CÂU HỎI TRẮC NGHIỆM (1)

1. 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?

A. Lớp biên
B. Lớp điều khiển
C. Lớp thực thể
D. Lớp giao diện

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

A. Xử lý dữ liệu
B. Điều khiển luồng công việc
C. Giao tiếp với người dùng hoặc hệ thống bên ngoài
D. Lưu trữ dữ liệu

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

A. Association
B. Aggregation
C. Composition
D. Inheritance

4. Câu hỏi 4: Sơ đồ lớp mô tả:

A. Quan hệ giữa các đối tượng trong một luồng xử lý
B. Các lớp và quan hệ giữa các lớp trong hệ thống
C. Giao diện người dùng của hệ thống
D. Thứ tự luồng xử lý giữa các đối tượng

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

A. Một use case mở rộng một use case khác
B. Một use case cần gọi một use case khác để hoàn thành chức năng
C. Một use case kế thừa một use case khác
D. Một use case được sử dụng bởi hệ thống bên ngoài

6. Câu hỏi 6: Scenario là gì?

A. Một sơ đồ mô tả lớp trong hệ thống
B. Một kịch bản mô tả cách hệ thống và người dùng tương tác
C. Một sơ đồ mô tả các luồng dữ liệu
D. Một tài liệu thiết kế giao diện

7. 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
B. Composition
C. Association
D. Inheritance

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

A. Quan hệ giữa các lớp
B. Thứ tự các thông điệp được trao đổi giữa các đối tượng
C. Cấu trúc dữ liệu của hệ thống
D. Các chức năng mà hệ thống cung cấp

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

A. Model

B. View
C. Control
D. Entity

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

A. Sơ đồ tuần tự
B. Sơ đồ cộng tác
C. Sơ đồ lớp
D. Sơ đồ use case

CÂU HỎI NGẮN (1)

1. Lớp thực thể là gì?

Lớp thực thể (Entity class) là lớp đại diện cho các đối tượng trong thế giới thực hoặc các khái niệm nghiệp vụ trong hệ thống. Nó chứa dữ liệu và các phương thức xử lý dữ liệu đó. Ví dụ: Trong hệ thống quản lý sinh viên, lớp SinhVien là lớp thực thể.

2. Lớp điều khiển có vai trò gì trong hệ thống?

Lớp điều khiển (Control class) chịu trách nhiệm xử lý logic nghiệp vụ, điều phối luồng dữ liệu giữa lớp biên và lớp thực thể. Nó không chứa dữ liệu mà chỉ thực hiện các thao tác xử lý. Ví dụ: Lớp DangKyHoc điều khiển quá trình đăng ký môn học.

3. Scenario là gì?

Scenario (kịch bản) là một chuỗi các bước mô tả cách người dùng tương tác với hệ thống để đạt được một mục tiêu cụ thể. Nó thường là một trường hợp cụ thể của một use case. Ví dụ: Kịch bản “Người dùng đăng nhập thành công” là một scenario của use case “Đăng nhập”.

4. Quan hệ Include giữa các use case là gì?

Quan hệ **Include** là mối quan hệ trong đó một use case luôn gọi đến một use case khác như một phần bắt buộc. Use case được include là phần dùng chung cho nhiều use case khác. Ví dụ: Use case “Xác thực người dùng” được include trong “Đăng nhập” và “Đổi mật khẩu”.

5. Mục đích của sơ đồ lớp là gì?

Sơ đồ lớp (Class diagram) dùng để mô tả cấu trúc tĩnh của hệ thống, thể hiện các lớp, thuộc tính, phương thức và mối quan hệ giữa các lớp. Nó giúp thiết kế kiến trúc phần mềm rõ ràng và dễ bảo trì.

6. Quan hệ Aggregation khác gì so với Composition?

Đặc điểm	Aggregation	Composition

Mức độ liên kết	Quan hệ “có nhưng không sở hữu”	Quan hệ “có và sở hữu”
Vòng đời	Đối tượng con có thể tồn tại độc lập	Đối tượng con bị hủy khi đối tượng cha bị hủy
Ví dụ	Lớp Giáo viên và Trường họ	Lớp Ngôi nhà và Phòng

7. Sơ đồ tuần tự là gì?

Sơ đồ tuần tự (Sequence diagram) mô tả sự tương tác giữa các đối tượng theo trình tự thời gian. Nó thể hiện các thông điệp được gửi giữa các đối tượng để thực hiện một chức năng cụ thể.

8. Quan hệ Extend giữa các use case là gì?

Quan hệ **Extend** là mối quan hệ trong đó một use case có thể mở rộng hành vi của use case khác trong một số điều kiện nhất định. Use case mở rộng không bắt buộc phải xảy ra. Ví dụ: Use case “Xem lịch sử mua hàng” có thể extend từ “Mua hàng” nếu người dùng yêu cầu.

9. Lớp biên có vai trò gì trong hệ thống?

Lớp biên (Boundary class) là lớp giao tiếp giữa người dùng và hệ thống. Nó xử lý các tương tác như nhập dữ liệu, hiển thị thông tin, và truyền yêu cầu đến lớp điều khiển. Ví dụ: Giao diện form đăng nhập là lớp biên.

10. Sơ đồ cộng tác là gì?

Sơ đồ cộng tác (Collaboration diagram) mô tả sự tương tác giữa các đối tượng trong hệ thống, tập trung vào mối quan hệ và vai trò của các đối tượng trong việc thực hiện một chức năng.

CÂU HỎI THẢO LUẬN NHÓM (1)

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

- Lớp thực thể (Entity): Quản lý dữ liệu, trạng thái và các quy tắc nghiệp vụ liên quan đến dữ liệu.
- Lớp biên (Boundary): Là cầu nối giữa người dùng hoặc hệ thống bên ngoài với hệ thống nội bộ (thường là giao diện hoặc API).
- Lớp điều khiển (Control): Xử lý luồng nghiệp vụ, điều phối tương tác giữa lớp biên và lớp thực thể.

→ Ba loại lớp này giúp phân tách rõ trách nhiệm, tăng tính mô-đun, dễ bảo trì và dễ mở rộng cho hệ thống.

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

Tiêu chí	Aggregation	Composition
Bản chất	Quan hệ "có – thuộc", nhưng đối tượng con có thể tồn tại độc lập.	Quan hệ "chứa – tạo", đối tượng con không thể tồn tại nếu đối tượng cha bị xóa.
Biểu diễn UML	Hình thoi rỗng (◇)	Hình thoi đen (◆)
Ví dụ	"Lớp học" chứa "Học sinh" – học sinh có thể tồn tại riêng.	"Ngôi nhà" chứa "Phòng" – khi nhà bị xóa, phòng cũng bị xóa.

→ Aggregation là liên kết lỏng lẻo, Composition là liên kết chặt chẽ.

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

- Sơ đồ lớp giúp xác định cấu trúc tĩnh của hệ thống: các lớp, thuộc tính, quan hệ và ràng buộc giữa chúng.
- Là cơ sở cho giai đoạn thiết kế hướng đối tượng, giúp hiểu rõ thành phần và mối quan hệ của hệ thống.
- Giúp phát hiện sớm lỗi logic hoặc thiếu sót trong yêu cầu trước khi lập trình.
- Là cầu nối giữa phân tích và thiết kế, đảm bảo hệ thống được mô hình hóa rõ ràng và chính xác.

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	Thể hiện thứ tự thời gian các thông điệp giữa các đối tượng.	Thể hiện cấu trúc tương tác giữa các đối tượng.
Trọng tâm	Trình tự gọi thông điệp theo thời gian (dọc).	Mối liên kết và sự phối hợp giữa đối tượng.
Ưu điểm	Dễ hiểu, thể hiện rõ trình tự xử lý.	Thể hiện mối quan hệ giữa đối tượng tốt hơn.

→ Cả hai sơ đồ đều mô tả tương tác giữa đối tượng, nhưng sơ đồ tuần tự nhấn mạnh thời gian, còn sơ đồ cộng tác nhấn mạnh mối liên kết.

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

- Trong mô hình MVC (Model – View – Controller), lớp điều khiển đóng vai trò trung gian giữa Model và View.
- Nhận yêu cầu từ View → xử lý logic → tương tác với Model → trả kết quả lại cho View.

- Giúp phân tách giao diện và xử lý, tăng khả năng tái sử dụng, mở rộng và bảo trì hệ thống.

→ Lớp điều khiển là “bộ não điều phối” của hệ thống MVC.

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

- Scenario giúp mô tả chi tiết luồng tương tác giữa người dùng và hệ thống cho từng use case cụ thể.
- Giúp phát hiện các trường hợp đặc biệt, ngoại lệ hoặc thiếu sót trong yêu cầu.
- Là tài liệu nền tảng cho việc thiết kế sơ đồ tuần tự và kiểm thử sau này.

→ Viết scenario giúp hiểu sâu cách hệ thống vận hành, đảm bảo yêu cầu được phân tích đầy đủ và chính xác.

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

Để đảm bảo trích xuất đầy đủ và chính xác các use case, cần thực hiện các bước sau:

- Phỏng vấn và khảo sát người dùng: Thu thập yêu cầu từ các bên liên quan để hiểu rõ mục tiêu và hành vi của hệ thống.
- Phân tích nghiệp vụ kỹ lưỡng: Hiểu rõ quy trình nghiệp vụ để xác định các chức năng cần thiết.
- Sử dụng kỹ thuật mô hình hóa: Áp dụng sơ đồ use case để trực quan hóa các chức năng và mối quan hệ.
- Xác định tác nhân (actor) rõ ràng: Mỗi use case phải gắn với một hoặc nhiều tác nhân cụ thể.
- Kiểm tra tính đầy đủ và nhất quán: So sánh với yêu cầu nghiệp vụ, đảm bảo không bỏ sót hoặc trùng lặp.
- Xác nhận với người dùng: Trình bày sơ đồ use case để người dùng kiểm tra và xác nhận.

8. Thảo luận về mối quan hệ giữa use case và scenario.

- **Use case** là mô tả tổng quát về một chức năng mà hệ thống cung cấp cho tác nhân.
- **Scenario** là một trường hợp cụ thể của use case, mô tả chi tiết từng bước tương tác.

Mối quan hệ:

- Một use case có thể có nhiều scenario: kịch bản chính (main flow) và các kịch bản thay thế (alternative flows).
- Scenario giúp làm rõ cách use case hoạt động trong các tình huống thực tế.
- Việc xây dựng scenario giúp kiểm tra tính đầy đủ và khả năng xử lý ngoại lệ của use case.[

9. Phân tích ưu và nhược điểm của việc sử dụng sơ đồ tuần tự trong thiết kế hệ thống.

Ưu điểm	Nhược điểm
Hiển thị rõ ràng thứ tự tương tác giữa các đối tượng	Có thể trở nên phức tạp nếu có nhiều đối tượng và thông điệp
Giúp hiểu luồng xử lý nghiệp vụ theo thời gian	Không thể hiện rõ cấu trúc hệ thống như sơ đồ lớp

Hữu ích trong việc phát hiện lỗi logic hoặc thiếu sót	Khó bảo trì nếu hệ thống thay đổi nhiều
Hỗ trợ lập trình viên trong việc triển khai chức năng	Không phù hợp để mô tả toàn bộ hệ thống

10.Thảo luận về cách cải thiện chất lượng kịch bản sử dụng (scenario) trong quá trình phân tích.

Để nâng cao chất lượng scenario, có thể áp dụng các cách sau:

- Xác định rõ mục tiêu của scenario: Mỗi kịch bản phải thể hiện rõ mục đích người dùng muốn đạt được.
- Mô tả chi tiết từng bước: Trình bày cụ thể các hành động, phản hồi và điều kiện xảy ra.
- Phân loại kịch bản: Tách biệt kịch bản chính và các kịch bản thay thế, xử lý ngoại lệ.
- Sử dụng ngôn ngữ đơn giản, dễ hiểu: Tránh dùng thuật ngữ kỹ thuật gây khó khăn cho người dùng.
- Kiểm tra và xác nhận với người dùng: Đảm bảo kịch bản phản ánh đúng nhu cầu và hành vi thực tế.
- Áp dụng mẫu chuẩn (template): Giúp thống nhất cách trình bày và dễ dàng so sánh, đánh giá.

CÂU HỎI TÌNH HUỐNG (1)

1. Trong quá trình phân tích hệ thống quản lý thư viện, nhóm phát triển phát hiện một số yêu cầu mới từ khách hàng sau khi đã viết xong các scenario. Nhóm phát triển nên xử lý như thế nào?

→ Ghi nhận và phân tích các yêu cầu mới, đánh giá ảnh hưởng đến các scenario đã có, sau đó cập nhật hoặc bổ sung scenario và tài liệu yêu cầu. Tất cả thay đổi phải được khách hàng xác nhận trước khi tiếp tục.

2. Một nhóm phát triển gặp khó khăn khi xác định các lớp điều khiển trong hệ thống. Hãy đề xuất giải pháp.

→ Nhóm nên xem lại các use case để tìm ra các hành vi điều phối giữa lớp biên và lớp thực thể.

Mỗi use case thường có một lớp điều khiển tương ứng chịu trách nhiệm xử lý logic nghiệp vụ chính.

3. Sau khi hoàn thành sơ đồ lớp, khách hàng yêu cầu thêm một số chức năng mới. Nhóm phát triển cần làm gì để cập nhật sơ đồ lớp?

→ Áp dụng quy trình quản lý thay đổi, phân tích chức năng mới để xác định lớp mới cần thêm hoặc lớp hiện có cần mở rộng, sau đó cập nhật sơ đồ lớp và tài liệu mô hình tương ứng.

4. Khi viết các scenario cho use case "Đăng ký khóa học", nhóm phát triển gặp tình huống có nhiều trường hợp ngoại lệ. Làm thế nào để xử lý tình huống này?

→ Liệt kê và phân loại các ngoại lệ (ví dụ: nhập sai, dữ liệu thiếu, khóa học đầy), sau đó mô tả mỗi ngoại lệ thành một scenario riêng hoặc dùng phần “Extension” trong use case để trình bày rõ luồng thay thế.

5. Trong quá trình xây dựng sơ đồ tuần tự, một số đối tượng không có vai trò rõ ràng. Nhóm phát triển nên làm gì?
→ Xem lại use case và sơ đồ lớp để xác định đúng vai trò từng đối tượng. Nếu đối tượng không cần thiết, loại bỏ; nếu cần, đặt lại trách nhiệm cho phù hợp theo nguyên tắc “mỗi đối tượng có một nhiệm vụ rõ ràng”.

6. Sau khi xây dựng sơ đồ lớp, nhóm phát triển phát hiện ra một số quan hệ giữa các lớp bị sai. Hãy đề xuất cách sửa chữa.
→ Phân tích lại mối quan hệ thực tế giữa các lớp (kế thừa, kết tập, kết hợp) và chỉnh sửa lại loại quan hệ tương ứng.
Đồng thời, kiểm tra sự thống nhất với sơ đồ tuần tự và use case để đảm bảo tính logic toàn hệ thống.

7. Một nhóm phát triển gặp khó khăn khi mô tả các quan hệ giữa các use case. Hãy đề xuất giải pháp.
→ Nên xem lại các kịch bản (scenario) và xác định hành vi dùng chung hoặc mở rộng để quyết định dùng Include hay Extend.
Nếu use case có hành vi độc lập, giữ tách biệt; nếu có hành vi lặp lại, dùng Include.

8. Trong dự án phát triển phần mềm quản lý bán hàng, nhóm phát triển cần xác định các lớp biên cho hệ thống. Hãy đưa ra đề xuất phù hợp.
→ Lớp biên (Boundary Class) nên được xác định dựa trên các điểm giao tiếp giữa người dùng và hệ thống, ví dụ:
“Giao diện đăng nhập”, “Màn hình quản lý sản phẩm”, “API đặt hàng”.

9. Khách hàng yêu cầu thêm chức năng mới sau khi các scenario đã được hoàn thiện. Nhóm phát triển nên làm gì?
→ Ghi nhận yêu cầu mới, phân tích tác động, sau đó thêm hoặc chỉnh sửa các scenario có liên quan, đồng thời cập nhật sơ đồ use case và tài liệu mô hình phân tích.

10. Trong quá trình xây dựng sơ đồ cộng tác, một số đối tượng không tương tác đúng theo yêu cầu. Hãy đề xuất cách giải quyết.
→ Kiểm tra lại logic luồng tương tác và vai trò các đối tượng, điều chỉnh lại thứ tự hoặc thông điệp giữa các đối tượng để đảm bảo đúng hành vi mong muốn.
Nếu cần, cập nhật sơ đồ tuần tự trước để làm rõ trình tự tương tác.

CÂU HỎI TRẮC NGHIỆM (2)

1. Pha nào trong tiến trình phát triển phần mềm tập trung vào việc thu thập yêu cầu từ khách hàng?

A. Pha thiết kế
B. Pha kiểm thử
C. Pha lấy yêu cầu

D. Pha triển khai

2. Mô hình phát triển phần mềm nào có các vòng lặp liên tục và cải tiến sản phẩm sau mỗi lần lặp?

A. Mô hình thác nước
B. Mô hình xoắn ốc
C. Mô hình Agile
D. Mô hình kiểm thử

3. Trong kỹ thuật lấy yêu cầu phần mềm, phương pháp nào giúp khai thác thông tin từ khách hàng bằng cách quan sát cách họ làm việc?

A. Phỏng vấn
B. Khảo sát
C. Quan sát
D. Phân tích tài liệu

4. Trong chuẩn hóa cơ sở dữ liệu, dạng chuẩn nào loại bỏ các phụ thuộc bắc cầu?

A. 1NF
B. 2NF
C. 3NF
D. BCNF

5. Trong UML, sơ đồ nào giúp mô tả sự tương tác giữa các đối tượng theo thời gian?

A. Sơ đồ lớp
B. Sơ đồ tuần tự
C. Sơ đồ trạng thái
D. Sơ đồ hoạt động

6. UML, sơ đồ nào giúp mô tả sự tương tác giữa các đối tượng theo thời gian?

A. Sơ đồ lớp
B. Sơ đồ tuần tự
C. Sơ đồ trạng thái
D. Sơ đồ hoạt động

7. CMMI mức 5 – Optimizing tập trung vào điều gì?

A. Kiểm soát quy trình
B. Định lượng quy trình
C. Cải tiến quy trình
D. Xác định quy trình

8. Đâu là ưu điểm chính của mô hình phát triển phần mềm Agile?

A. Linh hoạt và thay đổi nhanh chóng
B. Yêu cầu chi tiết ngay từ đầu
C. Kiểm thử chỉ diễn ra ở giai đoạn cuối
D. Phù hợp với tất cả các loại dự án

9. Nguyên tắc DRY (Don't Repeat Yourself) trong lập trình có ý nghĩa gì?

A. Hạn chế viết code lặp lại
B. Viết code dễ đọc hơn
C. Tăng tốc độ thực thi của chương trình
D. Giảm chi phí phát triển phần mềm

10. Yếu tố nào quan trọng nhất trong việc thiết kế một lớp trong lập trình hướng đối tượng?

A. Đặt tên biến dễ hiểu
B. Đảm bảo tính đóng gói và tái sử dụng
C. Viết phương thức càng nhiều càng tốt

D. Dùng số nguyên thay vì số thực

11. Phát biểu nào đúng về kiểm thử phần mềm?

A. Kiểm thử chỉ diễn ra sau khi phát triển xong sản phẩm
B. Kiểm thử giúp tìm ra tất cả lỗi trong phần mềm
C. Kiểm thử có thể thực hiện song song với phát triển phần mềm
D. Kiểm thử không quan trọng nếu phần mềm được viết tốt

CÂU HỎI NGẮN (2)

1. Định nghĩa phần mềm và công nghệ phần mềm

Phần mềm là tập hợp các chương trình máy tính thực hiện một hoặc nhiều chức năng cụ thể.

Công nghệ phần mềm là ngành nghiên cứu và áp dụng các nguyên lý kỹ thuật để phát triển, vận hành và bảo trì phần mềm một cách hệ thống.

2. Mô tả ngắn gọn về các mô hình vòng đời phát triển phần mềm phổ biến

Mô hình thác nước (Waterfall): Phát triển tuần tự qua các giai đoạn.
Mô hình xoắn ốc (Spiral): Kết hợp lặp và phân tích rủi ro.
Mô hình Agile: Phát triển linh hoạt, chia nhỏ thành các vòng lặp ngắn.
Mô hình V: Kiểm thử song song với phát triển.

3. Ba loại yêu cầu phần mềm chính

Yêu cầu chức năng: Mô tả hành vi và chức năng hệ thống.
Yêu cầu phi chức năng: Mô tả hiệu suất, bảo mật, tính khả dụng.
Yêu cầu miền nghiệp vụ: Ràng buộc từ môi trường hoặc lĩnh vực ứng dụng.

4. Vai trò của sơ đồ lớp UML trong thiết kế phần mềm

Sơ đồ lớp UML giúp mô hình hóa cấu trúc tĩnh của hệ thống, thể hiện các lớp, thuộc tính, phương thức và mối quan hệ giữa chúng, hỗ trợ thiết kế và hiểu kiến trúc phần mềm.

5. Tại sao kiểm thử phần mềm quan trọng?

Kiểm thử giúp phát hiện lỗi, đảm bảo chất lượng, độ tin cậy và hiệu suất của phần mềm trước khi triển khai, giảm rủi ro và chi phí sửa lỗi sau này.

6. Định nghĩa nguyên lý SOLID

SOLID là 5 nguyên lý thiết kế hướng đối tượng:

S: Single Responsibility
O: Open/Closed
L: Liskov Substitution
I: Interface Segregation
D: Dependency Inversion Giúp phần mềm dễ mở rộng, bảo trì và ít lỗi.

7. Khác biệt giữa kiểm thử hộp đen và hộp trắng

Hộp đen: Kiểm thử chức năng mà không biết nội dung bên trong mã nguồn.
Hộp trắng: Kiểm thử logic bên trong mã nguồn, kiểm tra luồng điều khiển và dữ liệu.

8. Quy trình thiết kế cơ sở dữ liệu từ sơ đồ lớp UML

Xác định các lớp làm bảng dữ liệu.
Chuyển thuộc tính thành cột.
Xác định khóa chính và khóa ngoại từ mối quan hệ.
Chuẩn hóa dữ liệu.
Tạo sơ đồ ER và triển khai vào hệ quản trị CSDL.

9. Ba ưu điểm của mô hình Agile

Phản hồi nhanh với thay đổi yêu cầu.
Tăng cường sự hợp tác giữa nhóm và khách hàng.

Phát hành phần mềm sớm và liên tục.

10. Các giai đoạn chuẩn hóa cơ sở dữ liệu

1NF: Loại bỏ dữ liệu lặp, đảm bảo tính nguyên tử.

2NF: Loại bỏ phụ thuộc một phần vào khóa chính.

3NF: Loại bỏ phụ thuộc bắc cầu. (Có thể tiếp tục đến BCNF, 4NF tùy yêu cầu hệ thống.)

CÂU HỎI THẢO LUẬN NHÓM (2)

1. So sánh mô hình phát triển phần mềm Agile và mô hình Waterfall.

- Waterfall là mô hình phát triển tuyến tính gồm các giai đoạn cố định, thực hiện nối tiếp nhau, khó thay đổi khi đã hoàn thành một pha:

  Yêu cầu → thiết kế → triển khai → kiểm thử → bảo trì
- Trong khi đó, Agile là mô hình phát triển linh hoạt, chia dự án thành nhiều vòng lặp (iteration/sprint), trong mỗi vòng có thể phân tích, thiết kế, lập trình và kiểm thử song song. Agile khuyến khích phản hồi liên tục từ khách hàng và dễ thích ứng khi yêu cầu thay đổi.

2.Lợi ích của việc sử dụng UML trong thiết kế phần mềm là gì?

UML (Unified Modeling Language) giúp mô tả, trực quan hóa và tài liệu hóa các thành phần của hệ thống phần mềm một cách thống nhất. Việc sử dụng UML giúp:

- Hiểu rõ yêu cầu và mối quan hệ giữa các thành phần.
- Giao tiếp dễ dàng hơn giữa các thành viên nhóm (phân tích, thiết kế, lập trình, kiểm thử).
- Giảm sai sót trong quá trình thiết kế và lập trình.
- Tạo tài liệu kỹ thuật có thể tái sử dụng cho các giai đoạn phát triển hoặc bảo trì sau này.

3.Làm thế nào để đảm bảo phần mềm có thể bảo trì tốt trong tương lai?

Để phần mềm dễ bảo trì, cần:

- Thiết kế hệ thống module hóa, phân lớp rõ ràng (theo mô hình MVC).
- Viết code tuân thủ chuẩn, có chú thích đầy đủ và đặt tên biến/hàm có ý nghĩa.
- Tách biệt logic nghiệp vụ và giao diện người dùng để dễ thay đổi từng phần.
- Viết unit test để kiểm thử các module độc lập.
- Lưu trữ tài liệu thiết kế và mã nguồn trong hệ thống quản lý phiên bản như Git để thuận tiện cho việc cập nhật và mở rộng.

4.Tại sao các công ty phần mềm thường sử dụng mô hình phát triển lặp (Iterative Development)?

- Mô hình phát triển lặp cho phép xây dựng phần mềm qua nhiều chu kỳ (iteration), mỗi lần lặp đều cải tiến dựa trên phản hồi từ người dùng hoặc kết quả kiểm thử.
- Lý do các công ty ưa chuộng mô hình này là vì:
  - Dễ thích ứng với thay đổi yêu cầu.
  - Có thể phát hành sớm sản phẩm thử nghiệm (prototype) để nhận phản hồi.
  - Giúp phát hiện lỗi sớm, giảm chi phí sửa lỗi về sau.
  - Cải thiện chất lượng phần mềm liên tục qua từng phiên bản.

5.Vai trò của kiến trúc phần mềm trong việc xây dựng một hệ thống phần mềm phức tạp.

Kiến trúc phần mềm là bộ khung tổng thể mô tả cách các thành phần của hệ thống tương tác với nhau. Nó giúp:

- Định hướng cho việc thiết kế, phát triển và mở rộng hệ thống.
- Tối ưu hiệu suất, khả năng bảo trì và khả năng mở rộng.

- Giúp các nhóm phát triển phối hợp dễ dàng hơn thông qua một cấu trúc thống nhất.
- Đảm bảo hệ thống đáp ứng được các yêu cầu phi chức năng như bảo mật, hiệu năng, khả năng phục hồi.

6.Làm thế nào để đảm bảo rằng một hệ thống phần mềm đáp ứng được yêu cầu bảo mật?

Để đảm bảo bảo mật, cần áp dụng nhiều lớp phòng thủ:

- Xác thực người dùng (Authentication) và phân quyền truy cập (Authorization) chặt chẽ.
- Mã hóa dữ liệu nhạy cảm (sử dụng SSL/TLS, AES, hashing mật khẩu).
- Kiểm tra đầu vào (Input Validation) để ngăn chặn SQL Injection, XSS, CSRF.
- Cập nhật định kỳ framework và thư viện để vá lỗ hổng.
- Thực hiện kiểm thử bảo mật (Penetration Testing) định kỳ.
- Áp dụng nhật ký hoạt động (Audit log) để theo dõi các hành vi đáng ngờ trong hệ thống.

7.So sánh kiểm thử đơn vị (Unit Testing) và kiểm thử tích hợp (Integration Testing).

- Kiểm thử đơn vị (Unit Testing) tập trung vào việc kiểm tra từng đơn vị nhỏ nhất của mã nguồn như hàm hoặc lớp riêng lẻ, trong khi kiểm thử tích hợp (Integration Testing) kiểm tra sự phối hợp giữa các mô-đun đã được kiểm thử đơn vị để đảm bảo chúng hoạt động đúng khi kết hợp lại.
- Unit Testing giúp phát hiện lỗi sớm và dễ dàng xác định vị trí lỗi, còn Integration Testing giúp phát hiện các lỗi phát sinh từ sự tương tác giữa các thành phần phần mềm

8.Những thách thức chính trong việc thu thập yêu cầu phần mềm là gì?

- Những thách thức chính trong việc thu thập yêu cầu phần mềm bao gồm sự không rõ ràng từ phía khách hàng, thay đổi yêu cầu liên tục, khó khăn trong việc ưu tiên tính năng, và sự khác biệt trong kỳ vọng giữa các bên liên quan.
- Ngoài ra, việc diễn đạt yêu cầu bằng ngôn ngữ kỹ thuật dễ gây hiểu nhầm, và đôi khi người dùng không thể mô tả chính xác những gì họ cần

9.Cách áp dụng quy trình Scrum vào dự án phát triển phần mềm thực tế.

- Để áp dụng quy trình Scrum vào dự án phát triển phần mềm thực tế, nhóm cần xác định rõ các vai trò như Product Owner, Scrum Master và Development Team.
- Dự án được chia thành các Sprint ngắn (thường từ 1 đến 4 tuần), trong đó nhóm phát triển hoàn thành một phần chức năng cụ thể. Mỗi Sprint bắt đầu bằng Sprint Planning, có Daily Scrum để cập nhật tiến độ, kết thúc bằng Sprint Review và Sprint Retrospective để cải tiến quy trình.

→ Scrum giúp tăng tính linh hoạt, phản hồi nhanh với thay đổi, và cải thiện chất lượng sản phẩm thông qua kiểm tra liên tục

10.Tại sao việc tối ưu hóa mã nguồn lại quan trọng

- Việc tối ưu hóa mã nguồn quan trọng vì nó giúp cải thiện hiệu suất hệ thống, giảm tiêu tốn tài nguyên, tăng khả năng bảo trì và mở rộng, đồng thời giảm thiểu lỗi tiềm ẩn.
- Mã nguồn tối ưu giúp chương trình chạy nhanh hơn, dễ đọc hơn và dễ kiểm tra hơn. Quá trình này thường bao gồm việc tái cấu trúc mã (refactoring), loại bỏ đoạn mã dư thừa, và áp dụng các kỹ thuật lập trình hiệu quả để đảm bảo phần mềm hoạt động ổn định và dễ nâng cấp về sau

CÂU HỎI TÌNH HUỐNG (2)

1. Bạn là một kỹ sư phần mềm trong một nhóm phát triển, khách hàng liên tục thay đổi yêu cầu. Bạn sẽ xử lý như thế nào?

→ Tôi sẽ áp dụng quy trình quản lý thay đổi (Change Management Process). Trước tiên, ghi nhận tất cả yêu cầu thay đổi từ khách hàng và phân tích ảnh hưởng của từng thay đổi đến phạm vi, chi phí và tiến độ dự án. Sau đó, tôi sẽ trao đổi với trưởng nhóm hoặc khách hàng để thống nhất ưu tiên và chỉ thực hiện những thay đổi thật sự cần thiết. Việc này giúp đảm bảo tiến độ dự án không bị ảnh hưởng quá nhiều và vẫn đáp ứng nhu cầu thực tế của khách hàng.

2. Trong quá trình kiểm thử, bạn phát hiện một lỗi nghiêm trọng nhưng trưởng nhóm quyết định không sửa chữa. Bạn sẽ làm gì?

→ Tôi sẽ trình bày rõ mức độ nghiêm trọng và rủi ro tiềm ẩn nếu lỗi này không được khắc phục, đồng thời cung cấp bằng chứng kiểm thử (log, ảnh chụp, báo cáo lỗi). Nếu nhóm vẫn giữ nguyên quyết định, tôi sẽ đề xuất lập kế hoạch khắc phục ở bản vá (patch) tiếp theo và ghi chú lỗi vào tài liệu QA để đảm bảo không bị bỏ qua trong các phiên bản sau. Mục tiêu là đảm bảo chất lượng sản phẩm mà không làm gián đoạn tiến độ chung.

3. Một dự án phần mềm gặp tình trạng chậm tiến độ do thay đổi yêu cầu liên tục từ khách hàng. Bạn sẽ đề xuất giải pháp gì?

→ Tôi sẽ đề xuất áp dụng **mô hình phát triển linh hoạt (Agile/Scrum)** để chia nhỏ yêu cầu thành các sprint ngắn, giúp nhóm dễ dàng thích ứng với thay đổi. Ngoài ra, sẽ cùng khách hàng thống nhất phạm vi tối thiểu khả thi (MVP) để hoàn thành trước, rồi phát triển bổ sung ở các giai đoạn sau. Điều này giúp dự án duy trì tiến độ mà vẫn đáp ứng được mục tiêu kinh doanh.

4. Nhóm của bạn đang thiết kế cơ sở dữ liệu cho một hệ thống thương mại điện tử. Làm thế nào để đảm bảo thiết kế cơ sở dữ liệu không bị dư thừa?

→ Tôi sẽ thực hiện **chuẩn hóa cơ sở dữ liệu (Database Normalization)** đến ít nhất dạng chuẩn 3NF để loại bỏ dữ liệu trùng lặp và phụ thuộc bắc cầu. Đồng thời, xây dựng **mô hình ERD (Entity-Relationship Diagram)** để xác định rõ quan hệ giữa các bảng, khóa chính – khóa ngoại. Sau đó, tiến hành rà soát và kiểm thử mô hình bằng dữ liệu mẫu để đảm bảo hệ thống lưu trữ hiệu quả, nhất quán và dễ mở rộng.

5. Bạn cần lựa chọn giữa hai mô hình phát triển phần mềm: Waterfall và Agile. Bạn sẽ chọn mô hình nào cho một dự án startup công nghệ? Tại sao?

→ Tôi sẽ chọn **mô hình Agile**, vì startup thường có yêu cầu thay đổi nhanh chóng và cần phản hồi sớm từ thị trường. Agile cho phép chia nhỏ dự án thành các sprint ngắn, linh hoạt trong việc điều chỉnh yêu cầu và liên tục kiểm thử, triển khai để cải thiện sản phẩm. Ngược lại, Waterfall cứng nhắc và không phù hợp với môi trường cần tốc độ và sáng tạo cao như startup.

6. Một dự án phần mềm lớn gặp vấn đề về hiệu suất. Những bước nào cần thực hiện để tối ưu hiệu suất phần mềm?

→ Trước hết, tôi sẽ tiến hành **đo lường và phân tích hiệu năng** (Performance Profiling) để xác định nguyên nhân chính (CPU, RAM, I/O, Database, Network…). Sau đó, tôi sẽ tối ưu truy vấn SQL, sử dụng **caching**, phân trang dữ liệu, tối ưu thuật toán, giảm tải cho máy

chủ bằng **load balancing** hoặc **microservices** nếu cần. Cuối cùng, thực hiện kiểm thử hiệu năng (stress test, load test) để đảm bảo hệ thống hoạt động ổn định.

7. Một hệ thống đang hoạt động có nhiều lỗi bảo mật. Bạn sẽ làm gì để tăng cường bảo mật mà không ảnh hưởng đến người dùng hiện tại?

→ Tôi sẽ lập danh sách các lỗ hổng, **đánh giá mức độ rủi ro**, sau đó triển khai vá lỗi theo thứ tự ưu tiên mà không ảnh hưởng trực tiếp đến hệ thống đang hoạt động. Tôi sẽ áp dụng các biện pháp như mã hóa dữ liệu, xác thực hai lớp, kiểm tra đầu vào (input validation), cập nhật framework/thư viện an toàn, và thực hiện kiểm thử bảo mật (penetration test). Tất cả thay đổi sẽ được kiểm tra trên môi trường staging trước khi áp dụng vào production.

8. Khi thiết kế phần mềm, nhóm của bạn có nhiều ý kiến trái ngược nhau về cách triển khai một tính năng. Làm thế nào để đưa ra quyết định tốt nhất?

→ Tôi sẽ tổ chức một buổi họp ngắn (meeting review) để mọi người trình bày quan điểm, sau đó đánh giá các phương án dựa trên tiêu chí: hiệu quả kỹ thuật, chi phí, thời gian và trải nghiệm người dùng. Nếu vẫn chưa thống nhất, tôi sẽ đề xuất **thử nghiệm A/B** hoặc tạo bản mô phỏng nhỏ (prototype) để so sánh thực tế, từ đó đưa ra quyết định dựa trên dữ liệu và kết quả đo lường khách quan.

9. Khách hàng yêu cầu hệ thống phải có giao diện thân thiện với người dùng. Bạn sẽ làm gì để đảm bảo hệ thống đáp ứng tiêu chí này?

→ Tôi sẽ bắt đầu bằng việc **phân tích người dùng mục tiêu (User Persona)**, sau đó tạo **wireframe/prototype** và tiến hành **kiểm thử khả dụng (usability testing)** với nhóm người dùng mẫu. Dựa vào phản hồi, tôi sẽ cải thiện giao diện để đảm bảo các yếu tố UX/UI: dễ dùng, trực quan, tương thích đa nền tảng và hỗ trợ truy cập nhanh. Mục tiêu là giúp người dùng thao tác tự nhiên và đạt được mục tiêu trong ít bước nhất.

10. Công ty bạn vừa tiếp nhận một dự án phần mềm cũ, nhưng không có tài liệu hướng dẫn. Bạn sẽ làm gì để hiểu và tiếp tục phát triển hệ thống này?

→ Tôi sẽ tiến hành **phân tích mã nguồn (code review)** để hiểu cấu trúc và luồng xử lý của hệ thống, đồng thời sử dụng các công cụ **reverse engineering** để tái tạo sơ đồ lớp, sơ đồ cơ sở dữ liệu và tài liệu thiết kế. Tôi cũng sẽ thiết lập môi trường chạy thử (staging) để quan sát hành vi thực tế. Nếu có thể, tôi sẽ phỏng vấn các thành viên cũ hoặc khách hàng để thu thập thêm thông tin. Sau khi hiểu rõ hệ thống, tôi sẽ cập nhật tài liệu kỹ thuật (technical documentation) và đề xuất kế hoạch bảo trì, mở rộng.