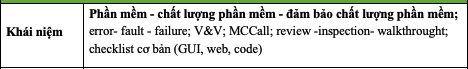
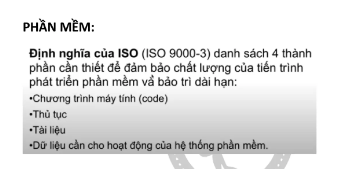
Test





1. **Chất lượng phần mềm là:**

* Mức độ mà một hệ thống, thành phần hay 1 tiến trình đáp ứng đặc tả theo yêu cầu
* Mức độ mà 1 hệ thống, thành phần hay tiến trình đáp ứng được nhu cầu/mong muốn của khách hàng/người dùng

1. **Đảm bảo chất lượng phần mềm là:**

* **Đảm bảo chất lượng phần mềm (SQA)** là một tập hợp các hoạt động được thực hiện để đảm bảo rằng phần mềm đáp ứng các tiêu chuẩn chất lượng. Các hoạt động SQA có thể bao gồm:
* **Lập kế hoạch và quản lý chất lượng:** Xác định các mục tiêu chất lượng, lập kế hoạch các hoạt động SQA và theo dõi tiến độ.
* **Kiểm tra yêu cầu:** Đảm bảo rằng các yêu cầu phần mềm đầy đủ, rõ ràng và nhất quán.
* **Thiết kế phần mềm:** Đảm bảo rằng thiết kế phần mềm đáp ứng các yêu cầu và các tiêu chuẩn chất lượng.
* **Kiểm thử phần mềm:** Thực hiện các bài kiểm tra để xác định và sửa lỗi phần mềm.
* **Quản lý cấu hình:** Theo dõi và kiểm soát các thay đổi đối với phần mềm.
* **Đảm bảo chất lượng quy trình:** Đảm bảo rằng các quy trình phát triển phần mềm được tuân thủ và hiệu quả.

1. **Giải thích sự khác nhau giữa failure, error, và fault.**

* **Lỗi phần mềm(Software Error)** là các phần code sai do lỗi 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:doerror 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)** Faultsẽ 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

1. **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.

1. **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.

– Tiêu chí vận hành sản phẩm

+ 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

6.Review-Inspection-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).

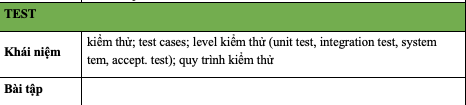
+ Mọi người sẽ đặt Pregunta hoặc cho ý kiến bổ sung về một số lĩnh vực để bảo đảm chất lượng kỹ thuật của tài liệu hoặc sản phẩm.

+ 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.

• **Inspection:** Kỹ thuật đánh giá chính thức. Tài liệu, sản phẩm... được những người không phải là tác giả hoặc trực tiếp liên quan(Người 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.



**1.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

**2.Level kiểm thử(unit test,integration test,system,accept,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 Acceptance 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!

**3.Quy trình kiểm thử**

• **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.

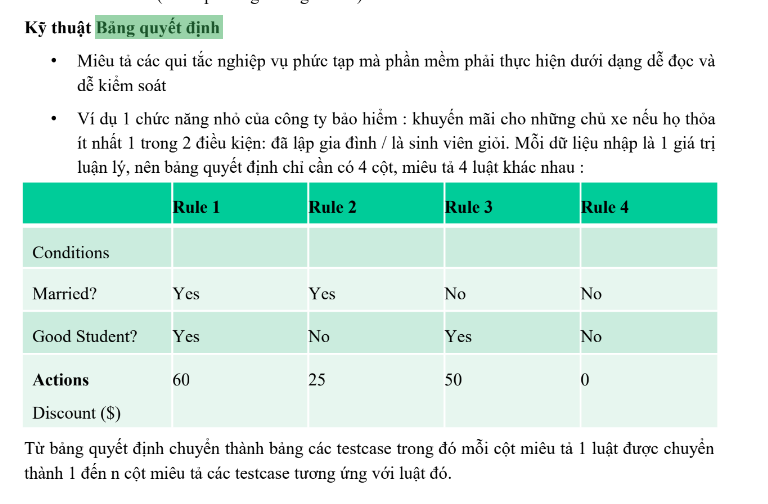
• Kiểm thử phát triển:Các quy trình kiểm thử, các kịch bản, Test Case, các dữ liệu được sử dụng trong kiểm thử phần mềm.

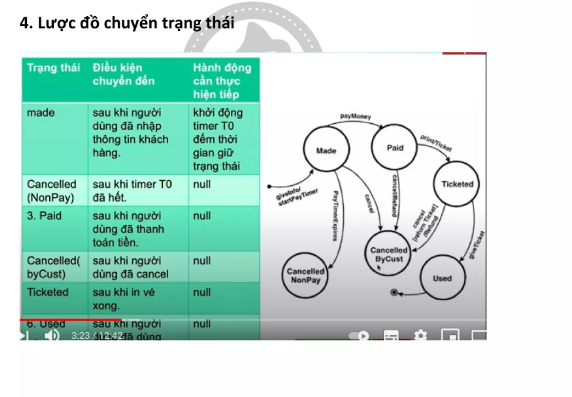
• **Kiểm thử thực hiện**:Dựa trên các kế hoạch, các văn bản kiểm thử và các báo cáo bất kỳ lỗi nào tìm thấy cho nhóm phát triển.

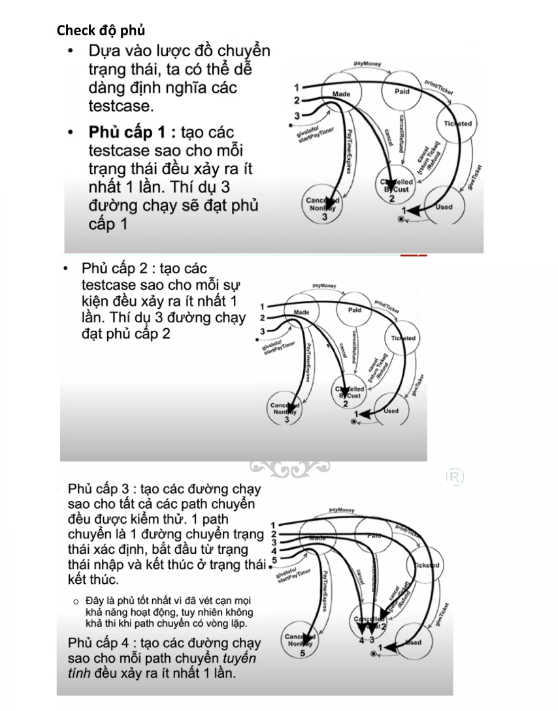
• **Kiểm thử báo cáo**:Sau khi hoàn tất kiểm thử, các Tester tạo ra các số liệu và báo cáo cuối cùng về nỗ lực kiểm thử của họ và có sẵn sàng phát hành phần mềm hay không.

•  **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.

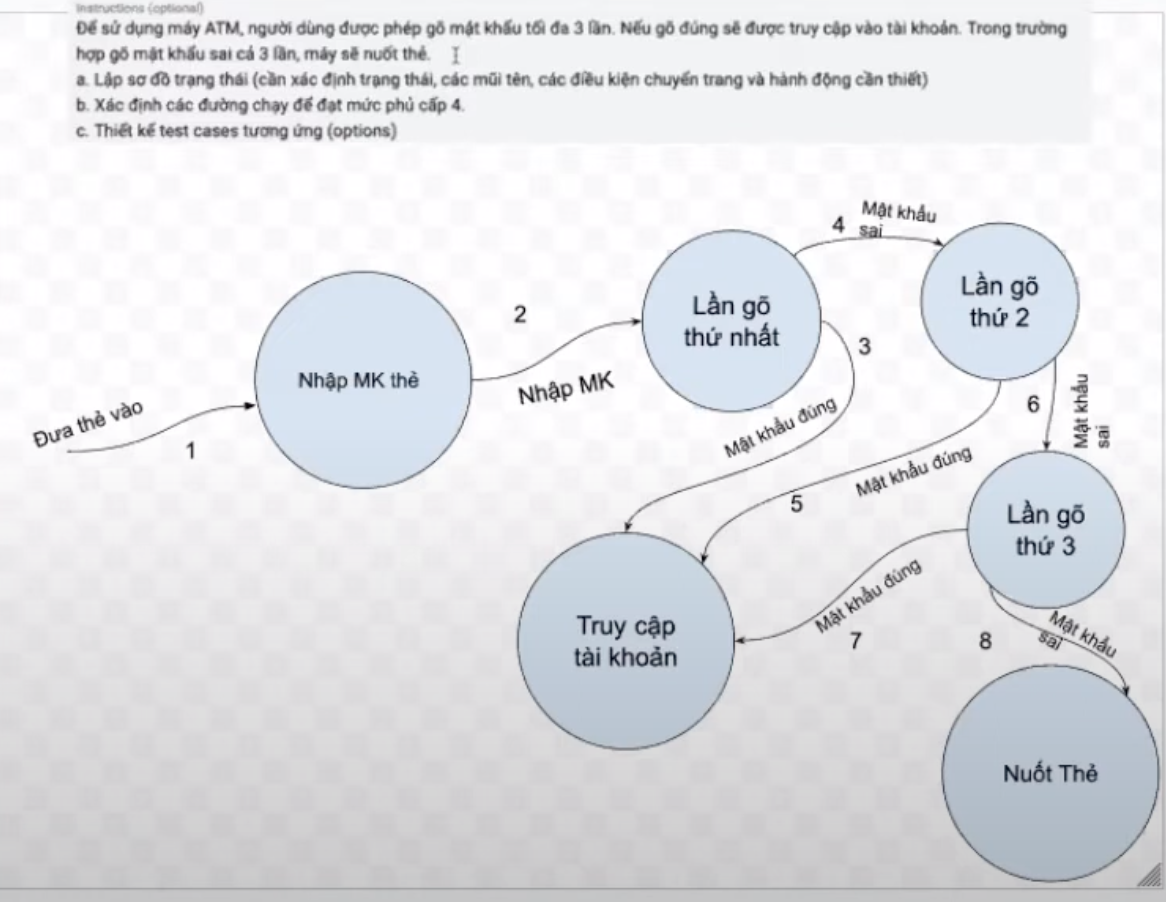
• **Kiểm thử hồi quy**:Người ta thường xây dựng một chương trình kiểm thử nhỏ là tập hợp của các bài kiểm tra cho mỗi tích hợp mới, sửa chữa hoặc cố định phần mềm, để đảm bảo rằng những cung cấp mới nhất đã không phá hủy bất cứ điều gì và toàn bộ phần mềm vẫn còn hoạt động một cách chính xác.

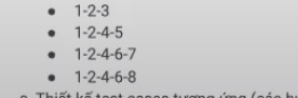


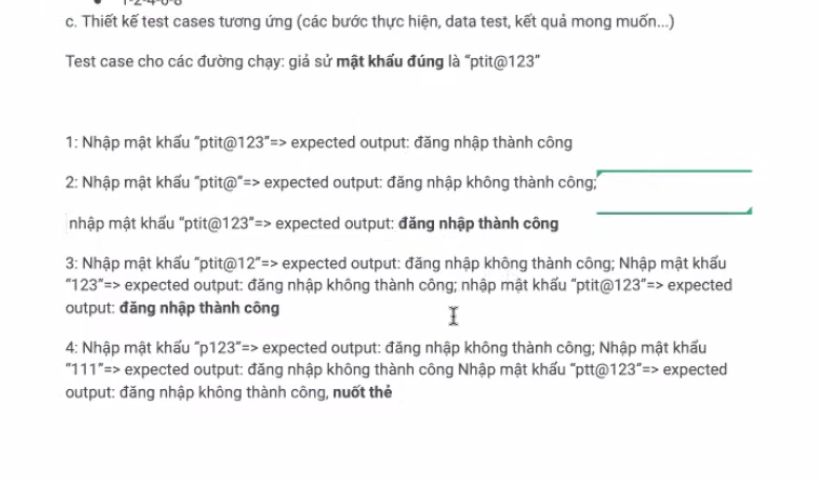




Ví dụ bt mẫu:







check list code : 183

check list GUI/web (giao diện) chức năng luồng, nghiệp vụ khác: 185

| Ma | Cau hoi | Y/N/NA | ghi chu |
| --- | --- | --- | --- |
| 1 | Đã có coding guide chưa? |  |  |
|  | Code có thể tự tài liệu không? (inline document) |  |  |
|  | Các quy ước đặt tên có phù hợp với kế hoạch quản lý cấu hình  không? |  |  |
|  | Code đã được định dạng đúng chưa? |  |  |
|  | Có một tập hợp các hàm chung được viết mà để tránh viết  trùng hàm? |  |  |
|  | Có bất kỳ code dư thừa hoặc rác? |  |  |
|  | Có bất kỳ lable nào không được tham chiếu? |  |  |
|  | Con trỏ đã được đặt thành NULL nếu cần thiết? |  |  |
|  | Con trỏ không trỏ tới vùng ngoài phạm vi bộ nhớ? |  |  |
|  | Có phải tất cả các chỉ số mảng nằm trong giới hạn? |  |  |
|  | Có phải tất cả các chỉ số mảng được khởi tạo chính xác? |  |  |
|  | Có phải tất cả các điều kiện rẽ nhánh chính xác? |  |  |
|  | Tất cả các vòng lặp có thể kết thúc? |  |  |
|  | Điều kiện để chấm dứt một vòng lặp có thực tế? |  |  |
|  | Không có mẫu số bằng 0 trong phép chia? |  |  |
|  | Có bất kỳ câu lệnh nào được đặt bên trong một vòng lặp có  thể được đặt bên ngoài? |  |  |
|  | Có phần nào trong code mà luồng thực hiện không bao giờ  chạy tới? |  |  |
|  | Có nhiều hơn 3 câu lệnh if lồng nhau? |  |  |
|  | Các tham số khai báo và khi cài đặt của interface có giống  nhau? |  |  |

|  | 20 Có biến nào khai báo mà không được sử dụng?  21 Bộ nhớ được khởi tạo chính xác?  22 Bộ nhớ động đã được cấp phát được giải phóng khi kết thúc?  23 Các truy vấn trên table có thực thi bằng việc sử dụng các  index ?  24 Trạng thái lỗi được kiểm tra sau mỗi câu truy vấn SQL?  25 Khóa được thực hiện trước khi update khi cần thiết? |  |  |
| --- | --- | --- | --- |
|  | 26 Các điều kiện sau đây đã được kiểm tra trong biểu thức:  a. Làm tròn số?  b. Có phép chia cho 0?  27 Các yêu cầu về thời gian phản hồi sẽ được đáp ứng?  28 Có một sự thay thế tốt hơn để cải thiện thời gian phản hồi?  29 Có các kiểm tra sau đây đã được thực hiện:  a. Kiểm tra bảng và file trống?  b. Kiểm tra lỗi IO?  30 Các thông báo lỗi có rõ ràng không? Các thông báo lỗi có đầy  đủ không?  31 Tất cả các điều kiện lỗi được bắt và xử lý?  32 Trong các biểu thức số học, có vấn đề sau đã được xác định:  a. Thứ tự xử lý không rõ ràng?  b. Có cần cuộn ngang để đọc toàn bộ biểu thức không?  c. Tả các dấu ngoặc đơn đã đóng đúng? Có đảm bảo đúng  trình tự xử lý không?  d. Làm tròn số được thực hiện cùng với các biểu thức?  e. Phép chia thực hiện cùng với biểu thức khác?  f. Có các biểu thức sử dụng trực tiếp các trường trong bảng ở  trong biểu thức?  33 Trong biểu thức quan hệ, các vấn đề sau cần được xác định:  a. So sánh cùng kiểu dữ liệu?  b. Có nhiều hơn một kết quả cho một biểu thức?  c. Biểu thức có phục vụ cho mục đích mà nó được sử dụng?  d. Có cần thanh cuộn ngang để đọc toàn bộ biểu thức?  34 Trong biểu thức logic, các vấn đề sau cần được xác định:  a. Biểu thức logic có phục vụ cho mục đích mà nó được sử  dụng?  b. Mỗi biểu thức quan hệ sử dụng kết quả là true hoặc false?  c. Mỗi biểu thức quan hệ được đặt bên trong ngoặc đơn?  d. Tại bất kỳ thời điểm nào chỉ có hai biểu thức quan hệ được  so sánh?  e. Có cần thanh cuộn ngang để xem được toàn bộ biểu thức?  35 Trong các hàm sử dụng file và table, các vấn đề sau cần được  xác định:  a. Có table hay file nào được mở sớm hơn so với yêu cầu?  b. Có bảng hay file nào vẫn mở khi kết thúc hàm?  36 Trong khai báo biến, các vấn đề sau cần được xác định:  a. Tất các các biến được khai báo là toàn cục hoặc tĩnh có  thực sự cần phải là toàn cục hoặc tĩnh không?  b. Có biến nào không cần thiết được khai báo không?  c. Có tên biến nào bị xung đột với từ khóa của ngôn ngữ lập |  |  |
|  | trình được sử dụng không?  d. Có bất kì hard coding nào trong code không? |  |  |

| ma | cau hoi | y/n/na | ghi chu |
| --- | --- | --- | --- |
| Chung | Giao diện người dùng có đúng với prototype không?  Danh sách menu và màn hình có thống nhất không?  Các link html có hoạt động không?  Tất cả các cửa sổ đều có thể truy cập từ toolbar?  Tất cả các trang web/cửa sổ đều có thể truy cập từ menu?  Các màn hình được gọi từ button có hiển thị đúng không?  Các label, textbox, combobox, etc có đúng font chữ, cỡ chữ, màu chữ  theo yêu cầu không?  Căn lề, độ rộng, khoảng cách có đồng bộ và đúng theo yêu cầu không?  Dữ liệu kiểu chữ căn lề trái?  Dữ liệu kiểu số căn lề phải?  Các form có được bố trí hợp lý và dễ sử dụng không?  Nếu các phím tắt được sử dụng, các phím tắt được gán có hoạt động  đúng không?  Màn hình KHÔNG có lỗi chính tả, cấu trúc câu, ngữ pháp?  Nếu các chữ viết tắt được sử dụng, nó có thống nhất trong tất cả các  giao diện không? Người dùng có hiểu được không?  Định dạng số, ngày tháng, thời gian có nhất quán không?  Có hiển thị mô tả khi di chuyển chuột vào tooltip không?  Nếu có tooltip, nó có ý nghĩa hoặc có hữu ích không?  Có một button hoặc check box được xét mặc định chưa?  Con trỏ di chuyển lần lượt theo thứ tự: từ trái qua phải, từ trên xuống  dưới khi nhấn Tab liên tục?  Con trỏ di chuyển lần lượt theo thứ tự: từ dưới lên trên, từ phải qua trái  khi nhấn Shift-Tab ?  Nếu chuột không focus vào button nào thì đã thực hiện chức năng của  button chính khi nhấn Enter chưa?  Nếu đang focus vào 1 button thì đã thực hiện chức năng của button khi  nhấn Enter chưa?  Màn hình thu nhỏ, phóng to tương ứng và không bị vỡ giao diện khi  nhấn Ctrl - và Ctrl + không ?  Các trường bắt buộc có dấu \* chưa?  Thông tin bản ghi hiển thị trên giao diện có đúng với thông tin bản ghi  được lưu trong Database không?  Kiểm tra thay đổi ngôn ngữ, ngôn ngữ thay đổi có đúng nội dung  không?  Mã captcha có hoạt động hay không?  Các trường KHÔNG cho phép nhập dữ liệu đã cùng màu theo đúng yêu  cầu chưa? |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**b. Phân tích lớp tương đương**

**- Valid:**

**+ Không kỳ hạn : 0 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 4 tháng : 4 tháng <= Thời gian gửi**

**+ Kỳ hạn 5 tháng : 5 tháng <= Thời gian gửi**

**+ Kỳ hạn 6 tháng : 6 tháng <= Thời gian gửi**

**- Invalid:**

**+ Không kỳ hạn : 0 ngày > Thời gian gửi**

**+ Kỳ hạn 1 tháng : 1 tháng > Thời gian gửi**

**+ Kỳ hạn 2 tháng : 2 tháng > Thời gian gửi**

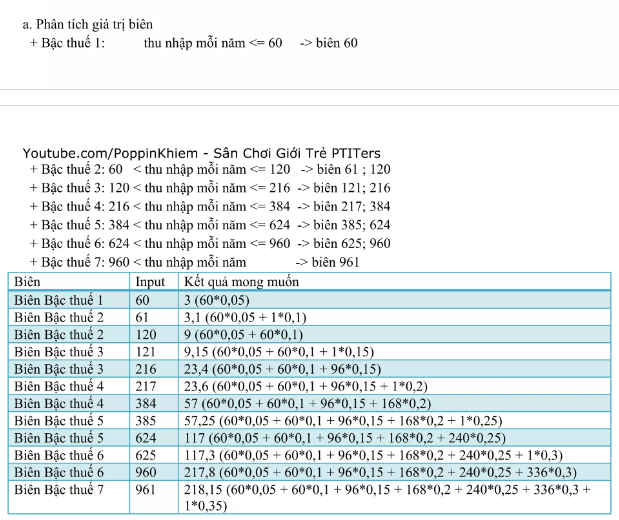
**+ Kỳ hạn 3 tháng : 3 tháng > Thời gian gửi**

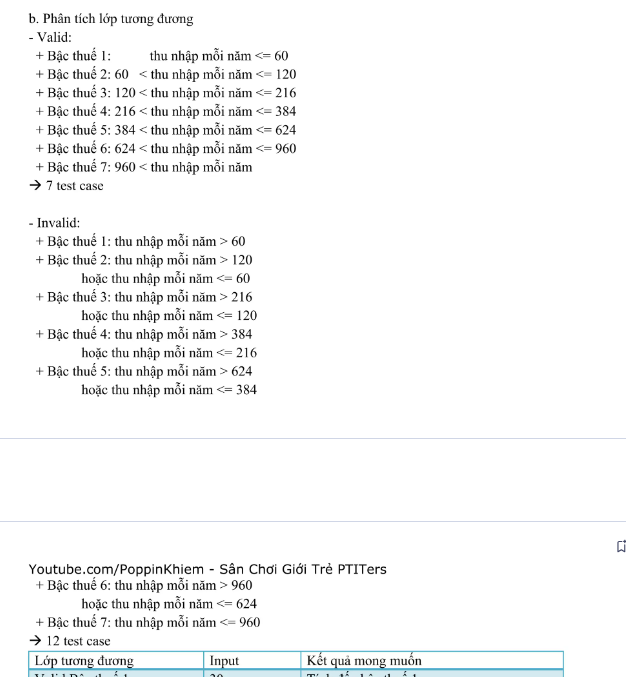
**+ Kỳ hạn 4 tháng : 4 tháng > Thời gian gửi**

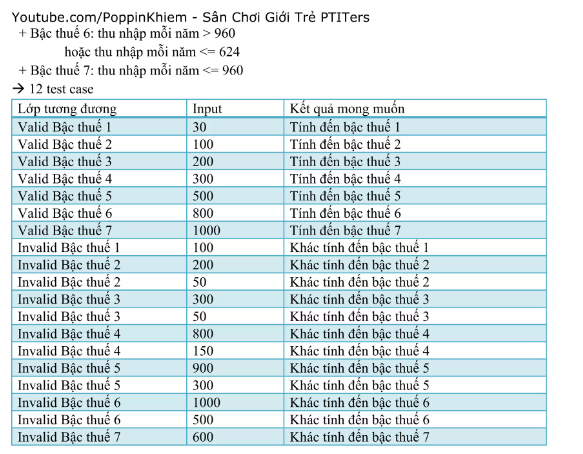
**+ Kỳ hạn 5 tháng : 5 tháng > Thời gian gửi**

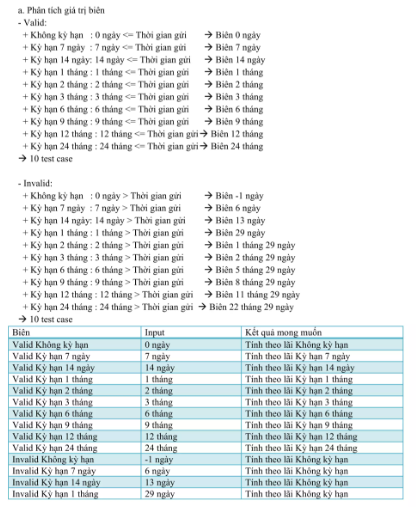
**+ Kỳ hạn 6 tháng : 6 tháng > Thời gian gửi**

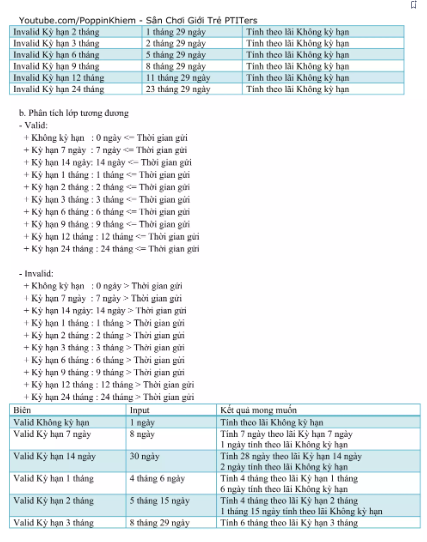
| Bien | Input | Expected output |
| --- | --- | --- |
| Valid Khong ky han | 1 | tinh theo lai ko ky han |
| Valid 1 thang | 4 thang 6 ngay | tinh 4 thang theo lai ky han 1 thang 6 ngay theo lai khong ky han |
| Valid 2 thang | 5 thang 15 ngay | tinh 4 thang theo lai ky han 2 thang 1 thang 15 ngay theo lai khong ky han |
| Valid 3 thang | 8 thang 29 ngay | tinh 6 thang theo ky han 3 thang  2 thang 29 ngay theo lai khong ky han |
| Valid 4 thang | 13 thang 4 ngay | 12 thang tinh theo ky han 4 thang  1 thang 4 ngay theo lai khong ky han |
| Valid 5 thang | 18 thang 5 ngay | 15 thang tinh theo ky han 5 thang  3 thang 5 ngay tinh theo lai khong ky han |
| Valid 6 thang | 38 thang 6 ngay | 36 thang tinh theo ky han 6 thang  2 thang 6 ngay tinh theo lai khong ky han |
| Invalid khong ky han | -1 ngay | tinh theo lai khong ky han |
| Invalid ky han 1 thang | 25 ngay | tinh theo lai khong ky han |
| Invalid ky han 2 thang | 1 thang 3 ngay | tinh theo lai khong ky han |
| Invalid ky han 3 thang | 2 thang 29 ngay | tinh theo lai khong ky han |
| Invalid ky han 4 thang | 3 thang 8 ngay | tinh theo lai khong ky han |
| Invalid ky han 5 thang | 4 thang 5 ngay | tinh theo lai khong ky han |
| Invalid ky han 6 thang | 5 thang 8 ngay | tinh theo lai khong ky han |

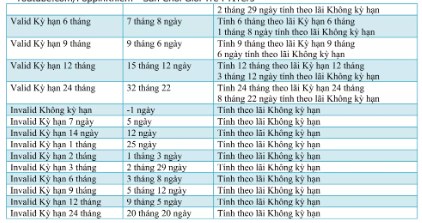










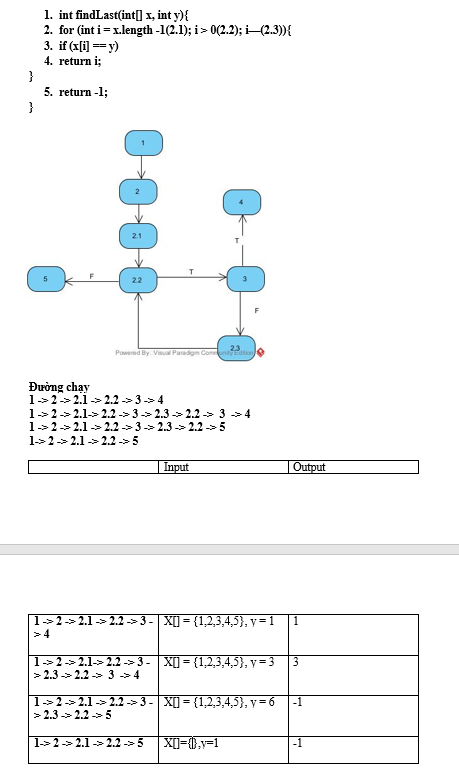


Xay dung dac ta theo MCcall

**RECORD OF CHANGE**

\*A: Thêm - M: Thay đổi - D: Xóa

| **Ngày hiệu lực** | **Các mục thay đổi** | **A\*, M, D** | **Chi tiết thay đổi** | **Phiên bản mới** |
| --- | --- | --- | --- | --- |
| 1/3/2022 |  | A | Tạo mới tài liệu đặc tả | 1.0 |
| 14/3/2022 | Đặc tả chức năng | M | Thay đổi mô tả chức năng | 1.1 |
| 25/3/2022 | Đặc tả chức năng | M | Thay đổi đặc tả chức năng | 1.2 |
| 28/3/2022 | Nghiệp vụ tính thuế | M | Bổ sung tính thuế cho cá nhân không cư trú | 1.3 |
| 13/04/2022 | Thuật ngữ  Các ràng buộc  Nghiệp vụ tính thuế | M | Thêm các thuật ngữ  Thêm hệ quản trị cơ sở dữ liệu  Chỉnh sửa chi tiết nghiệp vụ | 1.4 |
| 19/04/2022 | Yêu cầu chức năng | M | Thêm ngoại lệ | 1.5 |



**Câu 3** (3 điểm):

Biểu giá nước cho hộ gia đình dùng sinh hoạt cho ở bảng sau (mức sử dụng nước là số nguyên).

*Đơn vị tính đồng/m3*

| **Số TT** | **Mức sử dụng nước sinh hoạt của hộ dân cư (m3/tháng/hộ gia đình)** | **Giá bán nước** |
| --- | --- | --- |
| 1 | 10m3 đầu tiên | 5 973 |
| 2 | Từ trên 10m3 đến 20m3 | 7 052 |
| 3 | Từ trên 20m3 đến 30m3 | 8 669 |
| 4 | Trên 30m3 | 15 929 |

Ngoài ra, khách hàng cần phải thanh toán 5% thuế suất GTGT, và 10% phí bảo vệ môi trường với nước thải sinh hoạt trên giá bán nước.

Hệ thống quản lý tiền nước của thành phố cần xuất hoá đơn cho khách hàng với thông tin về số nước và chi tiết tiền như bên dưới (trường hợp số lượng tiêu thụ >10m3 thì bổ xung thêm dòng chi tiết định mức tiêu thụ, đơn giá, thành tiền tương ứng):

| **Số đọc tháng này** | **Số đọc tháng trước** | **Số lượng tiêu thụ (m3)** | **Định mức tiêu thụ** | **Đơn giá (đ)** | **Thành tiền (đ)** |
| --- | --- | --- | --- | --- | --- |
| 452 | 445 | 17 |  |  |  |
|  |  |  | 10 | 5 973 | 59730 |
| CỘNG TIỀN HÀNG | | | | | 41 811 |
| Thuế suất GTGT 5% | | | | | 2 091 |
| Phí bảo vệ môi trường với NTSH 10% | | | | | 4 181 |
| Tổng cộng tiền thanh toán | | | | | 48 083 |

***Thực hiện thiết kế test case theo phương pháp Test biên và Test phân lớp tương đương.***

.

Phân tích giá trị biên:

+ bậc 1 : mức sử dụng nước tiêu thụ đầu tiên <= 10m3 -> Biên 0,10

+ bậc 2 : mức sử dụng nước tiêu thụ thứ hai từ 10 < mức tiêu thụ <= 20 -> biên 11,20

+ bậc 3: mức sử dụng nước tiêu thụ thứ ba từ 20 < mức tiêu thụ <= 30 -> biên 21, 30

+ bậc 4: mức sử dụng nước tiêu thụ thứ từ mức tiêu thụ > 30 -> biên 31

| Biên | Input | Expected output |
| --- | --- | --- |
| Biên bậc 1.1 | 0 | Tổng phí = 0 |
| Biên bậc 1.2 | 10 | Tổng mức tiêu thụ = 10 \* 5973 = 59730  Định mức tiêu thụ = 10, đơn giá= 5973, Thành tiền = 59730  Tổng phí = 59730 + 59730\*10% + 59730 \* 5%= 68690 |
| Biên bậc 2.1 | 11 | Tổng mức tiêu thụ = 10 \* 5973 + (11-10) \* 7052 = 66782  Định mức tiêu thụ1 = 10, đơn giá= 5973, Thành tiền = 59730  Định mức tiêu thụ2 = 11-10, đơn giá =7052, Thành tiền = 7052  Tổng phí = 66782 + 3340 + 6679 = 76801 |
| Biên bậc 2.2 | 20 | Tổng mức tiêu thụ = 10 \* 5973 + (20-10) \* 7052 = 130250  Định mức tiêu thụ1 = 10, đơn giá = 5973, thành tiền= 59730  Định mức tiêu thụ 2 = 20-10= 10, đơn giá = 7052, Thành tiền = 70520  Tổng phí = 130250 + 6513 + 13025 = 149788 |
| Biên bậc 3.1 | 21 | Tổng mức tiêu thụ = 10 \* 5973 + 10\* 7052 + 1\* 8669 = 138919  Định mức tiêu thụ1 = 10, đơn giá = 5973, thành tiền= 59730  Định mức tiêu thụ 2 = 10= 10, đơn giá = 7052, Thành tiền = 70520  Định mức tiê thụ 3 = 1, đơn giá = 8669, thành tiền = 8669  Tổng phí = 138919 + 6946 + 13892 = 159757 |
| Biên bậc 3.2 | 30 | Tổng mức tiêu thụ = 10 \* 5973 + 10\* 7052 + 1\* 8669 = 138919  Định mức tiêu thụ1 = 10, đơn giá = 5973, thành tiền= 59730  Định mức tiêu thụ 2 = 10= 10, đơn giá = 7052, Thành tiền = 70520  Định mức tiê thụ 3 = 10, đơn giá = 8669, thành tiền = 86690  Tổng phí = 138919 + 6946 + 13892 = 159757 |
| Biên bậc 4 | 31 | Tự tính |

Phân vùng tương đương

Valid:

+ bậc 1 : mức sử dụng nước tiêu thụ đầu tiên 0<= mức <= 10m3 -> mức = 10, mức = 0

+ bậc 2 : mức sử dụng nước tiêu thụ thứ hai từ 10 < mức tiêu thụ <= 20 -> mức = 18

+ bậc 3: mức sử dụng nước tiêu thụ thứ ba từ 20 < mức tiêu thụ <= 30 -> mức = 25

+ bậc 4: mức sử dụng nước tiêu thụ thứ từ mức tiêu thụ > 30 -> mức 50

-> 4 testcase

Invalid:

+ mức tiêu thụ < 0 -> mức = -1

+ mức tiêu thụ không là số nguyên -> mức = 1.5

+ mức tiêu thụ input nhập vào không phải định dạng số -> mức abc

-> 3 testcase

| Vùng | Input | Expected output |
| --- | --- | --- |
| Invalid | | |
| Invalid 1 | -1 | Mức tiêu thụ phải >= 0 |
| Invalid 2 | 1.5 | Mức tiêu thụ phải là một số nguyên |
| Invalid 3 | Abc | Mức tiêu thụ phải là định dạng số nguyên |
| Valid | | |
| Valid bậc 1.1 | 0 | Tính mức tiêu thụ bậc 1 |
| Valid bậc 1.2 | 10 | Tính mức tiêu thụ bậc 1 |
| Làm theo giống với biên nhưng viết ngắn gọn thôi | | |

package org.example;

import java.util.Scanner;

public class Tinhtiennuoc {

public Tinhtiennuoc() {

}

public int Tinhtiennuoc1(int muctieuthu)

{

int tong = 0;

int muc1;

int muc2;

int muc3;

int muc4;

System.out.println("muc tieu thu:" + muctieuthu);

if(muctieuthu < 0)

{

System.out.println("so khong hop le");

return -1;

}

else{

if( muctieuthu <=10)

{

muc1 = muctieuthu;

System.out.println("muc1: " + muc1 + ", " +"don gia: " + 5973 +" "+ "tong gia muc1: " + (muc1 \* 5973));

tong = muc1 \* 5973;

}

else if(muctieuthu>10 && muctieuthu<=20){

muc1 = 10;

System.out.println("muc1: " + muc1 + ", " +"don gia: " + 5973 +", "+ "tong gia muc1: " + (10 \* 5973));

muc2 = muctieuthu - 10;

System.out.println("muc2: " + muc2 + ", " +"don gia: " + 7052 +", "+ "tong gia muc2: " + (muc2 \* 7052));

tong = muc1\*5973 + muc2\*7052;

}

else if(muctieuthu>20 && muctieuthu<=30)

{

muc1 = 10;

System.out.println("muc1: " + muc1 + ", " +"don gia: " + 5973 +", "+ "tong gia muc1: " + (muc1 \* 7052));

muc2 = 10;

System.out.println("muc2: " + muc2 + ", " +"don gia: " + 7052 +", "+ "tong gia muc2: " + (muc2 \* 7052));

muc3 = muctieuthu - 20;

System.out.println("muc3: " + muc3 + ", " +"don gia: " + 8669 +", "+ "tong gia muc3: " + (muc3 \* 8669));

tong = muc1\*5973 + muc2\*7052 + muc3\*8669;

}

else if(muctieuthu>30)

{

muc1 = 10;

System.out.println("muc1: " + muc1 + ", " +"don gia: " + 5973 +", "+ "tong gia muc1: " + (muc1 \* 7052));

muc2 = 10;

System.out.println("muc2: " + muc2 + ", " +"don gia: " + 7052 +", "+ "tong gia muc2: " + (muc2 \* 7052));

muc3 = 10;

System.out.println("muc3: " + muc3 + ", " +"don gia: " + 8669 +", "+ "tong gia muc3: " + (muc3 \* 8669));

muc4 = muctieuthu - 30;

System.out.println("muc4: " + muc4 + ", " +"don gia: " + 15929 +", "+ "tong gia muc4: " + (muc4 \* 15929));

tong = muc1\*5973 + muc2\*7052 + muc3\*8669 + muc4\*15929;

}

}

int thue = (int) Math.round(tong \* 0.05);

System.out.println("thue 5% : " + thue);

int phimt = (int) Math.round(tong \* 0.1);

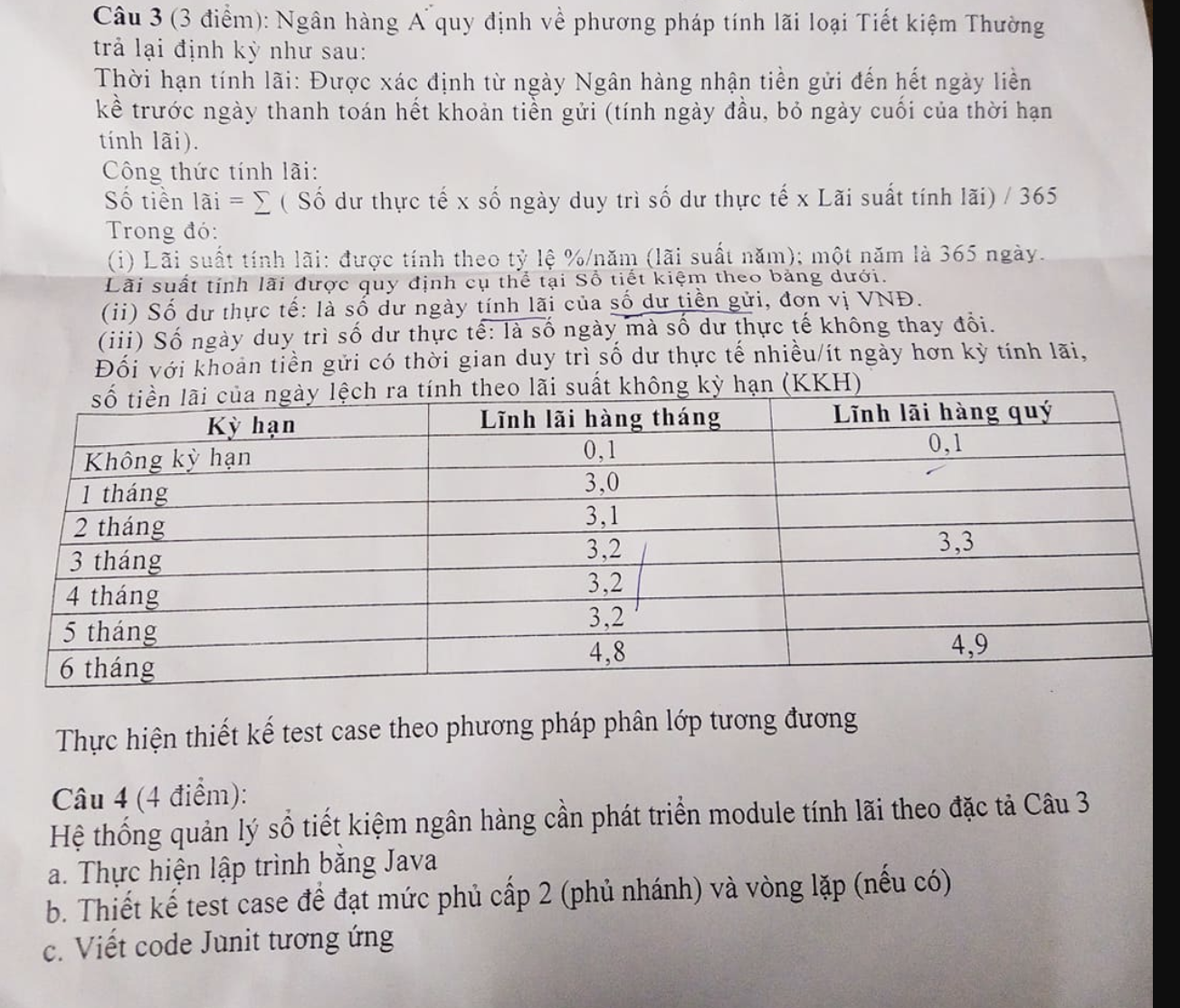
System.out.println("phi mt: "+ phimt);

int tongcong = tong + thue + phimt;

return tongcong;

}

}

package org.example;

import java.util.Scanner;

public class InterestCalculator {

private static int balance;

private static int tienLai;

private static int days;

private static boolean isQuarterly;

public InterestCalculator(int balance, int tienLai, int days, boolean isQuarterly) {

this.balance = balance;

this.tienLai = tienLai;

this.days = days;

this.isQuarterly = isQuarterly;

}

public InterestCalculator(){}

public double calculateInterest(double balance, int days, boolean isQuarterly, int tienLai) {

double rate = 0.0;

if (days < 30) {

tienLai = (int) Math.round(balance \* days \* 0.1)/365/100;

System.out.println(tienLai);

} else if (days >= 30 && days < 60) {

tienLai = (int) Math.round(balance ( 30 0.3 + (days - 30) \* 0.1 )/365/100);

} else if (days >= 60 && days < 90) {

tienLai = (int) Math.round(balance ( 60 0.31 + (days - 60) \* 0.1 )/365/100);

} else if (days >= 90 && days < 120) {

rate = isQuarterly ? 0.33 : 0.32;

tienLai = (int) Math.round(balance ( 90 rate + (days - 90) \* 0.1 )/365/100);

} else if (days >= 120 && days < 150) {

tienLai = (int) Math.round(balance ( 120 3.2 + (days - 120) \* 0.1 )/365/100);

} else if (days >= 150 && days < 180) {

tienLai = (int) Math.round(balance ( 150 3.2 + (days - 150) \* 0.1 )/365/100);

} else if (days >= 180) {

rate = isQuarterly ? 0.49 : 0.32;

tienLai = (int) Math.round(balance ( 180 rate + (days - 180) \* 0.1 )/365/100);

}

return tienLai;

}

}

[01:54]

Main

[01:54]

package org.example;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

InterestCalculator i = new InterestCalculator();

System.out.print("Nhập số dư thực tế (VNĐ): ");

double balance = scanner.nextDouble();

System.out.print("Nhập số ngày duy trì số dư thực tế: ");

int days = scanner.nextInt();

System.out.print("Lai theo thang: ");

int term = days/30;

System.out.print("Lãi hàng quý? (true/false): ");

boolean isQuarterly = scanner.nextBoolean();

double interest = i.calculateInterest(balance, days, isQuarterly, 0);

System.out.printf("Số tiền lãi: %.2f VNĐ\n", interest);

scanner.close();

}

}

[01:54]

Test

@Test

public void TinhLaiNh() {

InterestCalculator i = new InterestCalculator();

long output = (long) i.calculateInterest(1000000000, 20, false, 0);

long expected = 54794;

Assert.assertEquals(output, expected);

}

checklist ò Nam:

**Checklist for code review**

| **STT** | **Mục** | **Có** | **Không** | **Ghi chú** |
| --- | --- | --- | --- | --- |
| **1** | Hướng dẫn code cụ thể trong project plan đã được thêm vào chưa? |  | **x** | **Không có** |
| **2** | Code có được comment đầy đủ không ? |  | **x** | **Chưa đầy đủ** |
| **3** | Đặt tên có phù hợp với việc quản lý cấu hình không ? | **x** |  |  |
| **4** | Code đã được format tốt chưa | **x** |  |  |
| **5** | Các đoạn code có chung chức năng giữa các chương trình có bị lặp lại hay không ? |  | **x** |  |
| **6** | Có đoạn code nào dư thừa hoặc không cần thiết không ? |  | **x** |  |
| **7** | Có nhãn nào chưa được tham chiếu tới không ? |  | **x** |  |
| **8** | Các con trỏ có được set về NULL khi cần thiết không ? |  | **x** |  |
| **9** | Con trỏ số học có trỏ tới bộ nhớ nằm ngoài phạm vi hay không ? |  | **x** |  |
| **10** | Tất cả các chỉ số của mảng có nằm trong phạm vi không ? | **x** |  |  |
| **11** | Tất cả các chỉ số mảng có được khởi tạo đúng không? | **x** |  |  |
| **12** | Tất cả các điều kiện rẽ nhánh có đúng không? | **x** |  |  |
| **13** | Có phải tất cả các vòng lặp đều kết thúc không? | **x** |  |  |
| **14** | Điều kiện để kết thúc vòng lặp có thực tế không? | **x** |  |  |
| **15** | Mẫu số của các phép chia có được kiểm tra 0 trước khi thực hiện tính toán không? | **x** |  |  |
| **16** | Có thể đặt bất kỳ câu lệnh nào bên trong vòng lặp được đặt bên ngoài vòng lặp hay không ? |  | **x** |  |
| **17** | Có bất kỳ phần nào trong code mà luồng thực thi không chạy qua hay không |  | **x** |  |
| **18** | Các câu điều kiện if được lồng nhau hơn 3 lần hay không? |  | **x** |  |
| **19** | Các thông số và giao diện thực tế có khớp với nhau hay không ? | **x** |  |  |
| **20** | Có biến nào khai báo mà không dùng không |  | **x** |  |
| **21** | Bộ nhớ đã được khởi tạo chính xác chưa? | **x** |  |  |
| **22** | Phần bộ nhớ cấp phát động đã được giải phóng khi không còn dùng chưa | **x** |  |  |
| **23** | Các truy vấn tới các bảng có bắt buộc phải dùng index không (indice)? | **x** |  |  |
| **24** | Có kiểm tra trạng thái lỗi sau mỗi lệnh SQL không? | **x** |  |  |
| **25** | Hoạt động khóa có được thực hiện trước khi cập nhật lúc cần thiết không? |  | **x** |  |
| **26** | Đã kiểm tra các điều kiện trong biểu thức: |  |  |  |
| a. Làm tròn | **x** |  |  |
| b. Chia cho 0 | **x** |  |  |
| **27** | Liệu các yêu cầu về thời gian chạy chương trình có được đáp ứng không? | **x** |  |  |
| **28** | Có giải pháp thay thế nào tốt hơn để cải thiện thời gian phản hồi không? |  | **x** |  |
| **29** | Các điều sau đã được kiểm tra hay chưa? |  |  |  |
| a. Kiểm tra bảng và tệp có bị trống không? | **x** |  |  |
| b. Kiểm tra lỗi Input/Output? | **x** |  |  |
| **30** | Các thông báo lỗi có rõ ràng không? Các thông báo lỗi có đầy đủ không? | **x** |  |  |
| **31** | Các trường hợp có thể xảy ra lỗi đã có logic để kiểm tra và xử lý chưa? | **x** |  |  |
| **32** | Trong các biểu thức toán học: |  |  |  |
| a. Thứ tự ưu tiên thực hiện tính toán đã chính xác chưa? | **x** |  |  |
| b. Có cần cuộn ngang để đọc toàn bộ biểu thức không? |  | **x** |  |
| c. Tất cả các dấu ngoặc đơn đã được đóng đúng chưa? Chúng có đảm bảo trình tự thực hiện của tiến trình không? | **x** |  |  |
| d. Có làm tròn số trong biểu thức không? | **x** |  |  |
| e. Có phép chia nào được kết hợp với 1 biểu thức khác không? |  | **x** |  |
| f. Có biểu thức nào sử dụng trực tiếp trường bảng hoặc trường tệp không? |  | **x** |  |
| **33** | Trong biểu thức quan hệ, những điều sau đây đã được giải quyết? |  |  |  |
| a. Có phải so sánh giữa các loại dữ liệu giống nhau không? | **x** |  |  |
| b. Có thể có nhiều hơn hai kết quả cho bất kỳ biểu thức nào không? |  | **x** |  |
| c. Biểu thức có phục vụ cho mục đích mà nó được tạo ra hay không? | **x** |  |  |
| d. Có cần cuộn ngang để đọc toàn bộ biểu thức không? |  | **x** |  |
| **34** | Trong biểu thức logic, có những điều sau đây đã được giải quyết: |  |  |  |
| a. Biểu thức logic có phục vụ cho mục đích mà nó được tạo ra hay không? | **x** |  |  |
| b. Mỗi biểu thức quan hệ được sử dụng dẫn đến một kết quả đúng hay sai? | **x** |  |  |
| c. Mỗi biểu thức quan hệ có nằm trong một tập hợp các dấu ngoặc đơn không? | **x** |  |  |
| d.Tại bất kỳ thời điểm nào, chỉ có hai biểu thức quan hệ được so sánh với nhau? | **x** |  |  |
| e. Có cần cuộn ngang để đọc toàn bộ biểu thức không? |  | **x** |  |
| **35** | Các hoạt động trong tệp và bảng có những điều gì sau đây đã được giải quyết : |  |  |  |
| a. Có tệp hoặc bảng nào được mở sớm hơn nhiều so với yêu cầu không? |  | **x** |  |
| b. Có tệp hoặc bảng nào còn mở khi các thao tác hoàn tất không? |  | **x** |  |
| **36** | Trong khai báo biến, những điều sau đã được giải quyết chưa: |  |  |  |
| a. Tất cả các biến được khai báo là biến toàn cục hoặc biến tĩnh có thực sự cần phải là biến toàn cục hay biến tĩnh không? | **x** |  |  |
| b. Có bất kỳ khai báo nào của các biến không cần thiết không? |  | **x** |  |
| c. Có bất kỳ tên biến nào xung đột với từ khóa của ngôn ngữ lập trình đang được sử dụng hay không? |  | **x** |  |
| d. Có bất kỳ mã hóa cứng nào bên trong code không? |  | **x** |  |

**­**

**Checklist for GUI review**

| **Phương diện** | **Review** |
| --- | --- |
| Độ chính xác của chính tả | Tất cả các từ đều viết đúng chính tả |
| Định vị nhãn | 1. Tất cả các nhãn được đặt ở bên trái hoặc ở trên cùng của điều khiển.  2. Nếu nhãn nằm phía trên điều khiển, nó sẽ được căn thẳng với góc bên trái hoặc căn giữa.  3. Không có nhãn nào được căn bên phải của điều khiển hoặc nằm bên dưới điều khiển |
| Màu chữ | Màu đen nhạt. |
| Phông chữ | Phông chữ Roboto |
| Cỡ chữ | 13px |
| Bảng màu | Tất cả các màn hình sử dụng một bảng màu duy nhất |
| Đồ họa | Không có |
| Biểu tượng | Dùng font-awesome |
|  | Chương trình có thể chạy đa luồng khi lấy dữ liệu từ máy chủ |
| Thay đổi kích thước cửa sổ | Có thể phù hợp nhiều màn hình máy tính, tablet và điện thoại |
| Hiện và ẩn | Có các trường nhập id và tài khoản disable |
| Thông báo modal | Có thông báo modal khi đăng xuất |
| Thiết kế thông báo | Thiết kế lỗi hiển thị bên dưới component bị lỗi |
| Auto Focus | Khi mở form thêm mới thì tự động focus vào ô nhập liệu đầu tiên |
| Tiêu điểm con trỏ | Khi có lỗi xảy ra, con trỏ chuột sẽ tự động focus vào ô nhập liệu bị lỗi để người dùng có thể sửa luôn. |
| Thông báo | Thông báo khi xử lý thất bại hoặc thành công |
| Screen refresh | Sau khi thêm mới, sửa, xóa user thành công thì màn hình danh sách sẽ thực hiện reset lại, cập nhật thông tin mới nhất từ hệ thống. |
| Sự kiện click | Sự kiện click của các nút |
| Điều hướng | Sử dụng router để điều hướng các màn hình khác nhau |
| Mặc định | Các select box tự động chọn giá trị đầu tiên hoặc null nếu không có |
| Phản hồi hành động | Có các xác nhận khi người dùng sử dụng chức năng xóa dữ liệu |
| Tính nhất quán | Các thành phần button,input, label đều có tính nhất quán giữa các view |
| Thanh công cụ | Sử dụng menu đa dụng trái |
| Thứ tự tab | Tab từ trên xuống dưới,từ trái qua phải |

**Check list Web test**: kiểm thử phủ định (negative testing)

| **Khía cạnh** | **Mô tả test case** | **Review** |
| --- | --- | --- |
| Điều hướng | 1. Click vào tất cả các liên kết trên trang và đảm bảo rằng chúng điều hướng đến các trang Web chính xác. Cũng kiểm tra xem không có liên kết nào bị hỏng. | Kiểm tra tất cả các liên kết điều hướng |
| 2. Kiểm tra xem các liên kết dẫn đến các trang Web khác có mở trong cửa sổ khác không | Nhấp vào các liên kết trỏ đến các trang Web khác. |
| 3. Sử dụng nút Back sau khi đăng xuất khỏi một trang web sẽ không cho phép truy cập chức năng. Hệ thống sẽ phản hồi bằng một thông báo lỗi cho biết người dùng cần đăng nhập | Đăng xuất khỏi ứng dụng, sau đó nhấp vào nút Back. |
| Nhập dữ liệu sai | Khi nhập sai dữ liệu, một thông báo lỗi thích hợp sẽ được hiển thị ngay sau khi nhập hoặc sau khi nhấp vào nút Submit. | Nhập dữ liệu sai vào một số trường để xem liệu nó có bị từ chối hay không |

**ÔN TẬP**

**Câu 1** (1 điểm): Đảm bảo chất lượng phần mềm là gì? Thường gồm những hoạt động nào?

Trả lời:

là một quy trình hệ thống được sử dụng để đảm bảo rằng phần mềm phát triển đáp ứng các yêu cầu về chất lượng và tuân thủ các tiêu chuẩn, quy định và quy trình đề ra. Mục tiêu chính của SQA là cung cấp sản phẩm phần mềm chất lượng cao, giảm thiểu lỗi và nâng cao sự hài lòng của khách hàng.

**Các hoạt động thường gặp trong Đảm bảo chất lượng phần mềm:**

1. **Lập kế hoạch chất lượng (Quality Planning)**:

· Xác định các tiêu chuẩn và quy trình cần tuân theo.

· Thiết lập các mục tiêu chất lượng cụ thể cho dự án phần mềm.

2. **Quản lý chất lượng (Quality Management)**:

· Đảm bảo rằng các hoạt động và kết quả của dự án tuân thủ các tiêu chuẩn và quy trình đã được xác định.

· Theo dõi và đánh giá tiến độ thực hiện các hoạt động chất lượng.

3. **Kiểm thử phần mềm (Software Testing)**:

· Thực hiện các loại kiểm thử như kiểm thử đơn vị (unit testing), kiểm thử tích hợp (integration testing), kiểm thử hệ thống (system testing) và kiểm thử chấp nhận (acceptance testing).

· Xác định và sửa lỗi trong phần mềm.

4. **Đánh giá chất lượng (Quality Assessment)**:

· Sử dụng các công cụ và kỹ thuật để đo lường và đánh giá chất lượng phần mềm.

· Tiến hành các đánh giá chính thức (formal review) như kiểm tra mã nguồn (code review) và kiểm tra thiết kế (design review).

5. **Kiểm tra và đảm bảo quy trình (Process Assurance)**:

· Kiểm tra các quy trình phát triển phần mềm để đảm bảo chúng được tuân thủ và hiệu quả.

· Cải tiến liên tục các quy trình dựa trên phản hồi và kết quả thực tế.

6. **Quản lý cấu hình (Configuration Management)**:

· Quản lý các thay đổi và phiên bản của phần mềm để đảm bảo tính nhất quán và tính toàn vẹn của sản phẩm.

7. **Đánh giá rủi ro (Risk Management)**:

· Xác định, đánh giá và quản lý các rủi ro có thể ảnh hưởng đến chất lượng phần mềm.

· Lên kế hoạch và thực hiện các biện pháp giảm thiểu rủi ro.

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>