

Assignment  
Software Engineering  
ICT-3209

Name : Sagar Roy  
ID : IT-21044

(1)

### (i) Product Backlog for user stories

User story 1: secure login

- Design login UI (To Do)
- Implement authentication logic (In progress)
- Integrate database for user credentials (To Do)
- Implement password Hashing and Security (To Do)
- Test login functionality
- Deploy and Review

User story 2: product Search by category

- Design search UI (To Do)
- Implement category based filtering logic (To Do)
- Connect with product database (To Do)
- Optimize Search performance (To Do)
- Test search functionality (To Do)
- Deploy and Review (To Do)

### (ii) Sprint planning: Polarization

I) Value to customer: Secure login is critical for user access. Hence it gets higher priority.

II) Technical Feasibility: Login implementation is straightforward. Search requires database queries and filtering logic.

### (II) Scrum Board Tracking:

- To Do: Tasks planned but not started
- In progress: Tasks currently being developed

- Done: Completed tasks

(2)

### Comparison of Spiral, Agile and Extreme Programming for Risk Management

#### (i) Spiral Model:

- Risk Management: User iterative cycles where risks are identified and mitigated early

- Adaptability: Allows changes after each spiral, but major modification can be costly

- Best for: Project with high technical risks needing frequent risk assessment

#### (ii) Agile Methodology:

- Risk Management: Continuous feedback reduces uncertainty; working software is delivered in short iterations

- Adaptability: Highly flexible. Evolving requirements can be incorporated through regular client collaboration.

Best For: Projects with evolving client needs and a focus on quick delivery.

### (III) Extreme programming:

- Risk Management: Uses test-driven development and pair programming to minimize defects and risks.

Adaptability: Changes are easily accommodated due to continuous integration and refactoring.

- Best For: High risk projects requiring rapid changes with strict coding discipline.

In this case Agile is the best choice

Because it balances risks management and adaptability and frequent testing and integration reduce risk impact.

Final answer

Agile is the best choice because it balances risks management and adaptability and frequent testing and integration reduce risk impact.

(3)

Based on characteristics of Both project A and B there are the methodology for those

(I) Project A: Well-Defined Requirements, Strict Deadlines

Best Approach is : Waterfall

Because waterfall follows a structured approach, ensuring each phase is completed before the next begins.

Advantages:

- Predictable timeline and cost
- Clear documentation and planning
- Minimal scope changes prevent delays

(II) Project B: Evolving Requirements, Uncertain timeline

Best Approach is Agile Methodology

Advantages:

Continuous customer feedback

Frequent releases allow flexibility

and quick adjustments.

Reduces risk by prioritizing working software over rigid planning.

So waterfall is best for project A due to its fixed scope and deadline and Agile is best for Project B for adaptability and customer feedback.

#### (4) Principle of Software Engineering Ethics

Software Engineering Ethics focus on responsible and Ethical behavior in professional practice.

Key Principles include:

(I) Public interest : Ensure software benefits society.

(II) Client and Employer Responsibility

(III) Product Quality: Deliver High quality, Reliable, secured software

(IV) Fairness and Honesty: Avoid deception and plagiarism

(V) Respect for colleagues

where professional responsibility issues are

(I) Privacy violation: Mishandling user data can lead to breaches

(II) Software failures

(III) Intellectual property violation

(IV) Bias and Discrimination

#### Role of ACM/IEEE Code of Ethics

(I) Act in the public interest

(II) Avoid Harm

(III) Be Honest and transparent

(IV) Respect privacy and Confidentiality

(V) Promote Fairness

(VI) Maintain professional competence

By following these guidelines, software engineers can make ethical decisions that protect users, clients, and society.

(5)

## Functional and Non-Functional Requirements for an Airport Reservation System

### Functional Requirements:

(I) User Registration and Authentication:

Allows user to create accounts and log in securely.

(II) Flight Search and Booking: Enables users to search for flight based on data, destination and availability.

(III) Payment Processing: Supports multiple payment methods.

(IV) Booking Confirmation and Notification: Sends confirmation email or message.

(V) Cancellation and Refund Management: Allows user to cancel or modify reservation.

## Non-functional Requirements:

(i) Performance: System should handle 1000+ concurrent users with a response time.

(ii) Security: Implement encryption for payment and user data protection.

(iii) Usability: UI should be intuitive, modify friendly and accessible.

(iv) Scalability: System should support future expansion (new airlines).

(v) Maintainability: Modular Architecture for easier updates and debugging.

By addressing both functional and non-functional requirements the Airport Reservation System ensures high performance, strong security and an excellent user experience.

(8)

Process improvement cycle in software Engineering:

Process improvement cycle is a systematic approach to enhancing Software development efficiency and quality. It follows these key stages:

(I) Process Assessment - Evaluate current processes identify inefficiencies and collect data

(II) Process Planning - Define improvement goals and establish strategies for change

(III) Process implementation - Apply the planned changes to development workflows

(IV) Process Monitoring and Evaluation - Measure performance using process metrics to assess effectiveness

(V) Process Refinement - Adjust and refine processes based on evaluation results, ensuring continuous improvement.

Commonly used Process Metrics:

(I) Defect Density - Measures the number of defects per module or lines of code

(II) Cycle time - time taken to complete a development cycle.

(iii) Productivity - Measures the number of features or lines of code developed per developer

(iv) Customer satisfaction - Feedback scores from users

(v) Rework percentage - tracks the amount of work redone due to defects or requirement changes.

By using Process improvement cycle and process metrics, software teams can enhance efficiency, quality and customer satisfaction.

④ Retention rate

• Job satisfaction

• Employee retention

• Work-life balance

• Stress levels

• Overall job satisfaction

## (7) Prototype Development Process in Software Engineering

Prototyping is an iterative approach where a preliminary version of the software is built to refine requirements before full development.

### Key stages of prototype Development

#### (I) Requirements Gathering And analysis:

- identifies core system functionalities based on user needs

#### (II) Quick Design:

A simple user interface will be created.

#### (III) Prototype Development:

- Build a basic working model with limited functionality

#### (IV) User Evaluation and feedback

#### (V) Refinement and Iteration

#### (VI) Final product Development.

### Benefits of the prototyping Model:

#### ■ User Feedback and Requirement Refinement

- User can visualize and interact with the system early.
- Helps in understanding real user expectations.

## ④ Risk Reduction:

- Identifies potential issues early in development

## ⑤ Iterative Development & Flexibility:

- Changes can be incorporated at an early stage

Ideal for evolving requirements and uncertainty

Specialization.

The prototyping modeling enhances user satisfaction, reduces project risks.

(before) old approach - based (2)

before after form approach - based approach

based - based (3)

before after old approach

• based approach

before after approach based

(before) old approach - based (4)

before after old approach

old approach not based

(before) old approach - based (5)

before after old approach

## (9) Software Engineering Institute Capability Maturity Model (SEI CMM)

The Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI), is a framework for assessing and improving an organization's software development maturity.

Five levels of SEI CMM & their contributions

### (1) Level 1 - Initial:

Characteristics: No structured process. Success depends on individual efforts

### (2) Level-2 - Repeatable (Managed):

Basic project management is established.

### (3) Level-3 - Defined:

Organization wide standard processes are documented and followed. More consistency and efficiency across projects.

### (4) Level-4 Managed (Quantitatively Controlled)

Performance Metrics and data analysis are used for process optimization.

### (5) Level-5 - Optimizing (Continuous Improvement)

Continuous process improvement using innovation and feedback loops.

## (i) Core principles of Agile software development:

Agile software development is based on the

Agile Manifesto, which emphasizes:

(i) Individuals and Interactions over processes

and tools - Prioritizing teamwork and collaboration

(ii) Working software over comprehensive documentation

- Delivering functional software early and frequently

(iii) Customer Collaboration over contract Negotiation;

- Engaging users throughout development

(iv) Responding to change over following a plan

Application of Agile principles in different Environment

(i) Startup & small teams: Agile enables fast iteration and quick market adaption.

like SAFe

(ii) Large Enterprises: Framework help co-ordinate multiple teams.

like SAFe

(iii) Regulated Industries: Agile must balance flexibility with compliance and Security.

ambitionI

## Benefits of Agile Methods:

- (I) Faster Delivery
- (II) Enhanced flexibility
- (III) Higher customer satisfaction
- (IV) Better Risk Management

Agile development enhances speed, flexibility and user engagement, making it ideal for dynamic projects.

(I)

### Release Cycle of Extreme Programming (XP)

Extreme programming follows an iterative and incremental developed approach, emphasizing

continuous releases and customer collaboration

The XP release cycle consists of the following phases:

Exploration : (Gather user stories & define requirements)

Planning : (Create small releases, define iterations)

Iterations : (Develop, Integrate and test features in cycles)

bottom ↓  
Release | (Deploy working software for feedback)

Maintainance

Maintainance

| (Bug fixes, enhancements and updates)

Each iteration typically lasts 1-2 weeks with continuous integration and testing to ensure quality.

Influential programming practices in XP:

(I) Pair programming - two developers work together on the same code, improving quality and reducing errors.

(II) Test-Driven Development - writing tests before coding ensures robust and bug free software.

(III) Continuous integration - Code is integrated and tested frequently to detect defects early.

(IV) Simple Design - Aoids unnecessary complexity and makes the code easy to modify.

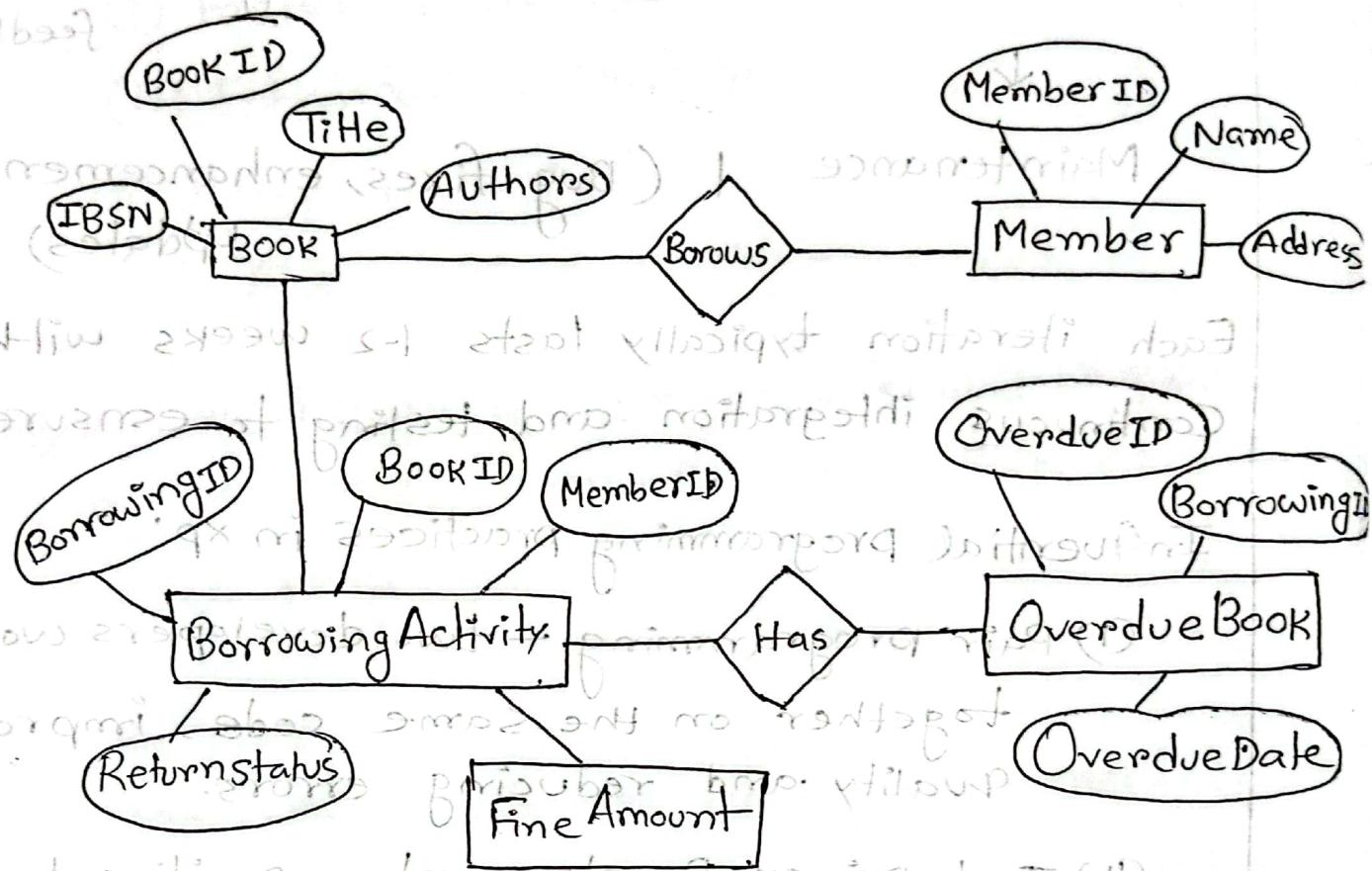
(V) Refactoring

(VI) Collective code ownership

(VII) Sustainable pace

(VIII) Customer collaboration.

⑫ Entity Relationship Diagram for Library Manag



The ERD captures the relationships and key attributes needed to manage the digital library's operations effectively tracks books, members, borrowing activities, Overdue status and fines which is essential for efficient Library Management

(13) ~~What are the 2 basic methods of testing?~~ (A)

Testing: Software testing means the process of evaluating a software application to ensure it meets the specified requirements and is free of defects.

### (Validation)

- (I) Ensures the software is built correctly according to specifications.
- (II) Checks compliance with design, code and documentation.
- (III) Reviews or walkthroughs and inspections which means static.
- (IV) Performed before development.
- (V) Code Reviews, requirement analysis, design and inspection.
- (VI) Checking if the software is designed as per requirements.

### Verification

- (I) Ensures the built software meets user needs and expectations.
- (II) Ensures functionality, usability and performance.
- (III) Dynamic - Actual execution of the software.
- (IV) Performed after development.
- (V) functional testing, user acceptance testing.
- (VI) Checking if the software actually solves user problems.

#### (14) Layered Architecture model for an Online Judge system

A Layered Architecture divides the online judge system into independent functional layers, ensuring scalability, maintainability and efficient performance.

Key layers and their Responsibilities:-

##### 1. Presentation Layer (User Interface)

- Provides a web-based interface
- Handles user authentication, input validation and displaying results.

##### 2. Application Layer (API Gateway & Controllers)

- Manages user requests and directs them to appropriate services.
- Handles authentication, session management

##### 3. Business logic layer (Execution and Evaluation)

- Executes user submitted code in an isolated, sandboxed environment. Compares the output with expected results and assign a pass/fail score.

##### 4. Data Layer (Database and storage)

- Stores user submission, problem sets, test cases and results. User caching for fast retrieval of frequently accessed data.

This Architecture Ensures Efficiency by

(I) Scalability —

the Application and Business logic layers can scaled using load balancing & cloud computing

(II) Maintainability —

Separation of concerns ensures easy updates without affecting other components

(III) Efficient performance

Parallel code execution speed up processing

Caching (Redis) & asynchronous job queues

Optimize performance

This layered architecture ensures a

robust, scalable and maintainable online

judge system.

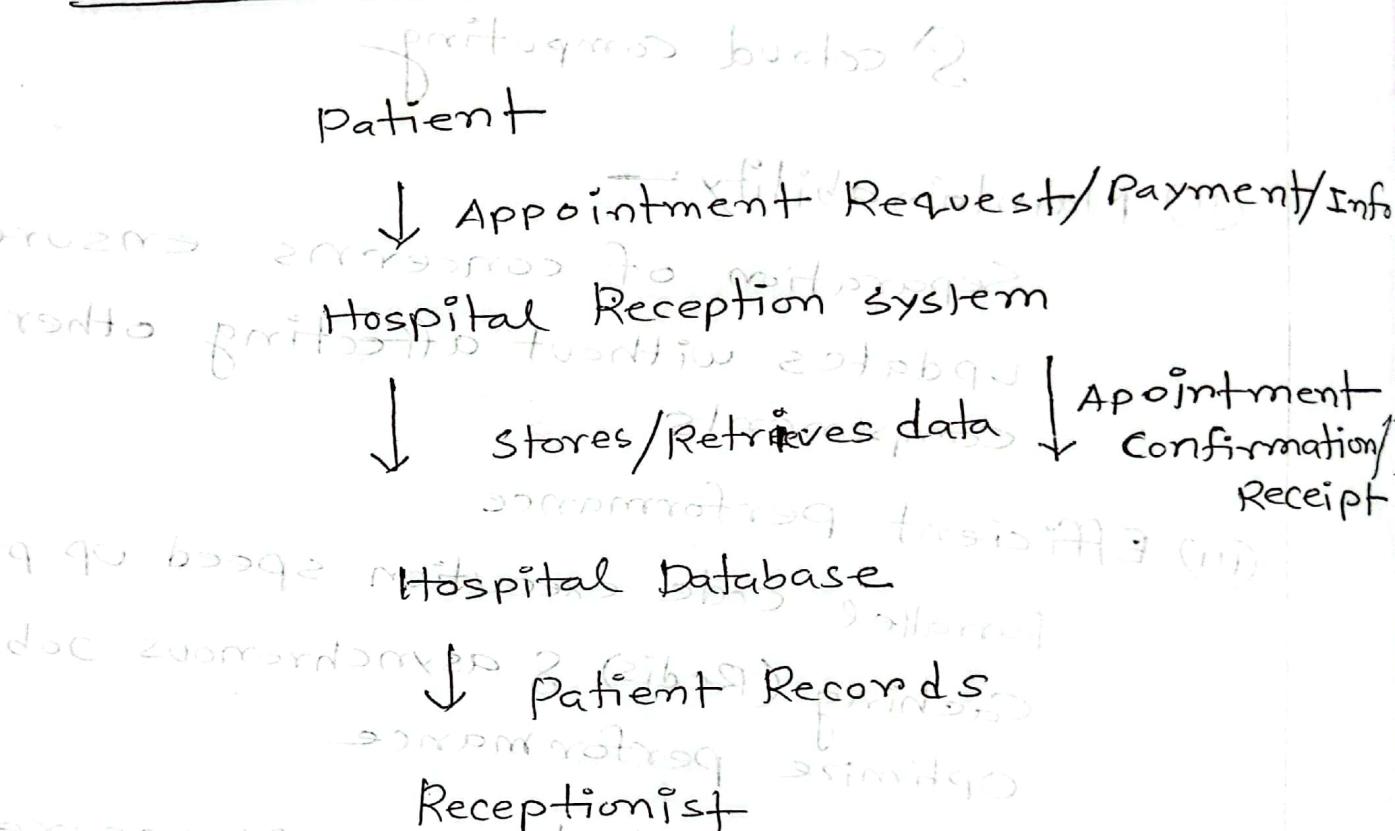
Feedback loop

Feedback loop

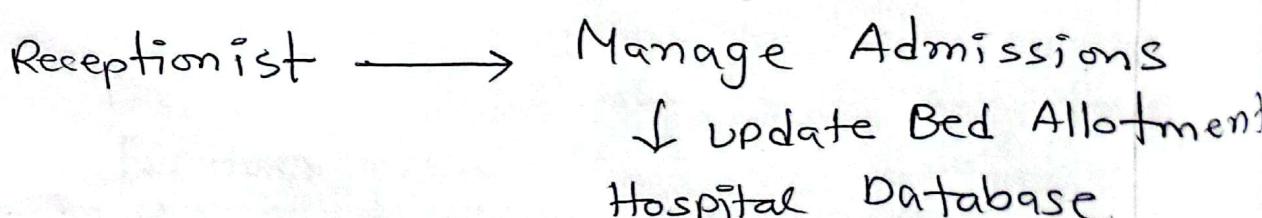
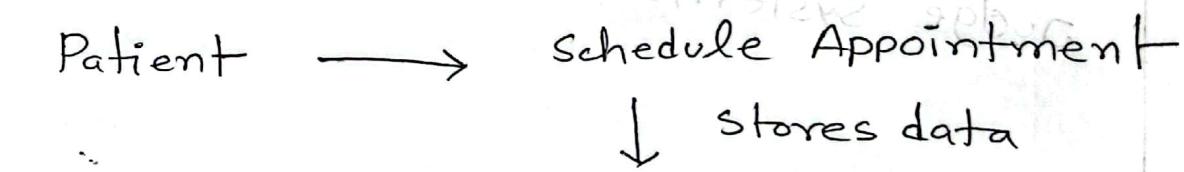
Feedback loop

15. For a Hospital Management System the DFD (Level 0 and Level-1) diagrams and UML use case diagram will be Here;

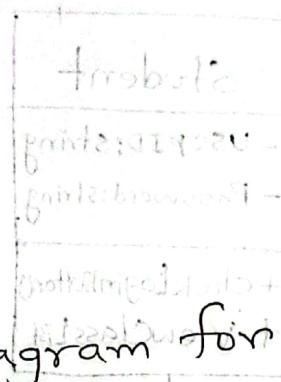
### DFD (Level-0) Context Diagram:



### DFD Level-1 (Detailed Breakdown):

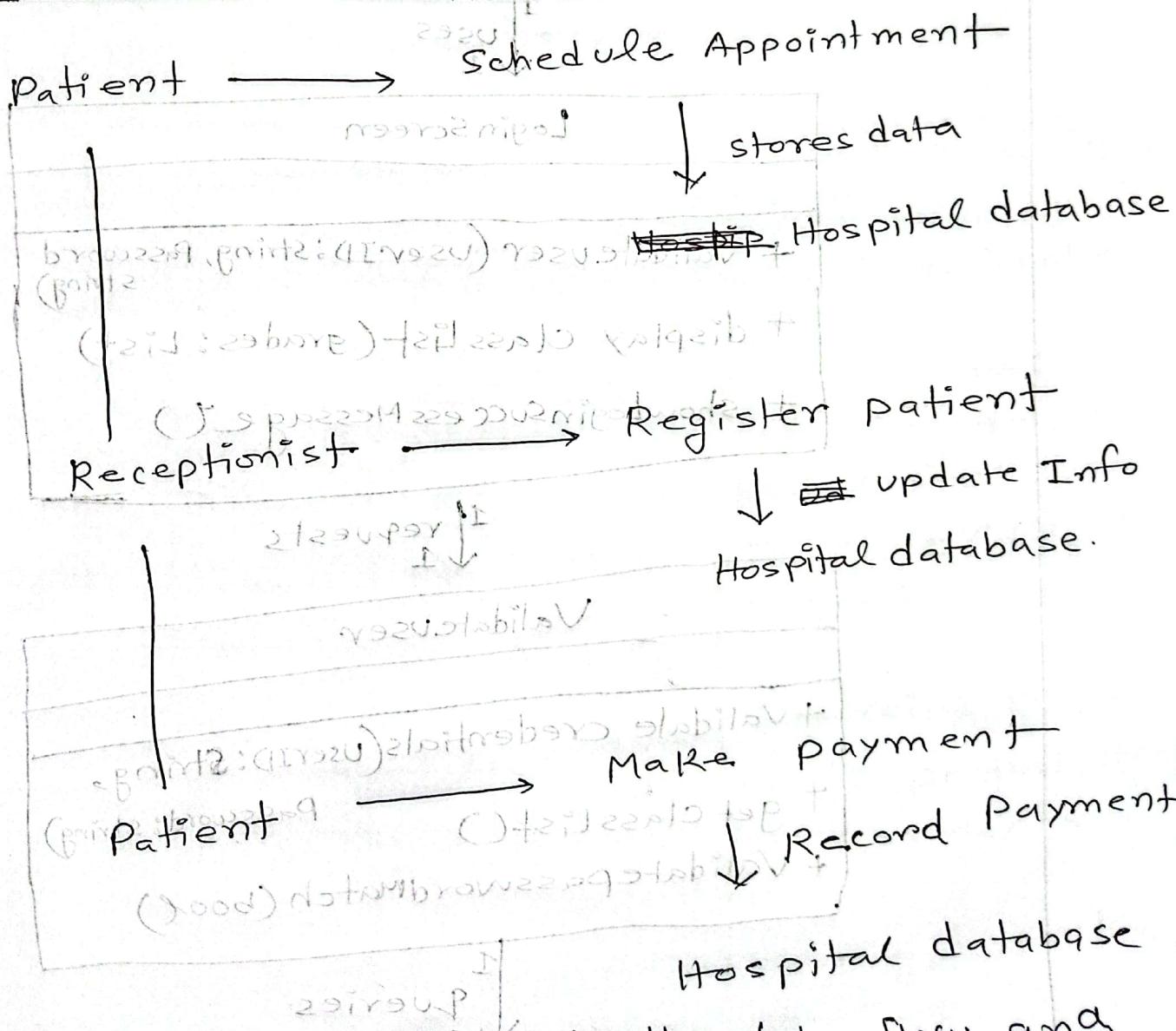


Patient → Process payment & claims



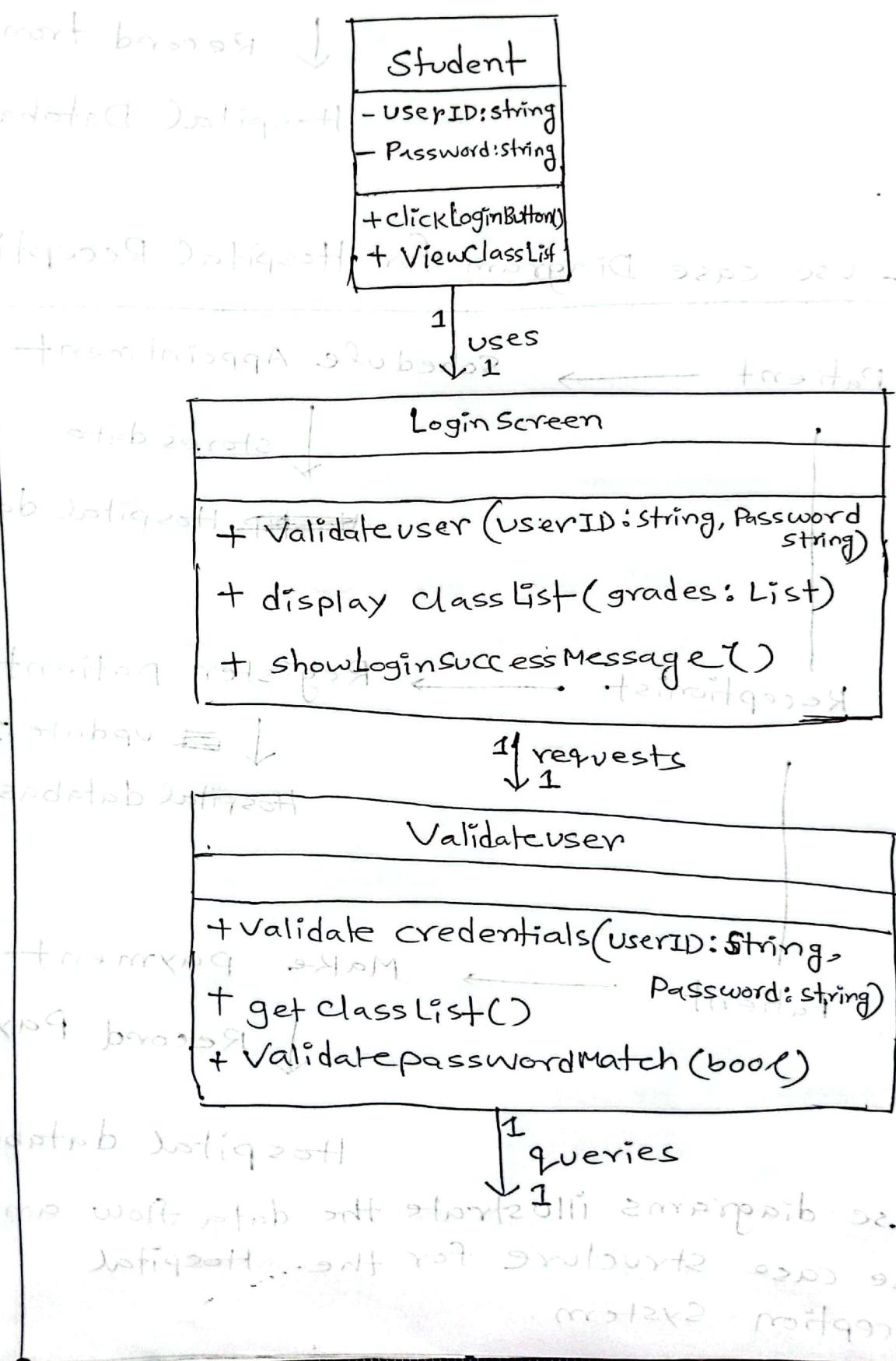
↓ Record transaction  
Hospital Database.

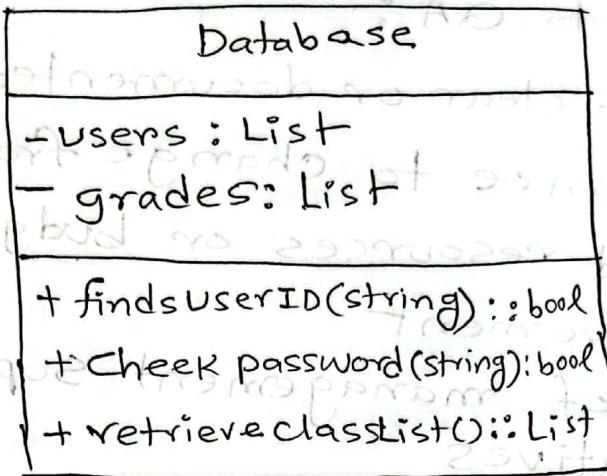
UML Use Case Diagram for Hospital Reception:



These diagrams illustrate the data flow and use case structure for the Hospital reception system.

(16) UML Class diagram for given scenario:





### (17) Difference Between QA and QC

Aspect	Quality Assurance (QA)	Quality Control (QC)
Definition	Process oriented activities to ensure quality	Product oriented activities to identify defects.
Focus	Preventing defects in processes	Detecting and correcting defects in products
Nature	Proactive (prevention-based)	Reactive (detection-based)
Goal	Improving processes to ensure consistent quality	Ensuring that the final product meets requirements
Methods	Standards, guidelines and process audits	Testing, inspection, reviews

## Impediments to QA:

- (I) Lack of clear or documented process
- (II) Resistance to change from team members
- (III) Limited resources or budget for process improvement
- (IV) Lack of management support for QA initiatives

## Impediments to QC:

- (I) Insufficient testing time due to tight deadlines.

(II) Poorly defined product requirements or scope

(III) Inadequate test cases or tools of skilled testers

(IV) Lack of resources

(18) Role of Quality Assurance (QA) at each phase of SDLC:

### (i) Requirement Analysis:

- QA ensures that the requirements are clear, complete and testable.
- Reviews and verifies requirement documents to avoid ambiguities.

### (ii) Design Phase:

- QA reviews system and software design to ensure alignment with requirements.

### (iii) Development Phase:

- QA ensures adherence to coding standards and guidelines.
- Conduct static testing, such as code review and inspection.

(iv) Testing Phase: QA test the entire software.

### (v) Testing Phase

- QA verifies that the product is ready for deployment through final testing.
- Ensures the deployment process follows the defined procedures.

Maintenance Deployment Phase:

- QA verifies that the product bug fixed and updates through regression testing

The goal of QA is not just to find bugs but to ensure quality throughout the SDLC by preventing defects, improving processes and validating that the software meets the customer's expectations.

(19)

## Rapid Application Development (RAD) Model

Key phases:

(I) Business Modeling: Define Business needs and Processes

(II) Data Modeling: Identify and structure data requirements

(III) Process Modeling: Design workflows for data manipulation

(IV) Application Generation: Build Prototypes using automated tools

(V) Testing and turnover: test and deploy with user feedback

Principles:

(I) User involvement and feedback

(II) Iterative prototyping

(III) Modular design for flexibility

(IV) Automated tools for rapid development

Advantages:

(I) Faster Delivery: Prototypes and automation reduce development time

(II) User-Centric: Continuous user feedback ensures satisfaction

Flexibility: Adapts to changing requirements

Quality Assurance: Frequent testing detects and resolves issues early.

RAD ensures fast delivery by prioritizing iterative development, user collaboration, and automation while maintaining quality through and feedback.

(20)

Test Table for given code:

Decision	X input	Y input
if ( $y == 0$ )	5	0
else if ( $x == 0$ )	0	2
if ( $i \% y == 0$ )	4	2

## Test JUnit 1 class in Java

```
import static org.junit.jupiter.api.Assertions.*;  
import org.junit.jupiter.api.Test;  
  
class DecisionTest {  
  
    void testyiszero() {  
        if (0 == x) {  
            int x = 5, y = 0;  
            String result = checkCondition(x, y);  
            assertEquals("y is zero", result);  
        }  
        if (++x == 5) {  
            int x = 0, y = 2;  
            String result = checkCondition(x, y);  
            assertEquals("x is zero", result);  
        }  
        if (x % 2 == 0) {  
            int x = 4, y = 2;  
            String result = checkCondition(x, y);  
            assertEquals("2 in m", result);  
        }  
    }  
}
```

Private string checkConditions (int x, int y) {

    StringBuilder output = new StringBuilder();

    if (y == 0) {

        output.append("y is zero");

}

    else if (x == 0) {

        output.append("x is zero");

}

    else {

        for (int i=1; i<=x; i++) {

            if (i%y == 0) {

                output.append(i).append("n");

            }

}

    }

    return output.toString();

}

}

}

(2)

## Exception testing:

- The `@ test(expected = exception.class)` annotation is used to test that an exception is thrown when expected
- You also can use the `try catch . block` inside the test method to check for specific exception if needed.

## Setup function:

- The `@ Before` annotation defines a setup method, which is run before each test. This is useful for initializing shared resources, test data or mocks that are used in multiple tests.

## Timeout Rule:

- The `@ test` annotation is used to specify that a test should fail if it takes longer than the specified timeout period.
- Alternative the rule can be used to apply the timeout in a more flexible manner.