

Ans. to. the. Q. no. 1

Product backlog for the user stories are broken down into task as follows:

User Story 1: As a user I want to log in securely so that I can access my account.

Tasks:

- ① Design the login page (ui/ux design)
- ② Implement the front end of the logic page
- ③ Create and integrate user authentication APIs
- ④ Configure database for storing and validating user
- ⑤ Implement password encryption and security measures
- ⑥ Conduct unit testing for login functionality
- ⑦ Perform integration testing with the database.
- ⑧ Write automated test case for security validation

User Story 2: As a user, I want to search for product by category to find items easily.

Tasks:

- ① Design the search page and category filters

- ② Implement the front end of search page.
- ③ Create backend API for fetching products by category.
- ④ Integrate the API with front end.
- ⑤ Optimized the database queries for efficient product search.
- ⑥ Implement validation for search input.
- ⑦ conduct unit testing for search input functionality.
- ⑧ perform usability testing to ensure ease of use.

→ Prioritization of user stories in sprint planning  
During sprint planning, the development team and product owner collaborate to prioritize user stories based on:

- ① Value of the customer
- ② Technical feasibility

Sprint prioritization order:

i) user stories 1 (Login functionality): High priority due to its criticality.

ii) user story 2 (Search functionality)

Tracking process using a Scrum board.

A Scrum board is divided into three columns: To Do, In progress, and Done.

To Do: All tasks from the product backlog that are selected from the sprint.

In progress: Task activity being worked on by team member

Done: completed tasks that meet the definition of done.

Main point

i) product backlog

ii) Sprint prioritization

iii) Scrum board.

Ans. to the Q. no. 2

How does the methodology address Risk management and Adaptability.

1. Spiral Methodology:

Risk Management:- Spiral focuses heavily on risk analysis. Each phase starts with identifying and mitigating risk. High Risk components are addressed early.

Adaptability:- The iterative nature allows for refinement and changes after each cycle, ensuring the product evolves as risks and requirements become clearer.

Key Advantage: Best for project with high uncertainty, as it emphasizes risk reduction through repeated evaluation.

2. Agile Methodology:

Risk Management: Agile minimizes risks by delivering working software in short iterations. Feedback after each iteration ensures potential

Tasks are identified and resolved early.

**Adaptability:** Agile is highly adaptable, because it embraces change, allowing the client to update requirements frequently.

**Key Advantage:** Suited for projects with evolving requirements as it ensures continuous client collaboration and quick responses to change.

### 3. Extreme programming (XP)

**Risk Management:** XP manages risks through practices like pair programming, continuous integration, and frequent releases. It reduces the likelihood of defects and ensures equality at every step.

**Adaptability:** XP is highly adaptive due to its focus on customer collaboration and delivering small increments of functional software. Changes can be integrated at any stage.

**Key Advantage:** Works best in projects requiring frequent delivery of small, functional components with evolving requirements.

Most suitable project Methodology for this project is Agile methodology because;

- ① It embraces change, allowing the team to adjust to evolving client needs.
- ② Regular feedback and short iterations reduces the risk of delivering a product that doesn't meet expectations.
- ③ continuous collaboration ensures alignment with client priorities, making it cost-effective and manageable.

However, If the tasks are too high and require in-depth analysis, the spiral methodology could be an alternative due to its strong focus on task management.

Ans. to the Q. no. 3

comparison of development models.

### 1. Waterfall model:

Key features:

- Sequential and linear process (Requirement → Design → Development → Testing → Deployment).
- Works best for projects with well-defined and stable requirements.

Strengths:

- Predictable and easy to manage.
- Suitable for project with strict timelines and clear deliverables.

Weaknesses

- Not flexible for evolving requirements.
- High risk if errors are found late in the process.

### 2. Agile model:

Key features:

- Iterative and incremental development with regular customer feedback.
- Divides work into short sprints or iterations.

Strengths:

- Highly adaptable to changing requirements.
- Encourages closer collaboration with the customer.

- Weakness:*
- Requires continuous involvement from the customer
  - Less suited for projects with strict deadlines and fixed requirements.

### 3. Extreme Programming (XP):

#### Key features:

- Focuses on frequent releases, continuous testing, and customer collaboration.
- Emphasizes practices like pair programming, test-driven development, and small functional updates.

#### Strengths:

- Excellent for projects with evolving requirements.
- Reduces technical risk through frequent testing and updates.

#### Weakness:

- Requires high levels of discipline and collaboration.
- Less structured than Waterfall or Spiral for well-defined projects.

### 4. Spiral model:

#### Key features:

- Combines iterative development with risk analysis.

analysis at every phase.

→ Focuses on Identifying and mitigating risks early.

Strength:

→ Handle high risk project effectively

→ Allow refinement of requirements over time.

Weakness:

→ Can be more expensive and more complex to implement.

→ Requires expertise in risk analysis.

Analysis of project A and project B.

1. project A (well-defined requirements, strict deadline):

Best methodology: waterfall model

Reasoning:

→ clear and stable requirements align with waterfall's structured approach

→ The sequential process ensures each phase is completed on time, meeting the strict deadline.

→ Predictability and a fixed timeline are crucial, and waterfall excels at delivering such projects.

→ Agile or XP's flexibility is unnecessary when requirements are well-defined.

→ The iterative process may be waste time that is critical for a strict deadline.

project B (Envolving requirement, uncertain timeline)

Best methodology: Agile or XP

Reasoning:

- Agile's iterative approach is perfect for handling evolving requirements.
- continuous customer feedback ensures the project product evolves to meet changing needs.
- XP is also a strong choice for its focus on frequent release and adaptability.
- Waterfall is rigid and does not handle changing requirements effectively so that it is not suitable.
- Spiral is suitable for high risk projects but may be unnecessary complex for evolving requirements with continuous feedback so it is not use.

Ans. fo. the. Q. no. 4

principle of software engineering ethics:

Software engineering ethics are guidelines that ensure professional practices are responsible and prioritize public welfare, quality, and integrity in their work. Key principles include:

public interest:

Software engineers should prioritize the well-being and safety of the public above personal or organizational interests.

Quality of work

Delivering high-quality, reliable, and maintainable software is an ethical responsibility.

Honesty and integrity:

Engineers must provide accurate information about the capabilities, risks and limitations of software.

Confidentiality:

Protecting sensitive data and respecting the privacy of users is critical.

Competence: preparing yourself to determine  
Engineers should work only within their areas  
of expertise and continuously improve their  
skills.

Fairness:  
Avoid discrimination and treat all clients  
colleagues, and users fairly.

Issues Related to professional Responsibility:

Accountability:

Engineers are responsible for the software  
they design, including its social, legal, and  
technical impacts.  
conflicts of interest: may arise when personal  
ethical dilemmas may arise when personal  
interest conflicts with professional duty.

Security and privacy concerns:

Engineers must ensure that software products  
protects user data and prevents unauthorized access.

~~Ignoring software quality~~

Failure and Risk Management: Ignoring potential risks or delivering defective software could harm users or organizations.

Respecting Intellectual property:

Using third party software or ideas without proper acknowledgment is unethical.

Role of ACM/IEEE code of Ethics

The ACM/IEEE code of ethics provides a framework for ethical decision-making in software engineering.

It emphasizes the following:

1. Act in the public interest.
2. Avoid Harm.
3. Be honest and transparent.
4. Respect privacy and confidentiality.
5. Provide fair treatment.
6. Maintain competence.
7. Support Ethical Standards.

How the code guides Ethical decision-making.

1. Public safety concerns
2. Privacy violations
3. Reporting defects
4. Intellectual property issues

Ans. to. the Q. no. 5

## Functional Requirements:

Functional Requirements describe what the system should do, focusing on the features and capabilities that support its primary purpose.

### Flight Search and Reservation:

Users should be able to search for available flights based on criteria like destination, date and time.

### User Registration and Authentication

The system must allow users to create accounts, log in securely and manage their profiles.

contribution: Improves security by ensuring that only authorized users can access their bookings.

### Payment Processing

The system should support secure payment options (credit/debit cards, digital wallets) for ticket purchases.

contribution: provides a seamless booking process and ensures final financial transaction.

are secure.

### Ticket management:

User should be able to view, modify or cancel their reservations.

contribution: keeps user informed, improving the user experience and reliability of the system.

### Non functional Requirements

Non-functional requirements describe how the system should perform and focus on quality attributes such as performance, reliability and maintainability.

#### 1. System performance

The system must be handle upto 10,000 concurrent users without significant delays.

contribution: Ensures smooth performance during peak usage times, improving user satisfaction.

#### 2. Availability and uptime:

The system must be available 99.9% of the time, with minimal down time for maintenance.

contribution: Ensures reliability and builds user trust in the system.

#### 3. Data Security

All user data, including payment information must be encrypted and comply with industry

Standards.

contribution: Protects sensitive user information, ensuring security and compliance.

#### 4. Scalability

The system should be scalable to accommodate increasing numbers of users and flights.

contribution: prepares the system for future growth ensuring long-term usability and cost effectiveness.

#### 5. Usability

The interface should be intuitive, with clear navigation and minimal learning curve for new users.

contribution: Enhances the user experience, ensuring that even non-technical non-technical users can use the system effectively.

Ans. to. the q.no. 6

The V-model is a software development approach where testing is planned alongside development. It connects every development activity to a matching testing activity, making it easier to find and fix errors early.

The 'V' shape shows the relationship between development and testing phases.

Phases of the V-model

### 1. Requirement phase:

What happens?

Gather what the system should do

Testing activity: Acceptance Testing  
checks if the final product meets user needs and works as expected.

### 2. System design phase:

What happens?

Define how the system will work, including its structure and components.

Testing activity:

Tests the entire system to ensure it meets the original requirements.

### 3. High level design phase (HLD)

What happens?

Break the system into modules and define how they interact.

Testing activity: Integration Testing

Ensures that different parts of the system work together correctly.

### 4. Low level design phase (LLD)

What happens?

Define the details of each module, such as logic and algorithms.

Testing Activity: unit testing

Checks if individual parts (modules) of the system work properly.

### 5. Implementation phase:

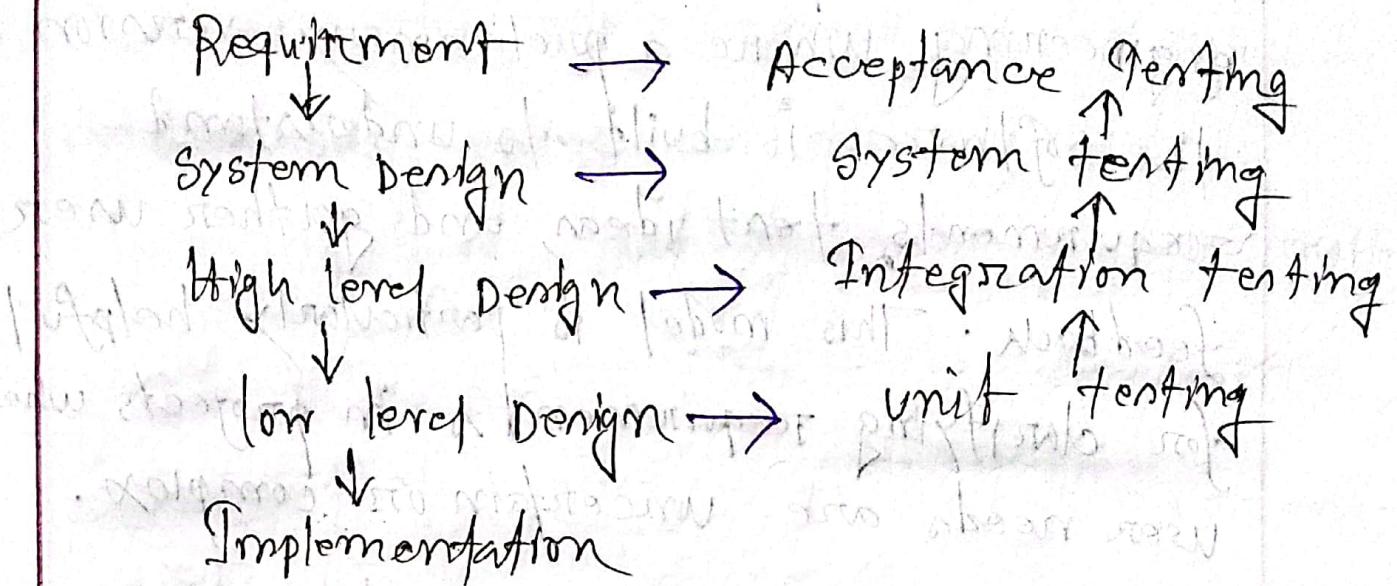
What happens?

Write the actual code of the system based on the design.

Test activity: code reviews / static testing

## V-model of software development

### Diagram of V-model



- The left side of the V is for development.
- The right side of the V is for testing.
- Both sides are connected: each development phase has a matching testing phase.

### Benefits of V-model

- ① Early Testing
- ② clear plan
- ③ Quality Assurance
- ④ well-documented

Ans. to. the. Q.no. 2

Prototype development is a process in software engineering where a preliminary version of the software is built to understand requirements, test ideas, and gather user feedback. This model is particularly helpful for clarifying requirements in projects where user needs are uncertain or complex.

### Phases in prototype Development :

#### 1. Requirement Gathering and Analysis:

- understand the initial high-level requirements of the system.
- focus on features that are unclear or need validation through user interaction.

#### 2. Quick design

- create a simple and preliminary design for the prototype.
- The design includes the core design for the features that need testing or demonstration.

### 3. Prototype building

- Develop the prototype based on the quick design.
- Use basic coding and tools to create a working model that demonstrates functionality, user interface or workflows.

### 4. User Evaluation

- Present the prototype to stakeholders or end-users for feedback.
- Identify gaps, areas for improvement or additional requirements.

### 5. Refinement of Requirements

- Based on user feedback, refine the system requirements.
- Repeat the process iteratively until the requirements are well-understood.

### 6. Development of the final product

Once the requirements are clear and finalized, move on the actual system design and development of the system.

Aim towards the reduction of mistakes and to avoid rework. It needs more time to bring forth solid information to work.

## How prototype Refines software Requirements

**Visual Representation:** The prototype provides a tangible way for stakeholders to see how the system will work, helping them articulate their needs better.

**Feedback Integration:** Feedback from users helps identify gaps in requirements or any missing functionality.

**Requirements validation:** Ensures that the initial requirements align with user expectations and business goals.

**Benefit of using prototype model.**

- ① Improve user feedback
- ② Risk reduction
- ③ Iterative development
- ④ Better communication

**Improve user feedback:**

→ prototype gives users to an early version of the system to interact with.

→ user can identify usability issues or suggest enhancements before the final product is

built

### Risk Reduction

- Identify potential design flaws or misunderstandings early, reducing costly change later.
- Helps avoid building unnecessary features by refining requirements iteratively.

### Iterative Development:

- Prototypes are updated multiple times based on feedback, leading to gradual improvement.
- Allows the development team to focus on features that add the most value.

Md. Lutfor  
ID: IT-21092

Ans. to. the. Q. no. 8

The process improvement cycle in software engineering is a continuous cycle of assessing, improving, and monitoring the processes used to develop software. The goal is to enhance the efficiency, effectiveness, and quality of the software development process over time. It involves the use of systematic methods to identify areas of improvement, make changes, and evaluate the outcomes to ensure better software development practices.

Key stages of the process improvement cycle.

1. Process Assessment: First, the current development process is reviewed to see what's working well and what needs improvement.
2. Goal Setting: After identifying areas for improvement, specific goals are set, for example, reducing bugs or speeding up development time.
3. Implementation of improvements: Changes or improvements are made to the process.

This could include adopting new tools, changing workflows, or improving communication.

4. Measurement and Monitoring: After the improvements, it's important to measure how well they are working.

5. Review and Evaluation: The final step is to review the results. If the goals were achieved, the improvements are kept. If not, more changes are made, and the cycle continues.

commonly used process metrics:

① Defect density

② cycle time

③ Velocity

④ customer satisfaction

⑤ code churn

How metrics help to improving Software processes

① spot problems Early.

② track improvement.

③ make informed decisions.

④ maintain quality.

Ans. to the Q. no. 9

The SEI CMM is a framework that helps organization improve their software development processes over time. It helps answer where an organization is in its software development journey and provides a roadmap for improvement. The model has five levels, each representing a different stage of growth in an organization's processes.

The five level of SEI model CMM

1. Level 1: unpredictable (initial)
2. Level 2: Managed (Repeatable processes)
3. Level 3: Defined (Standardized processes)
4. Level 4: Quantitatively managed (using data to manage processes)
5. Level 5: Optimizing (continuous improvement)

How the level helps:

Level 1: organizes the processes.

Level 2: Improves predictability and control

Level 3: Increase consistency and efficiency.

Level 4: make decisions based on data

and measurement.

Level 5: focus on continuous improvement and staying competitive.

By moving through these levels, an organization gradually improves its software development practices, making processes more efficient and predictable, and ensuring better quality and delivery in the long run.

Mr. Lutfor

ITD-IT-210%2

Ans. to the Q. no. 10

Agile is a method of software development that allows focuses on flexibility, teamwork, and customer feedback. The goal is to deliver small parts of the software quickly, get feedback from users, and improve the software step by step.

core principle of Agile :

1. customer collaboration : Agile values talking to the customer often to make sure the software meets their needs.
2. Responding to changes : Agile encourage adapting to change.
3. people over process : Agile believes that the team work skills and communication are more important than strict processes.
4. Working software over process documentation
5. Simplicity
6. self organized teams.
7. frequent delivery
8. continuous improvement.

## Learnings about Agile

How Agile works in different Environment

1. Startups
2. Large Enterprises
3. Software maintenance
4. customer facing projects.

Benefit of Agile

1. Faster delivery
2. Better customer satisfaction
3. flexibility
4. High quality
5. Better collaboration and Teamwork.

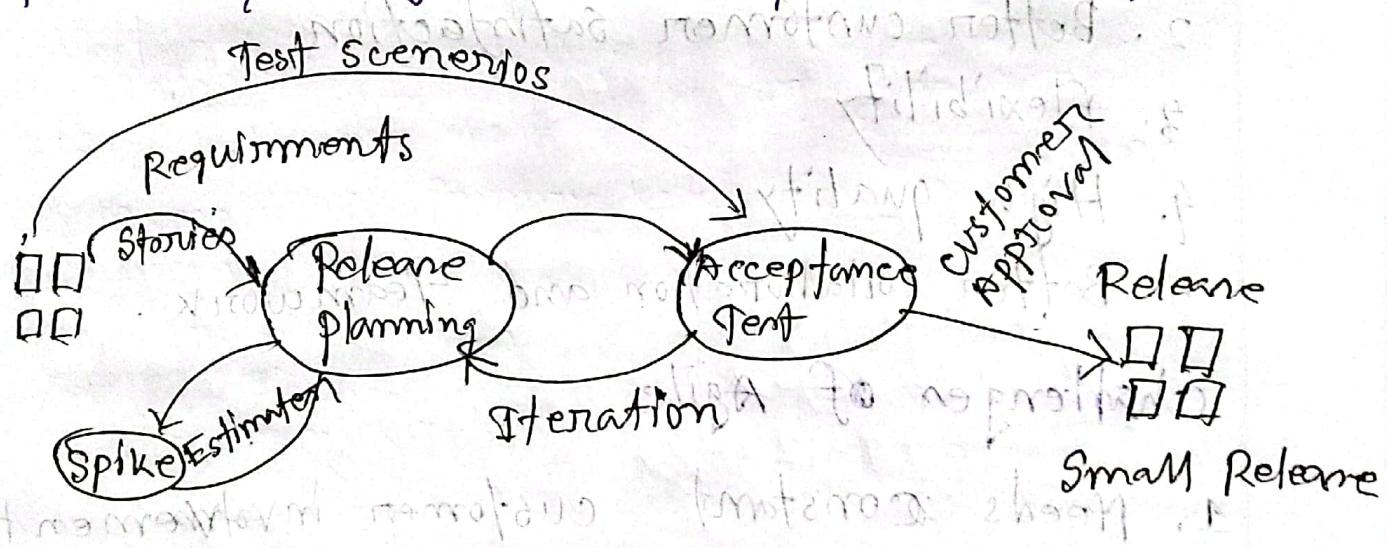
Challenges of Agile

1. Needs constant customer involvement
2. Scaling issues
3. Resistance to change
4. Relies on team Expertise.
5. Scope creep

Ans. to the Q no. 11

Extreme programming (XP) is a software development methodology designed for flexibility, high-quality software, and customer satisfaction. Its release cycle is iterative and emphasizes frequent release of small, functional software increments.

Release cycle of extreme programming



### 1. Exploration phase:

- collaborate with the customer to define user stories
- prioritize the stories based on value and feasibility

### 2. planning phase

- create a release plan for multiple iterations
- Select high priority user stories for first iteration

### 3. iteration cycle:

- ① Design
- ② Coding
- ③ Testing
- ④ Review

Software development

#### 4. Release:

- Delivery the working product to the customer after a few iterations (every 1-3 weeks)
- collect customer feedback for future development

#### 5. Maintenance

- Address bug and issues based on user feedback
- plan additional iteration if required.

### Influential programming practices in XP

Extreme programming includes several key practices that contributes to the success of the software development process.

#### 1. Test-driven development (TDD)

- Write automated tests before writing the code
- Ensures code meets requirements and produces bug-free code

#### 2. pair programming

#### 3. Continuous Integration

#### 4. simple design

#### 5. collective code ownership

#### 6. frequent releases

#### 7. on-site customer

#### 8. Refactoring

Ans. to, Q.no. 12

## Entity Relationship Diagram (ERD) for a Library Management System

### Entity and Attributes

#### 1. Book

##### Attributes:

- (1) Book-ID (primary key)
- (2) Title
- (3) Author
- (4) ISBN
- (5) Genre

#### 2. Member

##### Attributes:

- (1) Member-ID (primary key)
- (2) Name
- (3) Contact-details

#### 3. Borrowing Activity

##### Attributes:

- (1) Borrow-ID (primary key)
- (2) Borrow-Date
- (3) Return-Due-Date
- (4) Return-Status
- (5) Fine

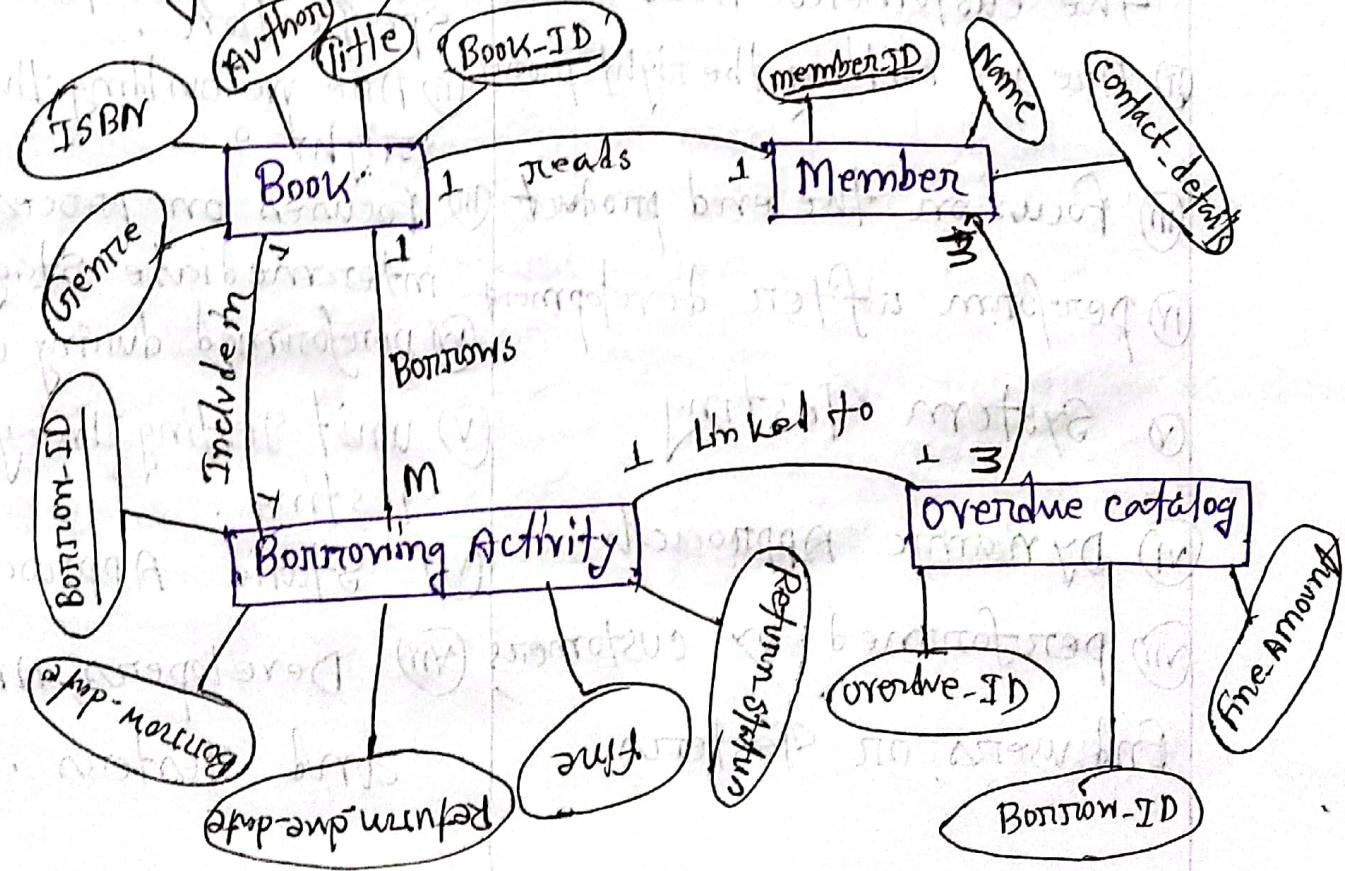
## q. overdue catalog

Attributes:

- (1) overdue-ID (primary key)
- (2) Borrow-ID (foreign key)
- (3) fine-amount

Relationships:

- Member(1) → Borrows → Borrowing Activity (M)  
one member can have multiple borrowing activities
- Book (+) → included in → Borrowing Activity (1)  
one book is tied to one borrowing activity at a time
- Borrowing Activity (1) → Linked to → Overdue Catalog (+)  
one Borrowing Activity can result in one overdue catalog entry if the book is overdue.



Ans. to the Q. no. 13

Testing in software engineering is the process of evaluation of a software application or system to ensure it meets the specified requirements, functions correctly and is free from defects. The goal of testing is identify bugs, ensure the software performs as expected and verify that it meets user needs.

Difference between Validation and Verification

Validation	Verification
i) Ensures the product meets the customer's need	i) Ensures the product is built correctly as per Specification.
ii) Are we building the right product?	ii) Are we building the product right?
iii) focus on the end product	iii) focuses on processes on intermediate stage.
iv) perform after development	iv) performed during development
v) System Testing	v) unit testing, Integration Testing.
vi) Dynamic Approach	vi) static Approach
vii) performed by customers End users, or Testers	vii) Developers, Analysts and Testers.

## Layered Architecture Model:

The Layered Architecture model is a software design pattern that separates the system responsibilities into different layers.

- Organized the system into layers with related functionality associated with each layer.
- A layer provides services to the layer above it.

presentation layer / user interface layer

Business Layer / Domain Layer

persistency layer

Database Layer

presentation layer / user interface layer:

- Responsible for handling all user interface and browser communication logic.

Business Layer / Domain Layer:

- Responsible for executing specific business rules associated with the request.
- We can put the method models and logic that is specific to business problem that we are trying to solve.

Persistence layer:

- It contains the code to access the database layer.

## PL/SQL - Programmatic interface

→ It is the set of code to maintain/manipulate the database: like SQL statements, connection details.

### Database Layer:

→ This is the layer where all the data is stored  
→ It is the underlying database technology like SQL Server.

Example: online judge system.

Presentation layer: A web page or mobile App that provides the interface for users. This layer manages forms, text boxes, buttons, etc.

Application layer: Validating user-submitted code, running the code, calling services to generate results.

Business layer: Managing the execution process of a piece of code, like running C++ code or executing python code correctly.

Database layer: Registration data, grading information, code archives, etc.

Ans. to the Q.no. 15

Data flow Diagram Hospital management system is used to create an overview of hospital management without going into much details.

→ context level (0-level) DFD of Hospital management system : The 0-level DFD for hospital management system depicts the overview of whole hospital management system. It is supposed to be an abstract view of overall system.

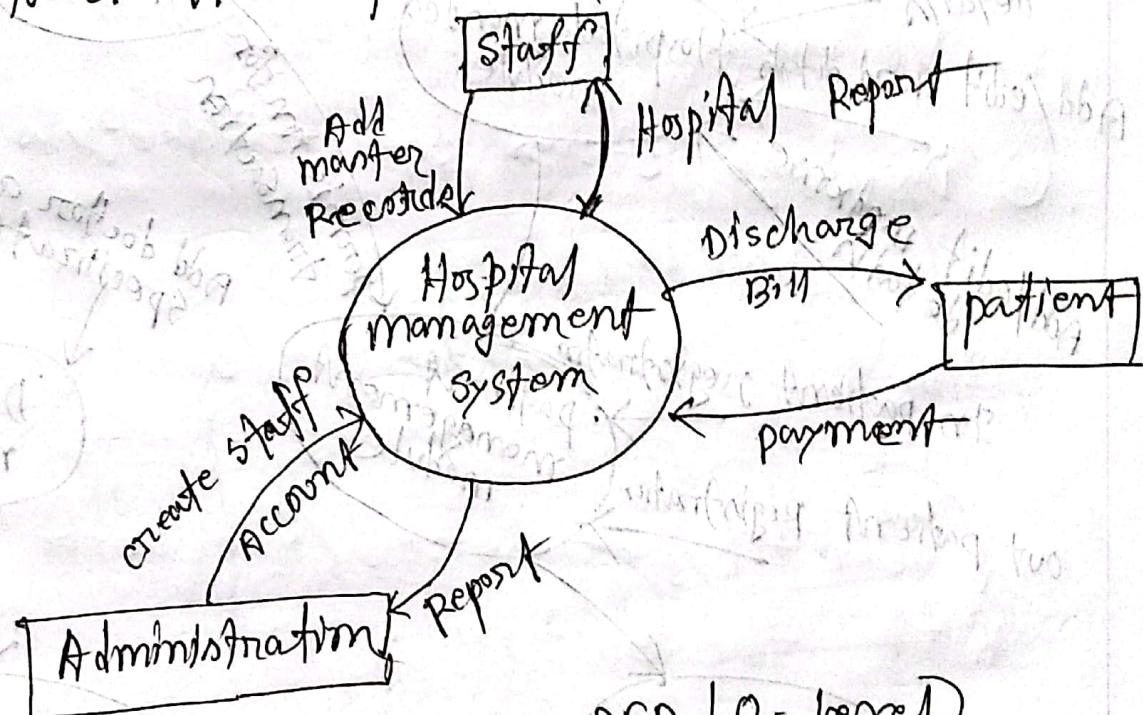
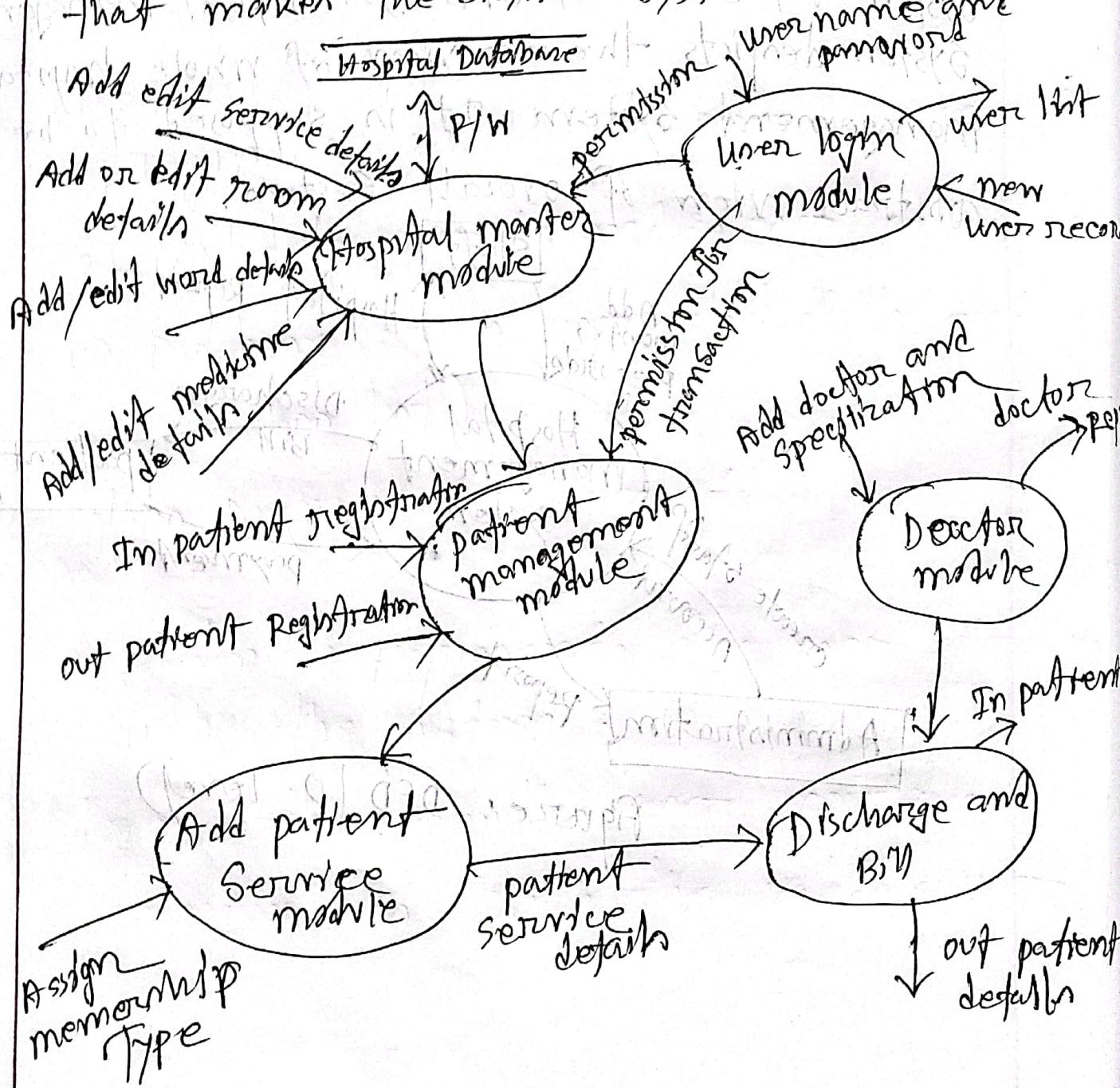


Figure: DFD (0-level)

## Front level Data Flow Diagram (Level-1 DFD)

OR Hospital management system.

The first level DFD (1st level) of Hospital management shows more details of processing  
Level - I DFD list all the major sub processes  
that makes the entire system.



## 3. Hospital Management

The important process to be covered are:

(1) User and login

(2) Hospital Master

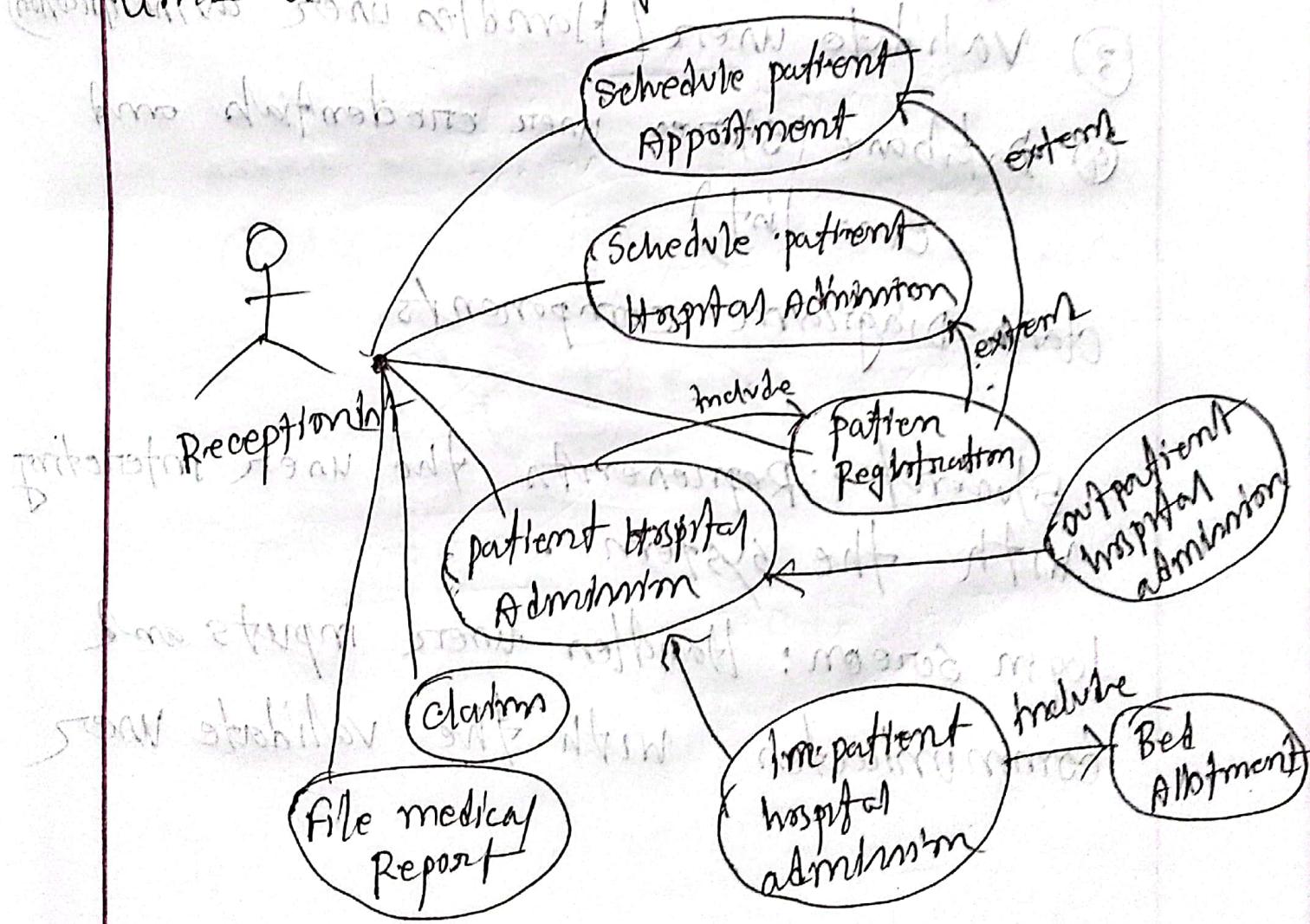
(3) patient registration

(4) Doctor module

(5) Add patient service

(6) Discharge billing

UML use case diagram:



## Ans. to the Q.no. 16

Based on the UML sequence diagram, we can derive a UML class diagram by identifying the key entities and their relationship. The main entities involved in this scenario are:

- ① Student (Actor)
- ② Login Screen (Handles user login interface)
- ③ Validate user (Handles user authentication)
- ④ Database (Stores user credentials and class info)

class diagram components

Student: Represents the user interacting with the system.

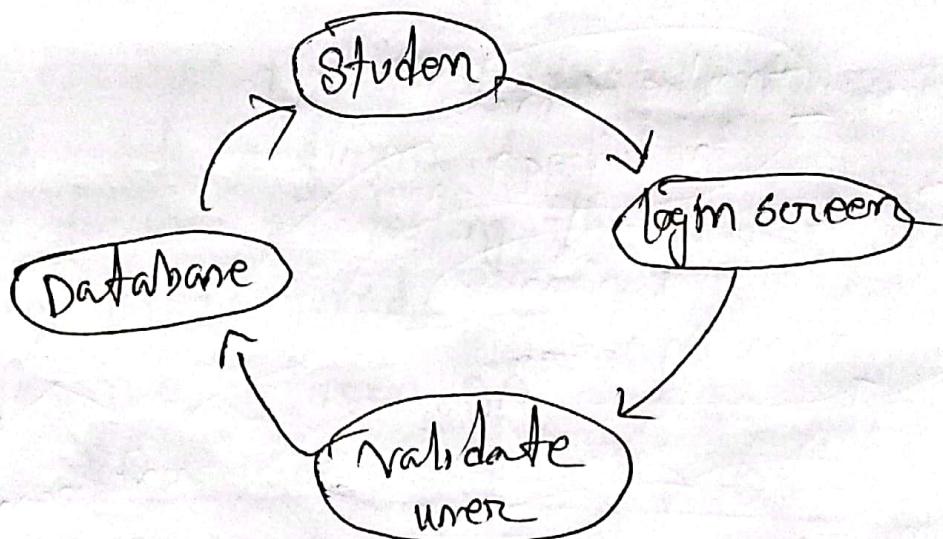
Login Screen: Handles user inputs and communicates with the validate user.

class -

Validate user: process login credentials and interacts with database.

Database: Stores user credentials and class info information.

class diagram relationship:



Ans. to the Q.no. 12

### Quality assurance (QA)

- QA is about improving processes to prevent problems.
- It ensures that the way software is developed is efficient and product produces high-quality results.
- QA focuses on planning, setting rules and following best practices to avoid mistakes.

### Quality control (QC)

- QC is about checking the product to find any mistakes or problems.
- It ensures the final product works as expected and meets requirements.
- QC focuses on testing the product and fixing any issues found.

### Difference between QA and QC

#### QA

- ① Focus on processes
- ② Nature proactive
- ③ Prevent problem before they happen
- ④ Responsible for entire organization

#### QC

- ① Focus on product
- ② Nature Reactive
- ③ Find and fix problem in the product.
- ④ Responsible for testing and development.

## QA & QC

### Impediments to QA and QC

challenges in QA (a) technical factors at AP <

- (i) Inconsistent process Adherence
- (ii) Lack of skilled professionals
- (iii) Time and Resource constraints
- (iv) Resistance to change

challenges in QC

- (i) Inadequate test coverage
- (ii) complex defect detection
- (iii) Poor Requirement clarity
- (iv) Tools and Resource Limitations.

→ QA and QC are complementary  
→ QA makes sure that processes are right  
to prevent problems.

→ QC checks the product to find and fix problems  
→ Both are important and work together  
to ensure high-quality software.

Ans. to the Q: no. 18

No, the goal of Quality Assurance (QA) is much more than finding bugs. It ensures that the processes used to build software are correct, efficient, and lead to high-quality products. Finding and fixing bugs early is part of QA, but the main focus is to prevent defects and ensure the software meets user needs.

#### Role of QA in Each SDLC phase

1. Requirement Analysis phase:
  - QA ensures that requirements are clear and testable
  - QA identifies any unclear or missing details in the requirements
  - This prevents problems later due to misunderstood requirements.

#### 2. Design phase:

- QA reviews the system design to ensure it meets requirements.
  - It checks for any potential problems in the design before development starts.
  - This helps avoid major changes during development.

## QA in Software Development

### 3. Development phase:

- QA ensures that developers follow coding standards.
- QA helps with unit testing and identifies errors in the code early.
- This reduces the chances of bugs during testing.

### 4. Testing phase:

- QA runs different types of tests like functional, integration and system tests.
- Bugs found during testing are reported and fixed.
- This ensures the software works as intended and meets user requirements.

### 5. Deployment phase:

- QA checks the software is ready for release by running final tests.
- QA ensures the development process is smooth and error-free.
- This helps deliver reliable software to users.

### 6. Maintenance phase:

- QA monitors the software for issues after release.
- QA ensures any update or changes are properly tested before going live.

Ans. to the Q. no. 19

The Rapid Application and development (RAD) model is a software development methodology that emphasizes speed and flexibility. It focuses on quickly developing prototypes and delivering functional components in short cycles. RAD aims to meet changing user needs and delivery quality software faster.

### Key phases of RAD model.

1. Requirements planning
2. User design
3. construction
4. cutover.

### principle of RAD model

- ① User involvement
- ② Iterative development
- ③ prototyping
- ④ component Range Reuse.

### Advantage of RAD model

- ① Faster delivery
- ② flexibility
- ③ High user satisfaction
- ④ Reduced Risk
- ⑤ Better quality.

## How RAD Supports Faster Delivery and maintains quality.

- ① prototyping
- ② Iterative process
- ③ user feedback
- ④ reusable components
- ⑤ parallel development

Ans. to the Q. no. 20

Let's break the problem into two parts

- ① Analyzing a simple code with decisions and designing test cases.
- ② Implementing a JUnit test class to test these decisions.

Simple code with decisions (if, else, and while)

public class DecisionExample {

    public static String process(int x, int y)

        if (y > 0) {

            return "y is zero";

    else {

        StringBuilder result = new StringBuilder();

        for (int i = 1; i <= x; i++) {

            if (i % y == 0) {

                result.append(i).append(" ");

        }

    } // for loop

    return result.toString().trim();

}

}

}

## Table for test cases

Decision	x Input	y Input	Expected output
$y = 0$	5	0	"y is zero"
$x = 20$	0	3	"x is zero"
for loop does not run	0	2	"empty string"
$y \neq 0$	9	2	"2 4"
$y \neq 0$	9	3	"3"

## Junit test cases

```
import org.junit.Test;
```

```
import static org.junit.Assert.*;
```

```
public class DecisionExampleTest {
```

// Test case for  $y=0$

@Test

```
public void testIsZero() {
```

```
    assertEquals("y is zero", DecisionExample.proc(5, 0));
```

}

// Test code for  $x=20$

@Test

```
public void testXIsZero() {
```

assertEqual ("x is zero", DecisionExample.process(0, 3)),  
}

// Test case where the loop does not run.

@Test

```
public void testLoopDoesNotRun () {
```

```
    assertEquals ("", DecisionExample.process(0, 2));
```

```
}
```

// Test case where number divisible by y are printed

@Test

```
public void testNumberDivisibleByY () {
```

```
    assertEquals ("2 4", DecisionExample.process(4, 2));
```

```
}
```

Test case where number not divisible by y

@Test

```
public void testNumbersNotDivisibleByX () {
```

```
    assertEquals ("3", DecisionExample.process(9, 3));
```

```
}
```

Ans. to the Q.no. 21

production code Example

public class utility {

// method for division

public static double divide (int numerator, int denominator){

if (denominator == 0) {

throw new IllegalArgumentException ("Denominator can  
not be zero");

return (double) numerator / denominator;

}

// method for reverse a string

public static String reverseString (String input) {

if (input == null) {

throw new NullPointerException ("Input string cannot be  
null");

return new StringBuilder (input).reverse().toString();

}

} // class definition

{ } // class definition

## Junit test class

Here's how we can write test case using Junit applying exceptions, a setup function and timeout rule.

```
import java.org.junit.Before;
```

```
import org.junit.Rule;
```

```
import org.junit.Test;
```

```
import org.junit.Rules.ExpectedException;
```

```
import org.junit.rules.Timeout;
```

```
import org.junit.Assert;
```

```
public class UtilityTest {
```

```
    @Rule public RuleForTimeout timeout = new RuleForTimeout();
```

```
    public Timeout globalTimeout = new Timeout(1000);
```

```
    @Rule public RuleForException exception = new RuleForException();
```

```
    @Rule
```

```
    public ExpectedException thrown = new ExpectedException();
```

```
    @Setup public void setup() {
```

```
        // Test for divide method
```

```
        @Test
```

```
        public void testDivide() {
```

```
            assertEquals(2.5, utility.divide(5, 2), 0.01);
```

```
}
```

ii) Test for divide exception

@Test

```
public void testDivideException() {
    thrown.expect(IllegalArgumentException.class);
    thrown.expectMessage("Denominator cannot be zero");
    utility.divide(10, 0);
}
```

ii) Test for ReverseString() method.

@Test

```
public void testReverseString() {
    assertEquals("cba", utility.reverseString("abc"));
}
```

ii) Test for reverse string exception

@Test (expected = NullPointerException.class)

```
public void testReverseStringException() {
    utility.reverseString(null); }
```

ii) Time out test example

@Test

```
public void testLongRunningProgram() {
    int result = 0;
    for (int i = 0; i < 100000; i++) {
        result += i; }
    assertEquals(0, result); }
```