Câu hỏi trắc nghiệm

Câu hỏi 1: Lớp nào dưới đây đại diện cho các đối tượng có dữ liệu và trạng thái cần quản lý trong hệ thống?

C. Lớp thực thể

Câu hỏi 2: Lớp biên trong hệ thống có vai trò gì?

C. Giao tiếp với người dùng hoặc hệ thống bên ngoài

Câu hỏi 3: Quan hệ nào giữa các lớp thể hiện sự kế thừa?

D. Inheritance

Câu hỏi 4: Sơ đồ lớp mô tả:

B. Các lớp và quan hệ giữa các lớp trong hệ thống

Câu hỏi 5: Quan hệ Include giữa các use case được dùng khi:

B. Một use case cần gọi một use case khác để hoàn thành chức năng6. Câu hỏi Câu hỏi 6: Scenario là gì?

B. Một kịch bản mô tả cách hệ thống và người dùng tương tác

Câu hỏi 7: Quan hệ nào sau đây biểu diễn việc một lớp chứa một lớp khác nhưng lớp con vẫn có thể tồn tại độc lập?

A. Aggregation

Câu hỏi 8: Sơ đồ tuần tự mô tả điều gì?

B. Thứ tự các thông điệp được trao đổi giữa các đối tượng

Câu hỏi 9: Lớp điều khiển trong mô hình MVC tương ứng với thành phần nào?

C. Control

Câu hỏi 10: Để biểu diễn quan hệ giữa các lớp, ta sử dụng sơ đồ nào?

C. Sơ đồ lớp

Câu hỏi ngắn

1. Lớp thực thể (Entity Class) là các lớp mô tả các thực thể trong hệ thống phần mềm, thường được ánh xạ trực tiếp đến bảng trong cơ sở dữ liệu. Lớp thực thể lưu trữ dữ liệu có trạng thái và thường tồn tại lâu dài trong hệ thống.
2. Lớp điều khiển (Control Class) là một loại lớp trong mô hình hướng đối tượng, đảm nhiệm vai trò trung gian giữa lớp giao diện (Boundary Class) và lớp thực thể (Entity Class). Lớp này chứa logic xử lý nghiệp vụ và điều phối luồng dữ liệu trong hệ thống.
3. Scenario là tập hợp các bước mô tả chi tiết về cách người dùng tương tác với hệ thống để hoàn thành một nhiệm vụ cụ thể. Mỗi scenario thể hiện một tình huống sử dụng phần mềm theo góc nhìn của người dùng.

4. Quan hệ Include giữa các use case được dùng khi một use case cần gọi một use case khác để hoàn thành chức năng.
5. Sơ đồ lớp (Class Diagram) là một loại sơ đồ trong UML (Unified Modeling Language) dùng để mô tả cấu trúc tĩnh của hệ thống phần mềm bằng cách biểu diễn các lớp, thuộc tính, phương thức và mối quan hệ giữa các lớp.
6. Quan hệ Aggregation (tập hợp) là một lớp chứa một hoặc nhiều thực thể của lớp khác, nhưng các thực thể này vẫn có thể tồn tại độc lập. Trong khi quan hệ Composition (thành phần) là một lớp chứa một hoặc nhiều thực thể của lớp khác, nhưng các thực thể này không thể tồn tại độc lập.
7. Sơ đồ tuần tự (Sequence Diagram) là một loại sơ đồ UML mô tả cách các đối tượng trong hệ thống tương tác với nhau theo thứ tự thời gian. Nó thể hiện trình tự các thông điệp (messages) được gửi giữa các đối tượng để thực hiện một chức năng cụ thể.
8. Quan hệ Extend giữa các use case được dùng khi một use case mở rộng một use case khác.
9. Lớp biên (Boundary Class) là một lớp đóng vai trò giao tiếp giữa hệ thống và môi trường bên ngoài, bao gồm người dùng và các hệ thống khác. Nó chịu trách nhiệm quản lý tương tác, xử lý đầu vào từ người dùng và hiển thị kết quả.
10. Sơ đồ cộng tác (Collaboration Diagram) (Communication Diagram) mô tả tương tác giữa các đối tượng như lược đồ tuần tự, nhưng tập trung vào mối quan hệ giữa các đối tượng thay vì trình tự thời gian.

Câu hỏi thảo luận nhóm

1. Thảo luận về vai trò của từng loại lớp (thực thể, biên, điều khiển) trong hệ thống.

- *Vai trò lớp thực thể:*

- Chứa dữ liệu và logic nghiệp vụ (business logic).
- Biểu diễn các đối tượng trong thế giới thực hoặc các khái niệm cốt lõi của hệ thống.
- Thường là các lớp mô hình hóa dữ liệu, ví dụ như User, Product, Order, v.v.

- *Vai trò lớp biên:*

• Giao diện người dùng: Xử lý tương tác giữa người dùng và hệ thống.

• Giao tiếp với hệ thống bên ngoài: Nhận dữ liệu từ API, hệ thống khác.

• Kiểm tra và xử lý đầu vào: Đảm bảo dữ liệu nhập vào hệ thống hợp lệ.

- *Vai trò lớp điều khiển:*

• Xử lý logic nghiệp vụ chính của ứng dụng.

• Điều phối các tương tác giữa giao diện người dùng và dữ liệu.

• Tạo điều kiện cho các lớp khác giao tiếp mà không làm tăng sự phụ thuộc lẫn nhau.

2. So sánh sự khác nhau giữa Aggregation và Composition.

Định nghĩa Aggregation:

- Là một kiểu quan hệ "has-a" giữa hai đối tượng, nhưng một đối tượng có thể tồn tại độc lập với đối tượng chứa nó.
- Nếu đối tượng cha bị xóa, đối tượng con vẫn có thể tồn tại.
- Thể hiện mối quan hệ "weak association" (liên kết yếu).

Ví dụ:

- Một trường đại học có nhiều sinh viên (University - Student): Nếu trường đại học bị đóng cửa, sinh viên vẫn tồn tại.
- Một đội bóng có nhiều cầu thủ (Team - Player): Nếu đội bóng giải thể, cầu thủ vẫn có thể chơi cho đội khác.

Đặc điểm Aggregation:

- Đối tượng con có thể thuộc nhiều đối tượng cha.
- Đối tượng con không bị ràng buộc bởi vòng đời của đối tượng cha.

Định nghĩa Composition:

- Là một kiểu quan hệ "has-a", nhưng mạnh hơn Aggregation vì đối tượng con phụ thuộc hoàn toàn vào đối tượng cha.
- Nếu đối tượng cha bị xóa, đối tượng con cũng sẽ bị xóa.
- Thể hiện mối quan hệ "strong association" (liên kết chặt chẽ).

Ví dụ:

- Một chiếc xe có động cơ (Car - Engine): Nếu xe bị hủy, động cơ cũng không còn tồn tại.
- Một ngôi nhà có các phòng (House - Room): Nếu ngôi nhà bị phá hủy, các phòng cũng mất theo.

Đặc điểm Composition:

- Đối tượng con không thể tồn tại nếu đối tượng cha bị hủy.
- Tạo mối quan hệ chặt chẽ giữa các đối tượng.

3. Thảo luận về tầm quan trọng của việc xây dựng sơ đồ lớp trong quá trình phân tích hệ thống.

- Trực quan hóa hệ thống: Giúp nhóm phát triển và khách hàng hiểu được cách các đối tượng liên kết với nhau.
- Hỗ trợ thiết kế và triển khai: Xác định các lớp chính, phương thức và thuộc tính cần có.
- Cải thiện bảo trì và mở rộng: Dễ dàng thay đổi, mở rộng hệ thống khi có yêu cầu mới.
- Giảm lỗi thiết kế: Phát hiện sớm các vấn đề như quan hệ sai giữa các lớp, giúp giảm chi phí sửa lỗi sau này.

4. Phân biệt sơ đồ tuần tự và sơ đồ cộng tác.

Tiêu chí	Sơ đồ tuần tự (Sequence Diagram)	Sơ đồ cộng tác (Collaboration Diagram)
Mục đích	Mô tả thứ tự các thông điệp giữa các đối tượng theo thời gian	Mô tả cách các đối tượng liên kết với nhau để thực hiện một tác vụ
Trọng tâm	Thứ tự thời gian của các tin nhắn (messages)	Cấu trúc quan hệ giữa các đối tượng
Biểu diễn	Sử dụng trục thời gian dọc để thể hiện luồng xử lý	Sử dụng các đường kết nối để thể hiện mối quan hệ giữa các đối tượng
Ví dụ	Đặt hàng trên website: Hiển thị quá trình từ khách hàng đặt hàng → kiểm tra kho → thanh toán → xác nhận	Sơ đồ liên kết giữa Customer, Order, Paymenttrong hệ thống thương mại điện tử

5. Thảo luận về vai trò của lớp điều khiển trong mô hình MVC.

- Điều phối luồng dữ liệu giữa Model và View.
- Xử lý logic nghiệp vụ trước khi cập nhật Model hoặc hiển thị dữ liệu lên View.
- Tách biệt giao diện và dữ liệu, giúp hệ thống dễ bảo trì và nâng cấp.
- Ví dụ: OrderController trong ứng dụng mua sắm online kiểm tra đơn hàng trước khi cập nhật vào cơ sở dữ liệu.

6. Tại sao cần viết các scenario khi phân tích hệ thống?

- Làm rõ yêu cầu: Giúp xác định các tình huống sử dụng phần mềm từ góc nhìn người dùng.
- Hỗ trợ thiết kế Use Case: Scenario giúp xây dựng các Use Case chính xác hơn.
- Cải thiện kiểm thử: Dựa vào scenario để viết test case, đảm bảo phần mềm hoạt động đúng.
- Ví dụ: Khi phân tích hệ thống ATM, một scenario có thể là "Người dùng nhập sai mật khẩu 3 lần liên tiếp".

7. Làm thế nào để đảm bảo rằng các use case được trích đầy đủ và chính xác?

- Phỏng vấn người dùng để hiểu rõ nhu cầu thực tế.
- Sử dụng kỹ thuật mô hình hóa như sơ đồ Use Case để xác định đầy đủ các trường hợp.
- Tổ chức workshop với các bên liên quan để xác minh và điều chỉnh Use Case.
- Xây dựng prototype hoặc mô phỏng để kiểm tra tính chính xác.
- Ví dụ: Khi thiết kế hệ thống đặt phòng khách sạn, cần kiểm tra xem các Use Case có bao gồm trường hợp hủy phòng, thay đổi đặt phòng hay không.

Câu 8: Đề xuất các công cụ hỗ trợ việc lập tài liệu kiểm thử và quản lý phiên bản.

- Công cụ lập tài liệu kiểm thử:
  - TestRail: Quản lý test case, theo dõi kết quả kiểm thử.
  - Zephyr (cho Jira): Tích hợp với Jira, hỗ trợ kiểm thử linh hoạt.
  - qTest: Hỗ trợ quản lý kiểm thử và báo cáo trực quan.
- Công cụ quản lý phiên bản:
  - Git (GitHub, GitLab, Bitbucket): Quản lý mã nguồn, theo dõi thay đổi.
  - Subversion (SVN): Quản lý phiên bản theo mô hình tập trung.
  - Perforce: Dùng cho các dự án lớn cần kiểm soát chặt chẽ.

Câu 9: Tại sao kiểm thử chấp nhận lại là một giai đoạn quan trọng trong phát triển phần mềm?

- Đảm bảo phần mềm đáp ứng yêu cầu của khách hàng: Kiểm thử chấp nhận (User Acceptance Testing - UAT) giúp xác nhận rằng sản phẩm phù hợp với nhu cầu thực tế.
- Phát hiện lỗi trước khi triển khai: Người dùng cuối kiểm tra phần mềm để phát hiện lỗi chưa được phát hiện trong quá trình phát triển.
- Tăng độ tin cậy của sản phẩm: Nếu phần mềm vượt qua UAT, nó có cơ hội cao hơn để hoạt động ổn định khi triển khai chính thức.
- Giảm rủi ro thất bại: Nếu phần mềm không đáp ứng yêu cầu, có thể điều chỉnh trước khi đưa vào sử dụng rộng rãi.

Câu 10: Thảo luận về các phương pháp hiệu quả để quản lý chất lượng phần mềm trong các dự án lớn.

- Áp dụng DevOps và CI/CD: Tích hợp kiểm thử tự động, triển khai liên tục giúp phát hiện lỗi sớm.
- Kiểm thử tự động: Dùng Selenium, JUnit, hoặc Cypress để giảm thiểu lỗi con người.
- Code Review và Static Analysis: Sử dụng SonarQube, ESLint để phát hiện lỗi trong code sớm.
- Quản lý yêu cầu rõ ràng: Dùng Jira, Trello để theo dõi yêu cầu và tiến độ.
- Bảo mật và kiểm thử hiệu năng: Dùng OWASP ZAP để kiểm thử bảo mật, JMeter để kiểm thử hiệu năng.

Câu hỏi tình huống

1. Một công ty phát triển phần mềm theo mô hình thác nước gặp vấn đề khi khách hàng yêu cầu thay đổi sau khi hoàn thành pha thiết kế. Đội phát triển nên xử lý như thế nào?

Trong mô hình thác nước, các giai đoạn phát triển phần mềm được thực hiện tuân theo trình tự từng bước, do đó việc thay đổi yêu cầu sau khi hoàn thành giai đoạn thiết kế sẽ gây khó khăn, tăng chi phí và làm trễn tiến độ dự án.

Các biện pháp xử lý bao gồm:

- Phân tích tác động: Xác định mức độ ảnh hưởng của sự thay đổi đối với tiến độ, chi phí và chất lượng dự án.
- Xem xét hợp đồng: Kiểm tra rà soát điều khoản hợp đồng để xem các thay đổi này có được phép không và xác định chi phí phát sinh.
- Thương lượng với khách hàng: Giải thích về độ phức tạp và chi phí tăng thêm, đề xuất phương án thay thế ít ảnh hưởng nhất.
- Lồng ghép quy trình linh hoạt: Sử dụng các nguyên tắc linh hoạt như Agile để giảm thiểu tác động của các thay đổi trên dự án.

2. Dự án phát triển phần mềm theo mô hình lặp và tăng trưởng liên tục bị trễ tiến độ do thiếu nhân lực. Là quản lý dự án, bạn sẽ làm gì?

- Đánh giá nguồn lực: Xác định nhân sự hiện tại, điểm yếu trong khâu nhân lực và khả năng làm việc của từng thành viên.
- Tuyển thêm nhân sự hoặc outsource: Tìm kiếm nhân sự mới hoặc hợp tác với bên thứ ba để bổ sung nhân lực cần thiết.
- Tối ưu quy trình: Xem xét các quy trình làm việc hiện tại và đề xuất cải thiện để giảm bốc lẫu công việc.
- Sử dụng công cụ tự động: Áp dụng các công cụ DevOps, CI/CD để tăng hiệu quả làm việc.

3. Trong quá trình phát triển phần mềm theo mô hình tiến trình linh hoạt, khách hàng không đưa ra phản hồi kịp thời. Đội phát triển nên xử lý ra sao?

- Chủ động nhắc nhở khách hàng: Gửi email, tổ chức họp để thu thập phản hồi.
- Tập trung vào các yêu cầu cốt lõi: Tiếp tục phát triển các chức năng quan trọng theo kế hoạch ban đầu.
- Linh hoạt trong quy trình: Chia nhóm làm việc độc lập để duy trì tiến độ.
- Dự báo trước rủi ro: Cảnh báo khách hàng về ảnh hưởng tiến độ do thiếu phản hồi

Câu 4. Dự án phát triển phần mềm gặp vấn đề khi khách hàng yêu cầu thay đổi lớn trong pha cài đặt. Đội phát triển nên xử lý thế nào?

Khi khách hàng yêu cầu thay đổi lớn trong pha cài đặt (deployment), đội phát triển cần xử lý một cách cẩn thận để tránh ảnh hưởng đến tiến độ và chất lượng phần mềm. Dưới đây là cách tiếp cận hợp lý:

1. Phân tích yêu cầu thay đổi

- Xác định rõ ràng nội dung thay đổi, lý do khách hàng muốn thay đổi.
- Đánh giá mức độ ảnh hưởng đến hệ thống hiện tại, bao gồm mã nguồn, cơ sở dữ liệu, UI/UX và các chức năng khác.

2. Đánh giá tác động và rủi ro

- Xác định thay đổi có ảnh hưởng đến bảo mật, hiệu suất hay không.
- Đánh giá công sức và thời gian cần thiết để thực hiện thay đổi.
- Nếu thay đổi quá lớn, có thể cần xem xét lại hợp đồng và ngân sách.

3. Thảo luận với khách hàng

- Nếu thay đổi ảnh hưởng lớn, đội phát triển cần trao đổi với khách hàng về tính khả thi, thời gian cần thiết và chi phí phát sinh (nếu có).
- Đề xuất các phương án thay thế hoặc triển khai thay đổi theo từng giai đoạn để giảm rủi ro.

4. Lập kế hoạch triển khai

- Nếu đồng ý với thay đổi, cần lập kế hoạch cập nhật phần mềm một cách hợp lý.
- Có thể áp dụng CI/CD (Continuous Integration/Continuous Deployment) để triển khai dễ dàng hơn.
- Nếu hệ thống đang chạy thực tế, cần có kế hoạch rollback (quay lại phiên bản cũ nếu có lỗi).

5. Thực hiện, kiểm thử và triển khai

- Triển khai thay đổi trong môi trường kiểm thử trước khi đưa vào hệ thống thực tế.
- Thực hiện kiểm thử toàn diện để đảm bảo phần mềm vẫn hoạt động ổn định.
- Chỉ triển khai khi mọi thứ đã sẵn sàng và được kiểm thử đầy đủ.

6. Tài liệu hóa và hỗ trợ sau triển khai

- Cập nhật tài liệu liên quan để đội phát triển và khách hàng dễ dàng theo dõi.

- Theo dõi phản hồi của khách hàng sau khi triển khai thay đổi để xử lý nhanh chóng nếu có vấn đề phát sinh.

Câu 5. Nhóm kiểm thử phát hiện nhiều lỗi chức năng trong phần mềm. Tuy nhiên, nhóm phát triển lại cho rằng đây không phải lỗi mà là tính năng. Là trưởng dự án, bạn sẽ làm gì?

Khi nhóm kiểm thử phát hiện nhiều lỗi chức năng nhưng nhóm phát triển lại cho rằng đó là tính năng, với vai trò trưởng dự án, bạn cần xử lý tình huống này một cách hợp lý để đảm bảo chất lượng phần mềm mà không gây xung đột nội bộ. Dưới đây là cách tiếp cận:

1. Xác minh vấn đề một cách khách quan

- Yêu cầu nhóm kiểm thử cung cấp bằng chứng cụ thể về các lỗi phát hiện được (log lỗi, ảnh chụp màn hình, mô tả tình huống xảy ra lỗi).
- Hỏi nhóm phát triển về lý do họ cho rằng đó là tính năng chứ không phải lỗi. Có tài liệu nào mô tả tính năng này không?

2. Kiểm tra lại tài liệu yêu cầu (SRS - Software Requirement Specification)

- So sánh hành vi của phần mềm với tài liệu yêu cầu ban đầu.
- Nếu hành vi của phần mềm không đúng với yêu cầu, thì đây là lỗi và nhóm phát triển cần sửa.
- Nếu hành vi phù hợp với tài liệu yêu cầu nhưng vẫn gây khó hiểu cho người dùng, có thể cần tinh chỉnh hoặc cập nhật tài liệu để tránh hiểu nhầm.

3. Họp thảo luận giữa hai nhóm

- Tổ chức một cuộc họp giữa nhóm kiểm thử và nhóm phát triển để thảo luận từng trường hợp cụ thể.
- Nếu hai bên không đồng thuận, có thể nhờ đến ý kiến của khách hàng hoặc nhóm phân tích nghiệp vụ (BA - Business Analyst) để xác định đúng sai.

4. Đưa ra quyết định cuối cùng

Dựa trên phân tích, có 3 kịch bản có thể xảy ra:
- Nếu đó thực sự là lỗi → Yêu cầu nhóm phát triển sửa lỗi.
- Nếu đó là tính năng nhưng gây nhầm lẫn → Cập nhật tài liệu hoặc tinh chỉnh giao diện để người dùng hiểu rõ hơn.
- Nếu đó là một tính năng quan trọng cần thay đổi → Trao đổi với khách hàng để xem xét cập nhật yêu cầu phần mềm.

5. Đảm bảo quy trình kiểm thử được thực hiện tốt

- Nếu phát sinh quá nhiều lỗi, có thể cần xem xét lại quy trình phát triển phần mềm, chẳng hạn như:
    - Bổ sung kiểm thử sớm hơn trong quy trình (Unit Test, Integration Test).
    - Áp dụng quy trình Code Review để giảm lỗi trước khi kiểm thử chính thức.
    - Cải thiện giao tiếp giữa nhóm phát triển và kiểm thử để tránh hiểu nhầm.
    -

Câu 6. Khách hàng yêu cầu bổ sung một tính năng mới khi phần mềm đã hoàn thành pha kiểm thử tích hợp. Đội phát triển nên làm gì?

Khi khách hàng yêu cầu bổ sung một tính năng mới sau khi phần mềm đã hoàn thành pha kiểm thử tích hợp, đội phát triển cần xử lý cẩn thận để tránh ảnh hưởng đến chất lượng và tiến độ của dự án. Dưới đây là cách tiếp cận hợp lý:

1. Phân tích yêu cầu mới

- Xác định rõ ràng yêu cầu của khách hàng: tính năng cần bổ sung là gì, mục tiêu ra sao, có ảnh hưởng đến các chức năng hiện tại không?
- Xác định mức độ phức tạp và thời gian cần thiết để phát triển tính năng này.

2. Đánh giá tác động

- Ảnh hưởng đến phần mềm: Tính năng mới có làm thay đổi các module đã kiểm thử hay không? Có gây lỗi hoặc xung đột với hệ thống không?
- Ảnh hưởng đến tiến độ: Nếu phần mềm sắp bàn giao, có thể phải lùi thời gian phát hành.
- Chi phí phát sinh: Nếu thay đổi lớn, có thể cần thương lượng lại hợp đồng với khách hàng.

3. Thảo luận với khách hàng

- Nếu tính năng mới có thể gây trì hoãn hoặc ảnh hưởng đến chất lượng phần mềm, đội phát triển cần trao đổi với khách hàng về các phương án:
    - Tích hợp tính năng mới vào phiên bản hiện tại: Nếu thay đổi nhỏ, có thể bổ sung ngay.
    - Lên kế hoạch cho bản cập nhật sau: Nếu thay đổi lớn, có thể đưa vào phiên bản tiếp theo để không ảnh hưởng đến tiến độ bàn giao hiện tại.

4. Lập kế hoạch triển khai

- Nếu đồng ý bổ sung, cần cập nhật tài liệu thiết kế và lộ trình phát triển.
- Phân chia công việc, đảm bảo kiểm thử lại sau khi bổ sung.
- Nếu cần, có thể sử dụng CI/CD (Continuous Integration/Continuous Deployment) để triển khai nhanh hơn.

5. Kiểm thử lại toàn bộ hệ thống

- Vì phần mềm đã qua pha kiểm thử tích hợp, việc thêm tính năng mới có thể gây lỗi không mong muốn.
- Cần thực hiện kiểm thử lại, bao gồm kiểm thử đơn vị (Unit Testing), kiểm thử tích hợp (Integration Testing) và kiểm thử hệ thống (System Testing).

6. Cập nhật tài liệu và triển khai

- Cập nhật tài liệu hướng dẫn sử dụng, tài liệu thiết kế hệ thống để đảm bảo đội phát triển và khách hàng đều hiểu rõ tính năng mới.
- Sau khi kiểm thử ổn định, triển khai tính năng mới vào hệ thống.

Câu 7. Một công ty phát triển phần mềm nhỏ muốn xây dựng nhóm SQA nhưng gặp khó khăn về ngân sách. Hãy đề xuất giải pháp.

Khi một công ty phần mềm nhỏ muốn xây dựng nhóm SQA (Software Quality Assurance) nhưng gặp khó khăn về ngân sách, cần có giải pháp tối ưu để đảm bảo chất lượng phần mềm mà không tốn quá nhiều chi phí. Dưới đây là một số phương án khả thi:

1. Tận dụng nhân sự nội bộ (Cross-functional Team)

- Đào tạo lập trình viên về kiểm thử phần mềm để họ có thể tự kiểm thử (Unit Test, Integration Test) trước khi gửi sang nhóm kiểm thử.
- Kết hợp QA vào nhóm phát triển, nghĩa là một số thành viên trong team có thể đảm nhận vai trò kiểm thử mà không cần tuyển dụng thêm nhân sự.
- Lập trình viên viết test case tự động thay vì kiểm thử thủ công hoàn toàn.

2. Sử dụng công cụ kiểm thử miễn phí

- Kiểm thử tự động:
    - Selenium (Web Testing)
    - JUnit/TestNG (Java Unit Testing)
    - PyTest (Python Testing)
    - Postman (API Testing)
- Kiểm thử hiệu năng:
    - JMeter (Performance Testing)
- Kiểm thử bảo mật:
    - OWASP ZAP (Security Testing)

Những công cụ này giúp tiết kiệm chi phí so với các phần mềm thương mại đắt tiền như LoadRunner, UFT.

3. Thuê kiểm thử viên bên ngoài (Outsourcing/Freelancer)

- Nếu không đủ ngân sách thuê nhân viên toàn thời gian, công ty có thể thuê freelancer QA trên các nền tảng như:
    - Upwork, Fiverr, Freelancer
    - Các công ty cung cấp dịch vụ kiểm thử theo nhu cầu

Cách này giúp tiết kiệm chi phí mà vẫn đảm bảo kiểm thử chất lượng.

4. Áp dụng kiểm thử dựa vào cộng đồng (Crowdsourced Testing)

- Tham gia các nền tảng kiểm thử như TestIO, uTest, Bugcrowd, nơi cộng đồng tester trên toàn thế giới kiểm thử phần mềm và báo lỗi.
- Phù hợp cho ứng dụng web, mobile hoặc game cần thử nghiệm trên nhiều thiết bị khác nhau.

5. Ưu tiên kiểm thử quan trọng (Risk-based Testing)

- Không kiểm thử tất cả, chỉ tập trung vào các chức năng quan trọng nhất và có nguy cơ lỗi cao.
- Sử dụng phương pháp Exploratory Testing (kiểm thử khám phá) thay vì viết test case đầy đủ, giúp tiết kiệm thời gian và nhân lực.

6. Xây dựng quy trình kiểm thử đơn giản nhưng hiệu quả

- Áp dụng các tiêu chuẩn như Agile Testing, CI/CD để giảm thiểu lỗi ngay từ đầu.
- Sử dụng Peer Review (đồng nghiệp kiểm tra lẫn nhau) để phát hiện lỗi trước khi đưa phần mềm đến kiểm thử viên.

Câu 8. Trong quá trình làm tài liệu kiểm thử, nhóm phát triển không thống nhất được về nội dung cần đưa vào tài liệu. Là trưởng nhóm, bạn sẽ giải quyết vấn đề này như thế nào?

Khi nhóm phát triển không thống nhất về nội dung tài liệu kiểm thử, với vai trò trưởng nhóm, bạn cần xử lý tình huống này một cách chuyên nghiệp để đảm bảo tiến độ và chất lượng tài liệu. Dưới đây là cách tiếp cận:

1. Xác định nguyên nhân gây ra sự không thống nhất

Có thể có nhiều nguyên nhân dẫn đến tranh cãi về nội dung tài liệu kiểm thử, chẳng hạn:

- Không có tiêu chuẩn rõ ràng về tài liệu kiểm thử.
- Mỗi người có cách hiểu khác nhau về phạm vi kiểm thử.
- Không có sự liên kết chặt chẽ với tài liệu yêu cầu (SRS - Software Requirement Specification).
- Tranh cãi về mức độ chi tiết: kiểm thử nên chi tiết hay chỉ tập trung vào các trường hợp quan trọng?

2. Dựa vào tài liệu yêu cầu (SRS) và tiêu chuẩn kiểm thử

- Mọi tài liệu kiểm thử cần bám sát yêu cầu phần mềm. Nếu một test case không dựa trên yêu cầu cụ thể, nó có thể không cần thiết.
- Nếu chưa có tiêu chuẩn tài liệu kiểm thử, hãy tham khảo IEEE 829 (Test Documentation Standard) để thống nhất format.

3. Tổ chức cuộc họp để thảo luận và thống nhất nội dung

- Mời tất cả thành viên nhóm phát triển và kiểm thử tham gia.
- Chia nội dung tài liệu kiểm thử thành các phần cụ thể (test case, test plan, test strategy...) để dễ thảo luận.
- Nếu có mâu thuẫn, ưu tiên quyết định dựa trên tài liệu yêu cầu và tiêu chuẩn thay vì ý kiến cá nhân.
- Nếu vẫn không thống nhất, có thể nhờ Product Owner hoặc Business Analyst hỗ trợ làm rõ yêu cầu.

4. Phân công trách nhiệm rõ ràng

- Chia nhóm làm việc theo từng phần của tài liệu. Ví dụ:
    - Tester chịu trách nhiệm viết test case.
    - Developer cung cấp thông tin kỹ thuật liên quan.
    - BA/Product Owner xác nhận tính hợp lệ của nội dung kiểm thử.
- Việc phân công này giúp giảm tranh cãi và làm việc hiệu quả hơn.

5. Xây dựng quy trình kiểm thử chuẩn để tránh tranh cãi về sau

- Xác định tiêu chuẩn chung về cách viết test case.

- Sử dụng công cụ quản lý kiểm thử như TestRail, Jira, Zephyr để giúp team dễ theo dõi và cập nhật tài liệu.
- Thiết lập quy trình review tài liệu kiểm thử trước khi hoàn thiện.

Câu 9. Dự án phát triển phần mềm cho một ngân hàng yêu cầu bảo mật cao. Đề xuất cách lập kế hoạch kiểm thử cho dự án này.

Phần mềm ngân hàng liên quan đến dữ liệu nhạy cảm, giao dịch tài chính và phải tuân thủ các tiêu chuẩn bảo mật nghiêm ngặt như ISO 27001, PCI-DSS. Vì vậy, kế hoạch kiểm thử cần được xây dựng kỹ lưỡng với các loại kiểm thử phù hợp.

1. Xác định phạm vi và mục tiêu kiểm thử

- Đảm bảo tính chính xác, hiệu suất, bảo mật và tuân thủ các quy định bảo mật.
- Kiểm thử cả chức năng (Functional Testing) và phi chức năng (Non-functional Testing).
- Đảm bảo không có lỗ hổng bảo mật có thể bị khai thác.
- Xác định các rủi ro bảo mật tiềm ẩn và cách xử lý.

2. Các loại kiểm thử cần thực hiện

- Kiểm thử chức năng (Functional Testing)

- Kiểm thử giao dịch tài chính: Chuyển tiền, thanh toán hóa đơn, nạp/rút tiền, tính lãi suất...
- Kiểm thử xác thực và phân quyền: Đăng nhập, OTP, vai trò người dùng (User, Admin, Teller)..
- Kiểm thử luồng xử lý lỗi: Xử lý khi nhập sai mật khẩu, lỗi kết nối, tài khoản bị khóa…
- Kiểm thử tính toàn vẹn dữ liệu: Kiểm tra số dư trước và sau giao dịch có khớp không.

- Kiểm thử bảo mật (Security Testing)

- Penetration Testing (Pentest): Tấn công giả lập để tìm lỗ hổng bảo mật.
- Kiểm thử SQL Injection: Thử nhập mã độc SQL để kiểm tra bảo vệ cơ sở dữ liệu.
- Kiểm thử XSS (Cross-Site Scripting): Kiểm tra xem hệ thống có bị tấn công chèn mã JavaScript không.
- Kiểm thử bảo vệ dữ liệu: Kiểm tra mã hóa dữ liệu (AES, TLS/SSL), lưu trữ thông tin thẻ ngân hàng đúng chuẩn PCI-DSS.
- Kiểm thử chống tấn công Brute-force: Giới hạn số lần đăng nhập sai, CAPTCHA.
- Kiểm thử bảo vệ API: Xác minh API có sử dụng xác thực và mã hóa đúng cách không.

- Kiểm thử hiệu suất (Performance Testing)

- Kiểm thử tải (Load Testing): Kiểm tra hệ thống có thể xử lý bao nhiêu giao dịch/giây.
- Kiểm thử stress (Stress Testing): Kiểm tra hệ thống có bị sập khi quá tải không.
- Kiểm thử khả năng mở rộng (Scalability Testing): Kiểm tra xem hệ thống có thể mở rộng để xử lý nhiều giao dịch hơn không.

- Kiểm thử tuân thủ (Compliance Testing)

Đảm bảo phần mềm tuân thủ các tiêu chuẩn như:

- ISO 27001 (Bảo mật thông tin).

- PCI-DSS (Bảo mật thẻ thanh toán).
- GDPR (Bảo vệ dữ liệu cá nhân nếu hoạt động tại châu Âu).

3. Công cụ hỗ trợ kiểm thử

- Kiểm thử tự động: Selenium, Appium (UI), JUnit (API).
- Pentest: Burp Suite, Metasploit, OWASP ZAP.
- Kiểm thử hiệu suất: JMeter, Locust.
- Kiểm thử tuân thủ: OpenSCAP, Nexpose.

4. Xây dựng chiến lược kiểm thử

- Kiểm thử sớm: Áp dụng Shift-Left Testing, kiểm thử bảo mật ngay từ khi viết code.
- Tự động hóa kiểm thử để tăng hiệu quả.
- Lập kế hoạch kiểm thử định kỳ, đặc biệt là kiểm thử bảo mật.
- Mô phỏng các kịch bản tấn công thực tế để kiểm tra khả năng phòng thủ.
- Làm việc với chuyên gia bảo mật để đảm bảo tuân thủ quy định.

5. Kế hoạch báo cáo và xử lý lỗi

- Ghi nhận lỗi trong công cụ như Jira, TestRail.
- Phân loại lỗi: Lỗi nghiêm trọng (Critical), cao (High), trung bình (Medium), thấp (Low).
- Lên kế hoạch vá lỗi và kiểm thử lại trước khi phát hành.

Câu 10.Sau khi triển khai phần mềm, khách hàng phát hiện ra một số lỗi bảo mật nghiêm trọng. Đội phát triển cần xử lý ra sao để khắc phục vấn đề và lấy lại niềm tin từ khách hàng?

Khi phần mềm đã được triển khai nhưng khách hàng phát hiện lỗi bảo mật nghiêm trọng, đội phát triển cần hành động nhanh chóng và chuyên nghiệp để khắc phục vấn đề và lấy lại niềm tin từ khách hàng. Dưới đây là kế hoạch xử lý:

1. Xác định mức độ nghiêm trọng của lỗi

- Phân loại lỗi bảo mật:
  - Rất nghiêm trọng (Critical): Lỗi cho phép hacker truy cập dữ liệu nhạy cảm, đánh cắp tài khoản, phá hủy hệ thống...
  - Nghiêm trọng (High): Lỗ hổng có thể bị khai thác nhưng cần điều kiện đặc biệt.
  - Trung bình (Medium): Ảnh hưởng đến một số tính năng nhưng không đe dọa toàn bộ hệ thống.
  - Nhẹ (Low): Lỗi không gây rủi ro ngay lập tức.
- Nếu là lỗi Critical/High, cần kích hoạt kế hoạch phản ứng sự cố (Incident Response Plan) ngay lập tức.

2. Tạm thời khắc phục và giảm thiểu rủi ro

- Nếu lỗi có thể bị khai thác ngay lập tức, cần tạm thời vô hiệu hóa tính năng liên quan.

- Chặn các IP đáng ngờ hoặc cập nhật firewall/WAF để ngăn chặn tấn công.
- Cảnh báo khách hàng và yêu cầu họ thực hiện các biện pháp bảo vệ (ví dụ: đổi mật khẩu nếu dữ liệu bị rò rỉ).

3. Điều tra nguyên nhân gốc rễ (Root Cause Analysis - RCA)

- Kiểm tra log hệ thống, dữ liệu bị ảnh hưởng và cách lỗi xảy ra.
- Xác định lỗi đến từ code, cấu hình hệ thống hay bên thứ ba.
- Nếu có khả năng bị tấn công, cần kiểm tra dấu hiệu xâm nhập và thu thập bằng chứng.

4. Cập nhật phần mềm và kiểm thử lại

- Sửa lỗi bảo mật bằng cách cập nhật code hoặc cấu hình hệ thống.
- Kiểm thử bảo mật lại để đảm bảo lỗi đã được khắc phục:
  - Penetration Testing để kiểm tra xem lỗi còn tồn tại không.
  - Kiểm thử hồi quy (Regression Testing) để đảm bảo bản vá không gây lỗi mới.
  - Kiểm thử hiệu suất (Performance Testing) nếu lỗi ảnh hưởng đến tốc độ hệ thống.
- Tung bản vá (Security Patch) ngay khi có thể.

5. Giao tiếp minh bạch với khách hàng

- Thông báo chính thức về lỗi và cách xử lý. Tránh giấu giếm vì có thể mất uy tín nếu bị phát hiện sau này.
- Nếu dữ liệu bị rò rỉ, cần thông báo theo đúng quy định (GDPR, PCI-DSS, ISO 27001...).
- Cam kết cải thiện hệ thống để tránh lỗi lặp lại.

6. Cải thiện quy trình để tránh lặp lại lỗi

- Tăng cường kiểm thử bảo mật (Security Testing) trong quy trình phát triển.
- Áp dụng DevSecOps: Kiểm thử bảo mật sớm thay vì đợi đến cuối.
- Sử dụng công cụ quét bảo mật tự động như OWASP ZAP, SonarQube, Snyk...
- Huấn luyện đội ngũ về bảo mật, đặc biệt là về lập trình an toàn.

7. Theo dõi sau khi khắc phục

- Giám sát hệ thống để đảm bảo lỗi không tái diễn.
- Tiếp tục lắng nghe phản hồi từ khách hàng.
- Nếu cần, bồi thường hoặc cung cấp hỗ trợ đặc biệt để lấy lại lòng tin.