1. An online shopping website is facing frequent downtime during sales.

How would you apply the SDLC phases to identify and permanently fix this issue?

Ans: A. **Requirement Analysis**

- Identify when downtime occurs (peak sales, flash sales, payment load).
- Gather logs, user complaints, server metrics.
- Define performance, scalability, and availability requirements.

B. **System Design**

- Design scalable architecture (load balancers, auto-scaling, caching, CDN).
- Plan database optimization and fault tolerance.

C. Implementation **(Development)**

- Optimize inefficient code and database queries.
- Implement caching (Redis), async processing, and scalable services.

D. **Testing**

- Perform load testing, stress testing, and performance testing.
- Simulate high traffic scenarios.

E. **Deployment**

- Deploy fixes using blue-green or canary deployment to reduce risk.

F. **Maintenance**

- Continuous monitoring, alerts, and periodic performance reviews.
- Apply fixes proactively before future sales.

2. A client wants to launch an MVP of an online food delivery app in 3 months.

Which SDLC model would you choose and why?

Ans: We will choose Agile Model because of the following below reason.
a. MVP needs **fast delivery** with limited features.

b. Agile supports **short iterations (sprints)** and rapid feedback.

c. Requirements may evolve based on user feedback.

d. Allows continuous improvement after launch.

3. After deployment, users report that the payment feature fails intermittently.

In which SDLC phase should this have been caught, and how would you prevent it next time?

Ans: In testing phase it should have been caught because intermittent failures indicate missing **integration, load, or reliability testing**.
We can prevent it using the below methods.

a. Add automated integration and regression tests.

b. Perform stress and concurrency testing on payment gateways.

c. Test under real-world scenarios (network delays, retries).

d. Monitor logs and failures continuously after release.

4. A business requirement changes midway while developing a subscription-based platform.

How does the traditional SDLC handle this, and what are its limitations?

Ans: a. Changes require revisiting earlier phases (requirements & design).

b. Leads to rework, increased cost, and delays.

The Limitations are:

a. Rigid structure, not flexible to change.

b. High dependency on early requirement clarity.

c. Late feedback from users.

d. Not suitable for evolving business needs.

5. An online banking application must meet strict security and compliance requirements.

How would you incorporate security testing into the SDLC lifecycle?

Ans: A. **Requirement Analysis**

- Define security standards (PCI-DSS, GDPR, ISO).
- Identify sensitive data and compliance needs.

B **Design**

- Secure architecture (encryption, authentication, authorization).
- Threat modeling and risk analysis.

C. **Development**

- Secure coding practices.
- Code reviews and static security analysis.

D. **Testing**

- Penetration testing, vulnerability scanning.
- Security and compliance testing.

E. **Deployment & Maintenance**

- Regular security audits.
- Patch management and continuous monitoring.

6. Your team delivered a feature, but it doesn't match the customer's expectation.

Which SDLC step likely failed, and how would you improve it?

Ans: SDLC step that likely failed is Requirement Analysis because Requirements were misunderstood, incomplete, or poorly documented.
We can improve it using below steps:
a. Conduct detailed requirement discussions.

b.Use user stories, wireframes, and prototypes.

c.Get written approvals and regular feedback.

d.Involve stakeholders throughout development.

7. A legacy e-commerce system needs to be migrated to a cloud-based architecture.

How would SDLC guide this migration process?

Ans:
A. **Requirement Analysis**

- Analyze legacy system limitations.
- Define migration goals (scalability, cost, performance).

B. **Design**

- Design cloud architecture (AWS/Azure/GCP).
- Decide migration strategy (re-host, re-platform, re-architect).

C. **Implementation**

- Migrate data, services, and applications incrementally.
- Refactor code where needed.

D. **Testing**

- Functional, performance, and security testing.
- Validate data integrity and system behavior.

E. **Deployment**

- Gradual rollout with rollback plans.

F. **Maintenance**

- Monitor cloud performance and costs.
- Optimize resources continuously.