Answer to the question no (1)

(i) Ans:

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

(i) Design login UI.
(ii) Frontend Implementation
(iii) Backend Implementation
(iv) Backend Authentication API.
(v) Frontend-Backend Integration
(vi) Security
(vii) Testing
(viii) Deployment

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

(i) Design search UI
(ii) Frontend Implementation
(iii) Backend Product.search API.
(iv) Frontend-Backend Integration
(v) Optimization
(vi) Testing
(vii) Deployment.

(i) Sprint planning and Prioritization:

The team will prioritize the login user story first due to its foundational importance. Without secure login functionality, users cannot interact with the app, making it the highest priority. The search by category user story will follow as it improves the user experience but is dependent on the login feature being functional first.

Scrum Board Tracking:

Tasks will be tracked on the scrum board with columns: To Do, In Progress, and Done. As work progresses, tasks move across the board from To Do (Planned tasks) to In Progress (actively worked on), and finally to Done (completed and reviewed). This method ensures visibility and allows the team to track progress efficiently throughout the sprint.

Answer to the question no (2)

For a project with significant risks and evolving requirements, Agile and Extreme Programming (XP) are the most suitable methodologies due to their strong focus on adaptability and continous risk management.

* Agile uses short, iterative cycles (sprints) that allow teams to adjust quickly based on feedback, helping to manage risk by identifying issues early. Frequent collaboration with the client ensures that the product aligns with their changing needs.
* XP takes this further with extreme adaptability, incorporating testing, pair programming and frequent releases. It's highly suited for projects where requirements evolve rapidly and where maintaining high quality is essential.

While Spiral also emphasizes risk management through iterative prototyping, its process is heavier and less flexible, making it less ideal for projects with fast-changing needs. In contrast, Agile and XP offer quicker adjustments, making them the better fit for evolving requirements and managing risk in a cost-effective way.

Answer to the question no (3)

For Project A, which has well-defined requirements and a strict deadline, the Waterfall model is the most suitable. Waterfall's linear and sequential approach ensures clear planning, defined stages, and a predictable timeline, making it ideal for projects with fixed scope and deadlines. Since the requirments are set upfront, there is little need for change during the development process, minimizing the risk of scope creep. This methodology also provides clear documentation, making it easier to track progress against deadlines.

For Project B, with evolving requirements and an uncertain timeline, Agile or the Spiral model would be better choices. Agile offers flexibility through iterative cycles (sprints), allowing the team to continously adapt to changing client needs and feedback. Its focus on customer collaboration ensures the product evolves as desired. Spiral also works well, ~~partially~~ particularly for high risk projects, by focusing on risk management and continous

P.T.O

feedback. Its iterative nature allows for adjustments based on changing requirements while addressing potential risks early.

In summary:

* Project A: Waterfall for its predictability and structured approach.
* Project B: Agile for adaptability and frequent customer feedback, or spiral for its focus on risk management and iterative refinement.

Answer to the question no (c)

The Principles of software engineering ethics, highlighting the issues related to professional responsibility:

Software engineering ethics focuses on integrity, accountability, and professionalism in development practices. Key principles include:

(i) Public Interest: Prioritize the safety, health and welfare of the public.

(ii) Client and Employer Responsibility: Act in the best interest of clients and employers, ~~maintaining~~ maintaining confidentiality and intellectual property.

(iii) Professionalism: Maintain high standards of competence and stay updated with industry best practices.

(iv) Accountability: Take responsibility for the work produced, ensuring its quality and impact.

ACM/IEEE code of Ethics guides ethical decision-making by emphasizing the importance of public safety, honesty, and integrity. It urges engineers to act with professionalism,

P.T.O

report issues truthfully, and ensure the quality of their work. The code also highlights the responsibility to safeguard privacy and maintain competence. These guidelines help engineers make decisions that are ethical, ensuring trust and accountability within the software development field.

—o—

Answer to the question no (5)

Functional Requirements for the Airport Reservation System:

(i) Flight search: The system must allow users to search for flights based on criteria such as departure date, destination, and class.

(ii) Booking and Ticketing: Users must be able to select flights, enter personal information, and make reservations.

(iii) Payment processing: The system must handle secure payment transactions for ticket purchases.

(iv) Reservation management: Users should be able to view, modify or cancel their reservations.

P.T.O

5. Flight status updates: The system should provide real-time status updates (e.g. delays, cancellations).

Non-Functional Requirements for the Airport Reservation System:

(i) System performance: The system should provide flight search results within 3 seconds.

(ii) Scalability: The system must be able to handle high traffic during peak travel seasons (e.g. .. holidays)

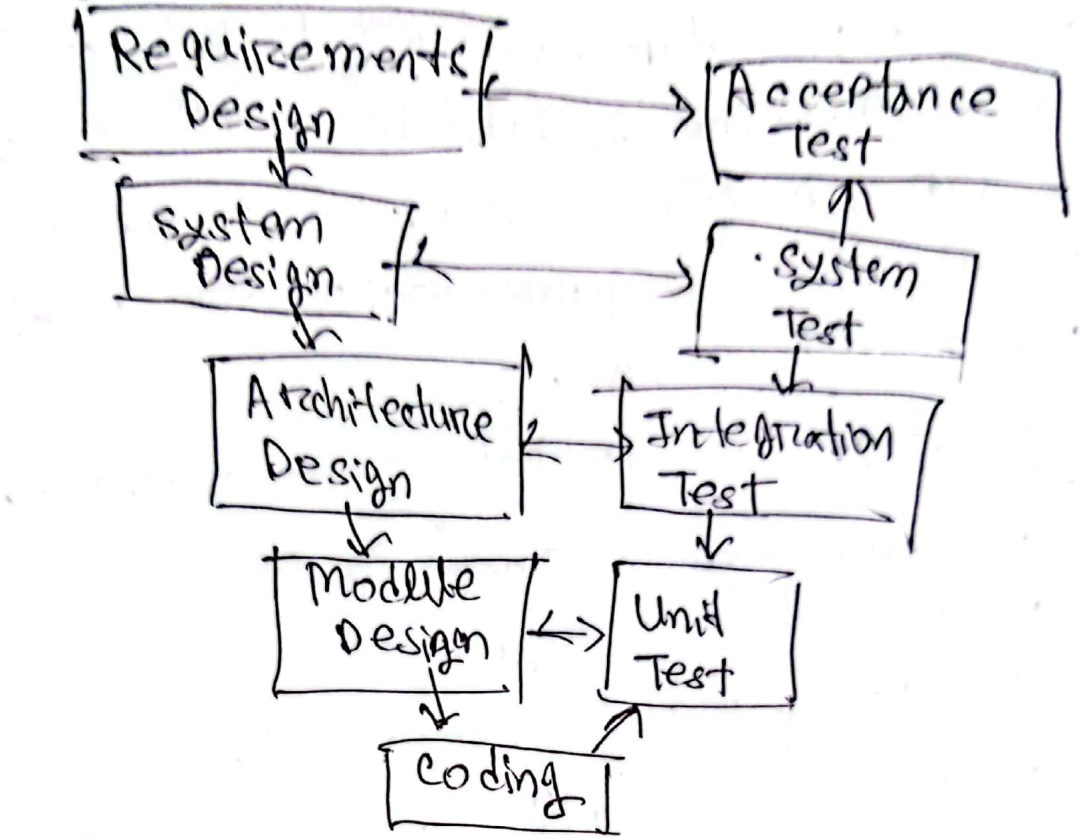
(iii) Availability: The system should be available 24/7 with at least 99.9% uptime.

(iv) Data security: All sensitive user data (e.g., payment details, personal information) must be encrypted and stored securely.

(v) Usability: The user interface must be intuitive, with clear navigation and minimal steps required to complete tasks.

Answer to the question no (6)

V-model testing Phases:



Explanation of Relationships:

* Top-down on the left arm: Progression of development activities from abstract to concrete (from requirement code).
* Bottom-up on the right arm: Corresponding testing phases validate each development phase in reverse order, ensuring the system meets requirements step-by-step.

By explicitly linking testing to development activities, the V-model ensures robust validation and verification throughout the software lifecycle.

Answer to the question no (7)

Prototype development is a software engineering approach where a working model of the software, called a prototype, is built to help understand user requirements, validate functionality, and refine system specifications. It is particularly useful when requirements are unclear or subject to change.

Key stages in Prototype Development:

(i) Requirement Identification
(ii) Prototype Design
(iii) Prototype Development.
(iv) User Evaluation and Feedback
(v) Refinement and Iteration
(vi) Finalization

How Prototyping Refines Software Requirements:

* Visual Representation.
* Identifying
* Iterative Refinement

Benifits of Using the Prototyping Model:

(i) User Feedback
(ii) Risk Reduction
(iii) Iterative Development

Answer to the question no (8)

The process improvement cycle in software engineering aims to enhance the quality, efficiency, and productivity of software development by refining processes over time. It typically involves a cyclical approach that includes planning, implementing, evaluating and optimizing.

Key stages of Process Improvement cycle:

(i) Assessment
(ii) Planning
(iii) Implementation
(iv) Monitoring and Evaluation
(v) Optimization

Commonly Used Process metrics:

(i) Effort metrics (e.g., person-hours)
(ii) Defect Density (Defects per KLOC)
(iii) Cycle Time and Lead Time
(iv) Velocity (Agile metric)
(v) Customer Satisfaction (CCSAT)
(vi) Escaped Defects

P.T.O.

Importance of Metrics in Process Monitoring and Improvement:

(i) **Benchmarking**: Helps compare current processes against industry or organizational standards.

(ii) **Data-Driven Decision**: Provides evidence for justifying changes in processes

(iii) **Efficiency Gains**: Identifies areas for automation and resource optimization.

(iv) **Quality Enhancement**: Ensures better software products by reducing defects and improving reliability.

———)———

Answer to the question no (3)

The Capability Maturity Model (CMM), developed by the Software Engineering Institute (SEI), is a framework for improving software development process within organizations. It provides structured levels that organizations can progess through to achieve greater process maturity, better quality, and improved project outcomes.

Five levels of Capability and Maturity:

(i) Initial (Level 1)
* Characteristics
* Contribution

(ii) Repeatable (Level 2)

(iii) Defined (Level 3)

(iv) Managed (Level 4)

(v) Optimizing (Level 5)

Benefits of SEI CMM for Software Development and Organizational Performance:

(i) Enhanced process quality: Ensures disciplined development practices at higher levels.

—P.T.O—

(iii) Predictable Outcomes: Achieved through data-based insights and standardized work-flows.

(iii) Customer Satisfaction: Improved through higher quality and on-time delivery.

(iv) Continous Improvement: Fosters a culture of innovation and learning at Level 5.

By moving through these levels, organizations achieve greater process maturity, leading to efficient and high-quality software development practices.

Answer to the question no (10)

The core principles of Agile software development methods:

(i) Customer collaboration over contract Negotiation

(ii) Responding to change over Following a plan

(iii) Working software over comprehensive Documentation

(iv) Individuals and Interactions over Processe and Tools

(v) Sustainable Development

(vi) Continous Delivery and Feedback.

(vii) Technical Excellence

Application in Different Software Development Environments:

(i) Startups

(ii) Large Enterprises

(iii) Regulated Industries (e.g. healthcare, finance)

(iv) Remote / Distributed Teams

Benefits of Agile methods:

(i) Increased Flexibility: Adaptability to changing requirements.

(ii) Higher Customer Satisfaction: Continous delivery and feedback loops.

(iii) Improved Collaboration: Empowered teams and better communication

(iv) Reduced Risks: Early detection of issues through iterative testing.

challenges of Agile methods:

(i) Scope creep: Frequent changes can lead to uncontrolled Project scope.

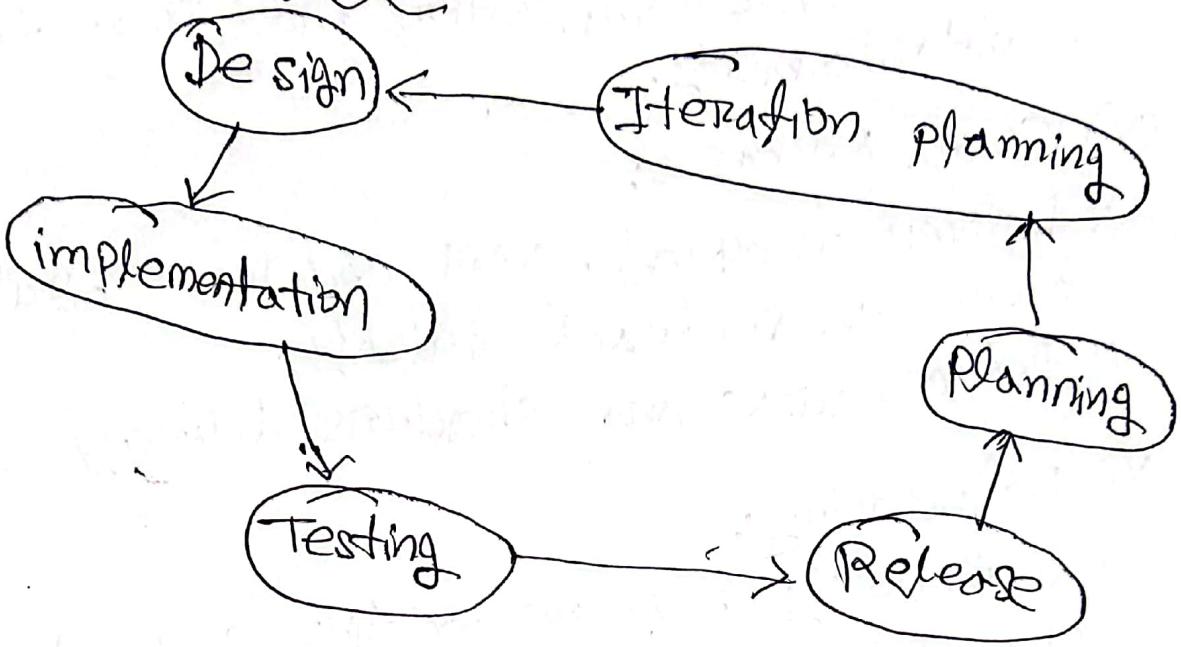
(ii) Requires strong Team Dynamics: Success depends on skilled, motivated team members

(iii) Limited Documentation.

(iv) Scaling Issues: Difficult to implement in large, multi-team environments.

Answer to the question no (11)

The release cycle of (Extreme Programming (XP)) Diagram:



Influential Programming Practices in XP:

(i) Pair Programming:
* Two ~~development~~ developers work together on the same code, with one writing code and the other reviewing.
* Improves code quality and fosters knowledge sharing.

(ii) Test-Driven Development (TDD):
* Write automated tests before writing code.
* Ensures code meets requirements and reduces bugs.

(iii) Continuous Integration (CI):
* Frequently integrate and test code changes.
* Detects issues early and ensures compatibility.

— P.T.O —

(iv) Simple Design :

* Keep designs as simple as possible to meet current needs.
* Avoids over complication and promotes maintainability.

(v) Small Releases:

* Deliver frequent, small updates to gather customer feedback quickly.
* Reduces risks and shortens delivery cycles.

(vi) Refactoring:

* Continously improve existing code without changing functionality.
* Enhances code readability and maintainability.

→

10

Answer to the question no (12)

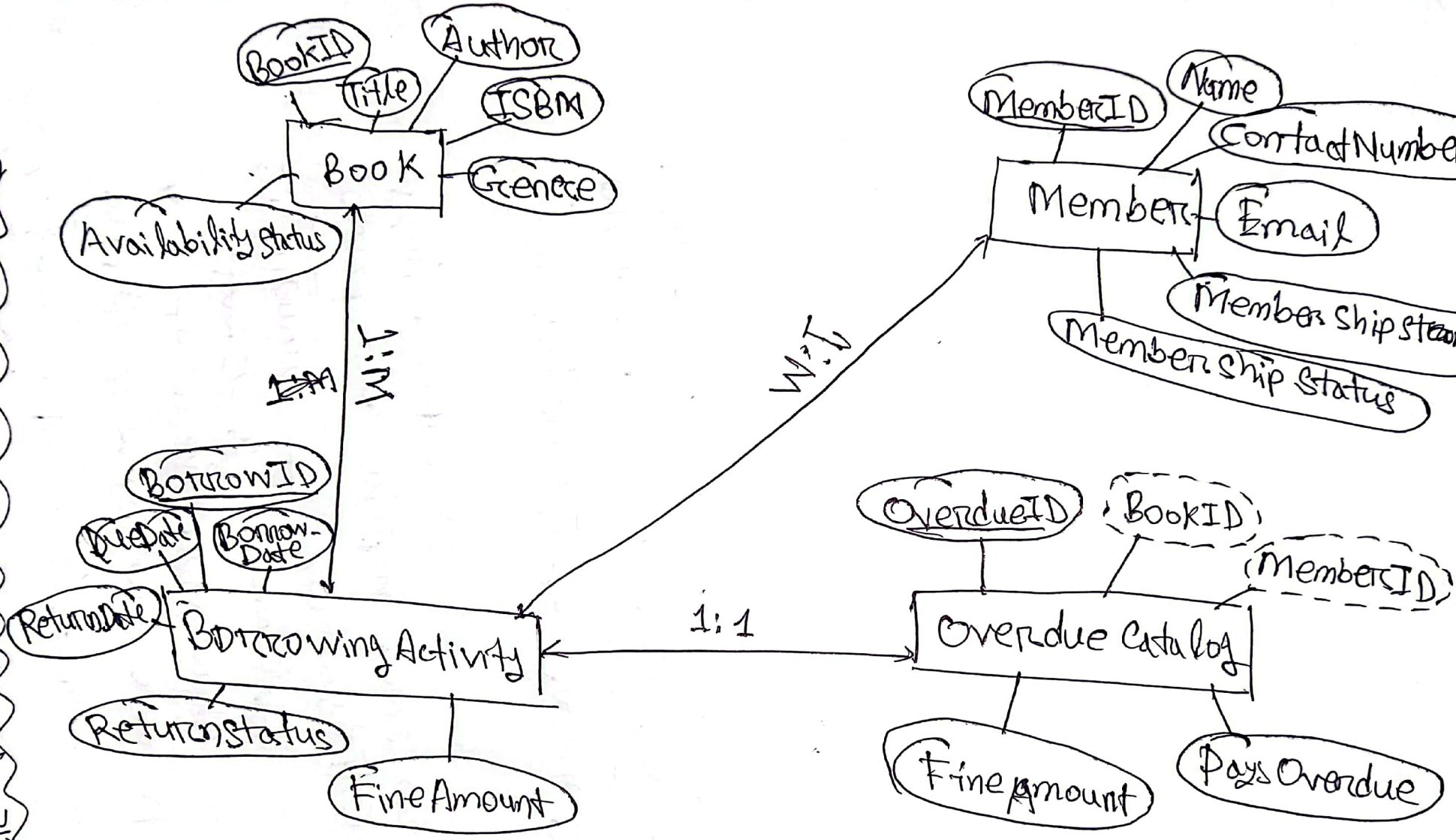


Fig: Entity-Relationship Diagram (ERD) for a Digital Library Management System.

Relationships:

1. Book to Borrowing Activity: One be borrowed multiple times but only once at a time, creating a 1 to Many relationship
   ~~* Book In n Borrowing Acti~~
2. Member to Borrowing Activity: One member can borrow multiple books, creating a 1 to Many relationship

3. Borrowing Activity to Overdue Catalog: An overdue borrowing entry links to the catalog with fine, forming 1 to 1 relationship.

Answer to the question no (13)

Testing refers to the process of evaluating a system, component, or application to ensure it functions as intended. It involves identifying and fixing any defects or issues to guarantee that the Product meets the desired requirements and quality standards.

Difference between Validation and Verification:

	Verification	Validation
1. Purpose	It ensures that the Product is built according to the specified requirements and design. It answers the question "Are we building the Product right?"	It ensures that the Product meets the users needs and requirements. It answers the question, "Are we building the right Product?"
2. Focus	Focuses on the internal process of development, including code correctness, design implementation, and system performance	Focuses on the end Product and whether it intended use and user expectations.
~~3. Meth~~		

— P.T.O —

	Verification	Validation
3. Methods	Involves activities like reviews, inspections, walkthroughs, and testing to confirm that the product was built correctly.	Involves activities like user acceptance testing (UAT), functional testing, and performance testing to confirm that the product fulfills its intended purpose
4. Timing	Occurs during the development phase, before the product is released.	Occurs after the product is built, usually during testing and deployment phases, often with real-world scenarios.

Answer to the question no (14)

Layered Architecture Model for an online Judge System:

An online judge system can be structured with 4 key layers.

1. Presentation Layer:

   * Responsibilities: Handles user interaction such as problem submissions, result viewing and account management. It validates inputs and display output.
   * Scalability: Supports multiple platforms (e.g. web, mobile) with minimal changes to business logic.

2. Application Layer:

   * Responsibilities: Orchestrates operations between the UI and business logic. It manages user sessions and handles requests such as code submission and problem evaluation.
   * Scalability: Distributes tasks effectively, supporting high user loads through queues and load balancing.

3. Business Logic Layer:

   * Responsibilities: Executes the core functionality, like evaluating code submission, running test cases, and managing problem solving rules.

* Scalability: Can scale horizontally by distributing tasks (e.g., multiple servers for code execution).

4. Data Layer:

* Responsibilities: Manages storage for user profiles, Problem data, test cases, and results.
* Scalability: Supports high-performance data retrieval and allows for easy integration of caching mechanisms for frequent data access.

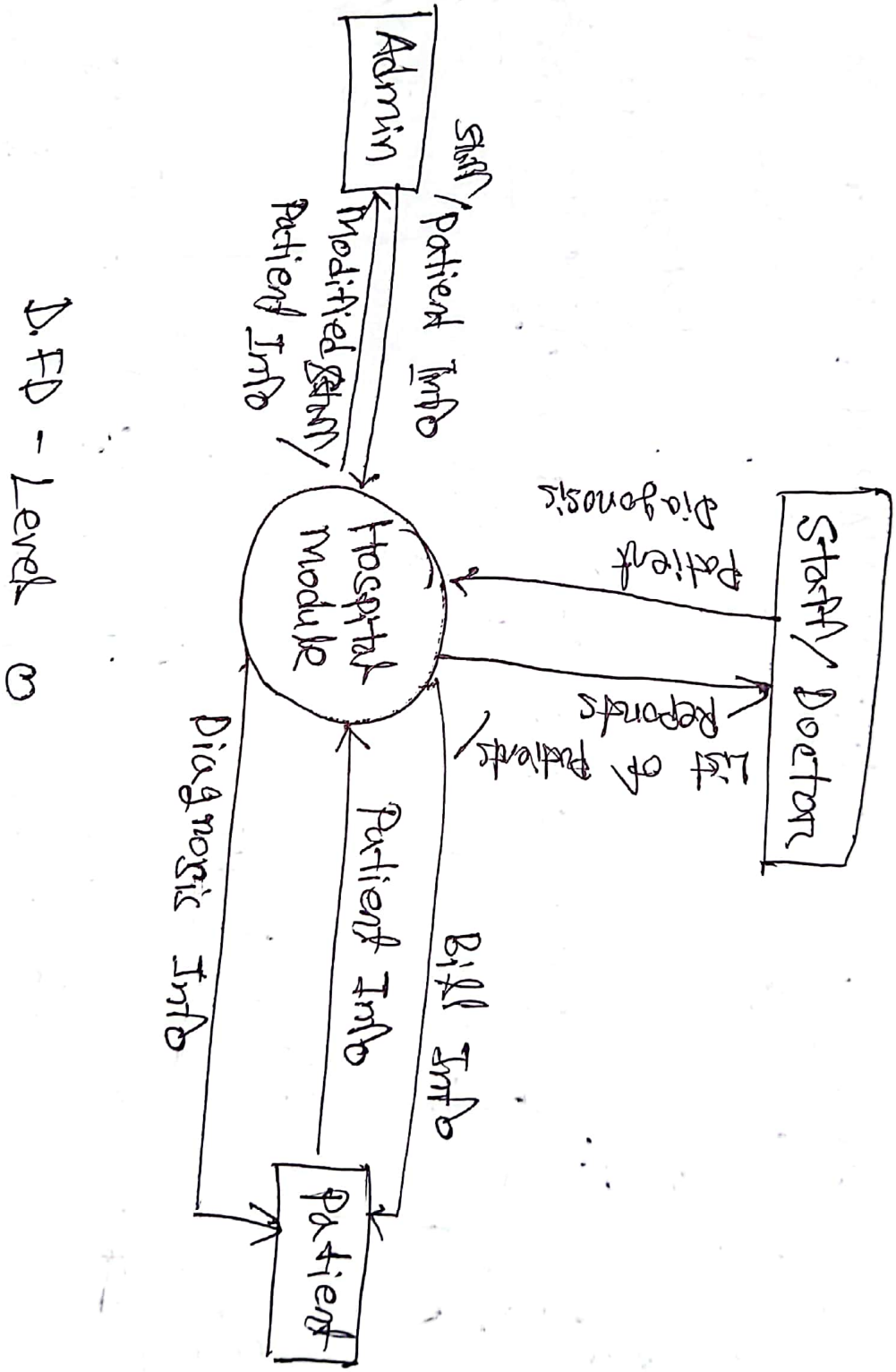
Benifits:

* Scalability: Layers can be scaled independently (e.g., adding more logic servers).
* Maintainability: Easy to update one layer without affecting others.
* Performance: Optimized processing within layers allowing efficient handling of multiple users and evaluations.

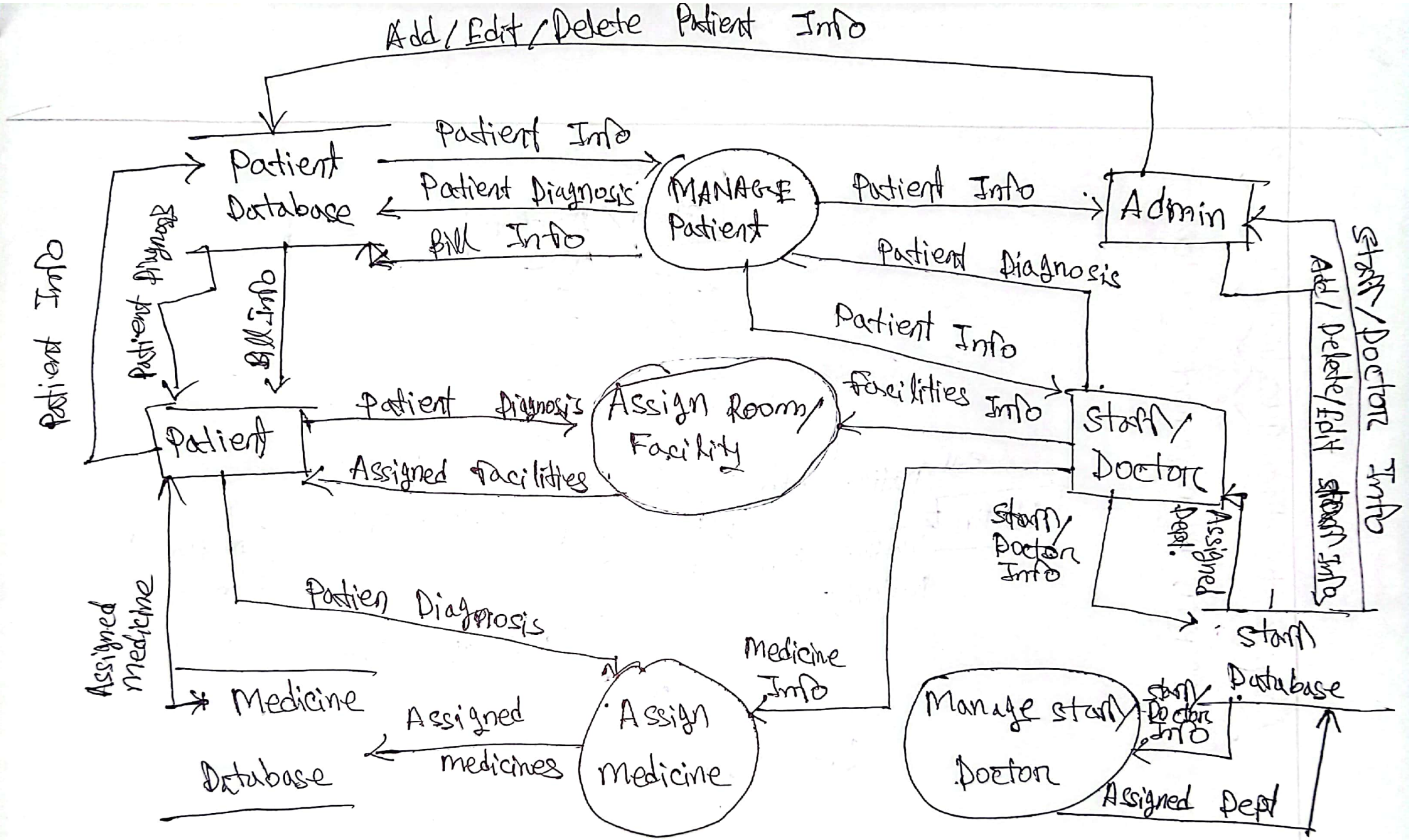
——>

Answer to the question no (13)

Hospital Management system



D.FD - Level 0


Add / Edit / Delete Patient Info
Patient Info
Patient Database
Patient Diagnosis
Bill Info
MANAGE Patient
Patient Info
Admin
Patient Diagnosis
Patient Info
Add / Delete / Edit Staff / Doctor Info
Staff / Doctor Info
Patient Info
Patient Diagnosis
Bill Info
Patient
Patient Diagnosis
Assigned Facilities
Assign Room / Facility
Facilities Info
Staff / Doctor
Staff / Doctor Info
Dept. Assigned
Staff Database
Assigned Medicine
Patien Diagnosis
Medicine Info
Medicine Database
Assigned medicines
Assign Medicine
Manage staff / Doctor
Staff / Doctor Info
Assigned Dept

DFD Level-1

Answer to the question no (14)

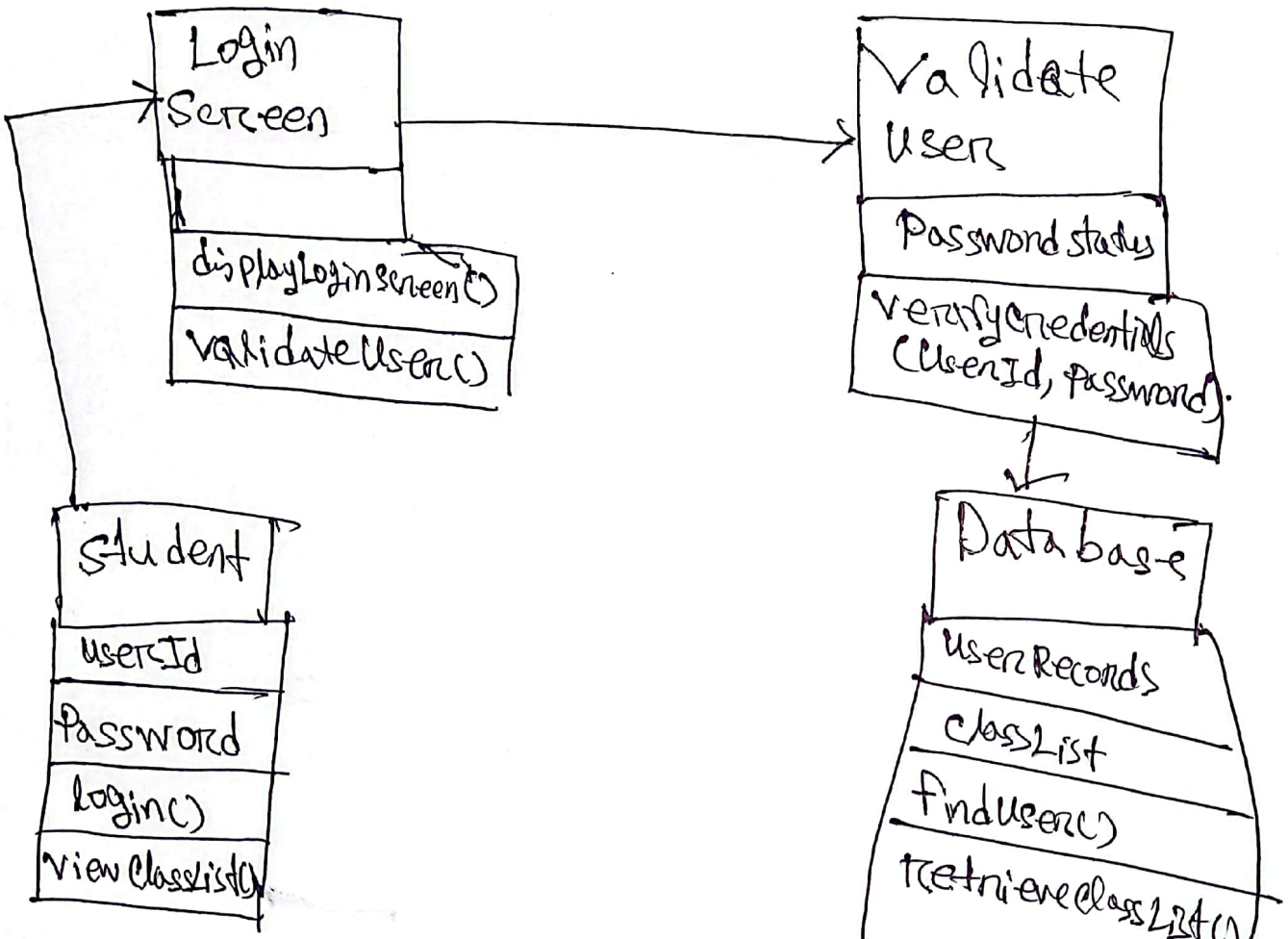


Fig: UML Class Diagram

Answer to the question no (17)

Differences between QA and QC :

Aspect	Quality Assurance (QA)	Quality Control (QC)
Focus	Process oriented (ensures are followed)	Product oriented (ensures the final product is defect free
Goal	Prevent defects through process improvement	Identify and fix defects in the final product
Scope	Broad, includes process design, implementation, and evaluation	Narrow, focuses on inspecting finished products
Activities	Process audits, training, documentation, reviews	Testing, inspections, defect logging, sample reviews
Approach	Proactive (Prevention of defects)	Reactive (detecting and fixing defects)
Timing	During the product development lifecycle	After the product is developed, before delivery
Tools used	Process management tools, training programs	Testing tools, inspection checklists

P.T.O

Impediments to QA and QC:

Impediments to Quality Assurance (QA):

(i) Lack of Leadership Commitment
(ii) Resistance to change
(iii) Insufficient Resources
(iv) Ineffective communication

Impediments of Quality Control (QC):

(i) Inadequate Testing Coverage
(ii) Time Constraints
(iii) Inconsistent Testing Environment
(iv) Subjective Evaluations.

Answer to the question no (18)

Quality Assurance (QA) plays a critical role throughout all phases of the software Development Life Cycle (SDLC) to ensure high-quality software delivery. Heres a breakdown of QA responsibilities during each phase:

1. Requirement Analysis phase:
   * Ensure that the requirements are clear, complete and testable.
   * Identify ambiguities and contradictions in requirements.
   * Participate in requirement reviews and offer feedback to minimize issues down stream.
2. Design Phase:
   * Review system architecture and design documents for completeness, consistency and compliance with best practices.
   * Identify potential design issues early on.
   * Create high-level test strategies and define testing requirements.
3. Development Phase:
   * Collaborate with developers to implement unit tests and continous integration (CI) pipelines.
   * Perform static code analysis and participate in code reviews.

P.T.O

* Begin preparing detailed test cases and scripts based on the design and requirements.

4. Testing phase:
* Execute test cases including functional, regression, performance, security, and usability testing.
* Identify, log, and prioritize bugs for resolution.
* Retest after bug fixes and verify adherence to quality standards.

5. Deployment and Maintenance phase:
* Conduct pre-release testing to ensure production readiness.
* Perform post-development testing to confirm successful implementation.
* Handle defect ~~tri~~ triaging and regression testing during maintenance.
* Ensure user feedback is incorporated for continous improvement.

QA's proactive at every phase ensures not only early bug detection but also comprehensive software quality, leading to a better user experience and reduced development costs.

Answer to the question no x.(19)

The Rapid Application Development (RAD) model is a software development methodology that focuses on delivering high-quality software solutions quickly through iterative development and prototyping. It prioritizes user feedback, collaboration and adaptability.

Key Phases of the RAD model:

1. Requirement planning.
2. User Design
3. Construction
4. Cutover

Principles of RAD:

- Active user involvement throughout the development process.
- Iterative and incremental development cycles.
- Component-based construction using reusable code.
- Continous testing and feedback loops.

Advantages of the RAD Model:

- Faster delivery of functional software compared to traditional methods.
- High flexible to accommodate changes in user requirements.

— P.T.O —

* Early detection and resolution of issues through frequent testing.
* improved user satisfaction due to active engagement and feedback loops.
* Efficient reuse of existing components reduces development time.

How RAD supports Faster Delivery and Quality:

* Prototyping and Iteration
* User Involvement
* Parallel Development
* Testing Throughout.

The RAD model strikes a balance between speed and quality, making it ideal for dynamic environments where requirements are subject to change and rapid delivery is essential for user satisfaction.

—o—

Answer to the question no (20)

Step 1 : Decision Table

The table below outlines the decision and test cases:

Decision	X input	y input	Expected output
if (y == 0)	5	0	"y is zero"
else if (x == 0)	0	3	"x is zero"
for loop doesn't run	0	2	No output
i % y == 0 (divisible)	4	2	"2", "4"
i % y == 0 (not divisible)	4	3	"3"
Edge case: y < 0	5	-2	"2", "4"
Edge case: x < 0	-3	2	No output

Step 2: Junit Test Class in Java

Below it the Java implementation of the Junit test class:

```
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import java.util.List;
import static org.junit.jupiter.API.Assertions.assertEqu

Class DecisionTest{
   Private List<String> output;
   Private void println(String message){
     output.add(message); }
```

— P.T.O —

```
private void Process(int x, int y){
    if(y==0){
      println("y is zero");}
      else if(x==0){
      println("x is zero");}
      else{
          for(int i=1; i<=x; i++){
            if(i%y==0){
          println(Integer.toString(i));}}}}
@Test
void TestYIsZero(){
   output = new ArrayList<>();
   Process(5,0);
   assertEquals(List.of("y is zero"), output);}
@Test
void testXIsZero(){
   output = new ArrayList<>();
   Process(0,3);
   assertEquals(List.of("x is zero"), output);}

@Test
void testLoopDoesNotRun(){
   output = new ArrayList<>();
   Process(0,2);
   assertEquals(List.of(), output);}

@Test
void testNumbersDivisible.
void testNumbersDivisibleByY(){
output = new ArrayList<>();
Process(4,2);
assertEquals(List.of("2","4"), output);}
```

P.T.O

```
@Test
Void testNumbersNotDivisibleBy() {
    output = new ArrayList<>();
    Process(4, 3);
    assertEquals(List.of("3"), output);
}
@Test
Void testEdgeCaseYNegative() {
    output = new ArrayList<>();
    Process(5, -2);
    assertEquals(List.of("2", "4"), output);
}
@Test
    Void testEdgeCaseXNegative() {
        output = new ArrayList<>();
        Process(-3, 2);
        assertEquals(List.of(), output);
    }
```

Answer to the question no (21)

To develop JUnit 4 test cases for production codes, incorporating exception handling, setup functions, and timeout rules, follow these steps with a sample code example:

Production code Example:

```
Public class Calculator {
  Public int add(int a, int b) {
  return a+b; }
Public int divide(int a, int b) {
if(b==0){
 throw new IllegalArgumentException("Division by zero
                                    is not allowed"); }
  return a/b; }
Public long longComputation(int a){
  try {
     Tread.sleep(2000); }
  Catch (InterruptedException e){
  Tread.currentThread().interrupt(); }
  return long a*a ; }
}
```

JUnit 4 Test Code Example:

```
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;
import org.junit.rules.Timeout;
import static org.junit.Assert.assertEquals;
Public class CalculatorTest {
  Private calculator calculator;
  @Before
    Public void setup(){
    Calculator = new Calculator();}
  @Rule
  Public ExpectedException exceptionRule =
                    ExpectedException.none();
@Rule
Public Timeout timeoutRule = Timeout.seconds(1);
@Test
Public void testAdd(){
assertEquals(5, calculator.add(2,3));}
@Test
Public void testDivisionByZeroException(){
exceptionRule.expect(IllegalArgumentException.class);
exceptionRule.expectMessage("Division by zero is not allowed");
calculator.divide(5,0);}
```

P.T.O

```
@Test
public void testLongComputation() {
calculator.longComputation(100); }
}
```

Key features and concepts:

1. Setup Function (@Before)
   The setUp() method initialize the calculator object before each test, ensuring a clean test environment.
2. Exception Handling (@Rule ExpectException)
   The exception Rule allows for precise exception testing, checking both the exception type and the message.
3. Timeout Rule (@Rule Timeout)
   The timeout Rule ensures that tests complete within a specified time limit (1 second in this case). If the longComputation() method exceeds this, the test fails.

This approach ensures robust and maintainable unit tests for production code while demonstrating key JUnit 4 features.