Software Engineering Assignment

Question - 01

In a Scrum-based software development project, the Product Owner has defined the following user stories for an e-commerce application:

- As a user, I want to log in securely so that I can access my account.
- As a user, I want to search for products by category to find items easily.

I. Create a product backlog for these user stories by breaking them into tasks.

II. Describe how the development team can prioritize these user stories during a Sprint Planning Meeting. Considering value to the customer and technical feasibily.

III. Illustrate how these task's will be tracked using a Scrum board. Use proper terms like "To Do", "In Progress" and "Done".

Sheth Saha (IT-21009)

Answer:

'I'

For the e-commerce application, the Product Backlog is created by breaking the user stories into actionable tasks. These tasks are small enough to be worked on during a Sprint and will help the development team complete each user story.

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

Task 1: Design the login page UI (e.g. fields for username, password and submit button)

Task 2: Implement client-side validation for login (e.g. check if fields are empty, validate email format)

Task 3: Set up backend authentication API (e.g. using JWT, OAuth, or session-based authentication)

Task 4 : Implement password hashing and secure storage (e.g. using bcrypt or Argon2)

Task 5 : Implement session management (e.g. handling login states with cookies or tokens).

Task 6 : Write unit tests for the login functionality (e.g. successful login, incorrect password, user not found)

Task 7 : Perform security testing (e.g. vulnerability checks like SQL injection, cross-site scripting).

Task 8 : Implement logout functionality and session expiration.

Shetu Saha (IT-21009)

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

Task 1 : Design the search UI with category dropdown or filter panel.

Task 2 : Create a database schema for product categories.

Task 3 : Implement search API to filter products by selected category.

Task 4 : Integrate frontend with the search API to display products dynamically based on selected categories.

Task 5 : Write unit tests for product search functionality (e.g. correct filtering, category not found)

Task 6 : Optimize search query for performance (e.g. indexing categories or using caching)

Task 7 : Test the search functionality for both accuracy and speed.

'II':

During Sprint Planning, the team will prioritize these user stories based on:

- Value to the Customer:

User Story 1 (Login): High priority. Secure log in is crucial for user trust and account access, enabling core functionalities like order history, saved items and personalized recommendations.

User Story 2 (Category Search): High priority. Effective product discovery is essential for a successful e-commerce platform. It enhances user experience and increases the likelihood of finding desired products.

- Technical feasibility:

User Story1 (Log in): Moderate complexity. Requires careful implementation of security measures and integration with authentication services.

Shatu Saha (IT-21009)

User Story 2 (Category Search):

Moderate to high complexity depending on the complexity of the product catalog and the desired search features.

"III"

The Scrum board will track the progress of tasks during the Sprint. The board consists of columns to visualize the workflow such as "To Do", "In Progress", and "Done".

Columns:

1. To Do : Tasks that have not yet started and are in the planning stage.

2. In Progress: Tasks that the development team is actively working on.

3. Done : Tasks that are completed and meet the Definition of Done (DoD), ready for review or deployment.

Shetu Saha (IT-21009)

To Do	In Progress	Done
User Story 1 (login):	User Story 1 (login)	User Story (Login):
-Design the login page UI	-Implement frontend validation	-Write unit test for login
-Implement password hashing and encryption	-Implement session management	-Implement password encryption.
-Conduct security testing	-Write unit tests for login	
User Story 2 (Search):		User Story 2 (search):
-Design search UI with category dropdown		-Write unit tests for search
-Create database schema for product categories		
-Implement search API		
-Integrate frontend with search API		
-Optimize search query performance		

Question-2:

A software development team is about to start a project for a new innovative product. The project has several high-risk components due to it's novelty, and there's uncertainly regarding the client's future needs. The client is open to interative changes, but the team must ensure that the software evolves in a manageable, cost-effictive way.

Considering the high risks and the evolving nature of the client's needs, discuss how the Spiral, Agaile and Extreme methodologies address risk management and adaptability. Which methodology would be the most suitable for a project with significant risk and evolving requirements, and why?

Answer: Risk Management and Adaptability in Spiral Agile and Extreme Methodologies:

- Spiral Methodology: focuses on risk management through frequent cycles of planning, risk analysis, and refinement. It's ideal for high-risk projects, because it proactively identifies and mitigates risks. It is adaptable due to iterative revisions, but it can be more formal and document-heavy.
- Agile Methodology: Uses short iterations (Sprints) and constant client feedback, allowing for ~~earg~~ early identification of risks and frequent adjustments based on evolving requirements. It is highly adaptable, making it ideal for projects with uncertain and changing needs.
- Extreme Programming (XP): Emphasizes continuous integration and test-driven development, ensuring high-quality code and minimizing defects. It's highly adaptable with close client collaboration

and frequent releases, but focuses more on technical practices and may not directly address broader project risks as effectively as Agile.

Given that the project in question has high-risk components and there is uncertainty regarding future client needs, Agile would likely be the most suitable methodology. Here's why:

1. Continuous Risk Mitigation:

Agile's frequent feedback loops allow the team to identify and address risks early. Risks are managed through short iterations, with each Sprint serving as an opportunity to adjust the approach based on client feedback and evolving requirements.

2. Flexibility and Adaptability : Agile's iterative process allows for flexibility, making it easy to adapt to new information or changes in client needs. The client is involved throughout the process, and requirements can be adjusted regularly based on real-world feedback, which is crucial for a project where the final product is not well-defined.

3. Cost-Effective Evolution :

Agile helps in delivering incremental improvements. This iterative approach ensures that the project evolves in a manageable, cost-effective way, allowing for changes to be incorporated without major disruptions.

Why not Spiral or XP ?

- Spiral is a great risk management approach, but it can be more formal and document-heavy than Agile, which might slow down adaptation, especially in a fast-changing, high-uncertainty project.

- XP is highly effective in ensuring high-quality code, but it may be overkill for a project where the client's evolving needs are the primary concern. XP emphasizes technical practices, which are excellent but might not fully address the need for iterative feature adjustments and feedback from the client.

Agile is the most suitable methodology for a project with significant risks and evolving requirements. Its ability to incorporate frequent client feedback, adjust priorities iteratively and manage risks through short development cycles makes it ideal for such high-risk, dynamic projects.

Comparison of Waterfall, Agile, Extreme and Spiral Development Models:

1. Waterfall Methodology:

• Overview: Waterfall is a linear and sequential development model. It involves distinct phases: requirements, gathering, design, implementation, testing, deployment and maintenance.

• Characteristics:

Predictability: Highly predictable for well-defined projects because each phase is completed before the next begins.

Customer Collaboration: Minimal customer involvement after the requirements phase

Risk Management: Limited risk management. Risks are addressed only at the beginning and end, which can be problematic if changes arise.

2. Agile Methodology

- Overview: Agile is an iterative and incremental approach. Development is split into short cycles (Sprints), and feedback is continuously incorporated into the process.

- Characteristics:

- Predictability: Less predictable in terms of timeline and scope, as priorities can change during each Sprint.

- Customer Collaboration: High customer involvement, with regular feedback loops to adjust features based on client needs.

- Risk Management: Focuses on managing risks by addressing them early in the process through continuous testing and feedback. Risks are identified and mitigated throughout the development.

3. Extreme Programming (XP)

- **Overview :** XP is an Agile methology that emphasizes technical excellence, continuous integration collaboration and frequent releases.
- **Characteristics :**
- **Predictability :** Less predictable due to frequent changes, but the focus on code quality ensures stability despite continuous changes.
- **Customer collaboration :** Very high customer involvement, with constant feedback and changes based on user needs.
- **Risk Management :** Strong emphasis on reducing technical risks through practices like test-driven-development (TDD) and pair programming.

4. Spiral Methodology:

- Overview: Spiral is a risk-driven model that combines iterative development with regular risk assessment. It involves continuous planning, prototyping and risk analysis throughout the project.

- Characteristics:

- Predictability: Offers moderate predictability as it breaks the project into smaller, manageable cycles, but requires ongoing assessments.
- Customer Collaboration: Regular feedback and iteration allow for high collaboration with the customer, ensuring their needs are met.

- Risk Management: Emphasizes proactive risk management by identifying and addressing risks at every phase of the cycle. This is especially valuable for high-risk-projects.

Question-3:

A company is working on two different projects. Project A has well-defined requirements and a strict deadline, while Project B has evolving requirements with an uncertain timeline and continuous customer feedback. Both projects involve high stakes, and the team must decide which development methodology to use.

- Compare and contract the Waterfall, Agile, Extreme and Spiral development models. Based on the characteristics of both projects (project-A and project B), which methodology would best suit each? Support your answer with a detailed analysis of how each methodology would address the specific needs of the projects, considering factors such as predictability, customer collaboration, and risk management.

Answer:

Project A (Well-defined requirements, strict deadline):

- Best Methodology: Waterfall
- Why: Since Project A has well-defined requirements and a strict deadline, Waterfall is the most suitable methodology. Its predictability and structured approach make it ideal for projects where the requirements are clear from the start and changes are minimal.
- How it addresses the project:
- Predictability: Waterfall's clear, linear progression ensures that the project stays on schedule with clear milestones.

- Customer Collaboration: Minimal, as requirements are set at the beginning.
- Risk Management: While Waterfall does not excel in handling changes once the project has started, its structured nature helps mitigate risks early in the planning phase. However, if unforeseen issues arise during implementation, it may be harder to adjust.

Project B (Evolving requirements, uncertain timeline, continuous feedback):

- Best Methodology: Agile or Spiral (with a slight preference for Agile)
- Why: Project B requires continuous customer feedback and has evolving requirements, making both Agile and Spiral suitable. However, Agile is slightly better suited because of its flexibility and adaptability.

- How it addresses the project:

- Predictability: Agile is less predictable, but this is acceptable given the uncertain timeline and evolving nature of the project. The focus is on delivering value iteratively with regular adjustments based on feedback.

- Customer Collaboration: Agile emphasizes close collaboration with the customer, allowing them to provide feedback at the end of each Sprint. This ensures that the product evolves based on real user needs and market changes.

- Risk Management: Risks are addressed continuously in Agile by identifying and resolving issues during each Sprint. This iterative process minimizes the risk of significant issues arising late in the project.

• Spiral Alternative: While Spiral also allows for continuous feedback and risk management, it may be more formal and documentation-heavy compared to Agile; making it less adaptable in fast-paced environments like Project B. However, Spiral's focus on risk assessment could be valuable if the project involves high technical or market risks.

Summary of Methodology Suitability:

• Project A: Waterfall is the most suitable methodology due to its predictability, structure, and ability to meet deadlines with well-defined requirements.

• Project B: Agile is the best fit for its flexibility, customer collaboration, and ability to adapt to evolving requirements. Spiral could also be an option, but Agile's speed and focus on iterative delivery make it more appropriate for a project with a high degree of uncertainty.

Question-4:

Explain the principles of software engineering ethics, highlighting the issues related to professional responsibility. Discuss how the ACM/IEEE Code of Ethics guides ethical decision-making in software engineering practices.

Answer:

Principles of Software Engineering Ethics:

Software engineering ethics emphasizes the responsibility of engineers to ensure their work prioritizes the safety, well-being and privacy of users, as well as the broader societal impact. Key principles include:

1. Public Welfare: Software engineers must prioritize the public's safety and interests,

ensuring their work does not cause harm or unethical consequences.

2. Professional Competence : Engineers should work within their areas of expertise, continuously improving their skills to deliver high-quality work.

3. Honesty and Integrity : Transparency, honesty, and integrity in communication, reporting progess and managing expectations are essential.

4. Confidentiality : Engineers must respect the confidentiality of client, employer and user information, ensuring that sensitive data is not misused.

Issues Related to Professional Responsibility:

- Accountability: Engineers must be accountable for their actions and decisions, ensuring the software is reliable and safe ~~fog~~ for users.

- Conflict of Interest: Engineers should avoid situations where personal or financial interests interfere with their professional duties.

- Privacy and Security: Protecting user data and ensuring compliance with privacy laws is a critical responsibility.

- Inclusivity: Ensuring software is accessible to all users, regardless of physical or cognitive limitations, is part of professional responsibility.

ACM/IEEE Code of Ethics and Ethical Decision-Making:

The ACM/IEEE Code of Ethics provides a framework for ethical decision-making by offering principles that guide software engineers in making responsible choices. It emphasizes:

1. Public Interest: Engineers are encouraged to act in the public's best interest, ensuring their work is safe, secure and ethically sound.

2. Competence and Quality: Engineers must produce high-quality, well-tested and reliable software, constantly improving their knowledge.

3. Honesty and Transparency: The code stresses the importance of being # truthful, reporting issues early, and avoiding misrepresentation of capabilities or risks.

4. Respect for Others : It encourages respect for colleagues, clients, users and society, emphasizing the need for fairness, privacy protection and avoiding harm.

5. Professionalism : The code supports ethical conduct throughout one's career, promoting lifelong learning and integrity in the face of challenges.

In summary, the ACM/IEEE Code of Ethics helps guide engineers by providing clear ethical principles that support responsible decision-making, ensuring their work is both effective and aligned with public interest.

Question - 5 :

Given the story of the Airport Reservation System, identify at least five functional and five non-functional requirements for the system. In your answer, explain how each requirement contributes to the overall performance, usability and security of the system. Consider factors such as performance user experience and system maintenance in your discussion.

Answer :

The Airport Reservation System (ARS) is designed to manage flight bookings, passenger reservations, and other related services in an airport or airline. Below are five functional and five non-functional requirements for such a system, explaining how each contributes to the system's performance, usability and security :

Functional Requirements:

1. User Registration and Login: Allows users to create and manage accounts, ensuring secure access to bookings and personal details.

2. Flight Search and Booking: Enables users to search for flights and book tickets based on various criteria, providing the core functionality of the system.

3. Payment Processing: Secures financial transactions for bookings, ensuring trust and security in payments.

4. Flight Reservation Management: Allows users to view, modify or cancel their bookings, enhancing flexibility and user control.

5. Notification System: Sends updates on bookings, cancellations and flight statuses, ensuring users are informed.

Non-functional Requirements:

1. Performance (Response Time): The system should process requests quickly (e.g. flight searches within 2 seconds) to ensure a smooth user experience, especially during peak times.

2. Scalability: The system must handle increased users and traffic without compromising performance as the airport or airline grows.

3. Availability: Ensures the system is accessible 24/7 with minimal downtime, offering reliability to users.

4. Security (Data Protection): Protects sensitive user data (e.g. payment and personal info) with encryption and secure authentication to maintain privacy and trust.

5. Maintainability: Allows for easy updates and bug fixes, ensuring long-term system reliability and adaptability.

Together, these requirements ensure the system is

secure, efficient and user friendly, providing a smooth and reliable experience for users while ensuring system longevity and security.

Question-6:

Illustrate and explain the V-model of testing phases in a plan-driven software process, detailing the relationships between development activities and corresponding testing activities.

Answer:

The V-Model is a sequential software development lifecycle model that emphasizes the synchronization of development and testing activities. It's named after the V-shape formed by the development and testing phases.

Key Relationships:

- Verification (Left side):
- Requirements Analysis: Defines user needs and system requirements.
- System Design: Creates high-level system architecture.
- Architectural Design: Establishes the system's structure and components.
- Module Design: Details the design of individual modules.
- Validation (Right side):
- Unit Testing: Tests individual modules for correctness
- Integration Testing: Ensures modules work together as a system.
- System Testing: Tests the entire system against requirements.
- Acceptance Testing: Verifies the system meets user needs.

How it Works:

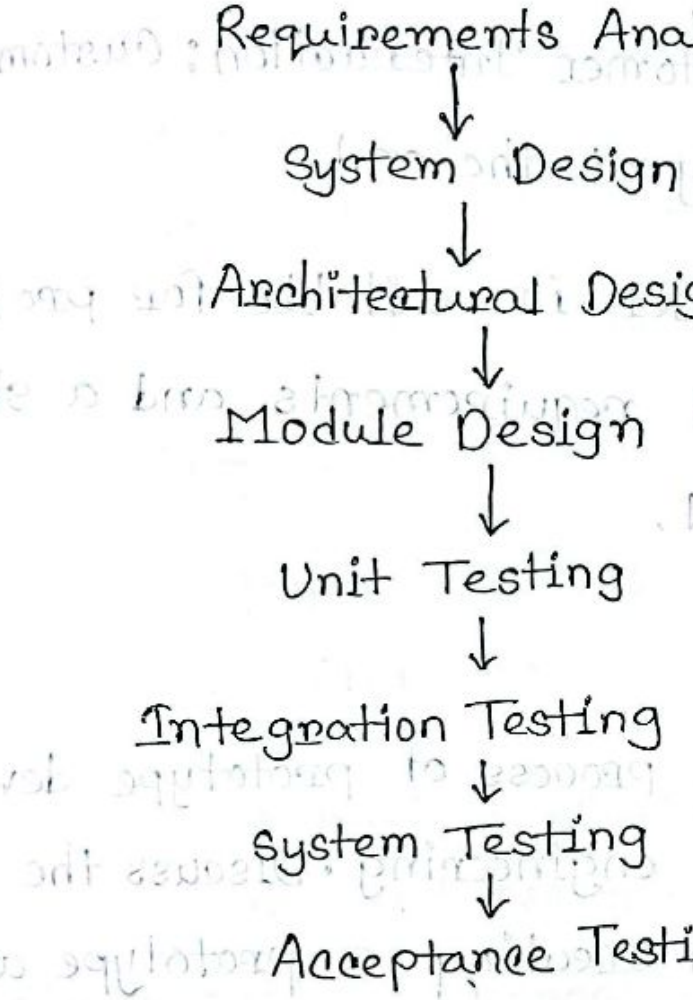
1. Development: Starts with requirements analysis and progresses through design phases.

2. Testing: Corresponding testing phases are Planned and executed in parallel with development.

3. Verification: Ensures the product is built correctly (meets specifications).

4. Validation: Ensures the product is built right (meets user needs).

Visual Representation:



Benefits:

- Early Defect Detection: Testing starts early, reducing rework costs.
- Improved Quality: Thorough testing at each stage leads to a more reliable product.
- Better Communication: Clearer communication between development and testing teams.

Limitations:

- Less Flexible: Difficult to accommodate changes in

requirements.

- Limited Customer Interaction: Customer feedback is primarily at the end.

The V-Model is suitable for projects with well-defined requirements and a stable environment.

Question-7

Explain the process of prototype development in software engineering. Discuss the key stages involved in creating a prototype and how it helps in refining software requirements. Analyze the benefits of using the prototyping model, particularly in terms of user feedback risk reduction and iterative development.

Answer:

Prototype Development is a software development model where a basic version of the system,

known as a prototype, is created to gather feedback and refine requirements. It's an iterative process that allows developers, stakeholders, and users to explore ideas, test functionalities and visualize the final product.

Key Stages of Prototype Development :

1. Requirements Gathering : Basic, high-level requirements are collected to build the prototype.

2. Prototype Development: A simple, working model is created with key features for initial testing.

3. User feedback : Users interact with the prototype and provide feedback on functionality and design.

4. Refinement : The prototype is revised based on feedback, adding features and improving functionality.

5. Final Product Development : Once refined, the prototype becomes the foundation for the final system.

Benefits of the Prototyping Model:

1. User Feedback: Users can interact with the prototype early, providing valuable input to improve the product.

2. Risk Reduction: Identifying issues early reduces the risk of building a product that doesn't meet user needs.

3. Iterative Development: The model allows for continuous refinement, aligning the product with user expectations.

4. Better Requirement Clarification: The prototype helps clarify and evolve requirements through user interaction.

In summary, prototyping helps clarify requirements, gather early feedback, and refine the system through interative development, reducing risks and improving user satisfaction.

Question-8:

Explain the process improvement cycle in software engineering and describe its key stages. Name and explain some commonly used process metrics, highlighting how they help in monitoring and improving software processes.

Answer:

The process improvement cycle focuses on continuously enhancing software development processes to improve quality, efficiency and performance. It typically follows a structured approach like PDCA (Plan-Do-Check-Act):

1. Plan: Identify areas for improvement and define goals or processes to enhance.
2. Do: Implement the changes or improvements in a controlled environment.
3. Check: Measure the impact of the changes using

process metrics.

4. Act : If the changes were successful, standardize them; if not, refine the approach and repeat.

Key Stages in Process Improvement :

1. Assesment : Evaluate the current process to identify inefficiencies or bottlenecks.

2. Analysis : Investigate the root causes of problems and determine where improvements are needed.

3. Implementation : Apply changes or improvements to the process.

4. Evaluation : Measure the results and impact of the changes.

5. Standardization : If successful, standardize the improvements and make them part of the regular process.

Commonly Used Process Metrics:

1. Defect Density: Measures the number of defects per unit of software (e.g. per 1000 lines of code). It helps monitor software quality and identify areas that need improvement.

2. Cycle Time: The time taken to complete a task, from start to finish. This helps measure efficiency and identifies bottlenecks in the process.

3. Velocity: In Agile development, velocity measures the amount of work completed in a sprint (e.g. story points). It helps team assess their capacity and plan future sprints effectively.

4. Test Coverage: The percentage of code covered by automated tests. High test coverage indicates better code quality and fewer defects.

5. Customer Satisfaction: Feedback from users about the software's performance and features. It helps ensure the final product meets user expectations.

How Metrics Help in Monitoring and Improving processes:

- Defect Density helps identify quality issues early.
- Cycle Time and Velocity help optimize process efficiency.
- Test Coverage ensures software reliability.
- Customer Satisfaction ensures the product meets user's needs and expectations.

In summary, the process improvement cycle is about continuously improving software processes by assessing, implementing and measuring changes. Process metrics help monitor progress and ensure improvements leads to better quality and efficiency.

Question-9:

Explain the Software Engineering Institute Capability Maturity Model (SEI CMM) and its five levels of capability and maturity. Analyze how each level contributes to improving the software development process and organizational performance.

Answer: The software Engineering Institude Capability Maturity Model (SEI CMM) is a framework used to assess and improve software development processes within an organization. It helps organizations evaluate their current process, identify areas for improvement, and provide a roadmap for achieving higher level of process maturity and capability.

[P.T.O]

Five Levels of the SEI CMM:

1. Level 1- Initial (Ad hoc):

At this level, processes are often chaotic and unorganized. Software development is typically reactive, with no standardized procedures or consistency.

The focus is on building the basic awareness of the need for improved processes. Organizations at this stage face a high level of risk and inefficiency due to lack of structure.

2. Level 2- Managed (Repeatable):

At this level, basic project management processes are established. The organization starts to repeat successful practices, focusing on ensuring projects are completed on time and within budget.

The focus shifts to improving consistency and predictability. Key practices like requirements management, scheduling, and tracking are implemented to manage projects more effictively.

3. Level 3 - Defined (Proactive):

Processes are well-defined and standardized across the organization. There is a focus on continuous process improvement, with practices for quality assurance, design and development being standardized.

This level leads to more proactive management of processes, with a focus on consistency across all projects. It reduces variability and enhances coordination within teams, leading to higher quality and more predictable outcomes.

4. Level 4 - Quantitatively Managed :

The organization begins using quantitative data and metrics to monitor and control processes. Performance is measured and statistical techniques are applied to predict future outcomes and improve process performance.

The focus here is on using data to make informed decisions, optimize processes, and improve quality. This level helps reduce defects and variances through data-driven improvements, leading to higher efficiency and performance.

5. Level 5 - Optimizing (Continuous Improvement):

The organization focuses on continuous Improvement by optimizing processes based on data and feedback. There is an

emphasis on innovation, technological improvements, and process optimization.

At this stage, organizations actively seek to innovate and refine their processes. It leads to high levels of efficiency, adaptability and maturity. Continuous improvement ensures that the organization remains competitive and capable of meeting new challenges.

How each level contributes to improving software Development and Organizational Performance:

- Level 1 (Initial): Provides a starting point for recognizing that processes need improvement. At this level, an organization may have chaotic or inconsistent processes, leading to unreliable software delivery.

[P.T.O]

- Level 2 (Managed) : Establishes basic project management practices, ensuring that projects are executed more predictably, with a focus on meeting deadlines and budgets.
- Level 3 (Defined) : Standardizes processes across the organization, reducing variability and making projects more efficient and consistent. This leads to better coordination and higher quality outcomes.
- Level 4 (Quantitatively Managed) : Introduces data-driven decisions, leading to more precise control over processes. The use of metrics helps in identifying areas of improvement and minimizing defects and rework, improving overall performance.

• Level 5 (Optimizing): Focuses on ongoing refinement and innovation, ensuring that the organization continuously improves and adapts to new challenges. This results in highly efficient, flexible and competitive software development processes.

Summary:

The SEI CMM helps organizations improve software development processes step by step, from chaotic and unpredictable practices (Level 1) to highly optimized and data-driven processes (Level 5). As organizations move through the levels, they achieve better consistency, efficiency, quality, and continuous improvement, leading to enhanced software performance and organizational success.

Question-10:

Describe the core principles of agile software development methods. Analyze how these principles are applied in different software development environments, and assess the benefits and challenges of using agile methods in various project types and organizational settings.

Answer:

Core Principles of Agile Software Development:

1. Customer Collaboration Over Contract Negotiation:

Agile emphasizes working closely with customers throughout the development process to ensure the product meets their needs.

2. Responding to Change Over Following a Plan: Agile values flexibility, allowing the project to adapt to changes in requirements or market conditions.

3. Delivering Working Software Frequently: Agile encourages frequent delivery of small, functional increments of the product, often every 1-4 weeks.

4. Individuals and Interactions Over Processes and Tools: Agile prioritizes communication and collaboration within the team rather than relying on rigid processes and tools.

5. Simplicity and focus: Agile promotes simplicity in design and functionality, focusing only on what's necessary for the project.

6. Self-Organizing Teams: Agile relies on teams to organize and manage themselves, empowering team members to make decisions and collaborate effectively

How Agile Principles Apply in Different Environments :

- Software Development : Agile works well in environments with rapidly changing requirements, where constant feedback and flexibility are needed. Frequent releases help manage customer expectations and adjust quickly.
- Startups : Agile is ideal for startups as it enables rapid iteration, quick feedback from early users and quick pivots if needed.
- Large Enterprises : Agile can be harder to scale, but using frameworks like Scrum or SAfe can help large teams work in smaller, more manageable chunks.

• Distributed Teams: Agile can be applied in remote settings with tools like Slack, Jira and zoom to ensure communication and collaboration are maintained.

Benefits of Agile:

1. Flexibility: Agile allows teams to adapt to changing requirements and feedback quickly.

2. Faster Delivery: Regular, incremental releases keep the project moving forward and allow early testing.

3. Customer Satisfaction: Continuous collaboration with the customer leads to better alignment with their needs.

4. Improved Quality: Frequent testing and feedback help catch defects early.

Challenges of Agile:

1. Scope Creep: Frequent changes and lack of a rigid plan can lead to scope creep, where the project grows beyond initial expectations.

2. Requires Experienced Teams: Agile relies on self-organizing teams, which may be challenging if the team lacks experience or maturity.

3. Overhead in Coordination: Agile requires regular meetings (like daily stand-ups), which can be time-consuming if not managed well.

4. Not Ideal for All Projects: For large, highly structured projects (e.g., hardware development or projects with strict

regulatory requirements), Agile might not be the best fit.

Conclusion:

Agile methods bring flexibility, faster delivery and customer satisfaction, making them great for projects with evolving requirements or those that need quick iterations. However, they can be challenging in environments with rigid structures or where team experience and coordination are lacking.

Question-11:

Draw the release cycle of Extreme Programming (XP) and explain the influential programming practices.

Answer:

The Extreme Programming (XP) release cycle focuses on frequent, small releases, ensuring that the software is always in a deployable state and customer feedback is continuously integrated.

Here's how the release cycle typically works:

Extreme Programming (XP) Release Cycle:

1. Planning: Initial and iteration planning define features and deliverables.

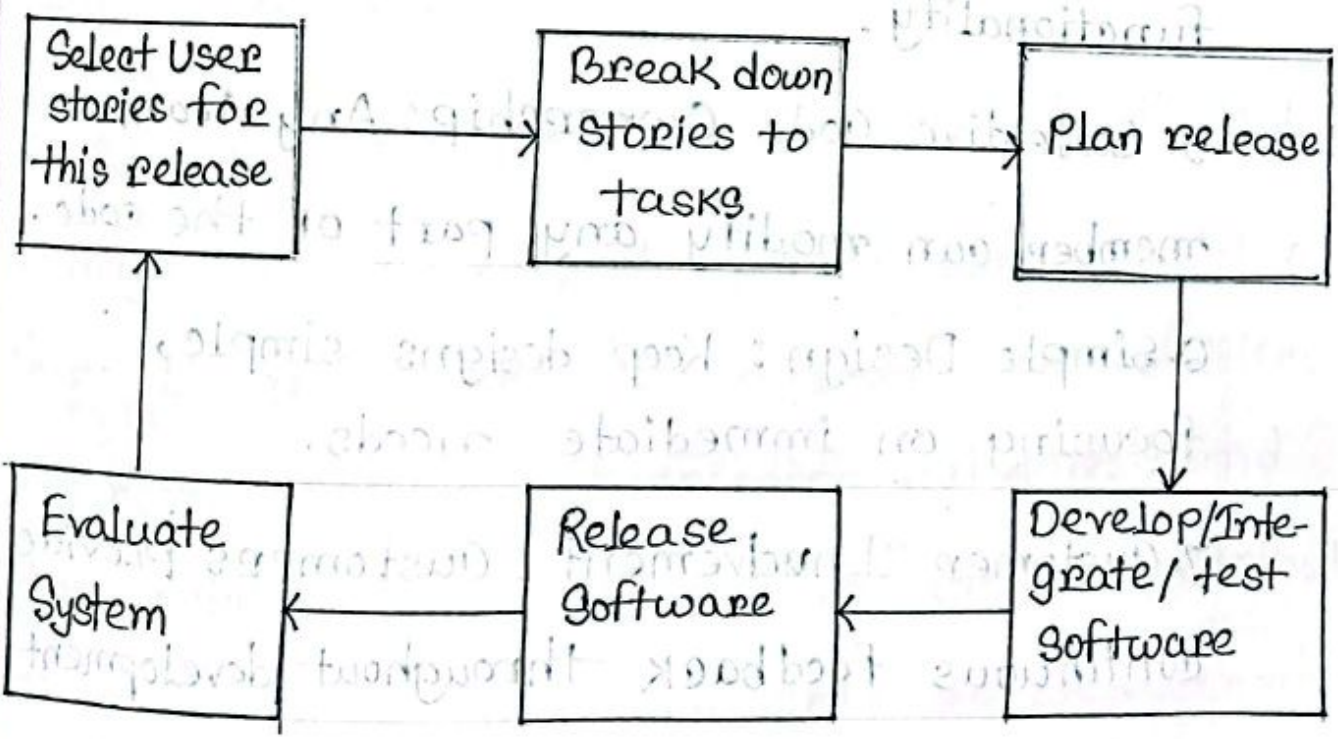
2. Development Iteration: Code is developed in short iterations (1-2 weeks).

3. Continuous Testing: Automated tests ensure functionality and quality.

4. Customer Feedback: Customer reviews each iteration and provides input.

5. Release: A small, working release is delivered after each iteration.

6. Repeat: The process repeats in cycles, refining the software with each iteration.



Influential XP Programming Practices:

1. Pair Programming: Two developers collaborate on the same code to improve quality.

2. Test-Driven Development (TDD): Write tests before code to ensure correctness.

3. Continuous Integration (CI): Integrate code frequently with automated testing.

4. Refactoring: Continuously improve the code structure without changing functionality.

5. Collective Code Ownership: Any team member can modify any part of the code.

6. Simple Design: Keep designs simple, focusing on immediate needs.

7. Customer Involvement: Customers provide continuous feedback throughout development.

Question-12:

A local library wants to create a digital system to manage its operations. The system will track books, members and borrowing activities. Each book has attributes like title, author, ISBN and genre. Members have attributes such as name, membership ID and contact details. When a member borrows a book, the system records the borrowing date, return due date and return status. The library also want to maintain a catalog of overdue books and their respective fines.

Using the seenario of a digital library management system, design an Entity-Relationship Diagram (ERD) to represent the entities (e.g. books, members, borrowing activities) and their relationships. Clearly explain the attributes of each entity and how they are interconnected.

Question-12:

Answer:

To design an Entity-Relationship Diagram(ERD) for the digital library management system, we'll break down the entities, their attributes, and the relationships between them based on the requirements provided. Here's how it can be structured:

1. Entities and Attributes:

Book

Attributes:

- BookID (Primary Key): Unique identifier for each book.
- Title: Title of the book.
- Author: Author of the book.
- ISBN: International Standard Book Number
- Genre: Genre of the book.

Member

- Attributes
- Member ID (Primary Key): Unique identifier for each member.
- Name : Name of the member
- Contact Details : Contact information (phone number, email, etc.)
- Address : Physical address of the member.

Borrowing Activity

- Attributes:
- BorrowingID (Primary Key) : Unique identifier for each borrowing activity.
- BookID (Foreign Key) : Refers to the borrowed book (links to the Book entity).
- MembemID (Foreign Key) : Refers to the member who borrowed the book (links to the Member entity).
- Borrowing Date : The date the book was borrowed.
- Return Due Date : The date the book is due for return.

- ReturnStatus: Whether the book has been returned or not (could be "Returned" or "Not Returned").
- Fine: The fine imposed if the book is returned late (can be NULL if returned on time).

Overdue Book

- Attributes:
- OverdueID (Primary Key): Unique identifier for each overdue book entry.
- BorrowingID (Foreign Key): Refers to the borrowing activity of the overude book (links to the BorrowingActivity entity).
- FineAmount: Fine imposed on the overdue book.
- OverdueDate: Date when the book became Overdue.

2. Relationships:

Book - BorrowingActivity (One-to-Many)

- One book can be borrowed many times, but each borrowing activity refers to a specific book.
- Relationship: A book can appear in many borrowing activities, but each borrowing activity involves only one book.

Member - BorrowingActivity (One-to-Many)

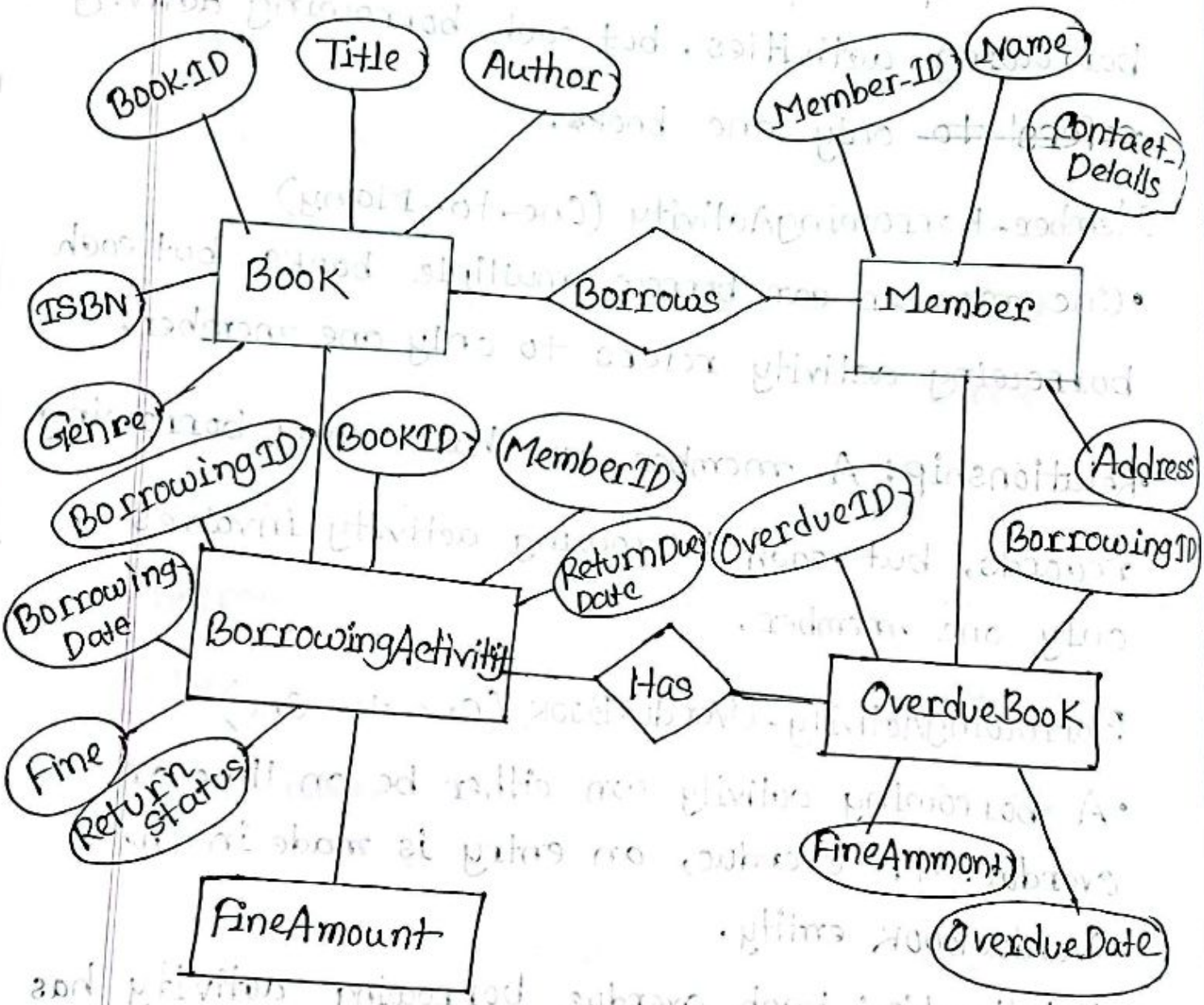
- One member can borrow multiple books, but each borrowing activity refers to only one member.
- Relationship: A member can have many borrowing records, but each borrowing activity involves only one member.

BorrowingActivity - OverdueBook (One-to-One)

- A borrowing activity can either be on time or overdue. If overdue, an entry is made in the OverdueBook entity.
- Relationship: Each overdue borrowing activity has

a corresponding overdue book record. If a borrowing activity is not overdue, no record is created in the OverdueBook entity.

3. ERD Diagram:



4. Explanation of the Relationships :

1. Book and BorrowingActivity (One-to-Many):

• A book can be borrowed multiple times by different members. This relationship helps keep track of the borrowing history of each book.

2. Member and BorrowingActivity (One-to-Many):

• A member can borrow multiple books, and the system will record the borrowing dates, return due dates and whether the books have been returned or not.

3. BorrowingActivity and OverdueBook (One-to-One):

• If a borrowed book is overdue, the system will create a corresponding record in the OverdueBook entity. The fine and Overdue date will be tracked here.

5. Additional Notes :

• The BorrowingActivity entity tracks the status of

the borrowed book and the fine (if any), while the OverdueBook entity specifically tracks the overdue books and their fines.

- If a book is not overdue, there is no need for an entry in the OverdueBook entity.

Summary:

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

Question-13:

What is called Testing? Differentiate between validation and verification.

Answer: In software Engineering, Testing refers to the process of evaluating and verifying that a software application or system works as intended. It involves executing the software to find defects, ensure it meets specified requirements and confirm its overall functionality, performance and security. The goal is to identify bugs, ensure the software's quality, and verify that it performs as expected under various conditions.

Testing is a crucial part of the Software Development Life Cycle (SDLC), and it typically occurs after the software has been developed and it before it is released to users.

Difference between Verification and Validation:

Verification	Validation
Verification refers to the set of activities that ensure software correctly implements the specific function.	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements
It includes checking documents, designs, codes, and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is dynamic testing.
It does not include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are BlackBox Testing, White Box Testing and non-functional testing
It checks whether the software conforms to specifications or not	It checks whether the software meets the requirements and expectations of a customer or not

It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validations	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.
After a valid and complete specification the verification starts.	Validation begins as soon as project starts.
Verification is for prevention of errors.	Validation is for detection of errors.
Verification finds about 50-60% of the defects.	Validation finds about 20-30% of the defects.

Question-14

Design a layered architecture model ofor an online judge system, identifying the key layers (e.g. presentation, application, bussiness logic and data). Explain the responsibilities of each layer and analyze how this architecture ensures scalability, maintainability and efficient performance.

Answer:

Layered Architecture for an Online Judge System

An online judge system (e.g. for coding competitions or evaluations) can be designed using a Layered Architecture. This approach separates the system into distinct layers, each with its own responsibilities, ensuring that the system is scalable, maintainable and efficient.

Here's a typical 4-layer architecture for an Online Judge System:

1. Presentation Layer (User Interface Layer):

Responsibilities:

- User Interaction: Handles interactions with the users (e.g., coding contestants, administrators)
- Display Information: Shows coding problems, test case results, feedback (e.g., passed, failed), and submission history.
- Input Validation: Collects code submissions, validates user input for correctness, and handles user authentication.

Key Components:

- Web Interface (e.g., using HTML, CSS, JavaScript for frontend)
- APIs for interactions with the other layers.

Scalability & Maintainability :

- Scalability : The user interface can be independently scaled with load balancing (e.g., serving static content, using CDN)
- Maintainability : frontend code (presentation) is separated from backend logic, so UI & updates can be made without affecting business logic.

2. Application Layer (Service Layer)

Responsibilities :

- Request Handling : Manages user requests and coordinates the flow of data between the presentation and business logic layers.
- Service Invocation : It invokes the business logic layer to process user requests like submitting code, checking results etc.

- Transaction Management: Manages transactional integrity (e.g., saving submissions, tracking results).

Key Components:

- APIs for each feature (e.g., code submission, evaluation)
- controllers/services that orchestrate actions and handle logic flow.

Scalability and Maintainability:

- Scalability: The application layer can be horizontally scaled to handle more user requests.
- Maintainability: Changes in the flow of data or request handling can be isolated to the service layer, ensuring minimal disruption.

~~3. Business Logic Layer (Domain Layer)~~

2. Business Logic Layer (Domain Layer)

Responsibilities:

- Core Logic: Implements the core functionalities such as compiling code, running test cases, evaluating solutions, and generating feedback.
- Execution of Algorithms: Contains the business rules for problem solving, such as time complexity evaluations, memory usage constraints, and algorithm-based problem-solving (e.g., matching a solution to the problem description).
- Result Computation: Determines the correctness of the submitted code based on predefined test cases and generates feedback for the user.

Key Components:

- Code Executor (compiling, running code)
- Judge (evaluates if a solution is correct based on output, time, and memory usage)
- Problem Solver (manages problem definition and constraints)

Scalability & Maintainability:

- Scalability: The business logic can be isolated into microservices, making it possible to scale parts of the logic (e.g. code execution service).
- Maintainability: Business rules and domain logic are contained in this layer, making it easy to update or add new evaluation criteria or features.

P.T-0.

4. Data Layer (Persistence Layer)

Responsibilities:

- Data Storage: Handles the persistence of user submissions, test case results, user profiles, problem definitions and historical data.
- Database Management: Manages interactions with the database (e.g., storing user submissions, managing coding problems, keeping track of results).
- Caching: Stores frequently accessed data (like problem definitions, results) for quick retrieval.

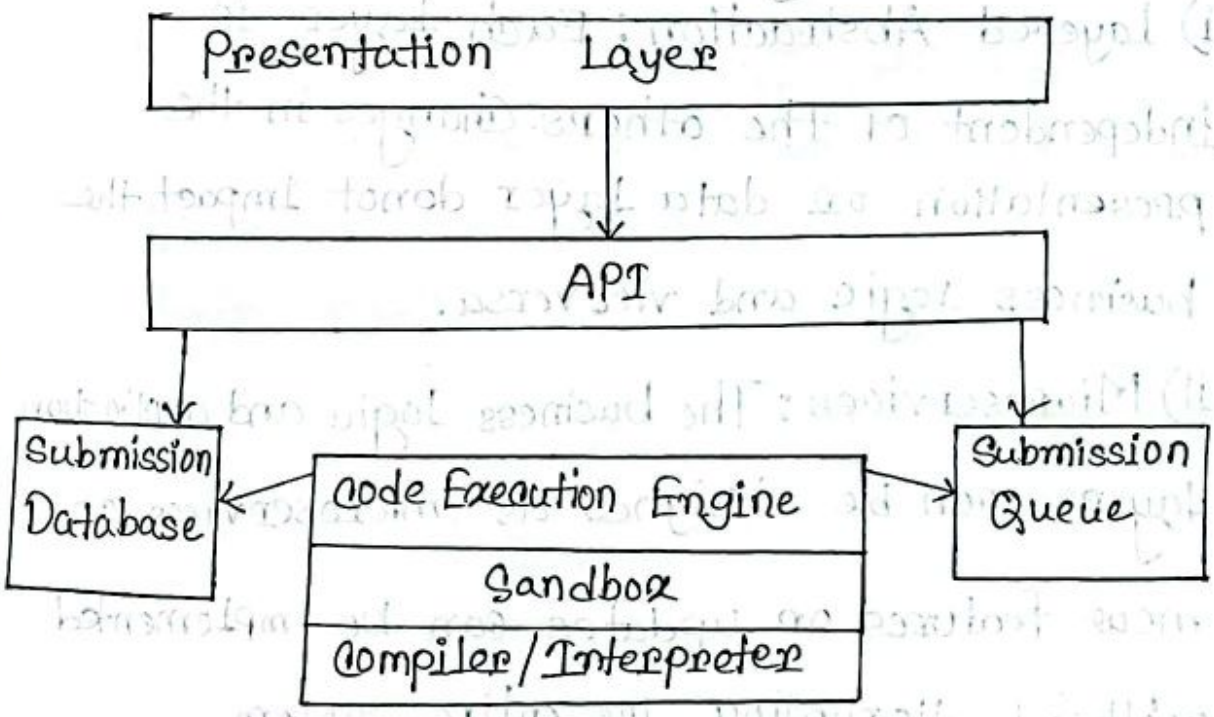
Key Components:

- Database (e.g., relational or NoSQL database for storing submission data, user profiles).
- File Storage for handling uploaded files (e.g., test case inputs, outputs)
- Cache (e.g., Redis or Memcached) for faster access to frequently used data.

Scalability & Maintainability:

- Scalability: The database layer can be vertically or horizontally scaled. You can shared data or use distributed databases for high availability.
- Maintainability: The data layer abstracts all database interaction, making it easier to modify or change the database structure without affecting other layers.

Online Judge Architecture



How this Architecture Ensures Scalability, Maintainability and Efficiency:

1. Scalability:

i) Horizontal Scaling: Layers can be independently scaled.

ii) Database Sharding: Data can be split across different databases to handle large amounts of data, allowing the system to scale as the number of users grows.

2. Maintainability:

i) Layered Abstraction: Each layer is independent of the others. Changes in the presentation or data layer donot impact the business logic and viceversa.

ii) Microservices: The business logic and application layers can be designed as microservices, so new features or updates can be implemented without disrupting the entire system.

3. Efficient Performance :

i) Caching : Frequently accessed data can be cached at the application layer (e.g. using Redis) to improve performance.

ii) Load Balancing : The presentation and application layers can use load balancing to distribute incoming requests, preventing any single server from becoming overwhelmed.

iii) Parallel Processing : The business logic layer can distribute code evaluation tasks across multiple servers or containers, enabling faster processing of submission.

Question: 15

Draw a DFD (Level-0 and Level-1) and UML Use case diagram for a Hospital Management system. A hospital management system is a large system that includes several subsystems or modules that provide various functions. Your UML use case diagram example should should show actors and use cases for a hospitals reception.

- Purpose: Describe major services (functionality) provided by a hospitals reception.
- Consider the scenario below:

The Hospital Reception subsystem or module supports some of the many job duties of a hospital receptionist. The receptionist schedules patient's arrival and/or by phone.

Answer:

Data flow diagram Hospital Management System is used to create an overview of Hospital management without going in too much detail.

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.

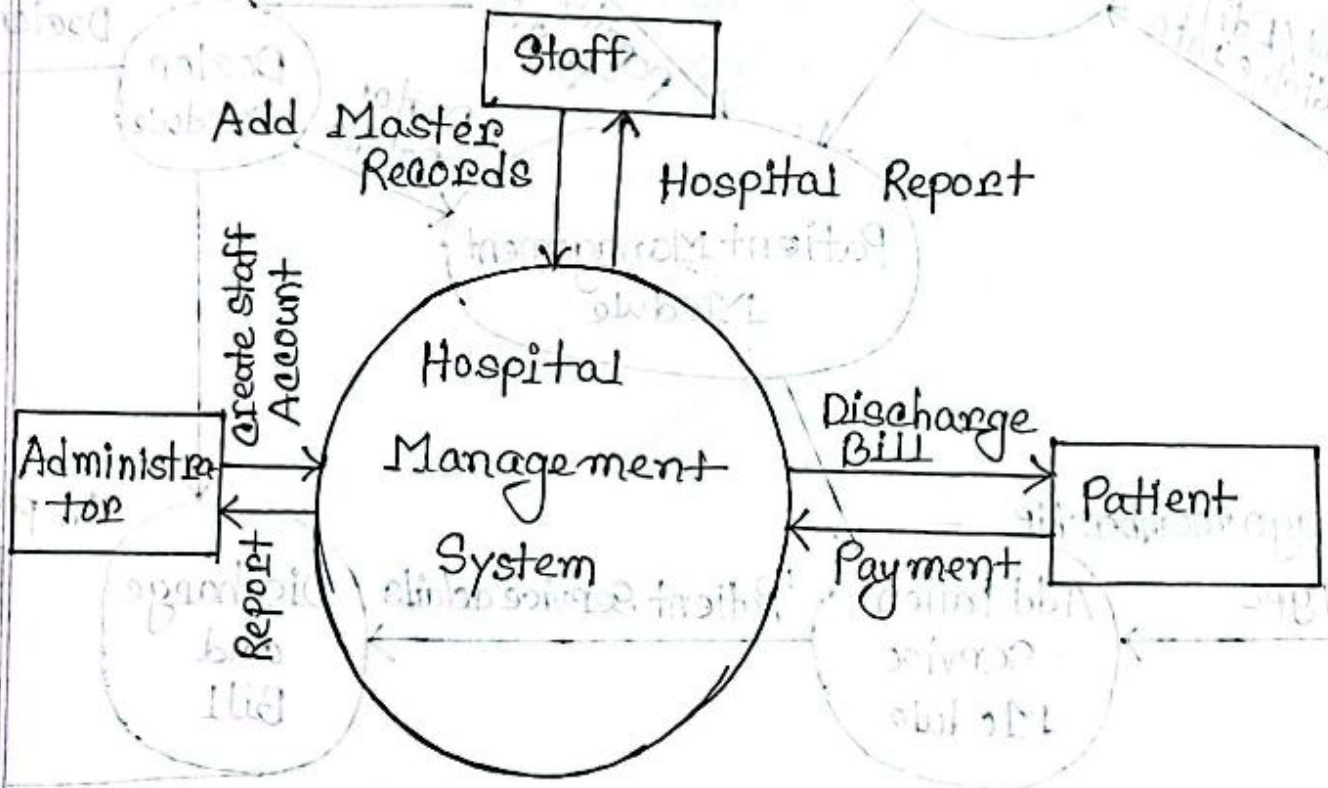
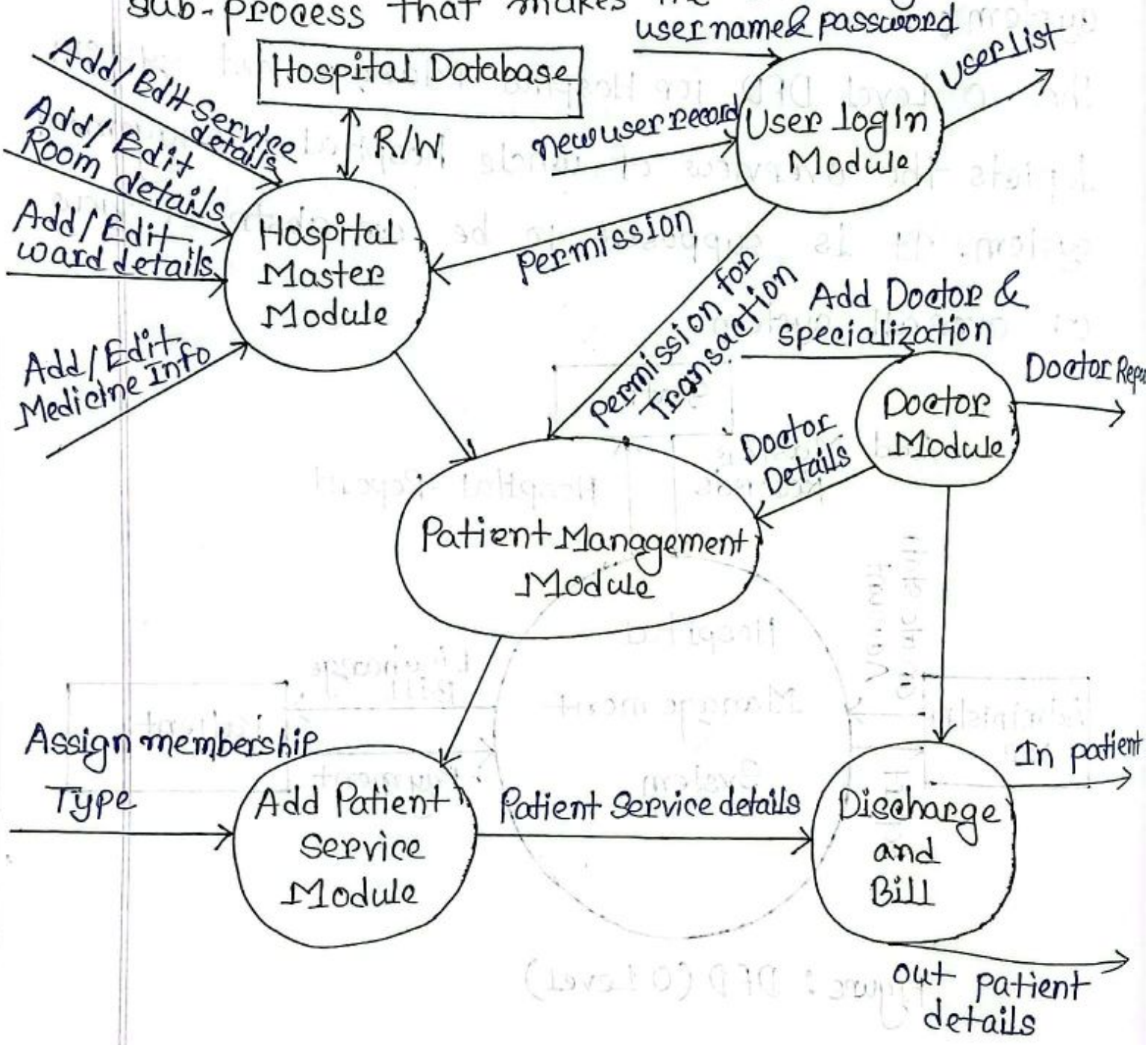


Figure: DFD (0 Level)

First Level Data Flow Diagram (Level-1 DFD) of Hospital Management System:

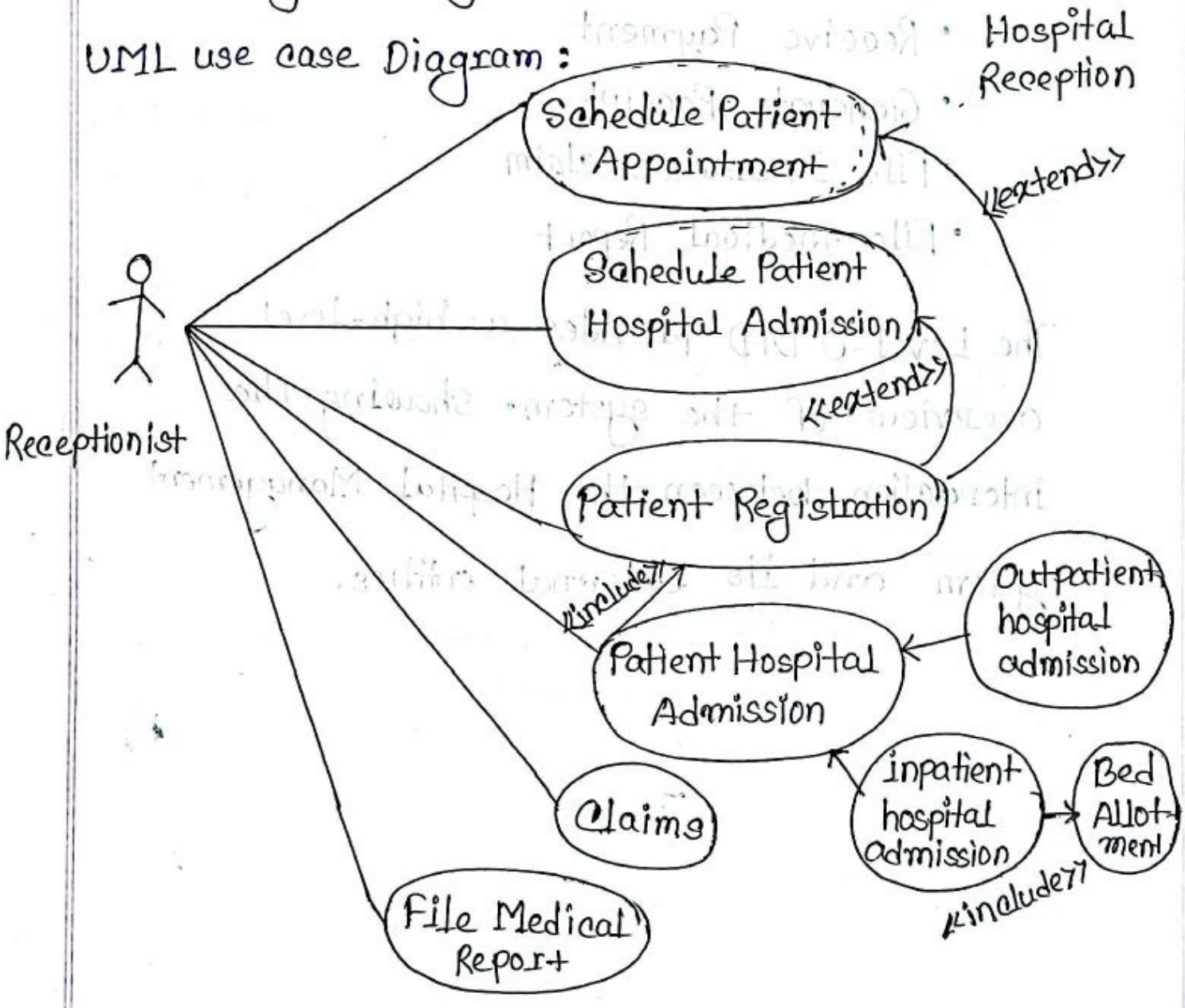
The first level DFD (1st level) of Hospital Management System shows more details of processing Level-1 DFD list all the major sub-process that makes the entire system.



The important process to be carried out are:

1. User & login
2. Hospital master
3. Patient Registration
4. Doctor Module
5. Add patient service
6. Discharge billing

UML use case Diagram:



In this diagram:

- Actor: Receptionist
- Use cases:
  - Schedule Appointment
  - Admit Patient
  - Collect Patient Information
  - Allocated Bed
  - Receive Payment
  - Generate Receipt.
  - File insurance claim
  - File medical Report

The Level-0 DFD provides a high-level overview of the system, showing the interaction between the Hospital Management System and its external entities.

Answer to the Question no-16

Based on the UML sequence Diagram given to the question, we can design a UML class Diagram to represent the relationships between the key classes involved in this system.

Identified classes from the Sequence Diagram:

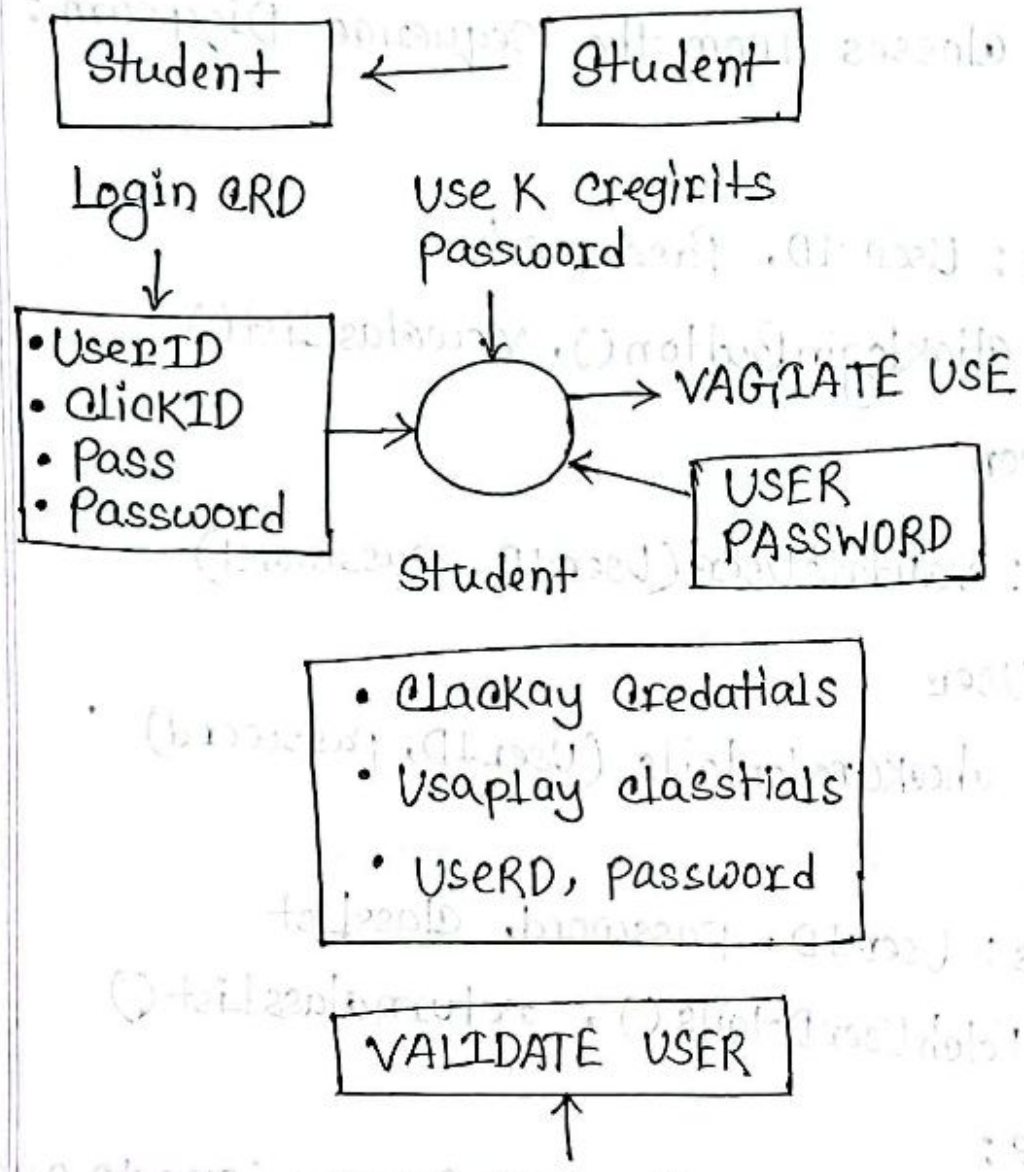
1. Student
   - Attributes: User ID, Password
   - Methods: clickLoginButton(), viewClassList()
2. LoginScreen
   - Methods: validateUser(UserID, password)
3. Validate User
   - Methods: checkCredentails(UserID, password)
4. Database
   - Attributes: UserID, password, classList
   - Methods: fetchUserDetails(), returnclassList()

Relationships:

- Student interacts with loginScreen (one to one)
- LoginScreen communicates with validateUser (one to one)

- ValidateUser fetches data from Database (one-to-one).
- Database stores user details and class lists

Now, let me generate the UML class Diagram based on this structure:



Here, is the UML class Diagram for the

Student login system based on the provided sequence diagram. It represents key classes such as Student, LoginScreen, ValidateUser, and Database, along with their attributes, methods, and relationships.

Question-17:

We know that Quality Assurance (QA) is not just Quality Controll (QC). For example, QA is process-oriented, and QC is product-oriented. Now, suppose you are said to get a straight explanatory description of it along with the Differences and Impediments between and to QA and QC.

Answer:

Quality Assurance (QA)

QA is a process-oriented approach. It focuses on ensuring the processes used in software development and testing are effective and efficient to produce high-quality software. QA involves systematic activities throughout the software development lifecycle to ensure quality is built into the product at every stage.

[P.T.O]

Some key points about QA:

- QA involves process design, audits, documentation and reviews.
- It focuses on preventing defects through proactive process improvements.
- QA aims to define, maintain and improve the processes used in development to ensure consistency and quality in the final product.
- It is ongoing and usually occurs throughout the project lifecycle.

Quality Control (QC)

QC, on the other hand, is product-oriented. It focuses on detecting and fixing defects in the final product through testing and inspections. QC is about checking whether the product meets the specified requirements and standards.

Some Key points about QC:

- QC involves testing the software product and inspecting the outputs to ensure they meet quality standards.
- QC is about identifying and correcting defects in the product.
- It is typically reactive, focusing on catching defects after they've occurred.
- QC includes activities like unit testing, integration testing, and system testing.

Differences Between QA and QC

Quality Assurance (QA)	Quality Control (QC)
Process-oriented (focus on how the product is developed).	Product-oriented (focus on the product itself)
Prevent defects from happening.	Detect defects in the final product.
Proactive (establishing good practices and procedures).	Reactive (testing to find issues after they've been created).

Process audits, process management, reviews, training	Testing, inspections, validation.
Continuous (throughout the project lifecycle)	After product development, before release.
Covers the entire development lifecycle	Focuses on the final product.
Preventative measures	Corrective actions (finding and fixing defects)
Process improvement tools, frameworks	Testing tools, defect tracking tools

Impediments to QA and QC

Impediments to Quality Assurance (QA)

1. Lack of Management Commitment: Without strong leadership support, QA processes may not be implemented effectively.

2. Inadequate Training: If the team isn't trained in quality management processes;

they may not follow best practices, leading to poor quality.

3. Resistance to Change: Team members or stakeholders might resist adopting new processes or methodologies for quality improvement.

4. Inconsistent Process Application: If QA processes aren't consistently applied across the organization or project, it leads to gaps in quality.

5. Time and Resource Constraints: Quality initiatives often require time and resources, and if these aren't allocated properly, the process might be rushed or overlooked.

Impediments to Quality Control (QC)

1. Late Testing: If testing is delayed until the end of the development cycle, it becomes harder to fix defects in time and within budget.

2. Limited Test Coverage: Inadequate test cases or incomplete testing can miss crucial defects, leading to poor product quality.

3. Lack of Test Automation: Without automated tests, it can be difficult to manage repetitive tasks, especially in large-scale projects, leading to inefficiency and missed defects.

4. Insufficient Resources or Tools: Limited access to appropriate testing tools or resources can hinder effective QC, leading to undetected defects.

5. Tight Deadlines: When deadlines are too aggressive, it may lead to rushed testing, leaving defects undetected or unaddressed.

In short, QA is about preventing defects by improving the development processes, while QC is about detecting defects in the final product. Both are crucial for delivering high-quality software, but they focus on different aspects and stages of the development process.

Question-18

Do you think the goal of QA is just to find the bugs as early as possible? The goal of the QA is to find the bugs as early as possible and make sure they get fixed. Quality Assurance was introduced after World War II when the weapons were tested before they came into action. Explain the role of Quality Assurance (QA) at each phase of the Software Development Life Cycle (SDLC).

Answer:

You're right that QA's goal isn't just to find bugs early, but also to ensure that quality is built into the entire development process. QA focuses on improving the processes to prevent defects, not just catching them. It involves more than just finding bugs - it's about creating the conditions where bugs are less likely to occur in the first place.

[P.T.O]

Here's a short explanation of QA's role at each Phase of the Software Development Life Cycle (SDLC):

1. Requirement Gathering: QA ensures that requirements are clear, complete, and testable, so they can be validated against the final product.

2. Design Phase: QA reviews the system design to ensure it meets the functional and non-functional requirements and is testable.

3. Development Phase: QA collaborates with developers to establish testing frameworks and ensures code quality practices are followed. They also prepare test cases and test data.

4. Testing Phase: QA conducts functional, integration, system, and regression testing to identify defects and ensure the product meets the requirements.

5. Deployment Phase: QA performs pre-release checks to ensure the product is ready for deployment and works in the production environment.

6. Maintenance Phase: QA monitors the product post-release, performing bug fixes, updates, and ensuring continued quality through regular testing.

In short, QA is involved in ensuring quality at every stage, from planning through to maintenance, not just in finding and fixing bugs, but in preventing them by improving processes and practices.

Question-19

Explain the Rapid Application Development (RAD) model in Software Engineering. Discuss its key Phases, Principles and Advantages. Analyze how the RAD model supports faster delivery of software solutions while maintaining quality and user satisfaction.

Answer:

Rapid Application Development (RAD):

Rapid Application Development (RAD) is a software development methodology that prioritizes speed and flexibility. It aims to produce high-quality software in a shorter time by emphasizing rapid prototyping, user feedback and iterative development. RAD is particularly useful in projects where the requirements are expected to change or evolve quickly.

Key Phases of the RAD Model

1) Requirements Planning Phase

2) User Design Phase

3) Construction Phase

4) Cutover Phase (Deplgyment Phase)

1. Requirements Planning Phase:

In this phase, the project team, including developers, stakeholders, and users, gather and define the high-level requirements for the application. Unlike traditional methods, the requirements are gathered more quickly and flexibly, without deep documentation.

Key Activities:

- Define project scope
- Identify key requirements
- Set priorities and timelines

2. User Design Phase:

This is a highly interactive Phase where users and developers collaborate to design the system. Prototypes are built to reflect the requirements, and users can provide feedback continuously.

Key Activities:

- Develop prototypes
- Gather user feedback
- Refine the design based on feedback.

3. Construction Phase:

Developers create the actual system using the iterative feedback from the user design phase. The system is constructed in small, functional modules rather than as a single monolithic build.

Key Activities:

- Build functional prototypes
- Iteratively improve the system based on user feedback.
- Perform continuous testing of the components.

4. Cutover Phase (Deployment Phase):

The final product is delivered and deployed. Since many iterations and testing have already occurred in the previous phases, this phase is focused on the final adjustments, documentation and actual deployment.

Key Activities:

- Final testing and bug fixes.
- Deployment of the system.
- Training and user support.

Key Principles of RAD

1. Prototyping : RAD relies heavily on building prototypes that simulate the user interface and functionality of the system. This allows for quick validation of design concepts and gathering of user feedback.

2. User Involvement : Users are deeply involved in every stage of development, especially in providing feedback on prototypes and design iterations. This ensures the product is closely aligned with their needs.

3. Iterative Development: The RAD model supports iterative cycles of development, where modules or features are developed and refined over time based on user feedback.

4. Time-boxing: The development process is broken into short, time-boxed iterations (often a few weeks). This ensures continuous progress and a quick turnaround.

5. Component Reusability: RAD emphasizes using pre-built components and tools to speed up the development process, allowing for more rapid assembly of the application.

Advantages of RAD:

1. Faster Development Time: By focusing on prototyping and iterative development, RAD drastically reduces the time required to develop and deploy software. Shorter development cycles and continuous feedback loops ensure that the project progresses quickly.

2. Improved User Involvement: Since users are actively involved throughout the process, the final product is more likely to meet their expectations

and requirements.

3. Flexibility and Adaptability: RAD allows for easy adaptation to changing requirements. As the software is developed iteratively, it can incorporate new features or adjust to evolving needs without major disruptions.

4. High-Quality Product: Since feedback is gathered continuously, issues can be identified and corrected early in the process, leading to higher-quality software.

5. Reduced Risk of failure: The use of prototypes and iterative development helps uncover potential issues early, reducing the likelihood of major project failures or misaligned expectations.

How RAD Supports faster Delivery While Maintaining Quality

- **Prototyping :** By building functional prototypes early in the process, RAD allows users to interact with the system and provide immediate feedback, reducing the changes of misunderstandings or missed requirements.
- **Iterative feedback:** The continuous feedback loop from users means developers can address issues as they arise, leading to quicker corrections and refinements.
- **Parallel Development:** RAD often employs parallel development, where multiple teams work on different parts of the system simultaneously, speeding up overall delivery.
- **Reusable Components:** Using pre-built or reusable components accelerates development and reduces the need for custom code from scratch. This leads to faster delivery and also allows for better consistency and quality across modules.

• Time-boxed Phases: By working within strict time frames (time-boxing), the RAD model ensures that each phase is completed promptly, avoiding delays that can occur in traditional waterfall models.

Challenges of RAD

While RAD offers several benefits, it may not be suitable for all types of projects. Some challenges include:

1. Complexity in Large Projects: RAD can be difficult to manage and scale for larger or more complex projects that require intricate systems or detailed design work.

2. Dependency on User Availability: RAD relies heavily on user involvement. If users are unavailable or unclear about requirements,

the process can break down.

3. Limited Documentation: Because RAD focuses on prototypes and iterations rather than detailed documentation, it can be harder to maintain comprehensive records for future updates or audits.

Conclusion

The Rapid Application Development (RAD) model offers a flexible, adaptive approach to software development that emphasizes speed, user involvement, and iterative feedback. By focusing on prototyping, continuous improvement, and reuse of components, RAD enables faster delivery while maintaining quality and aligning closely with user needs.

In projects with dynamic requirements or tight timelines, RAD can deliver highly functional and user-friendly software quickly, while ensuring that quality is ~~conte~~ consistently maintained

through frequent testing and refinements.

Question-20

White box testing consists of code coverage and a data coverage method. Consider the following decision (e.g. if, switch, while etc) and make one test for each side of each decision using a table with column caption Decision, X input and Y input. Then implement Junit test class that tests each decision described in the table using Java.

Answer to the Question no-20

White box testing ensures that all possible branches of a program are tested by covering decision points such as if, else and loops. Below, we create a test cases table for the given Java code and then implement a JUnit test class in Java.

Test cases Table

Decision Statement	X Input	Y Output	Expected Output
If (y==0)	5	0	"y is zero"
else if (x==0)	0	3	"x is zero"
For loop does not run	0	2	"" No output
For loop prints numbers divisible by y	4	2	"2", "4"
For loop prints numbers divisible by y	4	3	"3"
For loop with negative y	5	-2	"2", "4"
Negative x, loop doesnot execute	-3	2	"," (No output)

Junit Test Class in Java

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

Class DecisionTest{
  private List<String> Output=new ArrayList<>();
  private void println(String message){
   output.add(message);
  }
 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(String.valueOf(i));
    }
  }
 }
}
@Test
 void testYisZero(){
      output.clear();
      process(5,0);
     assertEquals(List of("y is zero"), output);
   }
@Test
void testLoopDoesNotRun(){
 output.clear();
  process(0,2);
 assertTrue(output.isEmpty());
}
@Test
void testEdgeCaseXNegative(){
     output.clear();
    process(-3,2);
   assertTrue(output.isEmpty());
  }
}
```

Explanation of the JUnit Test class:

i. Simulating

ii. Implementing the original logic

iii. Junit Test Methods

In conclusion, This Junit test class effectively tests all possible branches in the given Java program. White box testing principles are applied to ensure complete code coverage and data coverage. This implementation is structured similarly to the python version, ensuring easy verification of results.

Answer to the Question no-21

Junit 4 Black Box Unit Testing Approach:

Black Box Unit Testing detects early by testing functions independently without knowing internal logic.

To test exception handling, setup function, and timeout rule, we develop JUnit 4 test cases for the following production code.

Production Code (Calculator.java)

```
public class Calculator {
    private int memory;
    public Calculator () {this.memory=0;}
    public int divide (int a, int b)
    {
     if (b==0) throw new ArithmeticException("cannot
                                    divide by zero");

     return a/b;
    }
```

```
public double sqrt (double number){
if (number<0) throw new IllegalArgumentException
                    ("Negative number.");
try { thread.sleep(500); } catch (InterruptedException
                                        e) {}
}
public void storeInMemory (int value){
    this.memory = value;
 }
 public int getMemory() {
    return this.memory;
   }
 }
```

In Junit 4, we can apply:

i. Exception handling

ii. Setup function

iii. Timeout Rule

JUnit 4 Test Class for function Testing :

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.Timeout;
public class CalculatorTest {
  private Calculator calculator;
  public Timeout globalTimeout = Timeout.millis(1000);
@Before
  public void setup() {
    calculator = new calculator();
    calculator.storeInMemory(10);
  }
@Test(expected = ArithmeticException.class)
  public void testDivideByzero() {
      calculator.divide(5,0);
  }
```

```
@Test
public void testsqrtComputationTime()
{
double result = calculator.sqrt(16);
 assertEquals(4.0, result, 0.01);
}
@Test
 public void testMemoryfunction() {
  assertEquals(10, calculator.getMemory());
 }
}
```

Explanation of the JUnit 4 test class:

i. Setup Function

ii. Exception Handling Test

iii- Timeout rule

iv. Function Execution Test

v. Setup Verification

In conclusion, This JUnit 4 test class effectively applies exception testing, setup functions and timeouts.