**TRƯỜNG ĐẠI HỌC MỞ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\_\_\_\_\_\_\_\_\_\_\_ 🙞🕮🙜 \_\_\_\_\_\_\_\_\_\_\_**

A blue book with a ring around it

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN**

**NHÓM 18**

**KIỂM THỬ PHẦN MỀM**

**KHOA ĐÀO TẠO ĐẶC BIỆT**

Hồ Chí Minh -\_\_/\_\_/\_\_

**TRƯỜNG ĐẠI HỌC MỞ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\_\_\_\_\_\_\_\_\_\_\_ 🙞🕮🙜 \_\_\_\_\_\_\_\_\_\_\_**

A blue book with a ring around it

Description automatically generated

***ĐỀ TÀI***

**KIỂM THỬ PHẦN MỀM**

**THI TRẮC NGHIỆM TIN HỌC**

|  |  |  |  |
| --- | --- | --- | --- |
| Giáo viên hướng dẫn | | : | *Phan Trần Minh Khuê* |
| Nhóm số | | : | *18* |
| Mã lớp | | : | *DH2101C* |
|  |  | | |

Hồ Chí Minh -\_\_/\_\_/\_\_

**MỤC LỤC**

[CHƯƠNG 1: TỔNG QUAN KIỂM THỬ PHẦN MỀM 4](#_Toc70845938)

[1. Giới thiệu chung về kiểm thử phần mềm 4](#_Toc70845939)

[1.1. Khái niệm kiểm thử phần mềm 4](#_Toc70845940)

[1.2. Mục đích của kiểm thử phần mềm 4](#_Toc70845941)

[2. Các chiến lược kiểm thử phần mềm 5](#_Toc70845942)

[2.1. Quy trình kiểm thử phần mềm 5](#_Toc70845943)

[2.2. Các cấp độ kiểm thử phần mềm 7](#_Toc70845944)

[2.2.1. Kiểm thử đơn vị. 7](#_Toc70845945)

[2.2.2. Kiểm thử tích hợp - Integration Test 9](#_Toc70845946)

[2.2.3. Kiểm thử hệ thống – System Test 10](#_Toc70845947)

[2.3. Các loại hình kiểm thử phần mềm 14](#_Toc70845948)

[3. Các phương pháp kiểm thử phần mềm 15](#_Toc70845949)

[3.1. Kiểm thử hộp đen 15](#_Toc70845950)

[3.2. Kiểm thử hộp trắng 16](#_Toc70845951)

[3.3. Kiểm thử hộp xám 17](#_Toc70845952)

[4. Kiểm thử phần mềm tự động 18](#_Toc70845953)

[4.1. Tổng quan kiểm thử phần mềm tự động 18](#_Toc70845954)

[4.2. Quy trình của kiểm thử phần mềm tự động 19](#_Toc70845955)

[4.3. Các ưu, nhược điểm của kiểm thử phần mềm tự động 19](#_Toc70845956)

[4.4. Một số công cụ kiểm thử phần mềm tự động 20](#_Toc70845957)

[CHƯƠNG 2: THIẾT KẾ TEST CASE 22](#_Toc70845958)

[1. Thiết kế Test Case là gì ? 22](#_Toc70845959)

[2. Các kỹ thuật thiết kế Test case 22](#_Toc70845960)

[2.1. Kiểm thử hộp trắng – kiểm thử bao phủ logic 23](#_Toc70845961)

[2.1.1. Kỹ thuật bao phủ câu lệnh - Statement Coverage 23](#_Toc70845962)

[2.1.2. Kỹ thuật bao phủ quyết định – Decision Coverage 25](#_Toc70845963)

[1.1.3. Kỹ thuật bao phủ điều kiện – Condition Coverage. 26](#_Toc70845964)

[1.1.4. Kỹ thuật bao phủ quyết định/ điều kiện – Decision/ Condition Coverage 28](#_Toc70845965)

[1.1.5. Kỹ thuật bao phủ đa điều kiện – Multiple condition Coverage 29](#_Toc70845966)

[1.2. Kiểm thử hộp đen 31](#_Toc70845967)

[2.2.1. Kỹ thuật phân lớp tương đương 31](#_Toc70845968)

[2.2.2. Kỹ thuật phân tích giá trị biên 33](#_Toc70845969)

[2.2.3. Kỹ thuật đồ thị nguyên nhân – kết quả 35](#_Toc70845970)

[2.2.4. Đoán lỗi – Error Guessing 39](#_Toc70845971)

[CHƯƠNG 3: KIỂM THỬ PHẦN MỀM ATM SIMULATOR 41](#_Toc70845972)

[1. Giới thiệu phần mềm 41](#_Toc70845973)

[2. Phân tích thiết kế hệ thống 41](#_Toc70845974)

[2.1.1. Tổng quan 41](#_Toc70845975)

[2.1.2. Mô tả - Use case description 41](#_Toc70845976)

[2.1.3. Use case & Actor mapping 42](#_Toc70845977)

[2.1.4. Sơ đồ thực thể liên kết – Entity Relationship Diagram (ERD) 43](#_Toc70845978)

[2.2. Chi tiết về thiết kế chức năng – Details function design 44](#_Toc70845979)

[2.2.1. Use case 01 – Validation 44](#_Toc70845980)

[2.2.2. `Use case 02: Rút tiền – Withdraw Money 46](#_Toc70845981)

[2.2.3. Use case 03: Kiểm tra số dư - Check Balance 47](#_Toc70845982)

[2.2.4. Use case 04: Xem lịch sử giao dịch – View History 48](#_Toc70845983)

[2.2.5. Use case 05: Chuyển tiền – Cash Transfer 49](#_Toc70845984)

[2.2.6. Use case 06: Thay đổi mã PIN – Change PIN 50](#_Toc70845985)

[2.2.7. Use case 07: Ghi nhật ký – Logging 52](#_Toc70845986)

[3. Lập kế hoạch kiểm thử 53](#_Toc70845987)

[3.1. Mục đích 53](#_Toc70845988)

[3.2. Thông tin chung 53](#_Toc70845989)

[3.3. Phạm vi test 53](#_Toc70845990)

[3.4. Các công cụ hỗ trợ 54](#_Toc70845991)

[3.5. Tài liệu tham khảo 54](#_Toc70845992)

[4. Thực hiện kiểm thử 55](#_Toc70845993)

[4.1. Kiểm thử Use case 01: Validation 55](#_Toc70845994)

[4.2. Kiểm thử Use case 02: Rút tiền – Withdraw Money 58](#_Toc70845995)

[4.3. Kiểm thử Use case 03: Kiểm tra số dư – Check Balance 65](#_Toc70845996)

[4.4. Kiểm thử Use case 04: Xem lịch sử giao dịch – View History 66](#_Toc70845997)

[4.5. Kiểm thử Use case 05: Chuyển tiền – Cast Transfer 69](#_Toc70845998)

[4.6. Kiểm thử Use case 06: Thay đổi mã PIN – Change PIN 78](#_Toc70845999)

[4.7. Kiểm thử Use case 07: Ghi nhật ký – Logging 82](#_Toc70846000)

# **CHƯƠNG 1: TỔNG QUAN KIỂM THỬ PHẦN MỀM**

1. **Giới thiệu chung về kiểm thử phần mềm**
   1. **Khái niệm kiểm thử phần mềm**

Theo IEEE, phần mềm bao gồm các chương trình máy tính, các thủ tục, các tài liệu và dữ liệu có thể liên quan đến hoạt động của hệ thống máy tính. Kiểm thử phần mềm là quá trình vận hành hệ thống hoặc thành phần dưới những điều kiện xác định, quan sát hoặc ghi nhận kết quả và đưa ra đánh giá về hệ thống hoặc thành phần đó.

Theo Glenford Myers, kiểm thử phần mềm là quá trình vận hành chương trình để tìm ra lỗi.

* 1. **Mục đích của kiểm thử phần mềm**

Các lỗi chung của phần mềm :

* Lỗi chức năng: phần mềm có lỗi chức năng nếu cái được bạn hi vong lại không làm được.
* Lỗi giao tiếp: lỗi này xảy ra trong giao tiếp từ phần mềm đến người cuối cùng .
* Lỗi thiếu lệnh: lỗi này xảy ra khi lệnh mong muốn đang thiếu
* Lỗi cú pháp: là từ bị sai chính tả hoặc không đugs cấu trúc câu lệnh
* Lỗi xử lý lỗi: bất cứ khi nào có lỗi khi người dùng đang tương tác với phần mềm đều cần được xử lý.
* Lỗi tính toán: logic kém , sai công thức , lỗi coding, lỗi gọi hàm..
* Lỗi luồng điều kiện: luồng điều khiển của một phần mềm mô tả sẽ làm gì tiếp theo và trong điều kiện nào.

Mục đích của kiểm thử phần mềm:

* Tìm ra được càng nhiều lỗi càng tốt trong điều kiện về thời gian đã định cũng như là nguồn lực sẵn có.
* Chứng minh rằng sản phẩm phần mềm phù hợp với các đặc tả của nó.
* Xác thực chất lượng kiểm thử phần mềm đã dùng chi phí và nỗ lực ít nhất.
* Thiết kế tài liệu kiểm thử một cách có hệ thống, thực hiện nó sao cho có hiệu quả

1. **Các chiến lược kiểm thử phần mềm**
   1. **Quy trình kiểm thử phần mềm**

Khái niệm:

Quy trình kiểm thử phần mềm là một tập hợp các hoạt động, phương thức mà chúng ta phải làm để thực hiện kiểm thử cho một phần mềm hoặc một hệ thống phần mềm.

Tổng quát quy trình kiểm thử

* Lập kế hoạch kiểm thử là quá trình tạo ra bản kế hoạch kiểm thử, bao gồm 6 nhiệm vụ:
* Xác định phạm vi kiểm thử.
* Xác định cách tiếp cận việc kiểm thử.
* Thực thi theo chính sách và chiến lược kiểm thử.
* Xác định nguồn lực kiểm thử cần thiết.
* Lên kế hoạch cho hoạt động phân tích, thiết kế, thực thi.
* Xác định tiêu chí kết thúc kiểm thử.
* Phân tích và thiết kế kiểm thử là hoạt động chuyển các mục tiêu của kiểm thử thành các trường hợp kiểm thử. Gồm 7 hoạt động chính:
* Kiểm tra lại tất cả các loại tài liệu của dự án.
* Phân tích và đánh giả khả năng kiểm thử được của hệ thống.
* Xác định và đặt thứ tự ưu tiên cho các điều kiện kiểm thử.
* Thiết kế và đặt ưu tiên cho các tình huống kiểm thử mức cao.
* Xác định dữ liệu kiểm thử cần thiết cho các điều kiện và tình huống kiểm thử.
* Thiết kế cho việc thiết lập môi trường kiểm thử, xác định yêu cầu cơ sở hạ tầng.
* Thực hiện kiểm thử là quá trình bao gồm 8 nhiệm vụ:
* Phát triển và đặt thứ tự ưu tiên cho các thử tục kiểm thử.
* Xây dựng các bộ kiểm thử tự động.
* Cài đặt và kiểm tra môi trường kiểm thử.
* Thực hiện kiểm thử cho một bộ hoặc theo các.
* Ghi lại kết quả của kiểm thử.
* So sánh kết quả kiểm thử thực tế với kết quả mong đợi.
* Báo cáo sự cố và phân tích nguyên nhân.
* Lặp lại hoạt động kiểm thử cho mỗi sự cố đã được xác định.
* Báo cáo và đánh giá kiểm thử gồm 7 hoạt động chính:
* Kiểm tra các sản phẩm thực tế được bàn giao sao với kế hoạch.
* Đóng các báo cáo về sự cố hoặc ghi chép các thay đổi cho bất cứ vấn đề nào.
* Viết biên bản chấp nhận phần mềm.
* Lưu trữ các sản phẩm kiểm thử, môi trường và cơ sở hạ tầng cho các lần sử dụng sau.
* Bàn giao các sản phẩm kiểm thử cho các bộ phận quản lý dữ liệu và bảo trị sản phẩm.
* Phân tích các bài học để xác định những điểm cần thay đổi cho dự án sau.
* Sử dụng các thông tin thu nhập được để cải tiến công việc kiểm thử một cách định kì.
  1. **Các cấp độ kiểm thử phần mềm**

Có 4 cấp độ kiểm thử chinh: kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống, kiểm thử chấp nhận.

UNIT TEST

INTERGRATION TEST

ACCEPTANCE TEST

Các nhóm unit

Toàn bộ hệ thống

Toàn bộ hệ thống nhìn từ phía khách hàng

Các unit và compernent đơn giản

SYSTEM TEST

***Hình 1: các cấp độ kiểm thử***

* + 1. **Kiểm thử đơn vị.**

Kiểm thử đơn vị là một loại kiểm thử phần mềm. trong đó các đơn vị hay các thành phần của phần mềm được kiểm thử. Kiểm thử đơn vị được thực hiện trong quá trình phát triển (coding) ứng dụng.

Một đơn vị là một thành phần phần mềm nhỏ nhất mà ta có thể kiểm thử được. Ví dụ, các hàm *(Function)*, thủ tục *(Procedure),* lớp *(Class)* hay phương thức *(Method)* đều có thể được xem là Unit.

Unit Test thường là do lập trình viên thực hiện. Trong giai đoạn viết code và xuyên suốt chu kỳ phát triển phần mềm thì cần được test càng sớm càng tốt. Thông thường, Unit Test đòi hỏi tester có kiến thức về thiết kế và lập trình của chương trình. Mục đích của Unit Test là bảo đảm thông tin được xử lý và xuất ra thật chính xác, trong mối quan hệ giữa 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

Cùng với các mục kiểm thử khác, Unit Test cũng đòi hỏi phải chuẩn bị trước các ca kiểm thử (*Test case*) hoặc kịch bản kiểm thử (*Test script*), trong đó chỉ định rõ dữ liệu đầu vào, các bước thực hiện và dữ liệu đầu ra mong muốn. Các Test case và Test script này nên được giữ lại để tái sử dụng.

**Ưu điểm:**

* Cho phép lập trình viên cấu trúc lại code vào một ngày sau đó và đảm bảo mô-đun vẫn hoạt động chính xác .
* Quy trình viết các trường hợp kiểm thử cho tất cả các hàm và phương thức để mỗi khi thay đổi gây ra lỗi, nó có thể được xác định và sửa chữa nhanh chóng.
* Có thể kiểm thử các phần của dự án mà không cần chờ người khác hoàn thành.

**Nhược điểm:**

* Kiểm thử đơn vị không thể phát hiện được tất cả các lỗi trong một chương trình.
* Không thể đánh giá tất cả các luồng thực hiện ngay cả trong các chương trình tầm thường nhất
* Kiểm thử đơn vị không thể bắt lỗi tích hợp hoặc lỗi hệ thống lớn vì bả chất kiểm thử đơn vị chỉ tập chung vào một đơn vị code.

Kiểm thử đơn vị có thể đơn giản, cũng có thể phức tạp tùy thuộc vào ứng dụng đang được kiểm thử và các chiến lược, công cụ, triết lý kiểm thử được sử dụng. Kiểm thử đơn vị luôn luôn cần thiết ở nhiều cấp độ.

* + 1. **Kiểm thử tích hợp - Integration Test**

Integration test kết hợp các thành phần của một ứng dụng và kiểm thử 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.

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

Mục tiêu:

* Phát hiện lỗi giao tiếp xảy ra giữa các Unit.
* Tích hợp các Unit đơn lẻ thành các hệ thống nhỏ và cuối cùng là nguyên hệ thống hoàn chỉnh (*System*) chuẩn bị cho kiểm thử 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 thử đơn giản trên giao tiếp giữa Unit với các thành phần liên quan khác, tuy nhiên mọi giao tiếp liên quan đến Unit chỉ thật sự được kiểm tra đầy đủ khi các Unit tích hợp với nhau trong khi thực hiện Integration Test.

Trong Integration Test ta 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 các đợt Integration Test trước đó. Lúc này, ta chỉ cần kiểm thử giao tiếp của Unit mới thêm vào với hệ thống các Unit đã tích hợp trước đó, điều này sẽ làm cho số lượng can kiểm thử giảm đi rất nhiều, và sai sót sẽ giảm đáng kể.

Các phương pháp kiểm thử tích hợp:

* Phương pháp big bang

Ưu điểm: Thuận tiện cho các hệ thống nhỏ.

Nhược điểm:

* Khó để kiểm tra lỗi nội địa hóa (Localization) .
* Một số liên kết giao diện cần kiểm thử có thể dễ dàng bị bỏ qua.
* Người thử nghiệm sẽ có ít thời gian thực hiện hơn trong giai đoạn thử nghiệm. Vì Kiểm thử Tích hợp chỉ có thể bắt đầu sau khi "tất cả" các mô-đun được thiết kế.
* Có rủi ro cao không được ưu tiên hay kiểm tra riêng. Các mô-đun ngoại vi liên quan đến giao diện người dùng cũng không được ưu tiên hay kiểm tra riêng.
* Phương pháp gia tăng

Cách tiếp cận gia tăng được thực hiện bởi hai Phương pháp khác nhau:

* Phương pháp Bottom up
* Phương pháp Top Down
* Phương pháp Sandwich - Kết hợp từ trên xuống và từ dưới lên
  + 1. **Kiểm thử hệ thống – System Test**

Kiểm thử hệ thống là một phương pháp theo dõi và đánh giá hành vi của sản phẩm hoặc hệ thống phần mềm hoàn chỉnh và đã được tích hợp đầy đủ, dựa vào đặc tả và các yêu cầu chức năng đã được xác định trước.

Mục đích System Test là kiểm thử 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 phần mềm đã được tích hợp thành công. Thông thường loại kiểm thử này tốn rất nhiều công sức và thời gian. Trong nhiều trường hợp, việc kiểm thử 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 thử 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.

Sau khi hoàn thành Integration Test, một hệ thống phần mềm đã được hình thành cùng với các thành phần đã được kiểm tra đầy đủ. Tại thời điểm này, lập trình viên hoặc kiểm thử viên bắt đầu kiểm thử phần mềm 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.

System Test 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 thử này đặc biệt thích hợp cho việc phát hiện lỗi giao tiếp với phần mềm hoặc phần cứng bên ngoài, chẳng hạn các lỗi "tắc nghẽn" (deadlock) hoặc chiếm dụng bộ nhớ. Sau giai đoạn System Test, phần mềm thường đã sẵn sàng cho khách hàng hoặc người dùng cuối cùng kiểm thử chấp nhận sản phẩm (*Acceptance Test*) hoặc dùng thử (*Alpha/Beta Test*).

System Test thường được thực hiện bởi một nhóm kiểm thử viên hoàn toàn độc lập với nhóm phát triển dự án. Bản thân System Test lại gồm nhiều loại kiểm thử khác nhau, phổ biến nhất gồm:

* **Kiểm thử chức năng (Functional Test):** Bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
* **Kiểm thử hiệu năng (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 thử 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 như đang giao dịch thì ngắt kết nối (xuất hiện nhiều trong kiểm tra thiết bị như POS, ATM...)...
* **Kiểm thử cấu hình (Configuration Test).**
* **Kiểm thử bảo mật (Security Test):** Bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.
* **Kiểm thử 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...

Các cấp độ kiểm thử trên rất quan trọng khi nhìn từ phía người dùng: Chúng bảo đảm hệ thống đủ khả năng làm việc trong môi trường thực.

**Lưu ý** : không nhất thiết phải thực hiện tất cả các loại kiểm thử nêu trên. Tùy yêu cầu và đặc trưng của từng hệ thống, tuỳ khả năng và thời gian cho phép của dự án, khi lập kế hoạch, người Quản lý dự án sẽ quyết định áp dụng những loại kiểm thử nào.

* + 1. **Kiểm thử chấp nhận**

Thường là sau khi System Test là Acceptance Test và khách hàng là người thực hiện (hoặc ủy quyền cho một nhóm thứ ba thực hiện).

Mục đích của Acceptance Test là để chứng minh phần mềm 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 thử 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 thử gọi là kiểm thử Alpha –***Alpha Test*** và kiểm thử Beta – ***Beta Test***. Với Alpha Test, người dùng kiểm thử phần mềm ngay tại nơi phát triển phần mềm, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa. Với Beta Test, phần mềm sẽ được gửi tới cho người dùng để kiểm thử ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

Nếu khách hàng không quan tâm và không tham gia vào quá trình phát triển phần mềm thì kết quả Acceptance Test sẽ sai lệch rất lớn, mặc dù phần mềm đã trải qua tất cả các kiểm thử trước đó. Sự sai lệch này liên quan đến việc hiểu sai yêu cầu cũng như sự mong chờ của khách hàng. Ví dụ đôi khi một phần mềm xuất sắc vượt qua các phép kiểm thử về chức năng thực hiện bởi nhóm thực hiện dự án, nhưng khách hàng khi kiểm thử sau cùng vẫn thất vọng vì bố cục màn hình nghèo nàn, thao tác không tự nhiên, không theo tập quán sử dụng của khách hàng v.v...

Gắn liền với giai đoạn Acceptance Test thường là một nhóm những dịch vụ và tài liệu đi kèm, phổ biến như hướng dẫn cài đặt, sử dụng v.v... Tất cả tài liệu đi kèm phải được cập nhật và kiểm thử chặt chẽ.

* 1. **Các loại hình kiểm thử phần mềm**

Các loại hình kiểm thử gồm có: kiểm thử chức năng, kiểm thử phi chức năng, kiểm thử liên quan đến sự thay đổi.

Kiểm thử chức năng.

* Là quy trình cố gắng tìm ra các khác biệt giữa đặc tả bên ngoài của phần mềm và thực tế mà phần mềm cung cấp. Đặc tả về bên ngoài của phần mềm là đặc tả chính xác về hành vi của phần mềm theo góc nhìn của người dùng thấy.
* Mục tiêu: đảm bảo đúng mục tiêu của kiểm thử chức năng nhập dữ liệu – xử lý – lấy và kiểm tra kết quả trả về.
* Các loại kiểm thử chức năng:
* Kiểm thử chức năng của hệ thống.
* Kiểm thử tích hợp dữ liệu và cơ sở dữ liệu.
* Kiểm thử vòng lặp công việc.
* Kiểm thử truy cập.
* Kiểm thử giao diện.

Kiểm thử phi chức năng.

* Tập trung vào kiểm thử sản phẩm, hệ thống phần mềm cần kiểm thử có những đặc tính tốt như thế nào.
* Sử dụng hiệu quả nhất trong kiểm thử hệ thống và kiểm thử chấp nhận sản phẩm.
* 4 loại kiểm thử phi chức năng thường dùng:
* Kiểm thử hiệu năng
* Kiểm thử tải trọng
* Kiểm thử tập trung.
* Kiểm thử với dữ liệu lớn.

Kiểm thử liên quan đến sự thay đổi.

* Thực hiện hoạt động kiểm thử khi có sự thay đổi trên hoặc trong sản phẩm phần mềm.
* Sự thay đổi của sản phẩm phần mềm có thể là:
* Sửa chữa các lỗi tìm được.
* Sản phẩm được nâng cấp được thay đổi về chức năng.
* Gồm: kiểm thử lại và kiểm thử hồi quy.

1. **Các phương pháp kiểm thử phần mềm**
   1. **Kiểm thử hộp đen**

Kỹ thuật kiểm thử hộp đen coi hệ thống giống một “hộp đen” mà cấu trúc bên trong của chương trình không thể nhìn thấy. Người kiểm thử chỉ kiểm tra các chức năng của phần mềm không cần quan tâm vào cấu trúc bên trong hay hoạt động của nó.

Kiểm thử hộp đen có các đặc trưng như:

* Nhằm đảm bảo các chức năng đủ và vận hành đúng.
* Thực hiện các phép thử qua giao diện.

Mục tiêu cảu kiểm thử phần mềm là tìm ra các loại sai:

* Thiếu chức năng hoạc chức năng không đúng đắn.
* Giao diện sai.
* Sai trong cấu trúc hoặc truy cập dữ liệu ngoài.
* Thực thi chức năng sai.
* Sai khởi đầu hoặc kết thúc module.

Các phương pháp kiểm thử hộp đen

* Phương pháp phân hoạch tương đương
* Phương pháp phân tích giá trị biên
* Bảng hỗ trợ quyết định

Ưu và nhược điểm của kiểm thử hộp đen

Về ưu điểm:

* Người kiểm thử có thể đánh giá phần mềm mềm một cách khách quan mà không cần biết lập trình, tách biệt với quan điểm của lập trình viên.

Nhược điểm:

* Độ bao phủ hạn chế do chỉ có phần nhỏ các test case được thực hiện.
* Do người thực hiện kiểm thử không biết cấu trúc bên trong chương trình nên việc kiểm thử không hiệu quả.
* Người kiểm thử có hiểu biết hạn chế về chương trình.
  1. **Kiểm thử hộp trắng**

Tổng quan về kỹ thuật kiểm thử hộp trắng

Kiểm thử hộp trắng sử dụng để kiểm tra các đoạn mã chương trình phần mềm xem nó có vận hành đúng theo thiết kế hay không.

Đặc điểm của kiểm thử hộp trắng:

* Là chiến lược giải thuật phụ thuộc vào giải thuật, cấu trúc bên trong chương trình phần mềm.
* Người kiểm thử phải có kiến thức nhất định về ngôn ngữ lập trình được dùng, hiểu giải thuật được sử dụng bên trong chương trình.
* Việc kiểm thử được tiến hành kiểm tra xem giải thuật, câu lệnh có vận hành như thiết kế hay không.
* Phương pháp này bắt buộc phải viết test case bao phủ các nhánh trong giải thuật, đảm bảo thực hiện đầy đủ.

Mục đích của kiểm thử hộp trắng là bao phủ hết các câu lệnh, điều kiện, các rẽ nhánh trong mã nguồn chương trình.

1.3.2.2 Ưu và nhược điểm của while-box testing

Ưu điểm:

* Phù hợp để tìm kiếm lỗi và các vấn đề giải thuật trong mã lệnh.
* Có thể tìm ra được các lỗi tiềm ẩn bên trong chương trình.
* Lập trình viên có thể tự kiểm tra đoạn mã của mình.
* Việc kiểm thử rà soát lỗi hiệu quả nhất vì yêu cầu kiến thức cấu trúc bên trong của chương trình.

Nhược điểm:

* Kiểm thử hộp trắng không thể tìm thấy tính năng chưa thực hiện hoặc bỏ sót.
* Người thiết kế test case phải hiểu sâu về cấu trúc bên trong và đánh giá được chương trình.
* Yêu cầu truy suất mã lệnh bên trong chương trình.
  1. **Kiểm thử hộp xám**

Kỹ thuật kiểm thử hộp xám có sự kết hợp giữa kiểm thử hộp đen và kiểm thử hộp trắng. Kiểm thử hộp xám vừa quan tâm đến dữ liệu đầu vào (của kiểm thử hộp đen), xong cũng đòi hỏi truy cập đến cấu trúc và giải thuật (của kiểm thử hộp trắng) để thiết kế các ca kiểm thử.

Ưu và nhược điểm của kiểm thử hộp xám:

Ưu điểm:

* Chứa các lợi ích của cả 2 kỹ thuật kiểm thử hộp đen và kiểm thử hộp trắng.
* Xác định các yêu cầu từ ban đầu dựa theo đặc tả chức năng, mô tả và sơ đồ kiến trúc hệ thống.

Nhược điểm:

* Kiểm tra từng đường dẫn gây tốn nhiều thời gian.
* Khi thực hiện kiểm thử hộp xám cho một ứng dụng có hệ thống phân tán thì rất khó để liên kết lỗi.
* Không phù hợp khi thực hiện kiểm thử một số loại chức năng.

1. **Kiểm thử phần mềm tự động**
   1. **Tổng quan kiểm thử phần mềm tự động**

Kiểm thử tự độnglà hoạt động xử lý một cách tự động các bước thực hiện các testcase, kiểm thử với một công cụ nhằm rút ngắn thời gian kiểm thử. Là kỹ thuật tự động trong đó người kiểm thử tự viết các lệnh và sử dụng phần mềm phù hợp để kiểm thử phần mềm. Nó về cơ bản là một quá trình tự động hóa của một quy trình kiểm thử thủ công. Kiểm thử tự động có thể giúp giảm chi phí kiểm thử phần mềm bằng cách hỗ trợ quá trình kiểm thử thông qua các công cụ phần mềm.

Mục đích của kiểm thử tự động:

* Giảm bớt nhân lực và thời gian kiểm thử.
* Tăng độ tin cậy .
* Tăng độ hứng thú trong kiểm thử cho con người.
* Tăng kỹ năng lập trình
* Giảm thiểu được chi phí cho quá trình kiểm thử.

Cần kiểm thử khi:

* Không đủ tài nguyên
* Kiểm tra khả năng vận hành trong môi trường đặc biệt.
* Kiểm tra quy hồi
  1. **Quy trình của kiểm thử phần mềm tự động**

Quy trình kiểm thử phần mềm tự động trải qua 6 bước:

Bước 1: Lập kế hoạch Kiểm thử

Bước 2: Thiết kế ca kiểm thử

Bước 3: Phát triển test script

Bước 4: Thực hiện kiểm thự tự động

Bước 5: Kết quả

Bước 6: Đánh giá kết quả kiểm thử

Sau khi đánh giá kết quả kiểm thử:

* Cập nhật kế hoạch kiểm thử nếu kiểm thử chưa thỏa mức độ bao phủ yêu cầu phần mềm.
* Cập nhật các ca kiểm thử nếu gặp lỗi thiết kế sai yêu cầu
* Cập nhật test script nếu gặp lỗi do phát triển test script.
  1. **Các ưu, nhược điểm của kiểm thử phần mềm tự động**

Ưu điểm:

* Kiểm thử tự động không cần sự can thiệp của tester
* Độ tin cậy của kiểm thử tự động đạt mức tối ưu hơn so với kiểm thử thủ công
* Tiết kiệm chi phí thực hiện với số lượng lớn test case hoặc test case lặp lại nhiều lần
* Kiểm thử được các tình huống không thể thực hiện bằng tay
* Tốc độ kiểm thử nhanh
* Dễ dàng tái sự dụng

Khuyết điểm:

* Khó bảo trì và mở rộng
* Không áp dụng được với các lỗi mới
* Đòi hỏi tester phải có kỹ năng cao
  1. **Một số công cụ kiểm thử phần mềm tự động**

**Selenium**: Selenium là một trong những công cụ kiểm thử phần mềm tự động mã nguồn mở mạnh mẽ nhất hiện nay cho việc kiểm thử ứng dụng Web. Selenium script có thể chạy được trên hầu hết các trình duyệt như IE, Mozilla FireFox, Chrome, Safari, Opera và hầu hết các hệ điều hành như Windows, Mac, Linux. Selenium hỗ trợ nhiều ngôn ngữ lập trình.

**QTP (Quick Test Professional)**: QTP được sử dụng rộng rãi để kiểm tra chức năng và hồi quy, giải quyết các ứng dụng phần mềm và môi trường. QTP giúp tester tiến hành các kiểm tra một cách tự động để xác định errors, defects khác với kết quả mong muốn của ứng dụng, phần mềm hay chức năng mà ta đang kiểm tra

**Junit**: Junit là một framework đơn giản dùng cho việc tạo các unit testing tự động, chạy các test có thể lặp đi lặp lại. Nó chỉ là một phần của kiến trúc xUnit cho việc tạo các unit testing. Junit là một chuẩn thực tế cho unit testing trong Java. Junit tránh cho người lập trình phải làm đi làm lại những việc kiểm thử nhàm chán bằng cách tách biệt mã kiểm thử ra khỏi mã chương trình, đồng thời tự động hóa việc tổ chức và thi hành các bô số kiểm thử.

**Robotium**: Robotium là một khung kiểm thử tự động mã nguồn mở, được sử dụng để kiểm thử hộp đen mạnh mẽ và đặc biệt là các ứng dụng Android. Nó hỗ trợ đầy đủ cho các trường hợp kiểm thử ứng dụng gốc và lai. Ứng dụng gốc được phát trực tiếp trên thiết bị, được thiết kế cho một nền tảng cụ thể và có thể được cài đặt từ Cửa hàng Google Play. Trong khi đó ứng dụng lai chứ một phần web cơ bản và một phần là ứng dụng mobile, nó cũng có thể được cài đặt từ kho ứng dụng, nhưng yêu cầu HTML phải được hiển thị trong trình duyệt.

**Rational Function Tester**: Là một công cụ kiểm tra tự động hướng đối tượng có khả năng tự động kiểm tra dữ liệu, kiểm tra giao diện và kiểm thử hồi quy. Hỗ trợ một loạt các giao thức và ứng dụng như Java, HTML, .Net, Windows, SAP, Visual Basic

# **CHƯƠNG 2: THIẾT KẾ TEST CASE**

# **1. Thiết kế Test Case là gì ?**

* Tập hợp các trường điều kiện mà Tester dựa vào đó để xác định ứng dụng, hệ thống phần mềm là một trong các chức của nó có hoạt động như mong muốn hay không, đó được gọi là test – case.
* Thiết kế test – case trong kiểm thử phần mềm là quá trình xây dựng các phương pháp kiểm thử có thể phát hiện lỗi, sai sót, khuyết điểm của phần mềm để xây dựng phần mềm đạt tiêu chuẩn.
* Vai trò của việc thiết kế test – case:
* Tạo ra các ca kiểm thử tốt nhất có khả năng phát hiện ra lỗi, sai sót của phần mềm một cách nhiều nhất.
* Tạo ra các ca kiểm thử có chi phí rẻ nhất, đồng thời tốn ít thời gian và công sức nhất.

# **2. Các kỹ thuật thiết kế Test case**

Một trong những lý do quan trọng nhất trong kiểm thử phần mềm là thiết kế và tạo ra các ca kiểm thử (Test case) có hiệu quả (ca kiểm thử tốt nhất có khả năng phát hiện ra lỗi, sai sót của phần mềm một cách nhiều nhất). Với những ràng buộc về thời gian và chi phí đã cho, thì vấn đề then chốt của kiểm thử là phải trả lời câu hỏi: Tập con nào của tất cả ca kiểm thử có thể có khả năng tìm ra nhiều lỗi nhất?

Thông thường, phương pháp kém hiệu quả nhất là kiểm tra tất cả đầu vào ngẫu nhiên – quá trình kiểm thử một chương trình bằng việc chọn ngẫu nhiên một tập con các giá trị đầu vào có thể. Về mặt khả năng tìm ra nhiều lỗi nhất, tập hợp các ca kiểm thử được chọn ngẫu nhiên có rất ít cơ hội là tập hợp tối ưu hay gần tối ưu. Sau đây là một số phương pháp để chọn ra một tập dữ liệu kiểm thử một cách thông minh. Để kiểm thử hộp đen và kiểm thử hộp trắng một cách thấu đáo là không thể. Do đó, một chiến lược kiểm thử hợp lý là chiến lược có thể kết hợp sức mạnh của cả hai phương pháp trên: Phát triển một cuộc kiểm thử nghiêm ngặt vừa bằng việc sử dụng các phương pháp thiết kế ca kiểm thử hướng hộp đen nào đó và sau đó bổ sung thêm những ca kiểm thử này bằng việc khảo sát tính logic của chương trình, sử dụng phương pháp hộp trắng.

Những chiến lược kết hợp đó bao gồm:

|  |  |
| --- | --- |
| Hộp đen | Hộp trắng |
| 1. Phân lớp tương đương. 2. Phân tich giá trị biên. 3. Đồ thị nguyên nhân – kết quả. 4. Đoán lỗi. | 1. Bao phủ câu lệnh. 2. Bao phủ quyết định. 3. Bao phủ điều kiện. 4. Bao phủ điều kiện – quyết định. 5. Bao phủ đa điều kiện. |

Để có tập kiểm thử tối ưu, chúng ta phải kết hợp hầu hết các phương pháp. Quy trình thiết kế các ca kiểm thử sẽ bắt đầu bằng việc phát triển các ca kiểm thử sử dụng phương pháp hộp đen và sau đó phát triển bổ sung các ca kiểm thử cần thiết với phương pháp hộp trắng.

## **2.1. Kiểm thử hộp trắng – kiểm thử bao phủ logic**

### **2.1.1. Kỹ thuật bao phủ câu lệnh** - **Statement Coverage**

Tư tưởng: Thực hiện mọi câu lệnh trong chương trình ít nhất 1 lần.

Xét ví dụ với đoạn mã lệnh JAVA sau:

public void foo (int a, int b, int x) {

if (a>1 && b==0) {

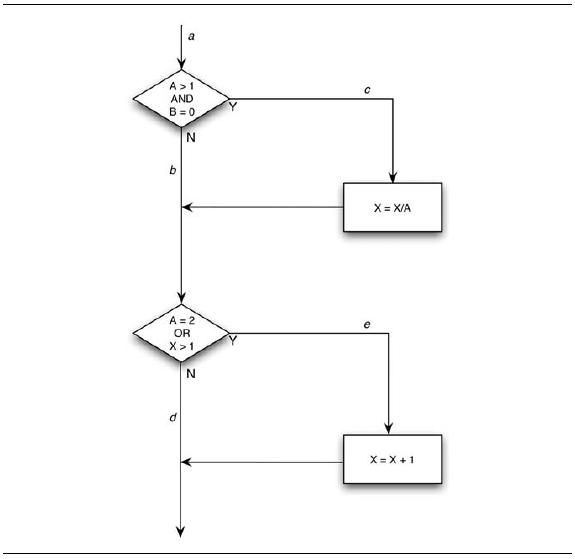
x=x/a;}

if (a==2||x>1{

x=x+1;

}

}



*Hình 3.1Một chương trình nhỏ để kiểm thử.*

Có thể thực hiện mọi câu lệnh bằng việc viết 1 ca kiểm thử đơn đi qua đường *ace*. Tức là, bằng việc đặt A=2, B=0 và X=3 tại điểm a, mỗi câu lệnh sẽ được thực hiện 1 lần (thực tế, X có thể được gán bất kỳ giá trị nào).

Thường tiêu chuẩn này khá kém. Ví dụ, có lẽ nếu quyết định đầu tiên là phép *or* chứ không phải phép *and* thì lỗi này sẽ không được phát hiện. Hay nếu quyết định thứ hai mà bắt đầu với *x>0*, lỗi này cũng sẽ không được tìm ra. Cũng vậy, có 1 đường đi qua chương trình mà ở đó *x* không thay đổi (đường đi *abd*). Nếu đây là 1 lỗi, thì lỗi này có thể không tìm ra. Hay nói cách khác, tiêu chuẩn bao phủ câu lệnh quá yếu đến nỗi mà nó thường là vô ích.

### **2.1.2. Kỹ thuật bao phủ quyết định – Decision Coverage**

Tư tưởng**:** Viết đủ các ca kiểm thử mà mỗi quyết định có kết luận đúng hay sai ít nhất 1 lần. Nói cách khác, mỗi hướng phân nhánh phải được xem xét kỹ lưỡng ít nhất 1 lần.

***Các ví dụ về câu lệnh rẽ nhánh hay quyết định là các câu lệnh switch, do-while, và if-else­. Các câu lệnh đa đường GOTO thường sử dụng trong một số ngôn ngữ lập trình như FORTRAN.***

Bao phủ quyết định thường thỏa mãn bao phủ câu lệnh. Vì mỗi câu lệnh là trên sự bắt nguồn một đường đi phụ nào đó hoặc là từ 1 câu lệnh rẽ nhánh hoặc là từ điểm vào của chương trình, mỗi câu lệnh phải được thực hiện nếu mỗi quyết định rẽ nhánh được thực hiện. Tuy nhiên, có ít nhất 3 ngoại lệ:

* Những chương trình không có quyết định.
* Những chương trình hay thường trình con/phương thức với nhiều điểm vào. Một câu lệnh đã cho có thể được thực hiện nếu và chỉ nếu chương trình được nhập vào tại 1 điểm đầu vào riêng.
* Các câu lệnh bên trong các ON-unit. Việc đi qua mỗi hướng rẽ nhánh sẽ là không nhất thiết làm cho tất cả các ON-unit được thực thi.

Vì chúng ta đã thấy rằng bao phủ câu lệnh là điều kiện cần thiết, nên một chiến lược tốt hơn là bao phủ quyết định nên được định nghĩa bao hàm cả bao phủ câu lệnh. Do đó, bao phủ quyết định yêu cầu mỗi quyết định phải có kết luận đúng hoặc sai, và mỗi câu lệnh đó phải được thực hiện ít nhất 1 lần.

Phương pháp này chỉ xem xét những quyết định hay những sự phân nhánh 2 đường và phải được sửa đổi cho những chương trình có chứa những quyết định đa đường.

Ví dụ, các chương trình JAVA có chứa các lệnh *select (case)*, các chương trình FORTRAN chứa các lệnh số học (ba đường) *if* hoặc các lệnh tính toán hay số học *GOTO*, và các chương trình COBOL chứa các lệnh *GOTO* biến đổi hay các lệnh *GO-TO-DEPENDING-ON* (các lệnh goto phụ thuộc). Với những chương trình như vậy, tiêu chuẩn này đang sử dụng mỗi kết luận có thể của tất cả các quyết định ít nhất 1 lần và gọi mỗi điểm vào tới chương trình hay thường trình con ít nhất 1 lần.

Trong hình 3.1, bao phủ quyết định có thể đạt được bởi ít nhất 2 ca kiểm thử bao phủ các đường *ace* và *abd* hoặc *acd* và *abe*. Nếu chúng ta chọn khả năng thứ hai, thì 2 đầu vào test-case là *A=3, B=0, X=3* và *A=2, B=1, X=1*.

Bao phủ quyết định là 1 tiêu chuẩn mạnh hơn bao phủ câu lệnh, nhưng vẫn khá yếu. Ví dụ, chỉ có 50% cơ hội là chúng ta sẽ tìm ra con đường trong đó *x* không bị thay đổi (ví dụ, chỉ khi bạn chọn khả năng thứ nhất). Nếu quyết định thứ hai bị lỗi (nếu như đáng lẽ phải nói là *x<1* thay vì *x>1*), lỗi này sẽ không được phát hiện bằng 2 ca kiểm thử trong ví dụ trước.

* + 1. **Kỹ thuật bao phủ điều kiện – Condition Coverage.**

Tư tưởng: Viết đủ các ca kiểm thử để đảm bảo rằng mỗi điều kiện trong một quyết định đảm nhận tất cả các kết quả có thể ít nhất một lần.

Vì vậy, như với bao phủ quyết định, thì bao phủ điều kiện không phải luôn luôn dẫn tới việc thực thi mỗi câu lệnh. Thêm vào đó, trong tiêu chuẩn bao phủ điều kiện, mỗi điểm vào chương trình hay thường trình con, cũng như các ON-unit, được gọi ít nhất 1 lần. Ví dụ, câu lệnh rẽ nhánh do k=0 to 50 while (j+k<quest) có chứa 2 điều kiện là k<=50, và j+k<quest. Do đó, các ca kiểm thử sẽ được yêu cầu cho những tình huống k<=50, k>50 (để đến lần lặp cuối cùng của vòng lặp), j+k<quest, và j+k>=quest.

Hình 3.1 có 4 điều kiện: A>1, B=0, A=2, X>1. Do đó các ca kiểm thử đầy đủ là cần thiết để thúc đẩy những trạng thái mà A>1, A<=1, B=0 và B<>0 có mặt tại điểm a và A=2, A<>2, X>1, X<=1 có mặt tại điểm b. Số lượng đầy đủ các ca kiểm thử thỏa mãn tiêu chuẩn và những đường đi mà được đi qua bởi mỗi ca kiểm thử là:

1. A=2, B=0, X=4 ace

2. A=1, B=1, X=1 abd

Chú ý là, mặc dù cùng số lượng các ca kiểm thử được tạo ra cho ví dụ này, nhưng bao phủ điều kiện thường tốt hơn bao phủ quyết định là vì nó có thể (nhưng không luôn luôn) gây ra mọi điều kiện riêng trong 1 quyết định để thực hiện với cả hai kết quả, trong khi bao phủ quyết định lại không. Ví dụ trong cùng câu lệnh rẽ nhánh: DO K=0 TO 50 WHILE(J+K<QUEST) là 1 nhánh 2 đường (thực hiện thân vòng lặp hay bỏ qua nó). Nếu bạn đang sử dụng kiểm thử quyết định, thì tiêu chuẩn này có thể được thỏa mãn bằng cách cho vòng lặp chạy từ K=0 tới 51, mà chưa từng kiểm tra trường hợp trong đó mệnh đề WHILE bị sai. Tuy nhiên, với tiêu chuẩn bao phủ điều kiện, 1 ca kiểm thử sẽ cần phải cho ra 1 kết quả sai cho những điều kiện J+K<QUEST.

Mặc dù nếu mới nhìn thoáng qua, tiêu chuẩn bao phủ điều kiện xem ra thỏa mãn tiêu chuẩn bao phủ quyết định, nhưng không phải lúc nào cũng vậy. Nếu quyết định IF (A&B) được kiểm tra, thì tiêu chuẩn bao phủ điều kiện sẽ cho phép bạn viết 2 ca kiểm thử - A đúng, B sai, và A sai, B đúng – nhưng điều này sẽ không làm cho mệnh đề THEN của câu lệnh IF được thực hiện.

Ví dụ, 2 ca kiểm thử khác:

1. A=1, B=0, X=3

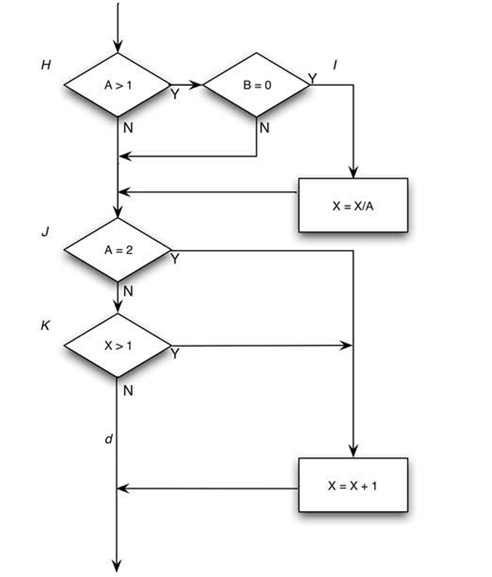
2. A=2, B=1, X=1

bao phủ tất cả các kết quả điều kiện, nhưng chúng chỉ bao phủ 2 trong 4 kết quả quyết định (cả 2 đều bao phủ đường đi abd và do đó, không sử dụng kết quả true của quyết định đầu tiên và kết quả false của quyết định thứ hai).

* + 1. **Kỹ thuật bao phủ quyết định/ điều kiện – Decision/ Condition Coverage**

Tư tưởng*:* Thực hiện đủ các ca kiểm thử mà mỗi điều kiện trong 1 quyết định thực hiện trên tất cả các kết quả có thể ít nhất 1 lần, và mỗi điểm vào được gọi ít nhất 1 lần.

Điểm yếu của bao phủ quyết định/điều kiện là mặc dù xem ra nó có thể sử dụng tất cả các kết quả của tất cả các điều kiện, nhưng thường không phải vậy vì những điều kiện chắc chắn đã cản các điều kiện khác.



*Hình 3.2 Mã máy cho chương trình trong hình 3.1*

Các quyết định đa điều kiện trong chương trình nguồn đã bị chia thành các quyết định và các nhánh riêng vì hầu hết các máy không được chế tạo để có thể thực hiện các quyết định đa điều kiện. Khi đó 1 bao phủ kiểm thử tỉ mỉ hơn xuất hiện là việc sử dụng tất cả các kết quả có thể của mỗi quyết định gốc. Hai ca kiểm thử bao phủ quyết định trước không làm được điều này; chúng không thể sử dụng kết quả *false* của quyết định H và kết quả *true* của quyết định K.

Lí do, như đã được chỉ ra trong hình 3.2, là những kết quả của các điều kiện trong các biểu thức and và or có thể cản trở hay ngăn chặn việc ước lượng các quyết định khác. Ví dụ, nếu 1 điều kiện and là sai, không cần kiểm tra các điều kiện tiếp theo trong biểu thức. Tương tự như vậy, nếu 1 điều kiện or là đúng thì cũng không cần kiểm tra các điều kiện còn lại. Do đó, các lỗi trong biểu thức logic không phải lúc nào cũng được phát hiện bằng các tiêu chuẩn bao phủ điều kiện và bao phủ quyết định/điều kiện.

* + 1. **Kỹ thuật bao phủ đa điều kiện – Multiple condition Coverage**

Tư tưởng: Viết đủ các ca kiểm thử mà tất cả những sự kết hợp của các kết quả điều kiện có thể trong mỗi quyết định, và tất cả các điểm vào phải được gọi ít nhất 1 lần.

Ví dụ, xét chuỗi mã lệnh sau:

*NOTFOUND = TRUE;*

*DO I=1 TO TABSIZE WHILE (NOTFOUND); /\*SEARCH TABLE\*/*

*…searching logic…;*

*END;*

Bốn tình huống để kiểm thử là:

1. *I<= TABSIZE* và *NOTFOUND* có giá trị đúng (đang duyệt)

2. *I<= TABSIZE* và *NOTFOUND* có giá trị sai (tìm thấy mục vào trước khi gặp cuối bảng).

3. *I>TABSIZE* và *NOTFOUND* có giá trị đúng (gặp cuối bảng mà không tìm thấy mục vào).

4. *I>TABSIZE* và *NOTFOUND* có giá trị sai (mục vào là cái cuối cùng trong bảng).

Dễ nhận thấy là tập hợp các ca kiểm thử thỏa mãn tiêu chuẩn đa điều kiện cũng thỏa mãn các tiêu chuẩn bao phủ quyết định, bao phủ điều kiện và bao phủ quyết định/điều kiện.

Quay lại hình 3.1, các ca kiểm thử phải bao phủ 8 sự kết hợp:

*1. A>1, B= 0*

*2. A>1, B<>0*

*3. A<=1, B=0*

*4. A<=1, B<>0*

*5. A=2, X>1*

*6. A=2, X<=1*

*7. A<>2, X>1*

*8. A<>2, X<=1*

Vì là ca kiểm thử sớm hơn, nên cần chú ý là các trường hợp từ 5 đến 8 biểu diễn các giá trị tại vị trí câu lệnh IF thứ hai. Vì X có thể thay đổi ở trên câu lệnh IF này, nên giá trị cần tại câu lệnh IF này phải được sao dự phòng thông qua tính logic để tìm ra các giá trị đầu vào tương ứng.

Những sự kết hợp để được kiểm tra này không nhất thiết ngụ ý rằng cần thực hiện cả 8 ca kiểm thử. Trên thực tế, chúng có thể được bao phủ bởi 4 ca kiểm thử. Các giá trị đầu vào kiểm thử, và sự kết hợp mà chúng bao phủ, là như sau:

A=2, B=0, X=4 Bao phủ trường hợp 1, 5

A=2, B=1, X=1 Bao phủ trường hợp 2, 6

A=1, B=0, X=2 Bao phủ trường hợp 3, 7

A=1, B=1, X=1 Bao phủ trường hợp 4, 8

Thực tế là việc có 4 ca kiểm thử và 4 đường đi riêng biệt trong hình 3.1 chỉ là sự trùng hợp ngẫu nhiên. Trên thực tế, 4 ca kiểm thử này không bao phủ mọi đường đi, chúng bỏ qua đường đi *acd.* Ví dụ, bạn sẽ cần 8 ca kiểm thử cho quyết định sau mặc dù nó chỉ chứa 2 đường đi:

*If (x==y&&length(z)==0&&FLAG) {*

*J=1;}*

*Else {*

*I=1;}*

Trong trường hợp các vòng lặp, số lượng các ca kiểm thử được yêu cầu bởi tiêu chuẩn đa điều kiện thường ít hơn nhiều số lượng đường đi.

Tóm lại, đối với những chương trình chỉ chứa 1 điều kiện trên 1 quyết định, thì 1 tiêu chuẩn kiểm thử nhỏ nhất là một số lượng đủ các ca kiểm thử để (1) gọi tất cả các kết quả của mỗi quyết định ít nhất 1 lần và (2) gọi mỗi điểm của mục vào (như là điểm vào hay ON-unit) ít nhất 1 lần, để đảm bảo là tất cả các câu lệnh được thực hiện ít nhất 1 lần. Đối với những chương trình chứa các quyết định có đa điều kiện thì tiêu chuẩn tối thiểu là số lượng đủ các ca kiểm thử để gọi tất cả những sự kết hợp có thể của các kết quả điều kiện trong mỗi quyết định, và tất cả các điểm vào của chương trình ít nhất 1 lần.

* 1. **Kiểm thử hộp đen**
     1. **Kỹ thuật phân lớp tương đương**

Phân lớp tương đương là một phương pháp kiểm thử hộp đen chia miền đầu vào của một chương trình thành các lớp dữ liệu, từ đó suy dẫn ra các ca kiểm thử. Phương pháp này cố gắng xác định ra một ca kiểm thử mà làm lộ ra một lớp lỗi, do đó làm giảm tổng số các trường hợp kiểm thử phải được xây dựng.

Thiết kế ca kiểm thử cho phân lớp tương đương dựa trên sự đánh giá về các lớp tương đương với một điều kiện vào. Lớp tương đương biểu thị cho tập các trạng thái hợp lệ hay không hợp lệ đối với điều kiện vào.

Một cách xác định tập con này là để nhận ra rằng 1 ca kiểm thử được lựa chọn tốt cũng nên có 2 đặc tính khác:

1. Giảm thiểu số lượng các ca kiểm thử khác mà phải được phát triển để hoàn thành mục tiêu đã định của kiểm thử “hợp lý”.

2. Bao phủ một tập rất lớn các ca kiểm thử có thể khác. Tức là, nó nói cho chúng ta một thứ gì đó về sự có mặt hay vắng mặt của những lỗi qua tập giá trị đầu vào cụ thể.

Thiết kế Test-case bằng phân lớp tương đương tiến hành theo 2 bước:

(1). Xác định các lớp tương đương

(2). Xác định các ca kiểm thử.

|  |  |  |
| --- | --- | --- |
| Điều kiện bên ngoài | Các lớp tương đương hợp lệ | Các lớp tương đương không hợp lệ |
|  |  |  |
|  |  |  |

*Hình 3.3 Một mẫu cho việc liệt kê các lớp tương đương.*

Với các lớp tương đương xác định được ở bước trên, bước thứ hai là sử dụng các lớp tương đương đó để xác định các ca kiểm thử. Quá trình này như sau:

1. Gán một số duy nhất cho mỗi lớp tương đương.

2. Cho đến khi tất cả các lớp tương đương hợp lệ được bao phủ bởi (hợp nhất thành) các ca kiểm thử, viết một ca kiểm thử mới bao phủ càng nhiều các lớp tương đương đó chưa được bao phủ càng tốt.

3. Cho đến khi các ca kiểm thử của bạn đã bao phủ tất cả các lớp tương đương không hợp lệ, viết một ca kiểm thử mà bao phủ một và chỉ một trong các lớp tương đương không hợp lệ chưa được bao phủ.

4. Lý do mà mỗi ca kiểm thử riêng bao phủ các trường hợp không hợp lệ là vì các kiểm tra đầu vào không đúng nào đó che giấu hoặc thay thế các kiểm tra đầu vào không đúng khác.

Nguyên tắc xác định lớp tương đương:

* Nếu điều kiện đầu vào định rõ giới hạn của một mảng hoặc 1 giá trị xác định thì chia vùng tương đương thành: 1 lớp tương đương hợp lệ, 2 lớp tương đương không hợp lệ, và 1 lớp đặc biệt (nếu có).

Nếu điều kiện đầu vào chỉ định là một tập giá trị hoặc xác định là 1 kiểu đúng sai thì chia vùng tương đương thành: 1 lớp tương đương hợp lệ, 1 lớp tương đương không hợp lệ, 1 lớp đặc biệt (nếu có).

* + 1. **Kỹ thuật phân tích giá trị biên**

Kinh nghiệm cho thấy các ca kiểm thử mà khảo sát tỷ mỷ các điều kiện biên có tỷ lệ phần trăm cao hơn các ca kiểm thử khác. Các điều kiện biên là những điều kiện mà các tình huống ngay tại, trên và dưới các cạnh của các lớp tương đương đầu vào và các lớp tương đương đầu ra. Phân tích các giá trị biên là phương pháp thiết kế ca kiểm thử bổ sung thêm cho phân lớp tương đương, nhưng khác với phân lớp tương đương ở 2 khía cạnh:

1. Phân tích giá trị biên không lựa chọn phần tử bất kỳ nào trong 1 lớp tương đương là điển hình, mà nó yêu cầu là 1 hay nhiều phần tử được lựa chọn như vậy mà mỗi cạnh của lớp tương đương đó chính là đối tượng kiểm tra.

2. Ngoài việc chỉ tập trung chú ý vào các trạng thái đầu vào (không gian đầu vào), các ca kiểm thử cũng nhận được bằng việc xem xét không gian kết quả (các lớp tương đương đầu ra).

Phân tích giá trị biên yêu cầu óc sáng tạo và lượng chuyên môn hóa nhất định và nó là một quá trình mang tính kinh nghiệm rất cao. Tuy nhiên, có một số quy tắc chung như sau:

1. Nếu một trạng thái đầu vào định rõ giới hạn của các giá trị, hãy viết các ca kiểm thử cho các giá trị cuối của giới hạn, và các ca kiểm thử đầu vào không hợp lệ cho các trường hợp vừa ra ngoài phạm vi.

2. Nếu một trạng thái đầu vào định rõ số lượng giá trị, hãy viết các ca kiểm thử cho con số lớn nhất và nhỏ nhất của các giá trị và một giá trị trên, một giá trị dưới những giá trị này.

3. Sử dụng quy tắc 1 cho mỗi trạng thái đầu vào. Ví dụ, nếu một chương trình tính toán sự khấu trừ FICA hàng tháng và nếu mức tối thiểu là 0.00$, và tối đa là 1,165.25$, hãy viết các ca kiểm thử mà khấu trừ 0.00$ và 1,165.25, khấu trừ âm và khấu trừ lớn hơn 1,165.25$. Chú ý là việc xem xét giới hạn của không gian kết quả là quan trọng vì không phải lúc nào các biên của miền đầu vào cũng mô tả cùng một tập sự kiện như biên của giới hạn đầu ra (ví dụ, xét chương trình con tính SIN). Ngoài ra, không phải lúc nào cũng có thể tạo ra một kết quả bên ngoài giới hạn đầu ra, nhưng tuy nhiên rất đáng để xem xét tiềm ẩn đó.

1. Sử dụng nguyên tắc 2 cho mỗi trạng thái đầu ra.

2. Nếu đầu vào hay đầu ra của một chương trình là tập được sắp thứ tự (ví dụ,1 file tuần tự hay 1 danh sách định tuyến hay 1 bảng) tập trung chú ý vào các phần tử đầu tiên và cuối cùng của tập hợp.

3. Sử dụng sự khéo léo để tìm các điều kiện biên.

Việc chọn lớp tương đương giá trị biên theo cách phân tích giá trị biên:

1. Nếu điều kiện vào là một miền giới hạn bởi a và b thì cần thiết kế các ca kiểm thử cho cả a và b, và cả trên, dưới a và b.

2. Nếu điều kiện vào đặc tả một số giá trị thì thiết kế các ca kiểm thử cho cả các số trên và dưới số nhỏ nhất và lớn nhất.

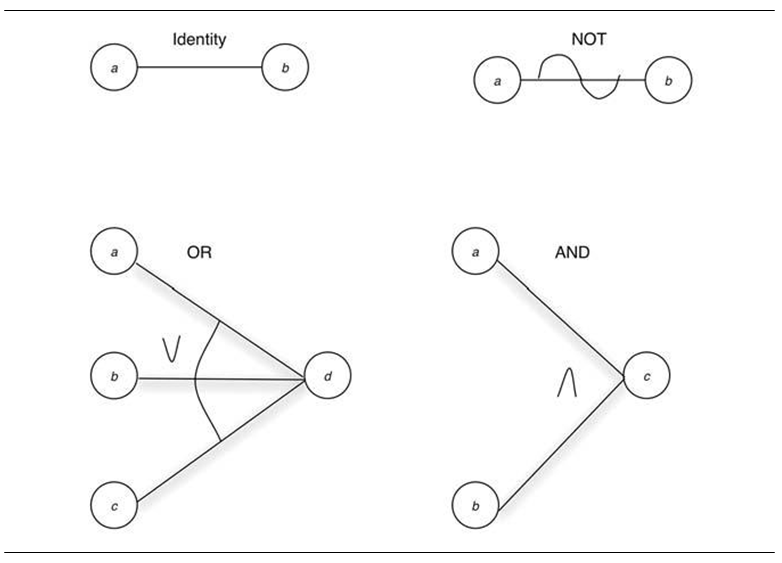
* + 1. **Kỹ thuật đồ thị nguyên nhân – kết quả**

Một yếu điểm của phân tích giá trị biên và phân lớp tương đương là chúng không khảo sát sự kết hợp của các trường hợp đầu vào. Việc kiểm tra sự kết hợp đầu vào không phải là một nhiệm vụ đơn giản bởi vì nếu bạn phân lớp tương đương các trạng thái đầu vào, thì số lượng sự kết hợp thường là rất lớn. Nếu bạn không có cách lựa chọn có hệ thống một tập con các trạng thái đầu vào, có lẽ bạn sẽ chọn ra một tập tùy hứng các điều kiện, điều này có thể dẫn tới việc kiểm thử không có hiệu quả.

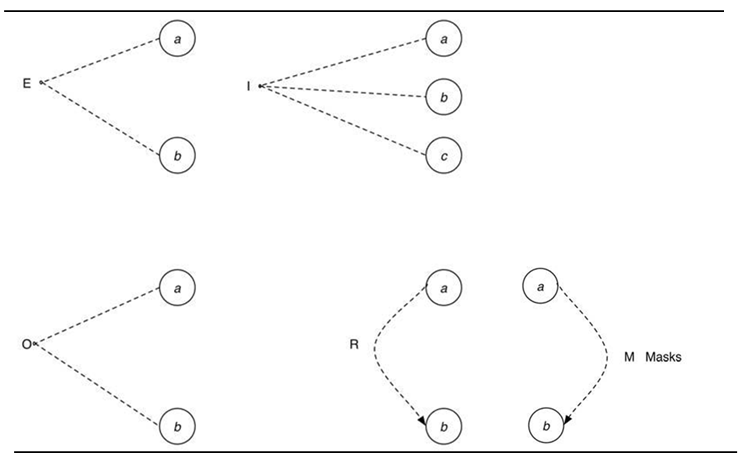
Đồ thị nguyên nhân – kết quả hỗ trợ trong việc lựa chọn một cách có hệ thống tập các ca kiểm thử có hiệu quả cao. Nó có tác động có lợi ảnh hưởng tới việc chỉ ra tình trạng chưa đầy đủ và nhập nhằng trong đặc tả. Nó cung cấp cả cách biểu diễn chính xác cho các điều kiện logic và hành động tương ứng

Quá trình dưới đây được sử dụng để xây dựng được các test – case:

1. Đặc tả được chia thành các phần có thể thực hiện được. Điều này là cần thiết bởi vì đồ thị nguyên nhân – kết quả trở nên khó sử dụng khi được sử dụng trên những đặc tả lớn.
2. Nguyên nhân và kết quả trong các đặc tả được nhận biết. Một nguyên nhân là một trạng thái đầu vào nhất định hay một lớp tương đương của các trạng thái đầu vào. Một kết quả là một trạng thái đầu ra hay 1 sự biến đổi hệ thống (kết quả còn lại mà 1 đầu vào có trạng thái của 1 chương trình hay hệ thống). Bạn nhận biết nguyên nhân và kết quả bằng việc đọc từng từ của đặc tả và gạch chân các từ hoặc cụm từ mô tả nguyên nhân và kết quả. Khi được nhận biết, mỗi nguyên nhân và kết quả được gán cho 1 số duy nhất.
3. Xây dựng đồ thị nguyên nhân – kết quả bằng cách phát triển và biến đổi nội dung ngữ nghĩa của đặc tả thành đồ thị Boolean nối giữa nguyên nhân và kết quả.
4. Đồ thị được được diễn giải với các ràng buộc mô tả những sự kết hợp của nguyên nhân và/hoặc kết quả là không thể vì các ràng buộc ngữ nghĩa và môi trường.
5. Bằng việc dò theo các điều kiện trạng thái trong đồ thị một cách cẩn thận, bạn chuyển đổi đồ thị thành một bảng quyết định mục vào giới hạn. Mỗi cột trong bảng mô tả một ca kiểm thử.
6. Các cột trong bảng quyết định được chuyển thành các ca kiểm thử. Ký hiệu cơ bản cho đồ thị được chỉ ra trong hình 3.4. Tưởng tượng mỗi nút có giá trị là 0 hoặc 1; 0 mô tả trạng thái vắng mặt và 1 mô tả trạng thái có mặt. Hàm đồng nhất nói là nếu a là 1 thì b là 1; ngược lại, b là 0. Hàm not là nói nếu a là 1 thì b là 0; ngược lại thì b là 1. Hàm or khẳng định rằng nếu a hoặc b hoặc c là 1, thì d là 1; ngược lại d là 0. Hàm and khẳng định nếu cả a và b là 1 thì c là 1; ngược lại c là 0. Hai hàm or và and được phép có số lượng đầu vào bất kỳ.

**

*Hình 3.4 Các kí hiệu đồ thị nguyên nhân – kết quả cơ bản*

**

*Hình 3.5 Các kí hiệu ràng buộc*

Bước tiếp theo là tạo bảng quyết định mục vào giới hạn – *limited-entry decision table*. Tương tự với các bảng quyết định, thì nguyên nhân chính là các điều kiện và kết quả chính là các hành động. Quy trình được sử dụng là như sau:

1. Chọn một kết quả để là trạng thái có mặt (1).
2. Lần ngược trở lại đồ thị, tìm tất cả những sự kết hợp của các nguyên nhân (đối tượng cho các rằng buộc) mà sẽ thiết lập kết quả này thành 1.
3. Tạo một cột trong bảng quyết định cho mỗi sự kết hợp nguyên nhân.
4. Với mỗi sự kết hợp, hãy quy định trạng thái của tất cả các kết quả khác và đặt chúng vào mỗi cột.

Trong khi biểu diễn bước 2, cần quan tâm các vấn đề sau:

1. Khi lần ngược trở lại qua một nút *or* mà đầu ra của nó là 1, không bao giờ thiết lập nhiều hơn 1 đầu vào cho nút *or* là 1 một cách đồng thời. Điều này được gọi là *path sensitizing* – *làm nhạy đường đi*. Mục tiêu của nó là để ngăn chặn dò lỗi thất bại vì một nguyên nhân che đi một nguyên nhân khác.
2. Khi lần ngược trở lại qua một nút *and* mà đầu ra của nó là 0, dĩ nhiên, phải liệt kê tất cả các sự kết hợp đầu vào dẫn tới đầu ra 0. Tuy nhiên, nếu bạn đang khảo sát trạng thái mà 1 đầu ra là 0 và một hay nhiều đầu ra khác là 1, thì không nhất thiết phải liệt kê tất cả các điều kiện mà dưới điều kiện đó các đầu vào khác có thể là 1.
3. Khi lần ngược trở lại qua một nút *and* mà đầu ra của nó là là 0, chỉ cần liệt kê 1 điều kiện trong đó tất cả đầu vào bằng 0. (Nếu nút *and*  ở chính giữa của đồ thị như vậy thì tất cả các đầu vào của nó xuất phát từ các nút trung gian khác, có thể có quá nhiều trạng thái mà trong trạng thái đó tất cả các đầu vào của nó bằng 0.)



*Hình 3.6 Những xem xét được sử dụng khi dò theo đồ thị*

* + 1. **Đoán lỗi – Error Guessing**

Một kỹ thuật thiết kế test-case khác là error guessing – đoán lỗi. Tester được đưa cho 1 chương trình đặc biệt, họ phỏng đoán, cả bằng trực giác và kinh nghiệm, các loại lỗi có thể và sau đó viết các ca kiểm thử để đưa ra các lỗi đó.

Thật khó để đưa ra một quy trình cho kỹ thuật đoán lỗi vì nó là một quy trình có tính trực giác cao và không thể dự đoán trước. Ý tưởng cơ bản là liệt kê một danh sách các lỗi có thể hay các trường hợp dễ xảy ra lỗi và sau đó viết các ca kiểm thử dựa trên danh sách đó. Một ý tưởng khác để xác định các ca kiểm thử có liên đới với các giả định mà lập trình viên có thể đã thực hiện khi đọc đặc tả (tức là, những thứ bị bỏ sót khỏi đặc tả, hoặc là do tình cờ, hoặc là vì người viết có cảm giác những đặc tả đó là rõ ràng). Nói cách khác, bạn liệt kê những trường hợp đặc biệt đó mà có thể đã bị bỏ sót khi chương trình được thiết kế.

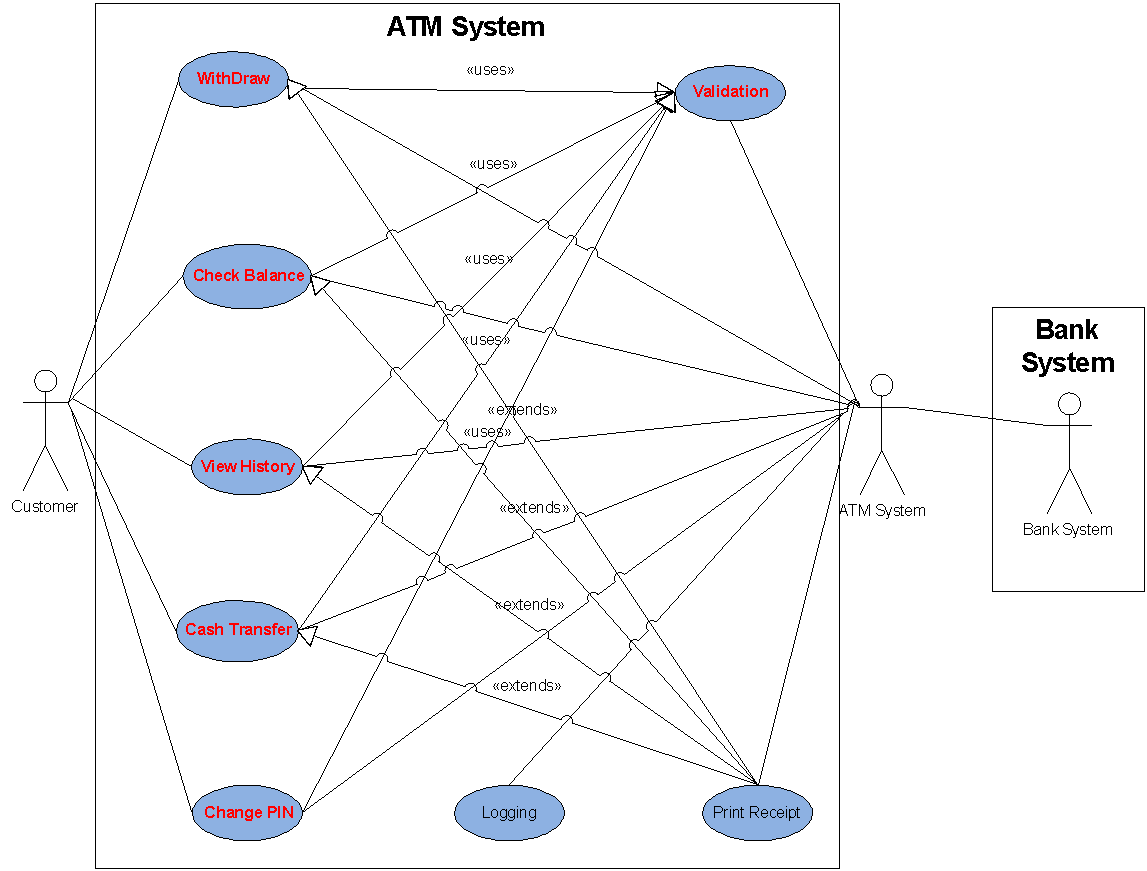
# **CHƯƠNG 3: KIỂM THỬ PHẦN MỀM ATM SIMULATOR**

1. **Giới thiệu phần mềm**

The ATM Simulator Application – Phần mềm mô phỏng máy rút tiền tự động, giúp mô phỏng cách thức hệ thống ATM hoạt động trong thực tế. Phần mềm giả lập nằm trong phạm vi bài tập của Fresher 11 – FPT giúp hiểu hơn cách xây dựng phần mềm từ .NET framework với ngôn ngữ C#.

1. **Phân tích thiết kế hệ thống**
   * 1. **Tổng quan**

Các use case được yêu cầu có trong ứng dụng giả lập máy ATM



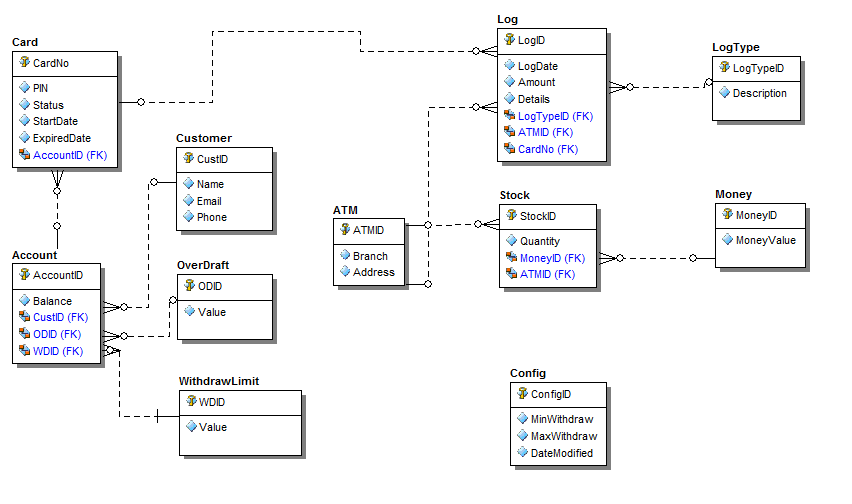
* + 1. **Mô tả - Use case description**

|  |  |  |
| --- | --- | --- |
| **UC** | **Name** | **Mô tả** |
| UC01 | Validation | Xác thực thẻ ATM và mã PIN khách hàng nhập vào |
| UC02 | Withdraw | Cho phép khách hàng rút tiến |
| UC03 | Check balance | Cho phép khách hàng kiểm tra số dư tài khoản của mình |
| UC04 | View history | Cho phép khách hàng xem giao dịch thành công của họ |
| UC05 | Cash Transfer | Cho phép khách hàng chuyển tiền sang tài khoản khác trong hệ thống ngân hàng được chấp nhận |
| UC06 | Change PIN | Cho phép khách hàng đổi mã PIN |
| UC07 | Logging | Hệ thống ghi nhật ký |

* + 1. **Use case & Actor mapping**

|  |  |  |  |
| --- | --- | --- | --- |
| **Actor**  **Usercase** |  | **Customer** | **ATM** |
| Validation |  |  | x |
| Withdraw |  | x | x |
| CashTransfer |  | x | x |
| CheckBalance |  | x | x |
| ChangePIN |  | x | x |
| ViewHistory |  | x | x |
| Logging |  |  | x |

* + 1. **Sơ đồ thực thể liên kết – Entity Relationship Diagram (ERD)**



|  |  |  |
| --- | --- | --- |
|  | Tên bảng | Mô tả |
| 1 | Customer | Danh sách tất cả các khách hàng |
| 2 | Account | Danh sách tất cả các tài khoản sử dụng trong hệ thống |
| 3 | Card | Danh sách tất cả các thẻ ATM sử dụng trong hệ thống |
| 4 | OverDraft | Số tiền mà một tài khoản có thể thấu chi (chi vượt quá số tiền có trên tài khoản) |
| 5 | WithdrawLimit | Số tiền tối đa mà một tài khoản có thể rút trong ngày |
| 6 | ATM | Danh sách các máy ATM trong hệ thống |
| 7 | Money | Loại tiền và giá trị |
| 8 | Stock | Loại tiền và số lượng mỗi loại có trong mỗi máy ATM |
| 9 | Log | Ghi tất cả các giao dịch của khách hàng |
| 10 | LogType | Loại log: Withdraw, transfer, check balance, change pin (rút tiền, chuyển tiền, kiểm tra tài khoản, đổi PIN) |
| 11 | Config | Lưu tất cả các cấu hình của hệ thống: số tiền rút tối thiểu, số tiền rút tối đa, số bản ghi trên các kết quả tìm kiếm |

* 1. **Chi tiết về thiết kế chức năng – Details function design**
     1. **Use case 01 – Validation**

Xác thực thẻ - Validate Card

|  |  |
| --- | --- |
| **Name** | Validate Card – Xác thực thẻ |
| **Mô tả** | Use case cho phép hệ thống ATM kiểm tra các thẻ mà người dùng đưa vào có hợp lệ hay không.. |
| **Actor** | ATM System |
| **Trigger** | Khi người dùng nhấp vào nút ‘Insert Card" ở màn hình chính. |
| **Pre-condition** | Thẻ đã được nhập vào máy ATM. |
| **Post-condition** | Nếu thẻ hợp lệ thì bước tiếp theo Xác thực (Authentication) được kích hoạt, đẩy thẻ ra nếu thẻ không hợp lệ. |

Xử lý chi tiết – Detail Processing

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (3) | BR01 | **Kiểm tra quy tắc:**   * IF <Card Reader> không thể đọc số thẻ THEN   + Set <<ShowedScreen>> = [Wrong Card Screen - Màn hình thẻ sai].   + Gửi yêu cầu để đẩy thẻ. |
| (7) | BR02 | **Kiểm tra quy tắc:**   * Khi người dùng lắp đúng thẻ vào ATM   + Nhận thông tin thẻ từ cơ sở dữ liệu với số thẻ như số đọc từ thẻ đã được khách hàng chèn vào. * IF Số thẻ không khớp với bất kỳ số thẻ nào trong cơ sở dữ liệu THEN   + Set <<ShowedScreen>> = [Invalid Card Screen - Màn hình thẻ không hợp lệ].   + Gửi yêu cầu để đẩy thẻ. |

Xác thực – Authentication

|  |  |
| --- | --- |
| **Name** | Authentication – Xác thực |
| **Mô tả** | Use case cho phép hệ thống ATM kiểm tra mã PIN mà khách hàng nhập vào có hợp lệ hay không. |
| **Actor** | ATM System |
| **Trigger** | Khi người dùng nhấp vào nút ‘Enter” hoặc nút “Submit” trên màn hình [Input PIN]. |
| **Pre-condition** | Thẻ đã được nhập vào máy ATM. |
| **Post-condition** | Khách hàng đã được xác thực thành công, hệ thống ATM hiển thị màn hình giao dịch được chọn. |

Detail Processing – Xử lý chi tiết

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (4) | BR01 | **Kiểm tra:**   * Kiểm tra PIN   + Lấy mã PIN của thẻ khách hàng từ cơ sở dữ liệu.   + So sánh mã PIN nhận được từ cơ sở dữ liệu với mã PIN khách hàng vừa nhập. * IF t khách hàng nhập mã PIN không khớp với mã PIN trong cơ sở dữ liệu của Thẻ khách hàng THEN   + Set <<ShowedScreen>> = [Wrong PIN Screen].   + Nhắc khách hàng nhập lại mã PIN. * IF khách hàng đã nhập sai ba lần PIN THEN   + Set <<ShowedScreen>> = [Block Card Screen – Màn hình bị khóa thẻ].   + Set <Status> of Card = “Block”.   + Nuốt thẻ. |

* + 1. **`Use case 02: Rút tiền – Withdraw Money**

|  |  |
| --- | --- |
| **Name** | Withdraw money – Rút tiền |
| **Mô tả** | Use case cho phép khách hàng rút tiền. |
| **Actor** | Customers |
| **Trigger** | Khi người dùng nhấp vào nút ‘Withdraw - Rút tiền’ trên màn hình. |
| **Pre-condition** | Sau khi Xác thực thành công, khách hàng nhập số tiền mà họ muốn rút. |
| **Post-condition** | Nhận tiền, viết nhật ký vào hệ thống, quyết định in biên lai. |

Detail Processing

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (6) | BR01 | **Xác thực nhập:**   * Hệ thống kiểm tra tiền còn lại: * IF enterCash (Tiền nhập) > MinValue (Giá trị tối thiểu) * OR enterCash (Tiền nhập) < MaxValue (Giá trị tối đa) * OR enterCash (Tiền nhập) mod 50.000 <> 0 THEN   + Set <<ShowedScreen>> = [Withdraw Failed Screen – Màn hình rút tiền thất bại]   + Return FALSE |
| (9) | BR02 | **Kiểm tra số dư:**   * IF enterCash < AccountBalance THEN   + Set <<Account Balance>> = <<Account Balance>> - enterCash   + Write Log. * ELSE   + Set <<ShowedScreen>> = [Withdraw Failed Screen – Màn hình rút tiền thất bại] |
| (11) | BR03 | **Dispenser money – Tính tiền trả:**   * Tính toán enterCash (Tiền nhập) khách hàng đã nhập và MoneyType (Loại tiền) và Value (Giá trị tiền), số lượng MoneyType (Loại tiền) còn trong ATM, Trả lại tiền mặt cho khách hàng. |

* + 1. **Use case 03: Kiểm tra số dư - Check Balance**

|  |  |
| --- | --- |
| **Name** | CheckBalance – Kiểm tra số dư |
| **Mô tả** | Use case cho phép khách hàng kiểm tra số dư tài khoản. |
| **Actor** | Customer |
| **Trigger** | Khi khách hàng chọn “Check balance – Kiểm tra số dư” trên màn hình chính |
| **Pre-condition** | Khách hàng đã được xác thực (Validation) vào ATM |
| **Post-condition** | Hệ thống ATM hiển thị số dư của Khách hàng. |

Detail Processing

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (2) | BR01 | **Hiển thị số dư:**   * + Lấy số dư của khách hàng từ cơ sở dữ liệu và hiển thị ra màn hình. |

* + 1. **Use case 04: Xem lịch sử giao dịch – View History**

|  |  |
| --- | --- |
| **Name** | View history - Xem lịch sử giao dịch |
| **Mô tả** | Use case cho phép khách hàng xem tất cả các giao dịch đã được thực hiện. |
| **Actor** | Authenticated Customer |
| **Trigger** | Khi người dùng ấn vào nút ‘View History – Xem lịch sử’ tại màn hình [Select Transaction – Chọn giao dịch]. |
| **Pre-condition** | Khách hàng đã được chứng thực (authenticate) thành công. |
| **Post-condition** | Tất cả các giao dịch đã được thực hiện bởi khách hàng sẽ hiển thị. |

Detail Processing

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (3) | BR01 | **Kiểm tra:**   * Tìm kiếm giao dịch/ nhật ký (transaction/ log)   + Set <<Criteria – Tiêu chí>> = [Filter Criteria – Tiêu chí lọc] khách hàng đã chọn.   + Hệ thống tìm kiếm từ cơ sở dữ liệu tất cả các giao dịch / nhật ký với:     - <<LogStatus>> = [Successful]     - <<LogDate>> = <<Criteria.Date>>     - <<LogType>> in ([Withdraw], [Transfer], [CheckBalance], [ChangePIN]) |
| (4) | BR02 | **Quy tắc phân trang:**   * Phân trang   + Set <<NumPerPage>> = [Number records per page - Số lượng hồ sơ trên mỗi trang] trong cấu hình hệ thống.   + Phân trang kết quả tìm kiếm theo <<NumPerPage>> |

* + 1. **Use case 05: Chuyển tiền – Cash Transfer**

|  |  |
| --- | --- |
| **Name** | Cash Transfer – Chuyển tiền |
| **Mô tả** | Use case cho phép khách hàng chuyển tiền mặt từ tài khoản của mình sang tài khoản khác |
| **Actor** | Customer |
| **Trigger** | Khi người dùng nhấp vào nút “Chuyển tiền – Cash Transfer” tại màn hình chính. |
| **Pre-condition** | Sau khi xác thực thành công, khách hàng nhập tài khoản và lượng tiền mặt muốn chuyển |
| **Post-condition** | Viết nhật ký vào hệ thống, quyết định in biên lai (Có / Không) |

Detail Processing

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (11) | BR01 | **Kiểm tra tiền**   * IF <Customer> chấp nhận số tiền vừa nhập THEN   + Lấy số dư của tài khoản này và so sánh với số tiền anh ấy / cô ấy muốn chuyển   + IF (Balance > amount) – (Số dư> số tiền) THEN     - Tiếp tục giao dịch   + ELSE     - Hiển thị “Your account not enough money to transfer – Tài khoản của bạn không đủ tiền để chuyển đi”     - Quay lại màn hình trước để nhập số tiền khác |
| (18) | BR02 | **Transfer Cash:**   * IF <Customer> bấm chấp nhận chuyển khoản THEN   + Lấy số tiền và số dư của tài khoản gửi và nhận tài khoản   + Cộng số tiền vào số dư tài khoản nhận và trừ vào số dư tài khoản gửi |

* + 1. **Use case 06: Thay đổi mã PIN – Change PIN**

|  |  |
| --- | --- |
| **Name** | Change PIN – Thay đổi mã pin |
| **Mô tả** | Use case cho phép khách hàng thay đổi mã PIN của mình |
| **Actor** | Customers |
| **Trigger** | Khi người dùng nhấp vào nút ‘Change PIN – Đổi mã PIN’ trên màn hình. |
| **Pre-condition** | Sau khi xác thực thành công |
| **Post-condition** | PIN của khách hàng sẽ thay đổi |

Detail Processing

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (4) & (9) | BR01 | **Kiểm tra cú pháp**   * Lấy mã PIN cũ của khách hàng và so sánh với mã PIN mới mà anh ấy / cô ấy vừa nhập * IF PIN cũ khớp với mã PIN mới THEN   + Hiển thị ‘New pin not allow! Please enter again - Pin mới không được chấp nhận! Vui lòng nhập lại’   + Hiển thị lại màn hình thay đổi PIN. |
| (15) | BR02 | **Đổi mã PIN**   * Lấy mã PIN mới và so sánh với mã PIN mới mà anh ấy / cô ấy nhập lại * IF 2 mã PIN mới khớp với nhau THEN   + Hiển thị “Your PIN changed - mã PIN của bạn đã thay đổi”   + Đẩy thẻ và ngừng giao dịch |

* + 1. **Use case 07: Ghi nhật ký – Logging**

|  |  |
| --- | --- |
| **Name** | Logging – Ghi nhật ký |
| **Mô tả** | Use case cho phép hệ thống ATM ghi nhật ký tất cả giao dịch đã được thực hiện bởi khách hàng |
| **Actor** | ATM System |
| **Trigger** | Khi người dùng hoàn thành bất kỳ giao dịch nào với hệ thống ATM |
| **Pre-condition** | Một giao dịch đã kết thúc |
| **Post-condition** | Bản ghi mới sẽ được chèn vào bảng Log trên cơ sở dữ liệu, lưu trữ thông tin về giao dịch của khách hàng: ngày, loại giao dịch, số tiền |

Detail Processing

|  |  |  |
| --- | --- | --- |
| Activity | BR Code | Mô tả |
| (3) | BR01 | * Insert new Log   + Set <<LogDate>> = [Current Time]   + Set <<LogType>> = [Transaction Type]   + Set <<ATM>> = [Current ATM Machine]   + Set <<Card>> = [Current Card]   + Set <<Amount>> = [Amount] |

1. **Lập kế hoạch kiểm thử**
   1. **Mục đích**

Tài liệu kế hoạch kiểm thử cho dự án “Phần mềm mô phỏng máy ATM” được dùng để:

• Xác định những thông tin dự án và các phần dự án cần được kiểm thử.

• Liệt kê những yêu cầu kiểm thử (Test Requirements)

• Nêu ra những phương pháp, chiến lược kiểm thử nên sử dụng

• Xác định nguồn lực cần.

• Nêu rõ các chức năng test và các chức năng không test

• Liệt kê môi trường test

* 1. **Thông tin chung**
  2. **Phạm vi test**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Feature** | **Functional Testing** | **Itegration Testing** | **Security & Access Control Tesing** |
| 1 | Xác thực thẻ – Validate Card  Xác thực – Authentication | 2 man days | 0.5 man days | 0.5 man days |
| 2 | Rút tiền – Withdraw Money | 3 man days |
| 3 | Kiểm tra số dư - Check Balance | 3 man days |
| 4 | Xem lịch sử giao dịch – View History | 1 man days |
| 5 | Chuyển tiền – Cash Transfer | 3 man days |
| 6 | Thay đổi mã PIN – Change PIN | 3 man days |
| 7 | Ghi nhật ký – Logging | 2 man days |

* 1. **Các công cụ hỗ trợ**

|  |  |  |  |
| --- | --- | --- | --- |
| Action | Tools | Supplier /  Self-constrcution | Version |
| Quản lý họat động kiểm thử | Excel | Microsoft | 365 |
| Kiểm soát lỗi | Excel | Microsoft | 365 |
| Các công cụ quản trị CSDL | MS SQL Server | Microsoft | 2014 |
| Quản lý tiến độ dự án | Microsoft Project | Microsoft | 2019 |

* 1. **Tài liệu liên quan**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Tài liệu | Nguồn | Mô tả |
| 1 | SWD Fresher11 ATM Sytem - Team HLD | Đã được cung cấp | Tài liệu mô tả hệ thống |
| 2 | Test\_plan\_ATMSimulatorApplication | Đã được cung cấp | Tài liệu kế hoạch kiểm thử |
| 3 |  |  |  |
|  |  |  |  |

1. **Thực hiện kiểm thử**
   1. **Kiểm thử Use case 01: Validation**

Mô tả điều kiện:

* + Validate card:
    - Nhập mã thẻ từ màn hình
    - Mã thẻ phải khớp với dữ liệu trong CSDL
  + Validate PIN:
    - Mã pin gồm 6 chữ số đi theo thẻ.
    - Mã pin phải hợp lệ với CSDL, nhập sai mã PIN 3 lần sẽ bị khoá thẻ.

Bảng phân vùng tương đương:

|  |  |  |
| --- | --- | --- |
| **Đầu vào** | **Các lớp tương đương hợp lệ** | **Các lớp tương đương không hợp lệ** |
| CardNo | Mã thẻ hợp lệ với CSDL | Mã thẻ không nằm trong CSDL |
| Để trống mã thẻ |
| PIN | Mã pin 6 ký tự hợp lệ với mã PIN của thẻ | Mã PIN <6 hoặc > 6 ký tự  Mã PIN không trùng với mã PIN của thẻ |
| Để trống mã PIN |

Dữ liệu ban đầu:

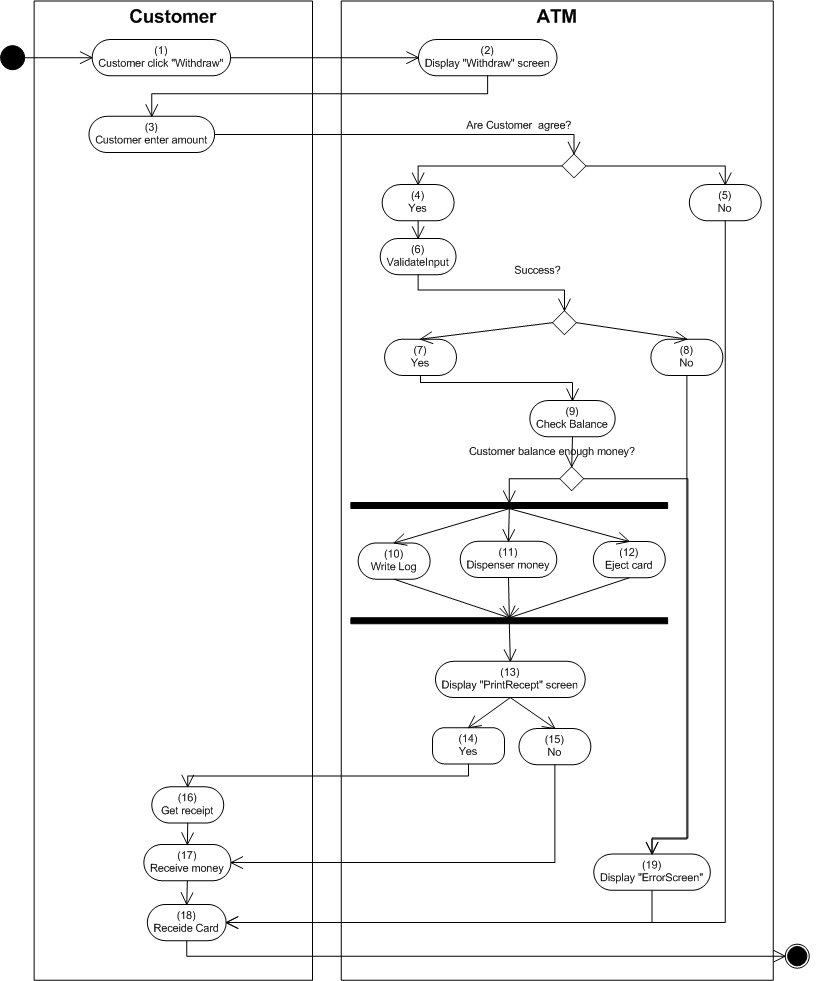
|  |  |  |
| --- | --- | --- |
| **CardNo** | **Status** | **PIN** |
| 1500150015001 | normal | 123456 |
| 1500150015002 | normal | 123456 |
| 1500150015003 | normal | 123456 |
| 1500220196521 | block | 456789 |
| 1500220150000 | Không tồn tại trong CSDL | |

Thực hiện kiểm thử

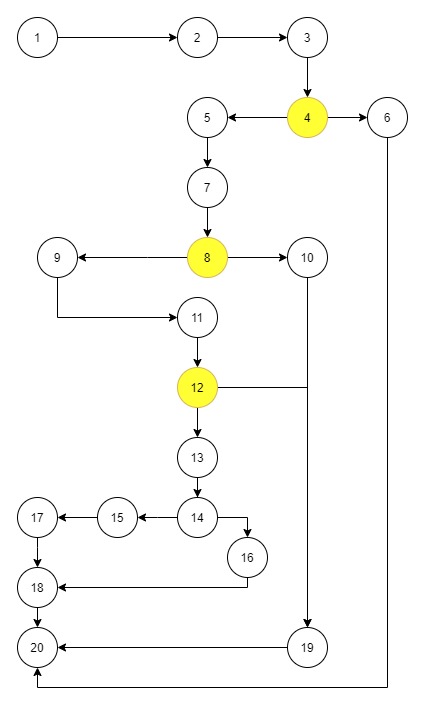
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Function** | **Test Case** | **Decription** | **Step Name** | **Step decription** | **Test Data** | **Expected result** | **Output** |
| Validate Card | 1 | Xác thực số thẻ hợp lệ trên hệ thống. | Step 1 | Nhập mã thẻ hợp lệ vào hệ thống ATM | CardNo: 1500150015001 | Hệ thống xác thực mã thẻ trong Database và chuyển màn hình nhập mã PIN | Passed |
| 2 | Xác thực số thẻ không hợp lệ | Step 1 | Nhập số thẻ không được lưu trên hệ thống | CardNo: 1500220150000 | Hiển thị lỗi "Your card number is not correct" và yêu cầu nhập lại mã thẻ | Passed |
| 3 | Để trống số thẻ | Step 1 | Bỏ trống | CardNo: null | Hiển thị lỗi "Your card number is not correct" và yêu cầu nhập lại mã thẻ | Passed |
| Authentication | 4 | Xác thực mã pin hợp lệ 6 số | Step 1 | Nhập mã thẻ hợp lệ vào hệ thống ATM | CardNo: 1500150015001 | Hệ thống xác thực mã thẻ trong Database và chuyển màn hình nhập mã PIN | Passed |
| Step 2 | Nhập vào mã pin hợp lệ của thẻ | PIN: 123456 | Hệ thống chuyển tới màn hình giao dịch | Passed |
| 5 | Xác thực mã pin sai | Step 1 | Nhập mã thẻ hợp lệ vào hệ thống ATM | CardNo: 1500150015002 | Hệ thống xác thực mã thẻ trong Database và chuyển màn hình nhập mã PIN | Passed |
| Step 2 | Nhập vào mã pin không hợp lệ với thẻ | PIN: 112345 | Hiển thị lỗi "Your PIN is not correct" và yêu cầu nhập lại mã PIN | Passed |
| 6 | Để trống mã PIN | Step 1 | Nhập mã thẻ hợp lệ vào hệ thống ATM | CardNo: 1500150015002 | Hệ thống xác thực mã thẻ trong Database và chuyển màn hình nhập mã PIN | Passed |
| Step 2 | Bỏ trống mã PIN | PIN: null | Hiển thị lỗi "Your PIN is not correct" và yêu cầu nhập lại mã PIN | Passed |
| 7 | 3 lần nhập sai mã PIN tài khoản sẽ bị khoá | Step 1 | Nhập mã thẻ hợp lệ vào hệ thống ATM | CardNo: 1500150015003 | Hệ thống xác thực mã thẻ trong Database và chuyển màn hình nhập mã PIN | Passed |
| Step 2 | Bỏ trống mã PIN | PIN: null | Hiển thị lỗi "Your PIN is not correct" và yêu cầu nhập lại mã PIN | Passed |
| Step 3 | Nhập vào mã pin không hợp lệ với thẻ | PIN: 111111 | Hiển thị lỗi "Your PIN is not correct" và yêu cầu nhập lại mã PIN | Passed |
| Step 4 | Nhập vào mã pin không hợp lệ với thẻ | PIN: 111122 | Hiển thị lỗi "Your PIN is not correct" và khoá thẻ | Passed |
| 8 | Xác thực thẻ bị khoá | Step 1 | Nhập mã thẻ hợp lệ vào hệ thống ATM | CardNo: 1500220196521 | Hệ thống xác thực mã thẻ trong Database và chuyển màn hình nhập mã PIN | Passed |
| Step 2 | Nhập mã PIN hợp lệ/ không hợp lệ | PIN: 456789 | Hiển thị lỗi "Your card has been locked" và yêu cầu nhập lại mã PIN | Passed |

* 1. **Kiểm thử Use case 02: Rút tiền – Withdraw Money**

Lưu đồ thuật toán hoạt động rút tiền



Sơ đồ đồ thị dòng:



Độ phức tạp của chu trinh: C = E – N + 2 = 23 – 20 + 2 = 5

Các đường đi độc lập:

* + 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 11 – 12 – 13 – 14 – 15 – 17 – 18 – 20
  + 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 11 – 12 – 13 – 14 – 16 – 18 – 20
  + 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 11 – 12 – 19 – 20
  + 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 10 – 19 – 20
  + 1 – 2 – 3 – 4 – 6 – 20.

Xây dựng test case (áp dụng phương pháp bảng quyết định)

Lập bảng quyết định:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Luật**  **1** | **Luật 2** | **Luật 3** | **Luật 4** | **Luật 5** | **Luật 6** | **Luật 7** | **Luật 8** | **Luật 9** |
| **Điều kiện** | Giá trị tối đa > Giá trị nhập > Giá trị tối thiểu | t | t | t | t | f | f | f | f | - |
| Giá trị nhập là bội của 50000 | t | t | f | f | t | t | f | f | - |
| Bỏ trống ô nhập | - | - | - | - | - | - | - | - | t |
| Giá trị nhập < Số dư tài khoản | t | f | t | f | t | f | t | f | - |
| **Hành động** | Trả lại thẻ | Y | y | y | y | y | y | y | y | Y |
| Rút tiền thành công và ghi nhật ký | y | n | n | n | n | n | n | n | n |

Xây dụng ca kiểm thử dựa trên bảng quyết định:

|  |  |  |
| --- | --- | --- |
| **Test case** | **Đầu vào** | **Đầu ra mong đợi** |
| 1 | Giá trị nhập hợp lệ  Giá trị nhập nhỏ hơn số dư tài khoản | Hiển thị màn hình lựa chọn nhận hóa đơn, rút tiền thành công |
| 2 | Giá trị nhập hợp lệ  Giá trị nhập lớn hơn số dư tài khoản | Hiển thị thông báo rút tiền không thành công |
| 3 | Giá trị nhập không là bội của 50000  Giá trị nhập nhỏ hơn số dư dài tài khoản | Hiển thị thông báo rút tiền không thành công |
| 4 | Giá trị nhập không là bội của 50000  Giá trị nhập lớn hơn số dư dài tài khoản | Hiển thị thông báo rút tiền không thành công |
| 5 | Giá trị nhập không thỏa mãn điều kiện  Giá trị nhập nhỏ hơn số dư dài tài khoản | Hiển thị thông báo rút tiền không thành công |
| 6 | Giá trị nhập không thỏa mãn điều kiện  Giá trị nhập lớn hơn số dư dài tài khoản | Hiển thị thông báo rút tiền không thành công |
| 7 | Giá trị nhập không thỏa mãn điều kiện và không là bội của 50000  Giá trị nhập nhỏ hơn số dư dài tài khoản | Hiển thị thông báo rút tiền không thành công |
| 8 | Giá trị nhập không thỏa mãn điều kiện và không là bội của 50000  Giá trị nhập lớn hơn số dư dài tài khoản | Hiển thị thông báo rút tiền không thành công |
| 9 | Bỏ trống phần nhập giá trị | Hiển thị thông báo rút tiền không thành công |

Thực hiện kiểm thử Use case Rút tiền – Withdraw Money

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Function** | **Test Case Name** | **Pre** | **Step** | **Expected Output** | **Actual Output** |
| WM01 | Rút tiền | Rút tiền thành công | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo “Do you want print receipt?”.  Trở vể màn hình chọn chức năng. | Hiển thị thông báo “Do you want print receipt?”.  Trở vể màn hình chọn chức năng. |
| WM02 | Rút tiền | Giá trị nhập lớn hơn số dư | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo Unsuccessful Withdrawal | Hiển thị thông báo Unsuccessful Withdrawal |
| WM03 | Rút tiền | Giá trị nhập không là bội 50000 | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo Unsuccessful Withdrawal | Hiển thị thông báo Unsuccessful Withdrawal |
| WM04 | Rút tiền | Giá trị nhập không là bội 50000,  Giá trị nhập lớn hơn số dư | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo Unsuccessful Withdrawal | Hiển thị thông báo Unsuccessful Withdrawal |
| WM05 | Rút tiền | Giá trị nhập không thỏa mãn | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo Unsuccessful Withdrawal | Hiển thị thông báo Unsuccessful Withdrawal |
| WM06 | Rút tiền | Giá trị nhập không thỏa mãn,  Giá trị nhập lớn hơn số dư | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo Unsuccessful Withdrawal | Hiển thị thông báo Unsuccessful Withdrawal |
| WM07 | Rút tiền | Giá trị nhập không thỏa mãn và không là bội của 50000 | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo Unsuccessful Withdrawal | Hiển thị thông báo Unsuccessful Withdrawal |
| WM08 | Rút tiền | Giá trị nhập không thỏa mãn và không là bội của 50000  Giá trị nhập lớn hơn số dư | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo Unsuccessful Withdrawal | Hiển thị thông báo Unsuccessful Withdrawal |
| WM09 | Rút tiền | Không nhập giá trị | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "withdraw"  2, Chọn số tiền muốn rút  3, Nhập vào số tiền muốn rút  4, Chọn OK | Hiển thị thông báo yêu cầu người dùng nhập giá trị | Báo lỗi chương trình: **System.FormatException:** 'Input string was not in a correct format.' |

Bộ dữ liệu kiểm thử

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Input value | Message | Test result | Actual result |
| WM01 | 200000 | “Do you want print receipt?”.  OK or Cancel | Passed |  |
| WM02 | 9000000000 | Unsuccessful Withdrawal | Passed |  |
| WM03 | 123123 | Unsuccessful Withdrawal | Passed |  |
| WM04 | 456456000 | Unsuccessful Withdrawal | Passed |  |
| WM05 | 10000 | Unsuccessful Withdrawal | Passed |  |
| WM06 | 1000000000 | Unsuccessful Withdrawal | Passed |  |
| WM07 | 5000 | Unsuccessful Withdrawal | Passed |  |
| WM08 | 123123000 | Unsuccessful Withdrawal | Passed |  |
| WM09 |  | Have not entered the amount to withdraw | Failed | **System.FormatException:** 'Input string was not in a correct format.' |

Các test case đã thực hiện: 100%

Thành công: 88.89%

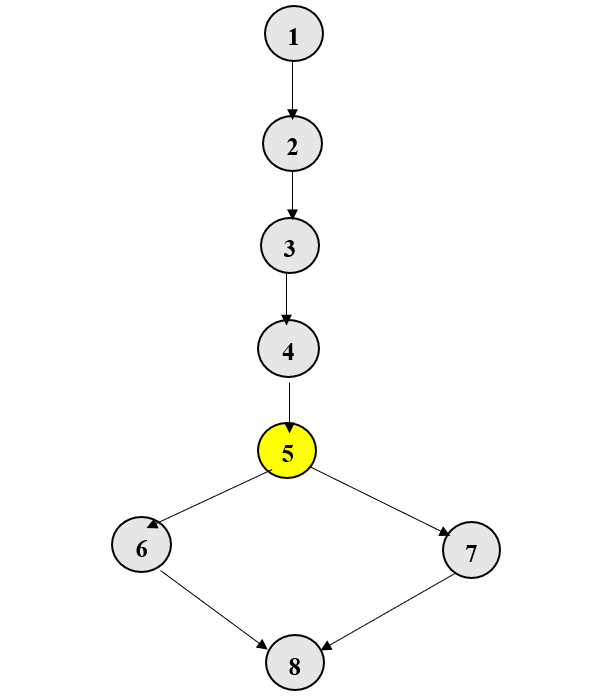
Thất bại: 11.11%

* 2. **Kiểm thử Use case 03: Kiểm tra số dư – Check Balance**
  3. **Kiểm thử Use case 04: Xem lịch sử giao dịch – View History**

Lưu đồ thuật toán



Đồ thị dòng



Độ phức tạp của chu trình: C = 8 – 8 + 1 =1.

Các đường độc lập:

* 1-2-3-4-5-6-8.
* 1-2-3-4-5-7-8.

các trường hợp kiểm thử.

|  |  |  |
| --- | --- | --- |
| TC | Đầu vào | Đầu ra |
| 1 | Chọn”view history ” và “time criteria” | Hiển thị các giao dịch đã thực hiện |
|  |  |  |

Thực hiện kiểm thử Use case View history

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Function** | **Test Case Name** | **Pre** | **Step** | **Expected Output** | **Actual Output** |
| T-01 | Xem lịch sử | xem thành công | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "view history"  2, hienr thị màn hình chọn “time criteria”  3,hiển thị màn hình với các giao dịch đã thành công  4, chọn “pre” để trở về home or “next” để thoát và trả thẻ | Hiển thị màn hình để nhập mật khẩu mới.  Hiển thị màn hình nhập lại mật khẩu mới để xác thực  Trở vể màn hình chọn chức năng. | Hiển thị các giáo dịch đã thực hiện thành công.  Chọn “pre” or “home” không hiển thị như mong muốn => chưa thành công |

* 2. **Kiểm thử Use case 05: Chuyển tiền – Cast Transfer**

Lưu đồ thuật toán chức năng chuyển tiền:



Mô tả yêu cầu: Thực hiện một giao dịch chuyển khoản từ tài khoản A sang tài khoản B với số tiền Amount. (tài khoản cho cá nhân)

Sau khi chuyển khoản thành công thì trên màn hình hiển thị thông báo "Transfer success" và “Do you want to receive receipt” và ghi thông tin chuyển khoản vào table TransferTransaction.

Chuyển khoản trong hệ thống Agribank, hạn mức tối thiểu là 10000 VND, hạn mức tối đa trên một giao dịch là 100000000 VNĐ, hạn mức giao dịch tối đa trong một ngày là 100000000 VNĐ.

Điều kiện:

* Đăng nhập với thông tin đã đăng ký.
* Vào màn hình chuyển khoản.
* Nhập thông tin chuyển khoản (Số tài khoản muốn chuyển, số tiền, nội dung).
* Kích nút “Chuyển khoản”.

Xây dựng test case (áp dụng phương pháp phân hoạch tương đương và phân tích giá trị biên)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | L1 | L2 | L3 | L4 | L5 | L6 | L7 | L8 |
| Điều kiện | Đồng ý với điều khoản chuyển tiền. | N |  |  |  |  |  |  |  |
| Nhập đúng mã khách hàng nhận tiền. | - | N |  |  |  |  |  |  |
| Bỏ trống số tiền. |  |  | Y |  |  |  |  |  |
| Nhập số tiền < 10k. | - | - | - | Y |  |  |  |  |
| Nhập số tiền >= 10k và <= 100tr. | - | - | - | - | Y | Y | Y | - |
| Nhập số tiền > 100tr. | - | - | - | - | - | - | - | Y |
| Nhập số tiền < = Số dư TK | - | - | - | - | Y | N | Y |  |
| Xác nhận chuyển tiền | - | - | - | - | N | Y | Y | Y |
| Hành động | Trở về màn hình chính | Y |  |  |  | Y |  |  |  |
| Yêu cầu nhập lại mã khách hàng nhận tiền |  | Y |  |  |  |  |  |  |
| Yêu cầu nhập lại số tiền |  |  | Y | Y |  | Y |  | Y |
| Chuyển tiền thành công và hỏi in hoá đơn |  |  |  |  |  |  | Y |  |

|  |  |  |
| --- | --- | --- |
| Test case | Đầu vào | Đầu ra mong đợi |
| 1 | Từ chối với điều khoản | Quay về màn hình chính |
| 2 | Đồng ý với điều khoản  Nhập sai mã người nhận | Yêu cầu nhập lại mã khách hàng |
| 3 | Đồng ý với điều khoản  Nhập đúng mã người nhận  Bỏ trống số tiền | Yêu cầu nhập lại số tiền |
| 4 | Đồng ý với điều khoản  Nhập đúng mã người nhận  Nhập số tiền < 10k | Yêu cầu nhập lại số tiền |
| 5 | Đồng ý với điều khoản  Nhập đúng mã người nhận  Nhập số tiền > 100tr | Yêu cầu nhập lại số tiền |
| 6 | Đồng ý với điều khoản  Nhập đúng mã người nhận  Nhập số tiền >= 10k và <= 100tr và nhỏ hơn hoặc bằng số dư tài khoản.  Từ chối xác nhận | Quay về màn hình chính |
| 7 | Đồng ý với điều khoản  Nhập đúng mã người nhận  Nhập số tiền >= 10k và <= 100tr và lớn hơn hoặc bằng số dư tài khoản. | Yêu cầu nhập lại số tiền |
| 8 | Đồng ý với điều khoản  Nhập đúng mã người nhận  Nhập số tiền >= 10k và <= 100tr và nhỏ hơn số dư tài khoản.  Đồng ý xác nhận | Chuyển tiền thành công và hỏi in biên lai |

Thực hiện kiểm thử Use case Chuyển tiền – Transfer

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Function | Test case Name | Pre | Step | Expected Output | Actual Output |
| TM01 | Chuyển tiền | Từ chối với điều khoản chuyển tiền | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank” chọn “Reject” | Quay về màn hình chính | Quay về màn hình chính |
| TM02 | Chuyển tiền | Sai mã TK người nhận | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank” chọn “Accept”. 3. Nhập số tài khoản muốn chuyển tiền đến. | Hiển thị thông báo “Your account not exist!”. “Re-enter account you want to transfer” | Hiển thị thông báo “Your account not exist!”. “Re-enter account you want to transfer” |
| TM03 | Chuyển tiền | Không nhập số tiền chuyển | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank?” chọn “Accept”. 3. Nhập số tài khoản muốn chuyển tiền đến. 4. Nhập số tiền cần chuyển đi. | Hiển thị thông báo “Enter amount you want to transfer”. | Hiển thị thông báo “Enter amount you want to transfer”. |
| TM04 | Chuyển tiền | Số tiền chuyển nhỏ hơn 10000 | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank?” chọn “Accept”. 3. Nhập số tài khoản muốn chuyển tiền đến. 4. Nhập số tiền cần chuyển đi. | Hiển thị thông báo “The amount to be transferred must be greater than or equal to 10000 VND”. | Hiển thị thông báo “The amount to be transferred must be greater than or equal to 10000 VND”. |
| TM05 | Chuyển tiền | Số tiền chuyển lớn hơn 100 triệu | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank?” chọn “Accept”. 3. Nhập số tài khoản muốn chuyển tiền đến. 4. Nhập số tiền cần chuyển đi. | Hiển thị thông báo “The amount to be transferred must be less than or equal to 100000000 VND” | Hiển thị thông báo “The amount to be transferred must be less than or equal to 100000000 VND”. |
| TM06 | Chuyển tiền | Từ chối xác nhận chuyển | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank?” chọn “Accept”. 3. Nhập số tài khoản muốn chuyển tiền đến. 4. Nhập số tiền cần chuyển đi. 5. Từ chối xác nhận chuyển “Reject”. | Quay về màn hình nhập số tiền muốn chuyển. | Quay về màn hình nhập số tiền muốn chuyển. |
| TM07 | Chuyển tiền | Không đủ số dư cần chuyển tiền | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank?” chọn “Accept”. 3. Nhập số tài khoản muốn chuyển tiền đến. 4. Nhập số tiền cần chuyển đi. 5. Xác nhận chuyển tiền. | Hiển thị thông báo “Your account isn’t enough to transfer. Re-enter amount you want to transfer  ”. | Hiển thị thông báo “Your account isn’t enough to transfer. Re-enter amount you want to transfer  ”. |
| TM08 | Chuyển tiền | Chuyển tiền thành công | Xác thực tài khoản thành công | 1. Tại màn hình chức năng chọn “Transfer”. 2. Tại màn hình “Rules of transfer cash in Agribank?” chọn “Accept”. 3. Nhập số tài khoản muốn chuyển tiền đến. 4. Nhập số tiền cần chuyển đi. 5. Xác nhận chuyển tiền | Hiển thị thông báo “Transfer Success !” | Hiển thị thông báo “Transfer Success !” |

Bộ dữ liệu kiểm thử

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Input value** | | | | **Message** | **Test result** | **Actual result** |
| **Confirm** | **Card number start** | **Accout Receive ID** | **Amout** |
| TM01 | Click “Reject” |  |  |  | “Rules of transfer cash in Agribank?”  Quay về màn hình chính | Passed |  |
| TM02 | Click “Accept” | 1500220195722  PIN: 654321 | 123456789 |  | “Your account not exist!”. “Re-enter account you want to transfer”. | Passed |  |
| TM03 | Click “Accept” | 1500220195722  PIN: 654321 | 103524801339 |  | “Enter amount you want to transfer”. | Failed | **System.Format**  **Exception:** 'Input string was not in a correct format.' |
| TM04 | Click “Accept” | 1500220195722  PIN: 654321 | 103524801339 | 9000 | “Số tiền cần chuyển phải lớn hơn hoặc bằng 10000 VNĐ”. | Failed | Transfer Success ! |
| TM05 | Click “Accept” | 1500220195722  PIN: 654321 | 103524801339 | 1100000000 | “Số tiền cần chuyển phải nhỏ hơn hoặc bằng 100000000 VNĐ”. | Failed | Your account isn’t enough to transfer. Re-enter amount you want to transfer |
| TM06 | Click “Accept” | 1500220195722  PIN: 654321 | 103524801339 | 500000 | Quay về màn hình nhập số tiền muốn chuyển. | Passed |  |
| TM07 | Click “Accept” | 1500220195722  PIN: 654321 | 103524801339 | 50000000 | “Your account isn’t enough to transfer”. | Passed |  |
| TM08 | Click “Accept” | 1500220195722  PIN: 654321 | 103524801339 | 200000 | “Transfer success. Would you like another transaction?” | Passed |  |

Các test case đã thực hiện: 100%

Thành công: 62.5%

Thất bại: 37.5%

* 1. **Kiểm thử Use case 06: Thay đổi mã PIN – Change PIN**

Lưu đồ thuật toán



Lập bảng quyết định:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Luật 1** | **Luật 2** | **Luật 3** | **Luật4** | **Luật 5** | **Luật 6** | **Luật 7** | **Luật 8** |
| **Điều kiện** | Nhập 6 chữ số | **Y** | **Y** | **N** | **N** |  |  |  |  |
| Nhập lại mật khẩu | **Y** | **N** | **Y** | **N** |  |  |  |  |
| Nhập mật khẩu cũ |  |  |  |  | **Y** | **Y** |  |  |
| Nhập < 6 chứ số |  |  |  |  |  |  | **Y** | **Y** |
| **Hành động** | Đổi thành công | **Y** |  |  |  |  |  |  |  |
| Nhập lại |  | **N** | **N** | **N** | **N** | **N** | **N** | **N** |

Xây dựng ca kiểm thử

|  |  |  |
| --- | --- | --- |
| **TC** | **Đầu vào** | **Đầu ra** |
| 1 | Nhập mật khẩu có 6 chữ số,nhập lại MK đúng | Bạn đã đổi mã PIN thành công |
| 2 | Nhập mật khẩu 6 chữ số, nhập lại MK sai | Nhập lại |
| 3 | Nhập mật khẩu trùng với MK cũ | Nhập lại |
| 4 | Nhập mật khẩu ít hơn 6 chữ số | Nhập lại |
|  |  |  |

**Thực hiện kiểm thử Use case Change PIN**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Function** | **Test Case Name** | **Pre** | **Step** | **Expected Output** | **Actual Output** |
| T-01 | Đổi PIN | Đổi mã PIN thành công | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "change PIN"  2, Nhập mật khẩu mới  3, chọn enter  4, nhập lại mật khẩu mới để xác thực  5, chọn enter | Hiển thị màn hình để nhập mật khẩu mới.  Hiển thị màn hình nhập lại mật khẩu mới để xác thực  Trở vể màn hình chọn chức năng. | Hiển thị thông báo “your change PIN?”.  Trở vể màn hình chọn chức năng. |
| T-02 | Đổi PIN | Nhập mật khẩu nhiều hơn 6 chữ số | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "change PIN"  2, nhập mật khẩu muốn đổi  3, chọn enter | Hiển thị màn hình để nhập mật khẩu mới.  Hiển thị thông báo “ please enter again” | Hiển thị thông báo “ please enter again” |
| T-03 | Đổi PIN | Nhập mật khẩu cũ | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "change PIN"  2, Nhập MK muốn đổi  3, chọn enter | Hiển thị màn hình để nhập mật khẩu mới.  Hiển thị thông báo “ please enter again” | Hiển thị thông báo “new pin not allow! please enter again” |
| T-04 | Đổi PIN | Nhập xác thực mật khẩu sai | Xác thực tài khoản thành công | 1, Tại màn hình chức năng chọn "change PIN"  2, nhập mật khẩu muốn đổi  3, chọn enter  4, nhập xác thực lại mật khẩu mới  4, chọn enter | Hiển thị màn hình để nhập mật khẩu mới.  Hiển thị màn hình nhập lại mật khẩu mới để xác thực | Hiển thị thông báo “please enter again” |
|  |  |  |  |  |  |  |

* 1. **Kiểm thử Use case 07: Ghi nhật ký – Logging**

Lưu đồ thuật toán



Hiển thị đầy đủ thời gian người dùng thực hiện hành động, kiểu hành động, ATM, Card thực hiện hành động và số lượng tiền khi thực hiện giao dịch

**KẾT LUẬN**

***1. Kết quả đạt được***

Qua quá trình học tập và tìm hiểu kiểm thử phần mềm, chúng em đã đạt được một số kết quả cơ bản sau:

* Nắm được các kiến thức, lý thuyết cơ bản về:
  + Kiểm thử phần mềm
  + Các phương pháp kiểm thử
  + Các chiến lược kiểm thử
  + Các cấp độ kiểm thử
* Áp dụng được kiến thức kiểm thử để kiểm thử phần mềm giả lập máy rút tiền tự động.

***2. Hạn chế***

Trong quá trình thực hiện đề tài, dưới sự hợp tác của các thành viên trong nhóm. Tuy đề tài đã hoàn thành nhưng vẫn có một số hạn chế:

* Phần mềm kiểm thử không phải dạng điển hình cho việc kiểm thử.
* Các test case có thể chưa đầy đủ.
* Bản kế hoạch kiểm thử chưa chi tiết, chưa bám sát kế hoạch kiểm thử đã đề ra.

***3. Lời cảm ơn***

Nội dung báo cáo được tham khảo từ nhiều nguồn và sự hướng dẫn từ giảng viên, song không tránh khỏi sự thiếu sót. Rất mong nhận được nhận xét từ các thầy cô.

Để hoàn thành bài tập lớn này, chúng em xin bày tỏ lòng biết ơn sâu sắc đến thầy Nguyễn Thái Cường đã tận tình hướng dẫn, định hướng đề tài cho bọn em trong suốt quá trình nghiên cứu và thực hiện đồ án. Em xin kính chúc thầy cùng quý thầy, cô khoa CNTT dồi dào sức khỏe và thành công trong sự nghiệp cao quý.

**TÀI LIỆU THAM KHẢO**

1. Tài liệu môn Phân tích thiết kế hệ thống, ThS. Ngô Thị Bích Thúy, Khoa CNTT – Đại học Công nghiệp Hà Nội.
2. Tài liệu môn Kiểm thử phần mềm, TS. Nguyễn Hoàng Tú, Khoa CNTT –Đại học Công nghiệp Hà Nội.
3. https://viblo.asia/p/cach-xay-dung-testcase-co-ban-bWrZnLvO5xw