Assignment

Software Engineering

ICT-3209

Name : Sagor Roy

ID : IT-21044

(1)

(i) Product Backlog for user stories

User story 1 : secure login

a) Design login UI (To Do)

b) Implement authentication logic (In progress)

c) Integrate database for user credentials (To Do)

d) Implement password Hashing and Security (To Do)

e) Test login functionality

f) Deploy and Review

User story 2: product search by category

a) Design search UI (To Do)

b) Implement category based filtering logic (To Do)

c) Connect with product database (To Do)

d) Optimize search performance (To Do)

e) Test search functionality (To Do)

f) 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.

(iii) Scrum Board Tracking:

- To Do : Tasks planned but not started
- In progress : Tasks currently being developed.
- Done : Completed task

(2)

Comparision of Spiral, Agile and Extreame Programming for Risk Management.

(i) Spiral Model:

- Risk Management: User iterative cycles where risks are identified and mitigated early
- Adaptibility : Allows changes after each spiral, but major modification can be costly
- Best for: Project with high technical risks needing frequent risk assesment

(ii) Agile Methodology:

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

- Adaptibility: Highly flexible. evolving requirement can be incorporate through regular client collaboration.
- Best for: Projects with evolving client needs and a focus on quick delivery

(II) Extreame programming:

- Risk Management: Uses test driven development and pair programming to minimize defects and risks.
- Adaptibility: Changes are easily accommodated due to continuous integration and refactoring
- Best For: High risk projects requiring rapid changes with strict coding dee discipline.

In this case Agile is the Best choice Because it balances risks management and Adaptibility and frequent testing and integration reduce risk impact.

(3)

Based of 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 adaptibility 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 / EEE 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 method.

(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 10000+ Concurrent users with a response time.

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

(iii) Usability: UI should be intutive, 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 ~~freq~~ performance, strong security and an excellent user experience.

(8)

Process improvement cycle in software Engineering:

Process improve cycle is a systematic approach to enchancing software development efficiency and quality. It follows these key stages:

(i) Process Assesment - 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 of 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 matrics, software teams can enhance efficiency, quality and customer 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 stage of prototype Development

(I) Requirement Gathering And analysis:
- identifies core system functionalities based on user needs

(II) Quick Design:
- Simple user interface will 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.

Risk Reduction:

- Identifies potential issues early in developmen

Iterative Development & Flexibility:

- Changes can be incorporate at an early stag
- Ideal for evolving requirements and uncertai specialization.

The prototyping modeling enhances user satisfaction, reduce project risks.

(9) Software Engineering institute capability Maturity Model (SEI CMM)

The capability Maturing Model (CMM) developed by the software engineering institute (SEI) is a framework for assessing and improving an organizations 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-mangement 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.

(10) Core principle 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 Negociation;
- Engaging users throughout development

(IV) Responding to change over following a plan

Application of Agile principles in diffrent Environment:

(I) startup & small teams: Agile enables fast iteration and quick market adaption.

(II) Large Enterprises: Framework like SAFe (scaled Agile framework) help co-ordinate multiple teams.

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

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 deal for dynamic projects.

(ii) Release cycle of Extreame 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)

↓

Release — (Deploy working software for feedback)

↓

Maintenance — (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 — Avoids unnecessary complexity make the code easy to modify

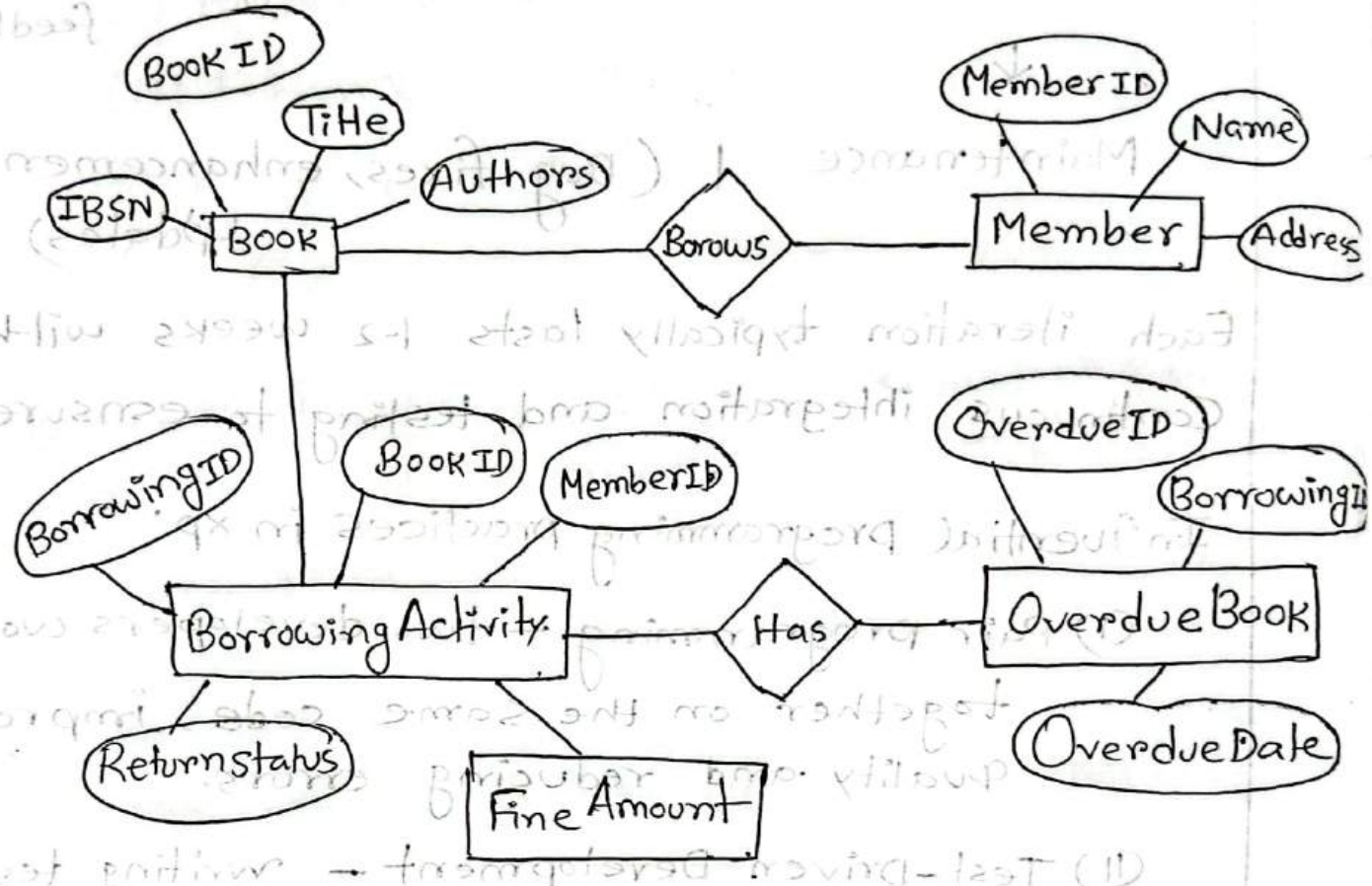
(V) Refractoring

(VI) Collective code ownership

(VII) Sustainable pace

(VIII) Customer collaboration.

(12) Entity Relationship Diagram for Library Manag



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

(13)

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

Validation	Verification
(i) Ensures the software is build correctly according to specification	(i) Ensures the built software meets user needs and expectations
(ii) checks compliance with design, code and documentation	(ii) Ensures functionality, usability and performence
(iii) Reviews, walktroughs and inspections which means static	(iii) Dynamic - Actual execution of the software
(iv) Performed before development	(iv) Performed after development
(v) code Reviews, requirement analysis, design and inspection	(v) functional testing, user acceptance testing
(vi) Cheeking if the software is designed as per requirements	(vi) Cheeking 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 scalibility, 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 & coloud computing

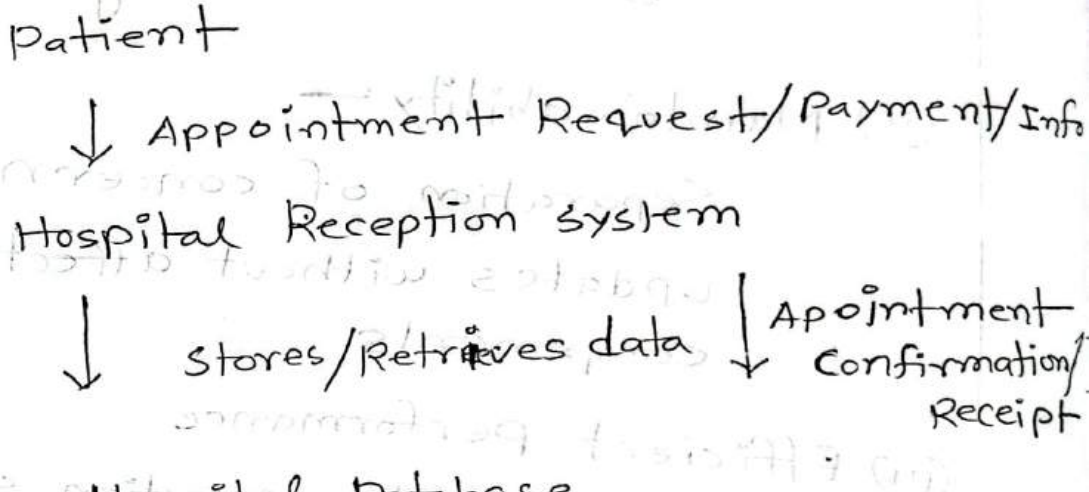
(II) Maintainability :—
Separation of concerns ensures easy updates without affecting other components

(III) Efficient performance
Parrallel code execution speed up processi
Caching (Redis) & asynchronous Job queues optimize performance

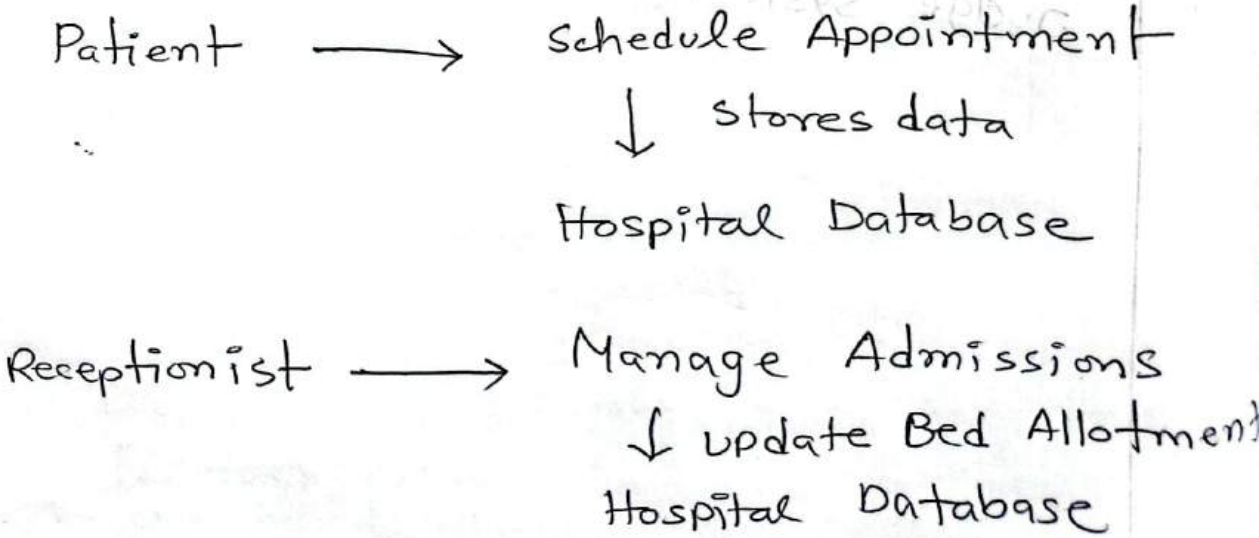
This layered Architecture ~~esn~~ ensures a robust, scalable and maintainable online judge system.

15. For a Hospital Mangement 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):

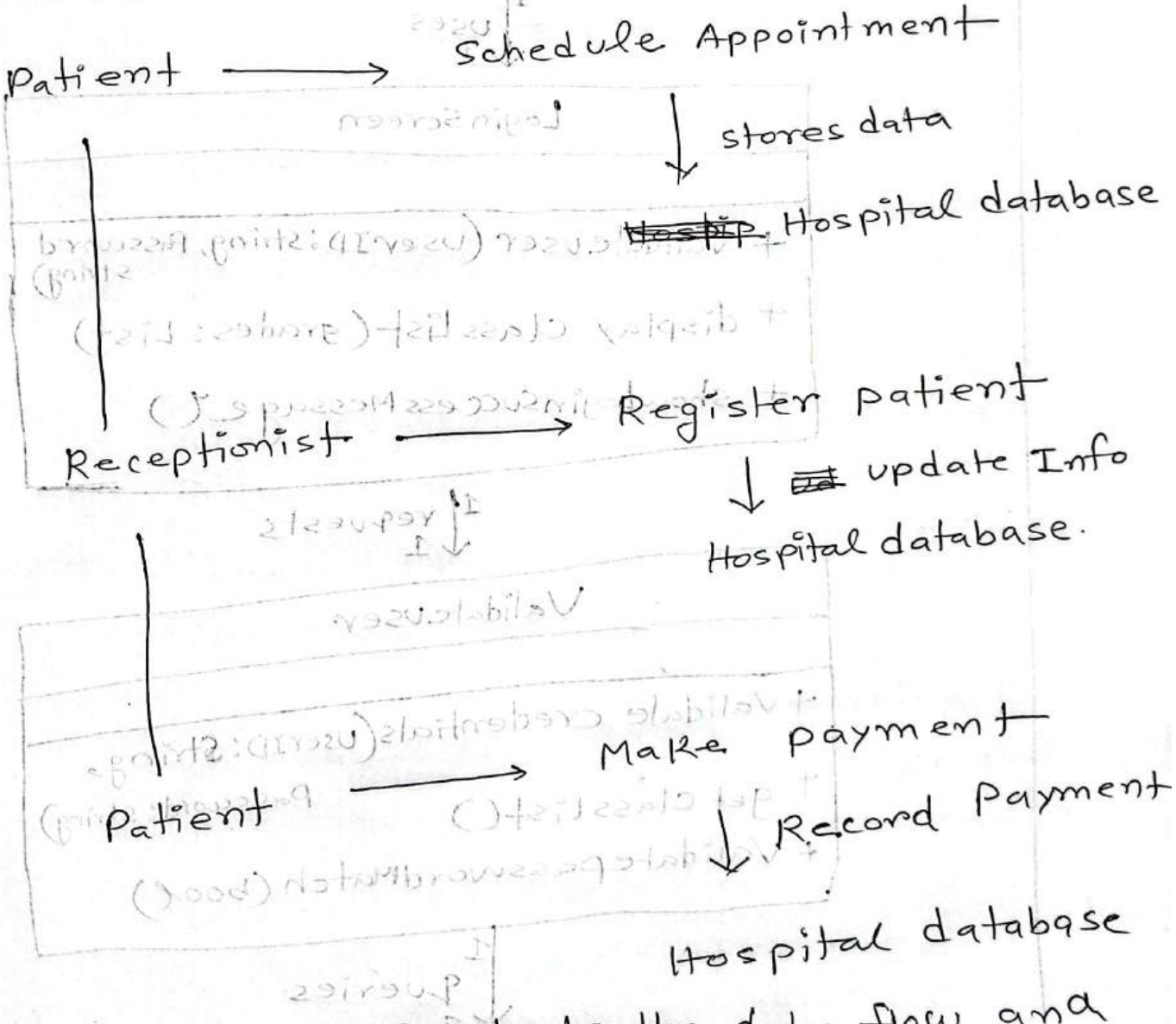


Patient ⟶ Process payment & claims

↓ Record transaction

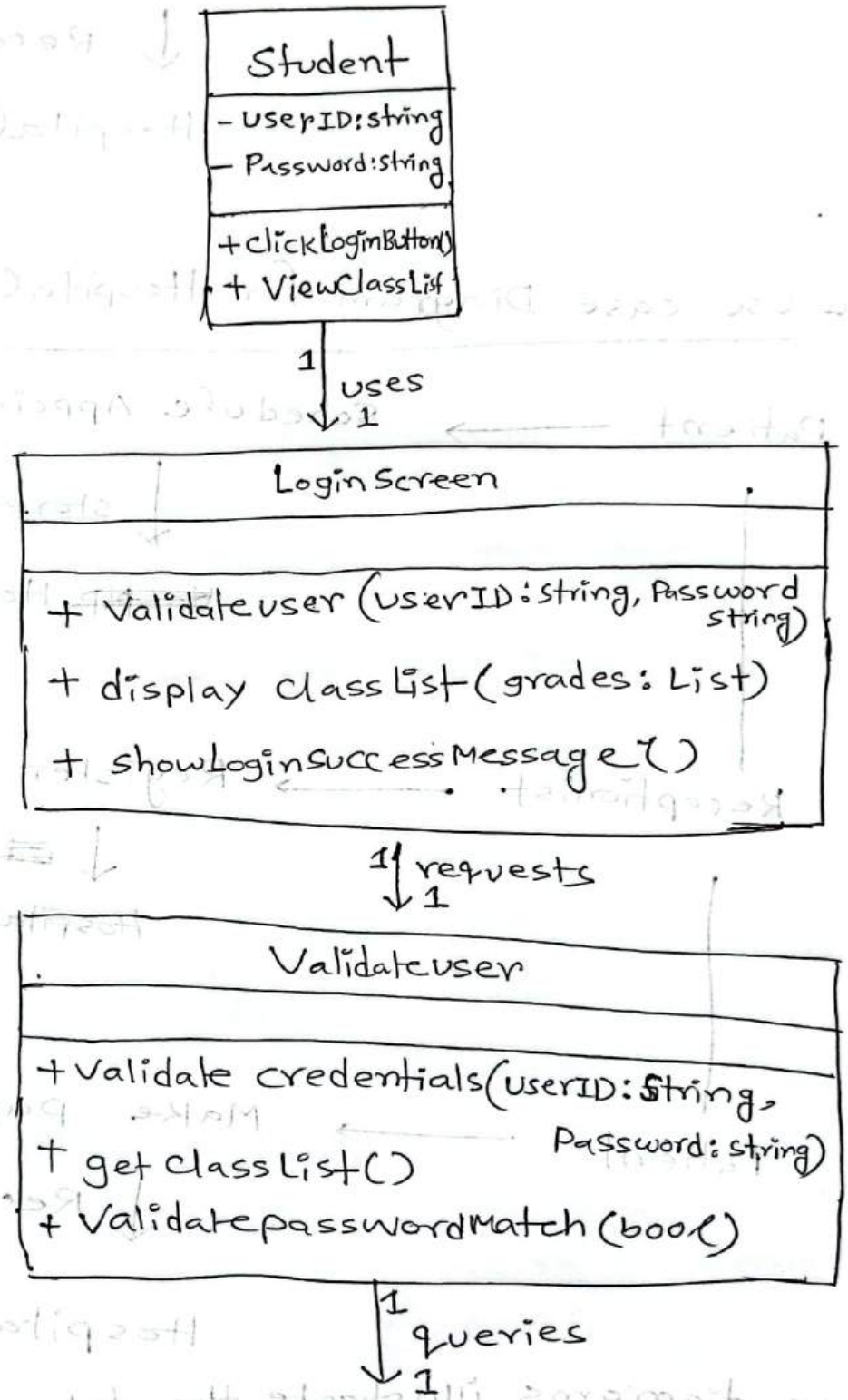
Hospital Database.

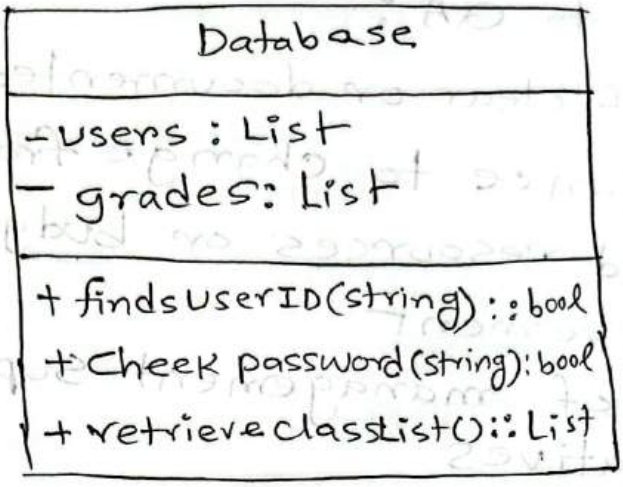
UML Use case Diagram for Hospital Reception:



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

(16) UML Class diagram for given scenario:





(17) Diffrence Between QA and QC

Aspect	Quality Assurance (QA)	Quality Contral (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	Ensureing that the final product meet requirements
Methods	standards, guideling and process audits	Testing, inspection, reviews

Impediments to QA:

(I) Lack of clear or documented process
(II) Resistance to change from team member
(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.
(IV) Lack of skilled testers

(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 allignment with requirements.

(III) Development phase:

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

(IV) Testing Phase: deployment
- 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.

(v) ~~Deployment~~ Maintanance 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 throught the SDLC by preventing defects, improving Processes and validating that the software meets customer 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 manupulation

(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 throught 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

JUnit Test Class in Java

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

class DecisionTest {
    void testyiszero(){
        int x=5, y=0;
        String result = checkcondition(x,y);
        assertEquals("y is zero", result);
    }

    void testxiszero(){
        int x=0, y=2;
        String result = checkcondition(x,y);
        assertEquals("x is zero", result);
    }

    void TestxModyiszero(){
        int x=4, y=2;
        String result = checkconditions(x,y);
        assertEquals("2 in 4 m", result);
    }
}
```

```
Private string checkconditions (int x, int y){
    StringBuilder output = new StringBuild
  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();
}
}
```

(21)

Exception testing:

- The @ test (expected = exception.class) annotation is used to test that an exception is thrown when excepted
- 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 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.