NGÂN HÀNG CÂU HỎI SQA - PTIT

ادعمني ارجوك حبي يا أجلك **من**
أكبر الله معك الله يكون قد الله اراده

Pregunta 1.1: ¿Qué es un error de software? ¿Razón?

- Lỗi phần mềm(Software Error) là các phần code sai do lôi cú pháp, logic hoăc lỗi do phân tích, thiết kế
- Nguyên nhân gây ra lỗi:
    1. Lỗi khi định nghĩa yêu cầu
    2. Quan hệ Client-developer tồi
    3. Sai phạm có chủ ý với yêu cầu phần mềm
    4. Lỗi thiết kế logic
    5. Lỗi lập trình
    6. Không tuân thủ các hướng dẫn viết tài liệu và code
    7. Thiếu sót của quá trình kiểm thử
    8. Lỗi giao diện người dùng và thủ tục
    9. Lỗi tài liệu

Pregunta 1.4: Kể ra các độ đo đặc trưng chất lượng chính của McCall? Giải thích nội dung của nó?

McCall có 11 tiêu chí; chia thành các nhóm.

<u>– Tiêu chí vận hành sản phẩm</u>

+ Tính đúng đắn – Correctness : Đặc tả về độ chính sác, tính toàn vẹn, thời gian của outputs.
+ Tính tin cậy – Reliability : Định ra tỉ lệ lỗi cho từng chức năng hoặc cả hệ thống
+ Tính hiệu quả - Efficiency : Tài ng phần cứng cần để thực hiện các chức năng của phần mềm
+ Tính toàn vẹn – Integrity : Bảo mật hệ thống, ngăn truy cập trái phép
+ Tính khả dụng - Usability : Tính dễ học, dễ dùng, hiệu quả.

– Tiêu chí sửa đổi sản phẩm

+ Tính bảo trì được – Maintainability : Mức công sức cần đề tìm nguyên nhân+ sửa + xác nhận đã sửa đc failures.(Liên quan đến cấu trúc modul, kiến trúc , thiết kế và các tài liệu)

+ Tính linh hoạt – Flexibility : Bảo trì cải tiến dễ dàng.

+ Tính kiểm thử được – Testability : Có lưu lại kq trung gian để hỗ trợ test? Có tạo file log, backup?

– Tiêu chí chuyển giao sản phẩm

+ Khả năng di động – Portability : Cài trong môi trường mới (phần cứng khác, hệ điều hành khác,…) mà vẫn duy trì môi trường cũ.

+ Khả năng tái sử dụng – Reusability : Có thể tái sử dụng các phần của phần mềm cho ứ/dụng khác

+ Khả năng tương thích – Interoperability : phần mềm có cần interface với các hệ thống đã có

Pregunta 1.6: Trình bày kỹ thuật Walkthrough

• **Walkthrough:** Kỹ thuật đánh giá không chính thức(nên ko có ng quản lý, giám đốc dự án). Những người tham gia phải xem tài liệu trước cuộc họp (ít nhất vài ngày). Tác giả giải thích tài liệu/ sản phẩm đó cho nhóm (tác giả, điều phối viên, giám định viên, đại diện ng dùng, chuyên gia bảo trì).

+ Mọi người sẽ đặt Pregunta hoặc cho ý kiến bổ sung về một số lĩnh vực để bảo đảm chất lượng kỹ thuật của tài liệu hoặc sản phẩm.

+ Buổi giám định có thể xảy ra vào bất kì lúc nào và bất kì đâu trong việc phát triển sản phẩm phần mềm. Mục đích chính của họp giám định chỉ là để tìm lỗi nhanh, ko tìm giải pháp. Sau giám định, tác giả của phải làm lại sửa mọi lỗi.

Pregunta 1.7: Trình bày kỹ thuật Inspection

• **Inspection:** Kỹ thuật đánh giá chính thức. Tài liệu, sản phẩm... được những người không phải là tác giả hoặc trực tiếp liên quan(Ngươi kiểm duyệt, tác giả, tester, thiết kế, coder) kiểm tra một cách chi tiết để phát hiện lỗi, các vi phạm tiêu chuẩn, hoặc các vấn đề khác (nếu có).

+ Về cơ bản, nó được tổ chức và thực hiện chặt chẽ hơn walkthrough. Vai trò của những người tham gia được phân định rõ ràng. Tài liệu chuẩn bị cho việc xem xét được chuẩn bị trước chu đáo.

+ Quá trình duyệt thảo bắt đầu sau giai đoạn code và unit test. Sau buổi họp các lỗi tìm đc sẽ đc sửa lại, rồi đem ra duyệt thảo lại cho đến khi đạt tiêu chuẩn mới kết thúc quá trình này.

Pregunta 1.10: Trình bày tóm tắt SQA trong tiêu chuẩn IEEE std1028

Chất lượng phần mềm là:

(1) Mức độ mà một hệ thống, thành phần, hay tiến trình đáp ứng được đặc tả yêu cầu

(2) Mức độ mà một hệ thống, thành phần, hay tiến trình đáp ứng được nhu cầu/mong muốn của khách hàng/người dùng.

- Lập kế hoạch và cài đặt một cách hệ thống!
- Chỉ ra tiến độ và và truyền tải sự tin cậy của phần mềm đang phát triển
- Với tiến trình phát triển phần mềm một phương pháp luận; một cách thức để làm;
- Với đặc tả yêu cầu kỹ thuật phải có.

- SQA bao gồm cả tiến trình phát triển và có thể cả bảo trì dài hạn. Do vậy, ta cần xem xét vấn đề về chất lượng cho cả phát triển và bảo trì trong SQA. Hành động SQA phải bao gồm cả lập lịch và lập ngân sách.
- SQA phải chỉ ra các vấn đề nảy sinh khi không đáp ứng được ràng buộc thời gian– bỏ bớt chức năng? Ràng buộc ngân sách có thể thoả hiệp được khi nguồn lực được phân bổ bị là không đủ cho phát triển và/hoặc bảo trì.

SQA là: "Tập các hoạt động có hệ thống cung cấp bằng chứng về khả năng của qui trình phần mềm tạo ra sản phẩm phần mềm khớp với việc sử dụng. Do đó hội tụ của SQA là giám sát liên tục trong toàn thể vòng đời phát triển phần mềm để đảm bảo chất lượng của sản phẩm được chuyển giao. Điều này yêu cầu giám sát cả qui trình và sản phẩm. Trong đảm bảo qui trình, SQA cung cấp việc quản lí với phản hồi khách quan liên quan tới tuân thủ các kế hoạch, thủ tục, chuẩn và phân tích đã được chấp thuận. Các hoạt động đảm bảo sản phẩm hội tụ vào mức độ thay đổi của chất lượng sản phẩm bên trong từng pha của vòng đời, như yêu cầu, thiết kế, viết mã và kế hoạch kiểm thử. Mục tiêu là nhận diện và khử bỏ khiếm khuyết trong toàn bộ vòng đời sớm nhất có thể được, do vậy giảm chi phí kiểm thử và bảo trì.

Pregunta 1.11: Trình bày các mức tiêu chuẩn trong CMM?
1. Khởi đầu: Quy trình sản xuất phần mềm có đặc điểm tự phát, thành công chỉ dựa vào nỗ lực của các cá nhân hoặc tài năng.

2. Lặp: Các quy trình quản lý dự án cơ bản được thiết lập để kiểm soát chi phí, kế hoạch và khối lượng hoàn thành. Nguyên lý quy trình cơ bản được hình thành nhằm đạt được thành công như những phần mềm tương tự.

3. Xác lập: Quy trình phần mềm cho các hoạt động quản lý cũng như sản xuất được tài liệu hóa, chuẩn hóa và tích hợp vào quy trình phần mềm chuẩn của nhà sản xuất. Các dự án sử dụng quy trình phần mềm hiệu chỉnh được phê duyệt dựa trên quy trình chuẩn của nhà sản xuất để phát triển và bảo trì sản phẩm phần mềm.

4. Kiểm soát: Thực hiện đo lường chi tiết quy trình phần mềm và chất lượng sản phẩm. Cả quy trình sản xuất và sản phẩm phần mềm được kiểm soát theo định lượng.

5. Tối ưu: Quy trình liên tục được cải tiến dựa trên những ý kiến phản hồi từ việc sử dụng quy trình, thí điểm những ý tưởng quản lý và công nghệ mới.

- CMMI viết tắt cho Capability Maturity Model Integration - Mô hình trưởng thành năng lực tích hợp - và là khuôn khổ cho cải tiến qui trình phần mềm. Nó dựa trên khái niệm về các thực hành tốt nhất về kĩ nghệ phần mềm và giải thích kỉ luật mà các công ty có thể dùng để cải tiến các qui trình của họ.

- CMM bao gồm 5 levels và 18 KPAs (Vùng quy trình quan trọng - Key Process Area).

Level 1: Khởi đầu (lộn xộn, không theo chuẩn) không có KPAs nào cả
Level 2: Lặp (quản lý dự án, tuân thủ quy trình) có 6 KPAs
Level 3: Xác lập (thể chế hóa) có 7 KPAs
Level 4: Kiểm soát (định lượng) có 2 KPAs
Level 5: Tối ưu (cải tiến quy trình) có 3 KPAs

Level 1: Initial(Ban đầu):

Level 1 là bước khởi đầu của CMM, mọi doanh nghiệp, công ty phần mềm, cá nhóm, cá nhân đều có thể đạt được. Ở lever này CMM chưa yêu cầu bất kỳ tính năng nào.

Ví dụ: không yêu cầu quy trình, không yêu cầu con người, miễn là cá nhân, nhóm, doanh nghiệp… đều làm về phầm mềm đều có thể đạt tới CMM này.

Đặc điểm của mức 1:

- Hành chính: Các hoạt động của lực lượng lao động được quan tâm hàng đầu nhưng được thực hiện một cách vỗi vã hấp tấp
- Không thống nhất: Đào tạo quản lý nhân lực nhỏ lẻ chủ yếu dựa vào kinh nghiệp cá nhân
- Quy trách nhiệm: Người quản lý mong bộ phận nhân sự điều hành và kiểm sóat các hoạt động của lực lượng lao động
- Quan liêu: Các hoạt động của lực lượng lao động được đáp ứng ngay mà không cần phân tích ảnh hưởng
- Doanh số thường xuyên thay đổi: Nhân viên không trung thành với tổ chức

Level 2: Repeatable (được quản lý):

Mục tiêu(Goal): các hoạt động và những đề xuất của một dự án phần mềm phải được lên kế hoạch và viết tài liệu đầy đủ

Có 6 KPA (Key Process Area) nó bao gồm như sau
- Requirement Management (Lấy yêu cầu khách hàng, quản lý các yêu cầu đó)
- Software Project Planning (Lập các kế hoạch cho dự án)
- Software Project Tracking (Theo dõi kiểm tra tiến độ dự án)
- Software SubContract Managent (Quản trị hợp đồng phụ phần mềm)
- Software Quality Assurance (Đảm bảo chất lượng sản phẩm)
- Software Configuration Management (Quản trị cấu hình sản phẩm=> đúng yêu cầu của khách hàng không)

Các KPA(Key Process Areas) của nó chú trọng tới các thành phần sau :
+ Chế độ đãi ngộ
+ Đào tạo
+ Quản lý thành tích
+ Phân công lao động
+ Thông tin giao tiếp
+ Môi trường làm việc

Để đạt được Level 2 thì người quản lý phải thiết lập được các nguyên tắc cơ bản và quản lý các hoạt động diễn ra. Họ có trách nhiệm quản lý đội ngũ của mình

Level 3: Defined (được định ra)

- Các vùng tiến trình chủ chốt ở mức 3 nhằm vào cả hai vấn đề về dự án và tổ chức, vì một tổ chức (công ty) tạo nên cấu trúc hạ tầng thể chế các quá trình quản lý và sản xuất phần mềm hiệu quả qua tất cả các dự án
- KPA chú trọng tới các yếu tố sau :

+ Văn hóa cá thể
+ Công việc dựa vào kỹ năng
+ Phát triển sự nghiệp
+ Hoạch định nhân sự
+ Phân tích kiến thức và kỹ năng

Để đạt được level 3 thì người quản lý phải biến đổi cải tiến các hoạt động đang diễn ra, cải tiến môi trường làm việc.

Level 4: Managed (được quản lý định lượng)

Các vùng tiến trình chủ yếu ở mức 4 tập trung vào thiết lập hiểu biết định lượng của cả quá trình sản xuất phần mềm và các sản phẩm phần mềm đang được xây dựng. Đó là Quản lý quá trình định lượng (Quantitative Process Management) và Quản lý chất lượng phần mềm (Software Quality Management)

Các KPA của level 4 chú trọng tới:

+ Chuẩn hóa thành tích trong tổ chức
+ Quản lý năng lực tổ chức
+ Công việc dựa vào cách làm việc theo nhóm
+ Xây dựng đội ngũ chuyên nghiệp
+ Cố vấn

Để đạt được level 4 thì phải đo lường và chuẩn hóa. Đo lường hiệu quả đáp ứng công việc, chuẩn hóac phát triển các kỹ năng, năng lực cốt lõi

Level 4 này sẽ chú trọng vào những người đứng đầu của một công ty, họ có khả năng quản lý các công việc như thế nào

Level 5: Optimising (tối ưu)

Các vùng tiến trình chủ yếu ở mức 5 bao trùm các vấn mà cả tổ chức và dự án phải nhắm tới để thực hiện hoàn thiện quá trình sản xuất phần mềm liên tục, đo đếm được. Đó là Phòng ngừa lỗi (Defect Prevention), Quản trị thay đổi công nghệ (Technology Change Management), và Quản trị thay đổi quá trình (Process Change Management)

Để đạt được Level 5 thì doanh nghiệp đó phải liên tục cải tiến hoạt động tổ chức, tìm kiếm các phương pháp đổi mới để nâng cao năng lực làm việc của lực lượng lao động trong tổ chức, hỗ trợ các nhân phát triển sở trường chuyên môn.

Chú trọng vào việc quản lý, phát triển năng lực của nhân viên, Huấn luyện nhân viên trở thành các chuyên gia

Pregunta 1.12: Mục tiêu của SQA là gì? Các hoạt động chính đảm bảo chất lượng phần mềm là những hoạt động nào?

- Đảm bảo chất lượng phần mềm là các hoạt động nhằm mục tiêu là sản xuất ra phần mềm có chất lượng cao.

- Mục tiêu của hoạt động SQA trong phát triển phần mềm.
  - Đảm bảo một mức độ tin cậy chấp nhận được là phần mềm sẽ tuân thủ các yêu cầu kỹ thuật về chức năng.
  - Đảm bảo một mức độ tin cậy chấp nhận được là phần mềm sẽ tuân thủ các yêu cầu quản lý về thời gian và tài chính.
  - Khởi đầu và quản lý các hoạt động để phát triển phần mềm và các hoạt động SQA được cải thiện và đạt hiệu
    quả cao hơn.
- Mục tiêu của hoạt động SQA trong bảo trì phần mềm
  - Đảm bảo một mức độ tin cậy chấp nhận được là các hoạt động bảo trì phần mềm sẽ tuân thủ các yêu cầu kỹ thuật về chức năng.
  - Đảm bảo một mức độ tin cậy chấp nhận được là các hoạt động bảo trì phần mềm sẽ tuân thủ các yêu cầu quản lý về thời gian và tài chính.
  - Khởi đầu và quản lý các hoạt động để bảo trì phần mềm và hoạt động SQA được cải tiến hiệu quả.

- Có 7 hoạt động chính:

(1) Áp dụng công nghệ kĩ thuật hiệu quả (phương pháp, công cụ)
(2) Tiến hành rà soát kỹ thuật chính thức
(3) Thực hiện kiểm thử nhiều tầng
(4) Tuân theo các chuẩn phát triển
(5) Kiểm soát tài liệu PM và thay đổi của chúng
(6) Thực hiện đo lường
(7) Báo cáo và bảo quản lý các báo cáo.

Pregunta 1.13: Khảo sát nhu cầu SQA gồm những nội dung gì? Nhằm trả lời các Pregunta gì?

- Gồm ba nội dung nhằm trả lời ba Pregunta

+ Kiểm kê các chính sách SQA: chính sách, thủ tục, chuẩn nào đã có trong các pha phát triển?
+ Đánh giá vai trò của kỹ nghệ phần mềm, bảo đảm chất lượng trong tổ chức hiện tại có quyền lực đến đâu?

+ Đánh giá mối quan hệ SQA: Giao diện chức năng giữa SQA với các đơn vị khác như thế nào? Với các người thực hiện rà soát kỹ thuật chính thức, quản lý cấu hình và thử nghiệm.
- Nếu có nhu cầu SQA thì cần phải tiến hành đánh giá cẩn thận bằng quy tắc bỏ phiếu.

Pregunta 1.15: Ca kiểm thử là cái gì? Mục tiêu thiết kế ca kiểm thử?
- Một ca kiểm thử (test case) trong công nghệ phần mềm là một tập hợp các điều kiện hay các biến để theo đó một thử nghiệm sẽ xác định xem một ứng dụng hoặc hệ thống phần mềm đang làm việc một cách chính xác hay không.
- Mục đích:
+ Muốn tìm ra được nhiều sai nhất với nỗ lực và thời gian là nhỏ nhất.
+ Chứng minh được sự tồn tại của lỗi
+ Không chứng minh được sự không có lỗi

Pregunta 1.16: Kiểm thử hộp trắng là gì? Nêu các đặc trưng của nó?
- Là hình thức kiểm thử mà kiểm thử viên biết được các cấu trúc bên trong của chương trình (mã nguồn, xử lý dữ liệu, …). Việc kiểm thử được dựa trên các phân tích về cấu trúc bên trong của thành phần/hệ thống.
Kiểm tra mã nguồn các chi tiết thủ tục (thuật toán), các con đường logic (luồng điều khiển), các trạng thái của chương trình (dữ liệu)

- *Đặc trưng*:
+ Kiểm thử hộp trắng dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu bên trong của đơn vị phần mềm cần kiểm thử để xác địnhđơn vị phần mềm đó có thực hiện đúng không.
+ Người kiểm thử hộp trắng phải có kỹ năng, kiến thức nhất định để có thể hiểu chi tiết về đoạn code cần kiểm thử.
+ Thường tốn rất nhiều thời gian và công sức nếu mức độ kiểm thử được nâng lên ở cấp kiểm thử tích hợp hay kiểm thử hệ thống.
+ Do đó kỹ thuật này chủ yếu được dùng để kiểm thử đơn vị. Trong lập trình hướng đối tượng, kiểm thử đơn vị là kiểm thử từng tác vụ của 1 class chức năng nào đó.
+ Có 2 hoạt động kiểm thử hộp trắng: Kiểm thử luồng điều khiển và kiểm thử dòng dữ liệu.

Pregunta 1.17: Kiểm thử hộp đen là gì? Nêu các đặc trưng của nó?
- Kiểm thử hộp đen (Black-box Testing): Là hình thức kiểm thử mà kiểm thử viên không cần biết đến cách thức hoạt động, mã nguồn, xử lý dữ liệu bên trong 1 thành phần/hệ thống.
Công việc cần làm là nhập dữ liệu đầu vào (input) và kiểm tra kết quả trả về có đúng như mong muốn hay không.

- *Đặc trưng:*
+ Cơ sở: đặc tả, các điều kiện vào/ra và cấu trúc dữ liệu
+ Nhằm thuyết minh các chức năng phần mềm đủ và vận hành đúng
+ Thực hiện các phép thử qua giao diện

+ Thường phát hiện các lỗi đặc tả yêu cầu, thiết kế
+ Dễ dàng thực hiện
+ Chi phí thấp
+ Được sử dụng để kiểm thử phần mềm tại mức: modul, tích hợp, hàm, hệ thống.
+ Đơn giản hoá kiểm thử tại các mức độ được đánh giá là khó kiểm thử
+ Khó đánh giá còn bộ giá trị nào chưa được kiểm thử hay không
+ Ít chú ý tới cấu trúc logic nội tại của nó

Pregunta 1.18: Có những loại công cụ tự động nào trợ giúp kiểm thử, mô tả nội dung của mỗi loại?

- Công cụ kiểm thử tự động (Testing Tools) được chia thành những nhóm chính: Design, GUI(Graphical User Interface), Load and Performance, Management, Implementation, Evaluation (Sự đánh giá), Static Analysis (Phân tích tĩnh)và ngoài ra còn có : Defect Tracking, Websites và Miscellaneous(Hỗn Hợp).

Test Design Tools

+ Các công cụ này giúp bạn quyết định "test" cần gì để được thực thi. Kiểm thử dữ liệu và kiểm thử trưởng hợp (Test data and test case) sinh ra(generators).
+ Test design tools giúp tạo các "Test case", hoặc số đầu vào test tối thiểu (là một phần của test case).
+ Có tổng cộng khoảng 15 Tools.

GUI Test Tools

+ Các công cụ này tự động thực thi các kiểm thử (Test) cho "products" với giao diện người dùng Graphic. Công cụ tự động kiểm tra Client/Server, bao gồm cả load testers .
+ GUI Testing là quá trinh kiểm thử giao diện người dùng của ứng dụng và phát hiện các chức năng chính xác của ứng dụng.

Load and Performance Tools

+ Load testing là quá trình đặt nhu cầu về hệ thống hoặc thiết bị và đo sự phản ứng của nó. Load testing được thực hiện để xác định hành vi của hệ thống theo 2 điều kiện tải bình thường và được mong đợi. Nó giúp để xác định khả năng hoạt động tối đa của ứng dụng cũng như bất kỳ vướng mắc và xác dịnh các yếu tố gây ra suy thoái.
+ Performance testing được thực hiện để xác định một hệ thống có đáp ứng được yêu cầu và ổn định trong khối công việc cụ thể. Nó cũng có thể phục vụ việc kiểm tra, đo lường, xác minh chất lượng các thuộc tính của hệ thống, chẳng hạn như khả năng mở rộng, độ tin cậy và sử dụng tài nguyên.

Test Management Tools

+ Test managerment tools được sử dụng để lưu trữ thông tin quá trình kiểm thử làm việc như thế nào, kế hoạch hoạt động kiểm thử và báo cáo tình hình hoạt động của đảm bảo chất lượng phần mềm.

Test Implementation Tools

+ Dụng cụ khác giúp bạn thực hiện các bài kiểm tra. Ví dụ, công cụ tự động tạo ra thói quen khi còn sơ khai ở đây, cũng như các công cụ mà cố gắng để làm cho thất bại rõ ràng hơn (máy phát điện khẳng định, vv)

Test Evaluation Tools

+ Công cụ giúp bạn đánh giá chất lượng của các bài kiểm tra của bạn. Các công cụ bảo hiểm mã đi đây.

Static Analysis Tools

Công cụ phân tích các chương trình mà không cần chạy chúng. Số liệu công cụ rơi vào thể loại này.

Pregunta 1.19: Đồ thị luồng điều khiển gồm những yếu tố nào? Đồ thị luồng điều khiển dùng để làm gì?

- Đồ thị dòng điều khiển là một đồ thị có hướng gồm các đỉnh tương ứng với các câu lệnh/nhóm câu lệnh và các cạnh là các dòng điều khiển giữa các câu lệnh/nhóm câu lệnh.
Những yếu tố trong đồ thị luồng điều khiển:
+ Điểm bắt đầu của đơn vị chương trình
+ Khối xử lý chứa các câu lệnh được khai báo hoặc tính toán.
+ Điểm quyết định ứng với các câu lệnh điều kiện trong các khối lệnh rẽ nhánh hoặc lặp.
+ Điểm nối ứng với các câu lệnh ngay sau các lệnh rẽ nhánh.
+ Điểm kết thúc ứng với điểm kết thúc của đơn vị chương trình

- Mục tiêu của phương pháp kiểm thử luồng điều khiển là đảm bảo mọi đường thi hành của đơn vị phần mềm cần kiểm thử đều chạy đúng. Rất tiếc trong thực tế, công sức và thời gian để đạt mục tiêu trên đây là rất lớn, ngay cả trên những đơn vị phần mềm nhỏ.

Pregunta 1.20: Đồ thị luồng dữ liệu gồm những yếu tố nào? Đồ thị luồng dữ liệu dùng để làm gì?

Định nghĩa:Đồ thị dòng dữ liệu của một chương trình/đơn vị chương trình là một đồ thị có hướng G = <N; E>, với:

- N là tập các đỉnh tương ứng với các câu lệnh def hoặc c-use của các biến được sử dụng trong đơn vị chương trình. Đồ thị G có hai đỉnh đặc biệt là đỉnh bắt đầu (tương ứng với lệnh def của các biến tham số) và đỉnh kết thúc đơn vị chương trình.
- E là tập các cạnh tương ứng với các câu lệnh p-use của các biến.

Một số khái niệm:

Def: là câu lệnh gán giá trị cho một biến.

Undef: khai báo biên nhưng chưa cấp giá trị cho nó.

Use: là câu lệnh sử dụng một biến (tính toán hoặc kiểm tra các điều kiện).

C-use: là câu lệnh sử dụng biến để tính toán giá trị của một biến khác .

P-use: là câu lệnh sử dụng biến trong các biểu thức điều kiện (câu lệnh rẽ nhánh, lặp,...) .

Lý do cần kiểm thử dòng dữ liệu:

• Cần chắc chắn biến được gán đúng giá trị, tức là chúng ta phải xác định được một đường đi của biến từ một điểm bắt đầu nơi nó được định nghĩa đến điểm mà biến đó được sử dụng.

• Ngay cả khi gán đúng giá trị cho biến thì các giá trị được sinh ra chưa chắc đã chính xác do tính toán hoặc các biểu thức điều kiện sai (biến được sử dụng sai).

Pregunta 1.21: Nêu các loại điều kiện trong cấu trúc điều khiển và cho ví dụ? Có những loại sai nào trong điều kiện khi kiểm thử?

- Điều kiện logic của cấu trúc điều khiển:

+ *Điều kiện đơn:* là một biến logic (Boolean) hoặc một biểu thức quan hệ, có thể có toán tử NOT (!) đứng trước, VD: NOT (a>b)

+ *Biểu thức quan hệ:* là một biểu thức có dạng E1 <op> E2, trong đó E1, E2 là các biểu thức số học và <op> là toán tử quan hệ có thể là một trong các dạng sau: <, <=, >, >=, = =, !=, ví dụ, a > b+1.

+ *Điều kiện phức:* gồm hai hay nhiều điều kiện đơn, toán tử logic AND (&&) hoặc OR (||) hoặc NOT (!) và các dấu ngoặc đơn "(" và ")". VD: (a > b + 1) AND (a <= max)

- Kiểu sai trong điều kiện logic kiểm thử:

+ Sai biến Bool

+ Sai toán tử Bool

+ Sai số hạng trong biểu thức toán tử Bool

+ Sai toán tử quan hệ

+ Sai biểu thức số học.

Pregunta 1.22: Chiến lược kiểm thử phân nhánh (Branch) nghĩa là gì? Yêu cầu đặt ra cho kiểm thử phân nhánh là gì?

- Kiểm thử phân nhánh là chiến lược kiểm thử từng điều kiện chương trình

- Kiểm thử nhánh: Với mỗi điều kiện kết hợp C thì các nhánh "true" và "false" của C và mỗi điều kiện đơn trong C phải được kiểm thử ít nhất 1 lần.

- Yêu cầu: Không chỉ phát hiện sai trong điều kiện đó mà còn phát hiện các sai khác trong chương trình.

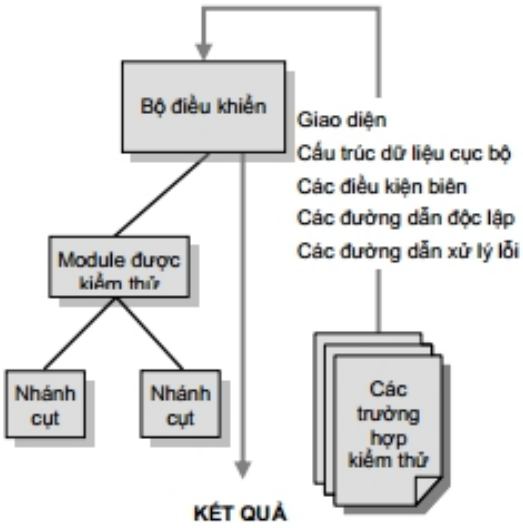
Pregunta 1.23: Kiểm thử đơn vị là gì? Hoạt động kiểm thử đơn vị gồm những nội dung gì? Nó liên quan đến những nhân tố nào?

- Kiểm thử đơn vị là quá trình kiểm thử các thành phần riêng biệt của hệ thống. Đây là quá trình kiểm thử khiếm khuyết vì vậy mục tiêu của nó là tìm ra lỗi trong các thành phần.

- Kiểm thử đơn vị có các nội dung:
+ Kiểm thử giao diện
+ Khám nghiệm cấu trúc dữ liệu cục bộ
+ Kiểm thử với các điều kiện biên
+ Các đường độc lập
+ Các đường xử lý sai

Kiểm thử đơn vị thường được xem như một phần phụ cho bước mã hoá. Sau khi mã nguồn đã được phát triển, được duyệt lại và được kiểm tra đúng cú pháp, thì bắt đầu thiết kế các trường hợp kiểm thử đơn vị. Việc xem lại thông tin thiết kế sẽhướng dẫn cho việc thiết lập trường hợp kiểm thử phù hợp nhằm phát hiện các loại lỗi trên. Mỗi trường hợp kiểm thử phải được gắn liền với tập các kết quả mong đợi.

Vì mỗi module không phải là một chương trình độc lập, nên phần mềm điều khiển và/hoặc nhánh cụt cần được phát triển cho mỗi kiểm thử đơn vị. Môi trường kiểm thử đơn vị được minh hoạ trong hình dưới đây.

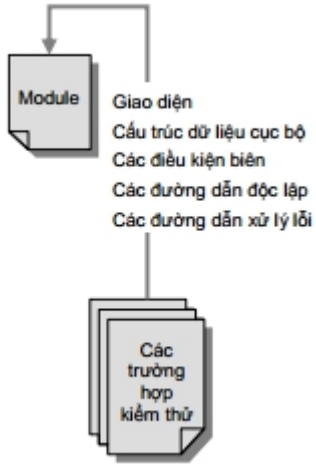


Các nhánh cụt dùng để thay thế các module cấp dưới được gọi bởi các module được kiểm thử.

Kiểm thử đơn vị được đơn giản hoá khi module có sự liên kết cao được thiết kế. Khi chỉ một chức năng được gọi bởi một module, số các trường hợp kiểm thửđược giảm xuống và các lỗi có thể dự đoán và phát hiện sớm hơn.

-Các nhân tố liên quan:
+Tham số
+vào ra

+Dữ liệu cục bộ: Cấu trúc dữ liệu cục bộ cho modul còn là nguồn gây lỗi chung



Các kiểm thử xuất hiện như một phần của kiểm thử đơn vị được minh hoạ trong hình dưới đây. Các kiểm thử nhằm phát hiện các lỗi trong các phạm vi của module bao gồm:

- Giao diện module,
- Cấu trúc dữ liệu cục bộ,
- Điều kiện biên,
- Đường dẫn độc lập,
- Đường dẫn xử lý lỗi.

Pregunta 1.24: Kiểm thử tích hợp là gì? Kiểm thử tích hợp thực hiện khi nào?

- Kiểm thử tích hợp là một kỹ thuật có tính hệ thống để xây dựng cấu trúc chương trình ngay khi đang tiến hành kiểm thử để phát hiện sai liên kết với giao diện. Kiểm thử tích hợp nhằm nhận được một phần hay toàn bộ hệ thống như mong đợi.

- Kiểm thử tích hợp đc thực hiện sau khi đã thực hiện kiểm thử đơn vị và ghép nối các đơn vị/thành phần phần mềm.

- Mục đích: tận dụng các modul đã kiểm thử đơn vị và xây dựng chương trình sao cho nó đảm bảo tuân theo thiết kế.

Phải kiểm thử tích hợp vì:
+ Dữ liệu có thể bị mất khi đi qua một giao diện
+ Một mođun có thể có một hiệu ứng bất lợi vô tình lên các modul khác
+ Các chức năng phụ khi kết hợp lại có thể không sinh ra chức năng chính mong muốn
+ Các điều không chính xác riêng rẽ có thể bị phóng đại đến mức không chấp nhận được.
+ Các cấu trúc dữ liệu toàn cục có thể để lộ ra các vấn đề.....

Ưu điểm:

+ Dễ dàng tìm ra các lỗi vào ngay giai đoạn đầu.
+ Dễ dàng khoanh vùng các lỗi (tích hợp n modules, sau đó n + 1modules).
+ Giảm việc sử dụng các stub và Driver

Pregunta 1.25: Nội dung chính của kiểm thử hệ thống?
- Hệ thống dựa vào máy tính do nhiều bên xây dựng, người phát triển phần mềm chỉ là một.
- Việc kiểm thử hệ thống dễ có nguy cơ "đổ lỗi cho nhau"
- Người phát triển phần mềm cần đoán trước các vấn đề giao diện có thể nảy ra
- Phát hiện các thiết kế đường xử lý sai thông qua kiểm thử tất cả các thông tin đến từ các phần tử khác của hệ thống
- Tiến hành một loạt kiểm thử mô phỏng các dữ liệu xấu hoặc các sai tiềm tàng khác tại giao diện phần mềm.
- Báo cáo các kết quả kiểm thử để làm chứng cớ phòng ngừa đổ lỗi cho nhau
- Những người tham gia vào trong việc hoạch định và thiết kế các kiểm thử hệ thống để bảo đảm rằng phần mềm được kiểm thử đầy đủ

Việc kiểm thử hệ thống thực tế là một loạt các bước kiểm thử khác nhau có mục đích chính là thử đầy đủ hệ thống dựa trên máy tính.

Pregunta 1.26: Khi nào nên dùng test tools? Ưu/nhược của việc dùng Test tools?
- Test tools được dùng khi:
+ Không đủ tài nguyên: Khi số lượng TestCase quá nhiều mà kiểm thử viên không thể hoàn tất trong thời gian cụ thể
+ Kiểm tra hồi quy: Nâng cấp phần mềm, kiểm tra lại các tính năng đã chạy tốt và những tính năng đã sửa. Tuy nhiên, việc này khó đảm bảo về mặt thời gian
+ Kiểm tra khả năng vận hành phần mềm trong môi trường đặc biệt:
  - Đo tốc độ trung bình xử lý một yêu cầu của Web server
  - Xác định số yêu cầu tối đa được xử lý bởi Web Server
  - Xác định cấu hình máy thấp nhất mà PM vẫn có thể hoạt động tốt

- Ưu điểm:
+ Giảm bớt công sức và thời gian thực hiện quá trình kiểm thử
+ Tăng độ tin cậy.
+ Giảm sự nhàm chán cho con người
+ Rèn luyện kỹ năng lập trình cho kiểm thử viên
+ Giảm chi phí cho tổng quá trình kiểm thử.

- Nhược điểm:
+ KTPM không cần can thiệp của KTV.
+ Giả lập tình huống khó có thể thực hiện bằng tay.
+ Mất chi phí tạo ra và bảo trì các script để thực hiện kiểm tra tự động.
+ Khó bảo trì, khó mở rộng các script.
+ Đòi hỏi KTV phải có kỹ năng tạo script kiểm tra tự động.
+ Không áp dụng được trong việc tìm lỗi mới của PM.

Pregunta 1.27: Kể tên một vài test tools cho kiểm thử chức năng, hiệu năng?

Test tools cho kiểm thử chức năng:

- QuickTest Professional: phần mềm kiểm soát việc kiểm thử tự động các chức năng của sản phẩm phần mềm cần kiểm thử.
- SoapUI: công cụ này được sử dụng chủ yếu cho kiểm thử chức năng các dịch vụ Web.
- Selenium IDE: có thể tạo được các test case ở mức đơn giản với Record-Playback.

Test tools cho kiểm thử hiệu năng:

- Load Runner: giả lập môi trường ảo gồm nhiều người dùng thực hiện các giao dịch cùng lúc nhằm giám sát các thông số kĩ thuật của phần mềm cần kiểm thử.
- Apache JMeter: kiểm thử khả năng chịu tải và hiệu năng cho các ứng dụng web và một số ứng dụng khác.
- Neo Load
- Appvance

Pregunta 1.28: Quy trình kiểm thử phần mềm nói chung?

• Phân tích yêu cầu:Kiểm thử thường sẽ bắt đầu lấy các yêu cầu trong các giai đoạn của vòng đời phát triển phần mềm. Trong giai đoạn thiết kế, các Tester làm việc với các nhà phát triển để xác định những khía cạnh của một thiết kế được kiểm chứng và những thông số được kiểm tra.

• Lập kế hoạch kiểm thử:Chiến lược kiểm thử, kế hoạch kiểm thử, kiểm thử sáng tạo… Và có một kế hoạch là cần thiết vì nhiều hoạt động sẽ được thực hiện trong thời gian kiểm thử.

• Kiểm thử phát triển:Các quy trình kiểm thử, các kịch bản, Test Case, các dữ liệu được sử dụng trong kiểm thử phần mềm.

• Kiểm thử thực hiện:Dựa trên các kế hoạch, các văn bản kiểm thử và các báo cáo bất kỳ lỗi nào tìm thấy cho nhóm phát triển.

• Kiểm thử báo cáo:Sau khi hoàn tất kiểm thử, các Tester tạo ra các số liệu và báo cáo cuối cùng về nỗ lực kiểm thử của họ và có sẵn sàng phát hành phần mềm hay không.

• Phân tích kết quả kiểm thử hoặc phân tích thiếu sót được thực hiện bởi đội ngũ phát triển kết hợp với khách hàng để đưa ra quyết định xem những thiếu sót gì cần phải được chuyển giao, cố định và từ bỏ (tức là tìm ra được phần mềm hoạt động chính xác) hoặc giải quyết sau.

• Test lại khiếm khuyết: Khi một khiếm khuyết đã được xử lý bởi đội ngũ phát triển, nó phải được kiểm tra lại bởi nhóm kiểm thử.

• Kiểm thử hồi quy:Người ta thường xây dựng một chương trình kiểm thử nhỏ là tập hợp của các bài kiểm tra cho mỗi tích hợp mới, sửa chữa hoặc cố định phần mềm, để đảm bảo rằng những cung cấp mới nhất đã không phá hủy bất cứ điều gì và toàn bộ phần mềm vẫn còn hoạt động một cách chính xác.

Pregunta 1.29: Test plan là gì, gồm những nội dung gì?

- Một kế hoạch kiểm thử dự án phần mềm (test plan) là một tài liệu mô tả các mục tiêu, phạm vi, phương pháp tiếp cận, và tập trung vào nỗ lực kiểm thử phần mềm. Quá trình chuẩn bị test plan là một cách hữu ích để suy nghĩ tới những nỗ lực cần thiết để xác nhận khả năng chấp nhận một sản phẩm phần mềm. Các tài liệu đã hoàn thành sẽ giúp mọi người bên ngoài nhóm test hiểu được 'tại sao' và

'như thế nào' chấp nhận sản phẩm. Nó cần phải hoàn hảo đủ để dùng được nhưng không đủ hoàn hảo vì không ai bên ngoài nhóm test sẽ đọc nó.

Nội dung:
chiến lược kiểm tra:

- Chiến lược kiểm tra đưa ra phương pháp tiếp cận để kiểm tra mục tiêu.
- Chiến lược kiểm tra bao gồm các kỹ thuật được áp dụng và điều kiện để biết khi nào việc kiểm tra hoàn thành.
  + Mô tả các kiểu kiểm tra dùng trong dự án.
  + Có thể liệt kê với mỗi kiểu kiểm tra tương ứng kiểm tra cho chức năng nào.
  + Việc kiểm có thể dừng khi nào.

Các kiểu kiểm tra

Mỗi kiểu kiểm tra phải bao gồm các điều kiện:
+ Kỹ thuật.
+ Điều kiện hoàn thành.
+ Các vấn đề đặc biệt liên quan.

* Kỹ thuật: Mô tả việc kiểm tra như thế nào, những gì sẽ được kiểm tra, các hoạt động chính được thực hiện trong quá trình kiểm tra và các phương pháp đánh giá kết quả.
* Điều kiện hoàn thành:
- Xác định chất lượng chương trình được chấp nhận.
- Thời điểm ktra hoàn tất.
* Các vấn đề đặc biệt: Các vấn đề gây ảnh hưởng đến việc kiểm tra.

1. Functional testing – kiểm tra chức năng
   a. Function testing – kiểm tra chức năng
   b. User interface testing – kiểm tra giao diện người sử dụng
   c. Data & database integrity testing – kiểm tra DL & tích hợp DL
   d. Business cycle testing – kiểm tra chu trình nghiệp vụ

2. Performance testing – kiểm tra hiệu xuất
   a. Performance profiling c. Stress testing
   b. Load testing d. Volume testing

4. Security & Access control testing – kiểm tra bảo mật & kiểm soát truy cập
5. Regression testing – kiểm tra hồi qui

Môi trường kiểm tra.

Tuỳ vào mỗi giai đoạn Unit test, Intergration test, System test, acceptance test sẽ ứnag với môi trờng kiểm tra nhất định. Từ đó xác định các yếu tố để xây dựng môi trường kiểm tra, sử dụng như môi trường thật hay tạo môi trường giả lập gần giống với môi trường chạy thật của chương trình.

- Khi test chạy chương trình bằng bản dịch hay chạy trên code. Thông thường, các giai đoạn System test, Acceptance test phải chạy trên bản dịch

- Với CSDL thì thông thường, từ Intergration test, ta phải thiết lập CSDL riêng và thiết lập các thông số cho CSDL gần giống hoặc giống hệt như khi chương trình sẽ chạy thật.

- Điều kiện về mạng: sẽ sử dụng mạng LAN hay Dial up… Thông thường, khi Unit test, có thể sử dụng mạng LAN nhưng khi System test trở đi thì nên sử dụng hệ thống đường truyền giống như hoặc gần giống như môi trường chạy thật.

- Mô hình sẽ cài đặt chương trình test: số lượng máy chủ, máy trạm; việc chia tách các server, các máy trạm, việc cài đặt các domain … Thông thường, trong Unit test có thể sử dụng viếc thiết lập như khi lập trình, nhưng khi System test trở đi, phải chú ý thiết lập sao cho gần giống mô hình sẽ chạy trong thực tế nhất.

Pregunta 2.1: Tại sao phải kiểm thử phần mềm? Mục tiêu kiểm thử là gì?

- Lý do cần kiểm thử phần mềm:
  • Kiểm thử phần mềm là yếu tố quyết định của SQA và khâu điển hình của rà soát đặc tả thiết kế và lập mã.
  • Muốn nhìn thấy phần mềm như là một phần tử của hệ thống hoạt động (xem sản phẩm)
  • Hạn chế chi phí phải trả cho các thất bại do lỗi gây ra sau này (hiệu quả)
  • Có kế hoạch tốt nâng cao chất lượng cho suốt quá trình phát triển (giải pháp)
- Tầm quan trọng của kiểm thử phần mềm:
  • Chi phí của kiểm thử chiếm: 40% tổng công sức phát triển ≥ 30% tổng thời gian phát triển
  • Với phần mềm ảnh hưởng tới sinh mạng chi phí có thể gấp từ 3 đến 5 lần tổng chi phí khác cộng lại.

- Mục tiêu trước mắt: Cố gắng tạo ra các ca kiểm thử để chỉ ra lỗi của phần mềm với chi phí(thời gian, chi phí) thấp nhất. Một ca kiểm thử thắng lợi là phải làm lộ ra khiếm khuyết, đồng thời mang lại các lợi ích phụ.
- Mục tiêu cuối cùng: có một chương trình tốt, chi phí ít

Pregunta 2.2: Giải thích sự khác nhau giữa *validation và verification.*

- Verification(xác minh)– tiến trình đánh giá một system hoặc component xem sản phẩm của một pha phát triển đã cho có thoả mãn điều kiện đưa ra ở đầu pha không.
(Kiểm tra điều có đang làm đúng hay không, ví dụ kiểm tra xây dựng sản phẩm đúng quy trình không.)

- Validation(xác nhận)– tiến trình đánh giá một system hoặc component trong hoặc sau development process để xác định xem nó có thảo mãn yêu cầu đã đặc tả hay không.
(Kiểm tra thực hiện có theo hướng đúng không, chẳng hạn kiểm tra thực hiện phần mềm theo đúng yêu cầu khách hàng không.)

- Khác nhau : Verification là tĩnh (Static) trong khi Validation là động (Dynamic).
VD: Verification phần mềm là kiểm thử từng dòng mã, từng hàm. Với Validation, chạy phần mềm và tìm lỗi. Vị trí lỗi có thể tìm thấy với Verification, mà không thể với Validation.
  + Verification: thẩm tra quan tâm đến việc ngăn chặn lỗi giữa các công đoạn. Phát hiện lỗi lập trình.

+ Validation: xác nhận quan tâm đến sản phẩm cuối cùng không còn lỗi. Phát hiện lỗi phân tích, thiết kế.

Pregunta 2.3: Giải thích sự khác nhau giữa *failure, error, và fault.*

- Lỗi phần mềm(Software Error) là các phần code sai do lôi cú pháp, logic hoặc lỗi do phân tích, thiết kế.
  + Thường là chỉ một lỗi của con người trong quá trình xây dựng phần mềm.

- Sai sót(Software Fault)là các errors dẫn tới hoạt động không chính xác của phần mềm. Không phải error nào cũng gây ra fault.
  + Là lỗi nằm trong mã nguồn, tài liệu của chương trình. Loại lỗi này có nhiều nguyên nhân như: do error của con người, do công nghệ phức tạp, áp lực công việc, do các thành phần của hệ thống tương tác với nhau, … Kiểm thử viên chủ yếu là bắt các loại lỗi này.

- Hỏng(Software Failures)Fault sẽ trở thành failure khi nó được kích hoạt. Một số đường chạy gây ra failures, một số không.
  + Dùng để chỉ các lỗi dưới góc độ của hệ thống. Khi một hệ thống không thực hiện được chức năng cần thiết, hoặc thực hiện chức năng không được phép làm thì được gọi là fail/failure. Một bug có thể là nguyên nhân của nhiều fail khi hệ thống hoạt động.

Pregunta 2.4: Điểm mạnh và điểm yếu của kiểm thử tự động và kiểm thử bằng tay?

Tiêu chí	Manual test	Automation test
Thời gian	Mất nhiều thời gian thực thi nhưng không phải test lặp đi lặp lại	Mất ít thời gian thực thi nhưng quá trình test lặp tăng hơn nhiều so với manual test
Độ linh động	Linh động do test thủ công nên có thể phát hiện và xử lý những tình huống phát sinh trong quá trình test. Và có thể tìm ra lỗi mới.	Không linh động vì test theo test script nên quá trình test không phát hiện ra lỗi mới. Chỉ thích hợp với kiểm thử hồi quy.
Phụ thuộc	Phụ thuộc vào trạng thái của con người nên kết quả test có thể kém chính xác đối với dự án lớn có nhiều test case.	Nhất quán, nên kết quả test là chính xác và không phụ thuộc vào các yếu tố ngoại cảnh.
Bảo trì	Không cần bảo trì	Cần bảo trì
Kết quả	Có kết quả ngay lập tức	Cần một thời gian mới có kết quả
Ưu điểm	Manual test linh động và trong quá trình test có thể tìm ra lỗi mới	Automation test thích hợp cho việc kiểm thử lặp đi lặp lại, có thể tái sử dụng test script. Thích hợp giả lập để test hiệu năng và khả năng chịu tải cũng như giả lập các hệ thống để kiểm thử.
Hạn chế	Nếu sử dụng manual test mà test một chức năng lập đi lặp lặp lại thì sẽ tốn nhiều thời gian và sẽ khó chính xác. Nên thay thế bằng automation test để đở mất thời gian giám sát, tối ưu hóa việc sử dụng tài nguyên máy tính để kiểm thử.	Nếu sử dụng automation test mà test ít thì sẽ rất lãng phí thời gian và nhân lực cho công việc viết test script, nên thay thế bằng manual test.

Pregunta 2.5: Kiểm thử Beta là cái gì? Kiểm thử Alpha là cái gì? Nêu sự giống và khác nhau cơ bản giữa chúng?

- Alpha testing: là việc kiểm thử hoạt động chức năng thực tế hoặc giả lập do người dùng/khách hàng tiềm năng hoặc một nhóm Tester độc lập thực hiện tại nơi sản xuất phần mềm.

Kiểm thử Alpha thường được sử dụng cho phần mềm đại trà (đóng gói sẵn để bán) như là một hình thức kiểm thử mức chấp nhận nội bộ trước khi phần mềm chính thức đi vào giai đoạn kiểm thử Beta.

- Beta testing: (sau khi kiểm thử Alpha) là một hình thức mở rộng của kiểm thử mức chấp nhận của người dùng. Các phiên bản của phần mềm, gọi là phiên bản beta, được phát hành cho một đối tượng hạn chế bên ngoài của nhóm lập trình.

Phần mềm này được phát hành cho nhiều nhóm người dùng để kiểm thử nhiều hơn nữa và nó có thể đảm bảo sản phẩm có ít thiếu sót và lỗi. Đôi khi, các phiên bản beta được phát hành rộng rãi để tăng phạm vi phản hồi thông tin từ một số lượng tối ta người dùng trong tương lai.

Sự khác nhau

<u>Kiểm thử Alpha: được bên phát triển tiến hành</u>

+ Phần mềm sẽ được dùng trong bối cảnh "tự nhiên".

+ Người phát triển "nhòm qua vai" người sử dụng và báo cáo các sai và các vấn đề sử dụng (vì thế còn gọi là kiểm thử sau lưng)

+ Được tiến hành trong một môi trường được điều khiển (theo kế hoạch của người phát triển)
+ Dữ liệu cho kiểm thử Alpha thường là dữ liệu mô phỏng.

Kiểm thử Beta: được nhiều người đặt hàng tiến hành, không có mặt người phát triển.
+ Áp dụng trong môi trường thực, không có sự kiểm soát của người phát triển
+ Khách hàng sẽ báo cáo tất cả các vấn đề (thực hoặc tưởng tượng) mà họ gặp trong quá trình kiểm thử cho người phát triển một cách định kỳ.
+ Theo báo cáo đó người phát triển tiến hành sửa đổi và chuẩn bị phân phối bản phát hành bản hoàn thiện cho toàn bộ những người đặt hàng.

Pregunta 2.6: Nêu các bước kiểm thử tích hợp từ trên xuống? Ưu nhược điểm của cách tiếp cận này?
- Phương pháp tích hợp trên xuống: đây là một phương pháp tích hợp tăng dần với việc xây dựng cấu trúc chương trình. Các modul được tích hợp bằng cách đi dần xuống theo trật tự điều khiển, bắt đầu với modul điều khiển chính (chương trình chính). Các modul phụ thuộc (và phụ thuộc cuối cùng) vào modul điều khiển chính sẽ được tổ hợp dần vào trong cấu trúc theo hoặc chiều sâu trước hay chiều rộng trước

- Quá trình tích hợp từ trên xuống được thực hiện theo 5 bước:
1. Modul điều khiển chính được dùng như bộ lái kiểm thử (test driver) và tất cả các modul phụ trợ được thay thế bởi các cuống (stub).
2. Thay thế dần từng cuống bởi modul thực thi tương ứng.
3. Sau khi tích hợp modul đó, tiến hành các kiểm thử tương ứng.
4. Sau khi hoàn thành đủ tập các kiểm thử này thì thay một cuống (stub) khác bằng modul thực (nghĩa là quay lại bước 2).
5. Có thể kiểm thử lại (toàn bộ hoặc một phần các kiểm thử trước) để bảo đảm rằng không có sai mới nào được sinh ra.
6. Tiếp tục lặp lại từ bước 2 cho tới khi toàn bộ cấu trúc chương trình được xây dựng.

- Ưu:
  + Phát hiện sớm các lỗi thiết kế
  + Có phiên bản hoạt động sớm
- Nhược: Cần thiết các cuống
  + Các khó khăn kèm theo cuống.
  + Khó có thể mô phỏng được các chức năng của mô đun cấp thấp phức tạp
  + Không kiểm thử đầy đủ các chức năng
  + Chiến lược này có vẻ không phức tạp, nhưng thực tế nảy ra các vấn đề logic: khi xử lý ở mức thấp lại đòi hỏi phải đủ tương xứng với mức cao.
  + Các cuống được thay thế cho các modul mức thấp, do đó không 1 dữ liệu có ý nghĩa nào có thể chảy ngược lên trong cấu trúc của chương trình. Người kiểm thử đứng trước 2 lựa chọn: (1) để trễ

nhiều việc kiểm thử tới khi cuống được thay thế bằng modul thực tế, (2) xây dựng các cuống thực hiện những chức năng giới hạn mô phỏng cho modul thực tại, và (3) tích hợp phần mềm từ đáy cấp bậc lên.

Pregunta 2.7: Nêu các bước kiểm thử tích hợp từ dưới lên? Ưu nhược điểm của cách tiếp cận này?

- Phương pháp tích hợp dưới lên: bắt đầu xây dựng và kiểm thử với các modul nguyên tử (tức là các modul ở mức thấp nhất trong cấu trúc chương trình). Vì các modul này được tích hợp từ dưới lên nên việc xử lý yêu cầu đối với các modul phụ thuộc vào một mức nào đó bao giờ cũng có sẵn và nhu cầu về cuống bị dẹp bỏ.

Bắt đầu xây dựng và kiểm thử từ các modul nguyên tố: việc xử lý nếu có đòi hỏi các modul phụ trợ thì các modul thực sự đã sẵn sàng (cuống đã bị loại).

Được thực hiện qua 4 bước:

1. Các modul mức thấp được tổ hợp vào trong các cụm (cluster) thực hiện một chức năng phụ trợ đặc biệt (các cluster gọi là các build)
2. Một bộ lái (chương trình điều khiển kiểm thử) được viết để phối hợp đầu vào và đầu ra củaca kiểm thử.
3. Kiểm thử cụm đó.
4. Tháo bỏ các driver và các cụm được tổ hợp ngược lên trong cấu trúc chương trình

- Ưu: Thiết kế ca kiểm thử dễ và không cần cuống
  + Tránh phải tạo các stub phức tạp hay tạo các kết quả nhân tạo
  + Thuận tiện cho phát triển các mô đun thứ cấp dùng lại được

- Nhược: luôn chứa chương trình như một chỉnh thể cho đến khi modul cuối cùng được thêm vào.
+ Phát hiện chậm các lỗi thiết kế
+ Chậm có phiên bản thực hiện được của hệ thống

Pregunta 2.8: Thế nào là một ca kiểm thử tốt? ca kiểm thử thành công? Lợi ích phụ của kiểm thử là gì?

- Ca kiểm thử tốt là ca kiểm thử có xác suất cao trong việc tìm ra một lỗi chưa được phát hiện.
- Một ca kiểm thử thắng lợi làm lộ ra khiếm khuyết, đồng thời mang lại các lợi ích phụ:
  + Thuyết minh rằng các chức năng phần mềm tương ứng với đặc tả (xác minh)
  + Yêu cầu thực thi là phù hợp (thẩm định)
  + Cung cấp thêm các chỉ số độ tin cậy và chỉ số về chất lượng phần mềm nói chung (thẩm định)

Pregunta 2.9: Giải thích sự khác nhau giữa kiểm thử hộp trắng và kiểm thử hộp đen?

- Kiểm thử hộp đen (Black-box Testing): Là hình thức kiểm thử mà kiểm thử viên không cần biết đến cách thức hoạt động, mã nguồn, xử lý dữ liệu bên trong một thành phần/hệ thống.

Công việc cần làm là nhập dữ liệu đầu vào (input) và kiểm tra kết quả trả về có đúng như mong muốn hay không.

- Kiểm thử hộp trắng (White-box Testing): Là hình thức kiểm thử mà kiểm thử viên biết được các cấu trúc bên trong của chương trình (mã nguồn, xử lý dữ liệu, …). Việc kiểm thử được dựa trên các phân tích về cấu trúc bên trong của thành phần/hệ thống.

Pregunta 2.10: Nếu chương trình thành công ở tất cả các bộ kiểm thử của kiểm thử hộp đen. Liệu ta có cần kiểm thử hộp trắng nữa không? Tại sao?

- Nếu chương trình thành công ở tất cả các bộ kiểm thử của kiểm thử hộp đen thì vẫn cần kiểm thử hộp trắng (Đặc biệt là các module quan trọng, thực thi việc tính toán
chính của hệ thống).

- Vì cả quá trình kiểm thử trong thực tế cũng ko thể phát hiện hết các lỗi.
  + Và kiểm thử hộp đen nhằm tìm 1 vài loại lỗi(Chức năng thiếu hoặc không đúng, Sai về giao diện, Sai trong cấu trúc hoặc trong truy cập dữ liệu ngoài, Sai thực thi chức năng, Sai khởi đầu hoặc kết thúc modul).
  + Bởi quá trình này tách biệt với code, tester ko biết hệ thống đc xây dựng thế nào, nên một vài phần test thiếu.

Pregunta 2.11: Khi nào dùng kỹ thuật bảng quyết định, kiểm thử biên, kiểm thử theo cặp?

Pregunta 2.12: Khi nào dùng kỹ thuật kiểm thử biên, kiểm thử theo cặp, sơ đồ chuyển trạng?

Kiểm thử biên được sử dụng khi:
+ Giá trị đầu vào ko ràng buộc lẫn nhau
+ Có nhiều giá trị đầu vào nhận giá trị trong các miền hoặc các tập hợp
+ Muốn test đơn giản, test tự động hóa.
(Kiểm thử đơn vị, tích hợp, hệ thống và chấp nhận)

Kiểm thử lớp tương đương được sử dụng khi:
+ Nhiều giá trị đầu vào và nhận giá trị trong các miền/tập
+ Test thủ công
(Kiểm thử đơn vị, tích hợp, hệ thống và chấp nhận)

Kiểm thử bảng quyết định được sử dụng khi:
+ Logic và điều kiện phức tạp. Có các quan hệ logic quan trọng giữa các biến đầu vào.
+ Đặc tả có thể chuyển về trạng thái bảng(Các chức năng có thể mô tả bằng quyết định)
+ Thứ tự các hành động sảy ra ko quan trọng
+ Thứ tự các luật ko ảnh hưởng đến hàng động

+ Khi một luật thỏa mãn và được chọn thì không cần xét luật khác
+ Các luật phải đầy đủ(Có mọi tổ hợp) và nhất quán(Mọi tổ hợp chân lý chỉ gây ra 1 tập hành động)

Kiểm thử theo cặp được sử dụng khi:
+ Rất nhiều biến/tham số
+ Nếu lỗi xảy ra thì sẽ rất nghiêm trọng

Sơ đồ chuyển trạng được sử dụng khi:
+ Khi muốn kiểm thử trạng thái, sự kiện, chuyển tiếp
+ Ko hữu dụng khi hệ thống ko có trạng thái hay ko phải đáp trả các sự kiện thời gian thực từ bên ngoài hệ thống

Pregunta 2.13: các thành phần cần có trong test plan?
Một số hạng mục có thể được bao gồm trong một test plan, tùy thuộc vào từng dự án cụ thể:

* Tiêu đề
* Định nghĩa version của phần mềm (version release)
* Lưu lại quá trình hiệu chỉnh tài liệu như tác giả, ngày cập nhật, duyệt
* Mục lục
* Mục đích của tài liệu, ý kiến chung
* Mục tiêu của chi phí kiểm thử (test)
* Giới thiệu tổng quan về sản phẩm
* Danh sách tài liệu liên quan như spec, tài liệu thiết kế, các kế hoạch test khác,...
* Các tiêu chuẩn thích hợp, các yêu cầu hợp lệ
* Nguồn gốc của các sự thay đổi
* Relevant naming conventions and identifier conventions
* Overall software project organization and personnel/contact-info/responsibilties
* Test organization and personnel/contact-info/responsibilities
* Assumptions and dependencies
* Phân tích rủi ro của dự án
* Các vấn đề ưu tiên và tập trung test
* Phạm vi và giới hạn test
* Test outline - a decomposition of the test approach by test type, feature, functionality, process, system, module, etc. as applicable
* Outline of data input equivalence classes, boundary value analysis, error classes
* Test environment - hardware, operating systems, other required software, data configurations, interfaces to other systems
* Test environment validity analysis - differences between the test and production systems and their impact on test validity.
* Test environment setup and configuration issues
* Software migration processes

* Software CM processes
* Test data setup requirements
* Database setup requirements
* Outline of system-logging/error-logging/other capabilities, and tools such as screen capture software, that will be used to help describe and report bugs
* Discussion of any specialized software or hardware tools that will be used by testers to help track the cause or source of bugs
* Test tự động - giải thích và tổng quan
* Các công cụ test được sử sụng, bao gồm các version, bản vá lỗi,...v.v
* các qui trình bảo trì và quản lý version của test script/test code
* Theo dõi và giải quyết vấn đề - Các công cụ và qui trình
* Các thước đo về test sản phẩm được sử dụng
* Báo cáo các yêu cầu và khả năng giao test
* Điều kiện đầu vào và đầu ra của phần mềm
* Giai đoạn và điều kiện test ban đầu
* Điều kiện dừng test và test lại
* Sự bố trí nhân sự
* Những người cần training trước khi tham gia
* Nơi test
* Các tổ chức test bên ngoài sẽ sử dụng và mục đích, trách nhiệm, khả năng hoàn thành, người liên hệ và các vấn đề hợp tác của họ
* Các vấn đề độc quyền thích hợp, phân loại, bảo mật và bản quyền.
* Các vấn đề mở
* Phụ lục - bảng chú giải, các từ viết tắt, ...v.v.

Pregunta 2.14: Các giai đoạn chính của 1 tiến trình test?

Unit Test – Kiểm tra mức đơn vị

Một Unit là một thành phần PM nhỏ nhất mà ta có thể kiểm tra được. Theo định nghĩa này, các hàm (Function), thủ tục (Procedure), lớp (Class), hoặc các phương thức (Method) đều có thể được xem là Unit.

Vì Unit được chọn để kiểm tra thường có kích thước nhỏ và chức năng hoạt động đơn giản, chúng ta không khó khăn gì trong việc tổ chức, kiểm tra, ghi nhận và phân tích kết quả kiểm tra. Nếu phát hiện lỗi, việc xác định nguyên nhân và khắc phục cũng tương đối dễ dàng vì chỉ khoanh vùng trong một đơn thể Unit đang kiểm tra. Một nguyên lý đúc kết từ thực tiễn: thời gian tốn cho Unit Test sẽ được đền bù bằng việc tiết kiệm rất nhiều thời gian và chi phí cho việc kiểm tra và sửa lỗi ở các mức kiểm tra sau đó.

Unit Test thường do lập trình viên thực hiện. Công đoạn này cần được thực hiện càng sớm càng tốt trong giai đoạn viết code và xuyên suốt chu kỳ PTPM. Thông thường, Unit Test đòi hỏi kiểm tra viên có kiến thức về thiết kế và code của chương trình. Mục đích của Unit Test là bảo đảm thông tin được xử lý và xuất (khỏi Unit) là chính xác, trong mối tương quan với dữ liệu nhập và chức năng của Unit. Điều này thường đòi hỏi tất cả các nhánh bên trong Unit đều phải được kiểm tra để phát hiện nhánh

phát sinh lỗi. Một nhánh thường là một chuỗi các lệnh được thực thi trong một Unit, ví dụ: chuỗi các lệnh sau điều kiện If và nằm giữa then ... else là một nhánh. Thực tế việc chọn lựa các nhánh để đơn giản hóa việc kiểm tra và quét hết Unit đòi hỏi phải có kỹ thuật, đôi khi phải dùng thuật toán để chọn lựa.

Cũng như các mức kiểm tra khác, Unit Test cũng đòi hỏi phải chuẩn bị trước các tình huống (test case) hoặc kịch bản (script), trong đó chỉ định rõ dữ liệu vào, các bước thực hiện và dữ liệu mong chờ sẽ xuất ra. Các test case và script này nên được giữ lại để tái sử dụng.

Integration Test – Kiểm tra tích hợp

Integration test kết hợp các thành phần của một ứng dụng và kiểm tra như một ứng dụng đã hoàn thành. Trong khi Unit Test kiểm tra các thành phần và Unit riêng lẻ thì Intgration Test kết hợp chúng lại với nhau và kiểm tra sự giao tiếp giữa chúng.

Integration Test có 2 mục tiêu chính:

Phát hiện lỗi giao tiếp xảy ra giữa các Unit.

Tích hợp các Unit đơn lẻ thành các hệ thống nhỏ (subsystem) và cuối cùng là nguyên hệ thống hoàn chỉnh (system) chuẩn bị cho kiểm tra ở mức hệ thống (System Test).

Trong Unit Test, lập trình viên cố gắng phát hiện lỗi liên quan đến chức năng và cấu trúc nội tại của Unit. Có một số phép kiểm tra đơn giản trên giao tiếp giữa Unit với các thành phần liên quan khác, tuy nhiên mọi giao tiếp liên quan đến Unit thật sự được kiểm tra đầy đủ khi các Unit tích hợp với nhau trong khi thực hiện Integration Test.

Trừ một số ít ngoại lệ, Integration Test chỉ nên thực hiện trên những Unit đã được kiểm tra cẩn thận trước đó bằng Unit Test, và tất cả các lỗi mức Unit đã được sửa chữa. Một số người hiểu sai rằng Unit một khi đã qua giai đoạn Unit Test với các giao tiếp giả lập thì không cần phải thực hiện Integration Test nữa. Thực tế việc tích hợp giữa các Unit dẫn đến những tình huống hoàn toàn khác.

Một chiến lược cần quan tâm trong Integration Test là nên tích hợp dần từng Unit. Một Unit tại một thời điểm được tích hợp vào một nhóm các Unit khác đã tích hợp trước đó và đã hoàn tất (passed) các đợt Integration Test trước đó. Lúc này, ta chỉ cần kiểm tra giao tiếp của Unit mới thêm vào với hệ thống các Unit đã tích hợp trước đó, điều này làm cho số lượng kiểm tra sẽ giảm đi rất nhiều, sai sót sẽ giảm đáng kể.

Có 4 loại kiểm tra trong Integration Test:

Kiểm tra cấu trúc (structure): Tương tự White Box Test (kiểm tra nhằm bảo đảm các thành phần bên trong của một chương trình chạy đúng), chú trọng đến hoạt động của các thành phần cấu trúc nội tại của chương trình chẳng hạn các lệnh và nhánh bên trong.

Kiểm tra chức năng (functional): Tương tự Black Box Test (kiểm tra chỉ chú trọng đến chức năng của chương trình, không quan tâm đến cấu trúc bên trong), chỉ khảo sát chức năng của chương trình theo yêu cầu kỹ thuật.

Kiểm tra hiệu năng (performance): Kiểm tra việc vận hành của hệ thống.

Kiểm tra khả năng chịu tải (stress): Kiểm tra các giới hạn của hệ thống.

System Test - Kiểm tra mức hệ thống

Mục đích System Test là kiểm tra thiết kế và toàn bộ hệ thống (sau khi tích hợp) có thỏa mãn yêu cầu đặt ra hay không.

System Test bắt đầu khi tất cả các bộ phận của PM đã được tích hợp thành công. Thông thường loại kiểm tra này tốn rất nhiều công sức và thời gian. Trong nhiều trường hợp, việc kiểm tra đòi hỏi một số thiết bị phụ trợ, phần mềm hoặc phần cứng đặc thù, đặc biệt là các ứng dụng thời gian thực, hệ thống phân bố, hoặc hệ thống nhúng. Ở mức độ hệ thống, người kiểm tra cũng tìm kiếm các lỗi, nhưng trọng tâm là đánh giá về hoạt động, thao tác, sự tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống.
Điểm khác nhau then chốt giữa Integration Test và System Test là System Test chú trọng các hành vi và lỗi trên toàn hệ thống, còn Integration Test chú trọng sự giao tiếp giữa các đơn thể hoặc đối tượng khi chúng làm việc cùng nhau. Thông thường ta phải thực hiện Unit Test và Integration Test để bảo đảm mọi Unit và sự tương tác giữa chúng hoạt động chính xác trước khi thực hiện System Test.
Sau khi hoàn thành Integration Test, một hệ thống PM đã được hình thành cùng với các thành phần đã được kiểm tra đầy đủ. Tại thời điểm này, lập trình viên hoặc kiểm tra viên (tester) bắt đầu kiểm tra PM như một hệ thống hoàn chỉnh. Việc lập kế hoạch cho System Test nên bắt đầu từ giai đoạn hình thành và phân tích các yêu cầu. Phần sau ta sẽ nói rõ hơn về một quy trình System Test cơ bản và điển hình.
System Test kiểm tra cả các hành vi chức năng của phần mềm lẫn các yêu cầu về chất lượng như độ tin cậy, tính tiện lợi khi sử dụng, hiệu năng và bảo mật. Mức kiểm tra này đặc biệt thích hợp cho việc phát hiện lỗi giao tiếp với PM hoặc phần cứng bên ngoài, chẳng hạn các lỗi "tắc nghẽn" (deadlock) hoặc chiếm dụng bộ nhớ. Sau giai đoạn System Test, PM thường đã sẵn sàng cho khách hàng hoặc người dùng cuối cùng kiểm tra để chấp nhận (Acceptance Test) hoặc dùng thử (Alpha/Beta Test).
Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan, System Test thường được thực hiện bởi một nhóm kiểm tra viên hoàn toàn độc lập với nhóm phát triển dự án.
Bản thân System Test lại gồm nhiều loại kiểm tra khác nhau (xem hình 6.2), phổ biến nhất gồm:
Kiểm tra chức năng (Functional Test): bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
Kiểm tra khả năng vận hành (Performance Test): bảo đảm tối ưu việc phân bổ tài nguyên hệ thống (ví dụ bộ nhớ) nhằm đạt các chỉ tiêu như thời gian xử lý hay đáp ứng câu truy vấn...
Kiểm tra khả năng chịu tải (Stress Test hay Load Test): bảo đảm hệ thống vận hành đúng dưới áp lực cao (ví dụ nhiều người truy xuất cùng lúc). Stress Test tập trung vào các trạng thái tới hạn, các "điểm chết", các tình huống bất thường...
Kiểm tra cấu hình (Configuration Test)
Kiểm tra khả năng bảo mật (Security Test): bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.
Kiểm tra khả năng phục hồi (Recovery Test): bảo đảm hệ thống có khả năng khôi phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến.

Acceptance Test - Kiểm tra chấp nhận sản phẩm

Thông thường, sau giai đoạn System Test là Acceptance Test, được khách hàng thực hiện (hoặc ủy quyền cho một nhóm thứ ba thực hiện). Mục đích của Acceptance Test là để chứng minh PM thỏa mãn tất cả yêu cầu của khách hàng và khách hàng chấp nhận sản phẩm (và trả tiền thanh toán hợp đồng).

Acceptance Test có ý nghĩa hết sức quan trọng, mặc dù trong hầu hết mọi trường hợp, các phép kiểm tra của System Test và Accepatnce Test gần như tương tự, nhưng bản chất và cách thức thực hiện lại rất khác biệt.
Đối với những sản phẩm dành bán rộng rãi trên thị trường cho nhiều người sử dụng, thông thường sẽ thông qua hai loại kiểm tra gọi là Alpha Test và Beta Test. Với Alpha Test, người sử dụng (tiềm năng) kiểm tra PM ngay tại nơi PTPM, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa. Với Beta Test, PM sẽ được gửi tới cho người sử dụng (tiềm năng) để kiểm tra ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

Regression Test - Kiểm tra hồi quy

Trước tiên cần khẳng định Regression Test không phải là một mức kiểm tra, như các mức khác đã nói ở trên. Nó đơn thuần kiểm tra lại PM sau khi có một sự thay đổi xảy ra, để bảo đảm phiên bản PM mới thực hiện tốt các chức năng như phiên bản cũ và sự thay đổi không gây ra lỗi mới trên những chức năng vốn đã làm việc tốt. Regression test có thể thực hiện tại mọi mức kiểm tra.
Ví dụ: một PM đang phát triển khi kiểm tra cho thấy nó chạy tốt các chức năng A, B và C. Khi có thay đổi code của chức năng C, nếu chỉ kiểm tra chức năng C thì chưa đủ, cần phải kiểm tra lại tất cả các chức năng khác liên quan đến chức năng C, trong ví dụ này là A và B. Lý do là khi C thay đổi, nó có thể sẽ làm A và B không còn làm việc đúng nữa.
Mặc dù không là một mức kiểm tra, thực tế lại cho thấy Regression Test là một trong những loại kiểm tra tốn nhiều thời gian và công sức nhất. Tuy thế, việc bỏ qua Regression Test là "không được phép" vì có thể dẫn đến tình trạng phát sinh hoặc tái xuất hiện những lỗi nghiêm trọng, mặc dù ta "tưởng rằng" những lỗi đó hoặc không có hoặc đã được kiểm tra và sửa chữa rồi!

Bài tập

2.**15,16,17,18(b),19,20,21,**22,23,**24,25,26(?),27(?)**
3.7,11,15,18,19,20,21

Pregunta 2.15: Cho Form gồm 1 ratio button Nữ, 1 ratio button Độc thân và 1 Danh sách chọn Chuyên ngành gồm: CNTT, QTKD, VT, Kế toán.
a. Nếu kiểm thử tất cả các trường hợp xảy ra thì cần bao nhiêu test cases,
b. Mỗi test case chứa bao nhiêu cặp giá trị?
c. Liệt kê các cặp giá trị có thể xảy ra?
d. Thiết kế bộ pairwise test suite (kiểm thử theo cặp)

Pregunta 2.16: Một phần mềm điều khiển chơi game đơn giản giữa 2 người chơi. Mỗi người điều khiển một nút bấm để đưa bóng vào lỗ. Mỗi lần bóng vào lỗ, người đó được cộng 1 điểm. Ai được 5 điểm trước sẽ thắng. Nếu bóng không vào lỗ, người còn lại được quyền điều khiển bóng.
a. Lập sơ đồ chuyển trạng thái
b. Xác định các đường chạy để phủ hết các cạnh
c. Thiết kế test case tương ứng

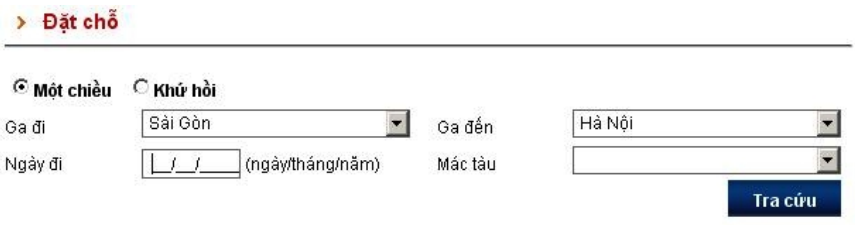
Pregunta 2.17: Thông tin về block1 bao gồm size(small, large), color(red, green, blue), shape (circle, triangle, square).
a. Nếu kiểm thử tất cả các trường hợp xảy ra thì cần bao nhiêu ca kiểm thử?
b. Số cặp tối đa mà một ca kiểm thử có thể chứa
c. Xác định các cặp giá trị có thể xảy ra
d. Thiết kế bộ kiểm thử theo cặp (pairwise test suite)

Pregunta 2.18: Cần phát triển module kiểm tra điều kiện dự thi của sinh viên gồm:
Nếu sinh viên đi học >=80% số buổi, điểm giữa kì >0, điểm bài tập lớn >0 sẽ được thi.
Nếu sinh viên đủ điều kiện dự thi và có điểm bài tập lớn = 10 hoặc điểm giữa kỳ = 10 sẽ được miễn thi.
a. Dùng kỹ thuật bảng quyết định để xác định test cases
b. Dùng kỹ thuật đồ thị nguyên nhân kết quả để xác định test cases

Pregunta 2.19: Cho hệ thống S nhận n tham số đầu vào, mỗi tham số có m giá trị. Trả lời Pregunta:
a. Số cặp tối đa của một ca kiểm thử
b. Trong trường hợp lý tưởng, ta cần bao nhiêu ca kiểm thử để bao phủ tất cả các cặp của hệ thống?
c. Tính tổng số cặp mà bộ kiểm thử phải bao phủ?
d. Cho n = 13, m = 3. Số ca kiểm thử tối thiểu cần chọn để thu được bộ kiểm thử theo cặp (pairwise test suite)?

a. Số cặp tối đa của 1 ca kiểm thử: [(n-1) + (n-2) +…+ 1] cặp
b. Trong trường hợp lý tưởng, số ca kiểm thử để bao phủ tất cả các cặp của hệ thống là: m^2 test case
c. Tổng số cặp mà bộ kiểm thử phải bao phủ **:** [(n-1) + (n-2) +…+ 1]* m^2
d. n = 13, m = 3. Số ca kiểm thử tối thiểu để thu được bộ kiểm thử theo cặp (pairwise test suite): là 9 test case

Pregunta 2.20: Form đăng ký mua vé tàu được cho như hình vẽ. Danh sách ga ở Ga đi và Ga đến là {Hà Nội, Vinh, Huế, Đà Nẵng, Sài Gòn}. Danh sách mác tàu là {SE,TN}. Không tính

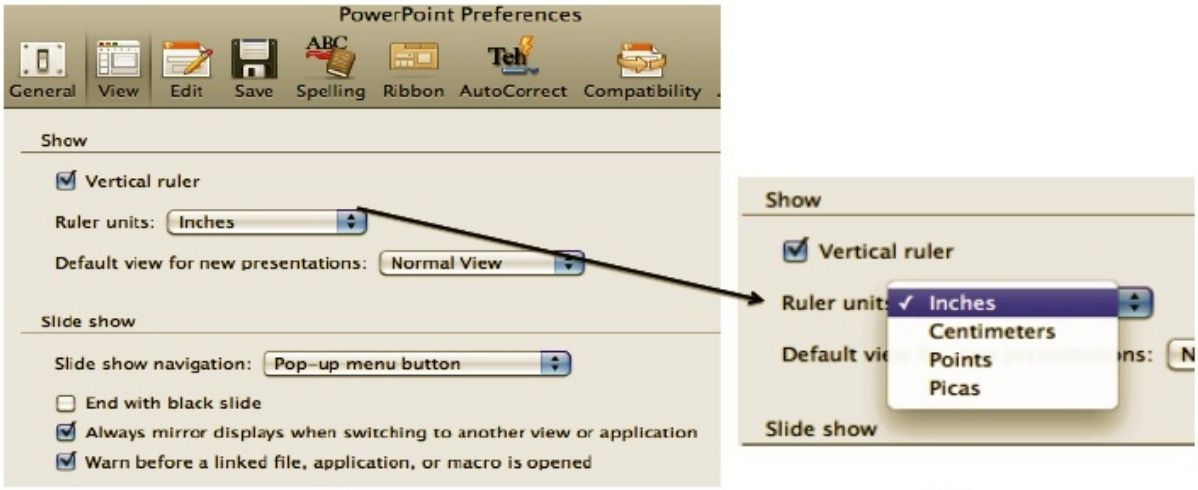


trường Ngày đi, hãy thực hiện:
a. Nếu kiểm thử tất cả các trường hợp xảy ra thì cần bao nhiêu ca kiểm thử?
b. Số cặp tối đa mà một ca kiểm thử có thể chứa
c. Xác định các cặp giá trị có thể xảy ra
d. Thiết kế bộ kiểm thử theo cặp (pairwise test suite)

a. Tổng số ca kiểm thử: 2*(5*5-5)*2 = 80 test case
(Vì Ga đi và Ga đến có 5 giá trị -> có 5*5 trường hợp. Nhưng loại bỏ 5 trường hợp Ga đi và Ga đến trùng nhau)
b. Số cặp tối đa của 1 ca kiểm thử: 6 cặp
c. Các cặp giá trị có thể xảy ra: 64 cặp
(Một chiều, đi Hà Nội), (Một chiều, đến Hà Nội), (Một chiều, SE), (Một chiều, TN)…
d. Một bộ test case(pairwises test suite) bao phủ được tất cả các cặp:
(20 test case)

Loại vé	**Ga đi**	**Ga đến**	**Mác tàu**
Một Chiều	**Hà Nội**	Vinh	SE
Cứ Hồi	Vinh	Huế	TN
Một Chiều	Huế	Đà Nẵng	SE
Cứ Hồi	Đà Nẵng	Sài Gòn	TN
Một Chiều	Sài Gòn	Hà Nội	SE
Cứ Hồi	**Hà Nội**	Huế	TN
Một Chiều	Vinh	Đà Nẵng	SE
Cứ Hồi	Huế	Sài Gòn	TN
Một Chiều	Đà Nẵng	Hà Nội	SE
Cứ Hồi	Sài Gòn	Vinh	TN
Cứ Hồi	**Hà Nội**	Đà Nẵng	TN
Một Chiều	Vinh	Sài Gòn	SE
Cứ Hồi	Huế	Hà Nội	TN
	Đà Nẵng	Vinh	
Một Chiều	Sài Gòn	Huế	SE
	Hà Nội	Sài Gòn	
	Vinh	Hà Nội	
	Huế	Vinh	
	Đà Nẵng	Huế	
	Sài Gòn	Đà Nẵng	

Pregunta 2.21: Xét tab tùy chọn View từ một phiên bản của phần mềm Powerpoint Microsoft. Bảng (c) chính là dữ liệu sau trích xuất.
a. Nếu kiểm thử tất cả các trường hợp xảy ra thì cần bao nhiêu ca kiểm thử?
b. Số cặp tối đa mà một ca kiểm thử có thể chứa
c. Xác định các cặp giá trị có thể xảy ra
d. Thiết kế bộ kiểm thử theo cặp (pairwises test suite)



(a) (b)

Vertical Ruler	Ruler Units	Default View	SS Navigation	End with Black	Always Mirror	Warn Before
Visible	Inches	Normal	Pop-up	Yes	Yes	Yes
Invisible	Centimeters	Slide	None	No	No	No
	Points	Outline				
	Picas					

(c)

a. Tổng số tất cả các trường hợp: 2*4*3*2*2*2*2 = 384 test case.

b. Số cặp tối đa chứa trong 1 test case: 21 cặp

c. Các cặp giá trị có thể xảy ra: 122 cặp

d. Một bộ test case(pairwises test suite) bao phủ được tất cả các cặp:

(13 test case)

Vertical Ruler	Ruler units	Default Vie	Ss Navigator	End Black	With Always Mirror	Warn Before
Visible	Centimetes	Norma	Non	No1	Yes2	Yes3
InVisible	Picas	Normal	Popup	Yes1	No2	No3
Visible	Picas	Slide	Popup	No1	No2	Yes3
InVisible	Inches	Normal	None	Yes1	Yes2	No3
InVisible	Points	Outline	None	No1	No2	No3
Visible	Centimetes	Slide	Popup	Yes1	Yes2	No3
Visible	Picas	Outline	None	Yes1	Yes2	Yes3
InVisible	Inches	Outline	Popup	No1	No2	Yes3
Visible	Points	Slide	None	Yes1	Yes2	Yes3
InVisible	Inches	Slide	None	No1	Yes2	Yes3
Visible	Inches	Normal	Popup	No1	Yes2	Yes3
Visible	Points	Normal	Popup	Yes1	Yes2	No3
InVisible	Centimetes	Outline	Popup	Yes1	No2	Yes3

Pregunta 2.22: Cần phát triển module tính thuế thu nhập cá nhân dựa trên phần thu nhập tính thuế. Biểu thuế thu nhập cá nhân được cho như bảng dưới:

Bậc thuế	Phần thu nhập tính thuế/năm (triệu đồng)	Phần thu nhập tính thuế/tháng (triệu đồng)	Thuế suất (%)
1	Đến 60	Đến 5	5
2	Trên 60 đến 120	Trên 5 đến 10	10
3	Trên 120 đến 216	Trên 10 đến 18	15
4	Trên 216 đến 384	Trên 18 đến 32	20
5	Trên 384 đến 624	Trên 32 đến 52	25
6	Trên 624 đến 960	Trên 52 đến 80	30
7	Trên 960	Trên 80	35

(a) Thiết kế các ca kiểm thử dùng kỹ thuật phân tích giá trị biên
(b) Thiết kế các ca kiểm thử dùng kỹ thuật phân vùng tương đương

a. Phân tích giá trị biên
+ Bậc thuế 1: thu nhập mỗi năm <= 60 -> biên 60

+ Bậc thuế 2: 60 < thu nhập mỗi năm <= 120 -> biên 61 ; 120
+ Bậc thuế 3: 120 < thu nhập mỗi năm <= 216 -> biên 121; 216
+ Bậc thuế 4: 216 < thu nhập mỗi năm <= 384 -> biên 217; 384
+ Bậc thuế 5: 384 < thu nhập mỗi năm <= 624 -> biên 385; 624
+ Bậc thuế 6: 624 < thu nhập mỗi năm <= 960 -> biên 625; 960
+ Bậc thuế 7: 960 < thu nhập mỗi năm -> biên 961

Biên	Input	Kết quả mong muốn
Biên Bậc thuế 1	60	3 (60*0,05)
Biên Bậc thuế 2	61	3,1 (60*0,05 + 1*0,1)
Biên Bậc thuế 2	120	9 (60*0,05 + 60*0,1)
Biên Bậc thuế 3	121	9,15 (60*0,05 + 60*0,1 + 1*0,15)
Biên Bậc thuế 3	216	23,4 (60*0,05 + 60*0,1 + 96*0,15)
Biên Bậc thuế 4	217	23,6 (60*0,05 + 60*0,1 + 96*0,15 + 1*0,2)
Biên Bậc thuế 4	384	57 (60*0,05 + 60*0,1 + 96*0,15 + 168*0,2)
Biên Bậc thuế 5	385	57,25 (60*0,05 + 60*0,1 + 96*0,15 + 168*0,2 + 1*0,25)
Biên Bậc thuế 5	624	117 (60*0,05 + 60*0,1 + 96*0,15 + 168*0,2 + 240*0,25)
Biên Bậc thuế 6	625	117,3 (60*0,05 + 60*0,1 + 96*0,15 + 168*0,2 + 240*0,25 + 1*0,3)
Biên Bậc thuế 6	960	217,8 (60*0,05 + 60*0,1 + 96*0,15 + 168*0,2 + 240*0,25 + 336*0,3)
Biên Bậc thuế 7	961	218,15 (60*0,05 + 60*0,1 + 96*0,15 + 168*0,2 + 240*0,25 + 336*0,3 + 1*0,35)

b. Phân tích lớp tương đương

- Valid:
  + Bậc thuế 1: thu nhập mỗi năm <= 60
  + Bậc thuế 2: 60 < thu nhập mỗi năm <= 120
  + Bậc thuế 3: 120 < thu nhập mỗi năm <= 216
  + Bậc thuế 4: 216 < thu nhập mỗi năm <= 384
  + Bậc thuế 5: 384 < thu nhập mỗi năm <= 624
  + Bậc thuế 6: 624 < thu nhập mỗi năm <= 960
  + Bậc thuế 7: 960 < thu nhập mỗi năm

→ 7 test case

- Invalid:
  + Bậc thuế 1: thu nhập mỗi năm > 60
  + Bậc thuế 2: thu nhập mỗi năm > 120
    hoặc thu nhập mỗi năm <= 60
  + Bậc thuế 3: thu nhập mỗi năm > 216
    hoặc thu nhập mỗi năm <= 120
  + Bậc thuế 4: thu nhập mỗi năm > 384
    hoặc thu nhập mỗi năm <= 216
  + Bậc thuế 5: thu nhập mỗi năm > 624
    hoặc thu nhập mỗi năm <= 384

+ Bậc thuế 6: thu nhập mỗi năm > 960
  hoặc thu nhập mỗi năm <= 624
+ Bậc thuế 7: thu nhập mỗi năm <= 960

→ 12 test case

Lớp tương đương	Input	Kết quả mong muốn
Valid Bậc thuế 1	30	Tính đến bậc thuế 1
Valid Bậc thuế 2	100	Tính đến bậc thuế 2
Valid Bậc thuế 3	200	Tính đến bậc thuế 3
Valid Bậc thuế 4	300	Tính đến bậc thuế 4
Valid Bậc thuế 5	500	Tính đến bậc thuế 5
Valid Bậc thuế 6	800	Tính đến bậc thuế 6
Valid Bậc thuế 7	1000	Tính đến bậc thuế 7
Invalid Bậc thuế 1	100	Khác tính đến bậc thuế 1
Invalid Bậc thuế 2	200	Khác tính đến bậc thuế 2
Invalid Bậc thuế 2	50	Khác tính đến bậc thuế 2
Invalid Bậc thuế 3	300	Khác tính đến bậc thuế 3
Invalid Bậc thuế 3	50	Khác tính đến bậc thuế 3
Invalid Bậc thuế 4	800	Khác tính đến bậc thuế 4
Invalid Bậc thuế 4	150	Khác tính đến bậc thuế 4
Invalid Bậc thuế 5	900	Khác tính đến bậc thuế 5
Invalid Bậc thuế 5	300	Khác tính đến bậc thuế 5
Invalid Bậc thuế 6	1000	Khác tính đến bậc thuế 6
Invalid Bậc thuế 6	500	Khác tính đến bậc thuế 6
Invalid Bậc thuế 7	600	Khác tính đến bậc thuế 7

Pregunta 2.23: Lãi suất tiền gửi theo năm của khách hàng cá nhân tại ngân hàng X được cho ở bảng dưới. Lãi được tính trên số ngày thực tế.

Kỳ hạn	**VND**	**EUR**	**USD**
Tiết kiệm			
Không kỳ hạn	**1.20 %**	**0.01 %**	**0.10 %**
7 ngày	**1.20 %**		
14 ngày	**1.20 %**		
1 tháng	**5.00 %**	**0.10 %**	**1.20 %**
2 tháng	**6.50 %**	**0.10 %**	**1.20 %**
3 tháng	**6.80 %**	**0.20 %**	**1.20 %**
6 tháng	**7.00 %**	**0.30 %**	**1.20 %**
9 tháng	**7.00 %**	**0.40 %**	**1.20 %**
12 tháng	**7.50 %**	**0.50 %**	**1.20 %**
24 tháng	**8.00 %**	**0.80 %**	**1.20 %**

(a) Thiết kế các ca kiểm thử dùng kỹ thuật phân tích giá trị biên
(b) Thiết kế các ca kiểm thử dùng kỹ thuật phân vùng tương đương

a. Phân tích giá trị biên

- Valid:
  + Không kỳ hạn : 0 ngày <= Thời gian gửi → Biên 0 ngày
  + Kỳ hạn 7 ngày : 7 ngày <= Thời gian gửi → Biên 7 ngày
  + Kỳ hạn 14 ngày: 14 ngày <= Thời gian gửi → Biên 14 ngày
  + Kỳ hạn 1 tháng : 1 tháng <= Thời gian gửi → Biên 1 tháng
  + Kỳ hạn 2 tháng : 2 tháng <= Thời gian gửi → Biên 2 tháng
  + Kỳ hạn 3 tháng : 3 tháng <= Thời gian gửi → Biên 3 tháng
  + Kỳ hạn 6 tháng : 6 tháng <= Thời gian gửi → Biên 6 tháng
  + Kỳ hạn 9 tháng : 9 tháng <= Thời gian gửi → Biên 9 tháng
  + Kỳ hạn 12 tháng : 12 tháng <= Thời gian gửi → Biên 12 tháng
  + Kỳ hạn 24 tháng : 24 tháng <= Thời gian gửi → Biên 24 tháng

→ 10 test case

- Invalid:
  + Không kỳ hạn : 0 ngày > Thời gian gửi → Biên -1 ngày
  + Kỳ hạn 7 ngày : 7 ngày > Thời gian gửi → Biên 6 ngày
  + Kỳ hạn 14 ngày: 14 ngày > Thời gian gửi → Biên 13 ngày
  + Kỳ hạn 1 tháng : 1 tháng > Thời gian gửi → Biên 29 ngày
  + Kỳ hạn 2 tháng : 2 tháng > Thời gian gửi → Biên 1 tháng 29 ngày
  + Kỳ hạn 3 tháng : 3 tháng > Thời gian gửi → Biên 2 tháng 29 ngày
  + Kỳ hạn 6 tháng : 6 tháng > Thời gian gửi → Biên 5 tháng 29 ngày
  + Kỳ hạn 9 tháng : 9 tháng > Thời gian gửi → Biên 8 tháng 29 ngày
  + Kỳ hạn 12 tháng : 12 tháng > Thời gian gửi → Biên 11 tháng 29 ngày
  + Kỳ hạn 24 tháng : 24 tháng > Thời gian gửi → Biên 22 tháng 29 ngày

→ 10 test case

Biên	Input	Kết quả mong muốn
Valid Không kỳ hạn	0 ngày	Tính theo lãi Không kỳ hạn
Valid Kỳ hạn 7 ngày	7 ngày	Tính theo lãi Kỳ hạn 7 ngày
Valid Kỳ hạn 14 ngày	14 ngày	Tính theo lãi Kỳ hạn 14 ngày
Valid Kỳ hạn 1 tháng	1 tháng	Tính theo lãi Kỳ hạn 1 tháng
Valid Kỳ hạn 2 tháng	2 tháng	Tính theo lãi Kỳ hạn 2 tháng
Valid Kỳ hạn 3 tháng	3 tháng	Tính theo lãi Kỳ hạn 3 tháng
Valid Kỳ hạn 6 tháng	6 tháng	Tính theo lãi Kỳ hạn 6 tháng
Valid Kỳ hạn 9 tháng	9 tháng	Tính theo lãi Kỳ hạn 9 tháng
Valid Kỳ hạn 12 tháng	12 tháng	Tính theo lãi Kỳ hạn 12 tháng
Valid Kỳ hạn 24 tháng	24 tháng	Tính theo lãi Kỳ hạn 24 tháng
Invalid Không kỳ hạn	-1 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 7 ngày	6 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 14 ngày	13 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 1 tháng	29 ngày	Tính theo lãi Không kỳ hạn

Invalid Kỳ hạn 2 tháng	1 tháng 29 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 3 tháng	2 tháng 29 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 6 tháng	5 tháng 29 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 9 tháng	8 tháng 29 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 12 tháng	11 tháng 29 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 24 tháng	23 tháng 29 ngày	Tính theo lãi Không kỳ hạn

b. Phân tích lớp tương đương

- Valid:
  + Không kỳ hạn : 0 ngày <= Thời gian gửi
  + Kỳ hạn 7 ngày : 7 ngày <= Thời gian gửi
  + Kỳ hạn 14 ngày: 14 ngày <= Thời gian gửi
  + Kỳ hạn 1 tháng : 1 tháng <= Thời gian gửi
  + Kỳ hạn 2 tháng : 2 tháng <= Thời gian gửi
  + Kỳ hạn 3 tháng : 3 tháng <= Thời gian gửi
  + Kỳ hạn 6 tháng : 6 tháng <= Thời gian gửi
  + Kỳ hạn 9 tháng : 9 tháng <= Thời gian gửi
  + Kỳ hạn 12 tháng : 12 tháng <= Thời gian gửi
  + Kỳ hạn 24 tháng : 24 tháng <= Thời gian gửi

- Invalid:
  + Không kỳ hạn : 0 ngày > Thời gian gửi
  + Kỳ hạn 7 ngày : 7 ngày > Thời gian gửi
  + Kỳ hạn 14 ngày: 14 ngày > Thời gian gửi
  + Kỳ hạn 1 tháng : 1 tháng > Thời gian gửi
  + Kỳ hạn 2 tháng : 2 tháng > Thời gian gửi
  + Kỳ hạn 3 tháng : 3 tháng > Thời gian gửi
  + Kỳ hạn 6 tháng : 6 tháng > Thời gian gửi
  + Kỳ hạn 9 tháng : 9 tháng > Thời gian gửi
  + Kỳ hạn 12 tháng : 12 tháng > Thời gian gửi
  + Kỳ hạn 24 tháng : 24 tháng > Thời gian gửi

Biên	Input	Kết quả mong muốn
Valid Không kỳ hạn	1 ngày	Tính theo lãi Không kỳ hạn
Valid Kỳ hạn 7 ngày	8 ngày	Tính 7 ngày theo lãi Kỳ hạn 7 ngày 1 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 14 ngày	30 ngày	Tính 28 ngày theo lãi Kỳ hạn 14 ngày 2 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 1 tháng	4 tháng 6 ngày	Tính 4 tháng theo lãi Kỳ hạn 1 tháng 6 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 2 tháng	5 tháng 15 ngày	Tính 4 tháng theo lãi Kỳ hạn 2 tháng 1 tháng 15 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 3 tháng	8 tháng 29 ngày	Tính 6 tháng theo lãi Kỳ hạn 3 tháng

		2 tháng 29 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 6 tháng	7 tháng 8 ngày	Tính 6 tháng theo lãi Kỳ hạn 6 tháng 1 tháng 8 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 9 tháng	9 tháng 6 ngày	Tính 9 tháng theo lãi Kỳ hạn 9 tháng 6 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 12 tháng	15 tháng 12 ngày	Tính 12 tháng theo lãi Kỳ hạn 12 tháng 3 tháng 12 ngày tính theo lãi Không kỳ hạn
Valid Kỳ hạn 24 tháng	32 tháng 22	Tính 24 tháng theo lãi Kỳ hạn 24 tháng 8 tháng 22 ngày tính theo lãi Không kỳ hạn
Invalid Không kỳ hạn	-1 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 7 ngày	5 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 14 ngày	12 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 1 tháng	25 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 2 tháng	1 tháng 3 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 3 tháng	2 tháng 29 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 6 tháng	3 tháng 8 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 9 tháng	5 tháng 12 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 12 tháng	9 tháng 5 ngày	Tính theo lãi Không kỳ hạn
Invalid Kỳ hạn 24 tháng	20 tháng 20 ngày	Tính theo lãi Không kỳ hạn

Pregunta 2.24: Một chương trình phân loại tam giác đọc vào các giá trị số nguyên trong khoảng [0,100]. 3 giá trị A, B, và C được dùng để biểu diễn độ dài 3 cạnh tam giác. Chương trình in ra thông báo 3 giá trị này có phải là 3 cạnh tam giác không.
(a) Thiết kế các ca kiểm thử dùng kỹ thuật phân tích giá trị biên
(b) Thiết kế các ca kiểm thử dùng kỹ thuật phân vùng tương đương

Pregunta 2.25: Một chương trình phân loại tam giác đọc vào các giá trị số nguyên trong khoảng [0,100]. 3 giá trị A, B, và C được dùng để biểu diễn độ dài 3 cạnh tam giác. Chương trình in ra thông báo 3 giá trị này có phải là 3 cạnh tam giác cân không.
(a) Thiết kế các ca kiểm thử dùng kỹ thuật phân tích giá trị biên
(b) Thiết kế các ca kiểm thử dùng kỹ thuật phân vùng tương đương

Pregunta 2.26: Một chương trình phân loại tam giác đọc vào các giá trị số nguyên trong khoảng [0,100]. 3 giá trị A, B, và C được dùng để biểu diễn độ dài 3 cạnh tam giác. Chương trình in ra thông báo 3 giá trị này có phải là 3 cạnh tam giác đều không.
(a) Thiết kế các ca kiểm thử dùng kỹ thuật phân tích giá trị biên
(b) Thiết kế các ca kiểm thử dùng kỹ thuật phân vùng tương đương

Pregunta 2.27: Một chương trình phân loại tam giác đọc vào các giá trị số nguyên trong khoảng [0,100]. 3 giá trị A, B, và C được dùng để biểu diễn độ dài 3 cạnh tam giác. Chương trình in ra thông báo 3 giá trị này có phải là 3 cạnh tam giác vuông không.
(a) Thiết kế các ca kiểm thử dùng kỹ thuật phân tích giá trị biên

(b) Thiết kế các ca kiểm thử dùng kỹ thuật phân vùng tương đương

Pregunta 3.1: Cho hàm tìm kiếm nhị phân viết bằng C. input array v đã được sắp xếp theo giá trị tăng dần, n là kích thước mảng, ta cần tìm chỉ số mảng của phần tử x. Nếu không tìm thấy x trong mảng, trả về giá trị -1.

```
int binSearch(int x, int v[], int n){
   int low, high, mid;
   low = 0;
   high = n - 1;
   while (low<=high){
      mid = (low + high)/2;
      if (x<v[mid])
         high = mid - 1;
      else if (x > v[mid])
         low = mid + 1;
      else
         return mid;
   }
   return -1;
}
```

a. Vẽ đồ thị luồng điều khiển

b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ câu lệnh

c. Bổ xung thêm đường (nếu cần) để bao phủ hết các ngã rẽ (branch)

d. Với mỗi đường xác định ở trên, tìm biểu thức tiền tố tương ứng

e. Giải biểu thức tiền tố trên để sinh ra các đầu vào ca kiểm thử và sau đó ước lượng đầu ra tương ứng

f. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.2: Giả mã bên dưới tính tổng các phần tử >0 của mảng a

```
 sum_of_all_positive_numbers(a, num_of_entries, sum)
1:       sum =0;
2:       init = 1;
3:       while(init <= num_of_entries)
4:          if a[init] > 0
5:            sum = sum + a[init]
6:          endif
7:          init = init + 1
```

8: endwhile
9:end sum_of_all_positive_numbers

a. Vẽ đồ thị luồng điều khiển

b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ câu lệnh

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9
1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 3 → 8 → 9

1 → 2 → 3 → 4 → 6 → 7 → 8 → 9
1 → 2 → 3 → 4 → 6 → 7 → 8 → 3 → 8 → 9

c. Bổ xung thêm đường (nếu cần) để bao phủ hết các ngã rẽ (branch)

d. Với mỗi đường xác định ở trên, tìm biểu thức tiền tố tương ứng

e. Giải biểu thức tiền tố trên để sinh ra đầu vào các ca kiểm thử và sau đó ước lượng đầu ra tương ứng
f. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.3: Hàm bên dưới trả về chỉ số phần tử cuối cùng trong x có giá trị bằng y. Nếu không tồn tại, trả về giá trị -1.

```
int findLast(int[] x, int y){
   for (int i = x.length -1; i > 0; i--){
      if (x[i] == y)
         return i;
   }
   return -1;
}
```

a. Vẽ đồ thị luồng điều khiển
b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ câu lệnh
c. Bổ xung thêm đường (nếu cần) để bao phủ hết các ngã rẽ (branch)
d. Với mỗi đường xác định ở trên, tìm biểu thức tiền tố tương ứng

e. Giải biểu thức tiền tố trên để sinh ra đầu vào các ca kiểm thử và sau đó ước lượng đầu ra tương ứng
f. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.4: Hàm bên dưới trả về chỉ số phần tử cuối cùng trong x có giá trị bằng 0. Nếu không tồn tại, trả về giá trị -1.

```
int lastZero(int[] x){
   for (int i = 0; i < x.length; i++){
      if (x[i] == 0)
         return i;
   }
   return -1;
}
```

a. Vẽ đồ thị luồng điều khiển
b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ câu lệnh
c. Bổ xung thêm đường (nếu cần) để bao phủ hết các ngã rẽ (branch)
d. Với mỗi đường xác định ở trên, tìm biểu thức tiền tố tương ứng
e. Giải biểu thức tiền tố trên để sinh ra đầu vào các ca kiểm thử và sau đó ước lượng đầu ra tương ứng
f. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.5: Hàm bên dưới trả về số phần tử là số >0.

```
int countPositive(int[] x){
   int count = 0;
   for (int i = 0 ; i < x.length; i++){
      if (x[i] >=0)
         count++;
   }
   return count;
}
```

a. Vẽ đồ thị luồng điều khiển
b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ câu lệnh
c. Bổ xung thêm đường (nếu cần) để bao phủ hết các ngã rẽ (branch)
d. Với mỗi đường xác định ở trên, tìm biểu thức tiền tố tương ứng
e. Giải biểu thức tiền tố trên để sinh ra đầu vào các ca kiểm thử và sau đó ước lượng đầu ra tương ứng

f. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.6: Cho đoạn code

```
public static void f1 (int x, int y) {
        if (x < y) { f2 (y); }
        else { f3 (y); };
}
public static void f2 (int a) {
        if (a % 2 == 0) {
                f3 (2*a);
        };
}
public static void f3 (int b) {
        if (b > 0) { f4(); }
        else { f5(); };
}
public static void f4()
        {... f6()....}
public static void f5()
        {... f6()....}
 public static void f6()
        {...}
```

a. Vẽ đồ thị luồng điều khiển
b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ câu lệnh
c. Bổ xung thêm đường (nếu cần) để bao phủ hết các ngã rẽ (branch)
d. Với mỗi đường xác định ở trên, tìm biểu thức tiền tố tương ứng
e. Giải biểu thức tiền tố trên để sinh ra đầu vào các ca kiểm thử và sau đó ước lượng đầu ra tương ứng
f. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.7: Cho hàm tìm kiếm nhị phân viết bằng C. Input array v đã được sắp xếp theo giá trị tăng dần, n là kích thước mảng, ta cần tìm chỉ số mảng của phần tử x. Nếu không tìm thấy x trong mảng, trả về giá trị -1.

1. **int binSearch(int x, int v[], int n){**
2. **int low, high, mid;**
   low = 0;
   high = n - 1;
3. **while (low<=high){**

```
4.  mid = (low + high)/2;
5.  if (x<v[mid])
6.  high = mid - 1;
7.  else if (x > v[mid])
8.          low = mid + 1;
9.  else
10. return mid;
11.}
12.return -1;
13.}
```

Cho đầu vào các ca kiểm thử dưới đây:

t1= (x=1, v={1,2,5,7,9},n=5) 1,2,3,4,5,6,11, 3,4,5,6,11, 3,4,5,7,9,10,13

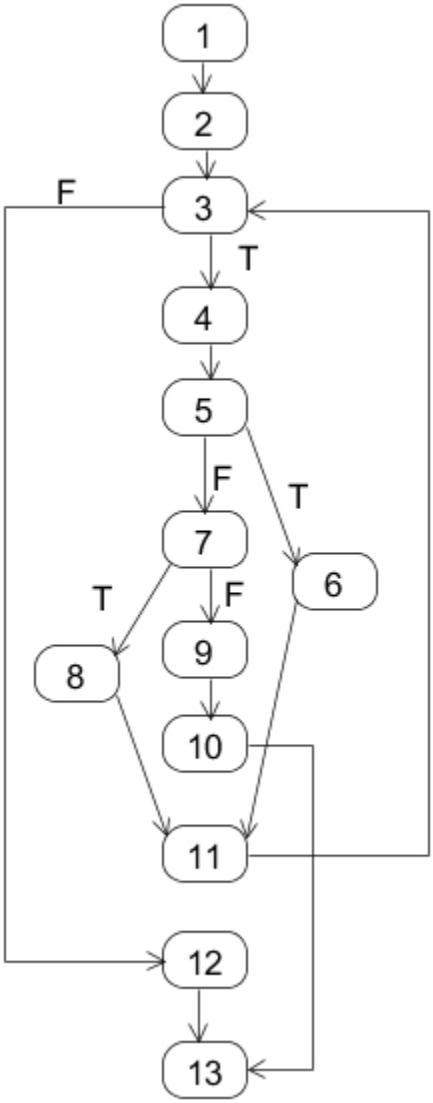
t2= (x=3, v={1,3,9},n=3)

t3= (x=9, v={1,2,5,7,9},n=5)

t4= (x=4, v={1,2,5,7,9},n=5)

t5= (x=10, v={1,2,5,7,9},n=5)

a. **Vẽ đồ thị luồng điều khiển**



b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào trên

t1= (x=1, v={1,2,5,7,9},n=5)

→ Đường trên đồ thị luồng điều khiển tương ứng với đầu vào t1: 1,2,3,4,5,6, 3,4,5,6, 3,4,5,7,9,10,13

…

c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh

1,2,3,12,13 : t6= (x=1, v={null},n=0)

1,2,3,4,5,6,11, 3,4,5,7,9,10,13 : t7= (x=2, v={2,5,7},n=3)

1,2,3,4,5,7,8,11, 3,4,5,7,9,10,13: t8= (x=5, v={1,2,5},n=3)

d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ

(Tương tự câu c)

1,2,3,12,13 : t6= (x=1, v={null},n=0)

1,2,3,4,5,6,11, 3,4,5,7,9,10,13 : t7= (x=2, v={2,5,7},n=3)

1,2,3,4,5,7,8,11, 3,4,5,7,9,10,13: t8= (x=5, v={1,2,5},n=3)

e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường với n = 4

1,2,3,4,5,7,8,11, 3,4,5,7,8,11, 3,12,13 : t6= (x=4, v={0,1,2,3},n=4)

1,2,3,4,5,6,11, 3,4,5,7,9,10,13 : t7= (x=2, v={1,2,5,7},n=4)

1,2,3,4,5,7,8,11, 3,4,5,7,9,10,13: t8= (x=5, v={0,1,2,5},n=4)

f. Liệu tất cả các đường tương ứng ở e có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.8: Giả mã bên dưới tính tổng các số dương của mảng a

```
sum_of_all_positive_numbers(a, num_of_entries, sum)
        sum =0;
        init = 1;
        while(init < num_of_entries)
          if a[init] > 0
                sum = sum + a[init]
          endif
          init = init + 1
        endwhile
end sum_of_all_positive_numbers
```

Cho đầu vào ca kiểm thử dưới đây:

t1= (a={0,2,-5,7,-9}, num_of_entries=5)

t2= (a={1,-3,9}, num_of_entries=3)

t3= (a={1,-2,5,7,0}, num_of_entries=5)

t4= (a={-1,2,0,7,-9}, num_of_entries=5)

t5= (a={1,-2,5,7,9}, num_of_entries=5)

a. Vẽ đồ thị luồng điều khiển

b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử

c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh

d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ

e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường với num_of_entries = 4

f. Liệu tất cả các đường tương ứng ở e có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.9: Hàm dưới đây tính số ngày giữa 2 ngày/tháng cho trước trong cùng 1 năm. Biết

1<=month1, month2<=12

1<=day1,day2<=31;

1<=year<=10000

```
public static int cal (int month1, int day1, int month2, int day2, int year) {
        int numDays;
        if (month2 == month1)
                numDays = day2 - day1;
        else {
                // bo qua thang thu  0.
                int daysIn[] = {0, 31, 0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
                // kiem tra nam nhuan
                int m4 = year % 4;
                int m100 = year % 100;
                int m400 = year % 400;
                if ((m4 != 0) || ((m100 ==0) && (m400 != 0)))
                        daysIn[2] = 28;
                else
                        daysIn[2] = 29;
                // bắt đầu tính từ ngày
                numDays = day2 + (daysIn[month1] - day1);
                //cộng thêm số ngày ở khoảng giữa các tháng
                for (int i = month1 + 1; i <= month2-1; i++)
                        numDays = daysIn[i] + numDays;
                }
        return (numDays);
}
```

Cho đầu vào ca kiểm thử dưới đây:
t1= (12,20,12,30,2013)
t2= (2,10,11,1,2013)
t3= (1,1,12,1,2004)
t4= (2,20,3,1,2004)
t5= (2,20,3,1,1900}

a. Vẽ đồ thị luồng điều khiển
b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử
c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh
d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ
e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường với month1 = 2
f. Liệu tất cả các đường tương ứng ở e có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.10: Hàm bên dưới trả về chỉ số phần tử cuối cùng trong x có giá trị bằng y. Nếu không tồn tại, trả về giá trị -1.

```
int findLast(int[] x, int y){
```

```
   for (int i = x.length -1; i > 0; i--){
      if (x[i] == y)
         return i;
   }
   return -1;
}
```

Cho đầu vào ca kiểm thử dưới đây:
t1= (x={5}, y=5)
t2= (x={1,-3,5}, y=5)
t3= (x={5,-2,5,7,0}, y=5)
t4= (x={-1,2,0,5,-9}, y=5)
t5= (x={1,-2,3,7,9}, y=5)
a. Vẽ đồ thị luồng điều khiển
b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử
c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh
d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ
e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường với số phần tử của mảng x = 4
f. Liệu tất cả các đường tương ứng ở e có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.11: Hàm bên dưới trả về chỉ số phần tử cuối cùng trong x có giá trị bằng 0. Nếu không tồn tại, trả về giá trị -1.

```
int lastZero(int[] x){
   for (int i = 0; i < x.length; i++){
      if (x[i] == 0)
         return i;
   }
   return -1;
}
```

Cho đầu vào ca kiểm thử dưới đây:
t1= (x={5})
t2= (x={0})
t3= (x={5,-2,5,7,0})
t4= (x={-1,2,0,5,-9})
t5= (x={0,-2,3,7,9})
a. Vẽ đồ thị luồng điều khiển
b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử
c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh
d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ
e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường với số phần tử của mảng x = 4

f. Liệu tất cả các đường tương ứng ở e có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.12: Hàm bên dưới trả về số phần tử là số >0.

```
int countPositive(int[] x){
   int count = 0;
   for (int i = 0 ; i < x.length; i++){
      if (x[i] >=0)
         count++;
   }
   return count;
}
```

Cho đầu vào ca kiểm thử dưới đây:
t1= (x={5})
t2= (x={0})
t3= (x={5,-2,5,7,0})
t4= (x={-1,2,0,5,-9})
t5= (x={0,-2,3,7,9})
a. Vẽ đồ thị luồng điều khiển
b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử
c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh
d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ
e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường với số phần tử của mảng x = 4
f. Liệu tất cả các đường tương ứng ở e có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.13: Phương thức printPrimes bên dưới tìm và in ra n số nguyên tố.

```
private static void printPrimes (int n)
{
        int curPrime;
        int numPrimes;
        boolean isPrime;
        int [] primes = new int [MAXPRIMES];
        primes [0] = 2;
        numPrimes = 1;
        curPrime = 2;
        while (numPrimes < n)
        {
                curPrime++;
                isPrime = true;
```

```
            for (int i = 0; i <= numPrimes-1; i++)
            {
                if (isDivisible (primes[i], curPrime))
                {
                    isPrime = false;
                     break;
                }
            }
            if (isPrime)
            {
                primes[numPrimes] = curPrime;
                numPrimes++;
            }
        }

        for (int i = 0; i <= numPrimes-1; i++)
        {
        System.out.println ("Prime: " + primes[i]);
        }
        }
```

Cho đầu vào ca kiểm thử dưới đây:
t1= (n=0)
t2= (n=1)
t3= (n=2)
t4= (n=3)
a. Vẽ đồ thị luồng điều khiển
b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử
c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh
d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ

Pregunta 3.14: Cho đoạn code

```
public static void f1 (int x, int y) {
        if (x < y) { f2 (y); }
        else { f3 (y); };
}
public static void f2 (int a) {
        if (a % 2 == 0) {
                f3 (2*a);
        };
```

```
}
public static void f3 (int b) {
        if (b > 0) { f4(); }
        else { f5(); };
}
public static void f4()
        {... f6()....}
public static void f5()
        {... f6()....}
 public static void f6()
        {...}
```

Sử dụng đầu vào ca kiểm thử dưới đây:

t1= f1(0, 0)
t2= f1(1, 1)
t3= f1(0, 1)
t4= f1(3, 2)
t5= f1(3, 4)

a. Vẽ đồ thị luồng điều khiển
b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử
c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết nút
d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết cạnh
e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường

Pregunta 3.15:

```
public void foo2(int a, int b, int x) {
  if (a>1 && b==0) {
        x=x/a;
  }
  for ( int i = 1; i < 3; i++){
        if true x=x+1;
  }
}
```

a. Vẽ đồ thị luồng điều khiển
b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ ngã rẽ (branch)
c. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.
d. Xác định test case tương ứng với các đường khả thi

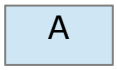
Pregunta 3.16:

```
sum(a, numEntry,sum){
                sum = 0;
```

```
        for (init =1; init <=numEntry;init++)
            if (a[init]>0)
                sum = sum + a[init];
        if (false)
            sum = 0;
}
```

a. Vẽ đồ thị luồng điều khiển
b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ ngã rẽ (branch)
c. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.
d. Xác định test case tương ứng với các đường khả thi

Pregunta 3.17: Cho sơ đồ gọi các module như sau



(còn thiếu)

a. Xác định thứ tự tích hợp và các stub/driver (nếu cần) khi dùng kỹ thuật topdown
b. Xác định thứ tự tích hợp và các stub/driver (nếu cần) khi dùng kỹ thuật bottom up
c. Xác định thứ tự tích hợp và các stub/driver (nếu cần) khi dùng kỹ thuật sandwich

Pregunta 3.18: Cho code

```
1. int modifiedbinsearch(int X, int V[], int n){
2. int low, high, mid;
3. low = 0;
4. high = n - 1;
5. while (low <= high) {
6.    mid = (low + high)/2;
7.    if (X < V[mid]) {
8.          high = mid - 1;
9.          mid = mid - 1; }
```

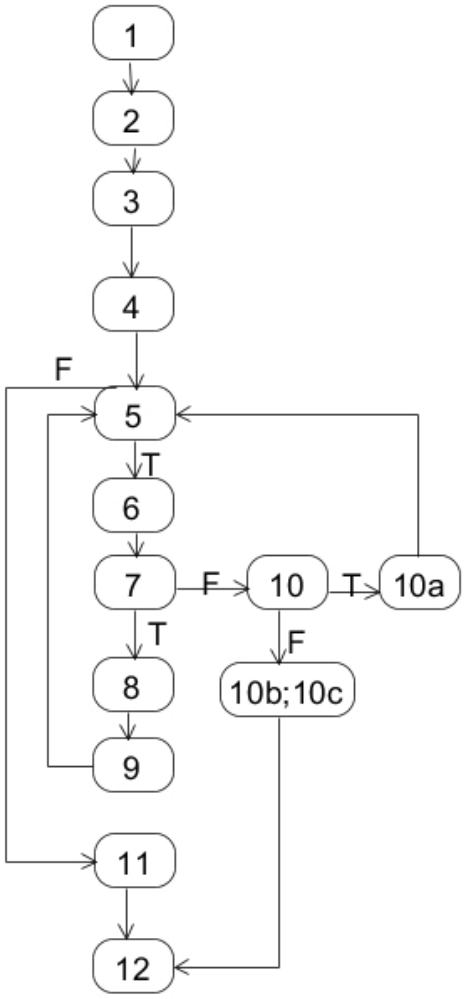
```
10.     else if (X > V[mid])
            a. low = mid + 1;
        b. else
            c. return mid;
    d. }
11. return -1;
12. }
```

a. Vẽ đồ thị luồng dữ liệu.

b. Xác định điểm bất thường (anomaly) của code trên

c. Giả sử mảng V[] có ít nhất 1 phần tử, xác định đường không khả thi.

a. Vẽ đồ thị luồng dữ liệu.



b. Xác định điểm bất thường (anomaly) của code trên :
Câu lệnh 9 (sẽ ko có tác dụng)

c. Giả sử mảng V[] có ít nhất 1 phần tử, xác định đường không khả thi.
1-2-3-4-5-11-12

Pregunta 3.19: Chương trình SquaresLoopRange(*start-number, stop-number*) hiển thị bình phương của 1 dãy số từ *start-number* tới *stop-number*. Nếu *start-number* lớn hơn *stop-number* error message cần được hiển thị: Start-limit greater than stop-limit!Sq

a. Viết chương trình

b. Thiết kế test cases

c. Viết code JUnit tương ứng

Pregunta 3.20: Viết chương trình MultiplesLoopRange(*start-number, stop-number, num*) hiển thị dãy số trong khoảng [*start-number, stop-number*] và dãy số phải là bội số của num.
Nếu *start-number* lớn hơn *stop-number*, chương trình sẽ hiển thị dãy giảm dần.

a. Viết chương trình

b. Thiết kế test cases

c. Viết code JUnit tương ứng

Pregunta 3.21:

Hệ thống ghi lại nhật ký nhiệt độ theo thời gian. Nhưng output ở một format riêng, bao gồm một dãy các symbols, đầu tiên là 1 số biểu diễn nhiệt độ bắt đầu, ký hiệu tiếp biểu diễn sự thay đổi nhiệt độ so với trước đó. Các symbols được giải mã như sau:

- **'.' không thay đổi**
- **'+' tăng 1 độ so với trước nó**
- **'-' giảm 1 độ so với trước nó**

Các giá trị được biên dịch thành các số kiểu int.

Ta cần tính median của dữ liệu nhiệt độ. Đầu tiên, ta cần sắp xếp. Sau đó:

- **Nếu mảng chứa số lẻ phần tử n, median là phần tử chính giữa:phần tử (*n*-1)/2 của mảng.**
- **Nếu mảng chứa số chẵn phẩn tử, median là giá trị trung bình của 2 phần tử *n*/2 và (*n*/2)-1 của mảng.**

Lưu ý: nhiệt độ là integer, nhưng giá trị median là float.

a. Viết chương trình TempMedian

b. Thiết kế test cases

c. Viết code JUnit tương ứng

Pregunta 3.22:

DNA được tạo bởi 2 *DNA-Strands (chuỗi)*, chúng xoắn với nhau tạo thành 1 *double helix*.
Mỗi chuỗi DNA là chuỗi các bases. Gồm 4 bases:

- **adenine (abbreviated A)**
- **cytosine (C)**
- **guanine (G)**
- **thymine (T)**

Bases có cặp: A bắt cặp với T, C bắt cặp với G. Ví dụ, một sợi là A-C-G-G-T-C

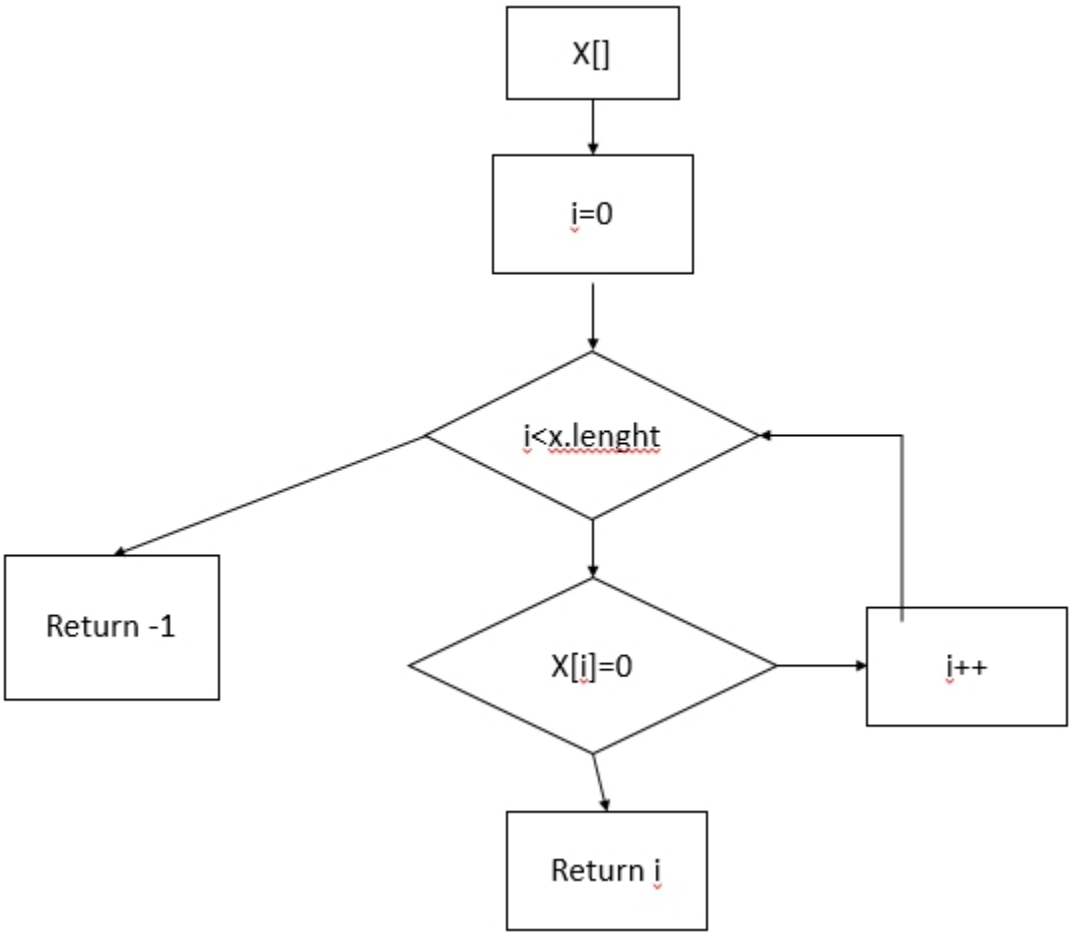
Sợi còn lại sẽ là : T-G-C-C-A-G
Vì ta có các cặp A:T, C:G, G:C, G:C, T:A và C:G. Cách lưu trữ thông tin như vậy là dư thừa, nhưng ta có thể phân đôi 1 double helix và bỏ 1 phía đi, ta vẫn có thể tái tạo lại nó. Ngoài ra, nếu là muốn nhân đôi double helix, ta có thể chia thành 2 chuỗi, tại tạo lại mỗi phía và sẽ thu được 2 bản copy từ bản gốc.
Ta cần tạo 1 class để biểu diễn 1 chuỗi DNA. API cho class DNAStrand:
public DNAStrand(String dna): khởi tạo
public boolean isValidDNA(): Trả về true nếu DNA là valid, nghĩa là chỉ có các ký tự hoa A, T, C, G và chứa ít nhất 1 ký tự.
public String complementWC(): Trả về *Watson Crick complement*, là chuỗi DNA bù – sợi còn lại trong double helix. Thay T bằng A, A bằng T, C bằng G và G bằng C.
public String palindromeWC(): Trả về *Watson Crick Palindrome*, chuỗi đảo của chuỗi DNA bù.
public boolean containsSequence(String seq): Trả về true nếu DNA chứa chuỗi con seq.
public String toString(): Trả về string DNA.
a. Viết chương trình TempMedian
b. Thiết kế test cases
c. Viết code JUnit tương ứng

Pregunta 3.11: Hàm bên dưới trả về chỉ số phần tử cuối cùng trong x có giá trị bằng 0. Nếu không tồn tại, trả về giá trị -1.

```
int lastZero(int[] x){
   for (int i = 0; i < x.length; i++){
      if (x[i] == 0)
         return i;
   }
   return -1;
}
```

Cho đầu vào ca kiểm thử dưới đây:
t1= (x={5})
t2= (x={0})
t3= (x={5,-2,5,7,0})
t4= (x={-1,2,0,5,-9})
t5= (x={0,-2,3,7,9})

a. Vẽ đồ thị luồng điều khiển



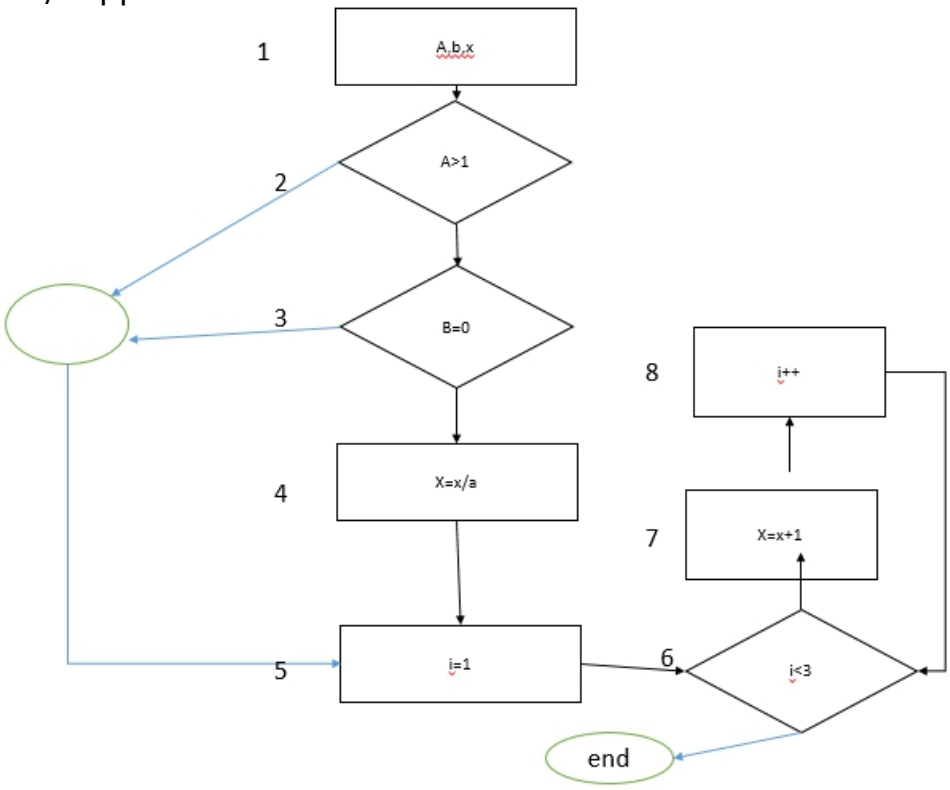
b. Chỉ ra đường trên đồ thị luồng điều khiển tương ứng với mỗi đầu vào ca kiểm thử
c. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết câu lệnh
d. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết ngã rẽ
e. Tìm tập đầu vào ca kiểm thử nhỏ nhất để bao phủ hết đường với số phần tử của mảng x = 4
f. Liệu tất cả các đường tương ứng ở e có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

Pregunta 3.15:

```
public void foo2(int a, int b, int x) {

  if (a>1 && b==0) {
        x=x/a;
  }
  for ( int i = 1; i < 3; i++){
        if true x=x+1;
  }
}
```

a. Vẽ đồ thị luồng điều khiển



b. Từ đồ thị luồng điều khiển, xác định tập các đường từ đầu vào tới đầu ra để bao phủ được toàn bộ ngã rẽ (branch) : có đường tất cả

1-2-5-6-end

1-2-5-6-7-8-6-end

1-2-3-5-6-end

1-2-3-6-7-8-6-end

1-2-3-4-5-6-end

1-2-3-4-5-6-7-8-6-end

c. Liệu tất cả các đường trên có khả thi hay không? Nếu không chỉ ra những đường không khả thi.

d. Xác định test case tương ứng với các đường khả thi

Pregunta 3.19: Chương trình SquaresLoopRange(*start-number, stop-number*) hiển thị bình phương của 1 dãy số từ *start-number* tới *stop-number*. Nếu *start-number* lớn hơn *stop-number* error message cần được hiển thị: Start-limit greater than stop-limit!Sq

a. Viết chương trình
b. Thiết kế test cases
c. Viết code JUnit tương ứng

a. code

```
public class Squares {

    int startNum;

    int stopNum;


    public Squares(int startNum, int stopNum) {


        this.startNum = startNum;

        this.stopNum = stopNum;

    }

    public String show(){

        if(startNum <= stopNum){

          for(int i= startNum;i<= stopNum;i++){

            System.out.print(i*i+" ");

        }

            System.out.println("\n");

          return "1";

        }

        System.out.println("Start-limit greater than stop-limit!");

        return "0";


    }
```

}

b. test case :
:java SquaresLoopRange 10 20

100 121 144 169 196 225 256 289 324 361 400

:java SquaresLoopRange 1 10

1 4 9 16 25 36 49 64 81 100

Nếu *start-number* lớn hơn *stop-number* error message

cần được hiển thị:

:java SquaresLoopRange 10 1

Start-limit greater than stop-limit!
c. JUnit :

```
public SquaresTest() {
   }

   @BeforeClass
   public static void setUpClass() {
   }

   @AfterClass
   public static void tearDownClass() {
   }

   @Before
   public void setUp() throws Exception {

   }

   @After
   public void tearDown() {

   }

     @Test
     public void testCase1(){
        int a = 10;
        int b = 20;
        System.out.println(":java SquaresLoopRange "+a+" "+b);
        Squares sq=new Squares(a, b);
        Assert.assertEquals("1", sq.show());
```

```
    }
    @Test
    public void testCase2(){
       int a=1;
       int b=10;
       System.out.println(":java SquaresLoopRange "+a+" "+b);
       Squares sq=new Squares(a, b);
       Assert.assertEquals("1", sq.show());
    }
    @Test
    public void testCase3(){
       int a=10;
       int b=1;
       System.out.println(":java SquaresLoopRange "+a+" "+b);
       Squares sq=new Squares(a, b);
       Assert.assertEquals("1", sq.show());
    }
```

Pregunta 3.20: Viết chương trình MultiplesLoopRange(*start-number, stop-number, num*) hiển thị dãy số trong khoảng [*start-number, stop-number*] và dãy số phải là bội số của num.
Nếu *start-number* lớn hơn *stop-number*, chương trình sẽ hiển thị dãy giảm dần.
a. Viết chương trình
b. Thiết kế test cases
c. Viết code JUnit tương ứng
a. code

```
public class MultiplesLoopRange {
    int startNum;
   int stopNum;
   int number;

   public MultiplesLoopRange(int startNum, int stopNum, int number) {
      this.startNum = startNum;
      this.stopNum = stopNum;
      this.number = number;
   }
   public String show(){
      if(startNum <= stopNum){
        for(int i= startNum;i<= stopNum;i++){
            if(i% number ==0)
         System.out.print(i+" ");
      }
         System.out.println("\n");
        return "1";
      }
      for(int i= startNum;i>= stopNum;i--){
         if(i% number ==0)
         System.out.print(i+" ");
      }
```

```
        System.out.println("\n");
        return "0";

    }
}
```

b. test case :

: java MultiplesLoopRange 20 30 3

21 24 27 30

Nếu *start-number* lớn hơn *stop-number*, chương trình sẽ hiển thị dãy giảm dần. Ví dụ:

: java MultiplesLoopRange 30 20 3

30 27 24 21

Nếu ko có số nào là bội num thì in ra

: java MultiplesLoopRange 3 20 30

don’t have any number

c. JUnit

```
public class MultiplesLoopRangeTest {


    public MultiplesLoopRangeTest() {
    }


    @BeforeClass
    public static void setUpClass() {
    }


    @AfterClass
    public static void tearDownClass() {
    }
```

```
    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }
    @Test
     public void testCase1(){
         int a = 20;
         int b = 30;
         int c=3;
         System.out.println(": java MultiplesLoopRange "+a+" "+b);
         MultiplesLoopRange sq=new MultiplesLoopRange(a, b,c);
         Assert.assertEquals("1", sq.show());
     }
     @Test
     public void testCase2(){
         int a=30;
         int b=20;
         int c=3;
         System.out.println(": java MultiplesLoopRange "+a+" "+b);
         MultiplesLoopRange sq=new MultiplesLoopRange(a, b,c);
         Assert.assertEquals("0", sq.show());
     }
}
```

Pregunta 3.21:
Hệ thống ghi lại nhật ký nhiệt độ theo thời gian. Nhưng output ở một format riêng, bao gồm một dãy các symbols, đầu tiên là 1 số biểu diễn nhiệt độ bắt đầu, ký hiệu tiếp biểu diễn sự thay đổi nhiệt độ so với trước đó. Các symbols được giải mã như sau:
- '.' không thay đổi
- '+' tăng 1 độ so với trước nó
- '-' giảm 1 độ so với trước nó

Các giá trị được biên dịch thành các số kiểu int.
Ta cần tính median của dữ liệu nhiệt độ. Đầu tiên, ta cần sắp xếp nhiệt độ theo thứ tự. Sau đó:
- Nếu mảng chứa số lẻ phần tử n, median là phần tử chính giữa:phần tử thứ $(n+1)/2$.
- Nếu mảng chứa số chẵn phần tử, median là giá trị trung bình của 2 phần tử thứ $n/2$ và $(n/2)+1$.

Lưu ý: nhiệt độ là integer, nhưng giá trị median là float.
a. Viết chương trình TempMedian
b. Thiết kế test cases
c. Viết code JUnit tương ứng

a. Code
b. Test case
c. jUnit

a. code

```
public class TempMedian {

   String temp;


   public TempMedian(String temp) {

      this.temp = temp;

   }

   public float process(){

      String []word=temp.split(" ");

      int []result=new int[word.length];

      int startTemp;

      try{

         startTemp=Integer.parseInt(word[0]);

      }
```

```
        catch(Exception e){return 0;}
        result[0]=startTemp;
//Giải mã các ký hiệu
        for(int i=1;i<word.length;i++){
            if(word[i].equals(".")) result[i]=result[i-1];
            else if(word[i].equals("+")) result[i]=result[i-1]+1;
            else if(word[i].equals("-")) result[i]=result[i-1]-1;
        }
//In dãy nhiệt độ sau khi giải mã
        for(int i=0;i<word.length;i++)
            System.out.print(result[i]+" ");
        System.out.println("\n");
        int tem;  //Biến trung gian để sắp xếp
//Sắp xếp theo thứ tự tăng dần
        for(int i=0;i<word.length;i++){
            for(int j=i+1;j<word.length;j++)//Đưa giá trị nhỏ nhất còn lại về vị trí i
                if(result[i]> result[j]) {
                tem=result[i];
                result[i]=result[j];
                result[j]=tem;
                }
        }
//In dãy nhiệt độ sau khi sắp xếp
        for(int i=0;i<word.length;i++)
            System.out.print(result[i]+" ");
        System.out.println("\n");
```

```
//Nếu số lượng phần tử của dãy là lẻ -> median là phần tử chính giữa dãy
        if((result.length)%2==1) {
            System.out.println(""+result[(result.length-1)/2]);
            return result[(result.length-1)/2];
        }else { //Số lượng phần tử của dãy là chẵn -> median là 2 phần tử chính giữa dãy
            float medium=(float)(result[result.length/2] + result[(result.length/2)-1])/2;
            System.out.println(""+medium);
            return medium;
        }
    }
}
```

b. test case

: java TempMedian 34 . . . + . - - - . +

34 34 34 34 35 35 34 33 32 32 33

32 32 33 33 34 34 34 34 34 35 35

34.0

: java TempMedian 7 - - - . .

7 6 5 4 4 4

4 4 4 5 6 7

4.5

: java TempMedian 10 - + . - . - . +

10 9 10 10 9 9 8 8 9

8 8 9 9 9 9 10 10 10

9.0

: java TempMedian 3 - - - -

3 2 1 0 -1

-1 0 1 2 3

1.0

c. junit

```
public class TempMedianTest {


    public TempMedianTest() {

    }


    @BeforeClass

    public static void setUpClass() {

    }


    @AfterClass

    public static void tearDownClass() {

    }


    @Before

    public void setUp() {

    }


    @After

    public void tearDown() {

    }


    @Test

    public void testCase1(){

        String s="34 . . . + . - - - . +";
```

```
        TempMedian temp=new TempMedian(s);
         float medium=34;
        System.out.println(": java TempMedian "+s);
        Assert.assertEquals(medium, temp.process());
    }
     @Test
    public void testCase2(){
         String s="7 - - - . .";
        TempMedian temp=new TempMedian(s);
         float medium=(float) 4.5;
        System.out.println(": java TempMedian "+s);
        Assert.assertEquals(medium, temp.process());
    }
      @Test
    public void testCase3(){
          String s="10 - + . - . - . +";
        TempMedian temp=new TempMedian(s);
         float medium=9;
        System.out.println(": java TempMedian "+s);
        Assert.assertEquals(medium, temp.process());
    }
       @Test
    public void testCase4(){
           String s="3 - - - -";
        TempMedian temp=new TempMedian(s);
```

```
        float medium=1;

        System.out.println(": java TempMedian "+s);

        Assert.assertEquals(medium, temp.process());

    }

}
```