BỘ CÔNG THƯƠNG

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THỰC PHẨM**

** THÀNH PHỐ HỒ CHÍ MINH**

**Kiểm định chất lượng phần mền: website bán mỹ phẩm**

**SINH VIÊN THỰC HIỆN:**

1. 2001190791 – Từ Huệ Sơn
2. 2001190248 – Trần Thành Tâm
3. 2001190794 – Lê Đức Tài
4. 2001190876 – Huỳnh Mỹ Trân
5. 2001190866 – Lư Phước Toàn
6. 2001190520 – Nguyễn Trung Hậu

**GVHD: Trần Văn Thọ**

TP. HỒ CHÍ MINH, tháng 03 năm 2022

**Mục Lục**

[Chương 1: TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM 1](#_Toc101384777)

[1.1 Các khái niệm cơ bản về kiểm thử phần mềm 1](#_Toc101384778)

[1.1.1 Kiểm thử phần mềm là gì? 1](#_Toc101384779)

[1.1.2 Các phương pháp kiểm thử 1](#_Toc101384780)

[1.1.2.1 Kiểm thử tĩnh – Static testing 1](#_Toc101384781)

[1.1.2.2 Kiểm thử động – Dynamic testing 2](#_Toc101384782)

[1.1.3 Các chiến lược kiểm thử 2](#_Toc101384783)

[1.1.3.1 Kiểm thử hộp đen – Black box testing 2](#_Toc101384784)

[1.1.3.2 Kiểm thử hộp trắng – White box testing 3](#_Toc101384785)

[1.1.3.3 Kiểm thử hộp xám – Gray box testing 4](#_Toc101384786)

[1.1.4 Các cấp độ kiểm thử phần mềm 4](#_Toc101384787)

[1.1.4.1 Kiểm thử đơn vị - Unit Test 4](#_Toc101384788)

[1.1.4.2 Kiểm thử tích hợp – Intergration Test 5](#_Toc101384789)

[1.1.4.3 Kiểm thử hệ thống – System Test 6](#_Toc101384790)

[1.1.4.4 Kiểm thử chấp nhận sản phẩm – Acceptance Test 8](#_Toc101384791)

[1.1.4.5 Một số cấp độ kiểm thử khác 9](#_Toc101384792)

[1.1.5 Các phương pháp kiểm thử con người 10](#_Toc101384793)

[1.1.5.1 Tổng duyệt – Walkthrough 10](#_Toc101384794)

[1.1.5.2 Thanh tra mã nguồn – Code Inspection 10](#_Toc101384795)

[1.2 Nguyên tắc kiểm thử phần mềm 11](#_Toc101384796)

[Chương 2: THIẾT KẾ TEST PLAN 12](#_Toc101384797)

[2.1 Giới thiệu chung 12](#_Toc101384798)

[2.1.1 Tổng quan 12](#_Toc101384799)

[2.1.2 Phạm vi 12](#_Toc101384800)

[2.1.3 Các định nghĩa và các từ viết tắt được dùng 12](#_Toc101384801)

[2.1.4 Những người sử dụng tài liệu này 13](#_Toc101384802)

[2.2 Các yêu cầu kiểm thử 13](#_Toc101384803)

[2.2.1 Yêu cầu chức năng 13](#_Toc101384804)

[2.2.2 Yêu cầu phi chức năng 13](#_Toc101384805)

[2.2.3 Công cụ kiểm thử 13](#_Toc101384806)

[2.3 Các tiêu chí chấp nhận sản phẩm 14](#_Toc101384807)

[2.3.1 Tiêu chí tạm dừng 14](#_Toc101384808)

[2.3.2 Tiêu chí hoàn thành 14](#_Toc101384809)

[2.4 Các chiến lược kiểm thử 14](#_Toc101384810)

[2.5 Nguồn lực cho kiểm thử 15](#_Toc101384811)

[2.5.1 Nguồn nhân lực: Vai trò và trách nhiệm 15](#_Toc101384812)

[2.5.2 Nguồn lực hệ thống 15](#_Toc101384813)

[2.5.3 Môi trường test 16](#_Toc101384814)

[2.6 Các mốc kiểm thử 16](#_Toc101384815)

[2.7 Deliverables of test: Test Plan, Test Case, Test Reports 17](#_Toc101384816)

[Chương 3: THIẾT KẾ TEST CASE VÀ TEST DATA 17](#_Toc101384817)

[3.1 Khái niệm 17](#_Toc101384818)

[3.2 Vai trò của thiết kế test case 18](#_Toc101384819)

[3.3 Quy trình của thiết kế test case 18](#_Toc101384820)

[3.3.1 Kiểm thử hộp đen 19](#_Toc101384821)

[3.3.1.1 Phân lớp tương đương – Equivalence Patitioning 19](#_Toc101384822)

[3.3.1.2 Phân tích giá trị biên – Boundary Value Analysis 21](#_Toc101384823)

[3.3.1.3 Sử dụng đồ thị nguyên nhân - kết quả - Cause & Effect Graphing 22](#_Toc101384824)

[3.3.1.4 Đoán lỗi – Error Guessing 27](#_Toc101384825)

[3.3.2 Kiểm thử hộp trắng – Kiểm thử bao phủ logic 28](#_Toc101384826)

[3.3.3 Chiến lược 28](#_Toc101384827)

[3.3.4 Kiểm thử API 29](#_Toc101384828)

[3.3.4.1 Định nghỉa 29](#_Toc101384829)

[3.3.4.2 Test case trong API Testing 29](#_Toc101384830)

[3.3.4.3 Lý do kiểm thử API 30](#_Toc101384831)

**LỜI MỞ ĐẦU**

Trong thời buổi công nghệ thông tin hiện nay có mặt khắp mọi lĩnh vực, nhu cầu sử dụng các phần mềm để thực hiện công việc được hiệu quả, nhanh chóng ngày càng tăng cao. Việc đảm bảo chất lượng phần mềm ngày càng quan trọng. Bên cạnh đó, để đáp ứng nhu cầu chia sẻ thông tin, cũng như truyền đạt thông tin một cách nhanh chóng thì website là phương tiện có khả năng đáp ứng nhu cầu đó.

Ngày nay, các website được phát triển một cách mạnh mẽ, phổ biến và nhanh chóng. Tuy nhiên với sự phát triển vượt bậc thì cũng có không ít các trở ngại dẫn đến việc website không hoạt động một cách hiệu quả. Do đó, cần thiết phải kiểm thử và đảm bảo chất lượng của website.

Với sự hướng dẫn của thầy, chúng em thực hiện đề tài “Kiểm định chất lượng phần mền: website bán mỹ phẩm” và thực hiện trên website đã được xây dựng nhưng chưa hoàn thiện, cũng như gặp nhiều thiếu sót trong quá trình phát triển sản phẩm. Do hạn chế về mặt kiến thức cho nên không thể tránh khỏi sai sót trong quá trình làm báo cáo, rất mong được sự giúp đỡ và chỉ dạy của thầy và các bạn.

**Danh mục hình**

[Hình 2.1 Biểu đồ Gantt 17](#_Toc101384377)

[Hình 3.1 Các ký hiệu cơ bản của đồ thị nguyên nhân – kết quả 24](#_Toc101384378)

[Hình 3.2 Các ký hiệu ràng buộc 25](file:///C:\Users\Admin\Downloads\BaoCao_kdclpm_websiteBanMyPham.docx#_Toc101384379)

[Hình 3.3 Những xem xét được sử dụng khi dò theo đồ thị 26](file:///C:\Users\Admin\Downloads\BaoCao_kdclpm_websiteBanMyPham.docx#_Toc101384380)

**Danh mục bảng**

[Bảng 2.1 Các định nghĩa 12](#_Toc101384391)

[Bảng 2.2 Các từ viết tắt 13](#_Toc101384392)

[Bảng 2.3 Công cụ kiểm thử 14](#_Toc101384393)

[Bảng 2.4 Các chiến lược kiểm thử 15](#_Toc101384394)

[Bảng 2.5 Vai trò và trách nhiệm của thành viên 15](#_Toc101384395)

[Bảng 2.6 Phần cứng 16](#_Toc101384396)

[Bảng 2.7 Phần mềm 16](#_Toc101384397)

[Bảng 2.8 Các mốc kiểm thử 17](#_Toc101384398)

[Bảng 3.1 Phương pháp thiết kế ca kiểm thử 19](#_Toc101384399)

[Bảng 3.2 Mẫu liệt kê các lớp tương đương 20](#_Toc101384400)

# **TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM**

## **Các khái niệm cơ bản về kiểm thử phần mềm**

### **Kiểm thử phần mềm là gì?**

Kiểm thử phần mềm là quá trình khảo sát một hệ thống hay thành phần dưới những điều kiện xác định, quan sát và ghi lại các kết quả, và đánh giá một khía cạnh nào đó của hệ thống hay thành phần đó. (Theo Bảng chú giải thuật ngữ chuẩn IEEE của Thuật ngữ kỹ nghệ phần mềm- IEEE Standard Glossary of Software Engineering Terminology).

Kiểm thử phần mềm là quá trình thực thi một chương trình với mục đích tìm lỗi. (Theo “The Art of Software Testing” – Nghệ thuật kiểm thử phần mềm). Kiểm thử phần mềm là hoạt động khảo sát thực tiễn sản phẩm hay dịch vụ phần mềm trong đúng môi trường chúng dự định sẽ được triển khai nhằm cung cấp cho người có lợi ích liên quan những thông tin về chất lượng của sản phẩm hay dịch vụ phần mềm ấy. Mục đích của kiểm thử phần mềm là tìm ra các lỗi hay khiếm khuyết phần mềm nhằm đảm bảo hiệu quả hoạt động tối ưu của phần mềm trong nhiều ngành khác nhau. (Theo Bách khoa toàn thư mở Wikipedia).

Có thể định nghĩa một cách dễ hiểu như sau: Kiểm thử phần mềm là một tiến trình hay một tập hợp các tiến trình được thiết kế để đảm bảo mã hóa máy tính thực hiện theo cái mà chúng đã được thiết kế để làm, và không thực hiện bất cứ thứ gì không mong muốn. Đây là một pha quan trọng trong quá trình phát triển hệ thống, giúp cho người xây dựng hệ thống và khách hàng thấy được hệ thống mới đã đáp ứng yêu cầu đặt ra hay chưa?

### **Các phương pháp kiểm thử**

Có 2 phương pháp kiểm thử chính là: Kiểm thử tĩnh và Kiểm thử động.

#### **Kiểm thử tĩnh – Static testing**

Là phương pháp thử phần mềm đòi hỏi phải duyệt lại các yêu cầu và các đặc tả bằng tay, thông qua việc sử dụng giấy, bút để kiểm tra logic, lần từng chi tiết mà không cần chạy chương trình. Kiểu kiểm thử này thường được sử dụng bởi chuyên viên thiết kế người mà viết mã lệnh một mình.

Kiểm thử tĩnh cũng có thể được tự động hóa. Nó sẽ thực hiện kiểm tra toàn bộ bao gồm các chương trình được phân tích bởi một trình thông dịch hoặc biên dịch mà xác nhận tính hợp lệ về cú pháp của chương trình.

#### **Kiểm thử động – Dynamic testing**

Là phương pháp thử phần mềm thông qua việc dùng máy chạy chương trình để điều tra trạng thái tác động của chương trình. Đó là kiểm thử dựa trên các ca kiểm thử xác định bằng sự thực hiện của đối tượng kiểm thử hay chạy các chương trình. Kiểm thử động kiểm tra cách thức hoạt động của mã lệnh, tức là kiểm tra sự phản ứng vật lý từ hệ thống tới các biến luôn thay đổi theo thời gian. Trong kiểm thử động, phần mềm phải thực sự được biên dịch và chạy. Kiểm thử động thực sự bao gồm làm việc với phần mềm, nhập các giá trị đầu vào và kiểm tra xem liệu đầu ra có như mong muốn hay không. Các phương pháp kiểm thử động gồm có kiểm thử Unit – Unit Tests, Kiểm thử tích hợp – Intergration Tests, Kiểm thử hệ thống – System Tests, và Kiểm thử chấp nhận sản phẩm – Acceptance Tests

### **Các chiến lược kiểm thử**

Ba trong số những chiến lược kiểm thử thông dụng nhất bao gồm: Kiểm thử hộp đen, Kiểm thử hộp trắng, và Kiểm thử hộp xám.

#### **Kiểm thử hộp đen – Black box testing**

Một trong những chiến lược kiểm thử quan trọng là kiểm thử hộp đen, hướng dữ liệu, hay hướng vào/ra. Kiểm thử hộp đen xem chương trình như là một “hộp đen”. Mục đích của bạn là hoàn toàn không quan tâm về cách cư xử và cấu trúc bên trong của chương trình. Thay vào đó, tập trung vào tìm các trường hợp mà chương trình không thực hiện theo các đặc tả của nó.

Theo hướng tiếp cận này, dữ liệu kiểm tra được lấy chỉ từ các đặc tả.

**Các phương pháp kiểm thử hộp đen:**

* Phân lớp tương đương – Equivalence partitioning.
* Phân tích giá trị biên – Boundary value analysis.
* Kiểm thử mọi cặp – All-pairs testing.
* Kiểm thử fuzz – Fuzz testing.
* Kiểm thử dựa trên mô hình – Model-based testing.
* Ma trận dấu vết – Traceability matrix.
* Kiểm thử thăm dò – Exploratory testing.
* Kiểm thử dựa trên đặc tả – Specification-base testing.

Kiểm thử dựa trên đặc tả tập trung vào kiểm tra tính thiết thực của phần mềm theo những yêu cầu thích hợp. Do đó, kiểm thử viên nhập dữ liệu vào, và chỉ thấy dữ liệu ra từ đối tượng kiểm thử. Mức kiểm thử này thường yêu cầu các ca kiểm thử triệt để được cung cấp cho kiểm thử viên mà khi đó có thể xác minh là đối với dữ liệu đầu vào đã cho, giá trị đầu ra (hay cách thức hoạt động) có giống với giá trị mong muốn đã được xác định trong ca kiểm thử đó hay không. Kiểm thử dựa trên đặc tả là cần thiết, nhưng không đủ để ngăn chặn những rủi ro chắc chắn.

**Ưu, nhược điểm:**

Kiểm thử hộp đen không có mối liên quan nào tới mã lệnh, và kiểm thử viên chỉ rất đơn giản tâm niệm là: một mã lệnh phải có lỗi. Sử dụng nguyên tắc “ Hãy đòi hỏi và bạn sẽ được nhận”, những kiểm thử viên hộp đen tìm ra lỗi mà những lập trình viên đã không tìm ra. Nhưng mặt khác, người ta cũng nói kiểm thử hộp đen “giống như là đi trong bóng tối mà không có đèn vậy”, bởi vì kiểm thử viên không biết các phần mềm được kiểm tra thực sự được xây dựng như thế nào. Đó là lý do mà có nhiều trường hợp mà một kiểm thử viên hộp đen viết rất nhiều ca kiểm thử để kiểm tra một thứ gì đó mà đáng lẽ có thể chỉ cần kiểm tra bằng 1 ca kiểm thử duy nhất, và/hoặc một số phần của chương trình không được kiểm tra chút nào.

Do vậy, kiểm thử hộp đen có ưu điểm của “một sự đánh giá khách quan”, mặt khác nó lại có nhược điểm của “thăm dò mù”.

#### **Kiểm thử hộp trắng – White box testing**

Là một chiến lược kiểm thử khác, trái ngược hoàn toàn với kiểm thử hộp đen, kiểm thử hộp trắng hay kiểm thử hướng logic cho phép bạn khảo sát cấu trúc bên trong của chương trình. Chiến lược này xuất phát từ dữ liệu kiểm thử bằng sự kiểm thử tính logic của chương trình. Kiểm thử viên sẽ truy cập vào cấu trúc dữ liệu và giải thuật bên trong chương trình (và cả mã lệnh thực hiện chúng).

**Các phương pháp kiểm thử hộp trắng**

* Kiểm thử giao diện lập trình ứng dụng - API testing (application programming interface): là phương pháp kiểm thử của ứng dụng sử dụng các API công khai và riêng tư.
* Bao phủ mã lệnh – Code coverage: tạo các kiểm tra để đáp ứng một số tiêu chuẩn về bao phủ mã lệnh.
* Các phương pháp gán lỗi – Fault injection.
* Các phương pháp kiểm thử hoán chuyển – Mutation testing methods.
* Kiểm thử tĩnh – Static testing: kiểm thử hộp trắng bao gồm mọi kiểm thử tĩnh.

Phương pháp kiểm thử hộp trắng cũng có thể được sử dụng để đánh giá sự hoàn thành của một bộ kiểm thử mà được tạo cùng với các phương pháp kiểm thử hộp đen. Điều này cho phép các nhóm phần mềm khảo sát các phần của 1 hệ thống ít khi được kiểm tra và đảm bảo rằng những điểm chức năng quan trọng nhất đã được kiểm tra.

#### **Kiểm thử hộp xám – Gray box testing**

Kiểm thử hộp xám đòi hỏi phải có sự truy cập tới cấu trúc dữ liệu và giải thuật bên trong cho những mục đích thiết kế các ca kiểm thử, nhưng là kiểm thử ở mức người sử dụng hay mức hộp đen. Việc thao tác tới dữ liệu đầu vào và định dạng dữ liệu đầu ra là không rõ ràng, giống như một chiếc “hộp xám”, bởi vì đầu vào và đầu ra rõ ràng là ở bên ngoài “hộp đen” mà chúng ta vẫn gọi về hệ thống được kiểm tra. Sự khác biệt này đặc biệt quan trọng khi quản lý kiểm thử tích hợp – Intergartion testing giữa 2 mođun mã lệnh được viết bởi hai chuyên viên thiết kế khác nhau, trong đó chỉ giao diện là được đưa ra để kiểm thử. Kiểm thử hộp xám có thể cũng bao gồm cả thiết kế đối chiếu để quyết định, ví dụ: giá trị biên hay thông báo lỗi.

### **Các cấp độ kiểm thử phần mềm**

Kiểm thử phần mềm gồm có các cấp độ: Kiểm thử đơn vị, Kiểm thử tích hợp, Kiểm thử hệ thống và Kiểm thử chấp nhận sản phẩm.

#### **Kiểm thử đơn vị - Unit Test**

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.

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 thử, ghi nhận và phân tích kết quả kiểm thử. 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 thử và sửa lỗi ở các mức kiểm thử 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 phát triển phần mềm. Thông thường, Unit Test đòi hỏi kiểm thử 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 thử và quét hết Unit đòi hỏi phải có kỹ thuật, đôi khi phải dùng thuật toán để chọn lựa.

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.

#### **Kiểm thử tích hợp – Intergration 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.

Hai mục tiêu chính của Integration Test:

* 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 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.

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 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 ca kiểm thử giảm đi rất nhiều, và sai sót sẽ giảm đáng kể.

Có 4 loại kiểm thử trong Integration Test:

* **Kiểm thử cấu trúc (Structure Test):** Tương tự White Box Test, kiểm thử cấu trúc nhằm bảo đảm các thành phần bên trong của một chương trình chạy đúng và chú trọng đến hoạt động của các thành phần cấu trúc nội tại của chương trình chẳng hạn các câu lệnh và nhánh bên trong.
* **Kiểm thử chức năng (Functional Test)**: Tương tự Black Box Test, kiểm thử chức năng chỉ chú trọng đến chức năng của chương trình, mà khôngquan tâm đến cấu trúc bên trong, chỉ khảo sát chức năng của chương trìnhtheo yêu cầu kỹ thuật.
* **Kiểm thử hiệu năng (Performance Test):** Kiểm thử việc vận hành của hệ thống.
* **Kiểm thử khả năng chịu tải (Stress Test):** Kiểm thử các giới hạn của hệ thống.

#### **Kiểm thử hệ thống – System Test**

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.

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 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 kiểm thử 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).

Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan, 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...

Nhìn từ quan điểm người dùng, các cấp độ kiểm thử trên rất quan trọng: Chúng bảo đảm hệ thống đủ khả năng làm việc trong môi trường thực.

Lưu ý là 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.

#### **Kiểm thử chấp nhận sản phẩm – Acceptance Test**

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 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.

Thực tế cho thấy, 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.

#### **Một số cấp độ kiểm thử khác**

Ngoài các cấp độ trên, còn một số cấp độ kiểm thử khác như:

**Kiểm thử hồi quy – Regression Testing:**

Theo chuẩn IEEE610.12-90, kiểm thử hồi quy là “sự kiểm tra lại có lựa chọn của một hệ thống hay thành phần để xác minh là những sự thay đổi không gây ra những hậu quả không mong muốn...”. Trên thực tế, quan niệm này là chỉ ra rằng phần mềm mà đã qua được các kiểm tra trước đó vẫn có thể được kiểm tra lại. Beizer định nghĩa đó là sự lặp lại các kiểm tra để chỉ ra rằng cách hoạt động của phần mềm không bị thay đổi, ngoại trừ tới mức như yêu cầu. Hiển nhiên là sự thỏa hiệp phải được thực hiện giữa sự đảm bảo được đưa ra bởi kiểm thử hồi quy mỗi lần thực hiện một sự thay đổi và những tài nguyên được yêu cầu thực hiện điều đó.

**Kiểm thử tính đúng – Correctness testing:**

Tính đúng đắn là yêu cầu tối thiểu của phần mềm, là mục đích chủ yếu của kiểm thử. Kiểm thử tính đúng sẽ cần một kiểu người đáng tin nào đó, để chỉ ra cách hoạt động đúng đắn từ cách hoạt động sai lầm. Kiểm thử viên có thể biết hoặc không biết các chi tiết bên trong của các mođun phần mềm được kiểm thử, ví dụ luồng điều khiển, luồng dữ liệu, v.v .... Do đó, hoặc là quan điểm hộp trắng, hoặc là quan điểm hộp đen có thể được thực hiện trong kiểm thử phần mềm.

### **Các phương pháp kiểm thử con người**

Hai phương pháp kiểm thử con người chủ yếu là Code Inspections và Walkthroughs. Hai phương pháp này bao gồm một nhóm người đọc và kiểm tra theo mã lệnh của chương trình. Mục tiêu của chúng là để tìm ra lỗi mà không gỡ lỗi.

Một Inspection hay Walkthrough là 1 sự cải tiến của phương pháp kiểm tra mà lập trình viên đọc chương trình của họ trước khi kiểm thử nó. Inspections và Walkthroughs hiệu quả hơn là bởi vì những người khác sẽ kiểm thử chương trình tốt hơn chính tác giả của chương trình đó.

Inspections/Walkthroughs và kiểm thử bằng máy tính bổ sung cho nhau. Hiệu quả tìm lỗi sẽ kém đi nếu thiếu đi 1 trong 2 phương pháp. Và đối với việc sửa đổi chương trình cũng nên sử dụng các phương pháp kiểm thử này cũng như các kỹ thuật kiểm thử hồi quy.

#### **Tổng duyệt – Walkthrough**

Walkthrough là một thuật ngữ mô tả sự xem xét kỹ lưỡng của một quá trình ở mức trừu tượng trong đó nhà thiết kế hay lập trình viên lãnh đạo các thành viên trong nhóm và những người có quan tâm khác thông qua một sản phẩm phần mềm, và những người tham gia đặt câu hỏi, và ghi chú những lỗi có thể có, sự vi phạm các chuẩn phát triển và các vấn đề khác. Walkthrough mã lệnh là 1 tập các thủ tục và các công nghệ dò lỗi cho việc đọc nhóm mã lệnh. Trong một Walkthrough, nhóm các nhà phát triển – với 3 hoặc 4 thành viên là tốt nhất – thực hiện xét duyệt lại. Chỉ 1 trong các thành viên là tác giả của chương trình.

Một ưu điểm khác của walkthroughs, hiệu quả trong chi phí gỡ lỗi, là 1 thực tế mà khi một lỗi được tìm thấy, nó thường được định vị chính xác trong mã lệnh. Thêm vào đó, phương pháp này thường tìm ra 1 tập các lỗi, cho phép sau đó các lỗi đó được sửa tất cả với nhau. Mặt khác, kiểm thử dựa trên máy tính, chỉ tìm ra triệu chứng của lỗi (chương trình không kết thúc hoặc đưa ra kết quả vô nghĩa), và các lỗi thường được tìm ra và sửa lần lượt từng lỗi một.

#### **Thanh tra mã nguồn – Code Inspection**

Thanh tra mã nguồn là 1 tập hợp các thủ tục và các kỹ thuật dò lỗi cho việc đọc các nhóm mã lệnh. Một nhóm kiểm duyệt thường gồm 4 người. Một trong số đó đóng vai trò là người điều tiết – một lập trình viên lão luyện và không được là tác giả của chương trình và phải không quen với các chi tiết của chương trình. Người điều tiết có nhiệm vụ: phân phối nguyên liệu và lập lịch cho các buổi kiểm duyệt, chỉ đạo phiên làm việc, ghi lại tất cả các lỗi được tìm thấy và đảm bảo là các lỗi sau đó được sửa. Thành viên thứ hai là một lập trình viên. Các thành viên còn lại trong nhóm thường là nhà thiết kế của chương trình (nếu nhà thiết kế khác lập trình viên) và một chuyên viên kiểm thử.

## **Nguyên tắc kiểm thử phần mềm**

Để kiểm thử đạt hiệu quả thì khi tiến hành kiểm thử phần mềm cần phải tuân thủ một số quy tắc sau:

**Quy tắc 1**: Một phần quan trọng của 1 ca kiểm thử là định nghĩa của đầu ra hay kết quả mong muốn.

**Quy tắc 2**: Lập trình viên nên tránh tự kiểm tra chương trình của mình.

**Quy tắc 3**: Nhóm lập trình không nên kiểm thử chương trình của chính họ.

**Quy tắc 4**: Kiểm tra thấu đáo mọi kết quả của mỗi kiểm tra.

**Quy tắc 5**: Các ca kiểm thử phải được viết cho các trạng thái đầu vào không hợp lệ và không mong muốn, cũng như cho các đầu vào hợp lệ và mong muốn.

**Quy tắc 6**: Khảo sát 1 chương trình để xem liệu chương trình có thực hiện cái mà nó cần thực hiện chỉ là 1 phần, phần còn lại là xem liệu chương trình có thực hiện cái mà nó không cần phải thực hiện hay không.

**Quy tắc 7**: Tránh các ca kiểm thử bâng quơ trừ khi chương trình thực sự là 1 chương trình bâng quơ.

**Quy tắc 8**: Không dự kiến kết quả của kiểm thử theo giả thiết ngầm là không tìm thấy lỗi.

**Quy tắc 9**: Xác suất tồn tại lỗi trong 1 đoạn chương trình là tương ứng với số lỗi đã tìm thấy trong đoạn đó.

**Quy tắc 10**: Kiểm thử là 1 nhiệm vụ cực kỳ sáng tạo và có tính thử thách trí tuệ.

# **THIẾT KẾ TEST PLAN**

## **Giới thiệu chung**

### **Tổng quan**

<https://thegioiskinfood.com/> là một website dành cho tất cả mọi người, với nhiều mặt hàng mỹ phẩm khác nhau giúp khách hàng nhất là phái đẹp có thể mua sắm, tham khảo và đặt hàng các loại mỹ phẩm để chăm sóc da và làm đẹp bản thân.

### **Phạm vi**

Tài liệu kế hoạch kiểm thử này được áp dụng cho việc kiểm thử những chức năng

của website <https://thegioiskinfood.com/>

Các dịch vụ, chức năng cần kiểm tra:

* Tìm kiếm sản phẩm.
* Đăng nhập.
* Đặt hàng.
* Thêm sản phẩm mới vào giỏ hàng.
* Thanh toán sản phẩm ở giỏ hàng.

### **Các định nghĩa và các từ viết tắt được dùng**

Các định nghĩa

|  |  |
| --- | --- |
| **Thuật Ngữ** | **Định Nghĩa** |
| Comment | Là nơi khách hàng đánh giá về sản phẩm đang được bán. |
| Cart | Là nơi khách thêm sản phẩm muốn mua và đăt hàng. |
| Product portfolio | Là danh mục sản phẩm để phân chia thành từng loại mỹ phẩm. |

Bảng 2.1 Các định nghĩa

Các từ viết tắt

|  |  |
| --- | --- |
| **Từ viết tắt** | **Mô tả** |
| IE11 | Trình duyệt Web Windows Internet Explorer 11 |

Bảng 2.2 Các từ viết tắt

### **Những người sử dụng tài liệu này**

Tài liệu này sử dụng cho Test Manager, Test Designer, Tester.

## **Các yêu cầu kiểm thử**

### **Yêu cầu chức năng**

Tìm kiếm: kiểm tra chức năng tìm kiếm cơ bản:

* Tên sản phẩm

Kiểm tra chức năng xem thông tin sản phẩm gồm các thông tin:

* Tên sản phẩm
* Giá
* Nhà sản xuất

Kiểm tra chức năng thanh toán ở giỏ hàng

* Nhập xuất hóa đơn.
* Thông tin hóa đơn.
* Tính tiền dư cho khách hàng.

Tài khoản

* Kiểm tra chức năng đăng ký, đăng nhập vào hệ thống

### **Yêu cầu phi chức năng**

* Tốc độ xử lí của web.
* Giao điện thân thiện với người dùng.
* Thông tin hiển thị dễ nhìn.

### **Công cụ kiểm thử**

|  |  |  |  |
| --- | --- | --- | --- |
| **Hoạt động** | **Công cụ** | **Nhà cung cấp** | **Phiên bản** |
| Quản lý Test Case | Microsoft Office Excel | Microsoft | 2010 |
| Quản lý Configuration | Microsoft Office Word | Microsoft | 2010 |
| Defect Tracking | Microsoft Office Excel | Microsoft | 2010 |
| Quản lý tiến độ kiểm thử | Microsoft Project Professional | Microsoft | 2010 |

Bảng 2.3 Công cụ kiểm thử

## **Các tiêu chí chấp nhận sản phẩm**

### **Tiêu chí tạm dừng**

Nếu các thành viên trong nhóm báo cáo rằng có 40% trường hợp thử nghiệm thất bại, tạm dừng thử nghiệm cho đến khi nhóm phát triển sửa chữa tất cả các trường hợp thất bại.

### **Tiêu chí hoàn thành**

Chỉ định các tiêu chí biểu thị hoàn thành thành công giai đoạn thử nghiệm

* Tốc độ chạy bắt buộc là 100% trừ khi có lý do rõ ràng.
* Tỷ lệ vượt qua là 80%, đạt được tỷ lệ vượt qua là bắt buộc.

## **Các chiến lược kiểm thử**

|  |  |
| --- | --- |
| Mục đích kiểm tra | Đảm bảo các chức năng được kiểm tra hoạt động chính xã theo đặc tả yêu cầu. |
| Kỹ thuật | Thực thi tất cả các trường hợp có thể có cho mỗi nhóm chức năng sử dụng dữ liệu hợp lệ để xác định:   * Kết quả mong đợi khi dữ liệu hợp lệ được sử dụng. * Cảnh báo phù hợp hiện ra khi dữ liệu không hợp lệ được sử dụng. |
| Tiêu chuẩn dùng | Tất cả các testcase đã được thiết kế đều được thực thi.  Tất cả các lỗi tìm thấy đều được ghi nhận lý do rõ ràng để có thể giúp cho người lập trình dễ dàng khắc phục. |
| Chịu trách nhiệm kiểm thử | Test Designer/Tester. |
| Các kiểm thử | Kiểm thử bằng tay thủ công, tuần tự theo các bước định nghĩa trong testcase. |
| Xử lý ngoại lệ | Liệt kê tất cả các vấn đề liên quan phát sinh trong quá trình thực thi kiểm thử. |

Bảng 2.4 Các chiến lược kiểm thử

## **Nguồn lực cho kiểm thử**

### **Nguồn nhân lực: Vai trò và trách nhiệm**

|  |  |
| --- | --- |
| **Thành viên** | **Vai trò** |
| Từ Huệ Sơn | Test Manager / Test Designer / Tester: Lập kế hoạch kiểm thử, quản lý tiến độ hoạt động kiểm thử, thiết kế các testcase bổ sung và thực thi các testcase bổ sung |
| Lê Đức Tài | Test Designer / Tester: Thiết kế và viết các test case, thực thi các testcase cho chức năng đăng nhập tài khoản, xem lại Test Plan |
| Lư Phước Toàn | Test Designer / Tester: Thiết kế và viết các test case, thực thi các testcase cho chức năng đăng ký tài khoản, xem lại Test Plan |
| Huỳnh Mỹ Trân | Test Designer / Tester: Thiết kế và viết các testcase bổ sung, thực thi các testcase bổ sung, đọc các tài liệu tham khảo để cung cấp thông tin cần thiết cho nhóm, xem lại Test Plan |
| Trần Thành Tâm | Test Designer / Tester: Thiết kế và viết các test case, thực thi các testcase cho 2 chức năng Đặt hàng và thanh toán hóa đơn đặt hàng |
| Nguyễn Trung Hậu | Test Designer / Tester: Thiết kế và viết các test case, thực thi các testcase cho chức năng Tìm kiếm |

Bảng 2.5 Vai trò và trách nhiệm của thành viên

### **Nguồn lực hệ thống**

Phần cứng

Máy tính cá nhân có kết nối mạng Internet

|  |  |  |  |
| --- | --- | --- | --- |
| **CPU** | **RAM** | **HDD** | **Architecture** |
| 2.2GHz Intel Core i7-8750H | 16 GB | 1TB | 64 bit |

Bảng 2.6 Phần cứng

Phần mềm

|  |  |  |
| --- | --- | --- |
| **Tên phần mềm** | **Phiển bản** | **Loại** |
| Mozilla Firefox | 98.0.2 | Trình duyệt Web |
| Google Chrome | 96 | Trình duyệt Web |
| Microsoft Windows 10 Single Language | 10 | Hệ điều hành |
| Adobe Flash Player plugin | 2022 | Tích hợp vào trình duyệt Web |

Bảng 2.7 Phần mềm

Máy tính cá nhân có kết nối mạng Internet để có thể truy cập vào trang web mỹ phẩm <https://thegioiskinfood.com/> bằng trình duyệt.

Các chức năng của trang web <https://thegioiskinfood.com/> được kiểm tra trên các trình duyệt Mozilla Firefox 98.0.2 và Google Chrome 96. Các trình duyệt này sử dụng phần mềm tích hợp Adobe Flash Player 2022 để chạy các sản phẩm.

Hệ điều hành được sử dụng là Microsoft Windows 10 Single Language.

### **Môi trường test**

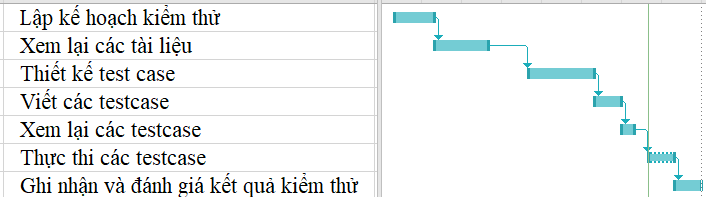
Test trên môi trường website

## **Các mốc kiểm thử**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Milestone** | **Deliverables** | **Duration (day)** | **Start Date** | **End Date** |
| Lập kế hoạch kiểm thử | Tài liệu Test Plan | 3 days | Tue 22/3/22 | Thu 24/3/22 |
| Xem lại các tài liệu | Tài liệu Test Plan | 3 days | Tue 22/3/22 | Thu 24/3/22 |
| Thiết kế test case | Tài liệu Test Case | 2 days | Fri 25/3/22 | Mon 28/3/22 |
| Viết các testcase | Tài liệu Testcase | 3 days | Fri 1/4/22 | Tue 5/4/22 |
| Xem lại các testcase | Tài liệu Testcase | 2 days | Wed 6/4/22 | Thu 7/4/22 |
| Thực thi các testcase | Tài liệu Testcase | 1 day | Fri 8/4/22 | Fri 8/4/22 |
| Ghi nhận và đánh giá kết quả kiểm thử | Tài liệu Testcase | 2 days | Sun 10/4/22 | Mon 11/4/22 |

Bảng 2.8 Các mốc kiểm thử

Biểu đồ gantt



Hình 2.1 Biểu đồ Gantt

## **Deliverables of test: Test Plan, Test Case, Test Reports**

* Liệt kê tên sản phẩm cần bàn giao
* Ngày bàn giao
* Người bàn giao
* Bàn giao cho ai.

# **THIẾT KẾ TEST CASE VÀ TEST DATA**

## **Khái niệm**

Thiết kế test case được định nghĩa là các đầu vào, điều kiện kiểm thử, quy trình kiểm thử và kết quả truy vấn với một mục tiêu kiểm thử cụ thể, đó là kiểm tra các tính năng của phần mềm trước khi đến với người dùng cuối để xây dựng phần mềm đạt tiêu chuẩn.

## **Vai trò của thiết kế test case**

Test case rất quan trọng trong bất kì dự án nào vì đây là bước đầu tiên trong quá trình test và nếu có gì đó sai sót ở bước này sẽ kéo theo hệ quả ở các giai đoạn tiếp theo trong vòng đời test.

Một tester luôn phải biết dữ liệu nào cần thực hiện test, thứ tự test đây là điều kiện quyết định cho việc test Test case liệt kê yêu cầu của khách hàng. Là việc quan trọng để xác định những thay đổi mà khách hàng mong muốn. Khi thay đổi một số chức năng của việc test nó cũng không thay đổi chức năng của phần mềm hay ứng dụng. Một bản test case bắt buộc phải có từng trạng thái riêng để người trưởng nhóm biết được chức năng nào của ứng dụng đã được test và chưa được test, những chức năng nào nhiều lỗi và ngược lại.

## **Quy trình của thiết kế test case**

Một trong những lý do quan trọng nhất trong kiểm thử chương trình là thiết kế và tạo ra các kiểm thử các test case có hiệu quả. Với những ràng buộc về thời gian và chi phí đã cho, thì vấn đề then chốt của kiểm thử trở thành tập con nào của tất cả ca kiểm thử có thể có khả năng tìm ra nhiều lỗi nhất.

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 và 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. Một chiến lược kiểm thử hợp lý là chiến lược có thể kết hợp sức mạnh của cả hai phương pháp trên: Phát triển 1 cuộc kiểm thử nghiêm ngặt vừa bằng việc sử dụng các phương pháp thiết kế ca kiểm thử hướng hộp đen nào có 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** |
| Phân lớp tương đương | Bao phủ câu lệnh |
| Phân tích giá trị biên | Bao phủ quyết định |
| Đồ thị nguyên nhân kết quả | Bao phủ điều kiện |
| Đoán lỗi | Bao phủ điều kiện – quyết định |
|  | Bao phủ đa điều kiện |

Bảng 3.1 Phương pháp thiết kế ca kiểm thử

Mỗi phương pháp có những ưu điểm cũng như khuyết điểm riêng, do đó để có được tập các ca kiểm thử tối ưu, chúng ta cần 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.

### **Kiểm thử hộp đen**

#### **Phân lớp tương đương – Equivalence Patitioning**

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”.

* + - * 1. 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.

Xác định các lớp tương đương:

Các lớp tương đương được xác định bằng cách lấy mỗi trạng thái đầu vào (thường là 1 câu hay 1 cụm từ trong đặc tả) và phân chia nó thành 2 hay nhiều nhóm (có thể sử dụng bảng 2.2 để liệt kê các lớp tương đương).

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

Bảng 3.2 Mẫu liệt kê các lớp tương đương

Chú ý là hai kiểu lớp tương đương được xác định: lớp tương đương hợp lệ mô tả các đầu vào hợp lệ của chương trình, và lớp tương đương không hợp lệ mô tả tất cả các trạng thái có thể khác của điều kiện (Ví dụ: các giá trị đầu vào không đúng). Với 1 đầu vào hay điều kiện bên ngoài đã cho, việc xác định các lớp tương đương hầu như là 1 quy trình mang tính kinh nghiệm. Để xác định các lớp tương đương có thể áp dụng tập các nguyên tắc dưới đây:

* Nếu 1 trạng thái đầu vào định rõ giới hạn của các giá trị, xác định 1 lớp tương đương hợp lệ và 2 lớp tương đương không hợp lệ.
* Nếu 1 trạng thái đầu vào xác định số giá trị, xác định 1 lớp tương đương hợp lệ và 2 lớp tương đương bất hợp lệ.
* Nếu 1 trạng thái đầu vào chỉ định tập các giá trị đầu vào và chương trình sử dụng mỗi giá trị là khác nhau, xác định 1 lớp tương đương hợp lệ cho mỗi loại và 1 lớp tương đương không hợp lệ.
* Nếu 1 trạng thái đầu vào chỉ định một tình huống “chắc chắn – must be”, xác định 1 lớp tương đương hợp lệ và 1 lớp tương đương không hợp lệ.
* Nếu có bất kỳ lý do nào để tin rằng chương trình không xử lý các phần tử trong cùng 1 lớp là như nhau, thì hãy chia lớp tương đương đó thành các lớp tương đương nhỏ hơn.

Xác định các ca kiểm thử:

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:

* Gán 1 số duy nhất cho mỗi lớp tương đương.
* Cho đến khi tất cả các lớp tương đương hợp lệ được bao phủ bởi (hợp nhất thành) các ca kiểm thử, viết 1 ca kiểm thử mới bao phủ càng nhiều các lớp tương đương đó chưa được bao phủ càng tốt.
* Cho đến khi các ca kiểm thử của bạn đã bao phủ tất cả các lớp tương đương không hợp lệ, viết 1 ca kiểm thử mà bao phủ một và chỉ một trong các lớp tương đương không hợp lệ chưa được bao phủ.
* 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.

Mặc dù việc phân lớp tương đương là rất tốt khi lựa chọn ngẫu nhiên các ca kiểm thử, nhưng nó vẫn có những thiếu sót. Ví dụ: nó bỏ qua các kiểu test – case có lợi nào đó. Hai phương pháp tiếp theo, phân tích giá trị biên và đồ thị nguyên nhân – kết quả, bao phủ được nhiều những thiếu sót này.

#### **Phân tích giá trị biên – Boundary Value Analysis**

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:

* 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.
* 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 1 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 1 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 1 trang web mua vé ngày khởi hành được quy định mức tối thiểu được chọn ngày hiện tại là ngày 19/05/2019 và ngày tối đa có thể chọn là ngày cuối cùng trong tháng là ngày 31/05/2019, hãy viết các ca kiểm thử mà ngày khởi hành là ngày 19/05/2019 và ngày 31/05/2019, ngày khởi hành trước ngày 19/05/2019 ví dụ ngày 18/05/2019 và ngày khởi hành sau ngày 31/05/2019 ví dụ ngày 01/06/2019 . 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 1 kết quả bên ngoài giới hạn đầu ra, nhưng tuy nhiên rất đáng để xem xét tiềm ẩn đó.
        4. Sử dụng nguyên tắc 2 cho mỗi trạng thái đầu ra.
        5. Nếu đầu vào hay đầu ra của 1 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.
        6. Sử dụng sự khéo léo của bạn để tìm các điều kiện biên.

#### **Sử dụng đồ thị nguyên nhân - kết quả - Cause & Effect Graphing**

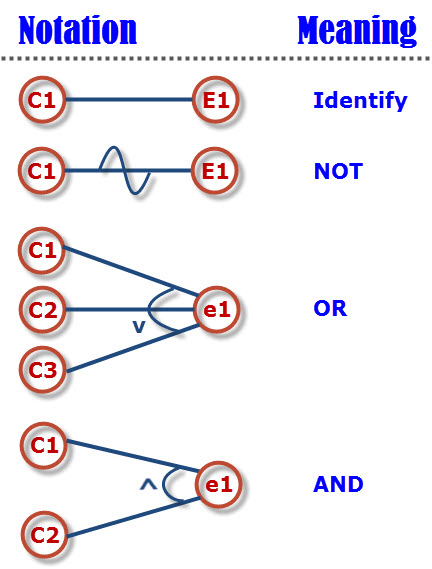
Một yếu điểm của phân tích giá trị biên và phân lớp tương đương là chúng không khảo sát sự kết hợp của các trường hợp đầu vào. Việc kiểm tra sự kết hợp đầu vào không phải là một nhiệm vụ đơn giản bởi vì nếu bạn phân lớp tương đương các trạng thái đầu vào, thì số lượng sự kết hợp thường là rất lớn. Nếu 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:

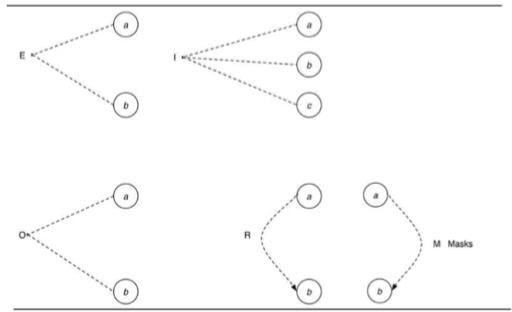
* + - * 1. Đặc tả được chia thành các phần có thể thực hiện được. Điều này là cần thiết bởi vì đồ thị nguyên nhân – kết quả trở nên khó sử dụng khi được sử dụng trên những đặc tả lớn.
        2. Nguyên nhân và kết quả trong các đặc tả được nhận biết. Một nguyên nhân là một trạng thái đầu vào nhất định 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 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 2.1. 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.



Hình 3.1 Các ký hiệu cơ bản của đồ thị nguyên nhân – kết quả

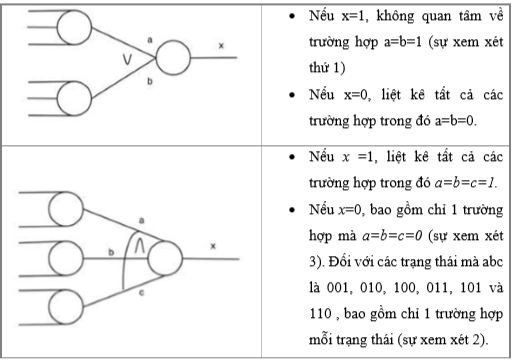
Trong hầu hết các chương trình, sự kết hợp nào đó của một số nguyên nhân là không thể bởi vì lý do ngữ nghĩa và môi trường (Ví dụ: một ký tự không thể đồng thời vừa là “A” vừa là “B”). Khi đó, ta sử dụng ký hiệu trong Hình 2.2. Ràng buộc E (Exclude – loại trừ) khẳng định rằng tối đa, chỉ có hoặc a hoặc b có thể là 1 (a và b không thể đồng thời là 1). Ràng buộc I (Include – bao hàm) khẳng định ít nhất một trong a, b hoặc c phải luôn luôn là 1 (a, b hoặc c không thể đồng thời là 0). Ràng buộc O (Only – chỉ một) khẳng định một và chỉ một hoặc a hoặc b phải là 1. Ràng buộc R (Request – yêu cầu) khẳng định rằng khi a là 1, thì b phải là 1 (Ví dụ: không thể có trường hợp a là 1, còn b là 0). Ràng buộc M (Mask – mặt nạ) khẳng định là nếu kết quả a là 1, kết quả b sẽ bắt buộc phải là 0.

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:

Hình 3.2 Các ký hiệu ràng buộc

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 3.3 Những xem xét được sử dụng khi dò theo đồ thị

Những sự xem xét này có thể xuất hiện thất thường, nhưng chúng có một mục đích rất quan trọng: để giảm bớt các kết quả được kết hợp của đồ thị. Chúng liệt kê các trường hợp mà hướng về các ca kiểm thử ít có lợi. Nếu các ca kiểm thử ít có lợi không được liệt kê, một đồ thị nguyên nhân – kết quả lớn sẽ tạo ra một số lượng ca kiểm thử cực kỳ lớn. Nếu số lượng các ca kiểm thử trên thực tế là quá lớn, bạn sẽ chọn ra 1 tập con nào đó, nhưng không đảm bảo là các ca kiểm thử ít có lợi sẽ là những ca kiểm thử được liệt kê. Do đó, tốt hơn hết là liệt kê chúng trong suốt quá trình phân tích của đồ thị.

**NHẬN XÉT**

Vẽ đồ thị nguyên nhân – kết quả là phương pháp tạo các ca kiểm thử có hệ thống mô tả sự kết hợp của các điều kiện. Sự thay đổi sẽ là 1 sự lựa chọn kết hợp không thể dự tính trước, nhưng khi thực hiện như vậy, có vẻ như bạn sẽ bỏ sót nhiều ca kiểm thử “thú vị” được xác định bằng đồ thị nguyên nhân – kết quả .

Vì vẽ đồ thị nguyên nhân – kết quả yêu cầu chuyển một đặc tả thành một mạng logic Boolean, nó cung cấp một triển vọng khác và sự hiểu biết sâu sắc hơn nữa về đặc tả. Trên thực tế, sự phát triển của 1 đồ thị nguyên nhân – kết quả là cách hay để khám phá sự mơ hồ và chưa đầy đủ trong các đặc tả.

Mặc dù việc vẽ đồ thị nguyên nhân – kết quả tạo ra tập các ca kiểm thử hữu dụng, nhưng thông thường nó không tạo ra tất cả các ca kiểm thử hữu dụng mà có thể được nhận biết. Ngoài ra, đồ thị nguyên nhân – kết quả không khảo sát thỏa đáng các điều kiện giới hạn. Dĩ nhiên, bạn có thể cố gắng bao phủ các điều kiện giới hạn trong suốt quá trình.

Tuy nhiên, vấn đề trong việc thực hiện điều này là nó làm cho đồ thị rất phức tạp và dẫn tới số lượng rất lớn các ca kiểm thử. Vì thế, tốt nhất là xét 1 sự phân tích giá trị giới hạn tách rời nhau.

Vì đồ thị nguyên nhân – kết quả làm chúng ta mất thời gian trong việc chọn các giá trị cụ thể cho các toán hạng, nên các điều kiện giới hạn có thể bị pha trộn thành các ca kiểm thử xuất phát từ đồ thị nguyên nhân – kết quả. Vì vậy, chúng ta đạt được một tập các ca kiểm thử nhỏ nhưng hiệu quả mà thỏa mãn cả 2 mục tiêu.

Khía cạnh khó nhất của kỹ thuật này là quá trình chuyển đổi đồ thị thành bảng quyết định. Quá trình này có tính thuật toán, tức là bạn có thể tự động hóa nó bằng việc viết 1 chương trình. Trên thị trường đã có một vài chương trình thương mại tồn tại giúp cho quá trình chuyển đổi này.

#### **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.

### **Kiểm thử hộp trắng – Kiểm thử bao phủ logic**

Kiểm thử Hộp Trắng là một phương pháp kiểm thử phần mềm trong đó tester biết về cấu trúc nội bộ / thiết kế. Người kiểm tra chọn đầu vào để thực hiện các đường dẫn thông qua mã và xác định đầu ra thích hợp. Kiến thức lập trình và kiến thức thực hiện là rất cần thiết trong kiểm thử hộp trắng.

Kiểm thử hộp trắng bao gồm phân tích dòng dữ liệu, điều khiển dòng, dòng thông tin, mã thực hành, ngoại lệ và những lỗi trình bày trong hệ thống để kiểm tra những hành động của phần mềm không được định hướng trước.

### **Chiến lược**

Các phương pháp thiết kế test-case đã được thảo luận có thể được kết hợp thành một chiến lược toàn diện. Vì mỗi phương pháp có thể đóng góp 1 tập riêng các ca kiểm thử hữu dụng, nhưng không cái nào trong số chúng tự nó đóng góp một tập trọn vẹn các các ca kiểm thử. Chiến lược hợp lý như sau:

1. Nếu đặc tả có chứa sự kết hợp của các điều kiện đầu vào, hãy bắt đầu với việc vẽ đồ thị nguyên nhân – kết quả.
2. Trong trường hợp bất kỳ, sử dụng phương pháp phân tích giá trị biên. Hãy nhớ rằng đây là một sự phân tích của các biên đầu vào và đầu ra. Phương pháp phân tích giá trị biên mang lại một tập các điều kiện kiểm tra bổ sung, và rất nhiều hay toàn bộ các điều kiện này có thể được hợp nhất thành các kiểm thử nguyên nhân – kết quả.
3. Xác định các lớp tương đương hợp lệ và không hợp lệ cho đầu vào và đầu ra, và bổ sung các ca kiểm thử được xác định trên nếu cần thiết.
4. Sử dụng kỹ thuật đoán lỗi để thêm các ca kiểm thử thêm vào.
5. Khảo sát tính logic của chương trình liên quan đến tập các ca kiểm thử. Sử dụng tiêu chuẩn bao phủ quyết định, bao phủ điều kiện, bao phủ quyết định/điều kiện, hay bao phủ đa điều kiện ( trong đó bao phủ đa điều kiện là được sử dụng nhiều nhất ). Nếu tiêu chuẩn bao phủ không đạt được bởi các ca kiểm thử được xác định trong bốn bước trước, và nếu việc đạt được tiêu chuẩn là không thể ( tức là, những sự kết hợp chắc chắn của các điều kiện có thể là không thể tạo vì bản chất của chương trình), hãy thêm vào các ca kiểm thử có khả năng làm cho thỏa mãn tiêu chuẩn.

Tuy việc sử dụng chiến lược này sẽ không đảm bảo rằng tất cả các lỗi sẽ được tìm thấy, nhưng nó đã được xác minh là đại diện cho một sự thỏa thuận hợp lý.

### **Kiểm thử API**

#### **Định nghỉa**

Về lý thuyết, API Testing (kiểm thử API) là một loại kiểm thử phần mềm bao gồm việc kiểm tra trực tiếp các giao diện lập trình ứng dụng và là một phần của kiểm thử tích hợp để xác định xem phần mềm có đáp ứng mong đợi về chức năng, độ tin cậy, hiệu suất và bảo mật hay không.

Kiểm thử API khác với những loại kiểm thử khác vì chưa có giao diện nên cần phải thiết lập môi trường khởi tạo. Gọi API với các tham số được yêu cầu và sau đó kiểm tra kết quả trả về.

* Những ví dụ kiểm thử API phổ biến như:
* Kiểm tra giá trị API được trả lại dựa trên điều kiện đầu vào
* Xác nhận API không trả lại kết quả gì hoặc kết quả sai
* Kiểm tra API có kích hoạt một số sự kiện khác hoặc gội một số sự kiện khác không
* Xác nhận API đang cập nhật cấu trúc dữ liệu nào đó

#### **Test case trong API Testing**

Test case trong API Testing được xây dựng dựa vào:

* Giá trị được trả về dựa trên điều kiện đầu vào: tương đối đơn giản khi kiểm tra, vì đầu vào có thể được xác định và kết quả có thể đã được xác thực.
* Không trả về bất cứ kết quả gì: Khi không có giá trị nào được trả về, một hành vị API trên hệ thống sẽ được tiến hành kiểm tra.
* Kích hoạt một số API/Interrupt/API: Nếu đầu ra của API được kích hoạt một số event hoặc gián đoạn, thì listener của interrupt hoặc event sẽ được theo dõi
* Cập nhật cấu trúc dữ liệu: Cập nhật cấu trúc dữ liệu sẽ trả về một số kết quả hoặc ảnh hưởng đến hệ thống và cần được xác thực.
* Sửa đổi một số tài nguyên: Nếu lệnh gọi API có sửa đổi một số tài nguyên thì nó phải được xác thực bằng các truy cập các tài nguyên tương ứng.

#### **Lý do kiểm thử API**

* Kiểm thử ứng dụng sớm mà không cần giao diện người dùng
* Tạo ra một chiến lược kiểm thử tự động, giảm thiểu chi phí