

## Chương 3

### PHẦN TRẮC NGHIỆM

Câu hỏi 1: Pha nào trong mô hình lý thuyết vòng đời phát triển phần mềm chịu trách nhiệm chuyển đổi yêu cầu thành đặc tả kỹ thuật?

- A. Pha thiết kế
- B. Pha lấy yêu cầu
- C. Pha phân tích**
- D. Pha cài đặt

2. Câu hỏi 2: Mô hình vòng đời nào phát triển phần mềm bằng cách tạo các phiên bản nhỏ và tăng dần tính năng?

- A. Mô hình thác nước
- B. Mô hình lặp và tăng trưởng**
- C. Mô hình bản mẫu nhanh
- D. Mô hình tiến trình linh hoạt

3. Câu hỏi 3: Pha bảo trì trong vòng đời phát triển phần mềm bao gồm hoạt động nào?

- A. Viết mã nguồn
- B. Sửa lỗi và cập nhật tính năng mới**
- C. Gỡ bỏ phần mềm
- D. Phân tích yêu cầu

4. Câu hỏi 4: Mô hình thác nước phù hợp nhất với loại dự án nào?

- A. Dự án nhỏ, đơn giản
- B. Dự án có yêu cầu rõ ràng và ít thay đổi**
- C. Dự án yêu cầu linh hoạt cao
- D. Dự án mã nguồn mở

5. Câu hỏi 5: Trong mô hình xoắn ốc, mỗi vòng xoắn tương ứng với:

- A. Một pha kiểm thử

**B. Một chu kỳ lặp của toàn bộ quy trình phát triển**

C. Một phiên bản phần mềm nhỏ

D. Một lần phân tích rủi ro

6. Câu hỏi 6: Điểm yếu lớn nhất của mô hình xây và sửa là gì?

A. Khó phát triển nhanh

B. Tốn nhiều chi phí bảo trì

**C. Khó kiểm soát chất lượng**

D. Không phù hợp với dự án nhỏ

7. Câu hỏi 7: Mô hình nào tập trung vào việc tạo các nguyên mẫu nhanh để thu thập phản hồi từ khách hàng?

A. Mô hình lặp và tăng trưởng

**B. Mô hình bản mẫu nhanh**

C. Mô hình xoắn ốc

D. Mô hình tiến trình linh hoạt

8. Câu hỏi 8: Pha nào kết thúc vòng đời phát triển phần mềm?

A. Pha bảo trì

B. Pha cài đặt

**C. Pha giải thể**

D. Pha thiết kế

9. Câu hỏi 9: Điểm khác biệt chính giữa mô hình lặp và tăng trưởng với mô hình thác nước là gì?

A. Mô hình thác nước lặp lại nhiều lần

**B. Mô hình lặp và tăng trưởng phát triển theo từng đợt nhỏ**

C. Mô hình lặp và tăng trưởng không có giai đoạn kiểm thử

D. Mô hình thác nước không có giai đoạn bảo trì

10. Câu hỏi 10: Mô hình nào có khả năng thích nghi tốt nhất với sự thay đổi của yêu cầu khách hàng?

A. Mô hình thác nước

B. Mô hình xoắn ốc

C. Mô hình xây và sửa

D. Mô hình tiến trình linh hoạt

## **PHẦN CÂU HỎI NGẮN**

### **1. Pha lấy yêu cầu là gì và có vai trò gì trong vòng đời phát triển phần mềm?**

Pha lấy yêu cầu đóng vai trò quan trọng trong sự thành công của dự án phần mềm, với các ý nghĩa sau:

- Xác định phạm vi dự án: Giúp định hình rõ ràng những gì cần được phát triển, tránh phát sinh tính năng không cần thiết.
- Cơ sở cho thiết kế và phát triển: Các yêu cầu thu thập được sẽ làm nền tảng cho việc thiết kế hệ thống, lập trình và kiểm thử.
- Giảm rủi ro và chi phí: Việc xác định sai hoặc thiếu yêu cầu có thể dẫn đến sửa đổi tốn kém về sau. Một pha lấy yêu cầu tốt giúp giảm thiểu nguy cơ này.
- Tăng sự hài lòng của khách hàng: Khi phần mềm được phát triển đúng theo yêu cầu ban đầu, khách hàng sẽ hài lòng hơn và ít cần thay đổi.
- Hỗ trợ kiểm thử và bảo trì: Các yêu cầu được ghi nhận rõ ràng giúp đội QA có tiêu chí để kiểm thử và giúp đội bảo trì hiểu rõ hệ thống khi cần nâng cấp.

### **2. Mô hình thác nước hoạt động như thế nào?**

- Mô hình thác nước (Waterfall Model) là một mô hình phát triển phần mềm tuyến tính, trong đó quá trình phát triển được chia thành các giai đoạn liên tiếp. Mỗi giai đoạn phải được hoàn thành trước khi chuyển sang giai đoạn tiếp theo, không có sự quay lại giai đoạn trước.

### **3. Mô hình lặp và tăng trưởng khác gì so với mô hình thác nước?**

- Mô hình lặp và tăng trưởng (Incremental and Iterative Model) và mô hình thác nước (Waterfall Model) có nhiều điểm khác biệt quan trọng:

#### **1. Cách phát triển phần mềm**

Mô hình thác nước: Phát triển theo từng giai đoạn tuần tự (phân tích, thiết kế, triển khai, kiểm thử, bảo trì). Mỗi giai đoạn phải hoàn thành trước khi chuyển sang giai đoạn tiếp theo.

Mô hình lặp và tăng trưởng: Phát triển theo từng vòng lặp, mỗi vòng là một phần của hệ thống hoàn chỉnh, có thể sử dụng được ngay.

#### **2. Tính linh hoạt**

Mô hình thác nước: Ít linh hoạt, khó thay đổi yêu cầu sau khi đã hoàn thành một giai đoạn.

Mô hình lặp và tăng trưởng: Linh hoạt hơn, có thể thay đổi yêu cầu trong các lần lặp tiếp theo.

### 3. Quản lý rủi ro

Mô hình thác nước: Rủi ro cao vì sản phẩm chỉ hoàn thành ở giai đoạn cuối, nếu có lỗi nghiêm trọng sẽ khó khắc phục.

Mô hình lặp và tăng trưởng: Rủi ro thấp hơn do có thể kiểm tra và điều chỉnh liên tục.

### 4. Thời gian ra sản phẩm

Mô hình thác nước: Chỉ có sản phẩm cuối cùng sau khi hoàn thành tất cả các giai đoạn.

Mô hình lặp và tăng trưởng: Có thể cung cấp phiên bản phần mềm hoạt động ngay từ những lần lặp đầu tiên.

### 5. Ứng dụng

Mô hình thác nước: Phù hợp với các dự án có yêu cầu rõ ràng ngay từ đầu, ít thay đổi (ví dụ: phát triển hệ thống nhúng, phần mềm quản lý cố định).

Mô hình lặp và tăng trưởng: Phù hợp với các dự án có yêu cầu thay đổi liên tục, cần phản hồi nhanh (ví dụ: phát triển phần mềm web, ứng dụng di động).

Tóm lại, mô hình thác nước phù hợp với các dự án có kế hoạch chặt chẽ, trong khi mô hình lặp và tăng trưởng linh hoạt hơn, giúp phát triển phần mềm nhanh chóng và thích nghi với thay đổi.

## 4. Mục tiêu của pha bảo trì là gì?

-Pha bảo trì (Maintenance Phase) là giai đoạn cuối cùng trong vòng đời phát triển phần mềm (SDLC), nhằm đảm bảo rằng phần mềm tiếp tục hoạt động tốt sau khi triển khai. Các mục tiêu chính của pha bảo trì bao gồm:

### 1. Sửa lỗi và cải thiện độ ổn định

Khắc phục lỗi phát sinh trong quá trình sử dụng mà chưa được phát hiện trước đó.

Cải thiện hiệu suất và tính ổn định của hệ thống.

### 2. Thích nghi với các thay đổi

Cập nhật phần mềm để phù hợp với các thay đổi trong môi trường hệ thống (hệ điều hành, phần cứng, mạng, v.v.).

Điều chỉnh phần mềm theo yêu cầu mới của khách hàng hoặc tổ chức.

### 3. Cải tiến và nâng cấp tính năng

Thêm các tính năng mới để đáp ứng nhu cầu của người dùng.

Tối ưu hóa hiệu suất, giao diện và trải nghiệm người dùng.

### 4. Bảo mật và vá lỗ hổng

Phát hiện và sửa các lỗ hổng bảo mật nhằm bảo vệ dữ liệu và hệ thống.

Cập nhật các cơ chế bảo mật để chống lại các mối đe dọa mới.

### 5. Hỗ trợ kỹ thuật và duy trì hoạt động liên tục

Hỗ trợ người dùng trong quá trình sử dụng.

Đảm bảo hệ thống hoạt động ổn định, không bị gián đoạn do sự cố kỹ thuật.

Tóm lại, mục tiêu của pha bảo trì là giữ cho phần mềm luôn hoạt động ổn định, an toàn, đáp ứng tốt nhu cầu của người dùng và thích ứng với các thay đổi trong môi trường công nghệ.

## **5. Mô hình xây và sửa có nhược điểm gì?**

-Mô hình Xây và Sửa là một phương pháp phát triển phần mềm đơn giản, trong đó phần mềm được xây dựng nhanh chóng mà không có kế hoạch cụ thể, sau đó được sửa đổi liên tục dựa trên phản hồi của người dùng. Tuy nhiên, mô hình này có nhiều nhược điểm, bao gồm:

#### 1. Thiếu kế hoạch và tài liệu

Không có kế hoạch phát triển chi tiết, dẫn đến việc khó theo dõi tiến độ và mở rộng sau này.

Thiếu tài liệu đầy đủ khiến việc bảo trì và nâng cấp trở nên khó khăn.

#### 2. Chất lượng phần mềm kém

Do không có quy trình thiết kế và kiểm thử chặt chẽ, phần mềm có thể chứa nhiều lỗi.

Dễ phát sinh các vấn đề về hiệu suất, bảo mật và tính ổn định.

#### 3. Khó bảo trì và mở rộng

Việc sửa lỗi liên tục mà không có cấu trúc rõ ràng khiến phần mềm trở nên rối rắm.

Rất khó để thêm tính năng mới mà không gây lỗi hoặc ảnh hưởng đến các phần khác của hệ thống.

#### 4. Tốn nhiều thời gian và chi phí về lâu dài

Ban đầu có thể phát triển nhanh, nhưng về sau việc sửa lỗi và cải tiến trở nên tốn kém.

Có thể phải viết lại toàn bộ hệ thống nếu phần mềm quá nhiều lỗi hoặc không đáp ứng được yêu cầu.

#### 5. Không phù hợp với các dự án lớn

Chỉ phù hợp với các dự án nhỏ, thử nghiệm hoặc phần mềm đơn giản.

Với các hệ thống lớn, việc thiếu kế hoạch và quy trình rõ ràng sẽ khiến dự án dễ thất bại.

#### Kết luận

Mô hình Xây và Sửa chỉ phù hợp với những dự án nhỏ, không quan trọng về chất lượng và không yêu cầu bảo trì lâu dài. Tuy nhiên, với phần mềm phức tạp, mô hình này không hiệu quả do thiếu kế hoạch, khó mở rộng và dễ phát sinh lỗi, dẫn đến chi phí bảo trì cao hơn về sau.

### 6. Mô hình bản mẫu nhanh là gì?

-Mô hình bản mẫu nhanh (Rapid Prototyping Model) là một phương pháp phát triển phần mềm trong đó một phiên bản thử nghiệm (prototype) của phần mềm được tạo ra nhanh chóng để thu thập phản hồi từ khách hàng, trước khi phát triển sản phẩm hoàn chỉnh.

### 7. Pha giải thể là gì?

**Pha giải thể** (Disposal Phase) là giai đoạn cuối cùng trong vòng đời phát triển phần mềm (SDLC). Đây là giai đoạn khi phần mềm không còn được sử dụng hoặc không đáp ứng được yêu cầu mới và cần được thay thế hoặc loại bỏ.

#### Các hoạt động trong pha giải thể:

**Đánh giá hệ thống:** Xác định lý do ngừng sử dụng phần mềm (lỗi thời, không còn phù hợp, tốn kém bảo trì...).

**Chuyển đổi dữ liệu:** Nếu có hệ thống mới thay thế, dữ liệu có thể được chuyển sang hệ thống mới.

**Gỡ bỏ phần mềm:** Xóa phần mềm khỏi hệ thống để tránh rủi ro bảo mật hoặc chi phí bảo trì không cần thiết.

**Tài liệu hóa:** Lưu trữ tài liệu, mã nguồn (nếu cần) hoặc cung cấp hướng dẫn cho việc ngừng sử dụng hệ thống.

Pha này quan trọng để đảm bảo quá trình ngừng sử dụng phần mềm diễn ra an toàn và không ảnh hưởng đến hoạt động của doanh nghiệp.

## **8.Mô hình xoắn ốc là gì?**

-Mô hình xoắn ốc (Spiral Model) là một mô hình phát triển phần mềm kết hợp giữa mô hình thác nước và mô hình lặp. Nó tập trung vào quản lý rủi ro bằng cách chia quá trình phát triển thành nhiều vòng lặp (iterations), mỗi vòng đại diện cho một chu kỳ phát triển phần mềm.

## **9.Tại sao mô hình tiến trình linh hoạt được đánh giá cao?**

-Mô hình tiến trình linh hoạt (Agile) được đánh giá cao vì nó thích nghi nhanh với sự thay đổi, tăng cường sự cộng tác, và đảm bảo chất lượng phần mềm thông qua các vòng lặp ngắn.

### **1. Thích nghi với sự thay đổi nhanh chóng**

Trong quá trình phát triển phần mềm, yêu cầu của khách hàng thường thay đổi liên tục. Agile cho phép điều chỉnh kế hoạch nhanh chóng thay vì tuân theo một lộ trình cố định như mô hình thác nước.

Mỗi vòng lặp (iteration) kéo dài khoảng 1-4 tuần, giúp nhóm phát triển phản hồi nhanh với các yêu cầu mới.

### **2. Cải thiện sự cộng tác giữa các bên liên quan**

Agile khuyến khích tương tác thường xuyên giữa nhóm phát triển và khách hàng.

Nhóm làm việc theo kiểu liên chức năng (cross-functional), nghĩa là lập trình viên, kiểm thử viên và quản lý dự án làm việc cùng nhau thay vì làm việc theo từng giai đoạn riêng biệt.

### **3. Phát hành sản phẩm nhanh hơn và liên tục**

Agile giúp tạo ra các sản phẩm có thể sử dụng sớm thay vì phải chờ đợi hoàn thành toàn bộ phần mềm.

Mỗi vòng lặp cung cấp một phiên bản phần mềm nhỏ nhưng có thể sử dụng, giúp kiểm tra tính khả thi và nhận phản hồi từ người dùng sớm hơn.

### **4. Cải thiện chất lượng phần mềm**

Agile sử dụng kiểm thử liên tục (Continuous Testing), đảm bảo lỗi được phát hiện sớm.

Mỗi lần phát triển nhỏ sẽ được kiểm tra kỹ trước khi chuyển sang bước tiếp theo.



## 5. Giảm rủi ro dự án thất bại

Do phần mềm được phát triển từng bước, nếu có vấn đề, nhóm có thể điều chỉnh ngay mà không phải đợi đến cuối dự án mới phát hiện.

Nếu một yêu cầu không khả thi hoặc không đáp ứng mong đợi, nhóm có thể thay đổi mà không làm ảnh hưởng lớn đến tiến độ tổng thể.

## 6. Tăng cường sự hài lòng của khách hàng

Khách hàng có thể thấy tiến độ thực tế sau từng vòng lặp và đề xuất thay đổi nếu cần.

Điều này giúp đảm bảo sản phẩm cuối cùng phù hợp với mong đợi của khách hàng.

## 10. Điểm khác biệt chính giữa mô hình mã nguồn mở và các mô hình khác là gì?

-Mô hình mã nguồn mở (Open Source) có một số điểm khác biệt quan trọng so với các mô hình phát triển phần mềm truyền thống như mô hình thác nước, mô hình xoắn ốc, mô hình Agile, v.v.

### 1. Quyền truy cập mã nguồn

Mã nguồn mở:

Mã nguồn được công khai, bất kỳ ai cũng có thể xem, chỉnh sửa, phân phối hoặc đóng góp vào dự án.

Ví dụ: Linux, Apache, MySQL, Firefox.

Mô hình truyền thống (đóng nguồn – proprietary):

Mã nguồn bị giữ kín, chỉ nhà phát triển hoặc công ty sở hữu có quyền sửa đổi.

Ví dụ: Windows, Microsoft Office, Adobe Photoshop.

### 2. Cộng đồng phát triển thay vì nhóm nội bộ

Mã nguồn mở:

Được phát triển bởi cộng đồng toàn cầu, không giới hạn trong một công ty hay tổ chức.

Các lập trình viên có thể đóng góp mã, sửa lỗi và cải thiện phần mềm.

Mô hình truyền thống:

Được phát triển bởi một nhóm cố định trong công ty.

Việc sửa lỗi, nâng cấp tính năng phụ thuộc vào công ty phát hành.

### 3. Quy trình phát triển linh hoạt hơn

Mã nguồn mở:

Phát triển theo mô hình phi tập trung, ai cũng có thể tham gia.

Quy trình phát triển thường theo kiểu Agile, phát hành liên tục khi có cải tiến mới.

Mô hình truyền thống:

Quy trình phát triển thường có kế hoạch cố định (thác nước, xoắn ốc,...).

Công ty phát triển và tung ra các bản cập nhật theo lịch trình.

### 4. Mô hình kinh doanh khác biệt

Mã nguồn mở:

Miễn phí hoặc có mô hình freemium (cung cấp miễn phí, kiếm tiền từ dịch vụ hỗ trợ, tùy chỉnh,...).

Ví dụ: Red Hat Linux kiếm tiền từ dịch vụ hỗ trợ doanh nghiệp.

Mô hình truyền thống:

Bán giấy phép bản quyền hoặc thu phí theo gói.

Ví dụ: Windows, Microsoft Office yêu cầu mua bản quyền.

### 5. Cách sửa lỗi và cập nhật phần mềm

Mã nguồn mở:

Khi có lỗi, bất kỳ ai cũng có thể sửa chữa và gửi bản vá (pull request).

Cập nhật có thể được thực hiện nhanh chóng bởi cộng đồng.

Mô hình truyền thống:

Người dùng phải chờ nhà phát triển chính thức sửa lỗi và tung bản cập nhật.

## PHẦN THẢO LUẬN NHÓM

### 1. So sánh ưu và nhược điểm của mô hình thác nước và mô hình xoắn ốc

Tiêu chí	Mô hình thác nước (Waterfall)	Mô hình xoắn ốc (Spiral)
Cấu trúc phát triển	Tuần tự từ đầu đến cuối, mỗi pha kết thúc trước khi bước sang pha tiếp theo.	Gồm nhiều vòng lặp, mỗi vòng thực hiện các giai đoạn phát triển, phân tích rủi ro.
Ưu điểm	<ul style="list-style-type: none"><li>- Dễ hiểu, dễ quản lý do có quy trình rõ ràng.</li><li>- Phù hợp với các dự án có yêu cầu ổn định.</li><li>- Dễ dàng lập kế hoạch ngân sách và tiến độ.</li></ul>	<ul style="list-style-type: none"><li>- Linh hoạt, thích ứng tốt với yêu cầu thay đổi.</li><li>- Giảm rủi ro nhờ phân tích và kiểm thử sớm.</li><li>- Có thể phát triển từng phần sản phẩm trước khi hoàn thiện toàn bộ.</li></ul>
Nhược điểm	<ul style="list-style-type: none"><li>- Khó thay đổi yêu cầu giữa chừng.</li><li>- Không phản hồi kịp thời từ khách hàng trong quá trình phát triển.</li><li>- Nếu có sai sót ở bước đầu, chi phí sửa đổi rất cao.</li></ul>	<ul style="list-style-type: none"><li>- Quản lý phức tạp, đòi hỏi tài nguyên và chuyên gia có kinh nghiệm.</li><li>- Tốn kém hơn do phải lặp đi lặp lại các bước phát triển.</li><li>- Không phù hợp với dự án nhỏ, ít rủi ro.</li></ul>
Ứng dụng	<ul style="list-style-type: none"><li>- Dự án có yêu cầu rõ ràng, ít thay đổi.</li><li>- Hệ thống phần mềm có độ ổn định cao (ví dụ: phần mềm doanh nghiệp, hệ thống tài chính).</li></ul>	<ul style="list-style-type: none"><li>- Dự án lớn, phức tạp, có nhiều rủi ro.</li><li>- Phần mềm cần phát triển dần theo yêu cầu khách hàng (ví dụ: hệ thống quân sự, phần mềm ngân hàng).</li></ul>

**Tóm lại:** Mô hình **thác nước** phù hợp với dự án có yêu cầu ổn định, trong khi **mô hình xoắn ốc** thích hợp cho dự án lớn, nhiều rủi ro và cần linh hoạt.

### 2. Tình huống thực tế có thể áp dụng mô hình lặp và tăng trưởng

Mô hình lặp và tăng trưởng (**Incremental and Iterative Model**) thường được áp dụng khi phần mềm có thể phát triển từng phần và cải thiện theo thời gian.

**Ví dụ thực tế:** Phát triển một ứng dụng thương mại điện tử

Một công ty khởi nghiệp muốn phát triển một nền tảng thương mại điện tử nhưng không có đủ ngân sách để xây dựng toàn bộ hệ thống ngay từ đầu. Họ có thể sử dụng mô hình lặp và tăng trưởng như sau:

**Phiên bản đầu tiên:** Chỉ có các tính năng cơ bản như đăng ký tài khoản, danh mục sản phẩm, giỏ hàng.

**Phiên bản thứ hai:** Thêm chức năng thanh toán online và theo dõi đơn hàng.

**Phiên bản thứ ba:** Cải thiện giao diện người dùng, tích hợp AI để gợi ý sản phẩm.

**Phiên bản tiếp theo:** Bổ sung tính năng chatbot hỗ trợ khách hàng và hệ thống đánh giá sản phẩm.

Lợi ích của mô hình này:

- Ra mắt sản phẩm sớm, có thể thu hút người dùng ngay từ đầu.
- Linh hoạt thay đổi theo phản hồi của khách hàng.
- Giảm rủi ro và chi phí phát triển do không cần hoàn thành toàn bộ phần mềm ngay từ đầu.

**Tóm lại:** Mô hình lặp và tăng trưởng rất phù hợp với các **startup**, phần mềm web, ứng dụng di động, nơi mà yêu cầu có thể thay đổi theo thời gian.

### 3. Tại sao mô hình xây và sửa không phù hợp với các dự án lớn?

Mô hình **xây và sửa (Build-and-Fix Model)** có cách tiếp cận đơn giản: phát triển nhanh một phiên bản, thử nghiệm, sửa lỗi và tiếp tục phát triển. Tuy nhiên, mô hình này **không phù hợp với dự án lớn** vì những lý do sau:

Nhược điểm khi áp dụng vào dự án lớn

-**Không có kế hoạch rõ ràng** → Dự án lớn cần quy trình chặt chẽ, trong khi mô hình xây và sửa chỉ tập trung vào sửa lỗi mà không có chiến lược phát triển tổng thể.

-**Tăng chi phí bảo trì** → Do không có tài liệu thiết kế chuẩn, mỗi lần sửa đổi có thể ảnh hưởng đến toàn bộ hệ thống, gây tốn kém.

-**Khó kiểm soát chất lượng** → Việc liên tục sửa lỗi và thay đổi làm phần mềm dễ phát sinh thêm lỗi mới, đặc biệt trong các hệ thống lớn và phức tạp.

-**Không tối ưu hiệu suất** → Không có giai đoạn thiết kế ban đầu, dễ dẫn đến phần mềm hoạt động kém hiệu quả, khó mở rộng khi cần.

Ví dụ thực tế

Nếu một công ty phát triển một **hệ thống ngân hàng**, nhưng không có thiết kế chặt chẽ mà chỉ xây dựng và sửa lỗi liên tục, hệ thống sẽ trở nên **khó bảo trì, dễ gặp lỗi nghiêm trọng**, ảnh hưởng đến dữ liệu và bảo mật.

Các phần mềm lớn như **hệ thống quản lý bệnh viện, điều khiển giao thông, hàng không** không thể sử dụng mô hình này vì rủi ro quá cao.

#### 4. So sánh giữa mô hình bản mẫu nhanh và mô hình tiến trình linh hoạt

Tiêu chí	Mô hình bản mẫu nhanh (Rapid Prototyping)	Mô hình tiến trình linh hoạt (Agile)
Mục tiêu chính	Xây dựng nguyên mẫu nhanh để thu thập phản hồi từ khách hàng.	Phát triển phần mềm theo từng đợt nhỏ, ưu tiên khả năng thích ứng với thay đổi.
Cách tiếp cận	Tạo ra một phiên bản thử nghiệm (prototype), sau đó cải tiến dựa trên phản hồi.	Chia nhỏ dự án thành các vòng lặp phát triển, mỗi vòng có thể cập nhật yêu cầu mới.
Tốc độ phát triển	Nhanh chóng tạo nguyên mẫu, nhưng cần sửa đổi nhiều để hoàn thiện sản phẩm.	Linh hoạt phát triển theo từng sprint (chu kỳ ngắn từ 1-4 tuần).
Mức độ linh hoạt	Tập trung vào phản hồi sớm, nhưng không có quy trình phát triển hoàn chỉnh.	Linh hoạt với thay đổi, có thể cập nhật yêu cầu ngay trong quá trình phát triển.
Ưu điểm	<ul style="list-style-type: none"><li>- Giúp khách hàng hình dung sản phẩm sớm.</li><li>- Phù hợp khi yêu cầu chưa rõ ràng.</li></ul>	<ul style="list-style-type: none"><li>- Khả năng thích ứng cao.</li><li>- Liên tục cải tiến và phản hồi nhanh.</li></ul>
Nhược điểm	<ul style="list-style-type: none"><li>- Nguyên mẫu có thể khác nhiều so với sản phẩm cuối cùng.</li><li>- Không tối ưu cho dự án lớn và phức tạp.</li></ul>	<ul style="list-style-type: none"><li>- Yêu cầu sự phối hợp tốt giữa các thành viên.</li><li>- Không phù hợp với dự án có yêu cầu cố định, ít thay đổi.</li></ul>

Tiêu chí	Mô hình bản mẫu nhanh (Rapid Prototyping)	Mô hình tiến trình linh hoạt (Agile)
Ứng dụng	- Thiết kế UI/UX, ứng dụng web, sản phẩm thử nghiệm.	- Phần mềm thương mại, dự án yêu cầu thay đổi liên tục (ví dụ: phần mềm quản lý, ứng dụng di động, hệ thống thương mại điện tử).

## 5. Phân tích vai trò của quản lý rủi ro trong mô hình xoắn ốc.

- Phát hiện và đánh giá rủi ro sớm: Trong mô hình xoắn ốc, mỗi vòng lặp đều có một giai đoạn dành riêng để nhận diện, phân tích và đánh giá các rủi ro có thể xảy ra trong quá trình phát triển. Điều này giúp đội ngũ phát triển phát hiện ra các vấn đề tiềm ẩn từ rất sớm, thay vì chỉ đối mặt với chúng khi dự án đã hoàn thành hoặc gần hoàn thành. Quản lý rủi ro giúp tạo ra các phương án dự phòng để ứng phó kịp thời.
- Điều chỉnh và cải tiến liên tục: Quản lý rủi ro giúp trong việc điều chỉnh chiến lược phát triển dự án sau mỗi vòng lặp. Nếu trong một vòng lặp xuất hiện rủi ro lớn, đội ngũ có thể điều chỉnh kế hoạch, phương pháp phát triển và tài nguyên để tránh những rủi ro đó hoặc giảm thiểu tác động của chúng trong các vòng tiếp theo. Điều này giúp giảm thiểu thiệt hại và duy trì tiến độ của dự án.
- Cung cấp quyết định thông minh hơn: Quản lý rủi ro giúp đội ngũ quản lý và các bên liên quan đưa ra các quyết định sáng suốt về cách thức thực hiện dự án. Các thông tin về các rủi ro tiềm ẩn giúp đưa ra những lựa chọn tốt hơn về công nghệ, nguồn lực và cách thức triển khai. Những quyết định này có thể giảm thiểu khả năng phát sinh sự cố nghiêm trọng trong tương lai.
- Tạo môi trường phát triển an toàn và hiệu quả: Mô hình xoắn ốc giúp dự án phát triển qua từng giai đoạn với các kiểm tra rủi ro rõ ràng, giúp tránh những thất bại tiềm tàng. Khi các rủi ro được đánh giá và xử lý ngay từ đầu, đội ngũ phát triển có thể làm việc trong một môi trường an toàn hơn, giảm bớt căng thẳng và sự không chắc chắn, từ đó nâng cao hiệu quả làm việc và tiến độ của dự án.
- Tối ưu hóa chi phí và tài nguyên: Bằng việc quản lý và đánh giá rủi ro, đội ngũ phát triển có thể tránh các chi phí phát sinh từ các sự cố không lường trước. Khi có chiến lược quản lý rủi ro rõ ràng, các nguồn lực có thể được phân bổ hiệu quả hơn, tránh việc phải dừng dự án hoặc làm lại một phần lớn công việc do không nhận diện và xử lý rủi ro kịp thời.

- Tăng cường cam kết của các bên liên quan: Một yếu tố quan trọng trong mô hình xoắn ốc là sự tham gia của các bên liên quan trong từng vòng lặp. Khi các bên liên quan được thông báo về các rủi ro và phương án xử lý, họ sẽ cảm thấy yên tâm hơn về tiến độ và kết quả của dự án. Điều này giúp tăng cường sự cam kết và hỗ trợ từ các bên liên quan, từ đó giảm thiểu khả năng gặp phải các vấn đề không mong muốn.

- Ứng phó với thay đổi và yêu cầu mới: Trong suốt quá trình phát triển dự án, yêu cầu có thể thay đổi hoặc thêm vào những yếu tố mới. Quản lý rủi ro giúp điều chỉnh dự án khi có những thay đổi không lường trước được, ví dụ như thay đổi yêu cầu của khách hàng, thay đổi công nghệ, hoặc sự thay đổi trong thị trường. Việc đánh giá và quản lý các rủi ro này sẽ giúp dự án tiếp tục phát triển dù có sự thay đổi.

## **6. Khi nào nên sử dụng mô hình thác nước thay vì mô hình tiến trình linh hoạt?**

Mô hình thác nước (Waterfall) phù hợp khi:

- Yêu cầu rõ ràng và ổn định: Các yêu cầu không thay đổi nhiều trong suốt quá trình phát triển.
- Dự án quy mô nhỏ, thời gian ngắn: Phù hợp với các dự án đơn giản, có phạm vi hẹp.
- Cần quy trình chặt chẽ: Dễ dàng quản lý và kiểm soát từng giai đoạn.
- Công nghệ ổn định: Khi sử dụng công nghệ đã thử nghiệm và ít thay đổi.
- Ngân sách và tài nguyên hạn chế: Không có sự thay đổi liên tục, giảm thiểu chi phí.
- Khách hàng ít tham gia: Không yêu cầu sự tham gia thường xuyên từ khách hàng.
- Kiểm soát tiến độ chặt chẽ: Dễ dàng xác định mốc thời gian và tiến độ hoàn thành.
- Mô hình thác nước thích hợp khi dự án ổn định, ít thay đổi và cần kiểm soát nghiêm ngặt.

## **7. Thảo luận về những khó khăn khi áp dụng mô hình mã nguồn mở.**

- Vấn đề về bảo mật: Mã nguồn mở có thể dễ dàng bị khai thác bởi các bên thứ ba, gây ra nguy cơ bảo mật. Mặc dù cộng đồng có thể giúp phát hiện và sửa lỗi nhanh chóng, nhưng việc không kiểm soát được toàn bộ mã nguồn có thể dẫn đến lỗ hổng bảo mật.
- Thiếu hỗ trợ chính thức: Nhiều dự án mã nguồn mở thiếu sự hỗ trợ chính thức từ nhà phát triển, khiến người sử dụng phải dựa vào cộng đồng để giải quyết vấn đề. Điều này có thể gây khó khăn khi cần giải quyết sự cố nhanh chóng hoặc yêu cầu hỗ trợ kỹ thuật.

- Vấn đề về tính tương thích và tích hợp: Việc tích hợp mã nguồn mở vào các hệ thống hiện tại có thể gặp khó khăn về tính tương thích với phần mềm và công nghệ hiện có. Điều này có thể yêu cầu thay đổi cấu trúc hệ thống hoặc phát triển thêm các phần mềm trung gian.
- Quản lý dự án và cập nhật: Mã nguồn mở thường có nhiều người đóng góp và cập nhật thường xuyên. Điều này có thể tạo ra sự bất ổn khi một phiên bản mới có thể thay đổi quá nhiều hoặc không tương thích với các phần mềm khác, gây khó khăn trong việc quản lý các phiên bản và cập nhật.
- Thiếu tài liệu và hướng dẫn: Mặc dù nhiều dự án mã nguồn mở có cộng đồng mạnh mẽ, nhưng tài liệu và hướng dẫn sử dụng có thể không đầy đủ hoặc thiếu chi tiết. Điều này có thể gây khó khăn cho những người mới bắt đầu hoặc không có kinh nghiệm kỹ thuật.
- Khó khăn trong việc đảm bảo chất lượng: Mã nguồn mở đôi khi thiếu sự kiểm soát chất lượng nghiêm ngặt, do đó sản phẩm cuối cùng có thể không đạt tiêu chuẩn chất lượng cao như các phần mềm thương mại.
- Khó duy trì và cập nhật: Dự án mã nguồn mở có thể bị ngừng phát triển nếu không còn người đóng góp hoặc thiếu sự quan tâm từ cộng đồng. Điều này có thể tạo ra rủi ro cho việc duy trì phần mềm lâu dài.

## **8. Phân tích cách mô hình tiến trình linh hoạt giúp cải thiện chất lượng phần mềm.**

- Phát triển liên tục và lặp lại: Trong mô hình Agile, phần mềm được phát triển qua các chu kỳ lặp (sprints), mỗi chu kỳ thường kéo dài từ 1 đến 4 tuần. Điều này giúp kiểm tra và cải tiến phần mềm liên tục. Mỗi vòng lặp tạo ra một phiên bản hoàn chỉnh của phần mềm, cho phép phát hiện lỗi và vấn đề sớm, từ đó cải thiện chất lượng.
- Phản hồi nhanh từ khách hàng: Agile khuyến khích sự tham gia của khách hàng trong suốt quá trình phát triển. Mỗi phiên bản phần mềm được cung cấp cho khách hàng sau mỗi chu kỳ phát triển, giúp nhận được phản hồi trực tiếp để điều chỉnh và cải tiến sản phẩm nhanh chóng, đảm bảo đáp ứng đúng nhu cầu và chất lượng.
- Kiểm tra và đảm bảo chất lượng liên tục: Agile khuyến khích kiểm tra phần mềm thường xuyên, với các bài kiểm tra tự động được tích hợp vào quy trình phát triển. Điều này giúp phát hiện lỗi sớm và giảm thiểu rủi ro, nâng cao chất lượng phần mềm qua mỗi vòng phát triển.
- Linh hoạt và điều chỉnh dễ dàng: Với phương pháp Agile, phần mềm có thể dễ dàng được điều chỉnh và cải thiện dựa trên các yêu cầu thay đổi của khách hàng hoặc phản hồi từ đội ngũ phát triển. Điều này giúp đảm bảo rằng sản phẩm cuối cùng sẽ đáp ứng được các tiêu chuẩn và yêu cầu chất lượng mà khách hàng mong muốn.



- Phát triển hợp tác và giao tiếp liên tục: Agile khuyến khích sự hợp tác chặt chẽ giữa các thành viên trong nhóm và với khách hàng. Việc này giúp giải quyết các vấn đề nhanh chóng, đảm bảo chất lượng và giúp đội ngũ phát triển hiểu rõ yêu cầu và mong muốn của khách hàng, tránh sai sót và cải thiện sản phẩm cuối cùng.

- Tập trung vào các tính năng quan trọng: Agile giúp nhóm phát triển tập trung vào những tính năng có giá trị cao nhất, ưu tiên hoàn thiện và kiểm tra các tính năng quan trọng, thay vì cố gắng phát triển tất cả các tính năng một cách đồng thời. Điều này đảm bảo rằng sản phẩm hoàn thành sẽ có chất lượng cao và phù hợp với yêu cầu của người dùng.

- Khả năng phát triển và cải tiến không ngừng: Agile tạo ra một môi trường phát triển nơi việc cải tiến liên tục được khuyến khích. Sau mỗi chu kỳ, nhóm phát triển sẽ tổ chức các cuộc họp "retrospective" để đánh giá những gì làm tốt và những gì cần cải thiện, từ đó liên tục nâng cao chất lượng quy trình và sản phẩm.

## **9. Thảo luận về vai trò của pha bảo trì trong vòng đời phát triển phần mềm.**

Pha bảo trì là giai đoạn cuối cùng trong vòng đời phát triển phần mềm (SDLC - Software Development Life Cycle) và có vai trò quan trọng trong việc đảm bảo phần mềm tiếp tục hoạt động hiệu quả sau khi triển khai. Vai trò chính của pha bảo trì bao gồm:

Sửa lỗi (Corrective Maintenance)

Khắc phục các lỗi phần mềm phát sinh sau khi triển khai (bug, lỗi bảo mật, lỗi logic).

Cải tiến và tối ưu hóa (Perfective Maintenance)

Cải thiện hiệu suất, tối ưu mã nguồn, nâng cấp giao diện người dùng.

Thích nghi với thay đổi (Adaptive Maintenance)

Cập nhật phần mềm để phù hợp với môi trường mới như hệ điều hành, phần cứng, hoặc tích hợp với hệ thống khác.

Bảo mật và tuân thủ (Preventive Maintenance)

Cập nhật các bản vá bảo mật, đảm bảo tuân thủ quy định pháp lý và tiêu chuẩn công nghiệp.

Hỗ trợ người dùng (User Support & Training)

Đào tạo người dùng, cung cấp tài liệu hướng dẫn, hỗ trợ kỹ thuật.

Pha bảo trì chiếm một phần lớn chi phí và thời gian trong vòng đời phần mềm, thường kéo dài trong nhiều năm sau khi triển khai. Việc quản lý bảo trì hiệu quả giúp phần mềm hoạt động ổn định, giảm rủi ro và tăng tuổi thọ sản phẩm.

## 10. Đề xuất mô hình vòng đời phù hợp cho dự án phát triển phần mềm ngân hàng và giải thích lý do.

Mô hình phù hợp: Mô hình V hoặc Mô hình Spiral (xoắn ốc)

Lý do chọn mô hình V

Mô hình V (Verification and Validation) là phiên bản cải tiến của mô hình thác nước, nhấn mạnh vào kiểm thử ở từng giai đoạn. Mô hình này phù hợp với phần mềm ngân hàng vì:

Tính ổn định cao: Ngân hàng yêu cầu hệ thống đáng tin cậy, ít lỗi.

Quy trình kiểm thử chặt chẽ: Mô hình V đảm bảo mỗi giai đoạn đều có kiểm thử đi kèm, giúp phát hiện lỗi sớm.

Tuân thủ quy định: Phần mềm ngân hàng cần đáp ứng nhiều tiêu chuẩn bảo mật và pháp lý, mô hình V giúp kiểm soát tốt các yêu cầu này.

Lý do chọn mô hình Spiral

Mô hình Spiral (xoắn ốc) kết hợp yếu tố lặp của mô hình Agile và kiểm soát rủi ro của mô hình V, phù hợp với:

Dự án phức tạp, có rủi ro cao: Ngành ngân hàng yêu cầu bảo mật cao, mô hình Spiral giúp đánh giá rủi ro liên tục.

Cần thích nghi với thay đổi: Nếu ngân hàng cần cập nhật tính năng thường xuyên, mô hình Spiral giúp linh hoạt hơn so với mô hình V.

Ưu tiên kiểm thử và bảo mật: Spiral cho phép kiểm thử và đánh giá rủi ro trong mỗi vòng lặp.

So sánh và đề xuất

Tiêu chí	Mô hình V	Mô hình Spiral
Tính ổn định	Cao	Trung bình
Kiểm soát rủi ro	Trung bình	Cao
Khả năng thích nghi với thay đổi	Thấp	Cao
Kiểm thử	Nghiêm ngặt ở từng pha	Liên tục trong mỗi vòng lặp

Tiêu chí	Mô hình V	Mô hình Spiral
Chi phí	Thấp hơn	Cao hơn

Nếu ngân hàng yêu cầu hệ thống ổn định, ít thay đổi → Chọn mô hình V.  
 Nếu ngân hàng cần cải tiến thường xuyên, có nhiều rủi ro bảo mật → Chọn mô hình Spiral.

## PHẦN 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?**

Mô hình thác nước (Waterfall) có đặc điểm là từng pha phải hoàn thành trước khi chuyển sang pha tiếp theo, nên thay đổi sau khi pha thiết kế đã hoàn thành có thể gây gián đoạn lớn. Đội phát triển có thể xử lý tình huống này bằng cách:

### **Đánh giá mức độ ảnh hưởng của thay đổi**

Xác định xem thay đổi có ảnh hưởng đến kiến trúc tổng thể, cơ sở dữ liệu, hoặc các thành phần quan trọng khác hay không.

Ước lượng thời gian, chi phí và rủi ro khi thực hiện thay đổi.

### **Trao đổi với khách hàng**

Nếu thay đổi nhỏ và có thể tích hợp dễ dàng, có thể thực hiện mà không ảnh hưởng nhiều đến tiến độ.

Nếu thay đổi lớn, cần giải thích với khách hàng về ảnh hưởng đến thời gian và chi phí, đồng thời đề xuất các phương án xử lý (ví dụ: thực hiện trong giai đoạn bảo trì hoặc phiên bản sau).

### **Xem xét hợp đồng và quy trình quản lý thay đổi**

Kiểm tra điều khoản về thay đổi trong hợp đồng (Change Management).

Nếu hợp đồng không hỗ trợ thay đổi giữa chừng, có thể thương lượng với khách hàng để bổ sung chi phí và thời gian.

### **Thực hiện thay đổi theo quy trình kiểm soát**

Nếu quyết định chấp nhận thay đổi, cần cập nhật tài liệu thiết kế, điều chỉnh kế hoạch dự án và thông báo cho các bên liên quan.

Nếu từ chối thay đổi, cần có giải thích rõ ràng và đề xuất giải pháp thay thế.

**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?**

Mô hình lặp và tăng trưởng liên tục (Iterative and Incremental) cho phép phát triển phần mềm theo từng giai đoạn nhỏ, giúp dễ dàng điều chỉnh khi gặp vấn đề. Khi dự án bị trễ tiến độ do thiếu nhân lực, người quản lý dự án có thể:

## **Đánh giá lại kế hoạch và ưu tiên công việc**

Xác định các tính năng quan trọng cần hoàn thành trước.

Điều chỉnh backlog và phân bổ lại công việc theo mức độ ưu tiên.

## **Tăng cường nhân lực**

Nếu có thể, tuyển thêm nhân sự hoặc mượn người từ nhóm khác trong công ty.

Nếu không thể tuyển, xem xét thuê ngoài (outsourcing) cho các phần ít quan trọng.

## **Tối ưu hóa năng suất nhóm hiện tại**

Tăng cường cộng tác giữa các thành viên, giảm thời gian chờ đợi hoặc phụ thuộc lẫn nhau.

Áp dụng kỹ thuật phát triển nhanh hơn như **pair programming** hoặc **code review hiệu quả**.

## **Thương lượng với khách hàng về thời gian và phạm vi dự án**

Nếu không thể bổ sung nhân lực, có thể thảo luận với khách hàng để kéo dài thời gian hoặc giảm bớt phạm vi công việc trong giai đoạn hiện tại.

## **Tận dụng tự động hóa**

Áp dụng CI/CD để giảm thời gian triển khai và kiểm thử.

Sử dụng các công cụ hỗ trợ phát triển giúp giảm khối lượng công việc thủ công.

Bằng cách kết hợp các giải pháp trên, dự án có thể giảm thiểu tác động của việc thiếu nhân lực mà vẫn đảm bảo tiến độ.

## **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 liên hệ và nhắc nhở khách hàng phản hồi kịp thời.

Trao đổi với khách hàng với tầm quan trọng của việc phản hồi, các hậu quả có thể xảy ra khi phản hồi chậm hoặc không phản hồi.

Thường xuyên có buổi gặp mặt khách hàng để cho khách hàng thấy sản phẩm đang phát triển ở giai đoạn nào, giải quyết các thắc mắc và đưa ra các gợi ý khuyến khích họ đưa ra các phản hồi.

Lập kế hoạch dành đủ thời gian để khách hàng có thể phản hồi.

Có kế hoạch dự phòng rõ ràng khi không nhận được phản hồi.

**4. Khách hàng yêu cầu bổ sung một số tính năng mới khi phần mềm đã bước vào pha cài đặt. Nên áp dụng mô hình nào để xử lý tốt nhất yêu cầu này?**

Để xử lý tốt yêu cầu này thì nên áp dụng mô hình tiến trình linh hoạt (Agile Processes). Vì: Mô hình này thích nghi tốt với các thay đổi và nâng cao chất lượng phần mềm.

Vì dự án được chia nhỏ thành các chu kỳ và phần mềm sẽ được phát triển theo từng phần rõ ràng. Nên khi có một yêu cầu mới từ khách hàng thì nhóm phát triển sẽ thêm vào danh sách thực hiện và sẽ phát triển nó trong một chu kỳ tiếp theo mà không ảnh hưởng đến các chức năng đã được phát triển.

Mô hình này cần sự hợp tác cao nên cần phản hồi nhanh từ khách hàng để đảm bảo các yêu cầu được thực hiện chính xác.

**5. Một công ty nhỏ muốn áp dụng mô hình bản mẫu nhanh nhưng gặp khó khăn do thiếu nguồn lực. Hãy đề xuất giải pháp.**

Chỉ nên tạo ra các bản mẫu tập trung vào các tính năng cốt lõi và các chức năng quan trọng mà khách hàng yêu cầu hoặc muốn thử nghiệm.

Sử dụng các phần mềm tạo bản mẫu nhanh để giảm thời gian và công sức lập trình.

Tái sử dụng lại các bản mẫu có sẵn và chỉnh sửa trên đó.

Thường xuyên kiểm tra để phát hiện lỗi và nhận được phản hồi sớm để đảm bảo quy trình luôn đi đúng kế hoạch và thỏa mãn yêu cầu khách hàng.

**6. Trong dự án phần mềm thương mại điện tử, khách hàng liên tục yêu cầu thay đổi giao diện. Mô hình nào sẽ phù hợp nhất?**

Nên sử dụng mô hình tiến trình linh hoạt, vì:

Mô hình này có khả năng thay đổi linh hoạt nên sẽ thích nghi tốt với môi trường thay đổi yêu cầu liên tục.

Mô hình này cần phản hồi nhanh từ khách hàng nên sẽ luôn đáp ứng kịp thời các yêu cầu của khách hàng mà không ảnh hưởng đến các công việc khác.

Mô hình chia nhỏ dự án thành các phần nhỏ nên cho phép phát triển và cải tiến theo từng phần. Thực hiện từng bước nhỏ nên trong quá trình phát triển không cần phải thay đổi toàn bộ giao diện từ đầu.

Các mô hình khác thì không phù hợp. VD: mô hình Waterfall là mô hình tuyến tính nên phải thực hiện tuần tự các giai đoạn, nên khi có yêu cầu thay đổi thì sẽ ảnh hưởng đến toàn bộ quá trình phát triển khiến cho việc thay đổi rất khó khăn và tốn kém.

### **Tình huống 7:**

-Với một dự án lớn, có nhiều nhóm phát triển phân tán ở các quốc gia khác nhau, mô hình phù hợp nhất là:

#### **1. Mô hình Agile kết hợp Scrum hoặc SAFe (Scaled Agile Framework)**

- Linh hoạt, thích nghi nhanh với yêu cầu thay đổi.
- Chia dự án thành nhiều phần nhỏ, mỗi nhóm có thể làm việc độc lập.
- Giao tiếp thường xuyên qua các cuộc họp Scrum hoặc SAFe PI Planning.
- Phân phối công việc theo Sprint (chu kỳ ngắn), giúp kiểm soát tiến độ hiệu quả.

*Phù hợp cho: Dự án có yêu cầu thay đổi nhanh, cần nhiều nhóm làm việc đồng thời.*

#### **2. Mô hình Phát triển Phần mềm Toàn cầu (Global Software Development - GSD)**

- Hỗ trợ làm việc từ xa với nhiều nhóm ở các múi giờ khác nhau.
- Phân công công việc theo chuyên môn của từng nhóm, tối ưu hóa nguồn lực.
- Sử dụng công cụ quản lý dự án như Jira, Trello, GitLab để theo dõi tiến độ.
- Giao tiếp hiệu quả qua email, Slack, Microsoft Teams để tránh hiểu lầm.

*Phù hợp cho: Dự án có nhiều nhóm phát triển ở các nước khác nhau.*

#### **3. Mô hình V (V-Model - Verification and Validation)**

- Kiểm thử song song với phát triển, giảm lỗi trong dự án lớn.
- Mỗi giai đoạn có kiểm thử tương ứng, giúp kiểm soát chất lượng.
- Phù hợp cho dự án quan trọng, yêu cầu chất lượng cao như phần mềm tài chính, y tế.

*Phù hợp cho: Dự án lớn, yêu cầu kiểm thử chặt chẽ, có tiêu chuẩn nghiêm ngặt.*

**Kết luận:**

- Nếu dự án cần linh hoạt, phản hồi nhanh → Agile/Scrum hoặc SAFe
- Nếu dự án có nhiều nhóm ở các quốc gia khác nhau → GSD (Global Software Development)
- Nếu dự án yêu cầu kiểm thử chặt chẽ, chất lượng cao → Mô hình V

Gợi ý tốt nhất: Kết hợp Agile/Scrum với mô hình GSD để tận dụng sự linh hoạt và khả năng làm việc từ xa của các nhóm quốc tế.

## **Tình huống 8:**

### **1. Xác định và đánh giá rủi ro ngay từ đầu**

Cách thực hiện:

- Thực hiện phân tích rủi ro ngay từ giai đoạn đầu của mỗi vòng xoắn.
- Sử dụng các phương pháp như FMEA (Failure Mode and Effects Analysis) hoặc SWOT để đánh giá các nguy cơ có thể xảy ra.
- Xây dựng danh sách rủi ro và phân loại theo mức độ nghiêm trọng.

Lợi ích:

- Giúp nhận diện sớm các vấn đề tiềm ẩn, tránh phát sinh chi phí sửa lỗi lớn về sau.

### **2. Chia dự án thành các vòng lặp nhỏ, kiểm soát từng giai đoạn**

Cách thực hiện:

- Áp dụng nguyên tắc phát triển lặp (Iterative Development) để triển khai phần mềm từng phần.
- Ở mỗi vòng xoắn, tập trung vào các rủi ro lớn nhất trước, sau đó kiểm tra và điều chỉnh.

Lợi ích:

- Dễ dàng theo dõi tiến độ và điều chỉnh kế hoạch khi cần thiết.
- Giảm thiểu thiệt hại nếu có lỗi hoặc thay đổi trong yêu cầu.

### **3. Kiểm tra và xác thực sản phẩm liên tục**

Cách thực hiện:

- Áp dụng kiểm thử sớm (Early Testing) ngay từ giai đoạn thiết kế.
- Sử dụng mô hình Prototyping (tạo bản mẫu thử nghiệm) để xác thực yêu cầu của khách hàng.
- Áp dụng các phương pháp kiểm thử tự động, kiểm thử đơn vị, kiểm thử tích hợp để phát hiện lỗi sớm.

Lợi ích:

Giúp giảm rủi ro về chất lượng sản phẩm, tránh sửa lỗi tốn kém ở giai đoạn cuối.

### **4. Cải thiện giao tiếp và quản lý yêu cầu khách hàng**

Cách thực hiện:

Thường xuyên trao đổi với khách hàng để cập nhật yêu cầu.



Sử dụng các công cụ quản lý dự án như Jira, Trello, Microsoft Teams để theo dõi tiến độ.  
Viết tài liệu chi tiết về yêu cầu và cập nhật liên tục.

Lợi ích:

Tránh hiểu sai yêu cầu, hạn chế rủi ro phải chỉnh sửa sản phẩm lớn về sau.

## 5. Quản lý ngân sách và tài nguyên hợp lý

Cách thực hiện:

- Xây dựng kế hoạch tài chính linh hoạt, đảm bảo có ngân sách dự phòng cho rủi ro.
- Lập kế hoạch phân bổ nguồn lực hợp lý để tránh thiếu hụt nhân lực hoặc quá tải.

Lợi ích:

- Giúp dự án không bị đình trệ do thiếu kinh phí hoặc nhân lực.

## 6. Áp dụng các tiêu chuẩn bảo mật và quản lý an toàn dữ liệu

Cách thực hiện:

- Sử dụng các tiêu chuẩn bảo mật như OWASP, ISO 27001 để bảo vệ phần mềm khỏi rủi ro an ninh.
- Thực hiện đánh giá bảo mật định kỳ để ngăn chặn lỗ hổng từ sớm.

Lợi ích:

- Giảm nguy cơ bị tấn công mạng hoặc mất dữ liệu quan trọng.

Kết luận

- Nhận diện rủi ro sớm bằng các phương pháp đánh giá chuyên sâu.
- Chia nhỏ dự án thành các vòng lặp để kiểm soát tốt hơn.
- Kiểm thử và xác thực sản phẩm liên tục để đảm bảo chất lượng.
- Giao tiếp chặt chẽ với khách hàng để tránh thay đổi lớn về sau.
- Quản lý ngân sách và tài nguyên hợp lý để tránh gián đoạn.

## Tình huống 9:

-Phần mềm ngân hàng đòi hỏi bảo mật cao, kiểm thử chặt chẽ, và tuân thủ các quy định nghiêm ngặt. Do đó, mô hình phù hợp nhất là:

### 1. Mô hình V (V-Model - Verification and Validation)

Lý do lựa chọn:

- Đảm bảo kiểm thử chặt chẽ từ giai đoạn thiết kế đến triển khai.

- Mỗi giai đoạn phát triển có kiểm thử tương ứng, giúp phát hiện lỗi sớm.
- Phù hợp với phần mềm ngân hàng, nơi bảo mật và độ chính xác là ưu tiên hàng đầu.

Lợi ích:

- Giảm thiểu rủi ro bảo mật do kiểm thử song song với phát triển.
- Phù hợp với các chuẩn bảo mật như ISO 27001, PCI DSS (bảo mật thẻ thanh toán).

## 2. Mô hình Xoắn ốc (Spiral Model)

Lý do lựa chọn:

- Tập trung vào phân tích rủi ro ngay từ đầu, giúp giảm thiểu lỗ hổng bảo mật.
- Kết hợp phát triển lặp với kiểm thử nghiêm ngặt, giảm thiểu lỗi hệ thống.
- Cho phép đánh giá rủi ro bảo mật qua từng vòng xoắn.

Lợi ích:

- Giúp dự án thích nghi với thay đổi nhưng vẫn kiểm soát tốt bảo mật.
- Phù hợp với hệ thống ngân hàng cần đánh giá mối đe dọa an ninh liên tục.

## 3. Mô hình Agile kết hợp DevSecOps

Lý do lựa chọn:

- DevSecOps tích hợp bảo mật ngay từ đầu vào quy trình phát triển.
- Agile giúp phản ứng nhanh với thay đổi về quy định và yêu cầu bảo mật.
- Kiểm thử bảo mật tự động bằng Static Code Analysis (SCA), Dynamic Application Security Testing (DAST).

Lợi ích:

- Giảm thiểu lỗi bảo mật ngay từ giai đoạn đầu.
- Phù hợp với hệ thống ngân hàng yêu cầu cập nhật nhanh nhưng vẫn an toàn.

Kết luận

- Nếu cần kiểm thử chặt chẽ, hạn chế rủi ro bảo mật → Mô hình V
- Nếu muốn quản lý rủi ro bảo mật ngay từ đầu → Mô hình Xoắn ốc
- Nếu cần phát triển nhanh nhưng vẫn đảm bảo bảo mật → Agile + DevSecOps

Gợi ý tốt nhất: Kết hợp Mô hình V/Xoắn ốc với DevSecOps để vừa đảm bảo bảo mật cao vừa có khả năng thích nghi linh hoạt!

**Tình huống 10:**

-Pha giải thể (Retirement/Decommissioning Phase) đánh dấu sự kết thúc của vòng đời phần mềm khi phần mềm không còn phù hợp hoặc có hiệu suất kém. Việc quyết định kết thúc vòng đời phần mềm thường dựa trên các yếu tố sau:

### 1. Công nghệ lỗi thời, không còn hỗ trợ

Dấu hiệu:

- Phần mềm dựa trên công nghệ cũ, không còn được nhà cung cấp hỗ trợ (ví dụ: Windows XP, Python 2).
- Không thể cập nhật bảo mật, gây rủi ro cao.
- Không tương thích với phần cứng hoặc hệ điều hành mới.

Ví dụ: Ngừng hỗ trợ Adobe Flash Player vào năm 2020 do công nghệ lỗi thời.

### 2. Không còn đáp ứng nhu cầu kinh doanh

Dấu hiệu:

- Quy trình kinh doanh thay đổi, phần mềm không còn phù hợp.
- Xuất hiện giải pháp thay thế hiệu quả hơn, chi phí vận hành thấp hơn.
- Người dùng dần chuyển sang nền tảng khác (ví dụ: từ phần mềm desktop sang ứng dụng web).

Ví dụ: Ngân hàng chuyển từ hệ thống Core Banking cũ sang giải pháp đám mây để tối ưu vận hành.

### 3. Chi phí duy trì quá cao

Dấu hiệu:

- Chi phí bảo trì, cập nhật quá lớn so với lợi ích mang lại.
- Hệ thống gặp nhiều lỗi nghiêm trọng, sửa chữa tốn kém hơn phát triển hệ thống mới.
- Đội ngũ kỹ thuật thiếu chuyên gia có kinh nghiệm với công nghệ cũ.

Ví dụ:

Nhiều doanh nghiệp ngừng sử dụng hệ thống SAP on-premise và chuyển sang ERP trên đám mây.

### 4. Lỗi hỏng bảo mật nghiêm trọng, không thể khắc phục

Dấu hiệu:

- Phần mềm bị tấn công thường xuyên do không còn bản vá bảo mật.
- Rủi ro mất dữ liệu người dùng cao.
- Không đáp ứng tiêu chuẩn bảo mật mới như GDPR, PCI DSS.

Ví dụ:

Một ứng dụng ngân hàng cũ bị phát hiện lỗ hổng bảo mật nghiêm trọng, ngân hàng quyết định thay thế bằng ứng dụng mới.

5. Không còn người dùng hoặc khách hàng chuyển sang giải pháp khác

Dấu hiệu:

- Lượng người dùng giảm mạnh, không còn nhu cầu sử dụng.
- Đối thủ cạnh tranh cung cấp dịch vụ tốt hơn, khách hàng dần rời bỏ.
- Doanh thu từ phần mềm không còn đủ để duy trì.

Ví dụ: Yahoo Messenger bị khai tử do sự cạnh tranh từ Facebook Messenger và WhatsApp.

Quy trình thực hiện pha giải thể:

- 1 Thông báo cho người dùng về kế hoạch ngừng hỗ trợ.
- 2 Chuyển dữ liệu quan trọng sang hệ thống mới (nếu có).
- 3 Vô hiệu hóa phần mềm và gỡ bỏ các dịch vụ liên quan.
- 4 Cung cấp hướng dẫn thay thế hoặc hỗ trợ người dùng chuyển đổi.
- 5 Xóa mã nguồn, tài liệu hoặc đưa vào kho lưu trữ nếu cần thiết.

Kết luận:

- Nên kết thúc vòng đời phần mềm và thực hiện pha giải thể khi:
- Công nghệ lỗi thời, không còn hỗ trợ.
- Phần mềm không đáp ứng nhu cầu kinh doanh.
- Chi phí bảo trì quá cao.
- Xuất hiện lỗ hổng bảo mật nghiêm trọng.
- Không còn người dùng hoặc thị trường biến mất.

Lời khuyên: Việc lên kế hoạch ngừng hoạt động có lộ trình rõ ràng giúp tránh gián đoạn cho người dùng và doanh nghiệp!

