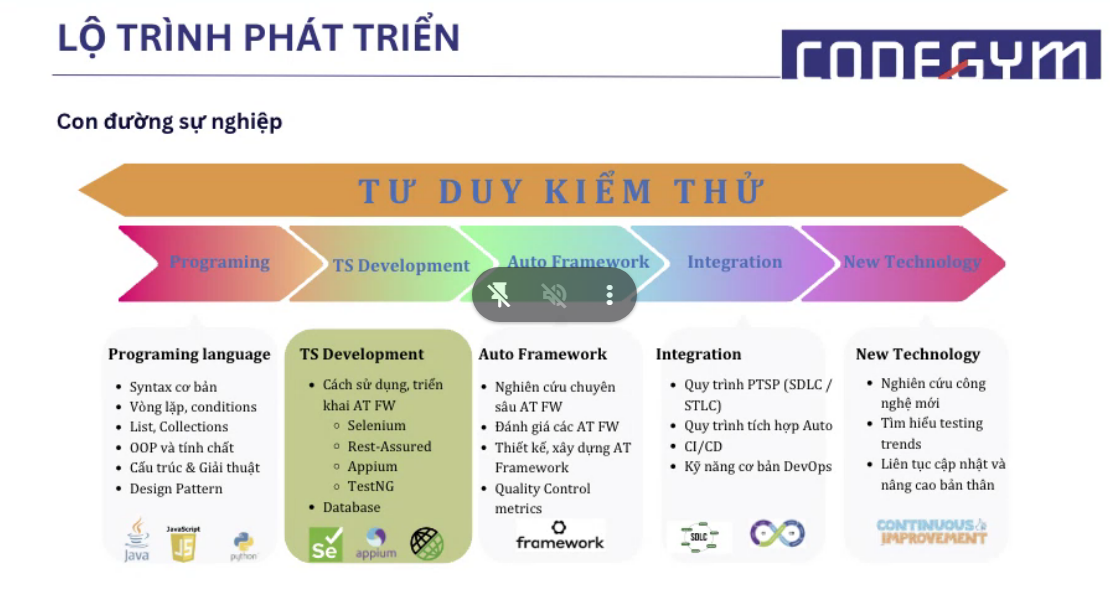
Trong lĩnh vực phần mềm và phần cứng, kiểm thử là một lĩnh vực rộng lớn và là điều hết sức cần thiết trước khi bất kỳ sản phẩm hoặc phần mềm nào được tung ra thị trường. Kiểm thử phần mềm đo lường chất lượng, tính chính xác và tính đầy đủ của phần mềm được phát triển. Kiểm thử phần mềm bao gồm nhiều lĩnh vực khác nhau, nhưng có thể chia thành hai loại - kiểm thử thủ công và kiểm thử tự động. Kiểm thử thủ công đóng một vai trò không thể thiếu trong lĩnh vực phát triển phần mềm vì kiểm thử tự động hóa không thể áp dụng cho mọi tình huống.

Bây giờ chúng ta hãy xem một số câu hỏi phỏng vấn Kiểm tra thủ công thường gặp được phân loại trong các phần sau:

* [Câu hỏi phỏng vấn kiểm tra thủ công dành cho người mới](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions-Freshers/tree/main/README.md#manual-testing-interview-questions-freshers)
* [Câu hỏi phỏng vấn kiểm tra thủ công dành cho người có kinh nghiệm](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions-Freshers/tree/main/README.md#manual-testing-interview-questions-for-experienced)
* [Câu hỏi phỏng vấn thời gian thực về kiểm tra thủ công](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions-Freshers/tree/main#real-time-interview-questions-on-manual-testing--scenario-based)



[**Manual-Test-Phỏng vấn-Câu hỏi-Người mới**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#manual-testing-interview-questions-freshers)

[**1. Kiểm tra thủ công là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#1-what-is-manual-testing)

Trong kiểm thử thủ công, người kiểm thử sẽ xác minh chức năng của phần mềm theo cách thủ công. Người thử nghiệm có một danh sách đầy đủ tất cả các trường hợp thử nghiệm thủ công mà họ nên thử nghiệm, cùng với dữ liệu thử nghiệm. Họ xem xét từng trường hợp một. Họ khởi chạy phần mềm với tư cách là người dùng cuối, nhập thông tin đầu vào và xác minh đầu ra theo cách thủ công.

Có vẻ như kiểm thử thủ công không hiệu quả khi so sánh với kiểm thử tự động. Nó chậm, không thể lặp lại một cách nhất quán và dễ bị con người đánh giá sai.

Tuy nhiên, kiểm tra thủ công cho phép người kiểm tra kiểm tra phần mềm một cách thực tế, sử dụng dữ liệu người dùng thực tế trong môi trường người dùng tự nhiên, tuân theo các điều kiện bên ngoài tương tự. Chỉ con người, không phải máy tính, mới có thể đánh giá khả năng sử dụng và khả năng truy cập của ứng dụng cũng như giao diện của ứng dụng đối với người dùng cuối. Nó cũng cung cấp một cái nhìn rộng hơn về hệ thống. Cuối cùng, một số kịch bản kiểm thử không thể được tự động hóa và cần phải được kiểm thử thủ công. Vì vậy, bạn nên kiểm tra phần mềm theo cách thủ công trước khi thử tự động hóa nó.

[**2. Ưu điểm và nhược điểm của Kiểm thử thủ công là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#2-what-are-the-advantages-and-disadvantages-of-manual-testing)

[**Ưu điểm của kiểm tra thủ công**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#advantages-of-manual-testing)

* Ưu tiên những sản phẩm có vòng đời ngắn.
* Tiết kiệm thời gian, tiền bạc và nguồn lực.
* Đảm bảo sản phẩm không có lỗi.
* Có thể sử dụng để thử nghiệm thăm dò, thử nghiệm đặc biệt và thử nghiệm khả năng sử dụng.
* Không cần phải thay đổi toàn bộ mã để thực hiện những thay đổi nhỏ.
* Nhận phản hồi giao diện người dùng chính xác.
* Khả năng xử lý các tình huống sử dụng khó khăn một cách tốt hơn.
* Kiểm tra GUI có thể được thực hiện chính xác.
* Độ tin cậy cao.
* Làm cho sự thân thiện với người dùng tốt hơn.
* Dễ học cho người mới thử nghiệm.

[**Nhược điểm của kiểm tra thủ công**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#disadvantages-of-manual-testing)

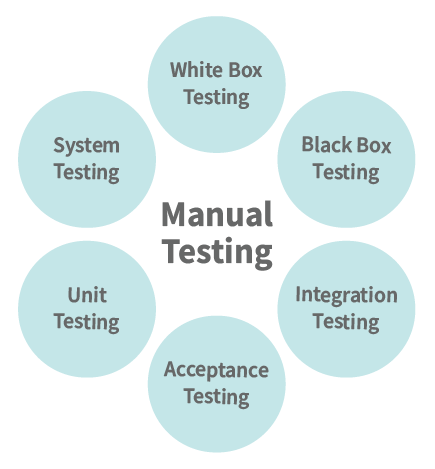
* Không phù hợp với các dự án có giới hạn thời gian và các tổ chức lớn
* Dễ mắc lỗi và sai sót của con người hơn
* Kém hiệu quả hơn do không có lựa chọn ghi lại quá trình kiểm tra
* Ít đáng tin cậy hơn
* Kiểm tra hồi quy tốn thời gian
* Không bao gồm tất cả các khía cạnh của thử nghiệm
* Kiểm tra tải và kiểm tra hiệu suất có thể được thực hiện thủ công
* Đắt hơn trong quá trình dài hạn

[**3. Kể tên một số công cụ kiểm tra thủ công.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#3-name-some-of-the-manual-testing-tools)

Một số công cụ kiểm tra thủ công hàng đầu bao gồm:

* Postman
* Message queue monitors
* DB tools, etc.

[**4. Có những loại thử nghiệm thủ công nào?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#4-what-types-of-manual-testing-are-there)

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/types_of_manual_testing.png)

Trong vòng đời kiểm thử, có nhiều loại kiểm thử thủ công hoặc kỹ thuật kiểm thử thủ công khác nhau có thể được sử dụng. Sau đây là danh sách của họ:

* Kiểm tra hộp đen
* Kiểm tra hộp trắng
* Kiểm tra đơn vị
* Thử nghiệm hệ thống
* Thử nghiệm hội nhập
* Kiểm tra chấp nhận

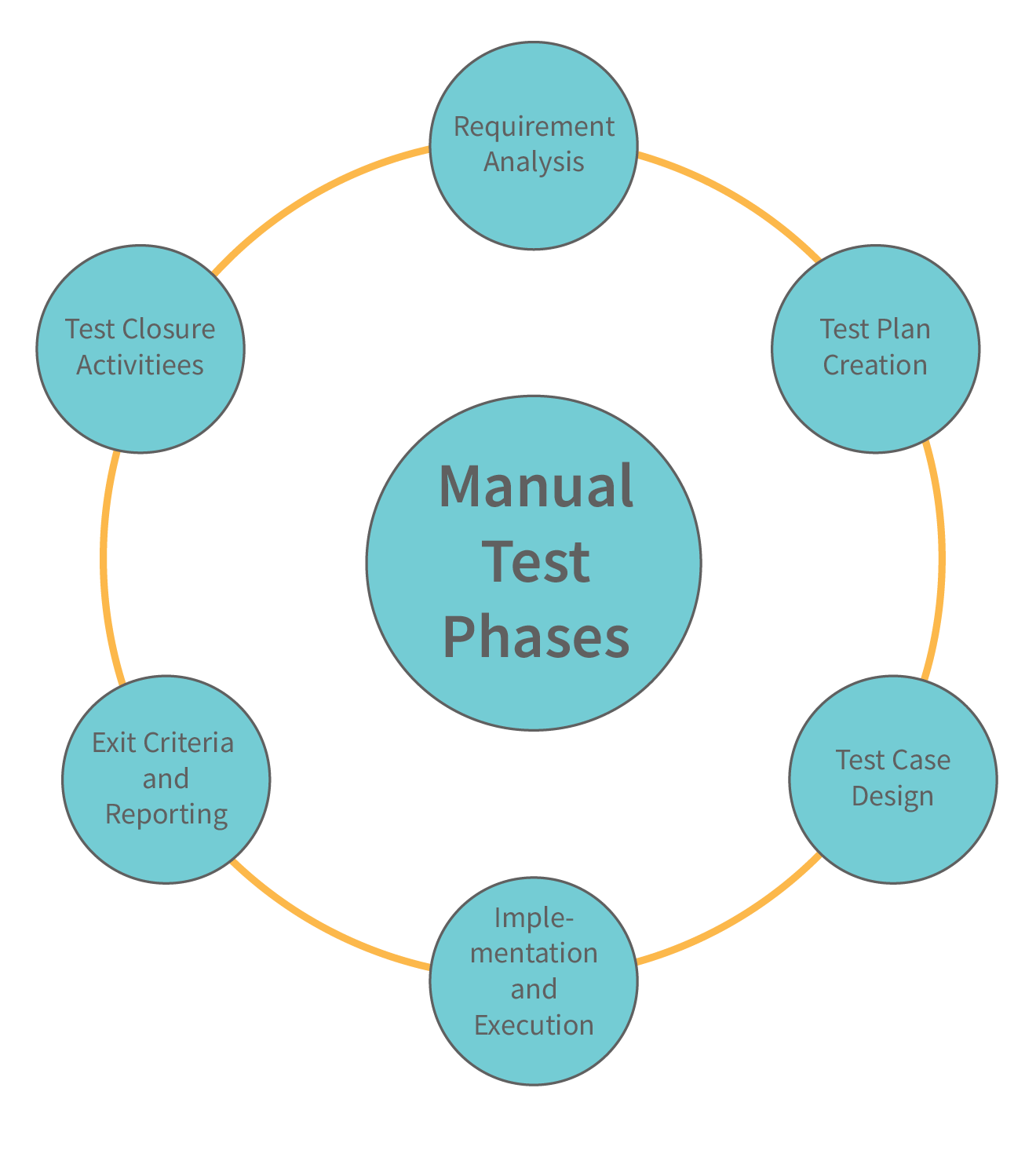
[**5. Người kiểm thử thủ công là ai? Viết vai trò và trách nhiệm của nó.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#5-who-is-a-manual-tester-write-its-roles-and-responsibilities)

Người kiểm tra thủ công là người chuyên nghiệp tiến hành kiểm tra chất lượng trên các ứng dụng phần mềm mà không sử dụng các công cụ tự động hóa hoặc tập lệnh. Về bản chất, chuyên ngành này liên quan đến việc kiểm tra lỗi phần mềm theo cách thủ công và sửa chúng. Người kiểm thử thủ công phải có kỹ năng phù hợp và có thể đáp ứng được yêu cầu của công ty.

Vai trò và trách nhiệm của người kiểm thử thủ công

* Phân tích yêu cầu của khách hàng.
* Xem xét mã đã viết để tuân thủ các thông số kỹ thuật của dự án.
* Tạo môi trường thử nghiệm để thực hiện các trường hợp thử nghiệm.
* Thiết lập các chiến lược đảm bảo chất lượng và tổ chức thử nghiệm theo từng giai đoạn.
* Tổ chức và tiến hành các cuộc họp đánh giá.
* Thực hiện và phân tích các trường hợp thử nghiệm.
* Phát hiện và sửa lỗi.
* Theo dõi lỗi hệ thống và thảo luận với đồng nghiệp.
* Giữ liên lạc với người quản lý kiểm tra, v.v.

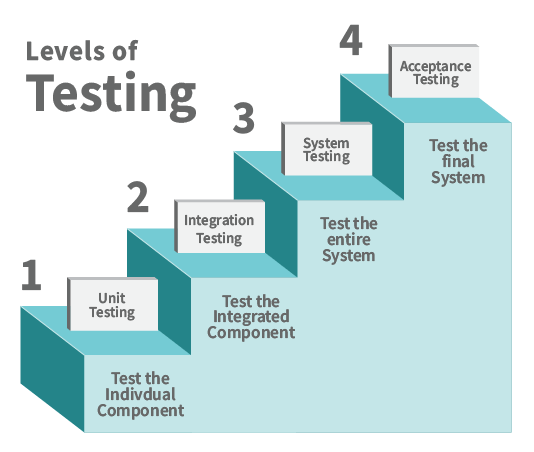
[**6. Mô tả quy trình kiểm thử thủ công.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#6-describe-the-manual-testing-process)

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/manual_testing_process.png)

Trong số các bước liên quan đến kiểm tra thủ công là:

* Phân tích yêu cầu
* Tạo kế hoạch kiểm thử
* Thiết kế các kịch bản thử nghiệm và các trường hợp thử nghiệm
* Thực hiện kiểm thử và báo cáo lỗi
* Đánh giá tiêu chí thoát và báo cáo
* Hoạt động kết thúc thử nghiệm

[**7. Bạn có thể cho tôi biết các cấp độ kiểm thử thủ công khác nhau là gì không?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#7-can-you-tell-me-what-the-different-levels-of-manual-testing-are)

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/levels_of_testing.png)

Các cấp độ thử nghiệm khác nhau có thể được thực hiện trong quá trình phát triển. Kiểm thử đa cấp tạo điều kiện thuận lợi cho việc xác định sớm các lỗi trong quá trình phát triển. Bốn cấp độ thử nghiệm như sau:

* **Kiểm thử đơn vị** : Về cơ bản, đây là một cách kiểm thử các đoạn mã được tách biệt một cách hợp lý trong một hệ thống được gọi là các đơn vị. Chủ yếu, nó tập trung vào độ chính xác chức năng của mô-đun độc lập.
* **Kiểm thử tích hợp** : Kiểm thử phần mềm ở cấp độ này bao gồm việc kết hợp và kiểm thử các đơn vị riêng lẻ để xem liệu chúng có hoạt động cùng nhau như bình thường hay không. Thử nghiệm này tập trung vào giao diện giữa các mô-đun.
* **Kiểm tra hệ thống** : Nó bao gồm việc kiểm tra toàn bộ các thành phần của sản phẩm để đảm bảo đáp ứng các yêu cầu chung của sản phẩm. Các loại thử nghiệm hệ thống bao gồm thử nghiệm hồi quy, thử nghiệm khả năng sử dụng và thử nghiệm chức năng.
* **Kiểm tra chấp nhận của người dùng** : Kiểm tra chấp nhận hay còn gọi là UAT (Kiểm tra chấp nhận của người dùng), là bước cuối cùng trong quy trình kiểm thử phần mềm. Thử nghiệm này xác định xem phần mềm đã sẵn sàng để phát hành hay chưa.

[**8. Để thực hiện manual testing cần có những kỹ năng gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#8-in-order-to-perform-manual-testing-what-skills-are-required)

Sau đây là những kỹ năng kiểm tra thủ công quan trọng cần có: -Định hướng chi tiết và có thể báo cáo kết quả kiểm tra một cách chuyên nghiệp.

* Một khả năng phân tích mạnh mẽ.
* Có khả năng thực hiện kiểm tra kỹ thuật.
* Làm quen với các phương pháp Agile.
* Lập kế hoạch và theo dõi quá trình thử nghiệm.
* Kiến thức về SDLC, STLC, SQL và các khái niệm thủ công.
* Hiểu biết về các công cụ quản lý kiểm thử, công cụ theo dõi kiểm thử và kỹ thuật kiểm thử.

[**9. Phạm vi kiểm tra là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#9-what-is-test-coverage)

Phạm vi kiểm thử là thước đo cho biết mức độ mã nguồn được kiểm thử, cho phép người kiểm thử xác minh chất lượng kiểm thử của họ. Người thử nghiệm có thể sử dụng nó để xác định xem liệu họ có đang thử nghiệm mọi thứ mà họ phải làm hay không. Tùy thuộc vào cách mọi người tiếp cận thử nghiệm, phạm vi thử nghiệm có thể có ý nghĩa khác nhau đối với những người khác nhau.

* **Sản phẩm** : Có nghĩa là xem xét phạm vi kiểm tra để trả lời câu hỏi: Kiểm tra của bạn bao gồm những tính năng hoặc lĩnh vực nào của phần mềm?
* **Yêu cầu** : Phần mềm có thể hoạt động tốt nhưng sẽ không hữu ích cho khách hàng nếu nó không đáp ứng được nhu cầu của họ. Phạm vi yêu cầu cho biết có bao nhiêu yêu cầu được kiểm tra.
* **Mã nguồn** : Đây thường là miền của nhà phát triển và là kỹ thuật kiểm thử hộp trắng. Nhà phát triển có thể kiểm tra xem bao nhiêu mã nguồn của họ được bao phủ bởi các bài kiểm tra đơn vị.

[**10. Kể tên một số phương pháp có thể được sử dụng trong việc bao phủ mã.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#10-name-some-methods-that-can-be-used-in-code-coverage)

Phạm vi mã là một thước đo kiểm thử phần mềm nhằm đo lường số khối, dòng hoặc vòng mã được thực thi khi bộ kiểm thử chạy. Phạm vi mã có thể được xác định bằng một số phương pháp, bao gồm:

* Tuyên bố bảo hiểm.
* Bao phủ quyết định.
* Bảo hiểm chi nhánh.
* Chuyển đổi phạm vi bảo hiểm.

[**11. Xác định khiếm khuyết tiềm ẩn.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#11-define-latent-defect)

Lỗi tiềm ẩn, đúng như tên gọi, là một loại lỗi hoặc lỗi đã có trong hệ thống phần mềm từ lâu nhưng hiện mới được phát hiện. Khiếm khuyết tiềm ẩn là khiếm khuyết hiện có có thể được phát hiện một cách hiệu quả bằng cách kiểm tra. Nó thường bị ẩn hoặc không hoạt động và là một lỗi có mức độ ưu tiên thấp.

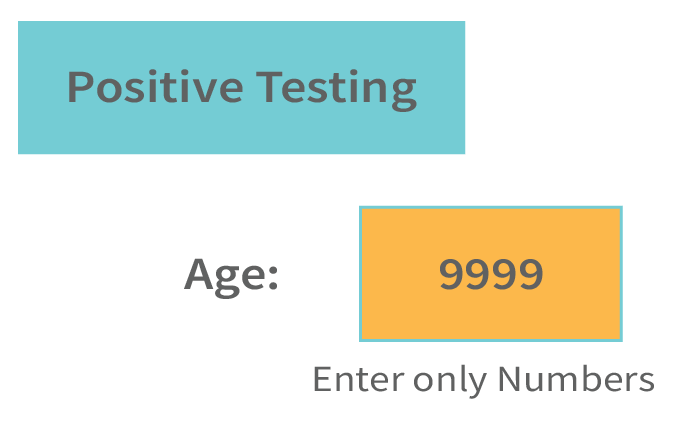
[**12. Kể tên một số thuộc tính của test case.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#12-name-some-attributes-of-the-test-case)

Có nhiều thuộc tính khác nhau của các trường hợp kiểm thử giúp chúng đáng tin cậy, rõ ràng và ngắn gọn hơn, tránh mọi loại dư thừa. Một số trong số họ được đưa ra dưới đây:

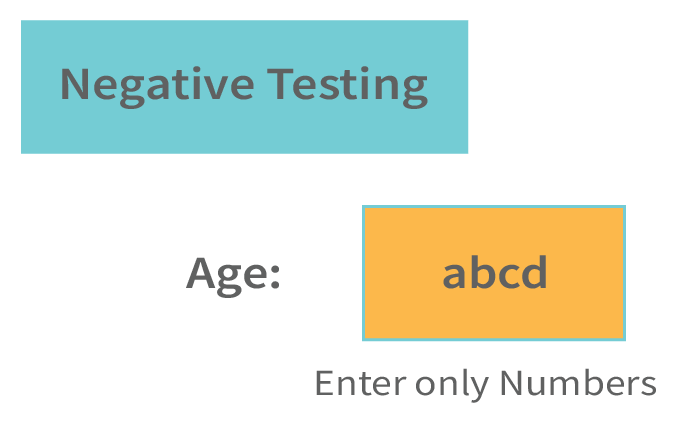
* **Id trường hợp thử nghiệm** : Mã định danh duy nhất của trường hợp thử nghiệm.
* **Tóm tắt thử nghiệm** : Tóm tắt một dòng về trường hợp thử nghiệm.
* **Description** : Mô tả chi tiết về test case.
* **Điều kiện tiên quyết** hoặc **điều kiện tiên quyết** : Tập hợp các điều kiện cần tuân theo trước khi thực hiện các bước kiểm tra.
* **Các bước kiểm thử** : Các bước chi tiết để thực hiện các trường hợp kiểm thử.
* **Test Data** : Giá trị dữ liệu test được sử dụng trong test case.
* **Kết quả mong đợi** : Kết quả ước tính để vượt qua bài kiểm tra.
* **Kết quả thực tế** : Kết quả thực tế sau khi thực hiện các bước kiểm tra.
* **Kết quả kiểm tra** : Trạng thái thực hiện kiểm tra (Đạt hoặc Không đạt).
* **Trạng thái tự động hóa** : Mã định danh cho tự động hóa.
* **Ngày** : Ngày thực hiện thử nghiệm.
* **Người thực hiện** : Tên người thực hiện test case

[**13. Thử nghiệm Tích cực và Tiêu cực là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#13-what-is-positive-and-negative-testing)

* **Kiểm tra tích cực** : Đây là một loại quy trình kiểm tra trong đó ứng dụng phần mềm được xác thực dựa trên các tập dữ liệu hợp lệ làm đầu vào. Nó chỉ đơn giản được sử dụng để kiểm tra xem ứng dụng có thực hiện những gì nó được yêu cầu hay không.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Positive_Testing.png)

* **Kiểm thử tiêu cực** : Đây là một loại quy trình kiểm thử trong đó ứng dụng phần mềm được xác thực dựa trên các tập dữ liệu không hợp lệ làm đầu vào. Nó chỉ đơn giản được sử dụng để kiểm tra xem hệ thống có hiển thị lỗi khi đáng lẽ phải thực hiện hay không. Trong việc thực hiện các trường hợp kiểm thử, kiểm thử âm tính được coi là một yếu tố rất quan trọng.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Negative_Testing.png)

[**14. UAT (Kiểm tra chấp nhận của người dùng) là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#14-what-is-uat-user-acceptance-testing)

Thử nghiệm chấp nhận của người dùng, còn được gọi là thử nghiệm của người dùng cuối, là một phương pháp thử nghiệm được khách hàng hoặc người dùng cuối thực hiện để phê duyệt bản phát hành chính thức. Là bước cuối cùng trong SDLC, nó chỉ diễn ra sau khi kiểm tra kỹ phần mềm. Công cụ này chủ yếu được sử dụng để xác thực các quy trình kinh doanh từ đầu đến cuối. Nó xác minh xem phần mềm được phát triển có sẵn sàng tung ra thị trường hay không.

[**15. Giải thích Trình điều khiển Kiểm tra và Sơ kết Kiểm tra.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#15-explain-test-driver-and-test-stub)

Trình điều khiển thử nghiệm và cuống thử nghiệm đều là các loại khai thác thử nghiệm mô phỏng môi trường để thử nghiệm mô-đun hoặc thành phần. Cả hai đều là mô-đun giả được thiết kế đặc biệt để thử nghiệm.

* **Sơ khai kiểm thử** : Sơ khai kiểm thử được sử dụng theo phương pháp thử nghiệm từ trên xuống và cho phép kiểm tra các cấp mã cao hơn khi các cấp mã thấp hơn chưa được phát triển. Nó phục vụ như một 'chương trình được gọi' khi các chương trình con đang được phát triển.
* **Trình điều khiển thử nghiệm** : Trình điều khiển thử nghiệm được sử dụng theo phương pháp thử nghiệm từ dưới lên và cho phép thử nghiệm các cấp mã thấp hơn khi các cấp mã cao hơn chưa được phát triển. Nó phục vụ như một 'chương trình được gọi' khi các chương trình chính đang được phát triển.

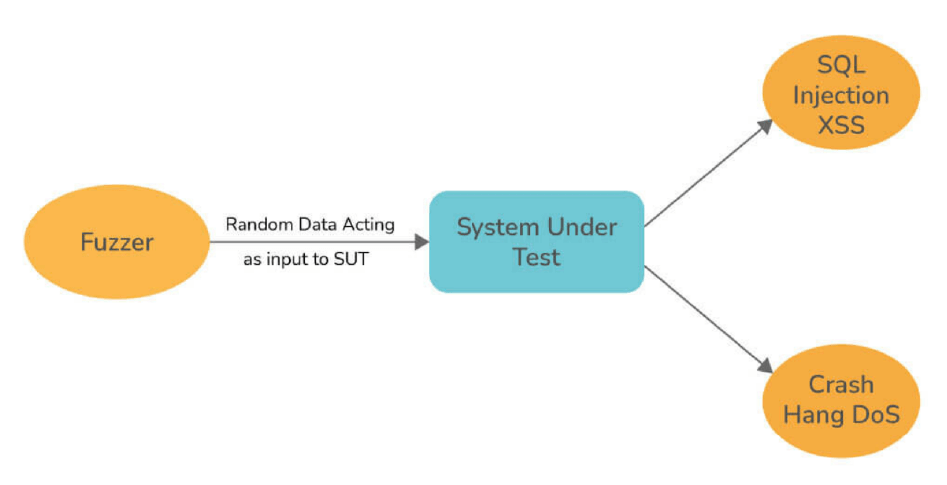
[**16. Tầm quan trọng của Kiểm tra bản địa hóa là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#16-what-is-the-importance-of-localization-testing)

Trong thử nghiệm bản địa hóa, một sản phẩm phần mềm được kiểm tra để đảm bảo liệu nó có cung cấp đầy đủ chức năng và khả năng sử dụng trong một ngôn ngữ cụ thể hay không. Đơn giản, nó xác minh tính chính xác và phù hợp của nội dung. Ngoài ngôn ngữ học, nó còn đề cập đến các truyền thống, hành vi chung của bầy đàn và các yếu tố tương tự khác. Nói chung, nó liên quan đến chức năng và GUI của ứng dụng.

[**17. Bạn có ý nghĩa gì khi nói Kiểm tra cơ sở và Kiểm tra điểm chuẩn?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#17-what-do-you-mean-by-baseline-testing-and-benchmark-testing)

* **Kiểm tra cơ bản** : Đây là một loại thử nghiệm phi chức năng trong đó một tập hợp các thử nghiệm được chạy để nắm bắt thông tin hiệu suất. Bằng cách sử dụng thông tin thu thập được này, chúng tôi có thể thực hiện các thay đổi cần thiết trong ứng dụng và cuối cùng là cải thiện hiệu suất cũng như khả năng của ứng dụng. Nói chung, nó đề cập đến một tiêu chuẩn được sử dụng làm điểm khởi đầu cho các tác phẩm mới. Thử nghiệm này phát hiện và giải quyết nhiều lỗi.
* **Kiểm tra điểm chuẩn** : Đây là một loại thử nghiệm có sự tham gia của cả nhà phát triển và DBA (Quản trị viên cơ sở dữ liệu) để xác định thông tin hiệu suất hiện tại. Sử dụng thông tin này, người ta có thể cải thiện hiệu suất tương tự bằng cách kết hợp nó với các điểm chuẩn (tiêu chuẩn ngành). Mục tiêu chính của nó là so sánh các bản phát hành phần mềm hiện tại và tương lai với điểm chuẩn cụ thể của chúng.

[**18. Mô tả Fuzz testing là gì và tầm quan trọng của nó.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#18-describe-what-fuzz-testing-is-and-how-important-it-is)

Kiểm thử fuzz là một kỹ thuật kiểm thử phần mềm sử dụng nhiều dữ liệu ngẫu nhiên, được gọi là fuzz, làm đầu vào để tìm hoặc phát hiện các lỗ hổng bảo mật và lỗi mã hóa trong một ứng dụng phần mềm. Điều này hữu ích hơn cho các dự án lớn hơn, nhưng nó chỉ phát hiện những lỗi hoặc khiếm khuyết nghiêm trọng. Nó chỉ đơn giản được sử dụng để kiểm tra lỗ hổng của phần mềm và cho kết quả hiệu quả hơn khi được sử dụng với thử nghiệm beta, thử nghiệm hộp đen, v.v. [](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Fuzz_testing.png)

[**19. Giải thích Kiểm tra cấu hình.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#19-explain-configuration-testing)

Kiểm thử cấu hình là một kỹ thuật kiểm thử phần mềm được sử dụng để đánh giá các yêu cầu về cấu hình của phần mềm. Nó phát hiện ra cấu hình tối ưu của hệ thống mà theo đó ứng dụng hoạt động tốt nhất, do đó việc kiểm tra cấu hình được coi là quan trọng. Nó cũng giúp xác định và giải quyết mọi vấn đề tương thích.

[**20. Kể tên hai tham số có thể hữu ích để kiểm tra chất lượng thực hiện kiểm thử.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#20-name-two-parameters-that-can-be-useful-to-check-the-quality-of-test-execution)

Hai tham số cần thiết để kiểm tra chất lượng thực hiện kiểm thử bao gồm:

* Tỷ lệ rò rỉ khuyết tật: Nó biểu thị tỷ lệ giữa tổng số sản phẩm bị loại bỏ tiềm năng trên tổng sản lượng tổng thể.
* Tỷ lệ loại bỏ khuyết tật: Nó biểu thị tỷ lệ tổng số loại bỏ trên tổng sản lượng tổng thể.

[**21. Kiểm thử API là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#21-what-is-api-testing)

Kiểm tra API (Giao diện lập trình ứng dụng) bao gồm kiểm tra API trực tiếp và là một phần của kiểm tra tích hợp để đảm bảo chúng đáp ứng các yêu cầu về độ tin cậy, chức năng, hiệu suất và bảo mật. Do thiếu GUI nên API được kiểm tra ở lớp thông báo. Người kiểm tra viết mã đưa ra yêu cầu API tới máy chủ cung cấp API, cung cấp đầu vào cần thiết, thu thập đầu ra từ phản hồi và khớp đầu ra thực tế với đầu ra dự kiến.

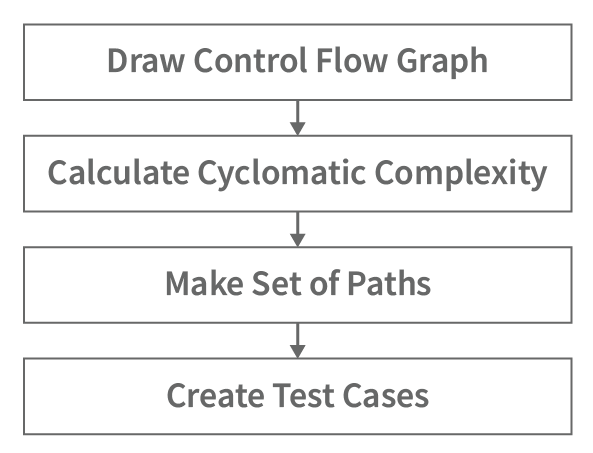
Kiểm tra API chủ yếu liên quan đến logic nghiệp vụ của phần mềm hiển thị API. Nó không liên quan đến giao diện, khả năng truy cập hoặc khả năng sử dụng của phần mềm. Kiểm tra API có thể được tự động hóa để làm cho nó có thể lặp lại và tái tạo mỗi khi chúng chạy.

[**22. Giải thích việc kiểm thử ca sử dụng.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#22-explain-use-case-testing)

Kiểm tra trường hợp sử dụng về cơ bản là một kỹ thuật dành cho nhà phát triển và người kiểm tra để xác định các trường hợp kiểm thử thực hiện toàn bộ hệ thống ngay từ đầu cho đến cuối mỗi giao dịch. Nói chung, nó là một phần của thử nghiệm hộp đen, được sử dụng để phát triển các thử nghiệm hoặc hệ thống đạt được mức hiệu suất chấp nhận được.

[**23. Giải thích Path testing.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#23-explain-path-testing)

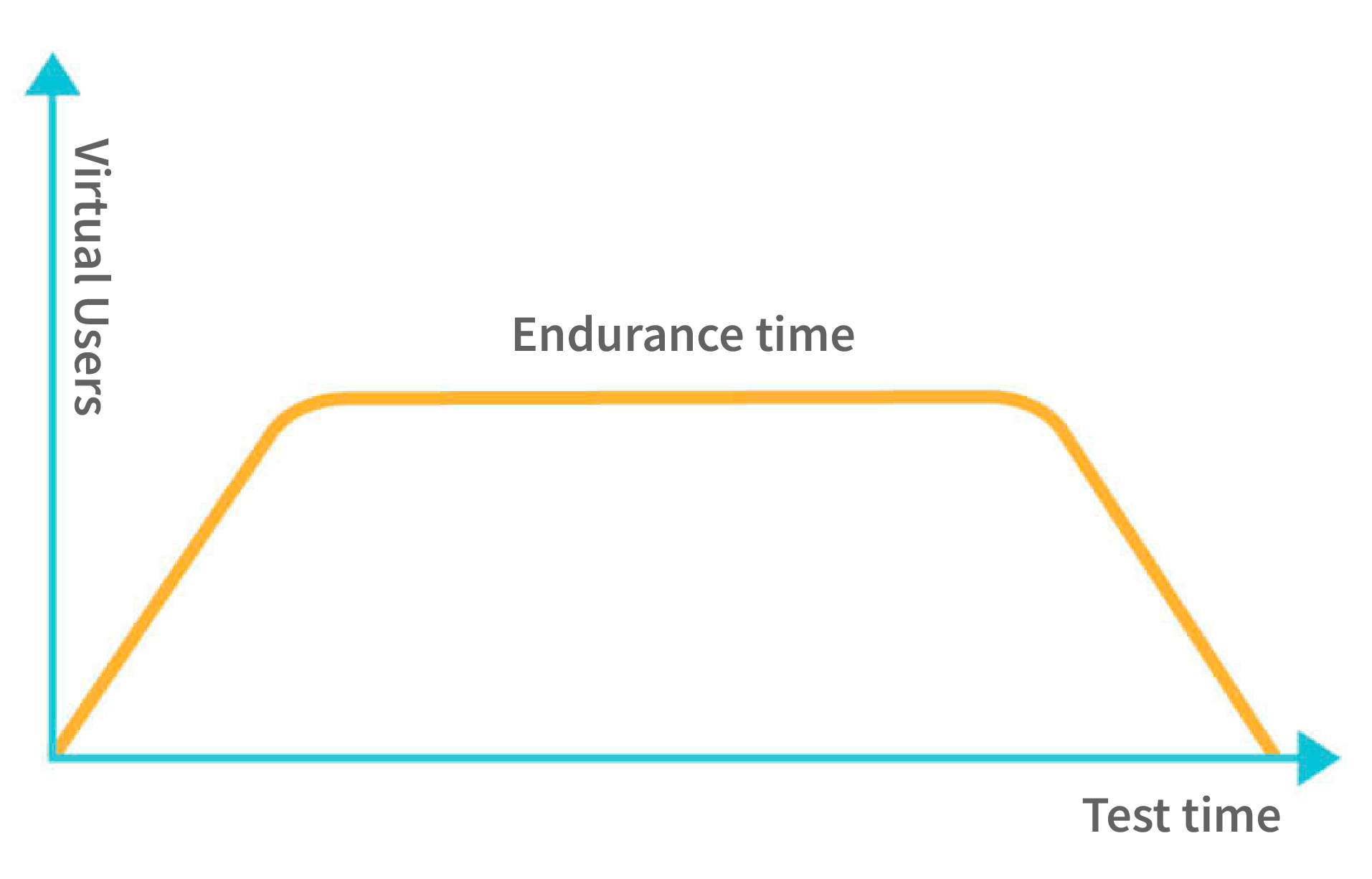
Path test là một loại thử nghiệm được thiết kế đặc biệt để thiết kế các trường hợp thử nghiệm. Nó liên quan đến việc sử dụng biểu đồ luồng điều khiển được thiết kế đặc biệt để xác định một tập hợp các đường dẫn thực hiện độc lập tuyến tính dẫn đến hoàn thành chương trình. Mục tiêu chính của quá trình này là đảm bảo rằng tất cả các đường dẫn đều được bao phủ và thực hiện tốt. Ngoài ra, nó làm giảm hoặc giảm thiểu khả năng xảy ra các thử nghiệm dư thừa.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Draw_testing.png)

[**24. Giải thích Thử nghiệm độ bền hay Thử nghiệm ngâm nước?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#24-explain-endurance-testing-or-soak-testing)

Kiểm tra độ bền, còn được gọi là kiểm tra Ngâm, là một loại kiểm tra hiệu suất thường được thực hiện để kiểm tra hiệu suất của hệ thống được sử dụng liên tục. Mục đích chính của nó là xác định xem hệ thống có thể chịu được tải cao liên tục hay không. Việc sử dụng bộ nhớ cũng được theo dõi để xác định các rò rỉ tiềm ẩn trong quá trình thử nghiệm này. Một số công cụ kiểm tra độ bền bao gồm:

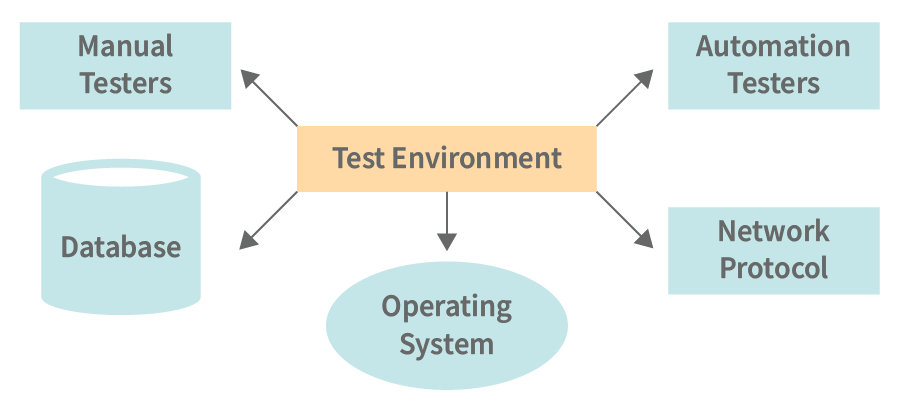
* TẢI Web
* TảiUI
* OpenSTA
* Tải hoàn thành
* Apache JMeter, v.v.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Endurance_Testing.png)

[**Câu hỏi phỏng vấn kiểm tra thủ công dành cho người có kinh nghiệm**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#manual-testing-interview-questions-for-experienced)

[**1. Giải thích thuật ngữ testbed.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#1-explain-the-term-testbed)

Testbed thường được gọi là nền tảng kỹ thuật số được sử dụng để thử nghiệm một ứng dụng. Nó bao gồm hệ điều hành, cơ sở dữ liệu, phần cứng, cấu hình mạng, ứng dụng phần mềm đang được thử nghiệm và tất cả các vấn đề khác liên quan đến phần mềm.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/testbed.png)

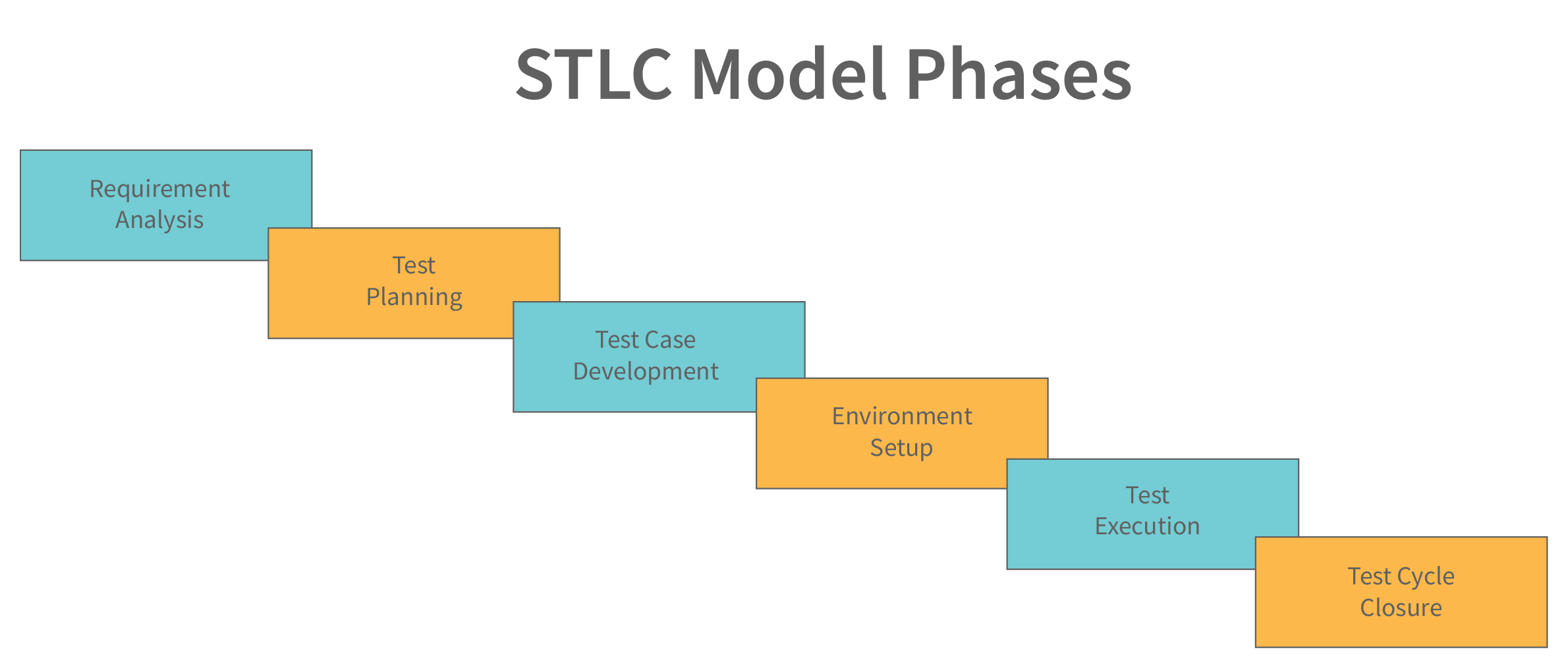
[**2. Giải thích các lỗi, khuyết điểm, sai sót.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#2-explain-bugs-defects-and-errors)

* **Error** : Lỗi xảy ra khi có lỗi lập trình trong code khiến chương trình không thể thực thi hoặc biên dịch.
* **Khiếm khuyết** : Khiếm khuyết là bất kỳ biến thể nào giữa kết quả thực tế và kết quả mong đợi được xác định bởi người kiểm tra hoặc nhà phát triển. Các lỗi thường được phát hiện sau khi sản phẩm được đưa vào sản xuất và chỉ được giải quyết trong giai đoạn phát triển.
* **Lỗi** : Một lỗi phần mềm được phát hiện trong giai đoạn thử nghiệm là lỗi hoặc không khớp. Điều này ảnh hưởng đến chức năng và hiệu suất của phần mềm.

[**3. Vòng đời kiểm thử phần mềm là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#3-what-is-the-software-testing-life-cycle)

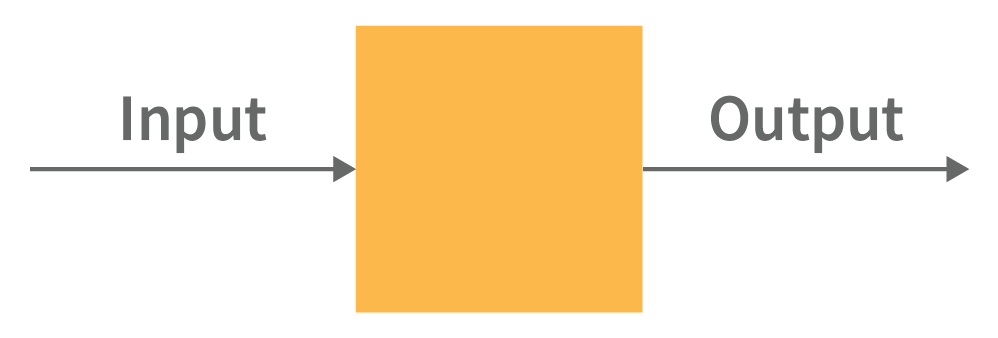
STLC (Vòng đời kiểm thử phần mềm) là một phần cơ bản của SDLC được sử dụng để kiểm tra phần mềm và đảm bảo đáp ứng các tiêu chuẩn chất lượng. Việc xác minh và xác nhận thường liên quan đến quá trình này. Trong đó, các hoạt động khác nhau được thực hiện theo một thứ tự cụ thể trong suốt quá trình kiểm thử phần mềm.

* **Phân tích yêu cầu** : ​​Nhóm QA phân tích các yêu cầu để xác định những gì chúng tôi sẽ kiểm tra và những yêu cầu có thể kiểm tra là gì.
* **Lập kế hoạch kiểm thử** : Giai đoạn này liên quan đến việc xác định chiến lược kiểm thử. Bước này xác định mục tiêu và phạm vi của dự án.
* **Phát triển trường hợp thử nghiệm** : Giai đoạn này liên quan đến việc xác định và phát triển các trường hợp thử nghiệm chi tiết. Dữ liệu thử nghiệm cũng được nhóm thử nghiệm chuẩn bị để thử nghiệm.
* **Thiết lập môi trường thử nghiệm** : Đây là thiết lập phần mềm và phần cứng mà các nhóm thử nghiệm sử dụng để thực hiện các trường hợp thử nghiệm.
* **Thực thi kiểm tra** : Điều này liên quan đến việc thực thi mã và so sánh nó với kết quả mong đợi.
* **Kết thúc chu trình kiểm thử** : Điều này liên quan đến việc triệu tập một cuộc họp của các thành viên trong nhóm kiểm thử để đánh giá và đánh giá các tiêu chí hoàn thành chu trình dựa trên chi phí, thời gian, phạm vi kiểm thử, chất lượng cũng như các mục tiêu và phần mềm kinh doanh quan trọng.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/stlc_model.png)

[**4. Kiểm thử hộp đen là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#4-what-is-black-box-testing)

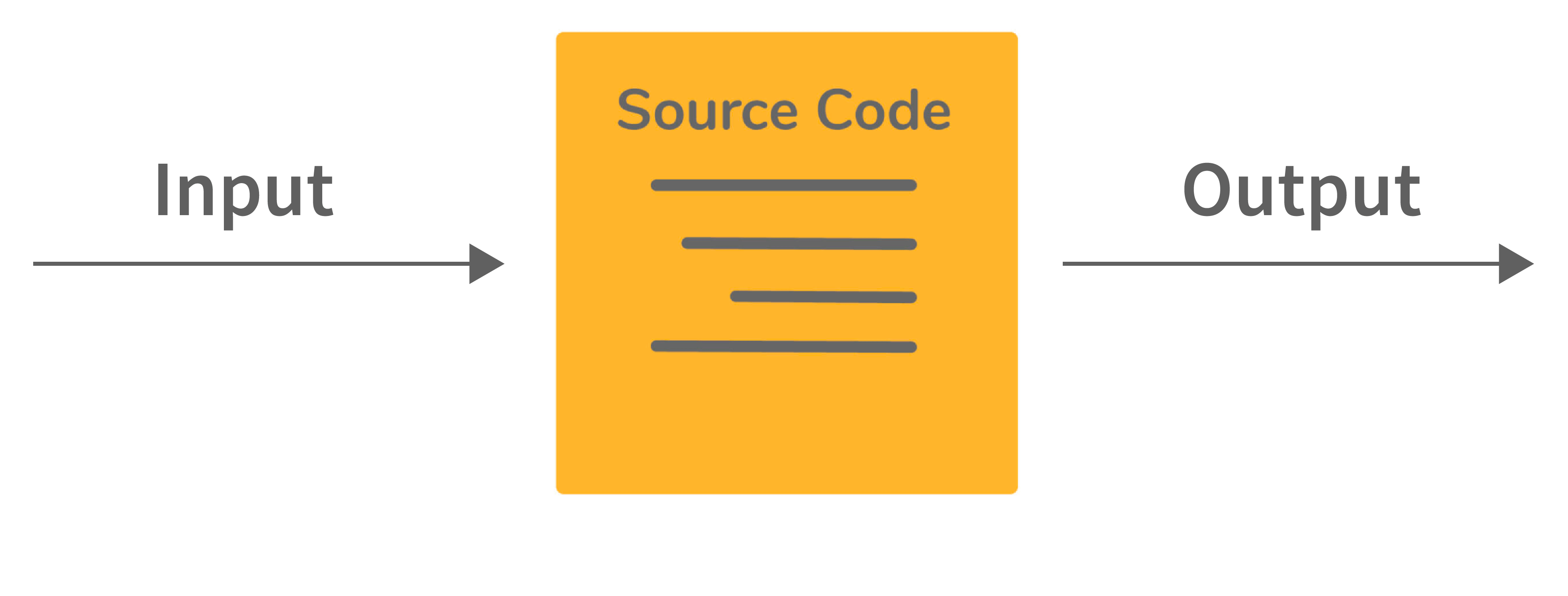
Trong kiểm thử hộp đen, người kiểm thử xem phần mềm như một hộp đen, bỏ qua tất cả cấu trúc và hành vi bên trong. Nó chỉ liên quan đến đầu vào do người dùng cung cấp và đầu ra do hệ thống tạo ra. Kiểm tra hộp đen xác minh hành vi của chương trình so với các yêu cầu đã chỉ định.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/black_box_testing.png)

Là một phần của thử nghiệm hộp đen, các điều kiện thử nghiệm được tạo dựa trên chức năng của phần mềm mà không cần biết nó hoạt động như thế nào trong nội bộ. Theo cách tiếp cận này, phần mềm được kiểm tra từ quan điểm của người dùng cuối và cung cấp cái nhìn rộng hơn về toàn bộ hệ thống. Do thực tế là người dùng chỉ quan tâm đến việc liệu phần mềm có đáp ứng được nhu cầu của họ hay không hơn là cách thức hoạt động của nó, kiểm tra hộp đen là một cách tuyệt vời để kiểm tra khả năng sử dụng phần mềm và dự đoán khách hàng sẽ sử dụng nó như thế nào.

[**5. Kiểm thử hộp trắng là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#5-what-white-box-testing)

Kiểm thử hộp trắng là một phương pháp thay thế cho kiểm thử hộp đen liên quan đến việc xem hệ thống như một hộp trong suốt. Người kiểm thử có thể quan sát quá trình triển khai nội bộ của hệ thống, điều này giúp họ tiến hành kiểm thử. Trong hầu hết các trường hợp, kiểm thử hộp trắng được các nhà phát triển phần mềm thực hiện trong quá trình phát triển.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/white_box_testing.png)

Nó còn được gọi là thử nghiệm hộp kín.

Trong kiểm thử hộp trắng, chúng ta giả định rằng người kiểm thử có một số kiến ​​thức lập trình. Bài kiểm tra bao gồm tất cả các nhánh có thể có mà một chương trình có thể đi theo trong một hệ thống đang chạy. Bạn càng biết nhiều về bên trong hệ thống, chẳng hạn như mã nguồn và chi tiết triển khai, bạn càng có thể kiểm tra nó kỹ lưỡng hơn

[**6. Bạn hiểu lỗi nghiêm trọng là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#6-what-do-you-mean-by-critical-bug)

Thuật ngữ lỗi nghiêm trọng dùng để chỉ một lỗi ảnh hưởng đến phần lớn chức năng của ứng dụng. Khi xảy ra lỗi nghiêm trọng, việc kiểm thử không thể tiến hành cho đến khi lỗi được khắc phục. Chúng chặn chức năng của toàn bộ hệ thống hoặc mô-đun. Một ứng dụng trả về thông báo lỗi máy chủ sau khi cố gắng đăng nhập là một ví dụ về lỗi nghiêm trọng.

[**7. Bạn có ý nghĩa gì khi kiểm tra luồng dữ liệu?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#7-what-do-you-mean-by-data-flow-testing)

Kiểm tra luồng dữ liệu liên quan đến việc phân tích luồng dữ liệu trong một chương trình. Nó là một loại thử nghiệm cấu trúc. Do đó, một lập trình viên có thể thực hiện các thử nghiệm khác nhau về các giá trị và biến dữ liệu. Phương pháp này cung cấp một cách để xác định các biến được sử dụng ở mọi giai đoạn của luồng điều khiển của chương trình. Nó giúp chúng tôi theo những cách sau:

* Loại bỏ hoặc loại bỏ các biến không bao giờ được sử dụng sau khi được khai báo.
* Xác định các biến được sử dụng nhưng không bao giờ được khai báo.
* Phân bổ biến trước khi nó được sử dụng.
* Xác định các biến được xác định nhiều lần trước khi sử dụng

[**8. Thuật ngữ 'chất lượng' có nghĩa là gì khi thử nghiệm?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#8-what-is-the-term-quality-mean-when-testing)

Nhìn chung, phần mềm chất lượng thường không có lỗi, được phân phối đúng thời hạn và phù hợp với ngân sách, đáp ứng hầu hết các yêu cầu và/hoặc mong đợi và dễ bảo trì. Tuy nhiên, “chất lượng” là một khái niệm chủ quan. Rất nhiều điều phụ thuộc vào việc “khách hàng” là ai cũng như mức độ ảnh hưởng của họ nói chung. Ví dụ: người dùng có thể định nghĩa chất lượng là tính thân thiện với người dùng và không có lỗi trong khi bộ phận kế toán có thể định nghĩa chất lượng là lợi nhuận

[**9. Sự khác biệt giữa Kiểm soát chất lượng (QC) và Đảm bảo chất lượng (QA)**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#9-what-is-the-difference-between-quality-controlqc-and-quality-assuranceqa)

**Đảm bảo chất lượng** :

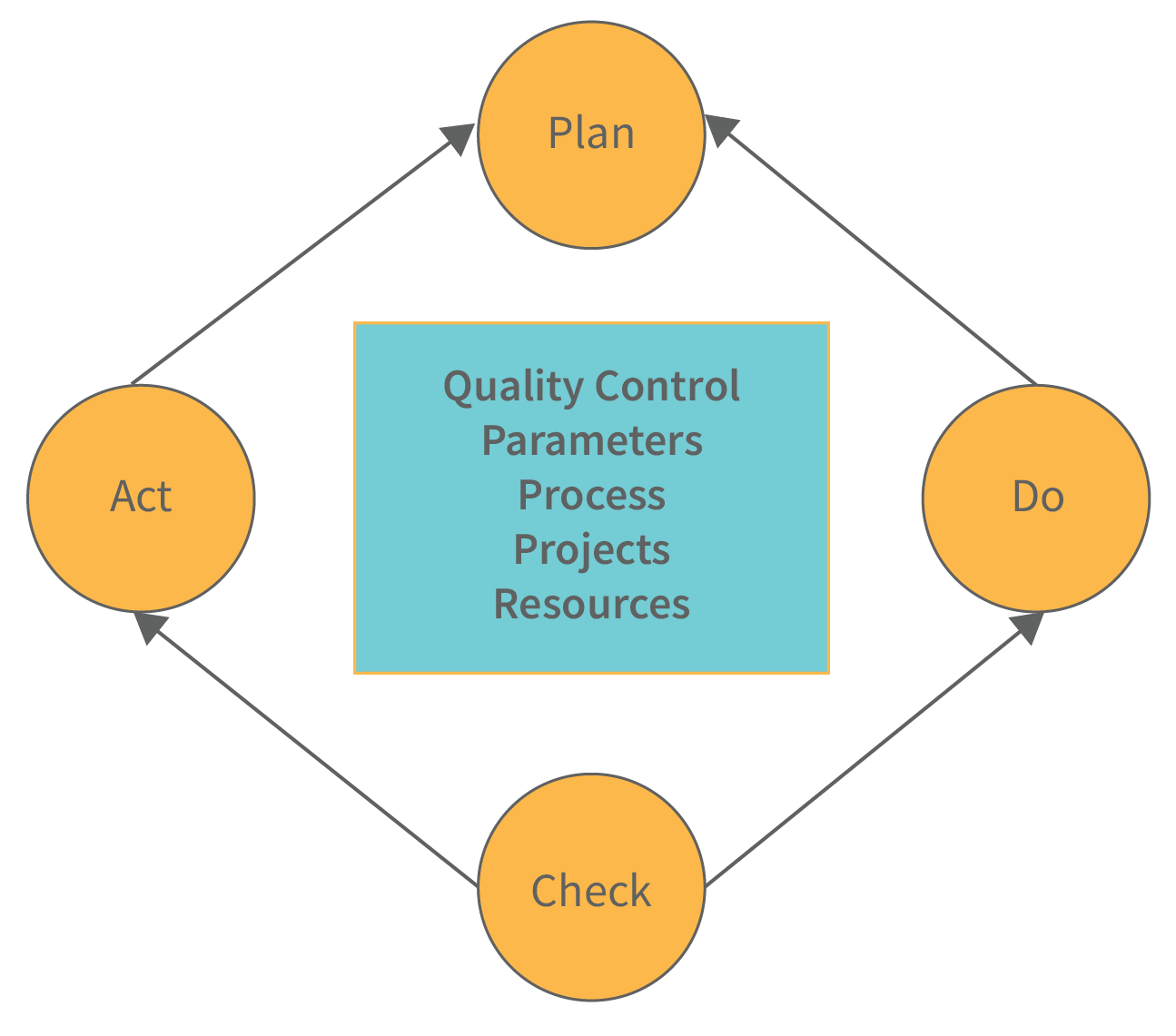
* QA là viết tắt của Đảm bảo chất lượng. Là một phần của nhóm phát triển phần mềm, QA đảm bảo rằng phần mềm được kiểm tra kỹ lưỡng trước khi phát hành cho người dùng cuối. Nó đảm bảo rằng phần mềm được chuyển giao có chất lượng cao.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Quality_Assurance.png)

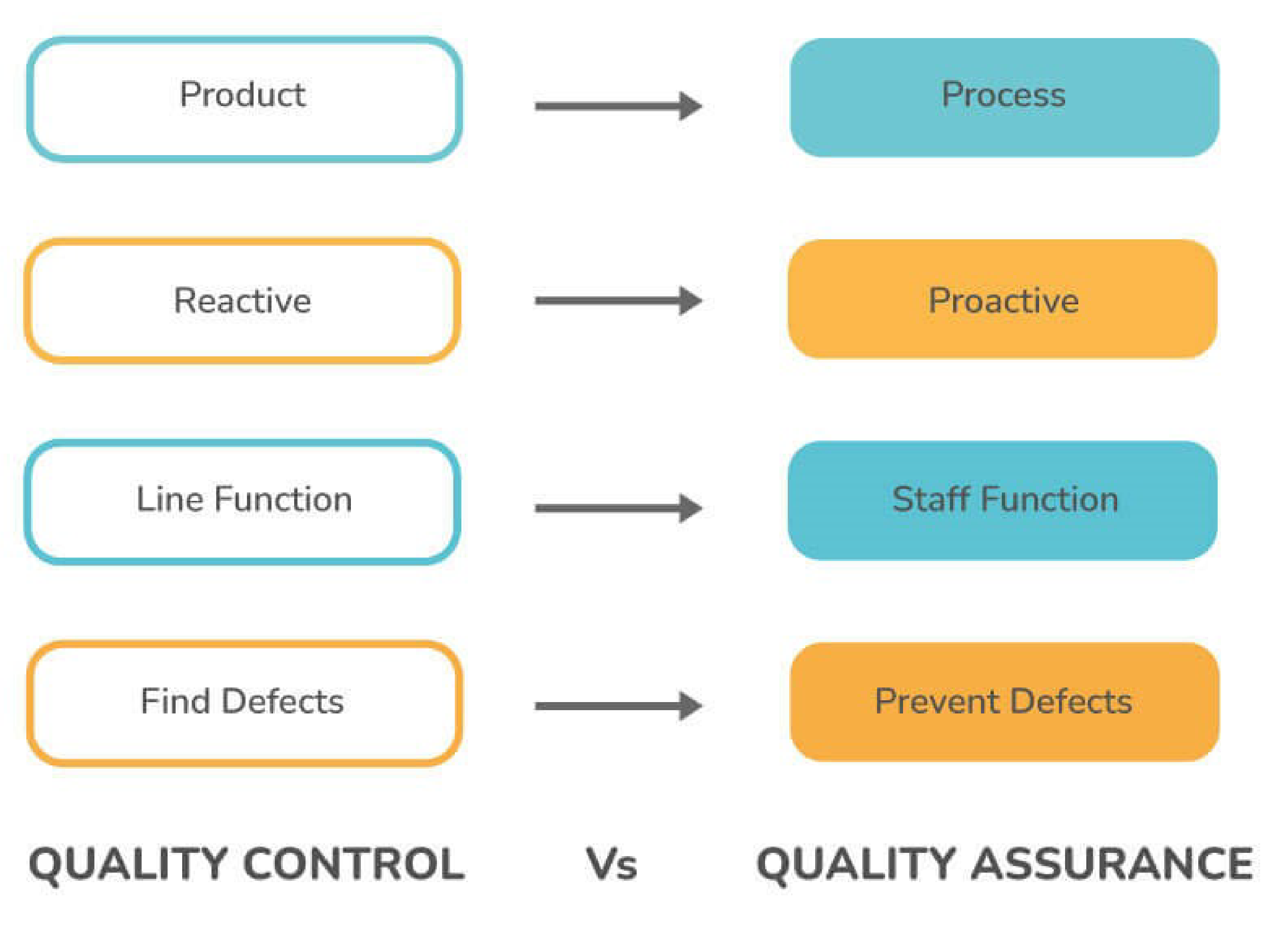
* Nó tập trung vào việc cải thiện quy trình phát triển phần mềm và thường được thực hiện trong giai đoạn phát triển. Người kiểm tra và nhân viên đảm bảo chất lượng có thể là cùng một người trong nhiều tổ chức phần mềm, mặc dù vai trò có thể khác nhau tùy thuộc vào quy mô của tổ chức.

**Kiểm soát chất lượng** :

* QC là viết tắt của Kiểm soát chất lượng. Mục tiêu chính của quá trình này là xác minh rằng các sản phẩm được phát triển có đáp ứng các tiêu chuẩn bắt buộc hay không.

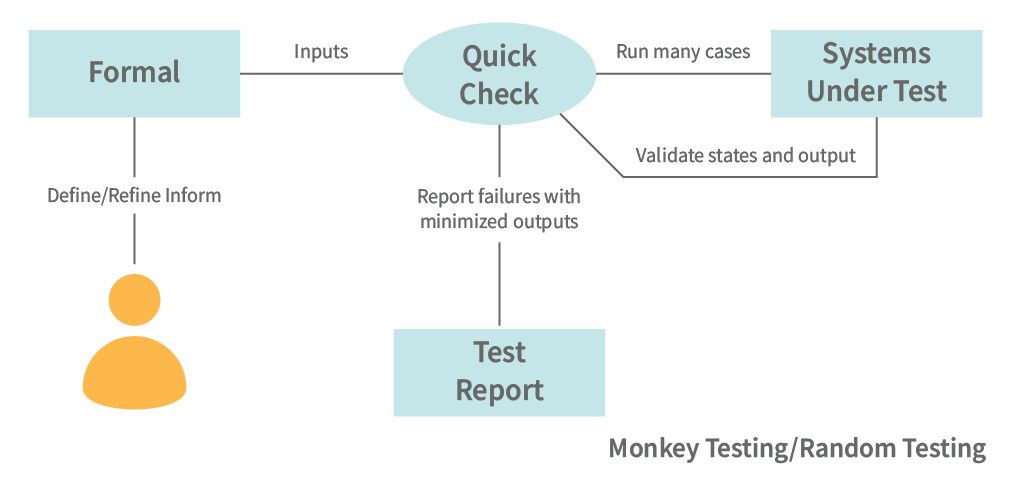
[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Quality_Control.png)

* Kiểm soát chất lượng phần mềm kiểm tra và xem xét các yêu cầu chức năng và phi chức năng của sản phẩm phần mềm để đảm bảo chất lượng của nó. Hoạt động QC thường được thực hiện sau khi sản phẩm được phát triển để đánh giá chất lượng của sản phẩm cuối cùng.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/QA_vs_QC.png)

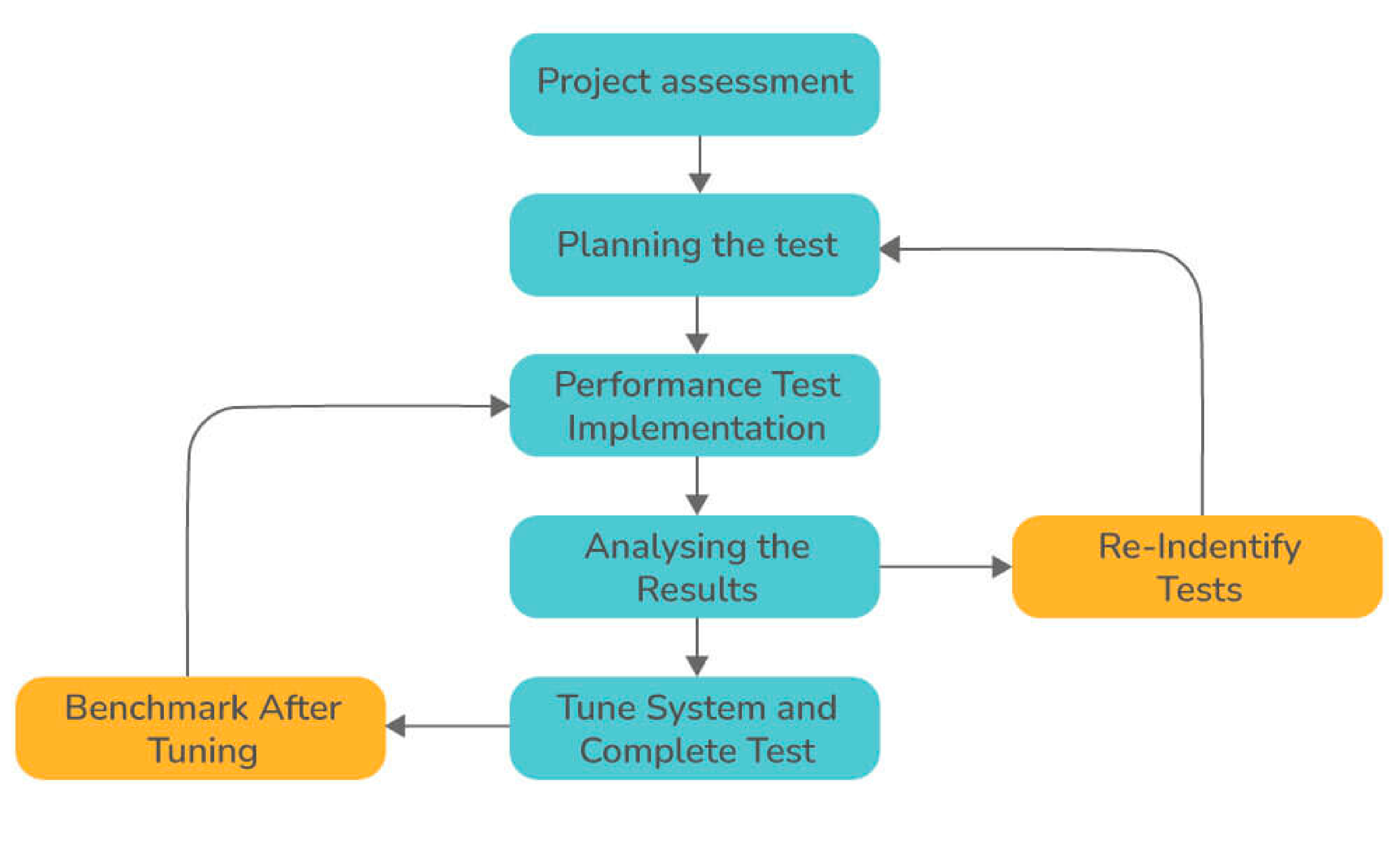
[**10. Giải thích về Monkey testing và Performance testing.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#10-explain-monkey-testing-and-performance-testing)

**Monkey testing** : Monkey testing hay còn gọi là Kiểm thử ngẫu nhiên là một loại kỹ thuật kiểm thử phần mềm trong đó dữ liệu được tạo ngẫu nhiên bằng cách sử dụng một công cụ hoặc một số cơ chế tự động. Đầu vào được tạo ngẫu nhiên này được sử dụng để kiểm tra hệ thống và kết quả sẽ được phân tích tương ứng. Việc kiểm tra loại này không tuân theo bất kỳ quy tắc nào.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Monkey_Testing.png)

**Kiểm tra hiệu suất** : Đây là một loại kỹ thuật kiểm tra phần mềm phi chức năng được sử dụng để xác định các tham số hệ thống như tốc độ, khả năng mở rộng và độ ổn định trong các điều kiện khối lượng công việc khác nhau. Mục đích chính của nó là loại bỏ các tắc nghẽn về hiệu suất chứ không phải để tìm lỗi. Một số thông số chính của kiểm tra hiệu suất bao gồm:

* Sử dụng CPU
* Sử dụng bộ nhớ
* QPS/TPS (Giao dịch mỗi giây)
* Thời gian tải trung bình
* Thông lượng hệ thống, v.v.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Performance_Testing.png)

[**11. Vai trò của tài liệu trong kiểm thử thủ công là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#11-what-is-the-role-of-documentation-in-manual-testing)

Kiểm thử phần mềm hiệu quả phụ thuộc rất nhiều vào tài liệu. Tài liệu phải bao gồm các chi tiết như thông số kỹ thuật yêu cầu, quy tắc nghiệp vụ, cấu hình, thiết kế, kế hoạch kiểm thử, thay đổi mã, trường hợp kiểm thử, báo cáo lỗi, báo cáo kiểm tra, hướng dẫn sử dụng, v.v. Là một phần của kiểm thử phần mềm, các tạo phẩm tài liệu sau thường được áp dụng:

* **Kế hoạch kiểm tra** : Về cơ bản nó là một tài liệu động được kiểm soát và giám sát bởi người quản lý kiểm tra. Một kế hoạch kiểm thử được viết tốt mô tả phạm vi và hoạt động kiểm thử phần mềm là rất quan trọng cho sự thành công của một dự án kiểm thử. Về bản chất, nó đóng vai trò như một bản kế hoạch chi tiết cho tất cả các khía cạnh của quá trình thử nghiệm, chẳng hạn như cái gì, khi nào, như thế nào và hơn thế nữa.
* **Kịch bản thử nghiệm** : Kịch bản thử nghiệm là mô tả chi tiết về một tập hợp các trường hợp thử nghiệm hoặc trường hợp sử dụng. Nó liên quan đến việc thử nghiệm một ứng dụng phần mềm từ quan điểm của người dùng cuối. Nó thường là cơ sở để xây dựng các trường hợp thử nghiệm hoặc trường hợp sử dụng cấp thấp hơn. Kịch bản thử nghiệm cũng có thể được gọi là điều kiện thử nghiệm hoặc khả năng thử nghiệm. Nhìn vào nó sẽ giúp bạn hiểu những gì chúng ta cần kiểm tra.
* **Test Case** : Đúng như tên gọi, test case là một tài liệu chứa dữ liệu test, kết quả mong đợi, điều kiện tiên quyết và điều kiện hậu. Mục đích của tài liệu này là đảm bảo sản phẩm phần mềm đáp ứng các yêu cầu cụ thể cho một kịch bản thử nghiệm cụ thể. Kiểm thử thủ công bao gồm việc người kiểm thử thực hiện các trường hợp kiểm thử một cách thủ công mà không cần dựa vào các công cụ tự động. Trong quá trình phát triển các trường hợp thử nghiệm, có thể xác định được các lỗ hổng trong thông số kỹ thuật.
* **Ma trận truy xuất nguồn gốc** : Đây là một tài liệu, thường được chứa trong một bảng biểu mẫu, minh họa mối quan hệ giữa các yêu cầu và các thành phần khác của dự án từ đầu đến cuối. Nói một cách đơn giản, nó ánh xạ các yêu cầu của khách hàng tới các trường hợp thử nghiệm.

[**12. Giải thích RTM (Ma trận truy xuất nguồn gốc yêu cầu).**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#12-explain-rtm-requirement-traceability-matrix)

RTM **(Ma trận truy xuất nguồn gốc yêu cầu)** được định nghĩa là một công cụ được sử dụng để xác định và theo dõi các yêu cầu cũng như sản phẩm bàn giao của một dự án. Điều này được thực hiện bằng cách thiết lập một luồng cho từng thành phần. Ngoài ra, nó quản lý các yêu cầu tổng thể của dự án. Không có gì phức tạp về phương pháp này và bất cứ ai cũng có thể làm được.

RTM có nhiều dạng. Ví dụ, một ma trận thử nghiệm chứng minh rằng các thử nghiệm đã được tiến hành. Ngoài ra, nó có thể được sử dụng trong quá trình phát triển phần mềm để xác định các vấn đề và yêu cầu.

[**13. Tầm quan trọng của thử nghiệm linh hoạt là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#13-what-is-the-importance-of-agile-testing)

Quá trình kiểm thử linh hoạt bao gồm kiểm thử phần mềm tuân thủ các nguyên tắc phát triển phần mềm linh hoạt. Phần mềm được đánh giá từ quan điểm của khách hàng. Thực hành phát triển phần mềm này bao gồm việc kiểm tra mã mới thường xuyên, tự động và sửa lỗi ngay lập tức ngay khi chúng được phát hiện. Mỗi tính năng đều được kiểm tra khi nó được phát triển. Trong số những ưu điểm của nó là:

* Tất cả người thử nghiệm và nhà phát triển có thể làm việc cùng nhau, nâng cao hiệu suất.
* Mỗi tính năng đều được kiểm tra khi nó được phát triển.
* Đảm bảo sản phẩm có chất lượng cao đáp ứng được sự mong đợi của khách hàng.
* Giảm chi phí và tiết kiệm thời gian.
* Có khả năng thích ứng cao và linh hoạt.
* Hỗ trợ các nhà phát triển phát hành phần mềm càng sớm càng tốt và cải thiện chất lượng sản phẩm.

[**14. Sự khác biệt giữa hồi quy và kiểm tra lại là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#14-what-is-the-difference-between-regression-and-retesting)

* **Kiểm thử hồi quy** : Kiểm thử hồi quy hay còn gọi là kiểm thử chung, xoay quanh việc chạy lại các kiểm thử chức năng và phi chức năng. Nó đặc biệt được thực hiện để đảm bảo liệu phần mềm được phát triển và thử nghiệm trước đó có còn hoạt động như cũ sau khi thay đổi hay không. Nó có thể được thực hiện bằng tay hoặc sử dụng các bài kiểm tra tự động
* **Kiểm tra lại** : Kiểm tra lại, còn được gọi là kiểm tra theo kế hoạch, được sử dụng cho các lỗi cụ thể sau khi được nhà phát triển sửa lỗi. Việc kiểm tra lại được thực hiện để kiểm tra tình huống trong cùng điều kiện môi trường sau khi phát hiện đã được khắc phục.

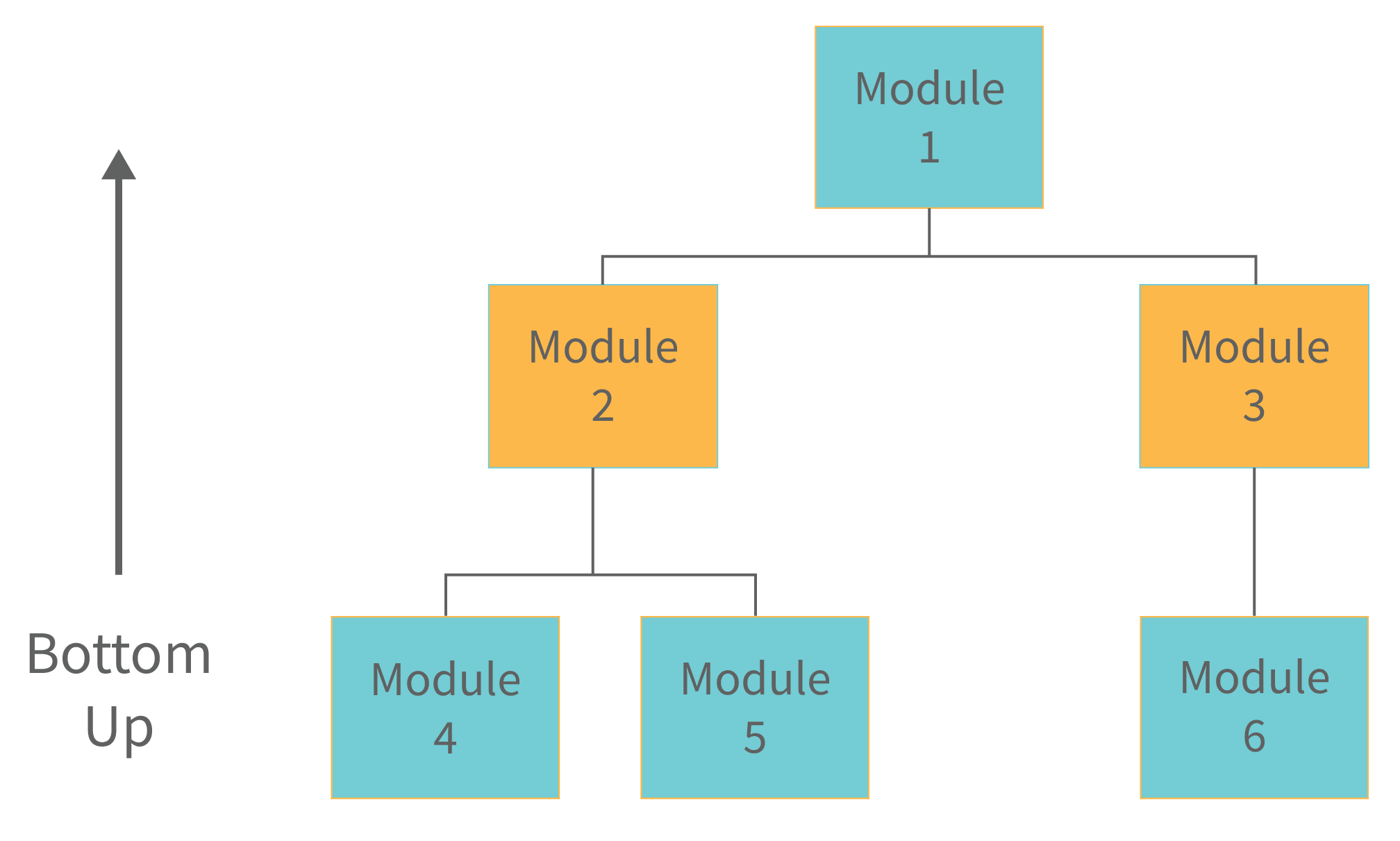
[**15. Kiểm thử hệ thống và Kiểm thử đơn vị là gì? Viết sự khác biệt giữa chúng.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#15-what-is-system-testing-and-unit-testing-write-the-difference-between-them)

* **Kiểm tra hệ thống** : Đây là một kỹ thuật kiểm tra hộp đen điển hình được thực hiện trong một hệ thống hoàn chỉnh và tích hợp đầy đủ để đánh giá sự tuân thủ của hệ thống với các yêu cầu cụ thể của nó. Nó phải điều tra cả yêu cầu chức năng và yêu cầu phi chức năng. Nói chung, nó được thực hiện bởi cả người thử nghiệm và nhà phát triển.
* **Kiểm thử đơn vị** : Trong kiểm thử đơn vị, mỗi thành phần của phần mềm được kiểm thử riêng lẻ. Nói chung, kiểm thử đơn vị được thực hiện bởi các nhà phát triển. Những hệ thống có nhiều sự phụ thuộc lẫn nhau giữa các mô-đun của chúng không thể được kiểm thử bằng thử nghiệm đơn vị.

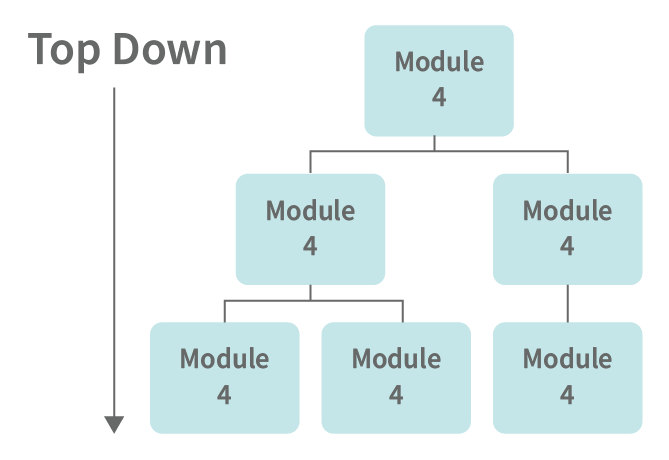
[**16. Các loại Kiểm thử Tích hợp là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#16-what-are-the-types-of-integration-testing)

Kiểm thử tích hợp bao gồm các loại sau:

* **Thử nghiệm vụ nổ lớn** : Nó liên quan đến việc tích hợp tất cả các mô-đun và thành phần cùng một lúc và sau đó thử nghiệm chúng dưới dạng toàn bộ (đơn vị duy nhất). Khi kiểm tra các thành phần này cùng nhau, chúng được coi là một thực thể. Quá trình tích hợp sẽ không được tiến hành nếu tất cả các thành phần của đơn vị không được hoàn thiện.
* **Thử nghiệm từ dưới lên** : Chiến lược này liên quan đến việc thử nghiệm các mô-đun cấp thấp hơn trước, sau đó chuyển sang các mô-đun cấp cao hơn. Miễn là các mô-đun cấp cao nhất đã được thử nghiệm thì quá trình này vẫn tiếp tục. Sau khi tích hợp và thử nghiệm các mô-đun cấp thấp hơn, cấp độ mô-đun tiếp theo sẽ được tạo.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Bottom-Up_Testing.png)

* **Kiểm tra từ trên xuống** : Chiến lược này bao gồm việc kiểm tra hệ thống phần mềm từ trên xuống dưới theo luồng điều khiển. Thử nghiệm được tiến hành đầu tiên trên các mô-đun cấp cao hơn, sau đó là thử nghiệm và tích hợp các mô-đun cấp thấp hơn để xác minh chức năng của phần mềm. Việc kiểm tra được thực hiện bằng cách sử dụng sơ khai khi một số mô-đun chưa sẵn sàng

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Top-Down_Testing.png)

[**17. Kể tên một số công cụ kiểm tra tích hợp phổ biến nhất.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#17-name-some-of-the-most-popular-integration-testing-tools)

Trong số các công cụ kiểm tra tích hợp được sử dụng phổ biến nhất là:

* DBUnit
* Mockito
* Thư xanh
* REST được đảm bảo
* JUnit 5
* Cơ sở dữ liệu H2, v.v.

[**18. Khai thác thử nghiệm và kết thúc thử nghiệm là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#18-what-is-test-harness-and-test-closure)

Khai thác thử nghiệm: Khai thác thử nghiệm, còn được gọi là khung thử nghiệm tự động, là tập hợp phần mềm và dữ liệu thử nghiệm cần thiết cho các mô-đun phần mềm thử nghiệm đơn vị trong quá trình phát triển. Nó chủ yếu được các nhà phát triển sử dụng và giúp tự động hóa cũng như thực hiện các trường hợp kiểm thử đơn vị. Nó thường bao gồm hai phần chính như được đưa ra dưới đây:

* Công cụ thực thi thử nghiệm
* Kho lưu trữ tập lệnh thử nghiệm

Kết thúc kiểm thử: Kết thúc kiểm thử về cơ bản là một tài liệu cung cấp bản tóm tắt tất cả các kiểm thử được thực hiện trong SDLC. Nó cung cấp các báo cáo phân tích chi tiết đầy đủ về các lỗi được phát hiện và loại bỏ. Nó thường được thực hiện trước khi kết thúc quá trình thử nghiệm.

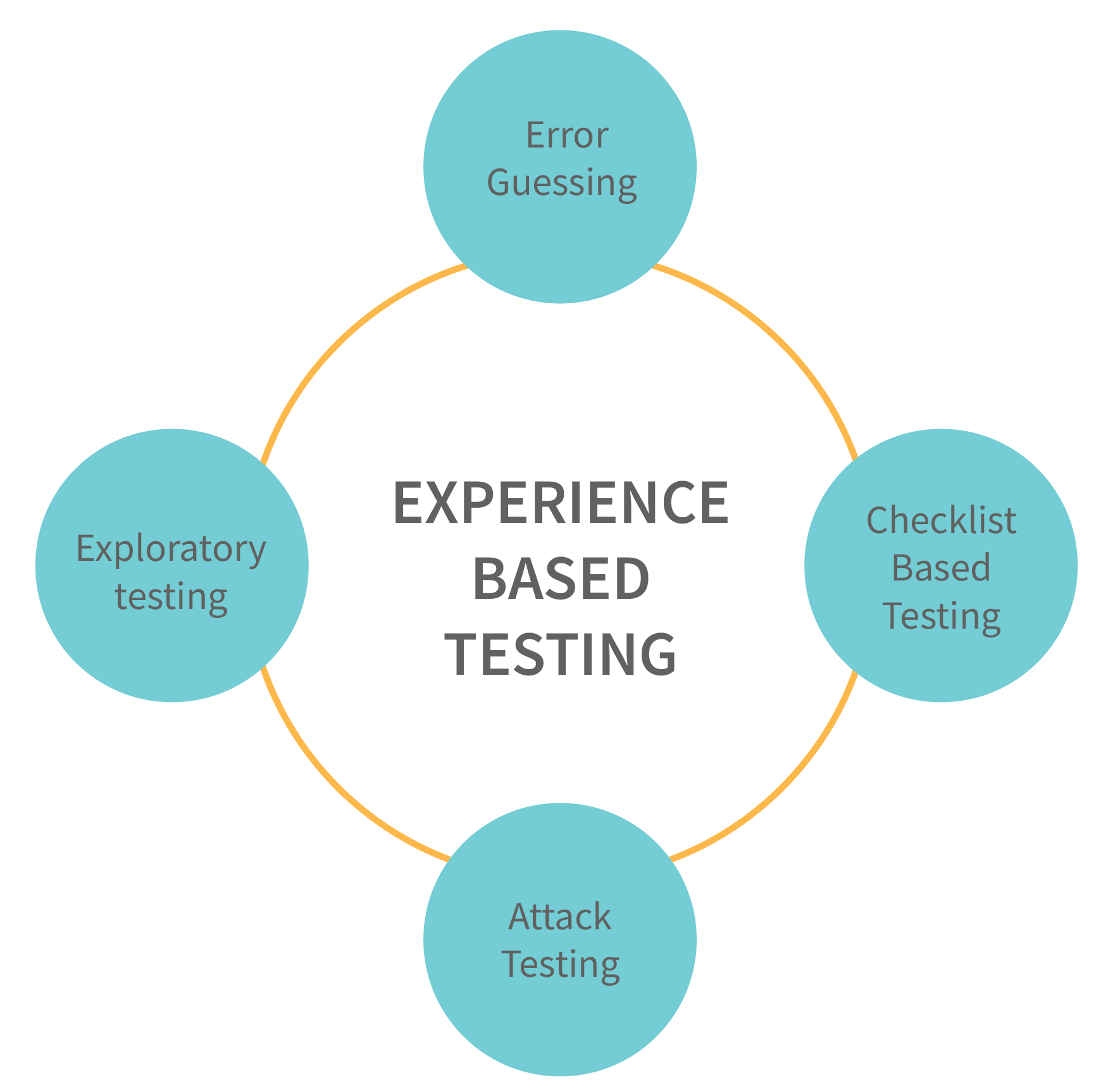
[**19. Giải thích các giai đoạn khác nhau của vòng đời lỗi.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#19-explain-different-stages-of-the-defect-life-cycle)

Vòng đời của lỗi bao gồm các giai đoạn sau:

* **Mới** : Lỗi tiềm ẩn chưa được xác thực.
* **Đã giao** : Đã giao cho một nhóm để giải quyết nhưng chưa giải quyết được.
* **Đang hoạt động** : Các nhà phát triển hiện đang điều tra lỗi và giải quyết nó. Trì hoãn hoặc bị từ chối là hai kết quả có thể xảy ra ở giai đoạn này.
* **Kiểm tra** : Lỗi đã được sửa và có thể kiểm tra.
* **Đã xác minh** : Lỗi đã được kiểm tra lại và kết quả đã được QA xác minh.
* **Đã đóng** : Lỗi ở trạng thái cuối cùng, có thể được đóng sau khi QA kiểm tra lại hoặc khi được coi là trùng lặp hoặc không phải là lỗi.
* **Đã mở lại** : QA mở lại/kích hoạt lại lỗi khi lỗi chưa được khắc phục.
* **Trì hoãn** : Một lỗi không thể giải quyết được trong chu kỳ hiện tại sẽ được hoãn lại để phát hành trong tương lai.
* **Bị từ chối** : Có ba lý do phổ biến để từ chối một lỗi: trùng lặp, không phải lỗi và không thể tái tạo.

[**20. Giải thích các kỹ thuật kiểm tra dựa trên kinh nghiệm.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#20-explain-experienced-based-testing-techniques)

Kỹ thuật kiểm thử dựa trên kinh nghiệm là một loại kiểm thử dựa trên kinh nghiệm của người kiểm thử với việc kiểm thử để hiểu các lĩnh vực thiết yếu của hệ thống. Loại thử nghiệm này thường được sử dụng trong một hệ thống có rủi ro thấp. Thông tin, khả năng và kiến ​​thức nền tảng của cá nhân là những yếu tố hỗ trợ chính cho các điều kiện thử nghiệm và thử nghiệm trong các kỹ thuật dựa trên kinh nghiệm.

[](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README/Experienced-based_testing_techniques.png)

[**21. Viết sự khác biệt giữa smoke testing và sanity testing.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#21-write-the-difference-between-smoke-testing-and-sanity-testing)

* **Kiểm tra khói** : Đây là một loại thử nghiệm được thực hiện để đảm bảo rằng các chức năng cấp tính của chương trình đang hoạt động tốt. Nó hoạt động như một sự xác nhận về việc liệu nhóm đảm bảo chất lượng có thể tiếp tục tiến hành thử nghiệm hay không.
* **Kiểm tra độ chính xác** : Đây là một hình thức kiểm tra không có mô tả được thực hiện để đảm bảo rằng các thay đổi mã được thực hiện đang hoạt động tốt. Nó được thực hiện bởi nhóm thử nghiệm đối với một số thử nghiệm cơ bản. Thử nghiệm này tập trung vào một hoặc một vài lĩnh vực chức năng và thường hẹp và sâu

[**22. Nghịch lý thuốc trừ sâu có ý nghĩa gì? Bạn có thể làm gì để vượt qua nó.**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#22-what-do-you-mean-by-pesticide-paradox--what-you-can-do-to-overcome-it)

Nghịch lý thuốc trừ sâu về cơ bản là một hiện tượng trong đó phần mềm càng được thử nghiệm nhiều thì phần mềm càng trở nên miễn nhiễm với các thử nghiệm của nó. Để khắc phục điều này, người kiểm thử phải luôn tìm ra các chiến lược, cách tiếp cận và trường hợp kiểm thử mới để họ có thể xác định lỗi và giải quyết chúng. Các phương pháp sau đây có thể được sử dụng để ngăn ngừa nghịch lý thuốc trừ sâu:

* Tạo một bộ trường hợp thử nghiệm mới cho các thành phần khác nhau của phần mềm.
* Thêm các trường hợp thử nghiệm mới vào các trường hợp thử nghiệm hiện có.

Việc sử dụng các phương pháp này có thể dẫn đến việc tìm ra nhiều lỗi hơn ở những khu vực mà mức độ lỗi đã giảm. Câu hỏi phỏng vấn thời gian thực về kiểm tra thủ công (Dựa trên kịch bản)

[**Câu hỏi phỏng vấn thời gian thực về kiểm tra thủ công (Dựa trên kịch bản)**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#real-time-interview-questions-on-manual-testing--scenario-based)

[**1. Khi nào nên chọn kiểm thử thủ công thay vì kiểm thử tự động và ngược lại?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#1-when-to-choose-manual-testing-over-automation-testing-and-vice-versa)

Chọn kiểm thử thủ công thay vì kiểm thử tự động

* Khi các trường hợp thử nghiệm cần được chạy trong một khoảng thời gian ngắn (một hoặc hai lần).
* Khi một người cần thực hiện thử nghiệm thăm dò, thử nghiệm khả năng sử dụng hoặc thử nghiệm đặc biệt.
* Khi đánh giá mức độ thân thiện với người dùng của một ứng dụng.
* Bất cứ khi nào cần sự linh hoạt.
* Bất cứ khi nào một người muốn quản lý tốt hơn các tình huống/kịch bản phức tạp.

Chọn Kiểm thử tự động thay vì Kiểm thử thủ công:

* Bất cứ khi nào các trường hợp thử nghiệm phải được chạy lặp đi lặp lại trong một khoảng thời gian dài.
* Khi cần thực hiện kiểm tra hiệu suất, kiểm tra tải hoặc kiểm tra hồi quy.
* Bất cứ khi nào một người muốn ghi lại quá trình thử nghiệm.
* Khi một người có một khoảng thời gian giới hạn để hoàn thành giai đoạn thử nghiệm.
* Khi cần thực hiện các thử nghiệm trong môi trường thời gian chạy tiêu chuẩn.
* Khi các bài kiểm tra liên quan đến các bước lặp đi lặp lại.
* Khi có nhiều triển khai nhanh chóng cho sản phẩm, hướng dẫn sử dụng sẽ trở nên rất tốn thời gian và dư thừa.

[**2. Bạn sẽ xác định thời điểm ngừng thử nghiệm bằng cách nào?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#2-in-what-way-will-you-determine-when-to-stop-testing)

Việc kiểm tra có thể khá khó khăn khi xác định thời điểm dừng. Trong thế giới hiện đại, nhiều ứng dụng phần mềm rất phức tạp và chạy trong nhiều môi trường phụ thuộc lẫn nhau nên việc kiểm thử hoàn chỉnh là không thể. Các yếu tố sau đây thường được xem xét khi quyết định thời điểm ngừng thử nghiệm:

* Nếu thời hạn được đáp ứng (thời hạn phát hành, thời hạn kiểm tra, v.v.) và không còn vấn đề nào có mức độ ưu tiên cao trong hệ thống.
* Hoàn thành các trường hợp thử nghiệm với tỷ lệ đậu nhất định.
* Ngay khi ngân sách thử nghiệm cạn kiệt.
* Thời gian trung bình giữa hai lần thất bại cố hữu được gọi là MTBF (Thời gian trung bình giữa các lần thất bại). Khi MTBF khá cao, giai đoạn thử nghiệm có thể bị dừng tùy theo quyết định của các bên liên quan.
* Ngay khi phạm vi bao phủ mã tự động đáp ứng giá trị ngưỡng được chỉ định và không có lỗi nghiêm trọng nào.
* Nếu tỷ lệ lỗi giảm xuống dưới một mức nhất định.
* Sau khi giai đoạn thử nghiệm Beta hoặc Alpha kết thúc.

[**3. Có thể đạt được phạm vi kiểm tra 100% không? Làm thế nào để bạn đảm bảo phạm vi kiểm tra?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#3-can-100-testing-coverage-be-achieved-how-do-you-ensure-test-coverage)

Việc kiểm tra sản phẩm 100% được coi là không thể. Tuy nhiên, bạn có thể tiến gần hơn đến mục tiêu của mình bằng cách thực hiện theo các bước bên dưới.

* Phát triển một chiến lược thử nghiệm hiệu quả.
* Chuẩn bị một danh sách kiểm tra cho tất cả các hoạt động liên quan đến thử nghiệm.
* Thiết lập danh sách ưu tiên cho các lĩnh vực quan trọng của ứng dụng.
* Liệt kê tất cả các yêu cầu ứng dụng.
* Xác định các rủi ro liên quan đến ứng dụng.
* Sử dụng thử nghiệm tự động.

[**4. Kiểm tra hệ thống có thể được thực hiện ở bất kỳ giai đoạn nào. Có hay không?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#4-system-testing-can-be-done-at-any-stage-yes-or-no)

Không, việc kiểm thử hệ thống không thể được tiến hành ở bất kỳ giai đoạn nào của quá trình phát triển. Trong kiểm thử hệ thống, tất cả các thành phần của phần mềm đều được kiểm thử cùng nhau để đảm bảo rằng sản phẩm tổng thể đáp ứng các thông số kỹ thuật đã chỉ định. Do đó, việc kiểm thử hệ thống không thể diễn ra ở bất kỳ giai đoạn nào; thay vào đó, việc này chỉ phải được thực hiện sau khi tất cả các mô-đun hoặc đơn vị đã sẵn sàng và hoạt động bình thường, nhưng trước Kiểm tra chấp nhận của người dùng (UAT).

[**5. Nếu không có tài liệu thích hợp để thử nghiệm, bạn sẽ thực hiện những bước nào để vượt qua thử thách?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#5-if-proper-documentation-is-not-available-for-testing-what-steps-will-you-take-to-overcome-the-challenge)

QA nên tham khảo các tài liệu tham khảo sau nếu họ không thể tìm thấy các tài liệu tiêu chuẩn như Đặc tả yêu cầu hệ thống hoặc Tài liệu mô tả tính năng.

* Ảnh chụp màn hình
* Khung dây
* Phiên bản trước của ứng dụng.

Ngoài ra, thảo luận với nhà phân tích kinh doanh và nhà phát triển là một phương pháp đáng tin cậy khác. Điều này rất hữu ích trong việc giải quyết những nghi ngờ và mang lại sự rõ ràng cho các yêu cầu. Các email được trao đổi cũng có thể dùng làm tài liệu tham khảo thử nghiệm.

Một tùy chọn khác để xác minh chức năng của ứng dụng là thực hiện kiểm tra khói. Điều này sẽ bộc lộ một số lỗi rất cơ bản trong ứng dụng. Trong trường hợp không có tùy chọn nào ở trên hoạt động, chúng tôi chỉ cần sử dụng kinh nghiệm trước đây của mình để kiểm tra ứng dụng phần mềm.

[**6. Một số phương pháp hay nhất bạn nên tuân theo khi viết test case là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#6-what-are-some-best-practices-that-you-should-follow-when-writing-test-cases)

Khi viết test case, bạn nên tuân theo các nguyên tắc sau:

* Đánh giá rủi ro và thời hạn của dự án trước khi lập kế hoạch và ưu tiên các trường hợp thử nghiệm và viết cho phù hợp.
* Hãy ghi nhớ quy tắc 80/20. Để có phạm vi bao phủ tốt nhất cho ứng dụng của bạn, 20% bài kiểm tra của bạn phải bao gồm 80% chức năng của ứng dụng.
* Hãy đảm bảo rằng bạn không cố gắng kiểm tra tất cả các trường hợp cùng một lúc mà hãy ứng biến chúng khi thực hiện.
* Tạo danh sách các trường hợp thử nghiệm của bạn và phân loại chúng theo các kịch bản và chức năng kinh doanh.
* Tính mô đun và chi tiết rất quan trọng khi thiết kế các trường hợp thử nghiệm.
* Cung cấp các trường hợp thử nghiệm theo cách mà người khác có thể dễ dàng hiểu và sửa đổi nếu cần thiết.
* Hãy nhớ rằng phần mềm được thiết kế cuối cùng là dành cho khách hàng, vì vậy hãy lưu ý đến các yêu cầu của họ.
* Quản lý chu kỳ phát hành ổn định bằng cách sử dụng công cụ quản lý kiểm tra.
* Theo dõi các trường hợp thử nghiệm của bạn một cách thường xuyên. Các trường hợp thử nghiệm phải là duy nhất và không liên quan hoặc trùng lặp phải được loại bỏ.

[**7. Khi các yêu cầu vẫn còn thay đổi, cách tốt nhất để thử nghiệm sản phẩm là gì?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#7-when-the-requirements-are-still-in-flux-what-is-the-best-way-to-test-a-product)

Đối với một số sản phẩm, ngăn xếp yêu cầu không có sẵn. Có thể cần nỗ lực đáng kể để xác định xem ứng dụng có chức năng không mong muốn hay không, điều này cho thấy có vấn đề sâu hơn trong quá trình phát triển phần mềm. Việc loại bỏ chức năng không cần thiết cho mục đích của ứng dụng là một ý tưởng hay. Nếu không, hãy tạo kế hoạch thử nghiệm dựa trên những giả định bạn đã đưa ra về sản phẩm. Tuy nhiên, điều quan trọng là bạn phải ghi lại kỹ lưỡng tất cả các giả định trong kế hoạch kiểm tra.

[**8. Điều gì khiến phân tích giá trị biên trở thành một phương pháp tốt để cung cấp các ca kiểm thử?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#8-what-makes-boundary-value-analysis-a-good-method-for-providing-test-cases)

Phân tích giá trị biên được định nghĩa là một kỹ thuật kiểm thử phần mềm sử dụng các giá trị biên của các lớp tương đương làm đầu vào cho các trường hợp kiểm thử. Kiểm thử hộp đen sử dụng phân tích giá trị biên là một trong những kỹ thuật thiết kế trường hợp được sử dụng phổ biến nhất.

Trong phân tích giá trị biên, các giá trị tại ranh giới được đưa vào các ca kiểm thử. Nhìn chung, đúng là có nhiều lỗi ở ranh giới của miền đầu vào hơn ở trung tâm của nó, điều này làm cho việc phân tích giá trị biên trở thành một phương pháp tuyệt vời để tạo ra các ca kiểm thử. Trong phân tích giá trị biên, các giá trị tại ranh giới được đưa vào các ca kiểm thử. Khi đầu vào nằm trong phạm vi ranh giới thì đó là phép thử dương, nhưng nếu nó nằm ngoài phạm vi ranh giới thì đó là phép thử âm. Các giá trị có thể bao gồm giá trị lớn nhất, nhỏ nhất, cạnh trong, cạnh ngoài, giá trị điển hình hoặc giá trị lỗi.

Ví dụ: Giả sử bạn đang kiểm tra hộp đầu vào chấp nhận các số từ 1 đến 20. Nhờ phân tích giá trị biên, chúng tôi có thể chia các trường hợp kiểm thử thành ba loại:

* Dữ liệu thử nghiệm sẽ giống với ranh giới đầu vào của đầu vào: 1 và 20.
* Giá trị đầu vào phía trên các cạnh cực trị: 2 và 21.
* Giá trị đầu vào bên dưới các cạnh cực trị: 0 và 19.

Do đó, các giá trị biên là 0, 1, 2 và 19, 20, 21.

[**9. Làm sao bạn biết mã đã đáp ứng các thông số kỹ thuật?**](https://github.com/HannachiHassen/Manual-Testing-Interview-Questions/blob/main/README.md#9-how-do-you-know-the-code-has-met-specifications)

Mã có thể bảo trì, dễ đọc và không có lỗi được coi là tốt. Hầu hết mọi tổ chức đều có 'tiêu chuẩn mã hóa' mà các nhà phát triển nên tuân thủ, nhưng mỗi người đều có những ý tưởng khác nhau về điều gì là tốt nhất và điều gì là quá nhiều hoặc quá ít. Có nhiều công cụ đảm bảo rằng các trường hợp kiểm thử sẽ ánh xạ tới các yêu cầu, chẳng hạn như ma trận truy xuất nguồn gốc. Trong trường hợp tất cả các trường hợp thử nghiệm được thực thi thành công thì mã sẽ đáp ứng yêu cầu.