Assignment

Software Engineering

ICT-3209

Name: S M Nasimul Hasan

ID: IT-21026

(1) "Nasimul"

(i) Product Blocking 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.

d) Implement password hashing and security.

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.

c) Connect with product database.

d) Optimize search performance.

e) Test search functionality.

f) Deploy and review.

(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.

(iii) Scrum Board Tracking:

- To Do: Tasks planned but not started
- In progress: Task currently being developed.
- Done: completed tasks.

(2) Comparison of spiral, Agile and Extreme programming for risk management.

(i) Spiral Model:

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

(ii) Agile Methodology:

- Risk management: continuous feedback reduces uncertainity; working software is delivered in short iterations.

- Adaptability: Highly flexible, evolving requirement can be incorporate through regular client collab.

iii) Extreme Programming:

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

In this case, Agile is best choice because it balances risks management and adaptibility. and frequent testing and integration reduce risk impact.

"Nasimul"

3. Comparison of development models:

1. Waterfall - A sequential, phase-based model best suited for projects with fixed requirements and strict deadlines. It offers predictability but lacks flexibility.

2. Agile :- An iterative approach that emphasizes adaptibility, customer collaboration and incremental deliveries, making it ideal for evolving require.

3. Extreme Programming (XP) - A type of Agile with short development cycles, continuous testing and high customer involvement. suitable for rapidly changing projects.

4. Spiral - A risk driven model combining iterative development with formal risk analysis, useful for complex with high uncertainity.

Best fit for each project: "Narimul"

* Project A :

. Since the requirements are clear and fixed, waterfall ensures a structured approach

with predictable timelines and deliverables.

- Risk is lower due to well difined scope, and customer feedback is not a priority during development.

* Project-B:

- Agile allows for adaptibility as customer needs change and enables frequent iterations for continuous improvements.
- XP work well if rapid development and testing cycles are required.
- Spiral could also be considered if risk management is a significant concern.

So, Waterfall is best choice for project-A.
and Agile or XP is best choice for Project-B.

(4) Principals of Software Engineering Ethics:

1. Public Interest: Prioritizing user safety, privacy and social well-being.
2. Integrity and honesty: Providing accurate estimates, avoiding deception and risks.

3. Confidentiality: Protecting sensitive data of clients and users.

4. fairness and Respect: Avoiding discrimination and ensuring fair treatment of colleagues and users.

Role of ACM/IEEE code of Ethics.

- Encourage the software engineers to act in the publics interest.
- Promoting honesty, fairness and integrity.
- Requiring disclosure of risks and avoidance of harm.
- Emphasizing lifelong learning to maintain competence.

(4)

Functional Requirements:

1. User registration and Authentication: Ensures secure access for passengers, reducing unauthorized use.
2. Flight Search and booking: Allows users to find and reserve flights efficiently.

3. Payment processing: Ensures secure transactions, enhancing customer trust and financial integrity.

4. Cancellation and Refund Management: Provides flexibility for users, improving customer satisfaction

Non-functional Requirements:

1: Performance and Scalibility: Handles high traffic efficiently, ensuring user experience

2. Security data protection: Implements encryption and authentication to protect user data

3. Usability and accessibility: Ensures an intuitive interface for diverse users, improving ease of use

4. Maintainibility and extensibility: Allows easy update and feature enhancements for long-term sustainibility.

(c)

The V-model (Verification and Validation model): is an extension of the waterfall model, where each development phase has a corres. testing phase. The process follows a V-shape, ensuring early defect detection.

Phases and Corresponding Testing Activities:

1. Requirement Analysis → Acceptance Testing
   - Ensures the system meets user needs.
2. System Design → System Testing
   - Verifies overall system functionality.
3. Architecture Design → Integration Testing
   - Ensures components interact correctly.
4. Module Design → Unit Testing
   - Test individual module for correctness.
5. Coding → Execution of Tests.
   - Developers implement code & execute planned tests.

(7) Prototyping is an iterative approach where a preliminary version of software is built to refine requirements before full scale development.

Key stages of prototyping:

1. Requirements gathering.
2. Quick Design
3. Prototype Development
4. User Evaluation
5. Refinement and iterations.

Benefits of Prototyping model:

* User feedback * Risk Reduction * iterative development

(8) Process improvement cycle in Software Engineering.

The process improvement cycle is an iterative approach to enhancing software development efficiency and quality.

Key stages:

1. Process Assesment 2. ~~Getting Started~~ Goal setting.
3. Process Redesign 4. Implementation
5. Monitoring & evaluation 6. continuous Improvement.

* Common Process Metrics and their importance:

1. Defect Density: Measures defects per module/LOC.
2. Cycle Time: Tracks the time taken from development to deployment, improving efficiency.
3. Effort Variance: Compares estimated vs. actual effort, aiding project planning.
4. Customer Satisfaction: Evaluates user feedback to improve software usability.
5. Rework Percentage: Indicates the amount of work needing correction, identifying process inefficiencies.

(2) SEI Capability Maturity Model CMM

The SEI CMM is a framework that assesses an organization's software development maturity, helping to improve processes systematically. It consists of five levels.

1. Initial Level: Identifies the need for structured processes.
2. Repeatable: Basic project management practices are established.

3. Defined (Level 3): Organization wide standardized processes are implemented.

4. Managed: Data driven decision-making enhances predictibility and quality.

5. Optimizing: Continuous process improvement through innovation and feedback.

10 Core principals of Agile Software Development:

1. Individuals and Interactions over process and Tools.
2. Working software over comprehensive Doc.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

Benefits:

- Faster delivery and continuous improvement.
- Higher customer satisfaction due to iterative feedback.
- Enhanced team collaboration and motivation.

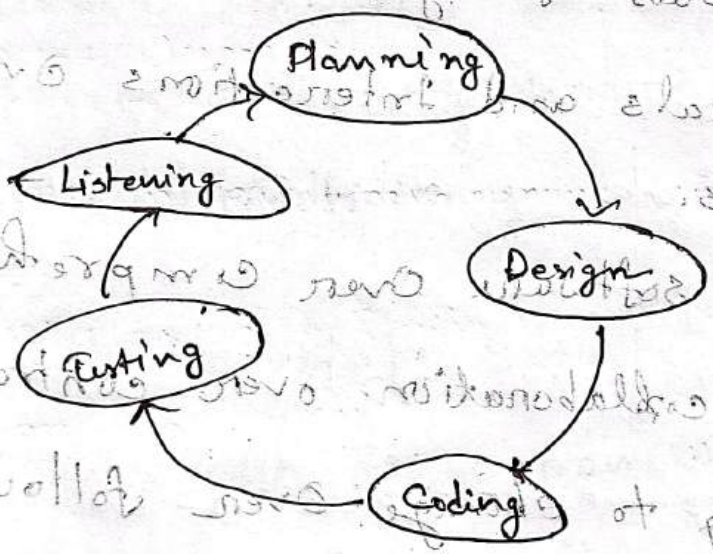
Narimul

Challenges:

- Requires cultural shift and strong team discipline.
- Difficult to implement in highly structured or regulated environments.
- Scalability can be complex for large organizations.

11

* Drawing the Life cycle of XP:



* Planning: Define stories of user and prioritize features
* Design: Create simple, adaptable designs
* Coding: Develop incrementally with pair programming and continuous integration.
* Testing: Perform unit and acceptance testing.

*Listening: Incorporate user feedback for the next cycle

12

* Entity-Relationship Diagram for a Digital Lib

Management System:

Entities and their Attributes

1. Book (Tracks book details)
   - Book ID (Primary key)
   - Title
   - Author
   - ISBN
   - Genre
   - Status

2. Member:
   - MemberID (PK)
   - Name
   - Contact
   - Statu

3. Borrowing:
   - BorrowID (PK)
   - MemberID
   - BookID
   - BorrowDate
   - DueDate
   - ReturnDate
   - Sta

4. Overdue Books:
   - OverdueID (PK)
   - BorrowID (Foreign key)
   - fine amount
   - Payment Status.

"Nasimul"

*Listening: Incorporate user feedback for the next cycle.

12

* Entity-Relationship Diagram for a Digital Library Management System:

Entities and their Attributes

1. Book (Tracks book details)
   - Book ID (Primary key)
   - Title - Author - ISBN - Genre - Status
2. Member:
   - MemberID (PK) - Name - Contact - Status
3. Borrowing:
   - BorrowID (PK) - MemberID - BookID
   - BorrowDate - DueDate - ReturnDate - Status
4. Overdue Book:
   - OverdueID (PK) - BorrowID (Foreign key)
   - fine amount - Payment Status.

"Nasimul"

(B) (5)

Testing is the process of evaluating a software system to identify defects and ensure it meets specified requirements. It involves executing code under controlled conditions.

Difference Between Validation & Verification:

Verification:	Validation:
(i) Ensures the software meets design specifications.	(i) Ensures the software meets user requirements.
(ii) Process Oriented	(ii) Product Oriented
(iii) Done in early development	(iii) Done after testing
(iv) Ensures correctness of design and architecture	(iv) Ensures software meets user needs and useful.

"Narimuth"

(5)

Testing is the process of evaluating a software system to identify defects and ensure it meets specified requirements. It involves executing code under controlled conditions.

Difference Between Validation & Verification:

Verification:	Validation:
(i) Ensures the software meets design specifications.	(i) Ensures the software meets user requirements.
(ii) Process Oriented	(ii) Product Oriented
(iii) Done in early development	(iii) Done after testing.
(iv) Ensures correctness of design and architecture	(iv) Ensures software meets user needs and useful.

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

2. Application Layer:
   - Manages user requests and directs them to appropriate services.

3. Business Logic Layer
   - Executes and compares users submitted code and assigns a pass/fail score.

4. Data Layers:
   - stores user submission problem sets, test cases and results.

"Narsimul

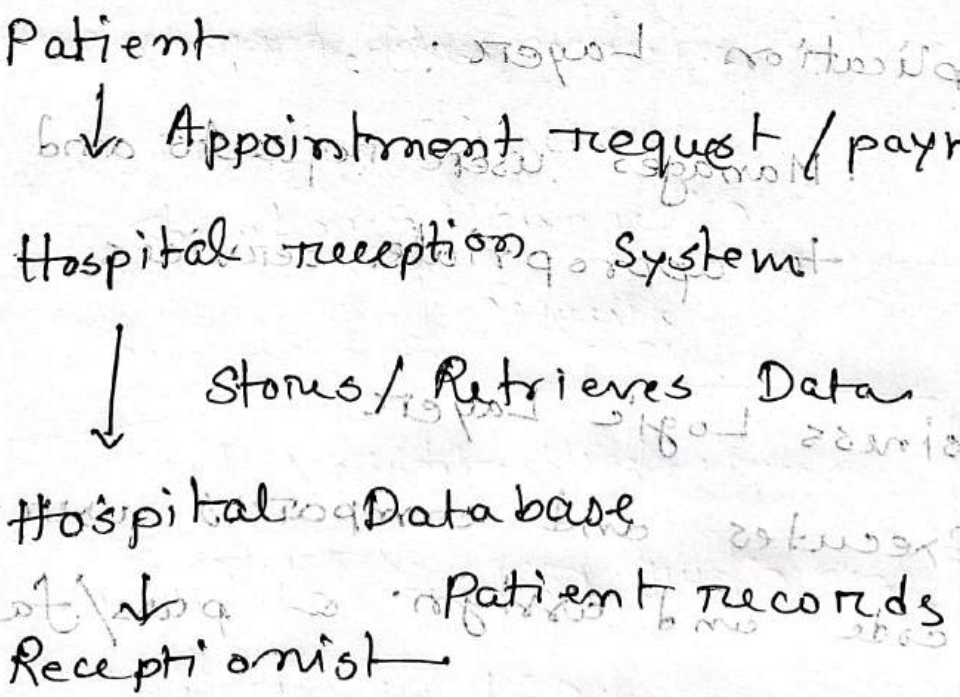
* This architecture ensures efficiency by
(i) Scalability
(ii) Maintainability
(iii) Efficient performance

— o —

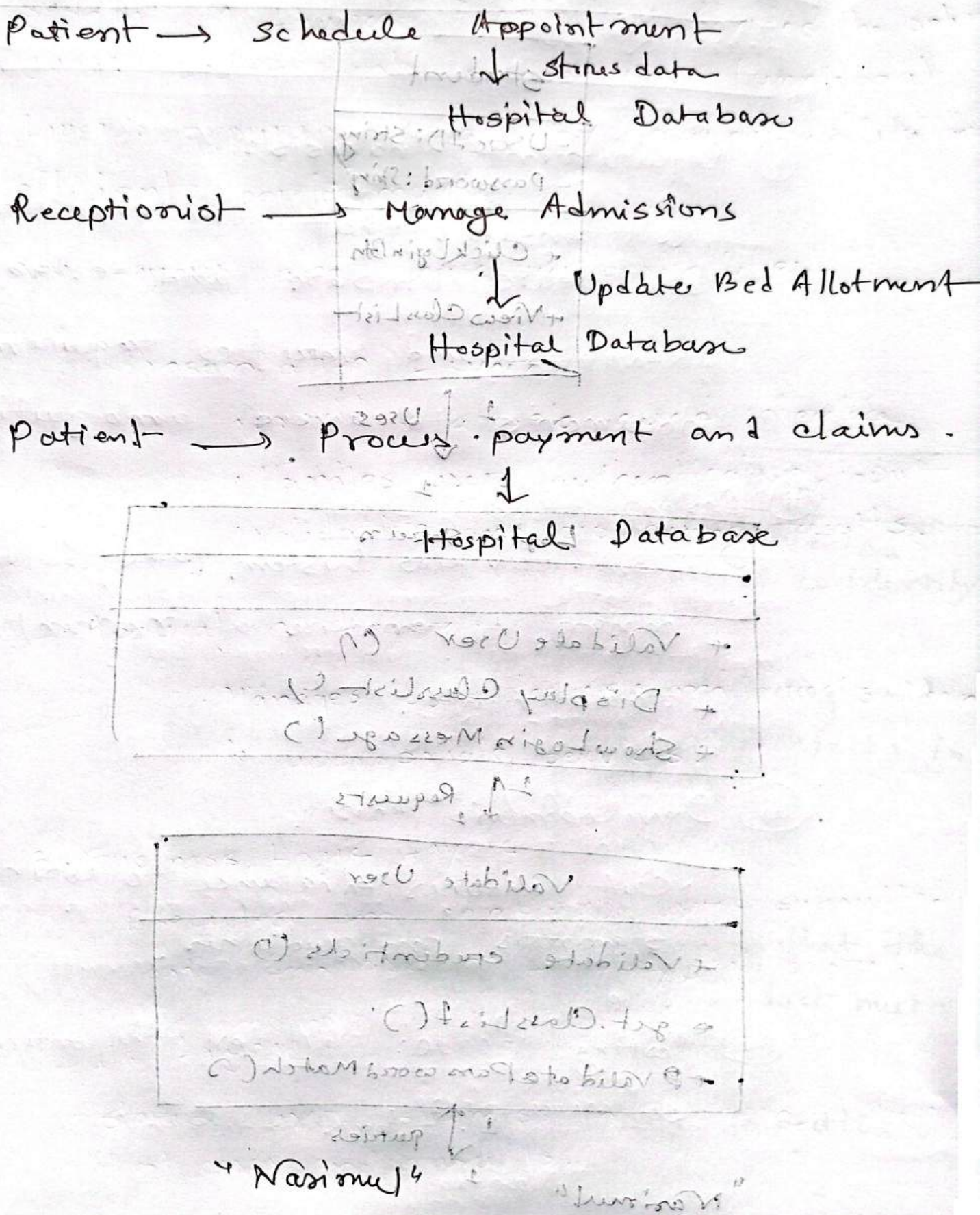
15

for a hospital management system the DFD diagrams and UML use case digram will be there:

DFD (Level - 0) Context Diagram:



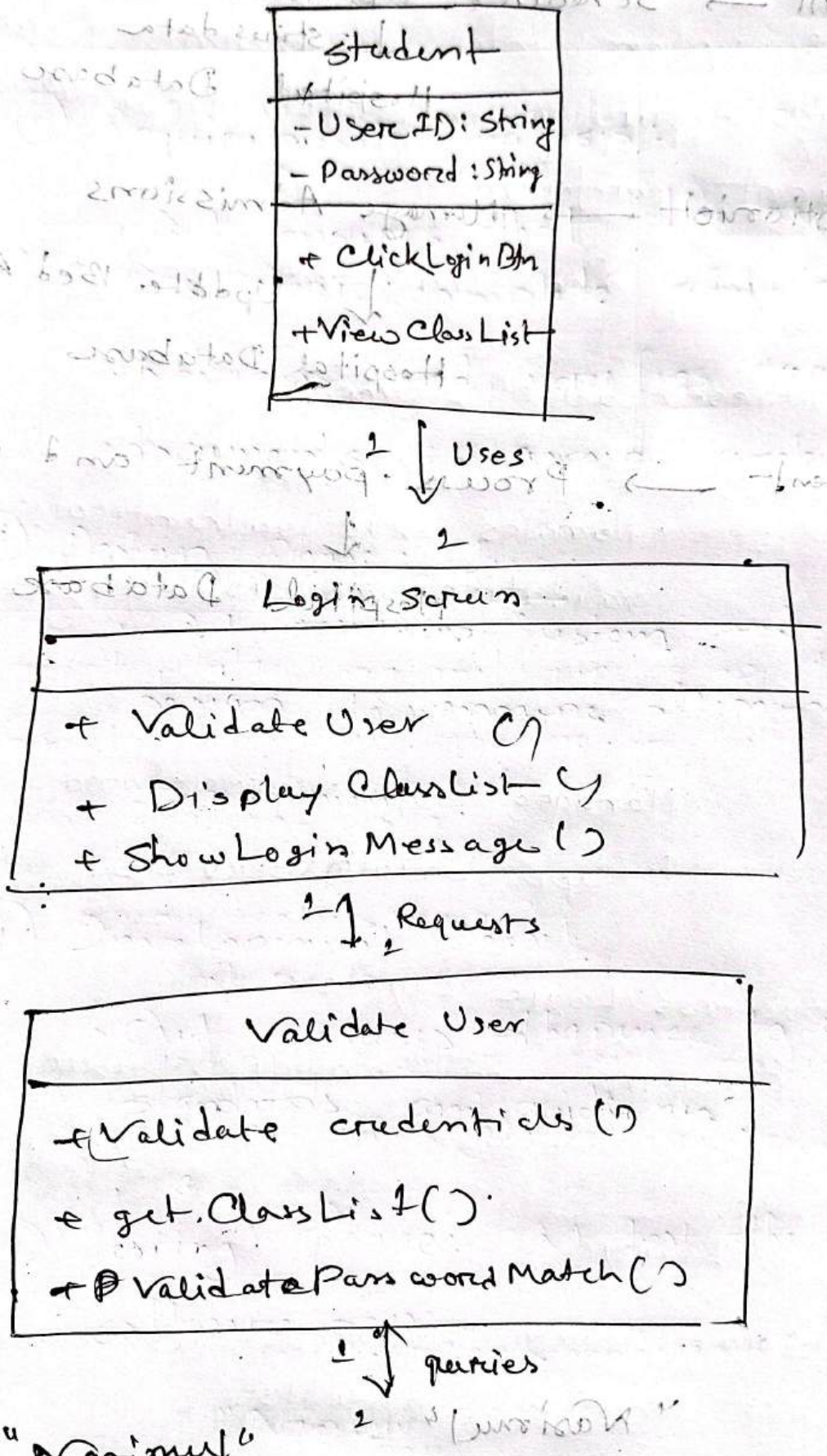
DFD Level-1



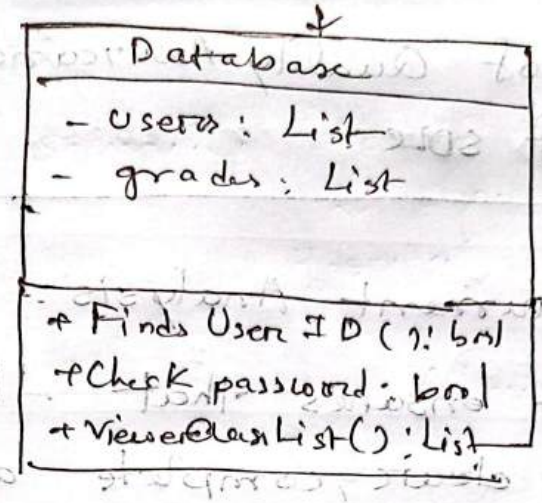
Nasima

16

UML class Diagram for given Scenario:



"Nasimul"



(17) Difference Between Q_A and Q_C

	Q_A	Q_C
	Process oriented activities to ensure quality	Product oriented activities to identify defects
	Preventing defects in processes	Detecting and correcting defects in products
	Proactive	Reactive
	Improving processes ensure consistent quality	Ensuring that the final product meet requirements
	Standards, guidelines and process audits	Testing, inspection, reviews.

"Nasimul"

'Nasimul

18. Role of Quality Assurance 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 requirement.

(iii) Development phase:

- QA ensures adherence to coding standards and guidlines.

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

"Nasimul"

Nasimul

(v) Maintenance Phase:

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

[19]

Rapid Application Development Model:

Key Phases:

(i) Business modeling

(ii) Data Modeling

(iii) Process Modeling

(iv) Application generation

(v) Testing and Turnover.

Principals:

(i) User involvement

(ii) Iterative prototyping

(iii) Modular design of flexibility

(iv) Automated tools for rapid development

Nasimul

20

Test Table for given code:

Decision	x input	y input
if (x==0)	3	0
else if (x==0)	0	2
if (x.y==0)	4	2

JUnit Test Class in Java:

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

class Decision_Test {

    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 TestXMody_is zero () {
    int x = 4; y = 2;
    string result = check conditions(x, y);
    assertequals ("2\n4\n", results);
}

Private string check conditions (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");
            }
        }
    }
}
```

Narimul

return output.toString();

}

}

21

Exception Testing:

- The @test (expected = exception.class)' annotation is used to test that on 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 functions:

- 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.

Time Out Rule:

- The @test annonatation 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.

"Nasimul"