1

a)

Actor

Definition: An actor represents any entity that interacts with the system. It can be a person, another system, or an external device that participates in one or more use-cases by exchanging information.

Example: In an online shopping system:

- **Customer**: A primary actor who browses products, places orders, and makes payments.
- **Warehouse System**: A secondary actor that processes shipping and inventory updates.

Use-Case

Definition: A use-case is a description of a specific interaction between an actor and the system to achieve a particular goal. It details the steps taken by the actor and the system to complete a function or process.

Example: In an online shopping system:

- **Place Order**: A use-case where the customer selects items, enters shipping details, and makes a payment.

System Boundary

Definition: The system boundary defines the scope of the system, indicating what is included within the system and what is external to it. It delineates the interface between the system and the actors.

Example: In an online shopping system:

- The system boundary would include all the internal processes such as product catalog, shopping cart, order processing, and payment handling.

- It would exclude external entities like the customer, payment gateways, and shipping services, which interact with the system but are not part of it.

3

b)

Functional Requirements

1. **Request Quiz**:
    - **User Function**: A user can request a quiz.
    - **System Function**: The system composes a quiz by selecting a set of questions from its database.
2. **Rate Answers**:
    - **System Function**: The system rates the user's answers to the quiz.
3. **Provide Hints**:
    - **User Function**: The user can request hints.
    - **System Function**: The system provides hints for questions if requested by the user.
4. **Manage Questions and Hints**:
    - **Tutor Function**: Tutors can provide questions and corresponding hints to the system.
5. **Certify Questions**:
    - **Examiner Function**: Examiners certify questions to ensure they are not too trivial and are sensible.
6. **User Authentication**:
    - **System Function**: The system should authenticate users (students, tutors, examiners) to allow appropriate access.
7. **Quiz Storage**:
    - **System Function**: The system should store quizzes and their results for later review by users and administrators.

8. **Question Database Management**:
    - **System Function**: The system should allow adding, updating, and deleting questions and hints in the database.

Non-Functional Requirements

1. **Performance**:
    - **System Requirement**: The system should compose and deliver quizzes within 2 seconds of the user's request.
    - **System Requirement**: Rating answers and providing hints should occur in real-time, with less than 1 second delay.
2. **Usability**:
    - **User Requirement**: The system interface should be intuitive and easy to navigate for users, tutors, and examiners.
    - **System Requirement**: The hint request process should be straightforward and accessible from each quiz question.
3. **Scalability**:
    - **System Requirement**: The system should be able to handle up to 10,000 concurrent users without performance degradation.
4. **Reliability**:
    - **System Requirement**: The system should have an uptime of 99.9%, ensuring it is almost always available for users.
5. **Security**:
    - **System Requirement**: User data, quiz results, and question databases must be securely stored and transmitted.
    - **System Requirement**: Access control mechanisms must be in place to ensure only authorized users (students, tutors, examiners) can perform their respective actions.
6. **Maintainability**:
    - **System Requirement**: The system should be designed in a modular way to facilitate easy updates and maintenance.
    - **System Requirement**: Documentation should be provided for all system functionalities and operations to assist in maintenance and upgrades.

7. **Compatibility**:
    - **System Requirement**: The system should be compatible with major web browsers and operating systems.
8. **Accessibility**:
    - **System Requirement**: The system should comply with accessibility standards (e.g., WCAG 2.1) to ensure it is usable by people with disabilities.

5

(b) Purpose of Black-Box and White-Box Testing Techniques:

Black-Box Testing:

- **Purpose**: Black-box testing focuses on evaluating the functionality and behavior of the software without knowledge of its internal structure or implementation details. Testers examine the inputs and outputs of the software to identify discrepancies between expected and actual behavior. The goal is to validate the software against specified requirements and user expectations, ensuring that it behaves as intended from an end-user perspective.

White-Box Testing:

- **Purpose**: White-box testing, also known as structural or glass-box testing, involves examining the internal structure, design, and implementation of the software. Testers analyze the code paths, logic, and control flow to identify potential defects and ensure code coverage. The objective is to verify the correctness of the code, identify vulnerabilities, and improve code quality by exercising different code paths and scenarios.

Need for Separate Operational Environment and Test Environment:

- **Operational Environment**: The operational environment is the production environment where the software is deployed and used by end-users. It represents the real-world conditions and constraints under which the software operates.
- **Test Environment**: The test environment is a controlled environment separate from the operational environment, specifically designed for testing and validation purposes. It allows testers to perform various types of testing, such as functional testing, performance testing, and security testing, without impacting the production environment.
- **Purpose**: Separating the test environment from the operational environment helps mitigate risks associated with testing activities, such as accidental data corruption, service disruptions, or security breaches. It provides a safe and isolated space for testers to experiment with different test scenarios and configurations, ensuring the reliability, stability, and security of the software before deployment to the production environment.

(c) Control Flow Graph, Branch Coverage, and Path Coverage:

Control Flow Graph:

css
```
         [1]
           |
          / \
        [2] [5]
         |     |
         |     |
        [3] [6]
         |     |
         |     |
        [4] [7]
```

```
  \  /
  [8]
```

- Branch Coverage:
  - Branches: [1-2], [2-3], [2-4], [5-6], [6-7], [7-8]
  - Total Branches: 6
  - Branch Coverage: (Number of Covered Branches / Total Branches) * 100% = (6 / 6) * 100% = 100%
- Path Coverage:
  - Paths:
    - Path 1: 1 → 2 → 3 → 4 → 8
    - Path 2: 1 → 2 → 5 → 6 → 7 → 8
  - Total Paths: 2
  - Path Coverage: (Number of Covered Paths / Total Paths) * 100% = (2 / 2) * 100% = 100%