# KIỂM THỬ PHẦN MỀM

## 1. Driver – Stub:

### a. Driver: gọi đơn thể cần được kiểm thử.

+ Là một đơn thể nhỏ được viết ra (đơn thể gọi) để truyền các test-case cho đơn thể cần được kiểm thử (đơn thể bị gọi) và hiển thị kết quả của đơn thể cần kiểm thử.

+ Là một chương trình giả đơn thể mà được dùng để gọi (đơn thể gọi) các đơn thể thấp hơn (đơn thể bị gọi) khi các đơn thể này không tồn tại.

+ Được dùng trong:

- Kiểm thử từng đơn thể riêng lẽ.
- Kiểm thử tích hợp từ dưới lên.
- Kiểm thử tăng tiến và không tăng tiến.
- Kiểm thử tích hợp Sandwich.
- ...

### b. Stub: được gọi bởi đơn thể được kiểm thử.

+ Là một đơn thể nhỏ được viết ra để giả lập chức năng của một đơn thể bị gọi mà đơn thể cần kiểm thử là đơn thể gọi.

+ Là một chương trình nhận vào những dữ liệu đầu vào, yêu cầu từ đơn thể được kiểm thử và trả về kết quả mong muốn.

+ Trong nhiều trường hợp việc tạo ra Stub rất phức tạp và tốn nhiều thời gian như đơn thể thực sự, một số trường hợp có thể trở nên lớn hơn đơn thể được giả lập. Ví dụ như kết nối với một sơ sở dữ liệu...

+ Được dùng trong:

- Kiểm thử từng đơn thể riêng lẽ.
- Kiểm thử tích hợp từ trên xuống.
- Kiểm thử tăng tiến và không tăng tiến.
- Kiểm thử tích hợp Sandwich.
- ...

## 2. Vài loại kiểm thử:

### a. Định nghĩa kiểm thử tích hợp:

+ Còn gọi là tích hợp và kiểm thử là một giải đoạn trong kiểm thử phần mềm. Mỗi đơn thể phần mềm riêng biệt được kết hợp lại và kiểm thử theo nhóm.

+ Kiểm thử tích hợp xảy ra sau kiểm thử đơn vị (unit test) và trước kiểm thử xác nhận. Kiểm thử tích hợp nhận các đơn thể đầu vào đã được kiểm thử đơn vị, nhóm chúng vào các tập lớn hơn, áp dụng các ca kiểm thử đã được định nghĩa trong kế hoạch kiểm thử tích hợp vào tập hợp đó, và cung cấp đầu ra cho hệ thống tích hợp.

### b. Tại sao cần kiểm thử tích hợp:

+ Mặc dù mỗi đơn thể đều được kiểm thử đơn vị nhưng các lỗi vẫn còn tồn tại với các nguyên nhân sau:

- Một đơn thể nói chung được thiết kế bởi một lập trình viên có hiểu viết và logic lập trình có thể khác với các lập trình viên khác. Kiểm thử tích hợp là cần thiết để đảm bảo tính hợp nhất của phần mềm.

- Tại thời điểm phát triển đơn thể vẫn có thể có thay đổi trong spec của khách hàng, những thay đổi này có thể không được kiểm tra ở giai đoạn unit test trước đó.

- Giao diện và cơ sở dữ liệu của các đơn thể có thể chưa được hoàn chỉnh khi được ghép lại.

- Khi tích hợp hệ thống các đơn thể có thể không tương thích với cấu hình chung của hệ thống.

- Thiếu các xử lý ngoại lệ có thể xảy ra.

**c. Các bước thực hiện kiểm thử tích hợp:**

+ Chọn đơn thể hoặc thành phần sẻ được kiểm thử.
+ Kiểm thử đơn vị.
+ Thiết kế các kịch bản thử nghiệm, trường hợp, và script.
+ Thực hiện kiểm tra theo test case đã viết.
+ Theo dõi và tái kiểm thử các lỗi ở trên.
+ Lặp lại các bước trên cho đến khi hệ thống hoàn chỉnh được kiểm tra đầy đủ.

**d. Kiểm thử tăng tiến:**

+ Tích hợp đơn thể cần kiểm thử kế tiếp vào các đơn thể đã được kiểm thử trước đó, sau đó kiểm thử các đơn thề này chạy chung với nhau.
+ Có thể bắt đầu từ đơn thể trên cùng, hoặc đơn thể dưới cùng của chương trình.

**e. Kiểm thử không tăng tiến:**

+ Kiểm thử mỗi đơn thể một cách độc lập sau đó tích hợp các đơn thể này để tạo ra chương trình.
+ Kiểm thử đơn vị được thực hiện trên các đơn thể.
+ Mỗi đơn thể được kiểm thử riêng biệt, cuối cùng tích hợp lại để tạo ra chương trình.

**f. Kiểm thử tăng tiến vs không tăng tiến:**

+ Kiểm thử tăng tiến:
    - Cần nhiều công sức hơn so với kiểm thử không tăng tiến.
    - Các lỗi liên quan đến giao tiếp không đúng giữa các đơn thể sẻ được sớm phát hiện.
    - Việc chuẩn đoán được dễ dàng hơn đối với các lỗi liên quan đến giao tiếp giữa các đơn thể.
    - Việc kiểm thử được thực hiện trọn vẹn hơn, phát hiện hiệu ứng lề (side effect) xảy ra khi chạy các đơn thể cùng với nhau.

+ Kiểm thử không tăng tiến:

- Thời gian kiểm thử nhanh hơn vì có thể kiểm thử đồng thời các đơn thể một cách độc lập với nhau.

- Ở đầu giai đoạn kiểm thử đơn thể, có nhiều cơ hội để kiểm thử đồng thời các đơn thể, nhất là đối với các dự án lớn có nhiều người và nhiều đơn thể.

**g. Kiểm thử tích hợp từ trên xuống:**

+ Bắt đầu từ đơn thể cao nhất và dần tiến tới các đơn thể thấp hơn. Chỉ đơn thể cao nhất là đơn thể được kiểm thử độc lập. Sau đó những đơn thể thấp hơn được tích hợp từng cái một. Quá trình được lặp lại cho đến khi tất cả các đơn thể được tích hợp và kiểm thử.

+ Ư điểm:

- Sản phẩm được kiểm thử rất phù hợp vì kiểm thử tích hợp về cơ bản được thực hiện trong một môi trường gần giống với thực tế.

- Cơ bản có thể được thực hiện với thời gian ít hơn bởi vì đơn giản hơn.

- Thu gọn phạm vi lỗi dễ dàng hơn.

- Đơn thể quan trọng đang được kiểm thử trên mức ưu tiên, lỗi trong thiết kế lớn có thể được tìm thấy và sửa đầu tiên.

+ Nhược điểm:

- Chức năng cơ bản được kiểm thử vào cuối chu kỳ.
- Cần nhiều stub.
- Đơn thể ở mức thấp hơn sẻ được kiểm tra không đầy đủ.

+ Các bước:

- Bước 1: kiểm thử đơn thể đầu tiên, trên cùng của chương trình
    a. Viết các stub cho các đơn thể được đơn thể đầu tiên gọi.

    b. Tạo các testcase cho đơn thể đầu tiên, dữ liệu kiểm thử có thể được cung cấp từ các stub (dữ liệu được lưu trong các tập tin và được các stub đọc và cung cấp cho đơn thể đầu tiên).

- Bước 2: chọn đơn thể cần kiểm thử kế tiếp.
    a. Đơn thể cần kiểm thử kế tiếp là đơn thể được gọi từ đơn thể đã được kiểm thử.

    b. Chọn một trong các stub và thay thế stub này bằng đơn thể thật, sau đó kiểm thử đơn thể này.

    c. Có thể có nhiều chuỗi các đơn thể khác nhau.

+ Thứ tự ưu tiên:

- Các đơn thể nhập/ xuất được kiểm thử càng sớm càng tốt.

- Các đơn thể quan trọng (giải thuật phức tạp, giải thuật mới hoặc có thể xảy ra nhiều lỗi) được kiểm thử càng sớm càng tốt.

- Đơn thể nhập xuất sẻ được ưu tiên kiểm thử trước đơn thể quan trọng.

**h. Kiểm thử tích hợp từ dưới lên:**

+ Bắt đầu từ đơn thể thấp nhất hoặc trong cùng của ứng dụng, và dần di chuyển lên, tiến tới các đơn thể cao hơn của ứng dụng. Sự tích hợp này tiếp tục cho tới khi tất cả các đơn thể được tính hợp và toàn bộ ứng dụng được kiểm thử như một đơn vị duy nhất.

+ Ưu điểm:

- Nếu một lỗi lớn tồn tại ở đơn vị thấp nhất của chương trình, sẻ dễ dàng phát hiện ra nó, và những biện pháp đúng sẻ được thực hiện.

- Thư gọn phạm vi lỗi dễ dàng hơn.
- Không mất thời gian chờ tất cả các đơn thể được tích hợp.

+ Nhược điểm:

- Chương trình chính thực sự không tồn tại cho đến khi đơn thể cuối cùng được tích hợp vào kiểm thử, kết quả là các lỗ hổng về thiết kế ở cấp cao hơn sẻ được phát hiện chỉ ở cuối vòng đời.

- Đơn thể quan trọng của hệ thống có thể bị lỗi.
- Không giữ được nguyên mẫu đầu tiên của hệ thống.

+ Các bước:

- Bước 1: kiểm thử các đơn thể dưới cùng của chương trình, là đơn thể không gọi đơn thể khác.
    a. Các đơn thể này được kiểm tra tuần tự hoặc song song.

    b. Mỗi đơn thể cần kiểm thử cần có một driver: nhận dữ liệu kiểm thử, gọi đơn thể cần kiểm thử và hiển thị kế quả hoặc so sách kết quả hiện tại với kết quả mong muốn.

    c. Không cần phải viết nhiều driver khác nhau cho một đơn thể cần kiểm thử.

- Bước 2: chọn đơn thể cần kiểm thử kế tiếp.
    a. Đơn thể cần kiểm thử kế tiếp là đơn thể gọi trực tiếp tiếp các đơn thể đã được kiểm thử.

    b. Các đơn thể nhập/xuất được kiểm thử càng sớm càng tốt.

    c. Các đơn thể quan trọng (giải thuật phức tạp, giải thuật mới hoặc có thể xảy ra nhiều lỗi) được kiểm thử càng sớm càng tốt.

**i. Kiểm thử tích hợp Sandwich:**

+ Là sự kết hợp của cả hai cách tiếp cận từ dưới lên và từ trên xuống. Nó cũng được gọi là kiểm thử tích hợp lai hoặc kiểm thử tích hợp hỗn hợp. Trong kiểm thử tích hợp Sandwich, hệ thống dược tạo thành từ ba lớp:

- Một lớp ở giữa sẻ là mục tiêu của thử nghiệm.
- Một lớp bên trên lớp đích và một lớp bên dưới lớp đích thử nghiệm bắt đầu từ các lớp ngoài và hội tự ở lớp giữa.

+ Ư điểm:

- Các lớp trên cùng và dưới cùng có thể được kiểm tra song song.

+ Nhược điểm:

- Việc kiểm thử mở rộng các hệ thống con không được thực hiện trước khi tích hợp.

**3. Kiểm thử hộp đen.**

+ Kiểm thử hộp đen: trong kiểm thử hộp đen phần mềm được xem là một hộp đen và các test-case được xác định từ các đặc tả yêu cầu chức năng của thành phần phần mềm, không dựa vào mã nguồn của thành phần phần mềm.
+ Kỹ thuật phân chia lớp tương đương

- Định nghĩa: kỹ thuật phân chia lớp tương đương phân chia các giá trị của dữ liệu nhập thành các nhóm dữ liệu, mỗi nhóm dữ liệu là một lớp tương đương.

- Lớp tương đương: bao gồn các giá trị mà khi chạy một thành phần phần mềm với các giá trị này thì nó thực hiện cùng một hành vi.

- Khi chạy một thành phần phần mềm với một giá trị nào đó thuộc một lớp tương đương và bị lỗi sai, thì vẫn bị lỗi sai này với các giá trị khác thuộc cùng lớp tương đương này.

- Khi chạy một thành phần phần mềm với một giá trị nào đó thuộc một lớp tương đương và không bị lỗi sai, thì vẫn không bị lỗi sai này với các giá trị khác thuộc cùng lớp tương đương này.

+ Các trường hợp:

- Trường hợp 1: kiểm thử theo hợp đồng:
    a. Là các tiếp cận để thiết kế phần mềm, người thiết phần mềm phải xác định các đặc tả giao tiếp chính thức, chính xác và có thể kiểm chứng cho các thành phần phần mềm.

    b. Hợp đồng phát biểu các điều mà cả hai bên phải làm, không phụ thuộc vào cách thức thực hiện. Hợp đồng thường được xác định bởi các khẳng định và các khái niệm liên quan.

    c. Không cần xác định các lớp tương đương.

    d. Khăng định là một biểu thực luận lý liên quan đến một số thực thể của phần mềm và nêu ra đặc tính mà các thực thể này có thể đáp ứng ở những giai đoạn nào đó khi thực hiện phần mềm.

e. Có 3 loại khẳng định:
    i. Điều kiện trước: điều kiện phải được thỏa mãm trước khi thực hiện phương thức, nó liên quan đến trạng thái của hệ thông và các đối số được truyền cho phương thức.

    ii. Điều kiện sau: điều kiện phải được thỏa mãn sau khi thực hiện phương thức.

    iii. Bất biến: điều kiện phải được thỏa mãn ở mọi thời điểm gọi phương thức, được kiểm tra trước và sau khi thực hiện phương thức. Vi phạm một khăng định có thể cho thấy một lỗi sai của thành phần phần mềm.

- Trường hợp hai: kiểm thử phòng vệ:
    a. Là dạng thiết kế phòng vệ nhằm đảm bảo chức năng của thành phần phần mềm trong các trường hợp chưa biết trước, cách thiết kế này được sử dụng khi thành phần phần mềm được sử dụng sai.

    b. Là cách tiếp cận để cải tiến phần mềm và mã nguồn về phương diện:
        i. Giảm số lượng các lỗi sai của phần mềm.

        ii. Làm cho mã nguồn dễ đọc, dễ hiểu và giúp ích cho cho kiểm thử hộp trắng.

        iii. Làm cho phần mềm chạy theo một cách thức biết trước với dữ liệu nhập bất kỳ.

    c. Tuy nhiên có thể dẫn đến việc tồn tại mã nguồn xử lý các lỗi sai không thể xảy ra nhưng vẫn được thực hiện trong giời gian chạy, kích khởi rất nhiều trường hợp ngoại lệ. Điều này làm tăng thời gian chạy và chi phí bảo trì phần mềm.

d. Xác định các lớp tương đương cho các giá trị không hợp lệ.

+ Kỹ thuật sơ đồ chuyển trạng thái.

- Sơ đồ chuyển trạng thái biểu diễn các trạng thái của các đối tượng của một lớp và cho thấy sự thay đổi các trạng thái khi có các sự kiện xảy ra.

- Các thành phần của sơ đồ chuyển trạng thái.
  a. Trạng thái:
     i. Trạng thái là tình trạng mà hệ thống đang chờ một hoặc nhiều sự kiện xảy ra.

     ii. Sự kiện gây ra sự chuyển trạng thái và/hoặc khởi động các hành động.

     iii. Trạng thái được biểu diễn bởi các giá trị của một hoặc nhiều biến trong hệ thống.

     iv. Trạng thái được biểu diễn bởi một hình tròn bên trong ghi tên trạng thái.

  b. Chuyển trang thái: chuyển trạng thái biểu diễn sự thay đổi từ một trang thái sang một trạng thái khác do một sự kiện gây ra.

  c. Sự kiện:
     i. Là sự việc xảy ra và làm cho hệ thống bị thay đổi trạng thái.

     ii. Sự kiện được đưa vào hệ thống thông qua các giao tiếp của hệ thông, hoặc được tạo ra ở bên trong hệ thống.

     iii. Các sự kiện có thể liên quan với nhau hoặc độc lập với nhau.

iv. Khi một sự kiện xảy ra, hệ thống có thể thay đổi trọng thái hoặc giữ nguyên trạng thái và/hoặc thực hiện một hành động.

v. Sự kiện có thể có các tham số kèm theo.

vi. Sự kiện được biểu diễn bởi một tên là động từ ghi bên cạnh mũi tên chuyển trạng thái

d. Hành động:
   i. Là một hoạt động xảy ra khi chuyển trạng thái.

   ii. Làm cho hệ thống phải thực hiện một công việc nào đó hoặc tạo ra các kết xuất.

   iii. Hành động được biểu diễn bởi một lệnh đi sau dấu /.

**4. Quy trình phát triển phần mềm RUP**

+ Bắt đầu -> phát triển -> xây dựng -> chuyển giao:

- Bắt đầu:
  a. Định nghĩa phạm vi.
  b. Xác định tính khả thi.
  c. Hiểu các yêu cầu của người sử dụng.
  d. Chuẩn bị kế hoạch phát triển phần mềm.
  e. Các yêu cầu về tài nguyên tương đối ít.
  f. Thời gian của giai đoạn này ngắn.
  g. Tập trung vào lập kế hoạch và phân tích.

- Phát triển:
  a. Chi tiết hóa các yêu cầu của người sử dụng.
  b. Thiết kế kiến trúc.
  c. Không cần nhiều tài nguyên.
  d. Thời gian của giai đoạn này tương đối dài.
  e. Tập trung vào phân tích và thiết kế.

- Xây dựng:
    a. Lập trình.
    b. Kiểm tra.
    c. Lập tài liệu lập trình.
    d. Cần nhiều tài nguyên.
    e. Thời gian của giai đoạn này là dài nhất.
    f. Tập trung và thực hiện các công việc.

- Chuyển giao:
    a. Triển khai hệ thống.
    b. Đào tạo và hỗ trợ người sử dụng.
    c. Cần nhiều tài nguyên.
    d. Thời gian của giai đoạn này ngắn.
    e. Tập trung vào cài đặt, đào tạo và hỗ trợ.

5. **Kiểm thử hộp trắng:**

+ Định nghia: kiểm thử hộp trắng, còn gọi là kiểm thử cấu trúc, phần mềm được xem là một hộp trắng và các test-case được xác định từ sự thực hiện của phần mềm.

+ Các kỹ thuật kiểm thử hộp trắng:

- Kiểm thử dòng điều khiển: sử dụng các cấu trúc điều khiển của chương trình để xây dựng các test-case.

- Kiểm thử dòng dữ liệu: tập trung vào các nơi mà các biến được gán giá trị và các nơi mà chúng được sử dụng trong chương trình.

+ Đường thi hành:

- Định nghĩa: danh sách có thứ tự các lệnh được thực hiện tương ứng với một lần chạy cụ thể của đơn vị phần mềm, bắt đầu từ điểm nhập cho đến điểm kết thực của đơn vị chương trình.

+ Đồ thị dòng điều khiển:

- Định nghĩa: mô tả các dòng điều khiển luận lý của các cấu trúc điều khiển được sử dụng trong chương trình.

- Ba cấu trúc điều khiển cơ bản:
    a. Cấu trúc tuần tự: thực hiện một chuỗi các phát biểu một cách tuần tự theo thứ tự xuất hiện của chúng trong chương trình.

    b. Cấu trúc điều kiện: thực hiện việc rẽ nhánh dựa vào một điều kiện.

    c. Cấu trúc lặp: thực hiện nhiều lần một hoặc nhiều phát biểu dựa vào một điều kiện.

- Đồ thị dòng điều khiển nhị phân là đồ thị dòng điều khiển chỉ chứa các nút quyết định có hai nhánh.

- Một đồ thị dòng điều khiển bất kỳ có thể được biến đổi thành đồ thị dòng điều khiển nhị phân.

- Một điều kiện phức hợp (bao gồn các toán tử luận lý AND, OR, NAND, NOR) có thể bao gồm các điều kiện đơn chỉ có một toán tử. Điều kiện đơn được gọi là vị từ.

- Nếu đồ thị dòng điều khiển nhị phân chỉ chứa các nút quyết định là vị từ thì được gọi là đồ thị dòng điều khiển cơ bản.

+ Đường độc lập:

- Đường độc lập là một đường thi hành có đi qua ít nhất một cạnh mà chúng chưa được đi qua trong các đường thi hành độc lập trước đó.

- Tập cơ sở của đồ thị dòng điều khiển bao gồm tất cả các đường độc lập.

- Độ phức tạp cyclomatic M của một đoạn mã lệnh là số lượng các đường độc lập của đồ thị dòng điều khiển của đoạn mã này.

+ kiểm thử dòng dữ liệu:

- Là kiểm thử bằng cách chọn các đường thi hành thông qua dòng điều khiển của chương trình sao cho chúng bao gồm chuỗi các sự kiện có liên quan đến trạng thái của các biến hoặc các đối tượng dữ liệu.

- Các mức kiểm thử dòng dữ liệu:
    a. Kiểm thử dòng dữ liệu tĩnh.
        i. Xác định các sai sót tiềm ẩn, được gọi là sự bất thường của dòng dữ liệu.
        ii. Phân tích mã nguồn.
        iii. Không chạy mã nguồn.

    b. Kiểm thử dòng dữ liệu động.
        i. Cần chạy chương trình thật.
        ii. Xác định và thực hiện các đường thi hành dựa vào các tiêu chí của dòng dữ liệu.

- Tầm vực của biến:
    a. Là một đoạn chương trình mà biến này được sử dụng.

    b. Tầm vực toàn cục là đoạn chương trình nằm ngoài tất cả các hàm.

    c. Tầm vực cục bộ là một khối phát biểu bao gồm các phát biểu được được ghi trong hai dấu {} có thể là thân hàm, khối nằm trong hàm...

- Đồ thị dòng dữ liệu: là một đồ thị biểu diễn các chuỗi trạng thái của các biến trong chương trình này.

## 6. Kiểm thử phần mềm là gì:

+ Mục đích:

- Kiểm tra xem phần mềm có thực hiện đúng các yêu cầu của người sử dụng.

+ Cho thấy:

- Phần mềm thực hiện đúng các yêu cầu.
- Phát hiện các khiếm khuyết trước khi đưa vào sử dụng.

+ Là:

- Quy trình chứng minh  phần mềm không có lỗi sai.
- Quy trình chạy phần mềm để tìm các lỗi sai.

+ Kiểm tra kết quả chạy thử để:

- Phát hiện lỗi sai.
- Phát hiện các bất thường.
- Có thông tin về các thuộc tính phi chức năng của chương trình.

## 7. Mục tiêu của kiểm thử phần mềm:

+ Kiểm tra kiểm chứng.

- Chứng minh cho nhà phát triển và khách hàng răng phần mềm thỏa mãn các yêu cầu của nó.

- Bằng cách sử dụng một tập các test-case để phản ánh hệ thống thực hiện đúng như mong muốn.

- Một kiểm tra thành công cho thấy hệ thống hoạt động như mong muốn.

+ Kiểm tra khiếm khuyết.

- Phát hiện các lỗi sai hoặc các thiếu sót trong phần mềm mà cách hoạt động của nó bị sai hoặc không đúng với đặc tả của nó.

- Thiết kế các test-case để phát hiện các khiếm khuyết. Các test-case có thể không rõ ràng và không phản ánh cách hệ thống thường được sử dụng.

- Một kiểm tra thành công là kiểm tra làm cho hệ thống thực hiện sai và bộc lộ khiếm khuyết của hệ thống.

- Trừ khử hoạt động không mong muốn: phần mềm không chạy, các tương tác không mong muốn với các hệ thống khác, tính toán bị sai, dữ liệu bị sai.

**8. Use-case, test-case.**

+ Use-case:

- Là kỹ thuật dùng để mô tả sự tương tác giữa người dùng và hệ thống với nhau trong một môi trường cụ thể vì một mục đích cụ thể. Cũng mô tả các yêu cầu đối với hệ thống.

- Mỗi use-case là một đơn vị công việc riêng lẽ.

- Cung cấp một cái nhìn cấp cao của người ngoài hệ thống về một hành vi có thể nhận thấy được của hệ thống.

- Mô tả cách thức người dùng bên ngoài tương tác với hệ thống để đạt được mục tiêu nào đó. Một hoặc nhiều kịch bản có thể được tạo ra từ mỗi use-case, tương ứng với chi tiết về mỗi cách thức đạt được mục tiêu nào đó.

+ Test-case:

- Là bộ dữ liệu cho một lần chạy chương trình.

- Mô tả một dữ liệu đầu vào (input), hành động (action), hoặc một sự kiện (event) và kết quả truy vấn (expected response).

- Nhằm kiểm tra từng chức năng của ứng dụng phần mềm hoạt động đúng hay không.

**9. Thiết kế test-case.**

+ Dựa vào source code của function/code segment
+ Xây dựng đồ thị dòng điều khiển.
+ Tìm tập các đường đi độc lập tuyến tính – tập cơ sở.
+ Tìm tập các đường thi hành.
+ Xây dựng các testcase từ các đường thi hành.

**10. Quy trình kiểm thử phần mềm là gì.**

+ Lập kế hoạch và kiểm soát việc kiểm thử:
    - Nhằm chỉ định và mô tả các loại kiểm tra sẻ được triển khai và thực hiện gồm:
        a. Lập kế hoạch: xác định phạm vi, rủi ro, mục đích, cách tiếp cận, chiến lược, nguồn lực, lên lịch...kiểm thử.

        b. Kiểm soát kiểm thử: đo lường, phân tích kết quả kiểm thử, theo dõi tiến độ kiểm thử, cung cấp thông tin kiểm thử, tiến hành khắc phụ lỗi và đưa ra các quyết định.

+ Phân tích và thiết kế:
    - Nhằm chỉ định các test-case và các bước kiểm tra chi tiết cho mỗi phiên.

    - Rà soát các yêu cầu cần thiết trước khi tiến hành kiểm thử: tài liệu, đặc tả...

    - Xác định các điều kiện kiểm thử, thiết kế test-case, chuẩn bị môi trường...

+ Thực thi kiểm thử:
    - Nhằm thực hiện các bước kiểm tra đã thiết kế và ghi nhận kết quả.
    - Thực hiện test: chuẩn bị test data, thiết kế và phân loại các trường hợp kiểm thử dựa vào độ ưu tiên, tự động hóa các trường hợp kiểm thử nếu thấy cần thiết.

- Chạy test: chạy các test-case theo các bước đã định trước đó, chạy các test-case bị failed trước để xác nhận test-case đó đã được sửa, so sánh kết quả thực thi và kết quả mong đợi, đánh giá, viết báo cáo.

+ Đánh giá kết quả thực thi và báo cáo kết quả:

- Đánh giá toàn bộ quá trình kiểm thử bao gồm xem xét và đánh giá kiểm thử lỗi, chỉ định các yêu cầu thay đổi và tính toán số liệu liên quan đến quá trình kiểm thử.

- Các tiêu chí: số lượng test-case tối đa được thực thi passed, tỷ lệ lỗi giảm xuống dưới mức nhất định...

- Việc kiểm thử kết thúc khi: đối chiếu kết quả thực thi test-case so với các tiêu chí kết thúc kiểm thử được định ra trong lúc lập kế hoạch kiểm thử -> xem xét có cần test thêm hay điều chỉnh các tiêu chí kết thúc trong kế hoạch. Viết báo cáo tóm tắt hoạt động kiểm thử cũng như kết quả kiểm thử cho các bên liên quan.

+ Đóng hoạt động kiểm thử:

- Kết thúc hoạt động kiểm thử và phần mềm sẵn sàng được giao cho khách hàng.

- Ngoài ra còn có các trường hợp đóng kiểm thử: khi tất cả các thông tin đã được thu thập đầy đủ cho hoạt động kiểm thử, khi dự án bị hủy bỏ, khi một mục tiêu nào đó đạt được, khi hoạt động bảo trì hay cập nhật hệ thống hoàn tất.

- Hoạt động đóng kiểm thử: kiểm tra lại đã giao đầy đủ cho khách hàng những phần đã cam kết từ đầu, kiểm tra lại các lỗi nghiêm trọng đã được sửa. Đóng gói tài liệu, kịch bản, môi trường... để dành cho dự án sau này. Đánh giá quá trình kiểm thử và rút ra bài học kinh nghiệm.

**11. Các nguyên tắc kiểm thử phần mềm:**

+ Thiết kế đầy đủ các test-case:
    - Dữ liệu nhập hợp lệ.
    - Dữ liệu nhập không hợp lệ.

+ Kiểm tra kết quả:
    - Kết quả mong muốn.
    - Kết quả không mong muốn.

+ Việc kiểm thử đòi hỏi tính độc lập: người lập trình nên tránh việc kiểm thử chương trình của mình viết.
+ Kiểm thử nên bắt đầu từ thanh phần đơn giản, rồi đến các thành phần phức tạp.
+ Nên lập kế hoạch và quy trình kiểm thử trước khi bắt đầu.

**12. Hạn chế của kiểm thử phần mềm:**

+ Các đặc tả phần mềm có thể chưa đúng.
+ Công cụ kiểm thử có thể chưa chắc đúng.
+ Không có công cụ kiểm thử nào tích hợp cho mọi phần mềm.
+ Kỹ sư kiểm thử có thể chưa hiểu đầy đủ về phần mềm.
+ Không thể thực hiện kiểm thử phần mềm một cách đầy đủ.

**13. Giải thuật là gì.**

+ Là một tập hữu hạn hay một dãy các quy tắc chặt chẽ của các chỉ thị, phương cách hay 1 trình tự các thao tác trên một đối tượng cụ thể được xác định và định nghĩa rõ ràng cho việc hoàn tất một số sự việc từ một trạng thái ban đầu cho trước; khi các chỉ thị này được áp dụng triệt để thì sẽ dẫn đến kết quả sau cùng như đã dự đoán trước.

+ Nói cách khác, thuật toán là một bộ các quy tắc hay quy trình cụ thể nhằm giải quyết một vấn đề nào đó trong một số bước hữu hạn, hoặc nhằm cung cấp một kết quả từ một tập hợp của các dữ kiện đưa vào.