g. presentation design or business logic) don't affect others, promoting ease of updates.

- Performance — The separation of concerns allows for optimizations at each layer (e.g caching results at the presentation layer or optimizing database queries), ensuring efficient performance.

Question-15

Draw a DFD (Level-0 and Level-1) and UML use case Diagram for a Hospital Management System. A hospital management system is a large system that includes several subsystems or modules that provide various functions. Your UML use case diagram example should show actors and use cases for a hospital's reception.

Purpose: Describe major services (functionality) provided by a hospital's reception.

Ans to the Que: No: 15

DFD Level-0 (context Diagram)

- External Entity : Patient, Doctor Receptionist, Insurance company.
- Process : Hospital Management System.
- Data Flow.
  - (i) Patient → Request for Appointment
  - (ii) Receptionist → schedule Appointment
  - (iii) Doctor → check Appointment
  - (iv) Insurance company → Provide Insurance Details.

DFD Level-1

1. Process : Appointment Scheduling

   (i) Inputs – Patient information, Request for Appointment.

   (ii) Outputs :– Appointment confirmation.

   (iii) Subprocesses —

   - validate Patient Info
   - Check Appointment Availability
   - confirm Appointment.

2. Process ; Patient Registration.

   (i) Inputs — New patient Info

   (ii) Outputs — Registered Patient Details

   (iii) Subprocesses —

   - Enter Personal Info
   - Assign Patient ID.

UML Use Case Diagram for Reception.

1. Actors — (i) Receptionist
   (ii) Patient
   (iii) Doctor.

2. Use cases — (i) Schedule Appointment
   (ii) Register Patient
   (iii) Update Patient Info
   (iv) Provide Appointment confirmation
   (v) Issue ID card.

3. Relationships — (i) Receptionist → Schedule App.
   (ii) Receptionist → Register Patien
   (iii) Patient → Request Appointment
   (iv) Doctor → check Appointment

Question-16

Consider the Scenario below:
The Hospital Reception subsystem or module supports some of the many job duties of a hospital receptionist. The receptionist schedules patient's appointments and admission to the hospital and collects information from the patients upon patient's arrival and/or by phone. The patient who will stay in the hospital ("impatient") should have a bed allowted in a ward. Receptionists might also recieve patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.

Consider the following UML sequence diagram and design the UML daigram for the same scenario. Seq.jpg. Now Draw a UML class diagram for the Scenario diepicted by the sequence Diagram.

Ans to the Que: No: 16

Here is a quick outline of the UML Class Design for the described Hospital Reception subsystem:

1. class: Receptionist

- Attributes:
  (i). name
  (ii) employeeID
  (iii) contactInfo.

- Methods:
  (i) scheduleAppointment()
  (ii) CollectPatientInfo()
  (iii) receivePayment()
  (iv) FileInsuranceClaims()
  (v) fileMedicalReports().

2. class: Patient

- Attributes:
  (i) PatientID
  (II) name
  (III) dob
  (IV) address
  (V) contactInfo
  (VI) insurance Details.

- Methods:
  (i) makePayment()
  (ii) requestAppointment().

3. Class: Appointment
   - Attributes:
     (i) appointmentID
     (ii) datetime
     (iii) patientID
     (iv) status.
   - Methods:
     (i) confirmAppointment()
     (ii) cancelAppointment()

4. Class: MedicalReport
   - Attributes:
     (i) reportID
     (ii) PatientID
     (iii) date
     (iv) reportType
     (v) doctorName
   - Methods:
     (i) generateReport()
     (ii) viewReport()

Question-17

We know that Quality Assurance (QA) is not just Quality control (QC). For example, QA is process-oriented, and QC is product-oriented. Now, suppose you are said to get a straight explanatory descriptio of it along with the differences and Impediments between and the QA and QC.

Ans to the Que: No: 17

QA (Quality Assurance) is focused on improving and ensuring the processes that lead to the final product. It is proactive and aims to prevent defects

QC (Quality control) is focused on identifying and correcting defects in the final product. It is reactive and aims to ensure the product meets the required standards.

Key Differences —

1. Approach — QA is process-driven, QC is product-driven.

2. Focus — QA focuses on prevention, QC focuses on detection.

3. Timing – QA occurs during the process, QC happens after production

4. Goal – QA aims for continuous improvement, QC aims for product conformance.

Impediments –

For QA : Resistance to change, lack of resources, unclear processes.

For QC : Inconsistent standards, over-reliance on final checks, not addressing root causes of defects.

Question-18

Do you think the goal of QA is just to find the bugs as early as possible? The goal of the QA is to find the bugs as early as possible and make sure that get fixed. Quality Assurance was introduced after World War II when the weapons were tested before they came into action. Explain the role of Quality

Assurance (QA) at each phase of the software Development life cycle (SDLC).

Ans to the Que: NO: 18

The goal of QA is not just to find bugs early but to ensure quality throughout the entire process. It focusses on preventing defects, improving processes, and ensuring the product meets standards.

Role of QA in each SDLC phase —

(1) Requirements phase-

(i) Review requirements to ensure clarity, completeness and testability.

(ii) Prevent ambiguities by defining clear acceptance criteria.

(2) Design phase —

(i) Ensure design compliance with requirements.

(ii) Conduct design reviews to identify potential risks.

(3) Development phase-

(i) Perform static analysis and code reviews.

(ii) Create unit tests and ensure developers follow best practices.

4. Testing phase—

(i) Execute test cases (Functional, integration, system)

(ii) Verify bug fixes and ensure traceability of requirements.

5. Deployment phase—

(i) Conduct pre-release validation (e.g. regression testing)

(ii) Verify deployment processes to ensure proper configuration and scalability

6. Maintainance phase—

(i) Monitor post-release quality (e.g. User feedback, performance)

(ii) Ensure continuous improvement based on defect trends.

QA ensures consistent quality and prevent defects throughout SDLC, not just at the end.

Question-19

Explain the Rapid Application Development (RAD) model in software engineering. Discuss its key phases, principles and advantages. Analyze how the RAD model supports faster delivery of software solutions while maintaining quality and user satisfaction.

Ans to the Que: No: 19

Rapid Application Development (RAD) is a software development model focused on quick development development and iteration with user feedback. It emphasizes flexibility and efficiency over strict planning and long development cycles.

Key phases:

1. Requirement planning –
   stakeholders and developers gather initial requirement in a short period, defining the project scope.
2. User Design –
   Users and developers collaboratively design the system with prototyping, iterating on feedback.

3. Construction –
Developers build the system increamentally, focusing on functional prototypes for fast feedback and refinement

4. Cutover (Transition) –
Final system is deployed and any necessary adjustments or refinements are made.

Key Principles –

1. Iterative prototyping :- Focus on rapid iterations to develop functional prototypes.

2. User Involvement :– continuous collaboration with users ensures the system meets their needs.

3. Time-boxed Development – Deliver working versions of the software quickly in shorter cycles.

Advantages –

1. Faster Development: Prototyping and user feedback speed up the process.

2. Increased Flexibility : Allows adjustments during development based on user feedback.

3. User satisfaction : Continuous involvement ensures the system meets user expectations.

How RAD supports Faster Delivery:

1. Prototyping and Incremental Development reduce time spent on rework.
2. Early user involvement ensures the the software aligns with user needs, reducing the chances of major changes later.
3. Short development cycles and reusable components streamline delivery without sacrificing quality.

Question-20

Write box testing consists of code coverage and a data coverage method. Consider the following decision (e.g. if, switch, while etc) and make one test for each side of each decision using a table with column caption Decision, x input and y input. Then implement JUnit test class that tests each decision described in the table using Java

(Follow the sample in python)

```
int x, y;
x = c.readInt();
y = c.readInt();

if (y==0)
    c.println("y in zero");

else if (x==0)
    c.println("x in zero");

else {
    For (int i=1; i<=x; i++) {
      if (i%y ==0)
        c.println(i);

    }
}
```

// JUnit test class that tests each decision described in the table using python as follows, do the same in Java.

```
import unittest
Class DecisionTest (unittest.TestCase):
    def setUp (self):
      # Mocking the behavior of c.println using a
       list to collect output
      self.output = []
```

```
def println (self, message):
 # Simulates the C.println function
 self.output.append(str(message))

def process(self,x,y):
  # The translated logic of the Java code

if y==0:
   self.println("y is zero")

elif x==0:
   self.println("x is zero")

else:
    for i in range(1,x+1):
       if i%y==0:
          self.println(i)

 def test_y_is_zero(self):
     self.output.clear()
     self.process(5,0)
     self.assertEqual(self.output,["y is zero"])

def test_x_is_zero(self):
     self.output.clear()
     self.process(0,3)
     self.assertEqual(self.output,["x is zero"])
```

```
def test_loop_does_not_run(self):
        self.output.clear()
        self.process(0,2)
        self.assertEqual(self.output,[])

def test_numbers_divisible_by_y(self):
        self.output.clear()
        self.process(4,2)
        self.assertEqual(self.output,["2","4"])

def test_numbers_not_divisible_by_y(self):
        self.output.clear()
        self.process(4,3)
        self.assertEqual(self.output,["3"])

def test_edge_case_y_negative(self):
        self.output.clear()
        self.process(5,-2)
        self.assertEqual(self.output,["4","4"])

deft test_edge_case_x_negative(self):
        self.output.clear()
        self.process(-3,2)
        self.assertEqual(self.output,[])

if __name__ == "__main__":
    unittest.main()
```

Ans to the Que: NO: 20

Here's the Java JUnit test class based on your Python example:

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;


import java.util.ArrayList;
import java.util.List;

class DecisionTest {
    private List<String> output;

@BeforeEach
 void setUp() {
        output = new ArrayList<>();

 }
void println(String message) {
    output.add(message);
 }
```

```
void process(int x, int y) {
  if (y == 0){
      println("y in zero");
  }
  else if (x == 0) {
      println("x in zero");
  }
  else {
      for (int i=1; i<=x; i++){
        if (i % y == 0) {
            println(String.valueOf(i));
        }
      }
  }
}
@Test
void testY Is Zero(){
    process(5,0);
    assertEquals(List.of("y in zero"), output);
}
```

```
@Test
void testXIsZero(){
  process(0,3);
  assertEquals(List.of("x is zero"), output);

}
@Test
void testLoopDoesNotRun() {
    process(0,2);
    assertTrue(output.isEmpty());

  }
@Test
void testNumbersDivisibleByY(){
   process(4,2);
   assertEquals(List.of("2","4"),output);

}
@Test
void testNumbersNotDivisibleByY(){
   process(4,3);
   assertEquals(List.of("3"),output);

  }
```

```
@Test
void testEdgeCaseYNegative() {
    process(5, -2);
    assertEquals(List.of("2", "4"); output);

}
@Test
void testEdgeCaseXNegative() {
    process(-3, 2);
    assertTrue(output.isEmpty());

    }

}
```

This Java code follows the same logic and test structure as the Python example. Each decision is tested for different inputs.

Question-21

Black Box Unit testing is earlier and more precise than Black Box System testing it can find errors very early, even before the entire first version is finished. Now, Consider the production codes that need function testing. Suppose you have JUnit 4 API in your IDE and you are said to develop test codes for these production codes showing the application of exception, Setup function and Timeout Rule. How do you solve it?

Ans to the Que: NO: 21

To develop test codes using JUnit 4 for produc codes involving exceptions, setup functions and timeout rules, you can follow these steps.

1. Exception Handling: Using the @Test(expecte = Exception.class) annotation to specify the expected expection.

For example:

```
@Test(expected = IllegalArgumentException.class)
public void testException(){
    myObject.someMethod();
}
```

2. Setup Function: Use the @Before annotation to set up any preconditions or mock objects needed for your test.

```
@Before
public void setUp(){
    //Setup code here
}
```

3. Timeout Rule: Use the @Rule annotation with Timeout to enforce a time limit on your test method.

```
@Rule
public TimeOut globalTimeout = Timeout.seconds(5);
        // 5 seconds timeout
```

This structure ensures that the tests hande exceptio are prepared with setup logic and respect timeout constraints during execution.