ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN THỊ TỰ

XÂY DỰNG CÔNG CỤ HỖ TRỢ SINH CA KIỂM THỬ CẶP

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm
Mã số: 60 48 01 03

LUẬN VĂN THẠC SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS ĐẶNG ĐỨC HẠNH

Hà Nội – 2016

LỜI CẢM ƠN

Lời đầu tiên tôi xin gửi lời cảm ơn chân thành và sâu sắc đến TS. Đặng Đức Hạnh và PGS. TS. Trương Anh Hoàng đã định hướng đề tài, liên tục quan tâm, tạo điều kiện thuận lợi trong suốt quá trình nghiên cứu và hoàn thành luận văn này.

Tôi xin được gửi lời cảm ơn đến các thầy, cô trong Bộ môn Công nghệ phần mềm cũng như Khoa Công nghệ thông tin đã mang lại cho tôi những kiến thức vô cùng quý giá và bổ ích trong quá trình theo học tại trường.

Tôi cũng xin chân thành cảm ơn đến gia đình tôi, đã tạo điều kiện để giúp đỡ để tôi có thời gian và nghị lực để hoàn thành luận văn này.

Cuối cùng, xin gửi lời cảm ơn chân thành nhất đến các bạn, các anh chị trong trường học và công ty Fpt software đã tạo điều kiện giúp đỡ tôi trong quá trình học tập và thực hiện luận văn này

Hà Nội, tháng 05 năm 2016

Học viên: Nguyễn Thị Tự

LỜI CAM ĐOAN

Tôi xin cam đoan luận văn này là công trình nghiên cứu của cá nhân tôi dưới sự hướng dẫn của thầy TS. Đặng Đức Hạnh, trung thực và không sao chép của tác giả khác. Trong toàn bộ nội dung nghiên cứu của luận văn, các vấn đề được trình bày đều là những tìm hiểu và nghiên cứu của chính cá nhân tôi hoặc là được trích dẫn từ các nguồn tài liệu có ghi tham khảo rõ ràng, hợp pháp. Nếu có vấn đề gì tôi xin hoàn toàn chịu trách nhiệm.

Người viết cam đoan

Nguyễn Thị Tự

MỤC LỤC

DANH SÁCH CÁC BẢNG KÝ HIỆU VÀ CHỮ VIẾT TẮT

Pairwise testing: Kiểm thử cặp dữ liệu

QTP: QuickTest Professional

Bug: Lỗi của phần mềm

Testcase: Ca kiểm thử

Test: Kiểm thử phần mềm

Design : Thiết kế

Create: Tạo

EC: Lớp tương đương

IPO: Thứ tự tham số

Requiment: Yêu cầu khách hàng

DANH SÁCH CÁC BẢNG

DANH SÁCH CÁC HÌNH

MỞ ĐẦU

Đặt vấn đề, định hướng nghiên cứu:

Trong những năm gần đây, chúng ta thấy rằng ngành công nghệ phần mềm phát triển ngày càng vượt bậc ở nhiều lĩnh vực. Đặc biệt tính ứng dụng cao bắt buộc cho phần mềm phải có một chất lượng nhất định. Việc phát triển phần mềm chỉ tập trung vào khâu thiết kế, lập trình là chưa đủ. Chúng ta cần tập chung cao vào cả khâu kiểm thử và đặc biệt hơn đó chính là kiểm thử chức năng (function). Nhưng kiểm thử như thế nào để có thể tiết kiệm chi phí, tối ưu nhất nguồn lực mà vẫn đảm bảo chất lượng.

Một giải pháp hợp lý cho các vấn đề đặt ra ở trên đó là áp dụng các kỹ thuật kiểm thử tối ưu và các công cụ kiểm thử tự động cho các phần mềm. Trong thực tế đã có rất nhiều công cụ kiểm thử tự động ví dụ như selenium IDE, QTP, nhưng nhìn trung lại chúng lại khá gò bó và mang nhiều nhược điểm.

Luận văn được thực hiện dựa trên ý tưởng từ nhu cầu thực tế và kiến thức được học. Cùng với đó là quá trình làm việc từ đó đưa ra cách thực hiện.

Luận văn được chia thành 3 chương, nội dung được phân bổ như sau:

Chương 1: Tổng quan về kiểm thử phần mềm.

Phần này nêu hệ thống cơ sở lý thuyết về kiểm thử như khái niệm cơ bản về kiểm thử, quy trình kiểm thử, các mức kiểm thử, các chiến lược kiểm thử và đặc biệt là các kỹ thuật trong kiểm thử chức năng.

Chương 2: Kỹ thuật kiểm thử cặp dữ liệu(Pairwise testing).

Phần này sẽ giới thiệu về kiểm thử cặp dữ liệu. Đây là một kỹ thuật trong kiểm thử chức năng. Trong đó luận văn sẽ nghiên cứu 2 kỹ thuật chính là mảng trực giao(OA) và thứ tự tham số(IPO). Ngoài ra phần này sẽ giới thiệu về công cụ sinh ra bộ dữ liệu kiểm thử theo phương pháp cặp dữ liệu là PICT.

Chương 3: Xây dựng công cụ sinh ca kiểm thử theo kỹ thuật cặp.

Phần này sẽ xây dựng một công cụ cho phép sinh ca kiểm thử dạng selenium IDE và kết hợp kỹ thuật cặp dữ liệu trong đó. Nó cho phép sinh một lúc nhiêu testcase.

Chương 1: TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM

1.1 Khái niệm kiểm thử phần mềm (Software Testing)

Kiểm thử phần mềm là quá trình thực thi một chương trình hay là một đơn vị (Module) của chương trình nhằm đánh giá chất lượng của sản phẩm phần mềm. Kiểm thử là một khâu mấu chốt và là bước phát triển cuối cùng để đảm bảo chất lượng phần mềm. Có thể nói đơn giản là kiểm thử là kiểm tra xem phần mềm có chạy đúng thiết kế (design) và đặc tả của nó hay không.

Mục đích của kiểm thử phần mềm là: tìm lỗi sai (bug), giải quyết bug và đưa ra những đánh giá, chứng nhận về chất lượng phần mềm

1.2 Một số thuật ngữ thường dùng trong kiểm thử phần mềm:

Bug: Là một khiếm khuyết trong một thành phần hoặc hệ thống mà nó có thể làm cho thành phần hoặc hệ thống này không thực hiện đúng chức năng yêu cầu của nó, Có thể là lỗi giao diện, lỗi chức năng, lỗi nghiệm vụ. Ví dụ như thông báo sai hoặc định nghĩa dữ liệu không đúng, hoặc là một nghiệm vụ bị sai so với yêu cầu…

Mối liên hệ giữa bug, Defect, fault, Error, failure, incident

Error: Hiểu đơn giản là sai sót trong quá trình viết code, một lỗi gì đó khiến cho chương trình không biên dịch được. Error có thể do lỗi hệ thống, chương trình or thành phần nào đó làm chương trình hoạt động không đúng.

Các mức độ của bug: Cosmetic, Medium, Serious, Fatal

Testcase: Được dịch ra trong tiếng việt là ca kiểm thử. Là một dạng thức mô tả một dữ liệu dầu vào, một số hành động (hành vy) hoặc sự kiện, và một kết quả mong đợi để xác định chức năng của một ứng dụng phần mềm hoạt động đúng hay không. Một testcase có thể có các phần đặc thù khác như mã testcase, tên testcase, mục tiêu test, điều kiện test (conditon), các yêu cầu input data, các bước thực hiện, và các kết quả mong đợi.

Có thể nôm la rằng testcase là một tình huống kiểm tra, được thiết kế để kiểm tra một đối tượng có thỏa mãn yêu cầu đặt ra hay không

Ví dụ một testcase được tạo bằng tay:

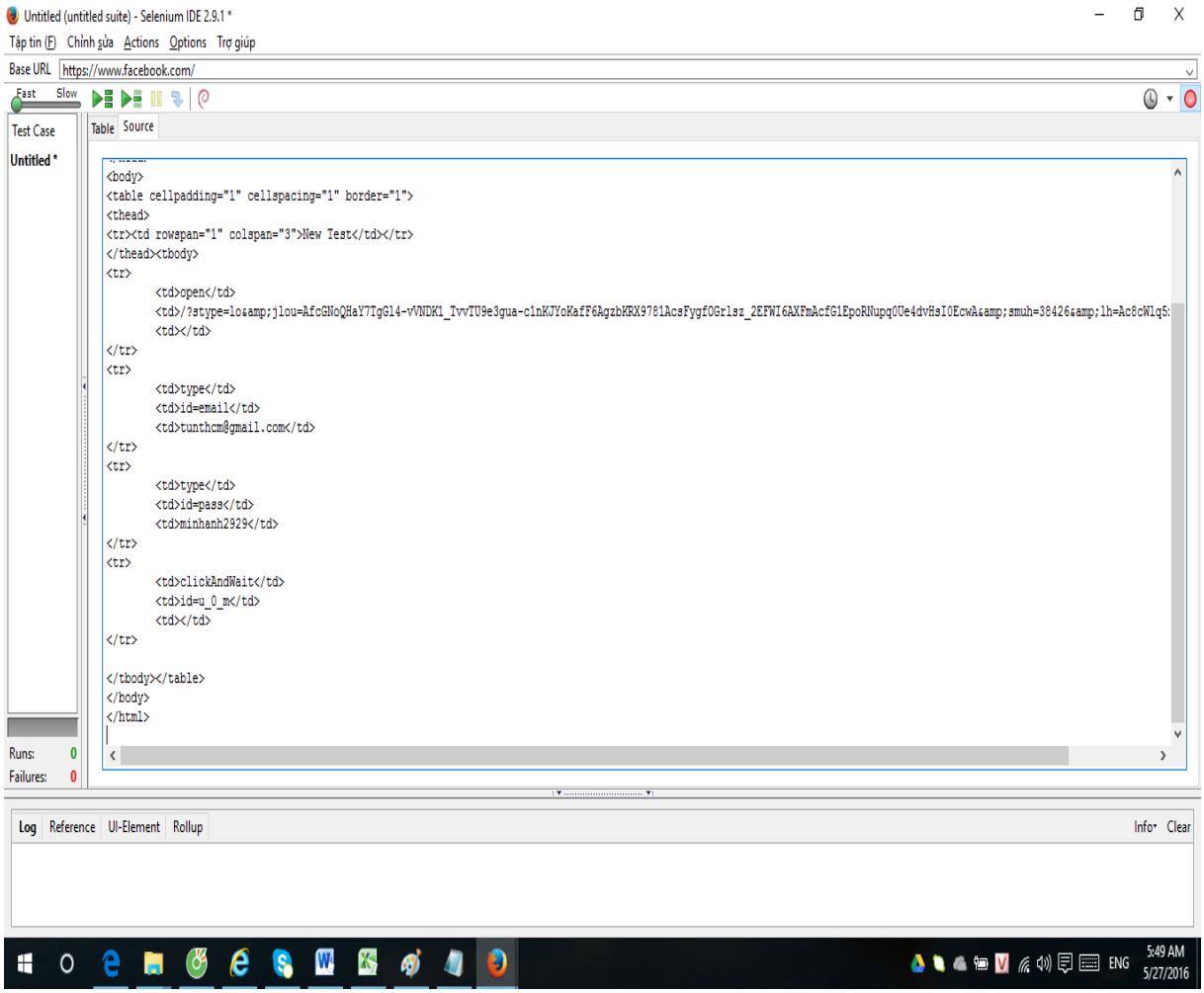
TT	Điều kiện tiền đề	Bước thực hiện	Mong muốn	Ghi chú	Kết quả	Kiểm thử viên	Ngày test
	Email: Tunthcm@gmail.com tồn tại	1. Open webpage: https://www.facebook.com/ 2. Tại [Username] input value "tunthcm@gmail.com] 3. Tại [password] input value: Minhanh2929 4. Click button [Login]	4. Login success facebook				

Bảng 1.1 Mẫu ca kiểm thử trong thực tế.

Ví dụ testcase automation với công cụ là selenium ide:

Testcase1		
open	/	
type	id=email	tunthcm@gmail.com
type	id=pass	minhanh2929
clickAndWait	id=u_0_w	

Bảng 1.2 Ca kiểm thử tự động selenium ide



Hình 1.1 Ca kiểm thử tự động selenium ide dạng mã nguồn

Test Script: Đây là một khái niệm mà nó liên quan đến công cụ kiểm thử tự động. Nó là một nhóm mã lệnh dạng đặc tả kịch bản dùng để tự động hóa một trình tự kiểm tra, giúp cho việc kiểm tra nhanh hơn hoặc cho những trường hợp mà kiểm tra bằng tay sẽ rất khó khăn hoặc không khả thi. Các Test Script có thể tạo thủ công hoặc tạo tự động dùng công cụ kiểm tra tự động.

OK/NG/NA/: Là những kết quả của testcase

Build và Release Version: Build và Release Version đều được dùng để chỉ một phiên bản của phần mềm. Tuy nhiên, ý nghĩa và trường hợp sử dụng thì khác nhau.

Build: Thường được dùng để chỉ 1 version phần mềm trong quá trình phát triển tại dự án. Các bản build liên tiếp nhau thường có một khác biệt nhỏ. Nó có thể fix thêm một bug, thay đổi một requite nhỏ.

Release Version: Được dùng để chỉ một bản build. Tuy nhiên, bản build này sẽ được gởi đến cho khách hàng kiểm thử chấp nhận. Những thay đổi giữa các Release Version liên tiếp nhau thường là khá lớn. Phải có nhiều build được viết và kiểm thử tại nhóm dự án thì mới có một Release Version

Fix bug: Giải quyết bug.

1.3 Quy trình kiểm thử phần mềm

Đây là quy trình kiểm thử phần mềm được áp dụng nhiều ở rất nhiều công ty hiện nay trong đó có fpt software.

Hình 1.2 Quy trình kiểm thử phần mềm

Đầu vào, đầu ra của quy trình

Thứ tự	Đầu vào	Hoạt động	Đầu ra
1	Requiment của khách hàng Hợp đồng, đơn đặt hàng.	Lập kế hoạch kiểm thử	Kế hoạch kiểm thử được chấp thuận
2	Testplan đã được chấp thuật. Tài liệu yêu cầu của sản phẩm được base.	Thiết kế tình huống test (Test design)	Test Design, Test ViewPoint được chấp thuận
3	Test Plan, Test design, hoặc là Test Viewpoint	Chuẩn bị dữ liệu	Test Cases: Unit Test Case, Integration Test Case, System Test Case - Test Script (có thể không) - Test Data (có thể không) - Test Environment
4	Test cases, Test data đã chuẩn bị và được chấp thuận	Thực hiện kiểm thử, ghi nhận kết quả, đánh giá kết quả	Defect List, Test Report, Test evident
5	According to project plan	Tổng hợp và báo cáo	Tổng hợp và báo cáo sẵn sàng.

Bảng 1.3 Mô tả quy trình

Sau đây chúng ta sẽ đi mô tả các bước quan trọng trong quy trình này:

1.3.1 Lập kế hoạch kiểm thử (Test plan)

a. Mục đích: Xác định nguồn nhân lực tham gia, lập lịch biểu, phạm vi kiểm thử, chiến lược kiểm thử, quy trình và công cụ sử dụng

b. Bước thực hiện

❖ Xác định các yêu cầu (requirement) cho việc kiêm thử gồm:

- Nghiên cứu requirements của khách hàng, tiêu chí chấp nhận, tài liệu đặc tả, tài liệu thiết kế (design) và những dàng buộc của khác hàng đối với sản phẩm.
- Xác định xem là những cái gì sẽ được kiểm thử, hay chính xác là phạm vi kiểm thử, ví dụ như kiểm thử ở giai đoạn nào, các kiểu kiểm thử, các module phải kiểm thử.
- Xác định phạm vi kiểm thử (Hạn chế của công việc, effort, lịch trình công việc, thời gian kiểm thử hồi quy.

❖ Xem xét và thống nhất các yêu cầu cho việc kiểm thử

❖ Đánh giá rủi ro và mức độ ưu tiên

- Đánh giá rủi ro đối với vấn đề liên quan
- Xác định và thiết lập mức độ ưu tiên cho các các chức năng cơ bản dựa trên mức độ nghiêm trọng của vấn đề, mong muốn của người dùng, mức độ quan trọng/ tần xuất sử dụng đối với từng chức năng.

❖ Xây dựng chiến lược thử:
- Phương pháp kiểm thử, giai đoạn kiểm thử (UT, Pre-IT,IT, ST, AT)
- Tiêu chí chấp thuận, và đánh giá việc kiểm thử (Tiêu chí này dựa trên TestDesign/Test Viewpoit/Test case/ Test report)
- Xem xét các trường hợp đặc biệt, nhân lực và điều kiện cơ sở vật chất để thực hiện test.

❖ Xác định nguồn nhân lực và môi trường bao gồm:
- Con người (số lượng và năng lực, kinh nghiệm)
- Môi trường kiểm thử(Bao gồm phần cứng và phần mềm)
- Công cụ sử dụng(Tools)
- Tất cả các loại dữ liệu kiểm thử(test data)

❖ Xác định lịch trình kiểm thử
- Dự đoán được effort test
- Tạo lịch trình kiểm thử và những mốc (milestones) quan trọng.
- Tạo kế hoạch kiểm thử
- Xem xét và thống nhất lập kế hoạch kiểm thử.

1.3.2 Thiết kế test (Test design):

a. Mục đích Thiết kế cho việc kiểm thử.

b. Bước thực hiện:

❖ Nghiên cứu tài liệu đặc tả, test plan

❖ Xác định Pass/Fail cho TestDesign/TestViewPoint (số items, tỉ lệ normal/abnormal/boundary case...)

❖ Xác định môi trường cho mỗi chức năng

❖ Liệt kê Test viewpoint/ Test suites cho mỗi chức năng dựa trên tài liệu, business/ domain knowledge, Q&A...

❖ Review TestDegign/Test viewpoint, đánh giá độ bao phủ (coverage) của test design.

❖ Approve TestDesign/ test Viewpoint

1.3.3 Chuẩn bị dữ liệu(Implement test):

a. Mục đích: Chuẩn bị cho việc test

b. Bước thực hiện

Tạo ca kiểm thử:

❖ Phân tích business process

❖ Phân tích sơ đồ use case, design, requirements, test plan

❖ Xác định test case: điều kiện test, kịch bản test, kết quả mong muốn.

❖ Xác định dữ liêu kiểm thử.

❖ Xác định cấu trúc thủ tục kiểm thử

❖ Phân tích test case

❖ Xác định thủ tục test

- ❖ Cấu trúc thủ tục test: xác định mỗi quan hệ và trình tự thực hiện của thủ tục test, điều kiện bắt đầu và kết thúc, mối quan hệ của thủ tục test và TC.
- ❖ Xác định thủ tục test: Hướng dẫn cách thực hiện, giá trị dữ liệu nhập vào, kết quả mong đợi
- ❖ Tạo test script cho việc thực hiện TC/Test procedure bằng cách:
  - Tạo
  - Gen ra
  - Thực hiện test script
- ❖ Chuẩn bị data test gồm new data và data cũ
- ❖ Chuẩn bị môi trường test bao gồm cơ sở vật chất, thiết bị, công cụ và các điều kiện yêu cầu khác.
- ❖ Review test script và kiểm tra lại các tool
- ❖ Review môi trường test, các điều kiện tiền đề và data test.

1.3.4 Thực hiện kiểm thử, ghi nhận kết quả và đánh giá kết quả test

a.Mục đích: Thực thi kiểm thử và đánh giá kết quả kiểm thử

b. Bước thực hiện

- ❖ Tiếp nhận sản phẩm test tài liệu, software package..
- ❖ Setup môi trường và cài đặt chương trình test
- ❖ Thực hiện test dựa trên TestDesign hoặc test script, ghi lại các dữ liệu thực tế liên quan đến môi trường, data test, hoạt động test và kết quả.
- ❖ Thực hiện phân tích nguyên nhân khi kết quả test khác với expect. Phối hợp với các team khác để điều tra bug như: lấy log, đánh dấu phần thay đổi để thay đổi design hoặc môi trường test.
- ❖ Theo dõi việc khắc phục lỗi

1.3.5 Tổng hợp và báo cáo:

a. Mục đích: Tóm tắt lại test result và đánh giá hoạt động test

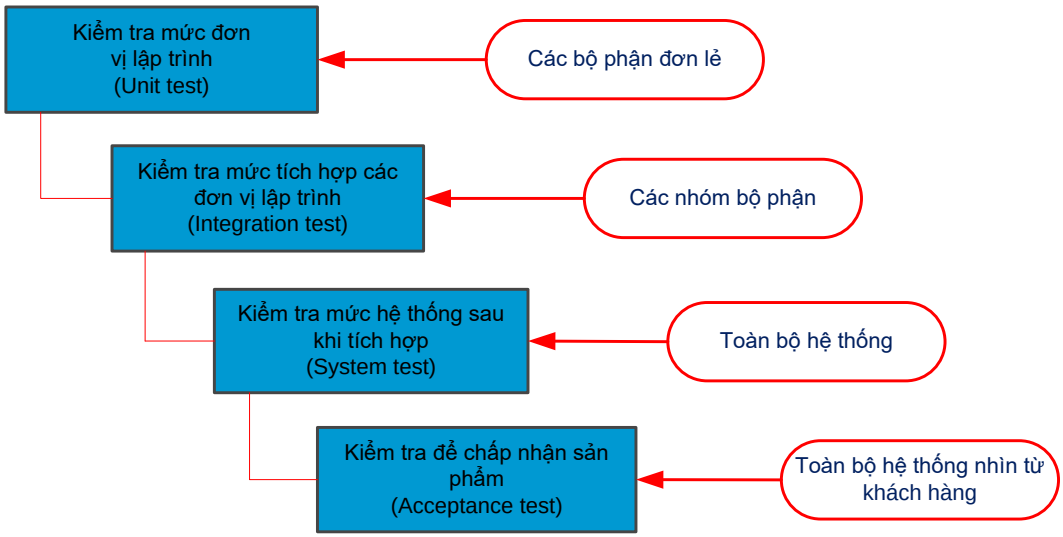
b. Bước thực hiện

- ❖ Tổng hợp các trường hợp (ca kiểm thử) lỗi, xác định mong muốn trong từng trường hợp.
- ❖ Tạo test report
- ❖ Review test report
- ❖ Maintain document.

Xác định sơ bộ hệ thống (sau khi tích hợp) có thỏa mãn yêu cầu đặt ra hay không.

1.4 Các mức kiểm thử phần mềm

Kiểm thử phần mềm không hoạt động một cách gò bó mà được thực hiện một cách linh hoạt . Điều đó phụ thuộc vào phần mềm đó phát triển theo mô hình nào và giai đoạn phát triển trong dự án phần mềm. **[3]**



Hình 1.3: 4 mức độ kiểm thử phần mềm cơ bản

1.4.1 Kiểm tra mức đơn vị (Unit Test)

Unit test là công đoạn thực thi test sớm nhất trong chu trình kiểm thử phần mềm. Đối tượng của unit test là những đơn vị có kích thước nhỏ. Hoạt động trong một chu trình đơn giản. Đôi khi nó cũng chỉ là một hàm, hoặc một chức năng.

Đặc điểm của unit test: Dễ tổ chức, kiểm tra, ghi nhận và phân tích kết quả. Nếu phát hiện lỗi thì dễ dàng phát hiện nguyên nhân, và dễ sửa chữa.

Có một nguyên lý là thời gian tốn trong việc unit test sẽ được đền bù bằng việc tích kiệm rất nhiều thời gian và chi phí cho việc kiểm tra và sửa lỗi ở các mức kiểm tra sau đó.

Unit Test thường do lập trình viên thực hiện. Công đoạn này cần được thực hiện càng sớm càng tốt trong giai đoạn viết code và xuyên suốt chu kỳ phát triển phần mềm. Và tất cả các đơn vị, các nhánh đều phải được thực hiện unit test.

Kỹ thuật được đặc biệt hay dùng với unit test là CFG và DFG. Và tool được sử dụng nhiều nhất cho unit test là junit và nunit.

1.4.2 Kiểm tra tích hợp (Integration Test)

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

Integration Test có hai mục tiêu chính

- Phát hiện lỗi giao tiếp xảy ra giữa các đơn vị Unit.
- Tích hợp các đơn vị Unit đơn lẻ thành các hệ thống nhỏ (subsystem) và cuối cùng là nguyên hệ thống hoàn chỉnh (system) chuẩn bị cho kiểm tra ở mức hệ thống (System Test).

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

tra giao tiếp của Unit mới thêm vào với hệ thống các Unit đã tích hợp trước đó, điều này làm cho số lượng kiểm tra sẽ giảm đi rất nhiều, sai sót sẽ giảm đáng kể.

Có bốn loại quan trọng nhất cần thực hiện kiểm tra trong Integration Test:

- Kiểm tra cấu trúc (Structure Test): Nhằm đảm bảo thành phần cấu trúc bên trong của một chương trình chạy đúng.
- Kiểm tra chức năng (Functional Test):Kiểm tra chức năng của chương trinìh theo yêu cầu kỹ thuật.
- Kiểm tra hiệu năng (Performance Test):Kiểm tra sự vận hành của hệ thống.
- Kiểm tra khả năng chịu tải (Stress Test): Kiểm tra các giới hạn của hệ thống.

1.4.3 Kiểm tra mức hệ thống (System Test)

System Test là kiểm tra 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 tra này tốn rất nhiều công sức và thời gian. Ở mức độ hệ thống người kiểm tra cũng tìm kiếm các lỗi nhưng trọng tâm là đánh giá về hoạt động, thao tác, sự tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống.

Điểm khác nhau then chốt giữa Integration Test và System Test là System Test chú trọng các hành vi và lỗi trên toàn hệ thống còn Integration Test chú trọng sự giao tiếp giữa các đơn thể hoặc đối tượng khi chúng làm việc cùng nhau. Thông thường ta phải thực hiện Unit Test và Integration Test để đảm bảo mọi Unit và sự tương tác giữa chúng hoạt động chính xác trước khi thực hiện System Test.

Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan nên System Test thường được thực hiện bởi một nhóm kiểm tra viên hoàn toàn độc lập với nhóm phát triển dự án.

Bản thân System Test lại gồm nhiều loại kiểm tra khác nhau, phổ biến nhất gồm:

- Kiểm tra chức năng(Functional Test): bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
- Kiểm tra khả năng vận hành (Performance Test): bảo đảm tối ưu việc phân bố tài nguyên hệ thống (ví dụ bộ nhớ) nhằm đạt các chỉ tiêu như thời gian xử lý hay đáp ứng yêu câu truy vấn…
- Kiểm tra khả năng chịu tải (Stress Test hay Load Test): bảo đảm hệ thống vận hành đúng dưới áp lực cao (ví dụ nhiều người truy xuất cùng lúc). Stress Test tập trung vào các trạng thái tới hạn, các “điểm chết”, các tình huống bất thường….
- Kiểm tra cấu hình (Configuration Test)
- Kiểm tra khả năng bảo mật (Security Test): bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.

• Kiểm tra khả năng phục hồi (Recovery Test): ảo 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 kiểm tra trên rất quan trọng bảo đảm hệ thống đủ khả năng làm việc trong môi trường thực. Nhưng không nhất thiết phải thực hiện tất cả các loại kiểm tra trên. Tùy yêu cầu và đặc trưng của từng hệ thống, tùy khả năng và thời gian cho phép của dự án mà áp dụng những loại kiểm tra nào.

1.4.4 Kiểm thử chấp nhận (Acceptance Test)

Acceptance Test thường có ý nghĩa là người dùng cuối kiểm thử chương trình xem sản phẩm phần mềm có đáp ứng đầy đủ những chức năng mà họ cần hoặc có đúng với quy trình công việc mà họ vẫn làm hay không? Tính dễ sử dụng, chức năng, khả năng chịu tải, …. Đây là mức quyết định xem sản phẩm đã thực sự hoàn thiện để chuyển tới người sử dụng. Chính là bước kiểm thử sau giai đoạn realease.

Gắn liền với giai đoạn Acceptance Test thường là một tài liệu đi kèm, phổ biến như hướng dẫn cài đặt, sử dụng, mã nguồn.….Tất cả phải được cập nhật và kiểm tra chặt chẽ.

1.4. 5 Kiểm tra hồi quy (Regression Test)

Kiểm thử hồi quy là kiểm tra lại phần mềm sau khi có một sự thay đổi xảy ra, để đảm bảo phiên bản phần mềm mới thực hiện tốt các chức năng như phiên bản cũ và sự thay đổi không gây ra lỗi mới trên những chức năng vốn đã làm việc tốt. Regression Test có thể thực hiện tại mọi mức kiểm thử khác.

1.5 Một số chiến lược kiểm thử

1.5.1 Kiểm thử hộp trắng (White-box Testing)

Kiểm thử hộp trắng là chiến lược kiểm thử dựa vào các cấu trúc, thuật toán bên trong của chương trình. Tức là dựa vào mã nguồn.

Kỹ thuật này thường dùng tại mức kiểm thử đơn vị. Và do Develop thực hiện. Giúp tối ưu hóa mã nguồn.

Kiểm thử hộp trắng có 3 kỹ thuật cơ bản là rà soát (review code) và các kỹ thuật kiểm thử động là CFG và DFG.

Kỹ thuật rà soát: Các kỹ thuật rà soát có thể chia ra làm các bước sau:

• Rà soát không chính thức (informal review): thực hiện bằng cách đọc các tài liệu liên quan và đưa ra các ghi chú, chưa cần phải phát hiện lỗi.

• Phản biện chéo (peer review): việc rà soát được thực hiện bởi đội ngũ lập trình dự án phần mềm, mọi người cùng tham gia thảo luận để đưa ra thống nhất chung về vấn đề kỹ thuật cho phù hợp với dự án. Cả đội sẽ cùng nhau rà soát mã nguồn, tìm lỗi và đưa ra cách giải quyết.

• Thông qua (walkthrough): người viết các mã nguồn, tài liệu đặc tả, thiết kế, .. sẽ giải thích từng bước trước toàn đội dự án nhằm đạt được sự hiểu rõ, đồng thuận, sau

đó nhận các phản hồi góp ý của thành viên trong đội dự án và đưa ra thay đổi hợp lý. • Thanh tra (inspection): các thành viên quan trọng trong đội dự án sẽ tham gia họp, một danh sách các vấn đề cần rà soát sẽ được lập và đưa ra để phát hiện lỗi và sữa chữa. Thanh tra khác thông qua ở chỗ người trình bày mã nguồn, tài liệu đặc tả, thiết kế không phải là người trực tiếp viết mà là một người khác, điều này khiến người trình bày phải thật sự hiểu về những gì sẽ giải thích. Vai trò của một số thành viên tham ra thực hiện kỹ thuật rà soát [6, tr.56-57]:

• Chủ tịch: người đóng vai trò chủ trì cuộc họp.

• Tác giả: người viết mã nguồn, tài liệu đặc tả, thiết kế đồng thời thực hiện các thay đổi được đề xuất.

• Người trình bày: người trình bày các tài liệu cần rà soát, có thể là tác giả nếu ở bước thông qua.

• Rà soát viên: người thực hiện việc nghiên cứu tài liệu, mã nguồn và đưa ra các câu hỏi và đề xuất thay đổi.

• Thư ký: người ghi chép lại các vấn đề được phát hiện trong cuộc họp. • Quan sát viên: những người chưa có kinh nghiệm về rà soát, tham gia cuộc họp để học kinh nghiệm

CFG:

Ý tưởng của kiểm thử dòng điều khiển chính là việc xây dựng một đồ thị dòng điều khiển và thiết kế các ca kiểm thử dựa trên các đường đi của đồ thị đó.

Đồ thị dòng điều khiển là đồ thị có các đỉnh tương ứng với các câu lệnh hay nhóm các câu lệnh và các cạnh là các dòng điều khiển giữa các câu lệnh hay nhóm các câu lệnh.

Ví dụ minh họa

DFG:

1.5.2 Kiểm thử hộp đen (Black-box Testing)

Kỹ thuật kiểm thử hộp đen xem chương trình như là một hộp đen. Được thực hiện bằng cách cho chạy chương trình và quan sát để tìm ra những hành vi, những hoạt động mong muốn và không mong muốn.

Các phương pháp kiểm thử hộp đen tập trung nhiều nhất vào các yêu cầu chức năng của phần mềm. Ngoài ra còn có thể có các yêu cầu khác như khả năng chịu tải, load test.

Về đặc điểm kiểm thử chức năng của chương trình, luận văn đưa ra riêng một phần trong phần 1.6 dưới đây.

1.5.3 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”..

1.6 Kiểm thử chức năng.

Kiểm thử chức năng là một tập hợp các kỹ thuật giúp cho việc kiểm thử chức năng năng của hệ thống phần mềm. Xem hệ thống phần mềm có hoạt động đúng với những yêu cầu

1.6.1 Các kiểu dữ liệu (type of variables)

Chúng ta sẽ xem xet một số kiểu dữ liệu phổ biến như numeric, arrays, substructures, and strings

Numeric Miền của biến số được định nghĩa theo 2 cách sau:

Thứ 1: Một tập các giá trị rời rạc (discrete). Ví dụ là biến mode = {23,79}

Thứ 2: Một đoạn, vài giá trị liên tục. Trong contigous segment được đặc trưng bởi giá trị nhỏ nhất và giá trị lớn nhất

Arrays:

Kiểu dữ liệu có các phần tử có kiểu dũ liệu giống nhau. Mỗi phần tử riêng lẻ được truy xuất bằng cách chỉ số(indices) .

- Một mảng có thể có một hoặc nhiều chiều. A[i][j] là một phần tử ở dòng i, cột j của mảng hai chiều. Thường được sử dụng ở các vòng lặp for, while..

- Những phần tử riêng lẻ được xem xét như là biến số. Tât cả các giá trị cần được xuất hiện trong kiểm thử trong quan sát hành vi của chương trình tổng khi xử lý các giá trị đặc biệt.

- Một phần của một mảng có thể được làm sáng tỏ một cách chung như là một cấu trúc con riêng biệt với những những thuộc tính ứng dụng phụ thuộc đặc tả.

Substructures: Loại dữ liệu có nhiều phần tử dữ liệu,.. mỗi phần tử có thể có những kiểu dữ liệu khác nhau..

Strings:Kiểu dữ liệu chuỗi

1.6.2 Khái niệm kiểm thử chức năng:

Theo Howden, một chức năng được xác định là một tập của cặp (X_i, Y_i), trong đó xi là một vector của biến nhập, và yi là một vector của biến xuất. Trong kiểm thử chức năng, một chương tình P được xem như là một chức năng khi truyền vector nhập X_i vào một vector xuất Y_i, giả sử Y_i =p(X_i)

Ví dụ1: Cho Y_1 = sqrt(x_1). ks

Ở đây p là là chức năng tính căn bậc 2, căn bậc 2 (Y_1) của số nguyên không âm(X_1),kết quả được gán cho (Y_1)

Ví dụ 2: Cho y = sort(x). Chương trình P trong ví dụ này là sự thực hiện của một thuật toán sắp xếp, đây là một thủ tục sắp xếp được một mảng y, từ một vecstor đầu vào x = {A,N}. Ở đây A là một mảng và N là số phần tử của trong A.

Một chức năng gồm 3 phần tử chính là đầu vào, đầu ra và thành phần biến đổi thông tin của đầu vào cho đầu ra.

Tóm lại Chúng ta có thể tóm tắt một số điểm chính trong kiểm thủ chức năng như sau:

- Indentify: nhận dạng biến đầu vào và đầu ra của chương trình và miền dữ liệu của chúng

- Tính toán: kết quả mong muốn cho mỗi sự lựa chọn giá trị đầu vào.

- Xác định giá trị đầu vào, cái là nguyên nhân cho chương trình để đưa ra được sự lựa chọn outputs .

Tạo ra dữ liệu kiểm thử bằng việc phân tích miền đầu vào (input domains) và miền đẩu ra.

1.6.3 Phân lớp tương đương (Equivalence class partioning)

Một miền đầu vào (input domain) có thể quá lớn để tất cả các phần tử có thể được sử dụng khi kiểm thử. Tuy nhiên miền này có thể được phân chia thành những miền con hữu hạn để lựa chọn kiểm tra. Mỗi miền con này được gọi là 1 lớp tương đương EC và đóng vai trò như là một tài nguyên chứa ít nhất một giá trị để kiểm tra.

Đây là phương pháp điển hình để giảm bớt tổng số lượng testcase, để hạn chế tập của các testcase có thể kiểm tra được, mà vẫn có thể cover được số lượng lớn requirements.

EC là một tập hợp (nhóm) của các đầu vào được xử lý tương tự như nhau, hành vi tương tự và cùng có kết quả mong muốn (expected result). Nó thường đại diện cho một điều kiện (condition) hoặc là một (Vị từ) predicate.

Nguyên tắc phân vùng tương đương cho một số trường hợp

[1]. Một điều kiện đầu vào (input conditon)chỉ rõ là một dẫy [a,b]:

Xác định 1 lớp tương đương hợp lệ(valid) cho a<=x<=b;

Và 2 lớp tương đương không hợp lệ(invalid) là x<a và X>b;

[2]. Một điều kiện đầu vào (input conditon) xác định là tập của các giá trị.

Tạo Ec cho mỗi phần tử của tập. Và một EC cho một member invalid.

[3]. Một input condition xác đinh cho mỗi giá trị riêng lẻ.

Nếu hệ thống có mỗi valid input khác nhau, thì tạo ra một Ec cho mỗi valid input.

Ví dụ. nếu input từ một menu, thì tạo ra một EC cho mỗi menu item.

[4]. Một input condition xác định số của nhiều giá trị valid(say N):

Tạo ra 1 EC cho corect number, và 2 EC invalid là giá trị 0 và giá trị lớn hơn 100.

Nếu một chương trình nhận 100 số tự nhiên để sắp xếp, thì tạo ra 3 lớp tương đương là EC valid nhập 100 số.

EC invalid: là không nhập số nào, và nhập lớn hơn 100 số.

[5] Một conditon input xác định là một giá trị " must-be": ký tự đầu tiên của password phải là một ký số. Thì chung sta sẽ yêu cầu tạo ra 2 EC.

Một là giá trị đúng| ký tự đầu tiên của password phải là ký tự số. một giá trị sai| ký tự đầu tiên không là một số.

[6] Phân chia Ec, nếu những phần tử trong phân vùng đã được chia của Ec được xử lý một cách khác nhau thì chia cắt EC đó thành các Ec nhỏ hơn.

Ví dụ xác định EC:

Khi đã xác định Ec tương đương, -→ sẽ xác định được các ca kiểm thử theo các bước sau:

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

Bước 2: Đối với mỗi Ec valid, chưa cover bởi 1 testcase, viết testcase mới cover càng nhiều EC càng tốt.

Bước 3: Đối với mỗi Ec invalid chưa cover bởi 1 testcase, viết 1 testcase mới cover một và chỉ một Ec đó.

Ví dụ: Xem xét phần mềm tính toán thuế dựa theo AGI(Adjusted Gross Income):

If AGI is between $1 and $29,500, the tax due is 22% of AGI.

If AGI is between $29,501 and $58,500, the tax due is 27% of AGI.

If AGI is between $58,501 and $100 billion, the tax due is 36% of AGI

EC1 valid AGI =[$1; $29,500] →TC=20000$

EC2 invalid: AGI <$1;-→ TC= -10000$

EC3 valid: AGI =[$29,501; $58,500]→ TC= 30000$

EC4 valid AGI [$58,501;$100 billion] → TC= 60000$

EC5 invalid AGI>$100 billion → TC= 150bilion$

1.6.4 Phân tích giá trị biên(Boundary value analysis)

Điều kiện biên cho mỗi lớp tương đương được phân tích để tạo ra testcase.

Điều kiện biên là tình trạng trực tiếp ở phía trên, dưới của các lợp tương đương đầu vào và đầu ra. Việc phân tích giá trị biên khác với phân hoach tương đương.

Một số quy tắc:

[1].Nếu điều kiện đầu vào xác định là một khoảng giá trị giữa a và b, thì biên sẽ là a, b và các giá trị sát trên và dưới của a và b.

[2]. Nếu một điều kiện đầu vào xác định một số giá trị, các testcase sẽ được tạo để kiểm thử giá trị cực đại, cực tiểu, các giá trị sát trên, sát dưới giá trị cực đại, cực tiểu.

[3]. Áp dụng cả đối với điều kiện đầu ra.

[4]. Nếu cấu trúc dữ liệu chương trình bên trong quy định các biên, thì tạo TC từ các biên của nó.

1.6.5 Bảng quyết định (Decision tables)

Là một kỹ thuật đơn giản nhưng mạnh để mô tả một hệ thống phức tạp. Kiểm tra được sự phối hợp giữa các điều kiện đầu vào.

Bảng quyết định bao gồm một tập các conditions(causes), effects(result) được sắp xếp vào một column bên trái của bảng, theo form.

		Rules or Combinations							
Conditions	Values	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
C_1	Y, N,—	Y	Y	Y	Y	N	N	N	N
C_2	Y, N,—	Y	Y	N	N	Y	Y	N	N
C_3	Y, N,—	Y	N	Y	N	Y	N	Y	N
Effects									
E_1		1		2	1				
E_2			2	1			2	1	
E_3		2	1	3		1	1		
Checksum	8	1	1	1	1	1	1	1	1

Ở column thứ hai, bên cạnh các condition, chúng ta có một số giá trị có thể có của nó làYes(Y), No(N) và gạch ngang--. Ở bên phải của cột values, chúng ta có một tập của các rules. Mỗi sự tổ hợp của 3 condition có tồn tại một số rule tự tập {R1,R2,R3,R..R8}

Trong mỗi một rules, các giá trị của condition có thể là yes, no, hoặc là gạch ngang và bao gồm một số danh sách effects liên quan{E1,E2,E3}. Cho mỗi effect thích hợp, một số thứ tự đặc biệt trong đó effect có thể mang ra ngoài nếu như một tập các điều kiện được trong đó được thỏa mãn.

Nó tồn tại một số nguyên tắc(rules) cho mỗi sự kết hợp của các condition.

Mỗi quy tắc sẽ bao gồm một câu trả lời Y(yes), N(no),-- (don't care) và bao gồm một tập các ảnh hưởng lien quan.

Thay vì đó, mỗi một rule trong bảng quyết định được thể hiện thành một testcase.

Các bước để triển khai testcase trong việc áp dụng kỹ thuật bảng quyết định:

Bước 1**:** Xác định các condition và các effects cho mỗi đơn vị riêng biệt.

Một condition là một trạng thái đầu vào riêng biệt hay là một EC của (input condition) . Một effect là một trạng thái đầu ra. Xác định mối quan hệ logical giữa những condition và effect.

Bước 2: Liệt kê tât cả các condition và effects vào trong form của bảng quyết định. Viết xuống những giá trị cho các condition có thể mang. Đặt condition quan trọng nhất ở trên, và condition mang nhiều giá trị ở cuối cùng.

Bước 3: Tính toán số lượng có thể kết hợp. Nó bằng với số lượng của các giá trị khác biệt làm tăng thêm nguồn lực của số lượng các conditions.

Nếu tất cả condition chỉ đơn giản là Y va N thì ta sẽ có $2^{\text{number of condition.}}$.

Nếu 1 condition có 3 giá trị và 3 condition còn lại có 2 giá trị thì ta sẽ có $3^1 \times 2^3 = 24$.

Bước 4: Hãy điền vào các columns với tất cả sự kết hợp có thể mỗi columns tương ứng với một sự kết hợp của các value. Mỗi một row(condition) làm như sau:

1. Xác định rõ những yếu tố lặp lại(RF): Chia số còn lại của kết hợp bằng số của các giá trị có thể cho mỗi condition đó.

2. Viết số lần RF giá trị đầu tiên, rồi số lần RF tiếp… cho đến khi row không trống.

Tiếp đến các dòng sau, … đến row 1.

Bước 5: Giảm sự kết hợp (rules). Tìm kiếm sự phối hợp không khác biệt và thay bằng --, vị trí dấu gạch ngang và nối column nơi mà những column giống hệt nhau. Trong khi làm điều này, hãy đảm bảo rằng mọi ảnh hưởng là như nhau.

Bước 6: Kiểm tra covered của điều kiện đầu vào sự kết hợp các rules. Cho mỗi một column tính sự kết hợp mà nó đại điện. Một dấu gạch ngang đại diện cho nhiều sự kết hợp của nhiều điền kiện. Làm tăng lên nhiều lần cho mỗi dấu gạch ngang xuống một column. Thêm vào tổng số và so sánh với bước 3. Nó có thể giống nhau.

Bước 7: Thêm những effects vào column của bảng quyết định. Đọc những column bằng mỗi column và xác định ảnh hưởng. nếu nhiều hơn một effect có thể xẩy ra sự kết hợp duy nhât, sau đó chỉ định một số thứ tự các effect. Do đó xác định thứ tự mà những tác động cần được thực hiện. Kiểm tra sự thống nhất của bảng quyết định.

Bước 8: The columns in the decision table are transformed into test cases.

Decision table – based testing is effective under certain conditions as follows:

• The requirements are easily mapped to a decision table.

• The resulting decision table should not be too large. One can break down a large decision table into multiple smaller tables.

• Each column in a decision table is independent of the other columns.

Những column trong bảng quyết định sẽ được chuyển đổi thành những testcase. Bảng quyết định được dựa vào để kiểm thử mang lại hiệu quả trong những điều kiện nhất định như sau:

Requirements được dễ dàng ánh xạ(mapped) tới bảng quyết định.

Kết quả bảng quyết định không phải quá lớn: người ta có thể phá một bảng thành nhiều bảng nhỏ hơn.

Mỗi cột trong bảng quyết định không phụ thuộc vào các column khác.

Ví dụ áp dụng:

Xem xét thủ tục trả lương. Consultants working hơn 40h một tuần được trả lương theo tỷ lệ số giờ làm việc của họ cho 40h đầu tiên và gấp 2 lần tỷ lệ số giờ làm việc cho những giờ thiếp theo. Consultants working làm việc ít hơn 40 h mỗi tuần, được trả lương cho giờ họ đã làm ở mức tỷ lệ số giờ của họ và đưa ra báo cáo vắng mặt. nhân viên lâu dài làm việc ít hơn 40h một tuần được trả theo mức lương của họ và đưa ra báo cáo vắng mặt. nhân viên lâu năm làm việc hơn 40h mỗi tuần được trả lương theo mức lương của họ.

Chúng ta cần mô tả thủ tục trả lương trên sử dụng kỹ thuật bảng quyết định để tạo ra testcase.

Bước 1: Xác định các condition và effects:

C1: lao động thường xuyên:

C2: làm việc <40h

C3: làm việc =40h

C4: Làm việc >40h.

E1: Trả lương.

E2: Đưa ra báo cáo vắng mặt.

E3: Trả lương theo giờ

E4: Trả lương gấp 2 lần theo giờ.

Bước 2: Tạo ra bảng quyết định

		Rules or Combinations															
Conditions	Values	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C_1	Y, N	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
C_2	Y, N	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N
C_3	Y, N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
C_4	Y, N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Effects																	
E_1																	
E_2																	
E_4																	
E_4																	

Bước 3: Tổng số của sự kết hợp là 16: $2^{4.}$

Bước 4: Xác định RF(repeating factors)

RF của row1: 16/2 = 8;

RF của row2: 8/2 = 4;

RF của row1: 4/2 = 2;

RF của row1: 2/2 = 1;

Vì vậy dòng đầu tiên sẽ được điền với 8 Y, 8 N.

Dòng thứ 2 sẽ được điền là 4Y,4N.

Dòng thứ 3 sẽ điền là 2Y,2N

Dòng thứ 4 sẽ điền là 1Y,1N ..

Bước 5: Giảm các rules.

Nếu là lao động cố định và làm viêc nhỏ hơn 40h thì C3 và C4 không cần quan tâm.

Thay vì vậy, ta sẽ giảm rule 1,2,3,4 thành các rules đơn mà không ảnh hưởng đến effects.

Nếu là lao động cố định và làm việc < 40h là N, thì C3 và C4 không phải là vấn đề, thay vì dó rules 5,6,7,8 có thể giảm bớt thành rule đơn – Không chú ý tới.

Nếu C1 là N và C2 là Y thì C3 và C4 không quan trọng. Thay vì đó, 9,10,11,12 sẽ được giảm thành rule đơn mà không ảnh hưởng đến effect.

Nếu C1 và C2 là N, nhưng C3 là Y thì rule 13,14 có thể được giảm thành rule đơn.

Rules 15 và 16 thì vẫn vậy.

Sau khi giảm ta có bảng sau:

		Rules or Combinations					
Conditions	Values	1	2	3	4	5	6
C_1	Y, N	Y	Y	N	N	N	N
C_2	Y, N	Y	N	Y	N	N	N
C_3	Y, N	—	—	—	Y	N	N
C_4	Y, N	—	—	—	—	Y	N
Effects							
E_1							
E_2							
E_4							
E_4							

Checksum cho columns 1,2,3,4,5, 6 là 4,4,2,1,1. Tổng của checksum là 16, giống với giá trị được tính ở bước 3.

Conditions	Values	1	2	3	4	5	6
C_1	Y, N	Y	Y	N	N	N	N
C_2	Y, N	Y	N	Y	N	N	N
C_3	Y, N	—	—	—	Y	N	N
C_4	Y, N	—	—	—	—	Y	N
Effects							
E_1							
E_1		1	1				
E_2		2		2			
E_4				1	1	1	
E_4						2	
Checksum	16	4	4	4	2	1	1

Bước 7: trong bước này, những effect được thêm vào cho mỗi column(rule). Column đầu tiên, nếu C1 và C2 được thỏa mãn, thì nhân viên phải được trả lương và có báo cáo vắng mặt đưa ra. ; thay vì đó E1 và E2 được đánh dấu. như là 1 và 2 của bảng quyết định.

Ví dụ áp dụng trong thực tế: Chức nang #002 tại cphone.

1.6.6 Kiểm thử ngẫu nhiên(Random testing)

Trong kỹ thuật này, đầu vào được lựa chọn một cách ngẫu nhiên tự tập miền đầu vào của hệ thống.

Bước 1: Xác định input domain.

Bước 2: Đầu vào được lựa chọn một cách độc lập từ miền.

Bước 3: Kiểm thử hệ thống được thực hiện trên các những đầu vào. Các inputs tạo thành một tập các kiểm thử ngẫu nhiên.

Bước 4: Kết quả của ramdon testing sẽ được so sánh với đặc tả của hệ thống. Kiểm thử sẽ thất bại nếu như đầu vào bất kỳ dẫn đến kết quả không chính xác, ngược lại thành công.

1.6.7 Đoán lỗi (Error guesing)

Error guessing – đoán lỗi. Đưa ra một chương trình và thực hiện phỏng đoán cả bằng trực giác và kinh nghiệm, các loại lỗi có thể và sau đó viết các testcase để đưa ra các lỗi đó.

Error guessing 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.

1.6.8 Category partition (CPM)

Đây là phương pháp kiểm thử chức năng cổ điển.

1. Phân vùng miền đầu vào của đơn vị chức năng để kiểm thử thành lớp tương đương.
2. Lựa chọn dữ liệu từ EC của phân vùng, phương pháp CPM dựa trên đặc tả kỹ thuật của hệ thống. Công việc chính của người thiết kế kiểm thử là phát triển categories (hạng mục). Mỗi category được phân thành các EC của những đầu vào được gọi là sự lựa chọn.

Sự lựa chọn trong mỗi category phải tách rời nhau, và cùng với các lựa chọn khác trong category phải cover được các input domain.

Các bước của phương pháp CPM:

Bước 1: phân tích đặc tả. Phương pháp này bắt đầu bằng việc phân tích (phân nhỏ) các đặc tả chức năng thành các đơn vị chức năng nhỏ. Mỗi một đơn vị chức năng được nhận dạng như sau:

- Những tham số của đơn vị chức năng.
- Đăc trưng của mỗi tham số, đó là, mỗi phần tử đặc chưng đó ảnh hưởng đến thực thi của đon vị.
- Một đối tượng trong môi trường, cái trạng thái của nó có thể ảnh hưởng đến các thao tác của đơn vị chức năng.
- Đặc chưng của mỗi đối tượng môi trường.

Nhiều tham số ..

Bước 2: Nhận dạng categories:

Một categorie là cái được phân loại của thuộc tính chính của tham số hay là một điều kiện môi trường.

Chương 2: KIỂM THỬ CẶP DỮ LIỆU

2.1 Tổng quan

Pairwise testing là kỹ thuật kiểm thử thuộc phạm vi của kiểm thử chức năng. Mục đích của nó là tạo ra bộ dữ liệu kiểm thử có kích thước nhỏ nhưng có thể cover được nhiều lỗi nhất có thể. Kỹ thuật này được biết đến gần 20 năm nay, nhưng nó chỉ phổ biến và gia tăng trong vòng 5 năm nay và hiện nay đã trở thành một kỹ thuật không thể thiếu trong kiểm thử phần mềm.

Nhiều kỹ thuật như là BVA và EP đã nói trong chương 1, giúp cho việc chuyển đổi số lượng lớn của biến vào trong một tập nhỏ nhiều. Và qua nhiều năm một số chiến lược kết hợp đã được đưa ra để giúp nhân viên kiểm thử chọn lựa được tập con của tổ input đầu vào như random testing, each-choice and base choice và cuối cùng chiến lược t-wise testing, với pairwise testing đã trở thành mạnh nhất trong số này.

Trong chương này tôi sẽ tìm hiểu về kiểm thử cặp dữ liệu với 2 kỹ thuật cơ bản là mảng trực giao và IPO. Chepter 9 [1]

Ngoài ra tôi sẽ trình bầy về bộ công cụ sinh ra bộ dữ liệu kiểm thử theo kỹ thuật pairwise đó là PICT[4]

2.2 Vector kiểm thử (Test vector.)

Test vector được gọi là test data, là một thể hiện của đầu vào cho chương trình. Nó là một dạng của những giá trị của tất cả biến đầu vào(it is a certain configuration of the value of all the input variables).

Giá trị của những biến riêng biệt đã chọn trong các phương pháp kiểm thử chức năng khác phải được phối hợp để tạo ra vector kiểm thử. Nếu chương trình có n biến đầu vào, mỗi biến sẽ có k giá trị tương ứng thì có k_1, k_2, k_3, ….. k_n…kn có thể phối hợp để tạo ra dữ liệu kiểm thử.

2.3 Kiểm thử cặp dữ liệu (Parirwise testing)

Đầu tiên chúng ta hãy xem xét khái niệm kiểm thử kết hợp tất cả "all-combination testing" hay có thể gọi theo một cụm từ khác là allwise. Nó được hiểu đơn giản là kiểm thử tất cả các kết hợp có thể có của các giá trị của một tập các biến. Chúng ta xét n biến đầu vào là :

$V = \{v_1, v_2, v_3, \ldots v_{n-1}, v_n\}$

Với mỗi biến đầu vào ta chọn k giá trị quan tâm. Vậy theo như " all combination testing " ta phải xem xét k^n vectors kiểm thử. Như vậy thì số lượng test case sẽ rất lớn. Khi mà số lượng biến lớn và giá trị nhiều.

Thay vì như vậy chúng ta có thể xem xét và áp dụng kiểm thử cặp dữ liệu (pairwise testing). Pairwise được hiểu là tât cả các kết hợp đôi một (cặp) có thể có của các giá trị của tập biến đầu vào. Mỗi cặp giá trị đó sẽ được xuất hiện ít nhất một lần trong một trường hợp kiểm thử. Nó là một trường hợp đặc biệt của " all combination testing".
Nó thường được gọi là "all-pair/two-way testing"

Vídụ: Ta xét 3 biến X,Y,Z là 3 biến đầu vào của hệ thống S.

X = {true,false}

Y = {0;5}

Z={Q;R}

Theo " all combination testing ”, tổng số trường hợp kiểm thử là = 2x2x2=8 vector kiểm thử :

Testcase ID	Input x	Input y	Input z
TC1	True	0	Q
TC2	True	0	R
TC3	True	5	Q
TC4	True	5	R
TC5	False	0	Q
TC6	False	0	R
TC7	False	5	Q
TC8	False	5	R

Nhưng với pairwise testing ta sẽ chỉ cố 4 vector:

Testcase ID	Input x	Input y	Input z
TC1	True	0	Q
TC2	True	5	R
TC3	False	0	Q
TC4	False	5	R

Tích kiệm các trường hợp kiểm thử như vậy, nhưng liệu chúng có hiệu quả trong việc cover lỗi không.

Kiểm thử được tất cả các kết hợp có thể có của giá trị của tập các biến đương nhiên sẽ tốt hơn. Hiệu quả được đưa ra theo thống kê được đưa ra tại [5] thì pairwise có thể phát hiện ra được 70% các lỗi. Còn four way testing thì có thể phát hiện ra 100% các lỗi. Và đương nhiên thì all combination cũng cover được 100% các lỗi.

Sau đây chúng ta sẽ đi nghiên cứu một số phương pháp kỹ thuật để phối hợp bộ dữ liệu kiểm thử.

2.3.1 Mảng trực giao (Orthogonal array ($L_{run}(Lever^{factors})$))

Phương pháp được nghiên cứu bởi nhà thống kê CR.Raoo va sau năm 1940 Genichi Tagumi là người đầu tiên sử dụng ý tưởng mảng trực giao trong những thiết kế thí nghiệm về quản lý chất lượng toàn diện(total quality management). Vì vậy mà phương pháp này được biết đến là phương pháp Tagumi, đã được sử dụng trong những kỹ thuật thiết kế thử nghiệm trong lĩnh vực sản xuất và cung cấp một cách có hiệu quả, hệ thống để tối ưu hóa thiết kế đảm bảo hiệu xuất, chất lượng và chi phí.

Phương pháp được sử dụng thành công tại nhật và mỹ, với mục tiêu là thiết kế có độ tin cậy cao, chất lượng sản phẩm cao với chi phí thấp trong ngành công nghiệp điện tử ô tô và tiêu dùng.

Mandl là ngươi đầu tiên sử dụng khái niệm của mảng trực giao trong việc thiết kế các testcase của pairwise.

Ưu điểm của phương pháp

- Đảm bảo sự kết hợp của tất cả các biến được lựa chọn.
- Tạo ra một bộ testcase hiệu quả và ngắn gọn
- Tạo ra một tập các testcase có sự phân bố (đồng đều)của tất cả các sự kết hợp trong kỹ thuật pairwise.
- Đơn giản để tạo ra và ít lỗi so với được tạo bằng tay(phương pháp khác).

Nhược điểm của phương pháp.

- Không phải cái gì cũng có thể sử dụng.
- Không phải tất cả đều có thể áp dụng kỹ thuật này. Ví dụ như kỹ thuật này chỉ áp dụng đối với các biến rời rạc.

Các bước của phương pháp mảng trực giao:

Bước 1: Xác định số lớn nhất(max) của biến độc lập của hệ thống. Số này sẽ được gán làm factors . 1 input variables → sẽ là 1 factor.

Bước 2: Xác định số lớn nhất giá trị của mỗi biến đầu vào. Số này được gán là Levels của mảng trực giao.

Bước 3: Tìm mảng trưc giao phù hợp với số run nhỏ nhất.

Ta có $L_{run}(x^y)$ trong đó x: Levers, y Factors;($L_{run}(Lever^{factors})$)

Trong bảng này ta sẽ có:

Runs: Số lượng của rows trong mảng, cũng chính là số test cases được tạo ra bởi phương pháp OA này.

Factors: Số Cột của mảng trực giao.

Levers: Số lớn nhất của values, được mang bởi một bất kỳ một factor đơn nào đó.

Bước 4: Ánh xạ mỗi biến vào 1 factors và mỗi giá trị vào 1 levers trên bảng

Bước 5: Check for any “left-over” levers in the aray that have not been mapped. Choose arbitrary valid value..

Bước 6: Chuyển đổi run thành testcase.

Sau đây là bảng giúp cho việc lựa chọn mảng trực giao phù hợp:

Orthogonal Array	Number of Runs	Maximum Number of Factors	Maximum Number of Columns at These Levels			
			2	3	4	5
L_4	4	3	3			
L_8	8	7	7			
L_9	9	4	—	4		
L_{12}	12	11	11			
L_{16}	16	15	15			
L'_{16}	16	5	—	—	5	
L_{18}	18	8	1	7		
L_{25}	25	6	—	—	—	6
L_{27}	27	13	—	13		
L_{32}	32	31	31			
L'_{32}	32	10	1	—	9	
L_{36}	36	23	11	12		
L'_{36}	36	16	3	13		
L_{50}	50	12	1	—	—	11
L_{54}	54	26	1	25		
L_{64}	64	63	63			
L'_{64}	64	21	—	—	21	
L_{81}	81	40	—	40		

Hình 2.1 Bảng lựa chọn mảng trực giao tùy theo số lượng lever và factors.

Một số ví dụ về chi tiết của mảng trực giao:

$L_4(2^3)$

	Factors		
Runs	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Hình 2.2 Mảng trực giao **$L_4(2^3)$**

$L_9(3^4)$

	Factors			
Runs	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Hình 2.3 Mảng trực giao $\mathbf{L_9\,(3^4)}$.

Một số mẫu bảng khác có thể tham khảo tại 2 website sau đây:

http://www.york.ac.uk/depts/maths/tables/orthogonal.htm

http://www.freequality.org/documents/tools/Tagarray_files/tamatrix.htm

Ví dụ: Hãy xem xét một trang web, được xem trên một số trình duyệt và với một vài plugin và một số hệ điều hành, thông qua một số kết nối khác nhau như sau:

Browser	Netscape, IE, FF
Plug in	Real player, media player
Os	Window, linux,macintosh.
Connection	Lan,PPP, Isps

Bước 1: Có 4 biến độc lập, là Browser, Plug-in, Os,Connecter ---Factor =4;
Bước 2: Mỗi biến mang 2 or 3 giá trị, maxvalue = 3,→lever=3;
Bước 3: Chọn mảng trực giao phù hợp là $L_9(3^4)$

	Factors			
Runs	**1**	**2**	**3**	**4**
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2

7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Bước 4: Ánh xạ các biến vào các factors, các giá trị vào các levers;

Theo thứ tự: Browser =1; plug-in =2; connection =3; Và tương ứng

Netscape =1; IE=2, FF=3
Real player=1, media player =2
Window=1, linux=2,macintosh=3
Lan=1,PPP=2, Isps=3

Ta sẽ được mảng trực giao sau đây.

	Factors			
Run	**Browsers**	**Phug-in**	**OS**	**Connection**
1	Netscape	Realplayer	Window	Lan
2	Netscape	2	Linux	PPP
3	Netscape	Mediaplayer	Macintosh	Isdn
4	IE	Realplayer	Linux	Isdn
5	IE	2	Macintosh	Lan
6	IE	Mediaplayer	Window	PPP
7	FF	Realplayer	Macintosh	PPP
8	FF	2	Window	Isdn
9	FF	Mediaplayer	Linux	Lan

Bước 5: Yếu tố Phug-in có 3 mức, nhưng chỉ có 2 giá trị cho biến được ánh xạ, 1. Chúng ta phải cung cấp một giá trị trong ô, việc lựa chọn giá trị tùy ý nhưng phải coverage. Bắt đầu ở phía trên của cột phug-in và vòng thông qua các giá trị có thể khi điền vào left-over. Chúng ta có bảng sau:

	Factors			
Runs	**Browsers**	**Phug-in**	**OS**	**Connection**
1	Netscape	Realplayer	Window	Lan
2	Netscape	Readplayer	Linux	PPP
3	Netscape	Mediaplayer	Macintosh	Isdn
4	IE	Readplayer	Linux	Isdn
5	IE	Mediaplayer	Macintosh	Lan
6	IE	Mediaplayer	Window	PPP
7	FF	Readplayer	Macintosh	PPP

8	FF	Readplayer	Window	Isdn
9	FF	Mediaplayer	Linux	Lan

Bước 6: Tạo ra 9 testcase từ mỗi run.

+ Mỗi browsers sẽ kiểm được kiểm tra ở tất cả các Plug-in, tất cả ó và Connection.

+ Mỗi plug-in sẽ được kiểm tra với tất cả mọi trình duyệnt và mội OS và connection

+ Mỗi os sẽ được kiểm tra với tất cả browser, plug-in, và connection

+ Mỗi connection sẽ được kiểm tra với tất cả browser, plugin, Os

2.3.2 Thứ tự tham số (In parameter order)

Tai và Lei đã đưa ra một thuật toán được gọi là IP0, để tạo ra các testsuite cho pairwise testing của các biến đầu vào.

Thuật toán:

Input: Tham số $p_1, p_2, p_{3.....}, p_{i,..}, p_n$ với i =1,2,3..n;

và $D(p_i) = \{v_1, v_2, v_3, v_4, \ldots v_q\}$

Out put: Một bộ test suite T thỏa mãn coverage pairwise

Tóm tắt thuận toán:

```
Strategy In-Parameter-Order
begin
   /* for the first two parameters p1 and p2 */
   T := {(v1, v2) | v1 and v2 are values of p1 and p2, respectively}
   if n = 2 then stop;
   /* for the remaining parameters */
   for parameter pi, i = 3, 4, …, n do
   begin
      /* horizontal growth */
      for each test (v1, v2, …, vi-1) in T do
         replace it with (v1, v2, …, vi-1, vi), where vi is a value of pi
      /* vertical growth */
      while T does not cover all pairs between pi and
            each of p1, p2, …, pi-1 do
         add a new test for p1, p2, …, pi to T;
   end
end
```

Hình 2.4 Thuật toán IPO.

```
Algorithm IPO_H(T, p_i)
// T is a test set. But T is also treated as a list with elements in arbitrary order.
{ assume that the domain of p_i contains values v_1, v_2, ..., and v_q;
  π = { pairs between values of p_i and values of p_1, p_2, ..., and p_{i-1} };
  if (|T| ≤ q)
  { for 1 ≤ j ≤ |T|, extend the jth test in T by adding value v_j and
    remove from π pairs covered by the extended test;
  }
  else
  { for 1 ≤ j ≤ q, extend the jth test in T by adding value v_j and
    remove from π pairs covered by the extended test;
    for q < j ≤ |T|, extend the jth test in T by adding one value of p_i
    such that the resulting test covers the most number of pairs in π, and
    remove from π pairs covered by the extended test;
  }
}
```

Hình 2.5 Thuật toán Horizontal growth

```
Algorithm IPO_V(T, π)
{ let T' be an empty set;
  for each pair in π
  { assume that the pair contains value w of p_k, 1 ≤ k < i, and value u of p_i;
    if (T' contains a test with "−" as the value of p_k and u as the value of p_i)
      modify this test by replacing the "−" with w;
    else
      add a new test to T' that has w as the value of p_k, u as the value of p_i,
      and "−" as the value of every other parameter;
  };
  T = T ∪ T';
};
```

Hình 2.6 Thuật toán vertical

Các bước cụ thể của thuật toán:

Bước 1: Với 2 tham số đầu vào p_1 và p_2 tạo ra test suite

T={(v_1,v_2)|v_1 và v_2 theo thứ tự là những giá trị của của p_1 và p_2}

Bước 2: Nếu i =2, ngừng. Còn không với i = 3,4,,, n sẽ lặp lại bước 3 và bước 4

Bước 3: Cho $D(p_i)$ ={v_1,v_2,v_3,v_4,…v_q}

Tạo cặp π_i ={cặp giữa các giá trị của p_i và tất cả các giá trị của p_1,p_2, p_3 …p_{i-1}}

Nếu |T| <=q thì

Xét j chạy từ 1→|T|(1<=j<=|T|), mở rộng kiểm thử thứ j trong T bằng cách thêm vào giá trị v_j và di chuyển từ π_i cặp đôi đã phủ bởi kiểm thử đã mở rộng.

Còn nếu không (|T| > q)

- Xét với j ∈[1;q], mở rộng phần tử thứ j trong T bằng cách thêm thêm vào giá trị v_j và remove từ π_i cặp đã được cover bởi kiểm thử mở rộng này.

- Còn với j ∈ (q; |T|) mở rộng kiểm thử thứ j trong T bằng cách thêm vào một giá trị(v_j) của p_i, giả sử rằng kết quả kiểm thử covers được hầu hết số lượng của cặp trong π_i, và remove từ π_i cặp đã được cover bởi phần tử mở rộng.

Bước 4:

Hãy gán cho T'=Φ(tập rỗng) và $|\pi_i| > 0$;

Với mỗi cặp trong π_i cặp chứa giá trị w của p_k, $1 \le k < i$, và những giá trị u của p_i) hãy làm:

- Nếu T' chứa một kiểm tra với – như là gí trị của P_k và u như là giá trị của p_i), thay đổi kiểm thử này và thay thế - bằng w

- Còn nếu không

+ Thêm vào một kiểm thử mới trong T', cái có w như là giá trị của p_k, u như là giá trị của P_i, và – như là giá trị của tất cả các tham số khác.

- T:= T ∪T'.

Minh họa thuật toán IPO:

Áp dụng với hệ thống S với 3 biến đầu vào ở trên là x,y,z và

D(x) = {True,False}

D(y) = {0;5}

D(z) = {Q;R}

Bước 1: Tạo ra một testsuite gồm 4 testcase tương ứng cho 2 tham số đầu tiên là x và y.

$$T = \begin{bmatrix} (\text{True}, & 0) \\ (\text{True}, & 5) \\ (\text{False}, & 0) \\ (\text{False}, & 5) \end{bmatrix}$$

Bước 2: Có 3 biến (p) là x,y,z nên i = 3>2, tiếp tục thực hiện bước 3 và 4 với i bằng 3.

Bước 3 Với biến thứ 3 là z, có D(z) = {P,Q,R}" *[D(p_i) =$\{v_1,v_2,v_3,v_4,\ldots v_q\}$]" vậy q(z) = 3* tạo tập π_3 ={ cặp giữa giá trị của z và x,y};

Ta có:

$$\pi_3 = \begin{bmatrix} (\text{True}, P), & (\text{True}, Q), & (\text{True}, R) \\ (\text{False}, P), & (\text{False}, Q), & (\text{False}, R) \\ (0, P), & (0, Q), & (0, R) \\ (5, P), & (5, Q), & (5, R) \end{bmatrix}$$

Và D(p_i)={P,Q,R} → q =3; T =4; T>q;

Xét với j=[1;T], mở rộng kiểm thử thứ j trong T bằng cách thêm giá trị v_j (v_j) là lần lượt là các giá trị của p_3 là z (P,Q,R) ta được 3 cặp (True;0;P); (True;5;Q); (False;0;R) và T sẽ là 4 cặp tương ứng sau đây:

$$T = \begin{bmatrix} (\text{True}, & 0, & P) \\ (\text{True}, & 5, & Q) \\ (\text{False}, & 0, & R) \\ (\text{False}, & 5, &) \end{bmatrix}$$

Remove từ π_i những cặp đã được phủ bởi 3 cặp trên đó là { True;P};{True;Q}(False;R);(0;P);(0;Q);(0;R) từ tập π_3 đi:

$$\pi_3 = \begin{bmatrix} & & (\text{True}, R) \\ (\text{False}, P), & (\text{False}, Q) & \\ & (0, Q) & \\ (5, P), & (5, R) & \end{bmatrix}$$

Với q<j<=T, tức là với j = 4; mở rộng kiểm thử j trong T bằng cách thêm một giá trị của p_i, giả sử rằng kết quả kiểm thử covers hầu hết số trường hợp(nhiều nhất có thể) của cặp trong π_i và remove từ π_i cặp đã được phủ bởi phần tử mở rộng;

Ở đây chúng ta đang muốn xét thêm giá trị nào trong 3 giá trị P,Q,R trong z vào cặp (False;5;…;) của T

Vấn đề là lựa chọn giá trị nào trong 3 cái là P;Q;R; Hãy xét cái nào có thể cover được nhiều nhất trong tập π_3;

Nếu chúng ta chọn P; tương ứng sẽ có T_4(False;5;P) sẽ cover được 2 cặp từ π3;(false;p) và (5;p);

Nếu chúng ta chọn Q; Tương ứng là T_4(False;5; Q) sẽ cover được 1 cặp từ π3 là (False;Q) .

Nếu chúng ta chọn R; Tương ứng là T_4(False;5; R) sẽ cover được 1 cặp từ π3 là (5;R)

Vậy chúng ta sẽ chọn giá trị P Tương ứng là T_4(False;5; P) và sẽ cover được 2 cặp từ π3 là ;(false;p) và (5;p);

Sau khi remove 2 cặp đã phủ trong T ta sẽ có π_3 cuối cùng là

$$T = \begin{bmatrix} (\text{True}, & 0, & P) \\ (\text{True}, & 5, & Q) \\ (\text{False}, & 0, & R) \\ (\text{False}, & 5, & P) \end{bmatrix} \qquad \pi_3 = \begin{bmatrix} (\text{True}, R) \\ (\text{False}, Q) \\ (0, Q) \\ (5, R) \end{bmatrix}$$

Bước 4:

T'=Φ(tập rỗng) và $|\pi_i|$ =4> 0;

- Với cặp đầu tiên trong $\pi_{i\,là}$(True,R), với giá trị True của x, và giá trị R của z,

Thuật toán sẽ tạo ra T'={(True,--,R),

Thay thế -- bằng giá trị 0, or 5. Ta sẽ có tương ứng là (True,0,R) or (True,5,R)

- Với cặp 2 (False, —,Q) ta sẽ có (False, 0,Q) or (False, 5,Q)
- Với cặp thứ 3 ta sẽ có (0, Q) vậy sẽ thay bằng (--,0, Q) và sẽ thay thế giá trị bằng (True, 0, Q) or (False, 0, Q)
- Với cặp thứ 4: (5,R) sẽ thay bằng (--,5,R) và sẽ thay bằng (True, 5,R) or (False, 5,R)

Vậy T cuỗi cùng sẽ có các cặp như sau:
(True,0,R) or *(True,5,R)* đã bao hàm cặp cuối cùng.
(False, 0,Q) or (False, 5,Q) có thể bao hàm được
(True, 0, Q) or (False, 0, Q)
(True, 5,R) or (False, 5,R)
Vậy cuối cùng ta sẽ chọn được 2 cặp *(True,5,R); (False, 0,Q)*
Hợp T và T' tạo ra

$$T = \begin{bmatrix} (\text{True}, & 0, & P) \\ (\text{True}, & 5, & Q) \\ (\text{False}, & 0, & R) \\ (\text{False}, & 5, & P) \\ (\text{False}, & 0, & Q) \\ (\text{True}, & 5, & R) \end{bmatrix}$$

2.4 Công cụ PICT.(Pairwise Independent Combinatorial Testing)

Trên đây chúng ta đã tìm hiểu về kỹ thuật pairwise, với 2 kỹ thuật chính là OA và IPO. Chúng ta hoàn toàn có thể chạy bằng tay với 2 kỹ thuật này để sinh ra bộ dữ liệu kiểm thử. Và tại phần này tôi sẽ giới thiệu một công cụ cho phép sinh ra bộ dữ liệu kiểm thử phát triển theo ý tưởng của pairwise. Đó là PICT.

2.4.1 Nguyên tắc thiết kết của PICT:

PICT được thiết kế dựa trên 3 nguyên tắc trọng tâm là:

Nguyên tắc 1: Tốc độ của việc generation test.(speed of test generation)

Nguyên tắc 2: Dễ sử dụng

Nguyên tắc 3: Mở rộng của các công cụ cốt lõi khác.

2.4.2 File đầu vào của PICT:

Đầu vào cho PICT là một tập tin đơn giản. Tập tin này có ít nhất 1 thành phần và nhiều nhất 3 thành phần như sau:

parameter definitions [sub-model definitions] [constraint definitions]

Tập tin luôn được định nghĩa theo thứ tự ở trên và không được trùng lặp. Viết chú thích bằng ký tự # ở đầu dòng. Nó có thành phần mở rộng là .txt
Ví dụ1 về nội dung file đầu vào của PICT:

TYPE: Primary (10), Logical, Single, Span, Stripe, Mirror, RAID-5

SIZE: 10, 100, 500, 1000, 5000, 10000, 40000
FORMAT: quick, slow
FSYSTEM: FAT, FAT32, NTFS (10)
CLUSTER: 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536
COMPRESSION: on, off

Vi du 2:

This is a sample model for different dates
#Negative Testing
Hour: ~0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ~13, ~14, ~15, ~16, ~17, ~18, ~19, ~20, ~21, ~22, ~23
AMPM: ~AM, ~PM, 24HR
#the numbers in parenthesis symbolise weight
#Weights can force PICT to prefer certain values
#Weights are positive integers
#One takes precedence over two
#Weights may not always be honoured because PICT must handle two contradictory requirements
#Cover all combinations with the least number of test cases
Choosing values according to their weights
Day: Monday(2), Tuesday(1), Wednesay(16), Thursday, Friday, Saturday, Sunday (99)
Month: January, February, March, April, May, June, July, August, September, October, November, December

a. Cách viết phần: [parameter definitions]

Mỗi tham số và giá trị được viết trên một dòng, và các giá trị được phân cách bằng dấu phẩy [,]

<ParamName>: <Value1>, <Value2>, <Value3>, ...

Dấu [:] ngăn cách giữa biến và giá trị.

b. Cách viết phần: [sub-model definitions]

Cho phép bạn gộp một số tham số vào nhóm . Điều này hữu ích nếu sự kết hợp của các tham số cần được kiểm tra kỹ lưỡng hơn or là nó cần được tách ra khỏi các thông số khác trong mô hình.

{<ParamName1>, <ParamName2>, <ParamName3>, ... } @ <Order>

Một ví dụ hữu ích khi tham số phần cứng và phần mềm được kết hợp với nhau. Bình thường g, mỗi test case sẽ có thể tạo ra một cấu hình phần cứng riêng. Đặt tất cả các tham số phần cứng vào sup model sẽ sinh ra ít các cấu hình phần cứng khác nhau và có khả năng làm giảm các chi phí test.

```
PLATFORM:  x86, ia64, amd64
CPUS:      Single, Dual, Quad
RAM:       128MB, 1GB, 4GB, 64GB
HDD:       SCSI, IDE
OS:        NT4, Win2K, WinXP, Win2K3
IE:        4.0, 5.0, 5.5, 6.0
APP:       SQLServer, Exchange, Office
{ PLATFORM, CPUS, RAM, HDD } @ 3
{ OS, IE } @ 2
```

c. Cách viết phần: [Constraint definitions]

Constraint gồm có 2 loại là conditional (IF-THEN-ELSE) and unconditional.

Conditional Constraints:

Mối liên hệ giữa tham số và giá trị là một phần nguyên tử của một predicate. Các mối quan hệ sau đây được sử dụng như là: =,<>, >, <=,<,<=, và like. Like trong đó có *(khớp toàn bộ) và ?(một ký tự)

```
[Size] < 10000
[Compression] = "OFF"
[File system] like "FAT*"
```

Toán tử In được sử dụng để xác định một tập các giá trị mà thỏa mãn quan hệ xác định nào đó.

```
IF [Cluster size] in {512, 1024, 2048} THEN [Compression] = "Off";
IF [File system] in {"FAT", "FAT32"} THEN [Compression] = "Off";
```

IF, Then, ELSE có thể chứa nhiều term được joined bởi các toán tử OR, AND, NOT. Dấu (được sử dụng để đặt lại độ ưu tiên.

```
IF [File system] <> "NTFS" OR
 ( [File system] =  "NTFS" AND [Cluster size] > 4096 )
THEN [Compression] = "Off";
IF NOT ( [File system] = "NTFS" OR
     ( [File system] = "NTFS" AND NOT [Cluster size] <= 4096 ))
THEN [Compression] = "Off";
```

Parameters có thể được so sánh với tham số khác, như là ví dụ

```
#
# Machine 1
#
OS_1:   Win2000, WinXP
SKU_1:  Professional, Server, Datacenter, WinPowered
LANG_1: EN, DE

#
# Machine 2
#
OS_2:   Win2000, WinXP
SKU_2:  Professional, Server, Datecenter
LANG_2: EN, DE

IF [LANG_1] = [LANG_2]
THEN [OS_1] <> [OS_2] AND [SKU_1] <> [SKU_2];
```

Unconditional Constraints (Invariants)

Xác định giới hạn hợp lệ của một miền

2.4.3 Cách thức sinh test case của PICT.

Quá trình xử lý trong PICT được thể hiện qua hai giai đoạn chính: preparation và generation.

Trong giai đoạn 1, PICT sẽ tính toán tất cả các thông tin cần thiết cho giai đoạn sau. Điều này bao gồm thiết lập tập P của tất cả các tham số tương tác để cover. Mỗi một sự kết hợp của giá trị có thể có của cặp dữ liệu của tập biến đầu vào được cover được phản ánh trong cấu trúc tương tác (in a parameter-interaction structure.)

Ví dụ, có 3 tham số A, B và C. A, B có 2 giá trị. Và c có 3 giá trị. Và pairwise sẽ tạo ra 3 cấu trúc tương tac tham số (parameter-interaction structures) là AB, AC và BC. Mỗi cấu trúc sẽ có một số slots, mỗi slots tương ứng với kết hợp giá trị có thể có. 4 Slots cho AB, 6 slót cho AC và BC.

```
A: 0, 1
B: 0, 1        translates to
C: 0, 1, 2
```

AB	AC	BC
00	00	00
01	01	01
10	02	02
11	10	10
	11	11
	12	12

Mỗi slot có thể được đánh dậu là uncovered, covered or là excluded(loại trừ). Tất cả các slot uncovered trong tất cả các tham số tương tác tạo thành tập của sự kết

hợp được covered. Nếu bất kỳ ràng buộc được định nghĩa trong một mô hình chúng được chuyển đổi thành tập exclusions tập mà giá trị kết hợp ở đó phải không được xuất hiện trong output cuối cùng. Slot trở thành covered khi thuật toán sinh ra các test case thỏa mãn sự kết hợp riêng(đặc biệt). Thuật toán kết thúc khi không có một slots nào không được cover.

Trong giai đoạn 2, sinh test case sẽ sử dụng thuật toán sinh ca kiểm thử là greedy heuristic. Nó xây dựng một ca kiểm thử ở một thời điểm và với giải pháp tối ưu hóa.

Thuật toán không giả định bất kỳ cái gì về sự kết hợp có thể được cover. Nó thao tác trên danh sách sự kết hợp cái mà xuất ra trong giai đoạn chuẩn bị.

Sự linh hoạt của thuật toán cho phép thêm những thông tin thú vị một cách dễ dàng. Thuật toán cũng khá hiệu quả. Và cũng khá nhanh để đạt được mục đích.

```
# Assume test cases r_1,...,r_{i-1} are already produced
# Slots in P reflecting combinations selected by r_1,...,r_{i-1} are set to covered

If there are any unused seed combinations not violating any exclusions
    Add a seed combination to r_i
    Mark all slots in P covered by the seed combination as covered

While there are parameters with no values in r_i

    If r_i is empty
        Choose a parameter interaction p from I' with most uncovered slots
        Pick the first uncovered combination from p
    Else
        # Assume values l_1,...,l_{k-1} have already been chosen and added to r_i

        Look at subset Q of P that covers at least one parameter with no
         representation in l_1,...,l_{k-1}

        Look at slots in Q which values are consistent with already chosen values in l_1,...,l_{k-1}

        If there exist uncovered combinations
            Pick a slot with values which when added to r_i would cover the most uncovered
             combinations with l_1,...,l_{k-1} and the resulting partial test case r_i would not
             contain an excluded combination
        Else
            Pick randomly a covered combination which when added to l_1,...,l_{k-1} would not
             contain an excluded combination

    Add values of this combination to r_i
    Mark the chosen combination in I' as covered
```

Hình 2. 7 thuật toán sinh ca kiểm thử của PICT

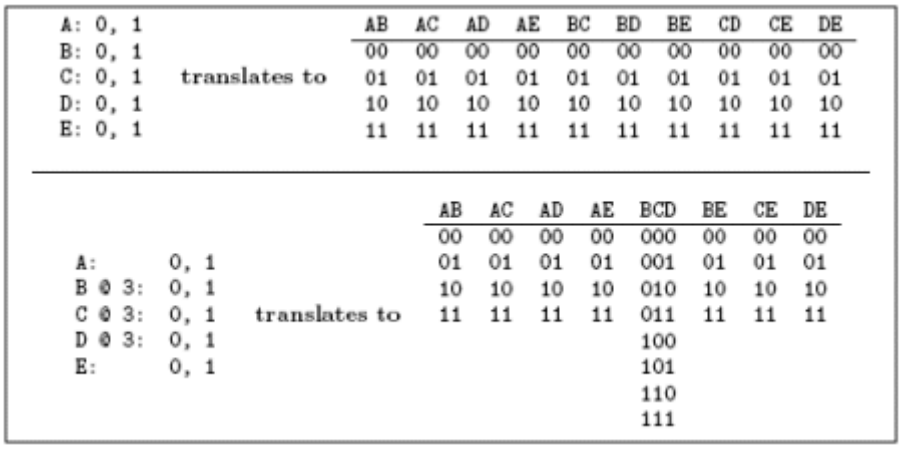
2.4.4 Sự ưu việt của PICT

PICT trở lên mạnh mẽ hơn các tool khác nhờ vào những đặc điểm sau:

a. Tạo ra một sự phối hợp đầy đủ(mạnh) (Mixed Strength Generation)

Có thể kết hợp được nhiều hơn bộ 2 tham số. Ví dụ hoàn toàn có thể kết hợp được theo bộ 3 tham số.

Ví dụ, sự tương tác của các tham số B, C, D có thể yêu cầu một sự cover tốt hơn so với các tương tác A or E. Chúng ta có thể tạo ra tất cả sự kết hợp bộ 3 có thể có của B, C, D và cover tất cả các cặp của tham số khác còn lại.



Hình 2.8 Cấu trúc tương tác tham số được tạo ra của PICT.

Đặc điểm này được thể hiện như thế nào: chính là cách thể hiện của đặc điểm b.

b. Tạo phân cấp tham số (Creating a Parameter Hierarchy)

Để bổ sung cho các đặc điểm trên, PICT cho phép người sử dụng tạo phân cấp cho các tham số. Đây là một kỹ thuật hữu ích có thể được sử dụng trong mô hình test domain với sự phân cấp rõ ràng của tham số test.

Tasumi, khi mô tả về quá trình phân tích tham só test, phân biệt giữa tham số đầu vào và tham số môi trường. Thông thường, các tham số đầu vào có thể được kiểm soát và thiết lập dễ dàng hơn nhiều so với tham số môi trường. Do đó đôi khi mong muốn để hạn chế số lượng của môi trường tới mức tối thiểu.

Hãy tham khảo ví dụ sau:

Để cover tất cả sự kết hợp có thể có của 9 tham số, PICT tạo ra 31 case test, trong đó có 17 case sự kết hợp của các tham số phần cứng khác nhau. Là platform, cpu, ram. Thay vào đó các thông số phần cứng có thể được thiết kế là sub- model thì sẽ tạo ra 54 case nhưng trong đó chỉ có 9 case có sự kết hợp độc đáo các tham số phần cứng.

Test domain consisting of ‘input’ and ‘environment’ parameters:
Input parameters Type: Single, Spanned, Striped, Mirror, RAID-5
Size: 10, 100, 1000, 10000, 40000
Format method: Quick, Slow
File system: FAT, FAT32, NTFS
Cluster size: 512, 1024, 2048, 4096, 8192, 16384
Compression: On, Off
Environment parameters
Platform: x86, x64, ia64
CPUs: 1, 2
RAM: 1GB, 4GB, 64GB
Environment parameters will form a sub-model
{ PLATFORM, CPUS, RAM } @ 2

c. Loại trừ sự kết hợp không mong muốn (Excluding Unwanted Combinations)

Định nghĩa của pairwise testing cho chúng ta biết một điều là là các biến tham số phải là độc lập (independent) . Điều này thật hiếm trong thực tế. Đó là lý do tại sao ràng buộc là một tính năng không thể thiếu của một công cụ tạo ra bộ dữ liệu kiểm thử. Chúng mô tả hạn chế của những miền kiểm thử, rằng là sự kết hợp này là không được.

PICT sử dụng một ngôn ngữ tương tự như chính ràng buộc đó làm rule. Đó chính là bộ ba IF Then ElSE.

PICT dịch những ràng buộc vào trong một tập hợp được gọi là exclusion và sử dụng chúng để đánh dấu slots phù hợp là excluded trong cấu trúc tương tác tham số.

Phương pháp này đặt ra hai vấn đề thực tế

1. Làm thế nào để đảm bảo rằng tất cả các kết hợp mà cần phải được loại trừ trong thực tế được đánh dấu loại trừ.
2. Làm thế nào để xử lý loại trừ đó là chi tiết hơn

Điều đầu tiên có thể được giải quyết bằng cách tính toán những loại trừ độc lập. xem xét ví dụ trong hình 2.9 . Ràng buộc trong ví dụ này tảo ra một sự mâu thuẫn. A= 0 thì A =1. Vậy nên tất cả mọi sự kết hợp có A=0 đều phải được loại trừ.

Input:

```
A: 0, 1
B: 0, 1
C: 0, 1

IF [A] = 0 THEN [B] = 0;
IF [B] = 0 THEN [C] = 0;
IF [C] = 0 THEN [A] = 1;
```

Before and after calculating dependent exclusions:

```
A:0, B:1
B:0, C:1    ⇒    A:0
A:0, C:0         B:0, C:1
```

Before and after excluding dependent combinations:

AB	AC	BC
00	~~00~~	00
~~01~~	01	~~01~~
10	10	10
11	11	11

⇒

AB	AC	BC
~~00~~	~~00~~	00
~~01~~	~~01~~	~~01~~
10	10	10
11	11	11

Hình 2.9 Ví dụ về ràng buộc trong PICT

```
Type:           Single, Spanned, Striped, Mirror, RAID-5
Size:           10, 100, 1000, 10000, 40000
Format method:  Quick, Slow
File system:    FAT, FAT32, NTFS
Cluster size:   512, 1024, 2048, 4096, 8192, 16384
Compression:    On, Off

# There are limitations on volume size

IF [File system] = "FAT"   THEN [Size] <= 4096;
IF [File system] = "FAT32" THEN [Size] <= 32000;

# And not all file systems support compression

IF [File system] <> "NTFS" or
  ([File system] =  "NTFS" and [Cluster size] > 4096)
THEN [Compression] = "Off";
```

Hình 2.10 Nội dung file có chức năng ràng buộc trong PICT.

Điều thứ 2, hãy xem ví dụ 2.11, trong đó có loại trừ 3 phần tử được tạo ra, nhưng các cấu trúc tham số tương tác chỉ đề cập đến 2 tham số tại một thời điểm. Nói cách khác, không có một cấu trúc tương tác ABC để đánh dấu loại trừ sự kết hợp của AB, AC hay BC.

Input:

```
A: 0, 1
B: 0, 1
C: 0, 1
D: 0, 1

IF [A] = 0 AND [B] = 0 THEN [C] = 0;
```

Exclusions:

```
A:0, B:0, C:1
```

Before and after adding new parameter interaction ABC:

AB	AC	AD	BC	BD	CD
00	00	00	00	00	00
01	01	01	01	01	01
10	10	10	10	10	10
11	11	11	11	11	11

⇒

AB	AC	AD	BC	BD	CD	ABC
00	00	00	00	00	00	000✓
01	01	01	01	01	01	~~001~~
10	10	10	10	10	10	010✓
11	11	11	11	11	11	011✓
						100✓
						101✓
						110✓
						111✓

Hình 2.11. Ràng buộc 3 biến.

d.Seeding file: Cho phép nhìn thấy file luôn trên command

Seeing có hai ứng dụng tham ra.

1. Nó cho phép xác định tường minh những kết hợp quan trọng. Tính trực quan.
2. Nó có thể được sử dụng để giảm thiểu sự thay đổi ở đầu ra khi mà mô tả

Test parameters:

```
Type:           Single, Spanned, Striped, Mirror, RAID-5
Size:           10, 100, 1000, 10000, 40000
Format method:  Quick, Slow
File system:    FAT, FAT32, NTFS
Cluster size:   512, 1024, 2048, 4096, 8192, 16384
Compression:    On, Off
```

Seeding file:

Type	Size	Format method	File system	Cluster size	Compression
Single	100	Quick	NTFS	4096	Off
Mirror		Slow	FAT32		

e. Có thể test với các giá trị phủ định (Testing with Negative Values)

Ngoài việc test tất cả những sự kết hợp hợp lệ, người ta thường sử dụng để kiểm tra giá trị ngoài phạm vi cho phép để đảm bảo phần mềm xử lý các điều kiện lỗi đúng. Trong ví dụ 12, một số a or b bất kỳ là số thực hợp lệ. tuy nhiên nó chỉ có ý nghĩa khi a và b đều lớn hơn or bằng không. Vì lý do đó cần phải xử lý trước khi tính toán.

PICT cho phép đánh dấu giá trị không hợp lệ trong hình 12, các đầu ra của mô hình phải có thuộc tính sau.

1. Tất cả giá trị valid sẽ được kết hợp với tất cả giá trị hợp lệ khác trong trường hợp là positive.

2. Nếu một case chứa giá trị invalid thì chỉ có một giá trị như vậy.

3. Tất cả những giá trị không hợp lệ được kết hợp với tất cả các giá trị valid được gọi là negative test case.

Việc thực hiện thực tế của PICT sử dung 2 giai đoạn. Đầu tiên, trên các tham số test

Routine to be tested:

```
float SumSquareRoots( float a, float b )
{
    if ( a < 0 ) throw error;
    if ( b < 0 ) throw error;

    return ( sqrt( a ) + sqrt( b ));
};
```

Sample model with *invalid* values marked with '*':

```
A: *LessThanZero, Zero, GreaterThanZero
B: *LessThanZero, Zero, GreaterThanZero
```

Resulting test cases:

A	B	
Zero	GreaterThanZero	(positive)
Zero	Zero	(positive)
GreaterThanZero	Zero	(positive)
GreaterThanZero	GreaterThanZero	(positive)
*LessThanZero	GreaterThanZero	(negative)
*LessThanZero	Zero	(negative)
GreaterThanZero	*LessThanZero	(negative)
Zero	*LessThanZero	(negative)

g. Thiết lập kết quả mong muốn (Specifying Expected Results)

Sample model for int Sum(int[] Array, int Start, int Count) with expected results specified:

```
# Input parameters:

Array:  *Null, Empty, Valid
Start:  *TooLow, InRange, *TooHigh
Count:  *TooFew, Some, All, *TooMany

# Result parameters:

$Result:  Pass, OutOfBounds, InvalidPointer

# Result rules:

IF [Array] IN {"Empty", "Valid"} AND
   [Start] IN {"InRange"}        AND
   [Count] IN {"Some", "All"}
THEN [$Result] = "Pass";

IF [Array] = "Null"
THEN [$Result] = "InvalidPointer";

IF [Start] IN {"TooLow", "TooHigh"} OR
   [Count] IN {"TooFew", "TooMany"}
THEN [$Result] = "OutOfBounds";
```

Test cases contain input data and expected results:

Array	Start	Count	$Result
Empty	InRange	All	Pass
Valid	InRange	Some	Pass
Valid	InRange	All	Pass
Empty	InRange	Some	Pass
Empty	InRange	*TooMany	OutOfBounds
Empty	*TooHigh	All	OutOfBounds
Valid	InRange	*TooMany	OutOfBounds
Empty	InRange	*TooFew	OutOfBounds
*Null	InRange	All	InvalidPointer
Empty	*TooLow	Some	OutOfBounds
Valid	InRange	*TooFew	OutOfBounds
*Null	InRange	Some	InvalidPointer
Valid	*TooHigh	Some	OutOfBounds
Valid	*TooLow	All	OutOfBounds

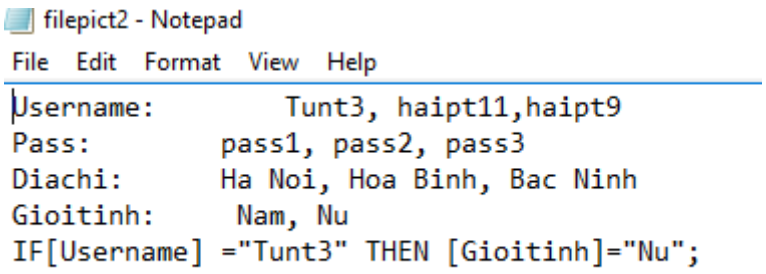
2.4.5 Cài đặt và chạy PICT

1. Download tịa link: http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/PICT33.msi

2.Thực hiện cài đặt

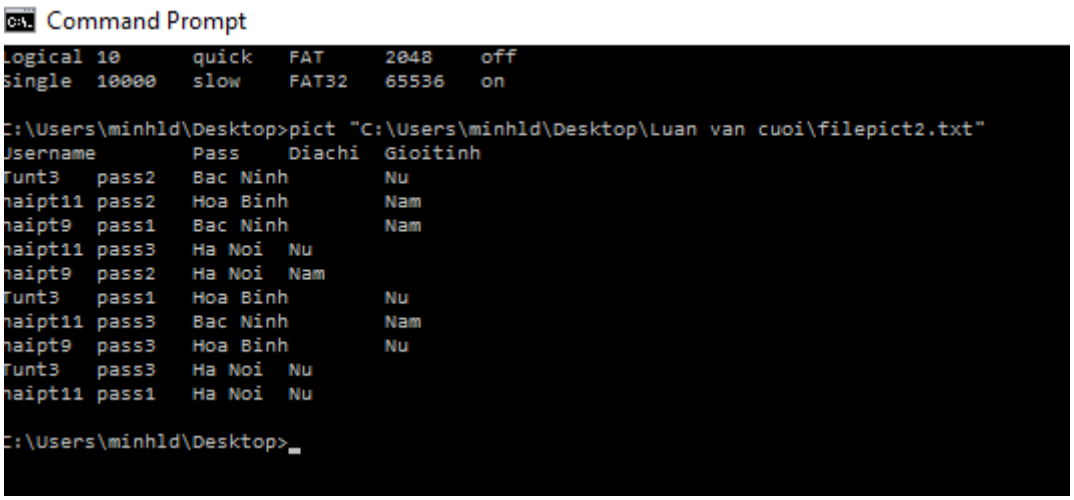
3.Từ Run ->CMD.

4.Trỏ đường dẫn đến thư mục đặt file để chạy PICT

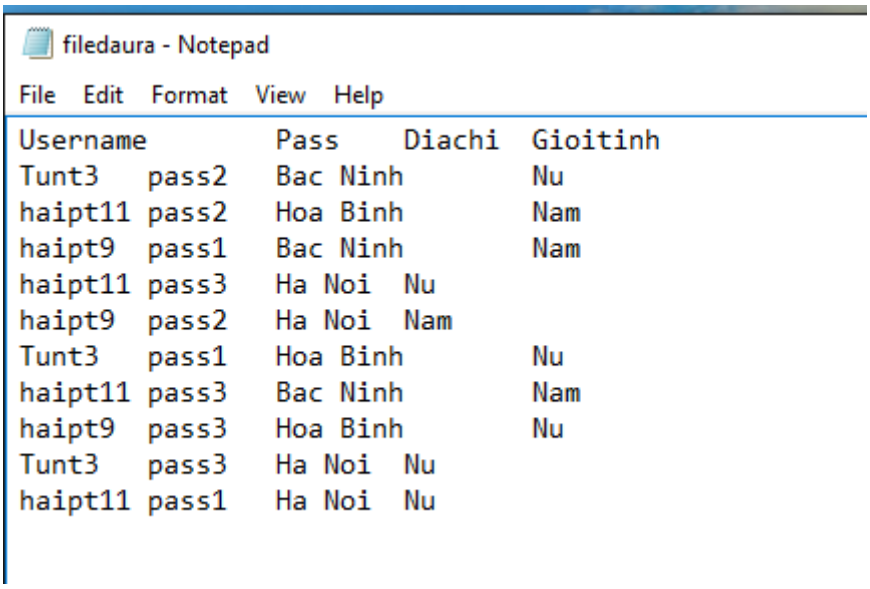


5. PICT vidu.txt > filedaura.txt

Kết quả hiển thị trên command



Kết quả hiển thị tại nơi đặt file:



2.4.6 Ứng dụng của pairwise testing hay cũng chính là PICT .

PICT được sử dụng trong 2 trường hợp sau.

a. Áp dụng viết itc khi viết bằng tay.

Với kỹ thuật mảng trực giao và IPO, chúng ta hoàn toàn có thể tạo ra bộ dữ liệu kiểm thử tốt. Tuy nhiên phải thực hiện chạy bằng tay. Giờ đây chúng ta ko phải nghĩ ngợi, tính toán gì nhiều nữa để đưa ra được các trường hợp kiểm thử. Chỉ việc áp dụng lấy các value và viết các ca kiểm thử. Tiết kiệm thời gian xuy nghĩ và tính toán.

b. Sử dụng làm data đầu vào cho selenium webdriver.

Ví dụ cụ thể như chúng ta hoàn toàn viết được testcase selenium drier để lấy dữ liệu từ file excell và chạy lần lượt các giá trị trong từng row đó.

Lấy dữ liệu từ file excell lên:

File excell có kết quả được xuất từ pairwise

STT	USER NAME	PASSWORD	Result ITC
1	tunthcm@gmail.com	minhanh2929	pass

2	luuminhvu5114@gmail.com	minhanh2929	pass
3	luuminhquyet20102015	minhanh2929	pass
4	..	..	..
5			
6			
7			
8			

Chạy selenium webdriver + apache poi

```
3.  import java.io.File;
4.  import java.io.FileInputStream;
5.  import java.io.IOException;
6.  import java.util.concurrent.TimeUnit;
7.  import org.apache.poi.xssf.usermodel.XSSFCell;
8.  import org.apache.poi.xssf.usermodel.XSSFRow;
9.  import org.apache.poi.xssf.usermodel.XSSFSheet;
10. import org.apache.poi.xssf.usermodel.XSSFWorkbook;
11. import org.openqa.selenium.By;
12. import org.openqa.selenium.WebDriver;
13. import org.openqa.selenium.firefox.FirefoxDriver;
14.
15.
16.     public class ReadExcellfile {
17.         private static WebDriver driver = null;
18.         public static XSSFWorkbook wordbook = null;
19.         public static String filepath= "D:\\DRIVERSELENIUM\\Selenium_java
poi_excell\\src\\testData\\Testdata.xlsx";
20.         public static String sheetname ="sheet1";
21.         public static void main(String[] args) throws IOException,
InterruptedException {
22.
23.                 // Tạo một thể hiện của driver file fox.
24.          driver = new FirefoxDriver();
25.          driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
26.           //Khởi động facebook
27.          driver.get("https://www.facebook.com/");
28.         // Doc file
29.          int rowcount=0;
30.          File file = new File(filepath);
31.                 FileInputStream stream = new FileInputStream(file);
32.                 wordbook = new XSSFWorkbook(stream);
33.                 XSSFSheet sheet = wordbook.getSheet(sheetname);
34.          rowcount = sheet.getLastRowNum();
35.          for (int i =1; i<=rowcount; i++ )
36.          {
37.          XSSFRow  row = sheet.getRow(i);
38.          XSSFCell cell1 = row.getCell(1);
39.          XSSFCell cell2 = row.getCell(2);
40.          String datacell1 = cell1.getStringCellValue();
41.          String datacell2 = cell2.getStringCellValue();
42.        driver.findElement(By.id("email")).sendKeys(datacell1);
43.         // tìm phần tử có id"pass" và input value vào
44.           driver.findElement(By.id("pass")).sendKeys(datacell2);
45.           // supmit lên form.
```

```
        46.
driver.findElement(By.xpath("//form[@id='login_form']/table/tbody/tr[2]/td[3]/label/input")
).click();
        47.                     //Đóng trình duyệt
        48.             System.out.println("Dang den phan row thu "+i);
        49.             driver.quit();
        50.             driver = new FirefoxDriver();
        51.             driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        52.              //Khởi động  facebook
        53.             driver.get("https://www.facebook.com/");
        54.            //driver.findElement(By.id("userNavigationLabel")).click();
        55.          //  Thread.sleep(5000);
        56.            //
driver.findElement(By.xpath("//div[@id='u_q_1']/div/div/div/div/div/ul/li[12]/a/span/span")
).click();
        57.             //driver.findElement(By.id("u_f_2")).click();
        58.                  System.out.println("Ra day chưa");
        59.
        60.                   }
        61.             }
        62.     }
```

Chương 3. XÂY DỰNG CÔNG CỤ SINH CA KIỂM THỬ TỰ ĐỘNG.

3.1 Ý tưởng của bài toán:

Hiện nay đã có một số công cụ hỗ trợ cho việc kiểm thử tự động ứng dụng Web như selenium ide, QTP. Tuy nhiên chúng mang nhiều nhược điểm.

Như Selenium ide chỉ có thể tạo ra các ca kiểm thử khi web đã hình thành. Và sau khi tạo bởi chính nó thì khi chạy lại nó cũng bị dừng ở nhiều chỗ. Nhưng đây là công cụ mà tôi thấy tâm đắc. Tôi mong muốn có thể ứng dụng được nó vào trong công việc thiết kế ca kiểm thử và thực hiện kiểm thử của mình.

Tuy nhiên, selenium ide chỉ có thể tạo ra test case khi trang web đó đã hình thành. Việc tạo ra testcase trước đó là không thể. Hơn nữa trong quá trình đưa ra kiểm thử thì nó cũng thay đổi liên tục. Và việc tạo ra các ca kiểm thử trên selenium ide mất khá nhiều thời gian. Vì vậy tôi mong muốn có thể tạo ra một công cụ có khả năng tự sinh ca kiểm thử dạng selenium ide và ngoài ra còn có được những đặc điểm sau :

+ Được sử dụng lại trong selenium ide.

+ Vẫn đảm bảo được khi webpage thay đổi.

+ Tạo trước khi webpage hình thành.

+ Kết hợp được kỹ thuật pairwise vào trong đó.

Trong luận văn này, tôi xin giới thiệu công cụ kiểm thử tự động được phát triển theo ý tưởng đã đặt đề ra.

3.2 Phân tích bài toán:

Để thực hiện ý tưởng trên, tôi đưa ra một số vấn đề sau:

• Xác định những thành phần trên trang web cần kiểm tra tương tác.

+ Địa chỉ URL của trang cần kiểm thử.

+ Các thành phần trên trang cần kiểm thử như là: Textbox, combobox, button, link, radio button, verify text.

+ Các thành phần trên trang có thể tương tác qua thuộc tính nào chung.

• Việc đứa một biến có nhiều giá trị vào sẽ xử lý thế nào. Xây dựng các thuật toán ra sao, sử dụng thuật toán nào để có thể sinh ra được nhiều ca kiểm thủ cùng một lúc. Làm sao tôi có thể ứng dụng được pairwise vào trong thuật toán này.

Và tôi thấy rằng IPO mà tôi đã nghiên cứu ở trên khá hay. Tôi đã quyết định sử dụng ý tưởng của nó để làm demo cho lần này.

• File selenium ide xuất ra có dạng như thế nào. Nhiều file hay ít file. Đưa ra một testscript cho tất cả hay đưa ra nhiều ca kiểm thử.

Tôi đã thấy rằng trong selenium ide thì mỗi một sự kiện sẽ dược định nghĩa là một dòng gồm có 3 column. Column thứ nhất luôn luôn thể hiện loại sự kiện input, Column thứ 2 chính là các id của sự kiện input đó. Và column thứ 3 chính là value của input đó.

Dòng nào ở trên sẽ được thực hiện trươc, dòng nào ở dưới sẽ thực hiện sau.

Tôi cũng thấy rằng nên đưa ra nhiều ca kiểm thử thay vì 1 script.

3.3 Giải quyết vấn đề.

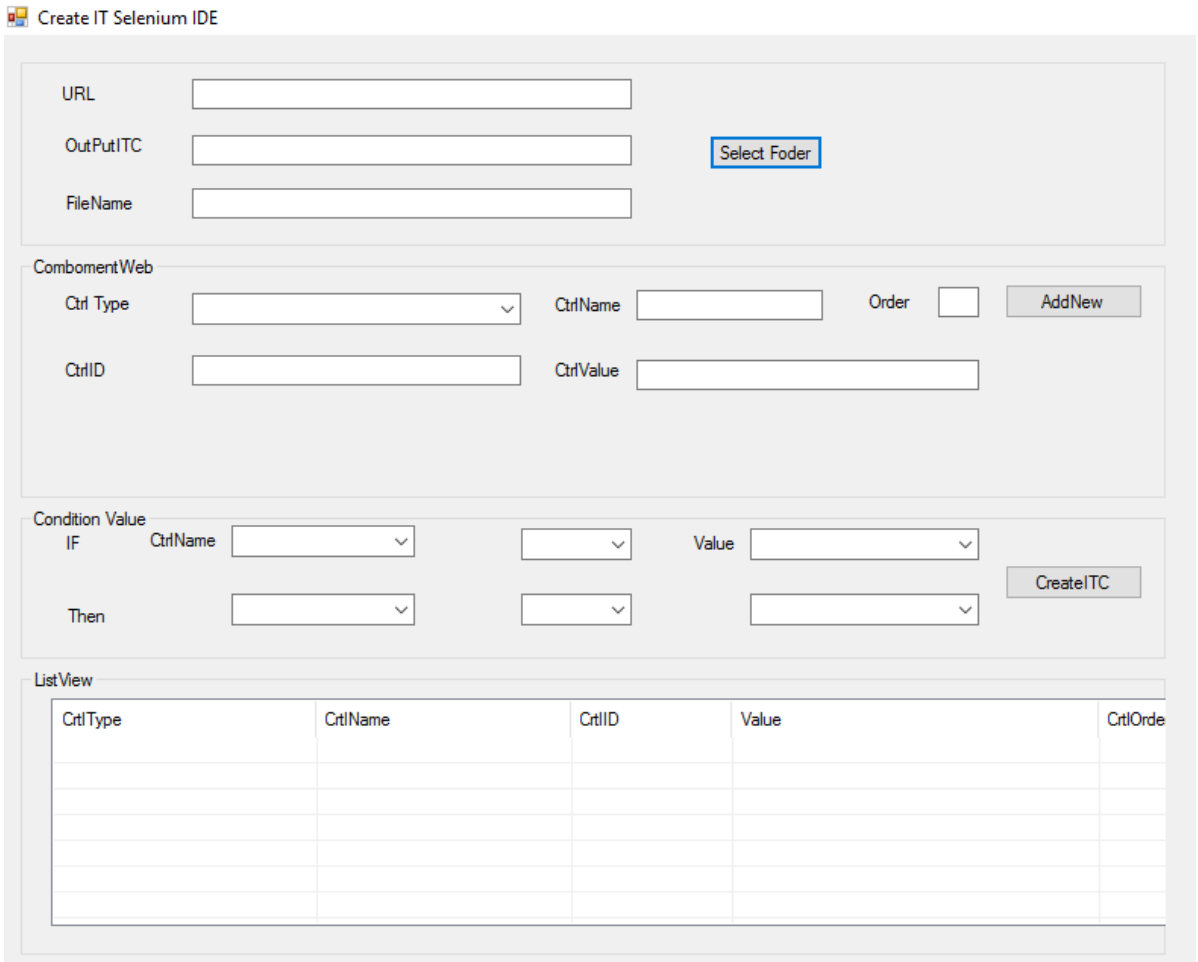
Để giải quyết cho trường hợp mà trên trang có nhiều thành phần, ban đầu tiên tôi đưa ra ý kiến là một thành phần như texbox sẽ cho phép truyền những thuộc tính như name, id, và allvalue. Tôi đưa ra chức năng add textbox cho cái trang nào mà có nhiều textbox.

Và tương tự với button, tôi sẽ cho phép truyền tên button, id của button đó.

- Để giải quyết việc đưa ra thứ tự tôi cho thêm thuộc tính order.
- Về thuật toán áp dung, ban đầu tiên, tôi mong muốn sử dụng ý tưởng của PICT để làm ý tưởng cho thiết kế này. Nhưng tôi thấy IPO khá hay. Và cuối cùng tôi đã chọn nó.
- Tôi thêm một đầu vào cho ràng buộc, dưới dạng đơn giản.

3.3 Giải quyết bài toán.

Từ những phân tích đó tôi tạo ra một form như sau:



Hình 3.1 Form nhập liệu

STT	Tên item trên form	Ý nghĩa	Định dạng nhập
Information web			
1	URL	Nhập URL trang web	Item link web
2	Output ITC	Nhập đường link lưu	
3	Select path	Lựa chọn foder lưu test case	
4	File name	Nhập tên file testcase	Chữ và số tiếng anh
Phần combomenent web			
5	Crtl Type	Loại control	Tiếng anh
6	Crtl Name	Tên control	Tiếng anh
7	Crtl ID	ID của control	Tiếng anh
8	Crtl Value	Value của control	Tiếng anh, ngăn cách dấu [,]
9	Crtl Order	Thứ tự của control	Số
10	Button Add new	Thêm một control vào listview	
Phần Contrains			
11	IF		
12	Crtl name	Tên của control	
13	Crtl Value	Value của control	
14	Condition	Conditon của control	
15	Create ITC	Tạo testcase selenium ide	
16	Then		
Phần List view			
Hiển thị danh sách các control sẽ tạo test case			Hiển thị

Bảng 3.1 Bảng mô tả các item trên form

Ý tưởng thiết kế như vậy, tôi đề suất tách thành 2 giai đoạn code độc lập.

Giai đoạn 1. Tôi thực hiện code thuật toán IPO. Thuật toán này sẽ sinh ra một list các value kết hợp theo thuật toán pairwise .

Kết quả thực hiện giai đoạn 1. Đươc đóng gói là thư viện : IPOlid.

Mã nguồn của thuật toán `HorizontalGrowth`:

```
private static void IpoHorizontalGrowth(TestSet testSet, IpoTestItem testItem,
PairInventory pairInventory)
        {
            // logging
            _logger.Log("START IPO_H");
            _logger.Log(pairInventory);

            int tCount = testSet.testCaseList.Count;
            int q = testItem.values.Length;
            // if |T| <= q
            if (testSet.testCaseList.Count <= testItem.values.Length)
            {
                // for 1 <= j <= |T|
                for (int j = 0; j < tCount; j++)
                {
                    // extend the jth test in T by adding value vj
                    var testCase = testSet.testCaseList[j];
                    testCase.setTestValue(testItem.itemId, testItem.values[j]);
                    // and remove from π pairs coverd by the extended test
                    pairInventory.RemovePairs(testCase, testItem.itemId);

                    // logging
                    _logger.Log(testCase);
                    _logger.Log(pairInventory);
```

```
                }
            }
            else
            {
                // for 1 <= j <= q
                for (int j = 0; j < q; j++)
                {
                    // extend the jth test in T by adding value vj
                    var testCase = testSet.testCaseList[j];
                    testCase.setTestValue(testItem.itemId, testItem.values[j]);
                    // and remove from π pairs coverd by the extended test
                    pairInventory.RemovePairs(testCase, testItem.itemId);

                    // logging
                    _logger.Log(testCase);
                    _logger.Log(pairInventory);
                }
                // for q < j <= T
                for (int j = q; j < tCount; j++)
                {
                    string bestValue = testItem.values[0];
                    int maxRemovable = 0;
                    int removable = 0;
                    var testCase = testSet.testCaseList[j];
                    var tmpTestCase = testCase.Clone();
                    for (int k = 0; k < q; k++)
                    {
                        tmpTestCase.setTestValue(testItem.itemId, testItem.values[k]);
                        removable = pairInventory.PredictRemoveCount(tmpTestCase,
testItem.itemId);
                        if (removable > maxRemovable)
                        {
                            maxRemovable = removable;
                            bestValue = testItem.values[k];
                        }
                    }
                    // extend the jth tesst in T by adding one value of pi
                    // such that resulting test cover the most number pairs in π
                    testCase.setTestValue(testItem.itemId, bestValue);
                    // and remove from π pairs covered by the extended test
                    pairInventory.RemovePairs(testCase, testItem.itemId);

                    // logging
                    _logger.Log(testCase);
                    _logger.Log(pairInventory);
                }
            }
        }
```

Mã nguồn của thuật toán `VerticalGrowth`

```
        private static void IpoVerticalGrowth(TestSet testSet, IpoTestItem testItem,
PairInventory pairInventory)
        {
            // logging
            _logger.Log("START IPO_V");

            IpoTestCase masterTC = testSet.testCaseList[0];

            // let T' be an empty set
            List<IpoTestCase> testCaseList = new List<IpoTestCase>();
            // for each pair in π
            foreach (IpoPair pair in pairInventory.pairList)
            {
                // assume that the pair contains value w of pk, 1<=k<i, and value u of pi;
                // if (T' contains a test with - as the value of pk and u as the value of
pi)
                IpoTestCase findTC = null;
                foreach (IpoTestCase testCase in testCaseList)
                {
                    if (testCase.getTestValue(pair.itemId1) == null &&
pair.itemValue2.Equals(testCase.getTestValue(pair.itemId2)))
                    {
                        findTC = testCase;
                        break;
                    }
                }

                if (findTC != null)
                {
                    // modify this test by replacing the - with w
                    findTC.setTestValue(pair.itemId1, pair.itemValue1);
                }
                else
                {
                    // add a new test to T' that has w as the value of pk, u ass the value
of pi
                    // and - as the value of every parameter
                    IpoTestCase newTestCase = new IpoTestCase();
                    testCaseList.Add(newTestCase);
                    foreach (string itemId in masterTC.itemMap.Keys)
                    {
                        if (itemId.Equals(pair.itemId1))
                        {
                            newTestCase.setTestValue(pair.itemId1, pair.itemValue1);
                        }
                        else if (itemId.Equals(pair.itemId2))
                        {
                            newTestCase.setTestValue(pair.itemId2, pair.itemValue2);
                        }
                        else
                        {
                            newTestCase.setTestValue(itemId, null);
                        }
                    }
                }
            }
            // fill missing value
            foreach (IpoTestCase testCase in testCaseList)
            {
                var itemIdList = testCase.itemMap.Keys.ToList();
                foreach (string itemId in itemIdList)
                {
                    if (testCase.getTestValue(itemId) == null)
                    {
                        testCase.setTestValue(itemId, masterTC.getTestValue(itemId));
                    }
                }
```

```
                // logging
                _logger.Log(testCase);
            }

            // T = T U T'
            testSet.testCaseList = testSet.testCaseList.Union(testCaseList).ToList();
        }
```

Mã nguồn của thuật toán ipo.

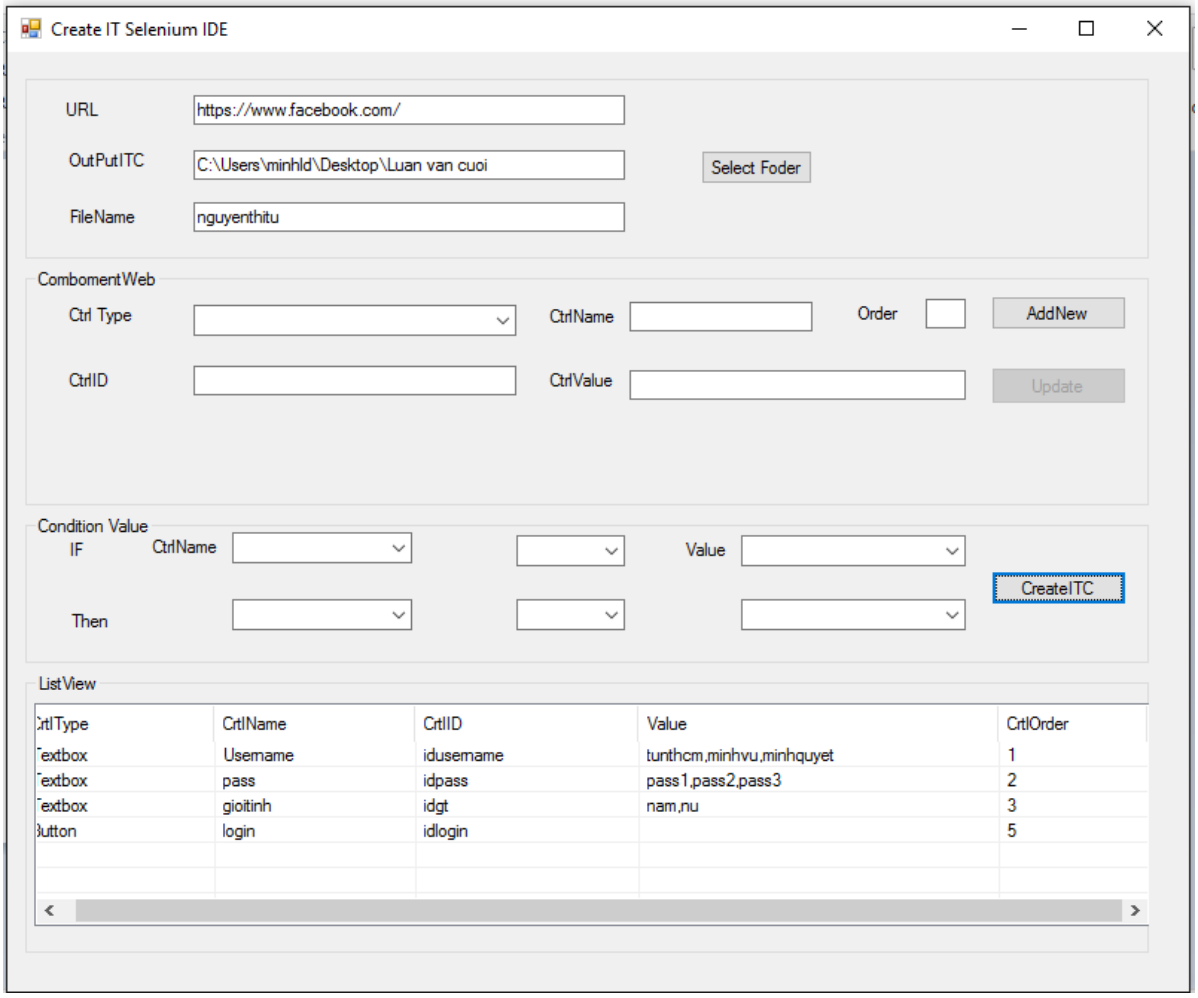
```
public static TestSet CreateTestSet(List<IpoTestItem> testItemList)
        {
            _logger.InitLog();

            TestSet testSet = new TestSet();
            testSet.Add(new IpoTestCase());

            _logger.Log("=========================");
            _logger.Log("START IPO");

            int n = testItemList.Count;

            // begin
            // for the first two parameters p1 and p2
            // T := {(v1, v2) | v1 and v2 are values of p1 and p2, respectively};
            if (n > 0)
            {
                testSet.UpdateTestItem(testItemList[0]);
            }
            if (n > 1)
            {
                testSet.UpdateTestItem(testItemList[1]);
            }

            // if n = 2 then stop;
            if (n > 2)
            {
                // for the remaining parameters
                // for parameter pi, i = 3, 4, ..., n do
                for (int i = 2; i < n; i++)
                {
                    // logging
                    _logger.Log("IPO i=" + i);

                    // begin
                    PairInventory pairInventory = new PairInventory(testItemList, i);
                    // horizontal growth
                    // for each test (v1, v2, ..., vi-1) in T do
                    // replace it with (v1, v2, ..., vi-1, vi), where vi is a value of pi
                    IpoHorizontalGrowth(testSet, testItemList[i], pairInventory);
                    // vertical growth
                    // while T does not cover all pairs between pi and each of p1, p2, ...,
pi-1 do
                    // add a new test for p1, p2, ..., pi to T;
                    IpoVerticalGrowth(testSet, testItemList[i], pairInventory);
                    // end
                }
```

Giai đoạn 2. Lắp ghép thuật toán vào, viết hàm tạo testcase dạng selenium ide.

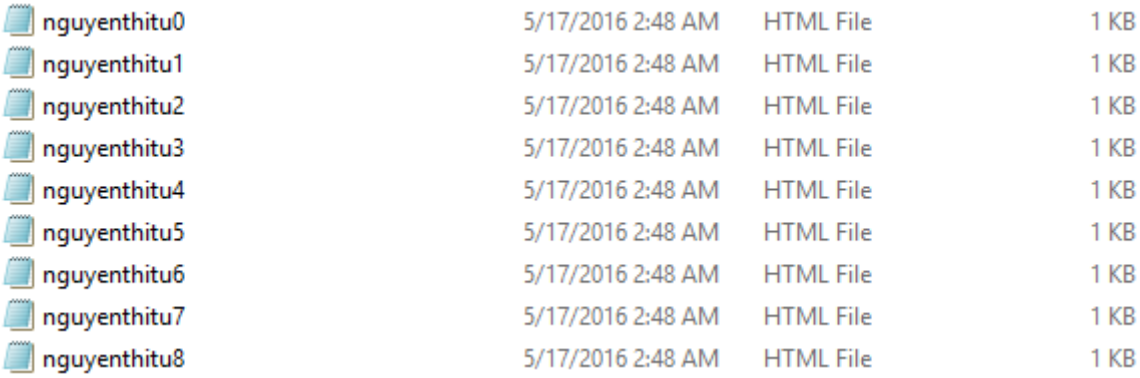
```
private void btnCreateITC_Click(object sender, EventArgs e)

        {
            // xuat file html
            Console.WriteLine("Req 1 --------------");
            // type
            string url = txtUrl.Text;
            string outPut = txtOutPut.Text;
            string fileName = txtFileName.Text;
            if (url == "") {
                MessageBox.Show("ban chua chon url");
                return;
            }
            if (outPut == "")
            {
                MessageBox.Show("ban chua chon outPut");
                return;
            }
            if (fileName == "")
            {
                MessageBox.Show("ban chua chon ten file");
                return;
            }
            List<String> listType = new List<string>();
            List<String> listCtrID = new List<string>();
            List<int> listIndex = new List<int>();
            List<int> listOrder = new List<int>();
            List<IpoTestItem> testItemList = new List<IpoTestItem>();
            int i = 0;
            for (i = grdList.Items.Count - 1; i >= 0; i--)
            {
                testItemList.Add(new IpoTestItem(i.ToString(),
grdList.Items[i].SubItems[3].Text));
                listType.Add(grdList.Items[i].SubItems[0].Text);
                listCtrID.Add(grdList.Items[i].SubItems[2].Text);
                listOrder.Add(int.Parse(grdList.Items[i].SubItems[4].Text));
                listIndex.Add(i);


            }

            try
            {
                TestSet testSet = IPOLib.CreateTestSet(testItemList);
                int k = 0;
                foreach (IpoTestCase testCase in testSet.testCaseList)
                {

                    String html = ExportDatatableToHtml(listOrder, listType,
listCtrID, listIndex, testCase, url);
                    System.IO.File.WriteAllText(outPut + "\\" + fileName + k +
".HTML", html);
                    k++;
                }
                MessageBox.Show("Xuat file html thanh cong !");
            }
            catch (IOException ex) {
                Console.WriteLine("IOException source: {0}", ex.Source);
                MessageBox.Show("Xuat file html ko thanh cong !");
```

```
}
```

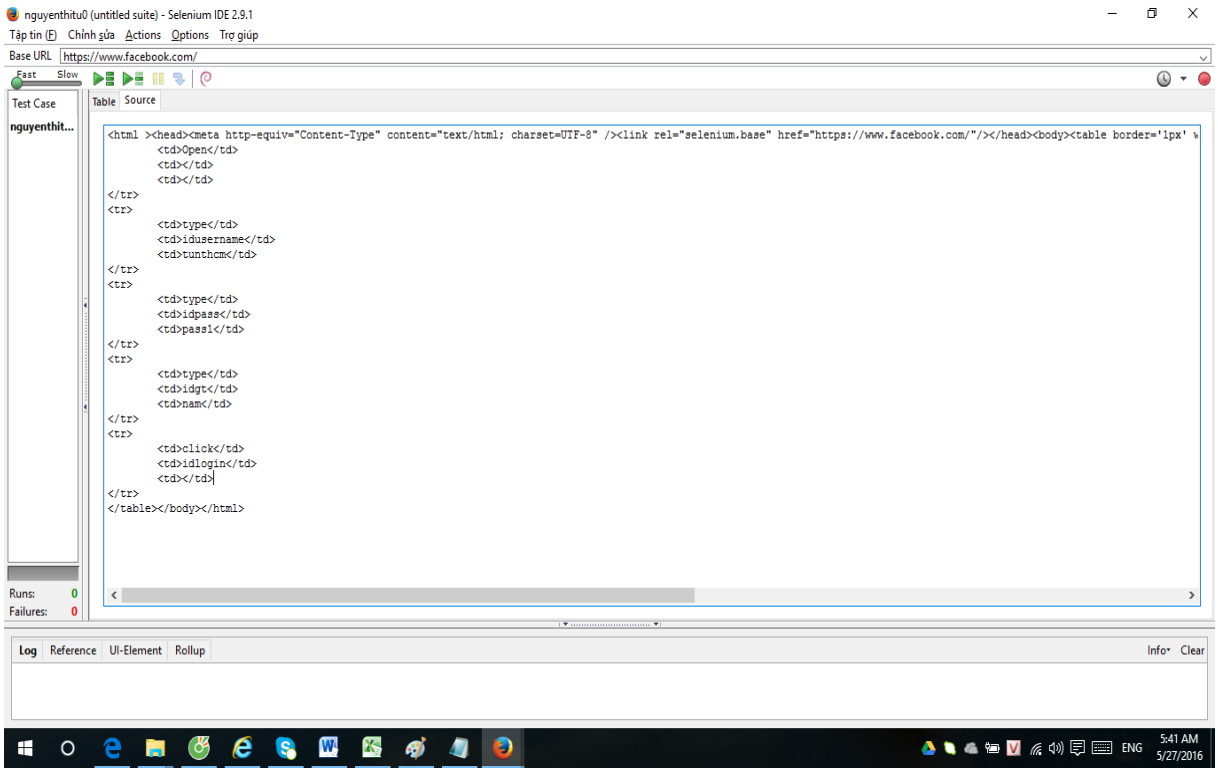
3.4 Kết quả của tool



Hình 3.2 Kết quả xuất ra trên list view



Hình 3. 3 Kết quả xuất file trên forder lựa chọn



Hình 3.4 Kết quả file được tạo khi mở bằng selenium ide

3.5 Ứng dụng công cụ vào thực tế:

Áp dụng vào trong trang facebook.com

3.6 Đánh giá ưu nhược điểm của công cụ

Ưu điểm:

Có thể tự sinh nhiều ca kiểm thử dựa trên điều kiện input.

Chương trình gọn nhẹ, dễ sử dụng.

Tiết kiệm thời gian, nhân lực và chi phí thực hiện.

Sử dụng lại trên selenium ide.

Tạo được ngay cả khi trang web chưa hình thành

Có thể ứng dụng cho nhiều trang web khác nhau.

Nhược điểm:

- Khi chạy nhiều trang vẫn bị những dừng do không có time out.
- Các thành phần còn chưa đầy đủ, thiếu verify text.
- Thiếu time out

Danh mục tài liệu tham khảo

[1]. Book "SOFTWARE TESTING AND QUALITY ASSURANCE Theory and Practice " of KSHIRASAGAR NAIK and RIYADARSHI TRIPATHY

[2]. " Giáo trình kiểm thử phần mềm " của các tác giả Phạm Ngọc Hùng, Trương Anh Hoàng, Đặng Văn Hưng

[3] http://www.phattrienvn.com/chia-se/kinh-nghiem-sqc/cac-loai-kiem-thu-phan-mem-7110.html

[4].https://github.com/Microsoft/PICT/blob/master/doc/Pairwise%20Testing%20in%20Real%20World.pdf

[5] http://www.bcs.org/upload/pdf/lcopeland-070312.pdf]

[6] https://testmuse.wordpress.com/2006/04/05/PICT-tool-available/