

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT HƯNG YÊN



GIÁO TRÌNH NỘI BỘ
KIỂM THỦ PHẦN MỀM

TRÌNH ĐỘ ĐÀO TẠO: **ĐẠI HỌC CHÍNH QUY**
NGÀNH ĐÀO TẠO: **KỸ THUẬT PHẦN MỀM**

MỤC LỤC

BÀI 1. CƠ BẢN VỀ KIỂM THỬ PHẦN MỀM	5
1.1. Những lỗi (bug) phần mềm nghiêm trọng trong lịch sử	5
1.2. Lỗi (bug) là gì?.....	11
1.3. Tại sao lỗi xuất hiện?	17
1.4. Chi phí cho việc sửa lỗi	18
1.5. Người kiểm thử phần mềm (<i>software tester</i>) làm những gì?.....	20
1.6. Những tố chất nào tạo nên một tester tốt?	21
1.7. Bảy nguyên tắc kiểm thử phần mềm.....	23
BÀI 2. QUY TRÌNH PHÁT TRIỂN PHẦN MỀM	27
2.1. Quy trình phát triển phần mềm	27
2.2. Thực trạng của quá trình kiểm thử phần mềm	36
2.3. Quá trình nghiên cứu bản đặc tả phần mềm.....	54
BÀI 3. QUY TRÌNH KIỂM THỬ PHẦN MỀM.....	67
3.1. Quy trình kiểm thử phần mềm tổng quát	67
3.2. Lập kế hoạch kiểm tra (Test Plan)	68
3.3. Chuẩn bị môi trường kiểm tra	71
3.4. Thiết kế kiểm tra (Test Design)	72
3.5. Thực hiện kiểm tra (Test Execute).....	76
3.6. Thẩm tra và đánh giá kết quả kiểm tra.....	80
3.7. Ghi nhận và xử lý lỗi.....	81
3.8. Lập kế hoạch và triển khai kiểm thử hồi quy.....	83
3.9. Thông báo phát hành sản phẩm.....	84
3.10. Xây dựng kế hoạch kiểm thử	85
3.11. Các thành phần chính trong kế hoạch kiểm thử	88
BÀI 4. CÁC MỨC ĐỘ KIỂM THỬ PHẦN MỀM – TEST LEVELS	97
4.1. Unit Test – Kiểm thử mức đơn vị	97
4.2. Integration Test – Kiểm thử mức tích hợp	103
4.3. System Test - Kiểm thử mức hệ thống.....	108
4.4. Acceptance Test - Kiểm thử chấp nhận sản phẩm	110
4.5. Regression Test - Kiểm thử hồi quy	111
BÀI 5. THIẾT KẾ TEST – TEST DESIGN	113

5.1. Tổng quan về test design	113
5.2. Cấu trúc test design.....	113
5.3. Ví dụ về test design.....	117
5.4. Test Case.....	119
BÀI 6. KỸ THUẬT KIỂM THỬ HỘP TRẮNG (WHITE BOX TESTING) (I)	127
6.1. Tổng quan về kiểm thử hộp trắng.....	127
6.2. Kiểm thử dựa trên luồng điều khiển (Control Flow)	127
6.2.4. Đồ thị dòng điều khiển cơ bản	133
BÀI 7. KỸ THUẬT KIỂM THỬ HỘP TRẮNG (II)	140
7.1. Kiểm thử luồng dữ liệu (Data Flow).....	140
7.2. Phân tích đồi sóng của một biến	141
7.3. Đồ thị dòng dữ liệu	142
BÀI 8. KỸ THUẬT KIỂM THỬ HỘP ĐEN (BLACK - BOX) (I)	148
8.1. Giới thiệu	148
8.2. Kỹ thuật Phân chia lớp tương đương	149
8.3. Phân tích giá trị biên - BVA - Boundary Value Analysis.....	154
BÀI 9. KỸ THUẬT KIỂM THỬ HỘP ĐEN (II)	160
9.1. Kỹ thuật dùng bảng quyết định (decision table)	160
9.2. Kỹ thuật dựa trên đặc tả Use Case (Use case)	163
BÀI 10. MỘT SỐ VẤN ĐỀ CẦN KIỂM THỬ	170
10.1. Kiểm thử giao diện	170
10.2. Kiểm thử chức năng.....	178
10.3. Kiểm thử cấu hình và khả năng tương thích.....	185
10.4. Kiểm thử hiệu năng.....	188
10.5. Kiểm thử bảo mật	206
10.6. Kiểm thử khả năng tiện dụng.....	215
10.7. Kiểm thử ngôn ngữ	221
10.8. Kiểm thử tài liệu	229
10.9. Kiểm thử khả năng phục hồi	233
BÀI 11. QUẢN LÝ LỖI PHẦN MỀM - BUG MANAGEMENT	236
11.1. Các thành phần của lỗi.....	236
11.2. Mẫu Defect log	242
11.3. Tổng quan về test report	244

11.4. Quy trình test report	244
11.5. Cấu trúc của test report	246
11.6. Ví dụ test report.....	246
BÀI 12. THỰC HÀNH 1: KỸ THUẬT KIỂM THỬ HỘP TRẮNG	249
BÀI 13. THỰC HÀNH 2: KỸ THUẬT KIỂM THỬ HỘP ĐEN (I).....	249
BÀI 14. THỰC HÀNH 3: KỸ THUẬT KIỂM THỬ HỘP ĐEN (II)	249
BÀI 15. THỰC HÀNH 4: THIẾT KẾ KIỂM THỬ PHẦN MỀM (TEST DESIGN)	249
BÀI 16. THỰC HÀNH 5: XÂY DỰNG CÁC TRƯỜNG HỢP KIỂM THỬ PHẦN MỀM (TESTCASE)	249
BÀI 17. THỰC HÀNH 6: QUẢN LÝ LỖI VÀ BÁO CÁO KIỂM THỬ	249

BÀI 1. CƠ BẢN VỀ KIỂM THỬ PHẦN MỀM

Năm 1947, máy tính cỡ lớn (to bằng cả 1 tòa nhà) được điều khiển dựa trên các relay và súc nóng của các ống chân không. Diễn hình cho máy tính giai đoạn này là *Mark II*, chiếc máy tính khổng lồ được xây dựng bởi trường đại học *Harvard*. Các kỹ thuật viên đang từng bước chạy chiếc máy tính mới thì nó đột nhiên dừng làm việc. Họ đã mất rất nhiều công sức để tính toán xem tại sao và họ đã khám phá ra: họ đang bị mắc kẹt giữa một tập các relay ở sâu bên trong ruột của máy tính. Dường như, chúng bị căng phồng lên trong hệ thống bởi ánh sáng và súc nóng, và bị hạ gục bởi điện áp cao khi nó đang hoạt động trên các relay. Như vậy, quá trình lập trình để điều khiển hoạt động của máy tính có vấn đề không ổn.

Vì thế mà chúng ta hãy đến với những bài học của môn *Software testing*. Nội dung chính của môn học này bao gồm:

- Lịch sử về lỗi phần mềm, những khái niệm cơ bản về lỗi phần mềm
- Các kỹ năng nền tảng của việc kiểm thử phần mềm
- Những yếu tố cơ bản cần kiểm thử trong một phần mềm
- Các giai đoạn trong khi kiểm thử một phần mềm
- Làm việc với các tài liệu kiểm thử: lập kế hoạch, viết và theo dõi các test case, báo cáo lỗi
- Chuẩn quốc tế của một phần mềm tốt

Trong bài này, chúng ta sẽ tìm hiểu về lịch sử của các lỗi phần mềm và kiểm thử phần mềm.

Những điểm cần chú ý trong bài này bao gồm:

- Các lỗi phần mềm tác động đến cuộc sống của chúng ta như thế nào?
- Lỗi là gì và tại sao chúng xuất hiện?
- Các tester là ai và họ phải làm những gì?

1.1. Những lỗi (bug) phần mềm nghiêm trọng trong lịch sử

- Hãy đánh giá thử xem các phần mềm đã thâm nhập vào cuộc sống của chúng ta như thế nào.

- Sau năm 1947, chiếc máy tính *Mark II* yêu cầu hàng tá những nhà lập trình phải bảo trì liên miên. Những người bình thường không bao giờ tưởng tượng được rằng một ngày nào đó trong căn nhà của họ sẽ có một chiếc máy tính của chính họ.
 - Nay giờ, máy tính tràn ngập khắp nơi, nó không chỉ đến với từng gia đình, mà còn đến với từng cá nhân. Những đĩa CD phần mềm miễn phí với các đoạn video game cho trẻ em, tặng kèm theo các hộp ngũ cốc còn nhiều hơn cả phần mềm trên các tàu con thoi.
- Hãy thử so sánh sự phát triển của các máy nhắn tin và các buồng điện thoại, dịch vụ chuyển phát nhanh... với sự phát triển của máy tính và phần mềm máy tính. Dường như không gì có thể theo kịp sự bùng nổ của ngành công nghiệp đầy chất xám này. Nay giờ, chúng ta có thể không thể không sử dụng các dịch vụ chuyển phát nhanh..., nhưng không thể bắt đầu một ngày mà không vào mạng và kiểm tra thư điện tử.
- Phần mềm ở khắp mọi nơi. Tuy nhiên, nó được viết bởi nhiều người, vì vậy mà nó không hoàn hảo. Chúng ta hãy cùng đi tìm hiểu một số ví dụ dưới đây:

1.1.1. Disney's Lion King, 1994 – 1995

Vào cuối năm 1994, công ty Disney đã tung ra thị trường trò chơi đa phương tiện đầu tiên cho trẻ em, *The Lion King Animated StoryBook*. Mặc dù rất nhiều công ty khác đã quảng bá các chương trình cho trẻ em trong nhiều năm, đây là lần đầu tiên Disney mạo hiểm lao vào thị trường. Nó đã được xúc tiến và quảng cáo mạnh mẽ. Số lượng bán ra vô cùng đồ sộ. Nó được mệnh danh là “*the game to buy*” cho trẻ em trong kỳ nghỉ. Tuy nhiên, chuyện gì đã xảy đến? Đó là một sự thất bại khủng khiếp. Vào 26/12, ngay sau ngày Giáng Sinh, khách hàng của Disney đã liên tục gọi điện. Ngay lập tức, các kỹ thuật viên trợ giúp bằng điện thoại đã bị sa lầy với các cuộc gọi từ các bậc cha mẹ đang giận dữ và những đứa trẻ đang khóc, vì chúng không thể cho phần mềm làm việc. Nhiều câu chuyện đã xuất hiện trên các mặt báo và trên bản tin của TV.

...Disney đã thất bại khi không kiểm tra phần mềm rộng rãi trên nhiều mô hình máy tính khác nhau có sẵn trên thị trường. Phần mềm đã làm việc trên một vài hệ thống mà các lập trình viên của Disney đã dùng để tạo ra trò game này, nhưng nó không phải là các hệ thống phổ biến nhất mà người dùng hay sử dụng.

1.1.1. Lỗi chia dấu phẩy động của bộ vi xử lý Intel Pentium (Intel Pentium Floating – Point Division Bug), 1994

Hãy mở phần mềm Calculator trong máy tính của bạn và thực hiện phép toán sau:

$$(4195835 / 3145727) * 3145727 - 4195835$$

Nếu kết quả là 0, máy tính của bạn hoạt động tốt. Nếu như bạn nhận được một kết quả khác, thì bạn đang sở hữu một Intel Pentium CPU với lỗi **floating – point division** (chia dấu phẩy động) – một lỗi phần mềm đã làm nóng chip của bạn mà vẫn được tái sản xuất liên tục.

Ngày 30/10/1994, **Thomas R. Nicely** thuộc trường cao đẳng **Lynchburg (Virginia)** đã phát hiện một kết quả không mong muốn trong khi thực hiện phép chia (**division**) trên máy tính của ông. Ông đã công bố kết quả nghiên cứu của mình trên internet và ngay lập tức ông đã làm bùng lên ngọn lửa với một số lượng lớn những người cũng gặp vấn đề như ông. Và họ tìm thêm những tình huống máy tính đưa ra câu trả lời sai. May thay những trường hợp này là hiếm thấy và kết quả đưa ra câu trả lời sai chỉ trong những trường hợp phục vụ cho Toán học chuyên sâu, Khoa học, và các Tính toán kỹ thuật. Hầu hết mọi người sẽ không bao giờ bắt gặp chúng trong khi đang thực hiện các tính toán thông thường hoặc khi đang chạy các ứng dụng thương mại của họ.

Điều gì đã làm cho vấn đề đáng chú ý này không được *Intel* coi là bug, mặt khác cái cách mà *Intel* điều khiển tình hình:

- Họ đã phát hiện ra các vấn đề trong khi thực thi các bài test của chính họ trước khi chip được tung ra thị trường. Các nhà quản lý của Intel đã quyết

định rằng vấn đề này không đủ nghiêm trọng và ít khả năng xảy ra để cần thiết phải *fixing* (sửa) nó hoặc thậm chí là *publicizing* (công khai) nó.

- Lỗi đã bị phát hiện, Intel cố gắng để giảm bớt tính chất nghiêm trọng của vấn đề đã bị nhận bằng cách công bố công khai (*press release*).
- Khi bị gây áp lực, Intel đã ngo ý muốn thay thế miễn phí những chip bị lỗi, nhưng chỉ với điều kiện là người sử dụng đó phải chứng minh được rằng anh ta đã bị ảnh hưởng do lỗi (bug).
- Họ đã gặp phải sự phản đối kịch liệt. Các diễn đàn trên Internet đã tạo sức ép với sự giận dữ của những khách hàng khó tính, đòi Intel phải fix vấn đề này. Các bản tin thì vã lèn hình ảnh về Intel giống như một công ty vô trách nhiệm với khách hàng. Cuối cùng, Intel đã phải xin lỗi bằng cách điều chỉnh bug và đã phải bỏ ra trên **400 triệu dollar** để chi trả cho quá trình thay thế các chip bị lỗi.

Bây giờ, Intel luôn công khai các vấn đề trên Website của họ và cẩn trọng giám sát sự hồi đáp của các khách hàng trên các diễn đàn (*newsgroups*).

Chú ý: Vào ngày 28/08/2000, một thời gian ngắn trước khi phiên bản đầu tiên của cuốn sách này được sản xuất, Intel đã thông báo việc thu hồi tất cả các bộ vi xử lý Pentium III 1.13MHz, sau khi các con chip này được tung ra thị trường khoảng 1 tháng. Một vấn đề đã bị phát hiện. Vì vậy họ phải thực thi cho lời khẳng định chắc chắn rằng các ứng dụng sẽ luôn chạy ổn định. Họ phải lập kế hoạch để thu hồi những chiếc máy tính đã tới tay khách hàng và tính toán giá thành để thay thế cho những con chip bị lỗi.

1.1.2. Tàu vũ trụ của NASA đáp xuống địa cực của sao Hỏa (NASA Mars Polar Lander), 1999

Ngày 3/12/1999, Tàu vũ trụ của NASA đáp xuống địa cực của sao Hỏa đã biến mất khỏi vòng kiểm soát trong khi nó đang cố gắng đáp xuống bề mặt của sao Hỏa. Ban Báo Cáo sự cố đã điều tra sự cố và xác định rằng nguyên nhân có thể xảy ra nhất của sự cố này là việc cài đặt một bit dữ liệu đơn lẻ. Điều đáng chú ý nhất là tại sao sự cố này lại chưa từng được xảy ra trong các cuộc thí nghiệm nội bộ.

Theo lý thuyết, kế hoạch đáp tàu như sau: khi tàu đang đáp xuống bờ mặt, nó sẽ mở ra chiếc dù nhằm làm giảm tốc độ. Một vài giây sau khi mở dù, 3 chân của máy dò sẽ mở ra và chết ở vị trí đáp. Khi máy dò ở vị trí cách bờ mặt sao hỏa 1.800m nó sẽ nhả dù và đốt nóng (thruster) để giảm khoảng cách còn lại so với bờ mặt sao hỏa

Để tiết kiệm, NASA đã đơn giản bộ máy quyết định thời gian ngắn. Thay thế Rada đặt tiền trên tàu vũ trụ, họ đã cài đặt công tắc tiếp xúc (*Contact switch*) ở chân của máy dò. Nói một cách đơn giản, khi các chân của máy rò mở ra sẽ bật công tắc để động cơ đốt cháy cho đến khi các chân chạm đất.

Thật không may ban báo cáo sự cố đã phát hiện ra trong quá trình kiểm tra của họ rằng khi các chân được tách ra để chạm tới đất, một rung động của máy đã làm trượt công tắc đốt cháy và việc thiết đặt bit này đã gây tai họa. Đây là một vấn đề rất nghiêm trọng, máy tính đã tắt bộ phận đốt nóng và con tàu đã bị vỡ ra tung mảnh sau khi rời từ độ cao 1.800m xuống bờ mặt sao Hỏa.

Kết quả thật là thảm, nhưng lý do lại rất đơn giản. Con tàu thám hiểm đã được kiểm tra bởi rất nhiều đội. Một đội đã kiểm tra chức năng mở các chân của con tàu và một đội khác thì kiểm tra việc đáp tàu xuống mặt đất. Đội đầu tiên đã không biết rằng: một bit được thiết đặt cho việc mở các chân của con tàu không nằm trong vùng kiểm tra của họ. Đội thứ 2 thì luôn luôn thiết lập lại máy tính, xóa bit dữ liệu trước khi nó bắt đầu được kiểm tra. Cả 2 đội trên đều làm việc độc lập và hoàn thành nhiệm vụ của mình rất hoàn hảo. Nhưng lại không hoàn hảo khi kết hợp các nhiệm vụ với nhau.

1.1.3. Hệ thống phòng thủ tên lửa Patriot, 1991

Hệ thống phòng thủ tên lửa *Patriot* (người yêu nước) của Mỹ là một phiên bản *scaled-back* của chương trình khởi động chiến lược phòng thủ “*Star Wars*” được khởi động bởi tổng thống *Ronald Reagan*. Nó đặt nền móng cho chiến tranh Vùng Vịnh (*Gulf war*) như một hệ thống phòng thủ tên lửa *Iraqi Scud*. Mặc dù đã có rất nhiều câu chuyện quảng bá về sự thành công của hệ thống, tuy nhiên vẫn tồn

tại lỗi khi chόng lại một vài tên lửa. Một trong số đó đã giết chết 28 lính Mỹ ở *Dhahran, Saudi Arabia*. Quá trình phân tích đã cho thấy rằng, phần mềm đã bị lỗi nghiêm trọng. Một thời gian trễ rất nhỏ trong đồng hồ hệ thống đã được tích lũy lại sau 14h, và hệ thống theo dõi không còn chính xác nữa. Trong cuộc tấn công *Dhahran*, hệ thống đã điều hành trong hơn 100h.

1.1.4. Sự cố Y2K (năm 2000), khoảng 1974

Vào đầu những năm 1970, một lập trình viên, tên anh ta là Dave, đang làm việc cho hệ thống trả tiền của công ty anh ta. Máy tính mà anh ta sử dụng có bộ nhớ lưu trữ rất nhỏ, buộc anh ta phải giữ gìn những byte cuối cùng mà anh ta có. Dave đã rất tự hào rằng anh ta có thể đóng gói các chương trình của mình một cách chặt chẽ (*tightly*) hơn so với một vài đồng nghiệp của anh ta. Một phương thức mà anh ta đã sử dụng là chuyển định dạng ngày tháng từ 4 chữ số, ví dụ 1973 thành định dạng 2 chữ số, ví dụ 73. Bởi vì, hệ thống trả tiền (*Payroll*) của anh ta phụ thuộc rất nặng vào xử lý ngày tháng, nhờ thế Dave có thể giữ lại những không gian nhớ có giá trị. Trong một thời gian ngắn, anh ta đã xem xét những vấn đề có thể xuất hiện khi đến thời điểm năm 2000 và hệ thống của anh ta đã bắt đầu thực hiện các công việc tính toán với các năm được đại diện bằng 00, 01... Anh ta cũng nhận thấy rằng, sẽ có một vài vấn đề xảy đến, nhưng anh ta đã nghĩ rằng chương trình của anh ta sẽ được thay thế hoặc cập nhật trong vòng 25 năm, và nhiệm vụ hiện tại của anh ta là quan trọng hơn là kế hoạch trong tương lai xa như vậy. Và thời hạn đó cũng đã đến. Năm 1995, chương trình của Dave vẫn được sử dụng, Dave đã nghỉ hưu. Và không một ai biết làm thế nào để vào được hệ thống kiểm tra xem nếu đến năm 2000 thì chuyện gì sẽ xảy ra. Chỉ một mình Dave biết cách để fix nó.

Người ta đã ước tính rằng, phải mất đến **vài trăm tỷ dollar** để có thể cập nhật và fix những lỗi tiềm tàng vào năm 2000, cho các chương trình máy tính trên toàn thế giới có sử dụng hệ thống của Dave.

1.1.5. Mối hiểm nguy của Virus, năm 2004

01/04/1994, một thông điệp đã được gửi tới một vài nhóm người sử dụng internet và sau đó nó được truyền bá như một email có chứa một loại virus ẩn trong

các bức ảnh có định dạng JPEG trên internet. Người ta đã cảnh báo rằng chỉ cần thao tác mở và xem những bức tranh bị nhiễm sẽ dẫn đến việc cài đặt virus trên PC của bạn. Sự thay đổi của những lời cảnh báo nói rõ rằng virus có thể làm hỏng màn hình máy tính của bạn và rằng những chiếc màn hình Sony Trinitron là “đặc biệt nhạy cảm”.

Nhiều người đã chú ý tới những lời cảnh báo và làm sạch những file ảnh JPEG trong hệ thống của họ. Thậm chí, một số người quản trị hệ thống còn tìm hiểu sâu bên trong những khối ảnh JPEG được nhận từ email trên hệ thống của họ.

Cuối cùng, mọi người cũng đã nhận thấy rằng, thông điệp ban đầu được gửi đi vào ngày cá tháng 4 (“April Fools Day”) và sự thật là không có chuyện gì cả, nhưng câu chuyện đùa này đã đi quá xa. Các chuyên gia đã rung hồi chuông cảnh báo rằng: không có một cách khả thi nào để một bức ảnh JPEG có khả năng làm máy tính của bạn bị nhiễm virus. Sau tất cả, người ta khẳng định rằng một bức tranh cũng chỉ là dữ liệu, nó không thể thực thi mã chương trình.

Mười năm sau, vào mùa thu năm 2004, một virus *proof-of-concept* đã được tạo ra, nó chứng minh rằng một bức ảnh JPEG có thể được tải về cùng với một virus. Nó sẽ gây ảnh hưởng tới hệ thống được sử dụng để xem nó. Những mẩu tin (*software patches*) được tạo ra một cách nhanh chóng và được thông báo rộng khắp để ngăn chặn virus lan tràn. Tuy nhiên, chỉ là vấn đề thời gian cho đến khi họ không chế được vấn đề này trên internet bằng cách làm sạch các bức ảnh trên đường truyền.

1.2. Lỗi (bug) là gì?

Bạn vừa được tìm hiểu về một số vấn đề có thể xảy ra khi phần mềm bị lỗi. Nó có thể dẫn đến những phiền phức, giống như khi một máy chơi *game* không thể làm việc một cách hợp lý, hoặc nó có thể dẫn đến một thảm họa khủng khiếp nào đó. Số tiền để giải quyết vấn đề và sửa lỗi có thể lên tới hàng triệu *dollar*. Trong các ví dụ ở trên, rõ ràng phần mềm không hoạt động như dự tính ban đầu. Nếu là một tester, bạn sẽ phải tìm thấy hầu hết những lỗi của phần mềm. Hầu hết là những lỗi

đơn giản và tinh vi, có khi quá nhỏ để nỗi không thể phân biệt được cái nào là lỗi và cái nào không phải là lỗi.

NHỮNG THUẬT NGỮ VỀ CÁC LỖI PHẦN MỀM:

Phụ thuộc vào nơi mà bạn được làm việc (như một tester), bạn sẽ sử dụng những thuật ngữ khác nhau để mô tả: điều gì sẽ xảy đến khi phần mềm bị lỗi. Dưới đây là một số thuật ngữ:

<i>Defect</i>	nhiệt điểm
<i>Fault</i>	khuyết điểm
<i>Failure</i>	sự thất bại
<i>Anomaly</i>	sự dị thường
<i>Variance</i>	bíén dị
<i>Incident</i>	việc rắc rối
<i>Problem</i>	vấn đề
<i>Error</i>	lỗi
<i>Bug</i>	lỗi
<i>Feature</i>	đặc trưng
<i>Inconsistency</i>	sự mâu thuẫn

(Chúng ta có một danh sách các thuật ngữ không nên nhắc đến, nhưng hầu hết chúng được sử dụng riêng biệt, độc lập giữa các lập trình viên)

Bạn có thể thấy ngạc nhiên rằng có quá nhiều từ để mô tả một lỗi phần mềm. Tại sao lại như vậy? Có phải tất cả chúng đều thật sự dựa trên văn hóa của công ty và quá trình mà công ty sử dụng để phát triển phần mềm của họ. Nếu bạn tra những từ này trong từ điển, bạn sẽ thấy rằng tất cả chúng đều có ý nghĩa khác nhau không

đáng kể. Chúng cũng có ý nghĩa được suy ra từ cách mà chúng được sử dụng trong các cuộc đàm thoại hàng ngày.

Ví dụ, *fault*, *failure* và *defect* có xu hướng ám chỉ một vấn đề thật sự quan trọng, thậm chí là nguy hiểm. Dường như là không đúng khi gọi một biểu tượng (*icon*) không được tô đúng màu là 1 lỗi (*fault*). Những từ này cũng thường ám chỉ một lời khiển trách: chính là do nó (*fault*) mà phát sinh lỗi phần mềm (*software failure*) (“it’s his fault that the software failure.”)

Anomaly, *incident*, *variance* thì không có vẻ là quá tiêu cực và thường được sử dụng để đề cập tới sự vận hành không được dự tính trước thay vì hoàn toàn là lỗi (*all-out failure*). “Tổng thống đã tuyên bố rằng nó là một sự dị thường phần mềm đã làm cho tên lửa đi sai lịch trình của nó” (“The president stated that it was a software anomaly that caused the missile to go off course.”)

Có lẽ, *Problem*, *error* và *bug* là những thuật ngữ chung nhất thường được sử dụng.

Một điều thú vị khi một số công ty và các đội sản xuất đã tiêu tốn khá nhiều thời gian quý báu của quá trình phát triển phần mềm vào việc thảo luận và tranh cãi về những thuật ngữ được sử dụng. Một công ty máy tính nổi tiếng đã mất hàng tuần để thảo luận với những kỹ sư của họ trước khi quyết định đổi tên *Product Anomaly Report (PARs)* thành *Product Incident Report (PIRs)*. Một số tiền lớn đã được sử dụng cho việc quyết định thuật ngữ nào là tốt hơn. Một khi quyết định đã được đưa ra (Once the decision was made), tất cả các công việc liên quan đến giấy tờ, phần mềm, định dạng... phải được cập nhật để phản ánh thuật ngữ mới. Nó sẽ không được biết tới nếu nó gây ra bất kỳ sự khác biệt nào đối với hiệu quả làm việc của lập trình viên và tester.

Vậy, tại sao phải đưa ra chủ đề này? Thực sự là quan trọng khi một tester hiểu khả năng cá nhân riêng sau nhóm phát triển phần mềm mà bạn đang làm việc cùng. Cách thức họ đề cập tới các vấn đề về phần mềm của họ là dấu hiệu thông thường về cách họ tiếp cận quá trình phát triển toàn bộ của họ.

Mặc dù tổ chức của bạn có thể chọn một cái tên khác, nhưng trong cuốn sách này tất cả các vấn đề về phần mềm sẽ được gọi là các **bug**. Không thành vấn đề nếu lỗi là lớn, nhỏ, trong dự định, hay ngoài dự định, hoặc cảm giác của ai đó sẽ bị tổn thương bởi họ tạo ra chúng. Không có lý do gì để mở xé các từ. *A bug's a bug's a bug.*

ĐỊNH NGHĨA VỀ LỖI PHẦN MỀM:

Các vấn đề về *software problems* bug nghe có vẻ đơn giản, nhưng chưa hẳn đã giải quyết được nó. Nay giờ, từ *problem* cần được định nghĩa. Để tránh việc định nghĩa vòng quanh (*circular definitions*), điều quan trọng là phải mô tả được lỗi là gì?

Đầu tiên, bạn cần một thuật ngữ trợ giúp (*supporting term*): đặc tả phần mềm (*product specification*). Đặc tả phần mềm có thể gọi một cách đơn giản là *spec* hoặc *product spec*, là luận cứ của các đội phát triển phần mềm. Nó định nghĩa sản phẩm mà họ tạo ra, chi tiết là gì, hành động như thế nào, sẽ làm gì, và sẽ không làm gì? Luận cứ này có thể vạch ra phạm vi về hình thức từ một dạng hiểu biết về ngôn từ đơn giản, một email, hoặc một chữ viết nguệch ngoạc trên tờ giấy ăn, tới một tài liệu thành văn được hình thức hóa, chi tiết hơn. Trong bài 2, “Quy trình phát triển phần mềm”, bạn sẽ học về đặc tả phần mềm và quy trình phát triển, nhưng không phải là bây giờ, định nghĩa này là đầy đủ.

Một lỗi phần mềm xuất hiện khi 1 hoặc nhiều hơn trong 5 quy tắc dưới đây là đúng:

1. Phần mềm không thực hiện một số thứ giống như mô tả trong bản đặc tả phần mềm
2. Phần mềm thực hiện một số việc mà bản đặc tả yêu cầu nó không được thực hiện
3. Phần mềm thực hiện một số chức năng mà bản đặc tả không đề cập tới

4. Phần mềm không thực hiện một số việc mà bản đặc tả không đề cập tới, nhưng là những việc nên làm
5. Trong con mắt của người kiểm thử, phần mềm là khó hiểu, khó sử dụng, chậm đối với người sử dụng

Để tìm hiểu kỹ hơn về mỗi quy tắc, hãy cố gắng xem ví dụ dưới đây để áp dụng chúng cho phần mềm *calculator* trong máy tính.

Có lẽ, đặc tả của 1 *calculator* đã nói rõ rằng: nó sẽ thực thi phép cộng, phép trừ, phép nhân, phép chia đúng. Nếu bạn là một tester, nhận việc kiểm thử phần mềm *Calculator*, nhấn phím “+” và không có chuyện gì xảy ra, đó là một lỗi theo quy tắc 1. Nếu bạn nhận được câu trả lời sai, cũng có nghĩa rằng đó là một lỗi, bởi vì theo **quy tắc 1**.

Bản đặc tả phần mềm yêu cầu rằng, *calculator* sẽ không bao giờ bị đột ngột ngưng hoạt động, bị khóa lại hoặc bị đóng băng. Nếu bạn đập thình thịch lên các phím và nhận được thông điệp từ *calculator* là “*not responding*” (dừng quá trình hồi đáp với dữ liệu vào), đây là một lỗi theo **quy tắc 2**.

Mục đích là bạn nhận được phần mềm *calculator* để kiểm thử và thấy rằng bên cạnh các phép cộng, trừ, nhân, chia; nó còn thực hiện các phép căn bậc 2. Điều này chưa từng được nêu trong bản đặc tả. Một lập trình viên có nhiều tham vọng vừa thêm nó vào bởi vì anh ta cảm thấy nó sẽ là một đặc tính tuyệt vời (*great feature*). Đây không phải là một *feature*, nó thật sự là một *bug* theo **quy tắc 3**. Phần mềm đang thực hiện một số công việc mà bản đặc tả phần mềm không hề đề cập tới. Mặc dù sự điều khiển không định hướng này có thể là tốt, nhưng nó sẽ yêu cầu thêm những lỗ lực lập trình và kiểm thử (vì có thể sẽ xuất hiện thêm nhiều lỗi). Lúc này chi phí cho việc sản xuất phần mềm cũng lớn hơn, điều này làm giảm hiệu quả kinh tế của quá trình sản xuất phần mềm.

Đọc **quy tắc thứ 4** có thể thấy hơi lạ với sự phủ định kép, nhưng mục đích của nó là tìm thấy những đặc điểm bị lãng quên, không được nhắc tới trong bản đặc

tả. Bạn bắt đầu kiểm thử phần mềm *Calculator* và khám phá ra rằng, khi Pin yếu, bạn không nhận được những câu trả lời đúng cho quá trình tính toán của bạn nữa. Chưa ai từng xem xét xem *calculator* phản ứng lại như thế nào trong chế độ này. Một giả định không tốt được tạo ra rằng: pin luôn được nạp đầy. Bạn mong muốn rằng công việc sẽ được duy trì cho đến khi pin hoàn toàn hết, hoặc ít nhất bằng cách nào đó báo cho bạn biết Pin đã yếu. Những phép tính đúng đã không xảy ra khi pin yếu, và nó cũng không chỉ rõ điều gì sẽ xảy đến. Quy tắc số 4 tạo nên lỗi này.

Quy tắc số 5 mang tính chất tổng hợp (*catch-all*). *Tester* là người đầu tiên thực sự sử dụng phần mềm như một khách hàng lần đầu sử dụng phần mềm. Nếu bạn phát hiện một vài điều mà bạn thấy không đúng vì bất cứ lý do gì, thì nó là một lỗi. Trong trường hợp của *calculator*, có lẽ bạn đã tìm thấy những nút có kích thước quá nhỏ. Hoặc có thể sự sắp xếp của nút “=” đã làm cho nó khó sử dụng. Hoặc sự bố trí màu sắc làm cho nó rất khó nhìn... Tất cả những điều này đều là lỗi (bug) theo quy tắc 5.

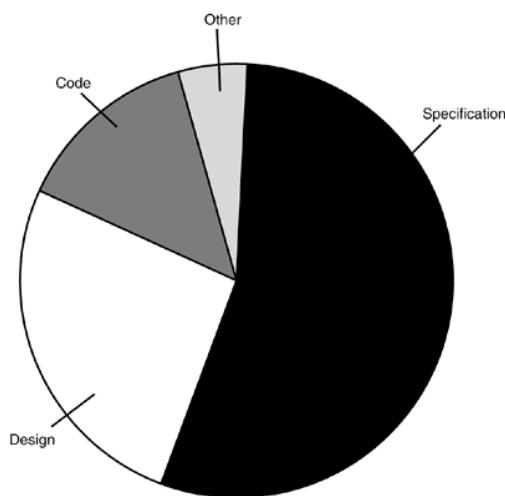
Chú ý: Mọi người sử dụng một phần của phần mềm sẽ có những mong đợi khác nhau và những ý kiến phần mềm nên hoạt động như thế nào. Sẽ không thể viết được phần mềm mà mọi người sử dụng đều thấy nó hoàn hảo. *Tester* sẽ luôn phải giữ những ý nghĩ này trong suy nghĩ của họ khi họ áp dụng quy tắc 5 để kiểm thử. Xét một cách thấu đáo, hãy sử dụng khả năng đánh giá tốt nhất của bạn, và **quan trọng nhất là phải hợp lý**. Ý kiến của bạn có giá trị, nhưng, bạn sẽ được tìm hiểu trong các phần sau, không phải tất cả các lỗi bạn tìm được có thể hoặc sẽ được sửa (*fix*).

Để có một số ví dụ đơn giản và điển hình, bạn hãy nghĩ xem các quy tắc được áp dụng như thế nào với phần mềm mà bạn sử dụng hàng ngày. Đâu là điều bạn mong đợi, đâu là điều không mong đợi? Bạn thấy điều gì đã được chỉ rõ và điều gì bị bỏ quên? Và điều mà bạn hoàn toàn không thích về phần mềm này?

Định nghĩa trên về lỗi của một phần mềm đã bao quát những vấn đề cơ bản, nhưng nếu sử dụng tất cả 5 quy tắc trên sẽ giúp bạn định nghĩa được các loại vấn đề khác nhau trong phần mềm mà bạn đang kiểm thử.

1.3. Tại sao lỗi xuất hiện?

Bây giờ, bạn biết lỗi là gì, bạn có thể tự hỏi tại sao lỗi xuất hiện. Bạn sẽ ngạc nhiên khi khám phá ra rằng hầu hết những lỗi này không phải là lỗi do lập trình. Quá trình khảo sát trên vô số dự án từ rất nhỏ tới cực lớn và kết quả luôn luôn giống nhau. Các lý do quan trọng gây ra lỗi phần mềm được mô tả như hình 1.1 dưới đây:



Hình 1.1: Các lỗi có thể bị phát sinh do nhiều lý do, nhưng trong quá trình phân tích các dự án mẫu thì lý do chính có thể được tìm thấy trong quá trình truy vết theo bản đặc tả.

Những lý do liên quan đến bản đặc tả là nguyên nhân chính làm xuất hiện lỗi *specification*:

- Một số bản đặc tả không viết cụ thể, không đủ kỹ lưỡng
- Hoặc nó liên tục thay đổi, nhưng lại không có sự phối hợp, trao đổi thông tin kịp thời với các đội phát triển dự án.
- Lập kế hoạch cho phần mềm là vô cùng quan trọng. Nếu nó không được thiết kế đúng, lỗi sẽ phát sinh

Lý do quan trọng tiếp theo mà dễ phát sinh lỗi là quá trình thiết kế. Đây là nơi mà các lập trình viên sắp xếp kế hoạch về phần mềm của họ. So sánh nó với kiến trúc được thiết kế cho một ngôi nhà. Các lỗi xuất hiện ở đây với những lý do tương tự như khi chúng xuất hiện trong bản đặc tả. Nó bị dồn lại, thay đổi, hoặc giao tiếp không tốt.

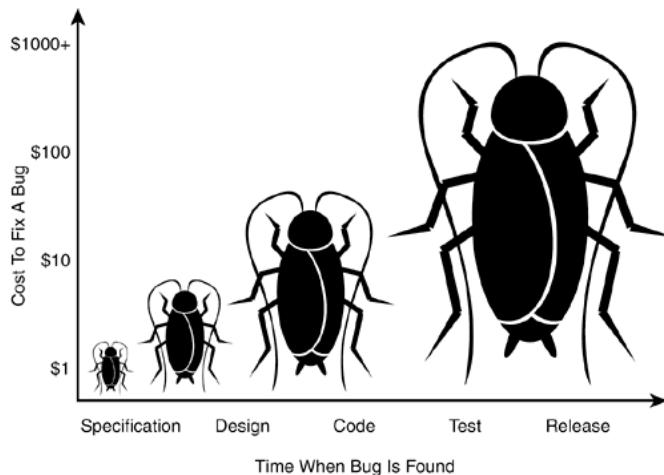
Chú ý: Có một câu nói quen thuộc như sau: “Nếu bạn không thể nói, bạn cũng sẽ không thể làm” (“If you can't say it, you can't do it”). Điều này có thể áp dụng một cách hoàn hảo cho quy trình phát triển và kiểm thử phần mềm.

Những lỗi về code có thể quen thuộc với bạn hơn nếu bạn là một lập trình viên. Điện hình, như là có thể theo dõi được độ phức tạp của phần mềm, văn bản nghèo nàn (đặc biệt trong các đoạn mã được cập nhật hoặc thay đổi), áp lực của lịch làm việc, hoặc những lỗi ngó ngắt. Điều quan trọng là phải chú ý rằng có nhiều lỗi thường xuất hiện bên ngoài là những lỗi lập trình được theo dõi qua bản đặc tả và lỗi thiết kế.

Một số thứ tưởng rằng là lỗi nhưng thực ra lại không phải. Một số lỗi bị nhân bản lên, bắt nguồn từ những nguyên nhân giống nhau. Một số lỗi bắt nguồn từ việc kiểm tra sai. Và cuối cùng, những bug (hay những thứ mà chúng ta tưởng là lỗi) hóa ra lại không phải là lỗi và chúng chiếm tỷ lệ nhỏ trong những bug được báo cáo.

1.4. Chi phí cho việc sửa lỗi

Bạn sẽ tìm hiểu trong bài 2, phần mềm không xuất hiện một các kỳ diệu mà thường là phải có cả 1 quá trình phát triển các phương thức, kế hoạch được sử dụng để tạo ra nó. Từ sự khởi đầu của phần mềm, qua quá trình lập kế hoạch, lập trình, và kiểm thử, đến khi nó được sử dụng bởi khách hàng, những lỗi này có khả năng được tìm thấy. Hình 1.1 biểu diễn một ví dụ về những chi phí cho việc sửa những lỗi có thể phát sinh trong toàn bộ thời gian thực hiện dự án.



Hình 1.2: Chi phí cho việc sửa lỗi có thể tăng đột ngột trên toàn bộ dự án

Chi phí được tính theo hàm loga, mỗi lần chúng tăng lên gấp 10 lần. Lỗi được tìm thấy và sửa lại trong thời gian gần nhất khi bản đặc tả bắt đầu được viết thì chi phí có thể không là gì cả, hoặc chỉ là 1\$ cho ví dụ của chúng ta. Cũng với lỗi tương tự, nếu nó không được tìm thấy cho đến khi phần mềm được lập trình và kiểm thử thì chi phí có thể lên tới 10\$ đến 100\$. Nếu một để một khách hàng tìm ra nó, thì chi phí có thể lên tới hàng nghìn thậm chí hàng triệu dollar.

Như một ví dụ trong cuốn sách này, hãy xem xét về *The Disney Lion King* đã được thảo luận gần đây. Lý do cơ bản của vấn đề là phần mềm này không làm việc được trên nền máy tính phổ biến lúc đó. Nếu như ngay ở giai đoạn đặc tả đầu tiên, ai đó đã nghiên cứu dạng máy PC phổ biến và chỉ ra rằng phần mềm cần được thiết kế và kiểm thử để làm việc được trên những cấu hình đó, thì chi phí cho những cố gắng trên sẽ là rất nhỏ. Nếu điều này không được thực hiện, một bản backup sẽ được gửi cho tester để thu thập những mẫu máy tính phổ biến và thay đổi phần mềm cho phù hợp với chúng. Họ sẽ tìm thấy lỗi, nhưng quá trình sửa lỗi sẽ tốn kém hơn bởi vì phần mềm sẽ phải gỡ lỗi, sửa lỗi và kiểm thử lại. Đội phát triển dự án có thể cũng phải gửi đi một phiên bản đầu tiên của phần mềm tới một nhóm nhỏ các khách hàng để họ kiểm tra, quá trình này gọi là kiểm thử *beta*. Những khách hàng này, đặc trưng cho một thị trường lớn, sẽ có khả năng khám phá ra nhiều vấn đề. Tuy nhiên, lỗi phần mềm không phải là hoàn toàn cho đến khi có hàng nghìn CD-ROM được

sản xuất và bán. Disney đã kiên trì trợ giúp khách hàng qua điện thoại trả lời về sản phẩm, thay thế các ổ CD-ROM, tốt như quá trình gỡ lỗi khác, sửa lỗi và vòng đời kiểm thử. Thật dễ dàng để làm tiêu tan toàn bộ lợi ích của sản phẩm nếu các lỗi là nguy hiểm đối với khách hàng.

1.5. Người kiểm thử phần mềm (*software tester*) làm những gì?

Bây giờ bạn có phần mềm với những lỗi ngớ ngẩn, bạn đã biết định nghĩa của một lỗi là gì, và bạn cũng biết chi phí cho chúng đắt đỏ như thế nào. Vậy mục đích của một tester là gì: *Mục đích của tester là tìm ra lỗi phần mềm* (“*The goal of a software tester is to find bugs*”)

Bạn có thể liên kết (run across) với các đội phát triển sản phẩm, người luôn muốn các tester xác nhận rằng phần mềm của họ làm việc tốt, không có lỗi. Hãy kiểm tra lại các Case study về con tàu thám hiểm lên địa cực của sao Hỏa, và bạn sẽ thấy tại sao đây là hướng tiếp cận sai. Nếu bạn chỉ kiểm tra những thứ mà sẽ làm việc và cài đặt cho quá trình kiểm tra của bạn, vì vậy, chúng sẽ luôn pass. Và bạn sẽ rất dễ bỏ quên những thứ không làm việc. Cuối cùng, bạn sẽ không phát hiện ra một số lỗi.

Nếu bạn đang bỏ sót một số lỗi, bạn sẽ phải trả giá cho dự án của bạn và tiền của công ty bạn. Là một *tester*, bạn không nên bằng lòng với những lỗi đã được tìm thấy, bạn nên nghĩ xem làm thế nào để tìm thấy chúng sớm nhất trong quy trình phát triển phần mềm, như vậy, chi phí để *fix* lỗi sẽ ít hơn.

Mục đích của một tester là tìm các lỗi và tìm thấy chúng một cách sớm nhất có thể (“*The goal of a software tester is to find bugs and find them as early as possible*”).

Nhưng, tìm kiếm các lỗi, thậm chí phát hiện chúng từ rất sớm vẫn là chưa đủ. Hãy nhớ lại định nghĩa về 1 lỗi. Bạn, 1 tester, là con mắt của khách hàng, trước tiên phải xem xét phần mềm. Bạn nói chuyện với khách hàng và phải tìm kiếm một sự hoàn chỉnh.

Mục đích của một tester là tìm các lỗi, tìm thấy chúng một cách sớm nhất có thể, và chắc chắn rằng chúng sẽ được sửa (“The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.”).

Câu nói cuối cùng này rất quan trọng. Bạn hãy ghi nhớ nó và lấy ra xem lại khi bạn tìm hiểu về các kỹ thuật kiểm thử được thảo luận trong toàn bộ những nội dung quan trọng của cuốn sách này.

Chú ý: Điều quan trọng là phải chú ý: lỗi phần mềm được sửa không có nghĩa rằng phần mềm đã hoàn hảo. Phần mềm nên được bổ sung thêm những hướng dẫn sử dụng hoặc cung cấp các khóa đào tạo đặc biệt cho khách hàng. Việc này có thể còn yêu cầu thay đổi những số liệu mà nhóm Marketing quảng cáo hoặc thậm chí trì hoãn việc giải phóng (bỏ qua) *buggy feature*. Trong suốt cuốn sách này, bạn sẽ học cách để trở thành người đi tìm kiếm sự hoàn hảo và phải chắc chắn rằng những lỗi được phát hiện sẽ được sửa, và sẽ có những bài thực hành thực tế cho bạn kiểm thử. Đừng có tìm kiếm đường xoắn ốc nguy hiểm của sự hoàn hảo không thể có thật (*Don't get caught in the dangerous spiral of unattainable perfection*).

1.6. Những tố chất nào tạo nên một tester tốt?

Trong đoạn phim *Star Trek II: The Wrath of Khan*, Spock nói rằng: “Là một vấn đề của lịch sử vũ trụ, phá hủy bao giờ cũng dễ hơn kiến tạo” (“As a matter of cosmic history, it has always been easier to destroy than to create”). Mới nhìn qua, có thể mọi người sẽ nghĩ công việc của một tester là đơn giản hơn so với công việc của một lập trình viên. Phát hiện ra những lỗi lập trình chắc chắn là dễ hơn so với viết code. Nhưng thật ngạc nhiên, điều đó lại không đúng. Cách tiếp cận rất bài bản và có kỷ luật với phần mềm mà bạn sẽ tìm hiểu trong cuốn sách này yêu cầu bạn phải công hiến và làm việc một cách chăm chỉ chẳng kém gì một lập trình viên. Hai công việc đều phải sử dụng rất nhiều kỹ năng tương tự nhau, và mặc dù kiểm thử phần mềm không nhất thiết phải cần là một lập trình viên chuyên nghiệp, nhưng họ lại tạo ra những khoản lợi nhuận lớn.

Ngày nay, hầu hết những công ty đã trưởng thành đều coi kiểm thử phần mềm như công việc của một kỹ sư kỹ thuật. Họ thừa nhận rằng phải đào tạo các tester trong các đội dự án của họ và cho phép họ áp dụng các kỹ thuật vào quá trình phát triển phần mềm để xây dựng được một phần mềm có chất lượng tốt hơn. Thật không may, vẫn có một vài công ty không đánh giá đúng nhiệm vụ khó khăn của quá trình kiểm thử và giá trị của những lỗ lực kiểm thử rất bền bỉ. Với sự giao thiệp trong thị trường tự do, có những công ty thường nổi bật hơn hẳn, bởi vì khách hàng thì thường nói rằng: không nên mua những sản phẩm có nhiều khiếm khuyết. Một tổ chức kiểm thử tốt (hoặc thiếu một cái) có thể tạo nên hoặc phá hỏng một công ty.

Hãy nhìn vào danh sách những đặc điểm mà một tester nên có:

- **Họ là những người thám hiểm:** tester không sợ mạo hiểm khi ở trong những hoàn cảnh mà họ chưa làm chủ được. Họ thích những khía cạnh mới của phần mềm, cài đặt nó trên máy của họ, và xem xét chuyện gì sẽ xảy ra.
- **Họ là những người thợ sửa chữa:** các tester làm rất tốt các công việc tính toán xem tại sao một số chức năng của phần mềm lại không làm việc. Họ rất thích những vấn đề khó giải quyết
- **Họ rất nghiêm khắc:** Các tester luôn phải thử nghiệm, họ có thể nhìn thấy một lỗi mà đã nhanh chóng biến mất hoặc là rất khó để tạo lại tình huống có lỗi đó. Đúng hơn là giải tán nó như một sự may mắn, họ sẽ cố gắng bằng mọi cách có thể để tìm ra nó.
- **Họ luôn sáng tạo:** việc kiểm thử những điều hiển nhiên, rõ ràng là không thể đủ với một tester. Công việc của họ cần những ý tưởng sáng tạo và thậm chí là các cách tiếp cận mới mẻ để tìm kiếm lỗi (*bug*).
- **Họ là những người cầu toàn:** Họ cố gắng để đạt đến sự hoàn hảo, nhưng họ cũng biết rằng điều đó là không thể đạt được và họ chấp nhận dừng quá trình kiểm thử khi họ thấy có thể.
- **Họ sử dụng óc phán đoán rất tốt:** tester cần đưa ra những quyết định về những thứ mà họ sẽ phải kiểm tra, và ước lượng quá trình kiểm tra sẽ diễn ra trong thời gian bao lâu, nếu như vấn đề mà họ tìm kiếm thật sự là một lỗi.

- **Họ là những người rất khéo léo và thích ngoại giao:** Tester luôn là người thông báo những tin tức xấu. Họ phải nói với lập trình viên những lỗi mà họ phát hiện. Một tester tốt sẽ biết cách để làm việc khéo léo và rất chuyên nghiệp, và họ cũng biết cách để làm việc với lập trình viên, những người không phải lúc nào cũng khéo léo và lịch thiệp.
- **Họ là người biết cách thuyết phục người khác:** các lỗi mà tester tìm thấy sẽ luôn được xem xét một cách đủ khắt khe để đảm bảo nó sẽ được sửa. Các tester cần chứng minh những luận điểm của họ rằng tại sao những lỗi mà họ phát hiện lại cần được sửa, và những lỗi này có thể gây ra những gì?

KIỂM THỬ PHẦN MỀM LÀ MỘT CÔNG VIỆC RẤT THÚ VỊ: Một đặc điểm cơ bản của các tester là họ rất thích thú với những thứ bị hỏng. Họ sống là để tìm kiếm những sai lầm của các hệ thống khó nắm bắt. Họ toại nguyện khi phát hiện lỗi trong các chương trình phức tạp. Họ thường nhảy lên sung sướng vì điều đó.

Trong những nét đặc trưng này, nếu tester có một số kiến thức về lập trình phần mềm là một ưu thế rất lớn. Bài này sẽ hiểu cách thức mà phần mềm được viết, nó đưa cho bạn một cách nhìn khác về nơi mà các lỗi phần mềm được tìm thấy. Vì vậy, bài này sẽ giúp bạn trở thành một tester làm việc có hiệu quả và gây ảnh hưởng nhiều hơn. Có thể nó cũng giúp bạn phát triển các công cụ kiểm thử.

Cuối cùng, nếu bạn là một chuyên gia trong lĩnh vực không phải là về máy tính, thì sự hiểu biết của bạn có thể vô cùng giá trị để đội dự án phần mềm tạo ra được một sản phẩm mới. Hàng ngày, phần mềm đang được viết để làm mọi thứ. Sự hiểu biết của bạn về dạy học, nấu ăn, máy bay, y học hoặc bất cứ cái gì sẽ là sự trợ giúp đặc lực cho các lỗi được tìm thấy trong phần mềm về lĩnh vực đó.

1.7. Bảy nguyên tắc kiểm thử phần mềm

Một số nguyên tắc kiểm thử đã được đề nghị từ 40 năm về trước và đã đưa ra một số phương châm chung phổ biến cho kiểm thử phần mềm, bao gồm các nguyên tắc sau:

Nguyên tắc 1 – Kiểm thử đưa ra lỗi

Kiểm thử có thể cho thấy rằng phần mềm đang có lỗi, nhưng không thể chứng minh rằng phần mềm không có lỗi. Kiểm thử làm giảm xác suất lỗi chưa tìm thấy vẫn còn trong phần mềm, thậm chí là không còn lỗi nào, nó không phải là bằng chứng của sự chính xác.

Nguyên tắc 2 – Exhaustive testing is impossible

Kiểm thử mọi thứ (tất cả các tổ hợp của điều kiện input đầu vào) là không thể thực hiện được, trừ phi nó chỉ bao gồm một số trường hợp bình thường (ít trường hợp tổ hợp thì có thể test toàn bộ được). Thay vì kiểm thử toàn bộ, việc phân tích rủi ro và dựa trên sự mức độ ưu tiên chúng ta có thể tập trung việc kiểm thử vào một số điểm cần thiết.

Nguyên tắc 3 – Kiểm thử sớm

Để tìm được bug sớm, các hoạt động kiểm thử nên được bắt đầu càng sớm càng tốt trong quá trình phát triển (vòng đời phát triển) phần mềm hoặc hệ thống, và nên tập trung vào các hoạt động đã định trước.

Nguyên tắc 4 – Phân loại lỗi

Nỗ lực kiểm thử nên tập trung một cách cân đối vào mật độ lỗi dự kiến và lỗi phát hiện ra sau đó trong các mô-đun. Một số ít các mô-đun thường chứa nhiều lỗi không phát hiện ra trong lúc kiểm thử trước khi phát hành (release), hoặc chịu trách nhiệm cho hầu hết các lỗi hoạt động của phần mềm.

Để hiểu rõ hơn nguyên tắc này, ta cần xem xét 3 điều sau:

1. Nguyên tắc tổ gián: chỗ nào có 1 vài con gián thì ở đâu đó xung quanh nó sẽ có cả tổ gián -> có rất nhiều gián >>> chỗ nào có 1 vài con bug thì xung quanh, gần gần chỗ đó sẽ có nhiều bug.
2. Nguyên tắc 80/20: thông thường 20% chức năng quan trọng trong một chương trình có thể gây ra đến 80% tổng số bug phát hiện được trong chương trình đó.

3. Exhaustive testing is impossible (nguyên tắc thứ 2): do đó cần phải analysis (phân tích) + priorities (tính toán mức độ ưu tiên) để quyết định tập trung vào test chỗ nào.

=> Test kỹ chức năng quan trọng => found bug => test những gì liên quan + những chức năng gần nó để tìm ra bug nhiều hơn.

Nguyên tắc 5 – Nghịch lý thuốc trừ sâu

Nếu việc kiểm thử tương tự nhau được lặp đi lặp lại nhiều lần, thì cuối cùng sẽ có một số trường hợp kiểm thử (case kiểm thử - test case) sẽ không còn tìm thấy bất kỳ lỗi nào mới. Để khắc phục "nghịch lý thuốc trừ sâu" này, các trường hợp kiểm thử cần phải được xem xét và sửa đổi thường xuyên, và cần phải viết các test case mới và khác nhau để thực hiện nhiều phần khác nhau của phần mềm hoặc hệ thống để tìm ra lỗi tiềm ẩn nhiều hơn nữa.

Nguyên tắc này giống như việc trừ sâu trong nông nghiệp, nếu chúng ta cứ phun một loại thuốc với nồng độ giống nhau trong một khoảng thời gian dài thì có một số con sâu sẽ quen dần và cuối cùng việc phun thuốc giống như là tắm chúng vậy (bị lòn thuốc) => lúc đó chúng ta không thể diệt sạch chúng được. Do vậy, để diệt sạch sâu một cách hiệu quả, người ta thường thay đổi loại thuốc trừ sâu, mỗi loại chỉ dùng trong khoảng thời gian ngắn.

Nguyên tắc 6 – Kiểm thử theo các ngữ cảnh độc lập

Nguyên tắc này là việc testing phụ thuộc vào ngữ cảnh, test trong nhiều ngữ cảnh khác nhau.

Để hiểu rõ hơn chúng ta xem ví dụ sau:

Ví dụ, cũng với một chương trình calculator có rất nhiều chức năng, nhưng:

- Nếu test chương trình này cho mẫu giáo thì chỉ cần test cộng trừ là OK
- Nếu test chương trình này cho cấp 2 thì cộng trừ nhân chia

- Nếu test chương trình này cho đại học thì tích phân, đạo hàm, v.v....

Nguyên tắc 7 – Sự sai lầm về việc không có lỗi

Việc tìm và sửa chữa lỗi sẽ không giúp được gì nếu hệ thống được xây dựng xong nhưng không thể dùng được và không đáp ứng được nhu cầu và sự mong đợi của người dùng. (Nghĩa là nếu sau khi code, test rồi fix bug, làm đủ tất cả các trường hợp và cuối cùng cho ra một sản phẩm không như mong đợi hoặc không đáp ứng được nhu cầu của khách hàng thì dự án phần mềm đó coi như thất bại mặc dù đã được test xong)

BÀI 2. QUY TRÌNH PHÁT TRIỂN PHẦN MỀM

Để trở thành một tester có hiệu quả, ít nhất bạn phải hiểu toàn bộ quy trình phát triển phần mềm ở mức cao. Nếu bạn viết một chương trình nhỏ như một người mới tập lập trình hoặc như một sở thích, bạn sẽ thấy rằng cách mà bạn làm khác hẳn với những cách thức mà các công ty lớn thường sử dụng để phát triển phần mềm. Để tạo ra được một sản phẩm phần mềm mới, có thể bao gồm hàng tá, hàng trăm, thậm chí hàng nghìn thành viên thực hiện các quy tắc khác nhau và làm việc cùng nhau dưới một lịch trình chặt chẽ. Hãy xem xét những nhiệm vụ mà họ phải thực hiện, họ gây ảnh hưởng tới nhau như thế nào, và cách mà họ đưa ra những quyết định ở tất cả các giai đoạn của quy trình phát triển phần mềm.

2.1. Quy trình phát triển phần mềm

Mục đích của phần này là hướng dẫn cho bạn mọi thứ về quy trình phát triển phần mềm sẽ được áp dụng trong môn học này. Mục đích là cho bạn một cái nhìn tổng quan về tất cả các phần bên trong một sản phẩm phần mềm và thấy được một vài hướng tiếp cận chung thường được sử dụng ngày nay. Với những hiểu biết này, bạn sẽ tự có cách tốt nhất để áp dụng các kỹ năng kiểm thử phần mềm mà bạn sẽ được học.

Phần chính của bài này bao gồm:

- Các thành phần (component) chính nào bên trong một sản phẩm phần mềm
- Những ai và các kỹ năng nào đóng góp vào một sản phẩm phần mềm
- Xử lý phần mềm như thế nào để từ một ý tưởng xây dựng lên một sản phẩm cuối cùng

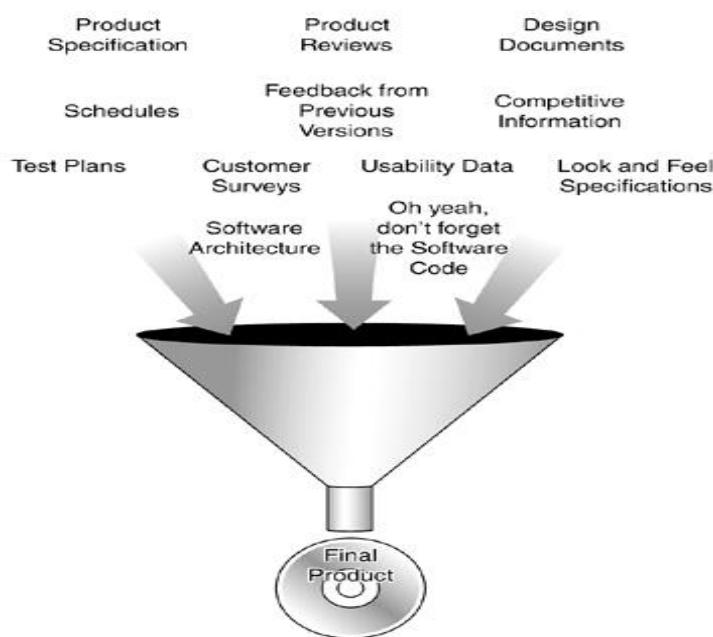
2.1.1. Các thành phần của phần mềm (product components)

Một sản phẩm mềm chính xác là cái gì? Nhiều người cho rằng, đơn giản nó là thứ mà người ta down được từ internet hoặc cài đặt được từ DVD để nó chạy được trên máy tính. Đây là một mô tả tốt, nhưng là một mô tả tốt trong một phạm vi nhỏ, nhưng thật sự, nhiều thành phần được ẩn bên trong phần mềm. Có nhiều phần bên trong hộp “*come in the box*”, mà chúng thường được lấy ra để trợ giúp hoặc có thể

bị bỏ qua. Mặc dù rất dễ quên tất cả các phần này, nhưng là một tester, bạn cần biết về chúng. Bởi vì chúng là những nội dung cần kiểm tra và chúng có thể chứa lỗi.

a) Lỗ lực đằng sau một sản phẩm phần mềm như thế nào?

Trước tiên, bạn hãy nhìn nhận những lỗ lực phía sau một sản phẩm phần mềm. Hình 2.1 chỉ ra một vài những thành phần trừu tượng mà bạn có thể không hề nghĩ tới.



Hình 2.1. Rất nhiều lỗ lực (effort) ẩn dấu phía dưới một sản phẩm phần mềm

Vậy, đâu sẽ là tất cả những thứ này, bên cạnh mã nguồn thật sự, liệu đây có phải là một cái phễu (*funnel*) phần mềm? Nhìn lướt qua, có lẽ chúng là những thứ hiển nhiên mà một lập trình viên tạo ra. Và rõ ràng chúng không phải là những thứ mà có thể được xem trực tiếp từ CD – ROM. Nhưng để dùng một dòng diễn tả cho “món mì ống thương mại”: “chúng ở đây”. Ít nhất cũng là như vậy.

Những thuật ngữ được dùng trong ngành công nghiệp phần mềm để mô tả thành phần một sản phẩm phần mềm, mà đã được tạo ra và tiếp tục tới một nơi nào khác có thể được lạm truyền. Cách dễ nhất để giải thích những thứ mà tất cả sự chuyển giao này là tổ chức chúng thành những loại lớn.

b) Yêu cầu của khách hàng

Phần mềm cần được viết một cách đầy đủ các yêu cầu mà một người hoặc một nhóm người đưa ra đó là những khách hàng. Để làm hợp lý những yêu cầu này, một nhóm phát triển phần mềm phải tìm ra những cái mà khách hàng muốn. Một nhóm thì phỏng đoán những yêu cầu, nhưng hầu hết các thông tin chi tiết được thu thập trong quá trình khảo sát, hỏi đáp từ những phiên bản trước của phần mềm, cạnh tranh các thông tin về sản phẩm, các nhìn tổng quan, các nhóm trọng tâm, và một số các phương thức khác, một số formal, một số các khác. Tất cả những thông tin này được tìm hiểu, xem xét và làm sáng tỏ để quyết định chính xác những đặc trưng nào mà sản phẩm phần mềm cần có.

HÃY ĐƯA CÁC FEATURES (ĐẶC TRƯNG) NÀY VÀO PERSPECTIVE VỚI CÁC FOCUS GROUP (NHÓM TRỌNG TÂM): một phương tiện phổ biến để nhận những hồi đáp trực tiếp từ các khách hàng tiềm năng là sử dụng các *focus group*. *Focus group* thường được tổ chức bởi các công ty khảo sát độc lập - những người thiết đặt các cơ quan trong các phố mua bán lớn. Các cuộc khảo sát hoặc những chuyến đi bộ diễn hình vòng quanh phố mua bán với một bìa kẹp hồ sơ và hỏi những người đi qua nếu họ muốn đóng góp một phần vào quá trình nghiên cứu. Họ sẽ hỏi một vài câu hỏi liên quan đến chất lượng như: “Bạn có một PC ở nhà không? Bạn sử dụng phần mềm X như thế nào? Bạn sử dụng bao nhiêu thời gian để online?” và tiếp tục... Nếu bạn phù hợp với yêu cầu về đối tượng, họ sẽ mời bạn quay trở lại một vài giờ để tham gia cùng với một vài người khác trong *focus group*. Ở đây bạn sẽ được hỏi các câu hỏi chi tiết hơn về phần mềm máy tính. Bạn có thể được biểu diễn những hộp phần mềm khác nhau và hỏi về những sở thích của bạn để bạn lựa chọn. Hoặc bạn có thể thảo luận như một đặc trưng nhóm (*group features*) giống như bạn nhìn thấy một sản phẩm mới. Tốt hơn hết bạn nên bỏ ra chút thời gian của mình. Hầu hết các *focus group* được hướng dẫn nhưng với tư cách là một công ty phần mềm mà yêu cầu thông tin được giữ kín. Nhưng thường thì rất dễ dàng để đoán ra họ là ai.

c) **Đặc tả**

Kết quả của quá trình nghiên cứu các yêu cầu của khách hàng chỉ là những dữ liệu thô. Nó không mô tả được những sản phẩm đề xuất, nó chỉ xác nhận những thứ nên hay không nên tạo ra và các đặc trưng mà khách hàng mong muốn. Bản đặc tả lưu giữ các thông tin trên với các yêu cầu bắt buộc và đưa ra những định hình xem phần mềm sẽ là gì, sẽ làm gì, và trông nó như thế nào.

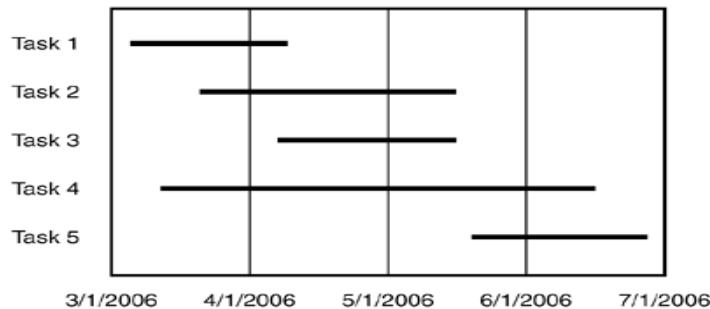
Định dạng của các bản đặc tả thay đổi rất nhiều. Đặc biệt, một số công ty mà các sản phẩm của họ dành cho chính phủ, cho vũ trụ, tài chính, và ngành công nghiệp được phẩm sử dụng một quy trình rất nghiêm khắc với nhiều sự kiểm tra ngặt nghèo và cân nhắc kỹ lưỡng. Kết quả thu được là một bản đặc tả vô cùng kỹ lưỡng, chi tiết được chốt lại, có nghĩa là không được phép thay đổi nó dưới mọi điều kiện. Mọi người trong nhóm phát triển biết chính xác chúng đang tạo nên cái gì.

Đây là các nhóm phát triển phần mềm, thường tạo ra những sản phẩm ít bị chê trách, những người đã đưa ra những bản đặc tả trên bàn ăn (*on cocktail napkins*), nếu họ tạo ra tất cả chúng. Đây là những thuận lợi dễ nhận thấy, chúng rất mềm dẻo, nhưng lại chứa đựng đầy rủi ro. Và sản phẩm cuối cùng không được biết đến cho đến khi nó được tung ra thị trường.

d) Kế hoạch làm việc (schedule)

Một phần rất quan trọng của quá trình sản xuất phần mềm là kế hoạch làm việc của nó. Là một dự án phát triển rất phức tạp với rất nhiều phần và nhiều người cùng tham gia vì vậy, cần một số cơ cấu để theo dõi quá trình xử lý này. Nó có thể là một danh sách các nhiệm vụ đơn giản để hình thành nên biểu đồ *Gantt* (hình 2.2) để theo dõi chi tiết mỗi nhiệm vụ với phần mềm quản lý dự án.

Mục đích của *Schedule* là biết được công việc nào đã được hoàn thành, có bao nhiêu việc bị bỏ quên, và khi nào thì công việc được hoàn thành.



Hình 2.2. Một biểu đồ Gantt biểu diễn các nhiệm vụ của một dự án tương phản với các đường ngang biểu diễn thời gian (horizontal timeline)

e) Tài liệu thiết kế phần mềm

Một nhận thức sai lầm rất phổ biến là khi một lập trình viên tạo ra một chương trình, đơn giản là anh ta ngồi xuống và bắt đầu viết code. Điều này có thể xảy ra trong một cửa hàng phần mềm nhỏ và không chuyên nghiệp. Nhưng ở các công ty lớn, dù là với phần mềm nhỏ nhất cũng phải trải qua quá trình thiết kế để lập kế hoạch về cách mà phần mềm sẽ được viết. Trong cuốn sách này, phần mềm cũng yêu cầu được phác thảo và lập kế hoạch trước khi những đoạn mã đầu tiên được gõ, hoặc được xây dựng.

Những tài liệu mà những lập trình viên tạo ra biến đổi rất nhiều phụ thuộc vào công ty, dự án, và nhóm phát triển, nhưng mục đích của chúng đều là lập kế hoạch và tổ chức mã được viết.

Đây là một danh sách một vài tài liệu thiết kế phần mềm rất phổ biến:

- **Kiến trúc (Architecture):** Một tài liệu mô tả cho toàn bộ thiết kế của phần mềm, bao gồm mô tả của tất cả các thành phần lớn và cách mà chúng gây ảnh hưởng tới các bộ phận khác.

- **Sơ đồ luồng dữ liệu (Data flow diagram):** Một sơ đồ chính thức biểu diễn dữ liệu xuyên suốt một chương trình. Thỉnh thoảng nó tham chiếu tới một *bubble chart* bởi vì nó sẽ gây chú ý hơn với các vòng tròn (*circle*) và các dòng (*line*)

- Sơ đồ chuyển trạng thái (*State transition diagram*): Một sơ đồ chính thức khác phá vỡ phần mềm ở trạng thái cơ bản, hoặc các điều kiện, và biểu diễn các phương tiện di chuyển từ trạng thái này đến trạng thái khác.

- Sơ đồ luồng (*Flow chart*): Các phương tiện truyền thống diễn đạt bằng hình ảnh mô tả luồng logic của phần mềm. Ngày nay, flowcharting không còn phổ biến, nhưng khi nó được sử dụng, viết code từ một flowchart chi tiết là một quá trình xử lý đơn giản.

- Mã chú giải (*commented code*): Có một cách nói cũ rằng bạn có thể viết code một lần, nhưng nó sẽ được đọc bởi bất kỳ ai, ít nhất là 10 lần. Bởi vậy, những lời chú giải hợp lý cho các đoạn code là rất quan trọng, vì vậy, các lập trình viên được giao nhiệm vụ bảo trì code có thể dễ dàng hiểu được đoạn mã đó làm gì và làm như thế nào.

f) Tài liệu kiểm thử

Là thành phần không thể thiếu để tạo nên một sản phẩm phần mềm. Với các lý do này, các lập trình viên phải lập kế hoạch và xây dựng tài liệu cho công việc của họ, tester phải hiểu rõ điều này. Không ai nghe thấy rằng một nhóm kiểm thử phần mềm phải tạo ra nhiều khả năng chuyển giao (*deliverables*) hơn các lập trình viên.

Đây là một danh sách *test deliverables* quan trọng:

- Kế hoạch kiểm thử (*test plan*) mô tả toàn bộ các phương thức được sử dụng để thay đổi phần mềm sao cho phù hợp với bản đặc tả và các yêu cầu của khách hàng. Nó bao gồm mục tiêu về chất lượng, các yêu cầu về tài nguyên, kế hoạch làm việc, những nhiệm vụ được giao, phương thức, và những thứ tương tự như thế (*and so forth*)

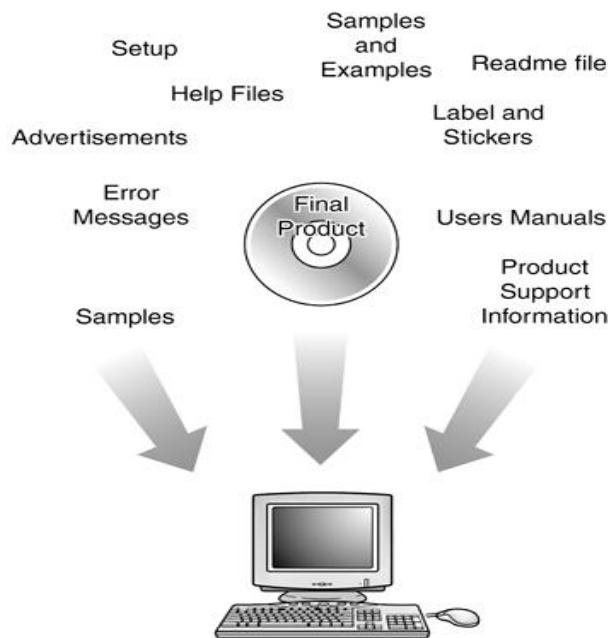
- Danh sách các trường hợp kiểm thử (*test case list*) Những phần sẽ được kiểm tra và mô tả từng bước chi tiết và sẽ được thực hiện theo để kiểm tra phần mềm.

- Báo cáo lỗi (*bug reports*) mô tả các vấn đề được phát hiện nhờ các *test case*. Có thể chúng không được ghi ra giấy nhưng chúng sẽ được theo dõi qua database.

- Các công cụ kiểm thử và kiểm thử tự động (*Test tools and automation*) được mô tả chi tiết trong bài 13. Nếu nhóm của bạn sử dụng các công cụ tự động để kiểm thử phần mềm, thì hoặc là chúng được mua hoặc được tự viết, và chúng đưa ra kết quả bằng tài liệu.

g) Thành phần tạo nên một sản phẩm phần mềm

Như vậy, trong bài này bạn đã được tìm hiểu về các lỗ lực để tạo ra một sản phẩm phần mềm. Cũng cần phải nhận thức rằng khi một sản phẩm đã sẵn sàng để được đóng gói và mang đi thì không phải mỗi code được chuyển đi, còn rất nhiều những bộ phận khác đi cùng với nó (hình 2.3). Bởi vì tất cả những phần này cũng được này cũng được thấy và sử dụng bởi khách hàng, chúng cũng cần được kiểm tra.



Hình 2.3: CD-ROM phần mềm là một trong rất nhiều phần tạo nên một sản phẩm phần mềm

Nhưng thật không may, các thành phần này thường xuyên bị bỏ qua trong quy trình kiểm tra phần mềm. Chắc hẳn rằng bạn cũng đã thử sử dụng các trợ giúp gắn liền với sản phẩm và thấy nó không được tiện dụng lắm thậm chí rất là tồi tệ. Hoặc có lẽ bạn đã kiểm tra yêu cầu hệ thống trên một sticker ở bên cạnh hộp phần mềm (*software box*) chi khám phá ra sau khi bạn mua phần mềm nhưng nó lại không hoạt động trên PC của bạn. Dường như, việc kiểm thử là rất đơn giản, nhưng không hẳn thế, hãy kiểm tra lại chúng một lần nữa trước khi đưa phần mềm ra thị trường. Bạn sẽ làm vậy chứ.

Sau khi đọc cuốn sách này, bạn sẽ được biết về những bộ phận không phải là phần mềm này (*non-software pieces*) và cách để kiểm tra chúng một cách hợp lý. Sau đó, hãy giữ lại danh sách này trong đầu như một ví dụ rằng một sản phẩm phần mềm thì không chỉ là code:

Help files	User's manual
Samples and examples	Labels and stickers
Product support info	Icons and art
Error messages	Ads and marketing material
Setup and installation	Readme file

ĐÙNG QUÊN KIỂM TRA NHỮNG THÔNG ĐIỆP LỖI: thông điệp lỗi (error message) là những phần dễ bị bỏ qua nhất trong một sản phẩm phần mềm. Các lập trình viên, không phải những người thực sự có kinh nghiệm, mà cụ thể là những người viết ra chúng. Hiếm khi họ lập kế hoạch mà thường sửa chữa dần chương trình trong khi kiểm tra các lỗi. Vì vậy, thật khó khăn khi các tester muốn tìm thấy và hiển thị đầy đủ các lỗi. Đừng đưa ra những thông điệp lỗi gây sơ sẩy trong phần mềm của bạn.

Error: Keyboard not found. Press F1 to continue.

Can't instantiate the video thing.

*WindowPs has found an unknown device and is installing a driver
for it.*

A Fatal Exception 006 has occurred at 0000:0000007.

2.1.2. Các nhân lực của dự án phần mềm (Software Project Staff)

Bây giờ, bạn đã biết những thứ bên trong một phần mềm và những lợi ích của nó. Đây chính là thời điểm thích hợp nhất để bạn tìm hiểu về những con người tạo nên một phần mềm. Dĩ nhiên, tùy thuộc vào các công ty và các dự án được đề cập tới, điều này sẽ còn thay đổi rất nhiều. Nhưng hầu hết các quy tắc là giống nhau, chỉ khác nhau ở tên gọi.

Danh sách dưới đây (không có một sự sắp xếp đặc biệt nào cả) bao gồm những người tham gia chính và những công việc mà họ phải làm. Những cái tên thông dụng nhất được sử dụng, nhưng chúng ta vẫn chờ đợi những sự thay đổi và bổ sung phù hợp:

- Những người quản lý dự án, quản lý chương trình, hoặc quản lý sản phẩm(*Project managers, program managers, hoặc producers*): điều khiển dự án từ khi bắt đầu đến khi kết thúc. Thường thì họ chịu trách nhiệm về việc viết các bản đặc tả (*product spec*), quản lý kế hoạch thực hiện công việc (*schedule*), và đưa ra những quyết định đảm bảo sự phối hợp tốt nhất (*trade-offs*).
- Các kỹ sư hệ thống và kiến trúc (*Architects or system engineers*): là những chuyên gia về công nghệ của nhóm xây dựng sản phẩm. Thường thì họ là những người có nhiều kinh nghiệm và có đủ khả năng để thiết kế toàn bộ kiến trúc hệ thống, hoặc thiết kế phần mềm. Có kết hợp rất chặt chẽ với các lập trình viên.
- Các lập trình viên, người phát triển dự án, hoặc người viết code (*Programmers, developers, or coders*): thiết kế, viết phần mềm, và sửa lỗi được tìm thấy. Họ kết hợp rất chặt chẽ với những người quản lý dự án và người xây dựng kiến trúc phần mềm để tạo nên phần mềm. Sau đó, họ cùng làm việc với người quản lý và tester để sửa lỗi phần mềm.
- *Testers* hoặc nhân viên *QA* (*Quality Assurance*: có nhiệm vụ tìm kiếm và báo cáo các vấn đề mà phần mềm gặp phải. Họ làm việc rất mật thiết với tất cả

các thành viên của dự án. Họ phát triển, chạy các trường hợp test và báo cáo các vấn đề họ phát hiện ra.

- Người viết các kỹ thuật, trợ giúp người dùng, giáo dục người dùng, người viết thủ công, hoặc hình ảnh minh họa (*Technical writers, user assistance, user education, manual writers, or illustrators*) trên giấy và trên mạng đến với một sản phẩm phần mềm.
- Quản lý cấu hình hoặc xây dựng (*Configuration management or builder*): điều khiển quy trình cùng làm việc (*the process of pulling together*) toàn bộ phần mềm bởi lập trình viên và toàn bộ tài liệu được xây dựng bởi người viết và cùng đặt nó trong một gói đơn.

Như bạn thấy, một vài nhóm người góp phần tạo nên một sản phẩm phần mềm. Trong các đội lớn, có hàng tá hoặc hàng trăm người làm việc cùng nhau. Để kết nối thành công và tổ chức hướng tiếp cận, họ cần lập kế hoạch, một phương thức thực thi từ điểm A đến điểm B. Đây là cái sẽ được mô tả tiếp theo.

2.2. Thực trạng của quá trình kiểm thử phần mềm

Trong phần 1, bạn đã được tìm hiểu những khái niệm cơ bản về kiểm thử phần mềm và quy trình phát triển phần mềm. Những thông tin đã biểu diễn trong các bài này chỉ là ở mức tổng quan, và cho bạn cái nhìn về cách mà các dự án phần mềm có thể hoạt động. Nhưng thật không may, trong thế giới thật, bạn sẽ không bao giờ thấy một phần mềm hoàn hảo theo một bất kỳ một mô hình phát triển phần mềm nào. Bạn sẽ không thể đưa ra được một bản đặc tả chi tiết hoàn toàn đầy đủ mà khách hàng cần và bạn cũng sẽ không đủ thời gian để làm tất cả những bài kiểm tra mà bạn cần phải làm. Không có vấn đề gì cả. Nhưng để trở thành một tester làm việc có hiệu quả, bạn cần phải biết tưởng tượng ra quy trình phần mềm làm việc như thế nào để đạt được mục đích.

Mục đích của bài này là làm dịu đi tác động của chủ nghĩa lý tưởng lên quá trình kiểm tra thực tế trên phần mềm. Nó sẽ giúp bạn thấy rằng, trong thực tế, **sự thỏa hiệp và nhượng bộ phải xuyên suốt vòng đời phát triển phần mềm**. Nhiều điều trong những sự thỏa hiệp này là liên quan trực tiếp đến nỗ lực kiểm thử

phần mềm. Những lỗi mà bạn tìm thấy và những vấn đề mà bạn ngăn chặn, tất cả đều có ảnh hưởng đặc biệt tới dự án của bạn. Sau khi đọc bài này, bạn sẽ thu nhận được rất nhiều quy tắc, sự tiếp xúc, và những khả năng hồi đáp mà tester cần và hi vọng rằng bạn sẽ đưa ra những quyết định giúp kiến tạo ra một sản phẩm phần mềm.

Trọng tâm của bài này bao gồm:

- Tại sao phần mềm không bao giờ là hoàn hảo
- Tại sao kiểm thử phần mềm không phải là một vấn đề mang tính chất khuôn mẫu
- Những thuật ngữ phổ biến được sử dụng trong kiểm thử phần mềm

2.2.1. Phương châm của việc kiểm thử

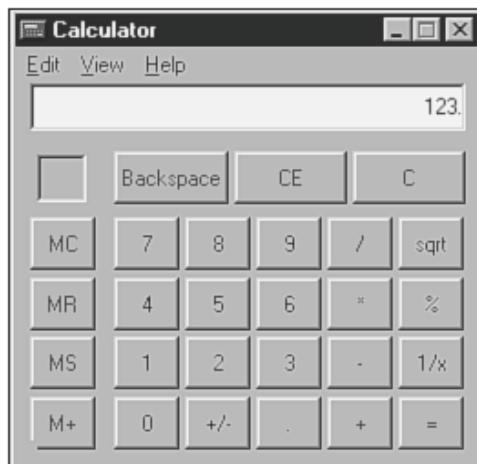
Đoạn đầu của bài này là một danh sách các phương châm hoặc các chân lý. Hãy coi chúng giống như “*quy tắc đi đường*” (“rules of the road”) hoặc “*chân lý của cuộc sống*” (“facts of life”) dành cho quá trình kiểm thử hoặc phát triển phần mềm. Mỗi một phương châm này là một sự hiểu biết nhỏ giúp họ đặt một số khía cạnh của toàn bộ quá trình xử lý vào viễn cảnh tương lai.

a) Tầm quan trọng của việc kiểm thử đầy đủ một chương trình:

Là một tester, bạn có thể tin rằng bạn có khả năng tiếp cận với một khía cạnh của phần mềm, kiểm tra nó, tìm ra tất cả các lỗi, và đảm bảo rằng phần mềm là hoàn hảo. Nhưng thật không may, điều này là không thể được, thậm chí là với một chương trình rất đơn giản, vì 4 lý do sau:

- Số lượng các dữ liệu có thể là đầu vào là rất lớn
- Số lượng các dữ liệu có thể đưa ra cũng vô cùng lớn
- Số lượng các “lỗi đi” trong phần mềm là rất lớn
- Đặc tả phần mềm có tính chất chủ quan. Bạn có thể nói rằng lỗi là những khuyết điểm dưới con mắt của độc giả.

Tất cả các trường hợp trên nếu kết hợp cùng nhau, bạn sẽ thu được một *tập các điều kiện vô cùng lớn đến mức không thể thử hết được*. Nếu bạn không tin điều này thì có thể xem xét trong hình 2.4, phần mềm Microsoft Windows Calculator.



Hình 2.4. . Thậm chí một chương trình đơn giản như Windows Calculator cũng quá phức tạp để kiểm thử đầy đủ

- Khi bạn được phân công kiểm tra phần mềm Windows Calculator. Bạn quyết định là sẽ bắt đầu kiểm tra phép cộng. Bạn thử nghiệm xem $1+0=?$ Bạn nhận được câu trả lời là 1. Phép kiểm tra này cho kết quả đúng. Sau đó, bạn tiếp tục kiểm tra $1+1=?$ Kết quả nhận được là 2. Bạn đi bao xa? Máy tính chấp nhận một số có 32 chữ số, vì vậy, bạn phải cố gắng thử tất cả các khả năng có thể:

$$1+9999999999999999999999999999=$$

Sau lần đầu tiên, bạn hoàn thành chuỗi số trên, bạn cũng có thể thử trên các phép toán $2+0=?$, $2+1=?$, $2+2=?$... Và cứ tiếp tục như vậy. Cuối cùng, bạn sẽ phải thử nghiệm trên phép tính:

$$9999999999999999999999999999+999999999999999999999999=$$

- Tiếp theo bạn sẽ phải thử trên các giá trị thập phân: $1.0+0.1=?$, $1.0+0.2=?$
- Một khi bạn đã kiểm tra tính đúng đắn của tổng các số, không nên kiểm tra các giá trị liên tiếp, bạn cần cố gắng đưa vào các dữ liệu bất hợp lý để kiểm tra tính đúng đắn của phần mềm. Hãy nhớ rằng, *bạn không được phép giới*

hạn những số mà người sử dụng nhập vào, họ có quyền nhấn bất kỳ phím nào trên bàn phím. Những giá trị nên được thử nghiệm có lẽ là: 1+a, z+1, 1a1+2b2,... Như vậy, thì sẽ có đến hàng tỷ trường hợp cần kiểm tra.

- Những dữ liệu được biên tập cũng phải được kiểm tra. Phần mềm *Windows Calculator* cho phép sử dụng các phím *Backspace* và *Delete*. Vì vậy, bạn nên thử nghiệm với chúng. Ví dụ, $1<\text{backspace}>2+2$ phải cho ra kết quả là 4. Những thứ mà bạn đã thực hiện kiểm tra trong chừng mực nào đó, phải được kiểm tra lại bằng cách nhấn vào phím *Backspace* cho mỗi “lối vào” (*entry*), cho mỗi cặp “lối vào” (*two entries*), và cứ tiếp tục như vậy.
- Nếu bạn hoặc nhân viên của bạn được giao nhiệm vụ hoàn thiện tất cả các trường hợp này. Sau đó, bạn có thể tiếp tục tiến hành trên 3 chữ số, sau đó là 4 chữ số,...

Có rất nhiều lối (*entry*) vào có thể khiến bạn không bao giờ hoàn thành được chúng, thậm chí, nếu bạn sử dụng một siêu máy tính để chuyển dữ liệu vào *Calculator*. Nếu bạn quyết định loại bỏ một vài điều kiện kiểm tra bởi vì bạn thấy chúng dư thừa hoặc không cần thiết, hoặc chỉ để tiết kiệm thời gian (*or just to save time*), thì có nghĩa là bạn đã không kiểm tra chương trình một cách đầy đủ.

b) Kiểm thử phần mềm là một bài kiểm tra phụ thuộc vào sự rủi ro

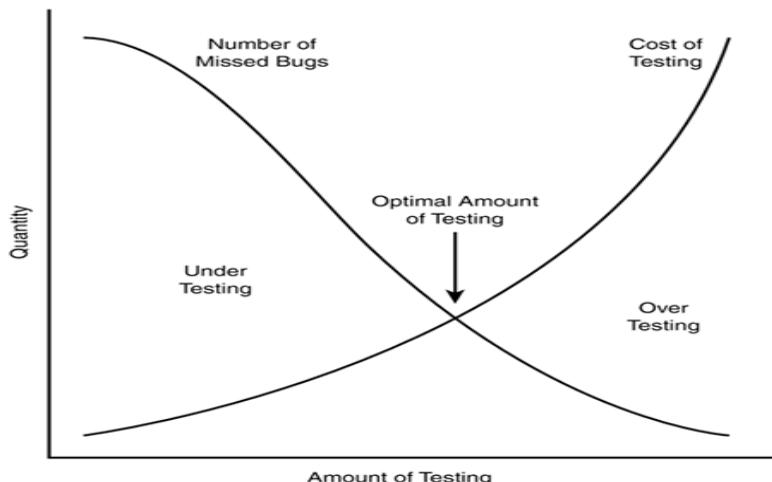
Nếu bạn quyết định không kiểm tra mọi trường hợp kiểm thử, bạn sẽ phải chịu trách nhiệm về những rủi ro. Trong ví dụ về *Calculator*, trường hợp mà bạn lựa chọn để kiểm thử có lẽ là những trường hợp thông thường: $1024+1024=2048$? Và có thể, lập trình viên đã tình cờ loại bỏ lỗi trong hoàn cảnh này. Nhưng trong những trường hợp không được kiểm tra, bạn cũng không thể đảm bảo rằng nó không có lỗi, và sẽ đến lúc khách hàng khám phá ra nó. Và khi đó, chi phí cho việc sửa lỗi sẽ là lớn hơn rất nhiều so với việc sửa lỗi ngay từ đầu.

Điều này thật đáng sợ (*This may all sound pretty scary*). Bạn không thể kiểm tra mọi thứ, và nếu không kiểm tra mọi trường hợp thì bạn sẽ bỏ sót lỗi. Sản phẩm

phải được tung ra thị trường, vì vậy, bạn cần dừng việc kiểm tra, nhưng nếu dừng quá sớm thì một số vùng sẽ không được kiểm thử. Bạn phải làm như thế nào?

Một nội dung quan trọng mà tester cần phải tìm hiểu là làm thế nào để giảm số lượng các trường hợp kiểm thử rất lớn thành một tập các *test case* có thể thực thi được, và làm thế nào để sáng suốt lựa chọn những quyết định ít rủi ro nhất. Điều này buộc *tester* phải xác định được đâu là vấn đề quan trọng và đâu là vấn đề không quan trọng.

Hình 2.5 mô tả mối quan hệ giữa số lượng các trường hợp test với số lượng các lỗi được tìm thấy. Nếu bạn cố thử kiểm tra mọi thứ, chi phí có thể tăng lên đột ngột và những lỗi bị bỏ quên sẽ giảm xuống thấp nhất, nhưng cũng sẽ không còn chi phí để tiếp tục dự án. Nếu bạn cắt giảm công việc kiểm thử thì chi phí cho nó sẽ ít, nhưng bạn sẽ bỏ quên rất nhiều lỗi. Mục đích là bạn phải lọc ra số các trường hợp kiểm thử tối ưu, để đảm bảo bạn không phải kiểm thử quá nhiều hay quá ít các trường hợp.



Hình 2.5: Mọi dự án phần mềm đều có một điểm nỗ lực kiểm thử tối ưu

Bạn sẽ được tìm hiểu làm thế nào để thiết kế và lựa chọn các kịch bản kiểm thử (*test scenarios*) sao cho ít rủi ro nhất và quá trình kiểm tra là tối ưu nhất.

c) Quá trình kiểm thử không thể biểu diễn những lỗi không tồn tại

Hãy nghĩ về điều này trong chốc nát. Bạn là một “kẻ hủy diệt” (*exterminator*) với bài kiểm tra các lỗi. Bạn xem xét hàng giờ và tìm ra dấu vết của các lỗi, lỗi này có thể vẫn đang tồn tại (*live bug*), đã được sửa (*dead bug*), hoặc còn đang tiềm ẩn (*nest*). Bạn có thể nói một cách an toàn rằng “*the house has bugs*”

Bạn đến thăm một “*house*” khác. Lần này, bạn không tìm thấy dấu vết của lỗi. Bạn hãy nhìn vào tất cả những địa điểm rõ ràng (*obvious place*) và tìm xem không có dấu hiệu nào của sự tàn phá. Có lẽ bạn nên tìm một vài lỗi đã từng được xử lý hoặc tiềm ẩn từ lâu, nhưng bạn hãy coi như không thấy gì cả. Có thể bạn tuyên bố một cách chắc chắn rằng “*the house is bug free*”? Không. Kết luận cuối cùng, có thể bạn không tìm thấy một *live bug* nào cả. Ngược lại, bạn tháo gỡ hoàn toàn *the house* thành *foundation*, bạn không thể chắc chắn rằng: bạn không bỏ quên một số lỗi đơn giản.

Tester làm việc chính xác như một “kẻ hủy diệt”. Nếu có thể biểu diễn những lỗi đang tồn tại, nhưng không thể biểu diễn những lỗi không tồn tại. Bạn có thể thực hiện bài kiểm tra của bạn, tìm và báo cáo các lỗi, nhưng bạn có thể kết luận rằng: lỗi không được tìm thấy nữa. Bạn có thể tiếp tục kiểm tra và khả năng tìm thấy lỗi là lớn hơn.

d) Những lỗi được tìm thấy và những lỗi không thể tìm thấy (The More Bugs You Find, the More Bugs There Are)

Thậm chí, có rất nhiều điểm tương đồng giữa real bug và software bug. Cả hai loại này đều cần đưa vào một nhóm. Thường thì một tester sẽ chạy phần mềm như thể nó không có một lỗi nào. Sau đó, anh ta sẽ tìm ra một lỗi rồi những lỗi khác, lỗi khác nữa. Có một vài lý do cho điều này:

Các lập trình viên có những ngày thật tội tệ: Giống như tất cả chúng ra, những người lập trình có thể có những lúc không được minh mẫn lắm. Vào một thời điểm này code có thể được viết rất hoàn hảo, nhưng lúc khác anh ta lại viết code rất cẩu thả. Một lỗi có thể là một dấu hiệu tell – tale rất quen thuộc.

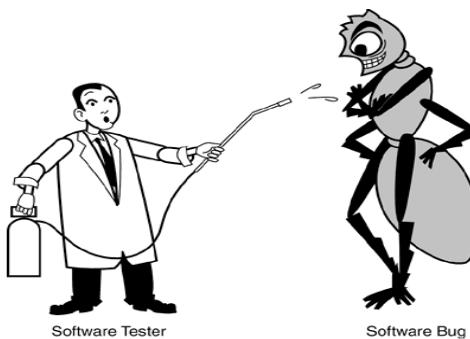
Một lập trình viên thường xuyên mắc những lỗi tương tự nhau: Ai cũng có những thói quen. Một lập trình viên thiên về một loại lỗi nào đó mà thường xuyên mắc đi mắc lại.

Một số lỗi thật sự nguy hiểm như đinh của một tảng băng trôi: Thường thì trong các bản thiết kế và kiến trúc của phần mềm đều ẩn chứa một số vấn đề chưa được phát hiện. Tester đã tìm được một số lỗi mà dường như nó không liên quan đến nhau. Nhưng không hẳn thế, những lỗi này lại có những quan hệ mật thiết với nhau và đều xuất phát từ một lý do chính vô cùng quan trọng.

Vấn đề quan trọng bây giờ là cần chú ý rằng ngược với ý tưởng “bugs follow bugs” này cũng có thể coi là đúng. Nếu bạn không thể tìm ra lỗi của phần mềm thì cũng không có vấn đề gì. Sẽ rất thuận lợi nếu các đặc trưng mà bạn kiểm tra được viết một cách trong sáng và quả thực sẽ có một vài điều nếu như bất kỳ một lỗi nào được tìm ra.

e) Nghịch lý về thuốc trừ sâu (The Pesticide Paradox)

Vào năm 1990, Boris Beizer, trong cuốn sách Software Testing Techniques, tái bản lần 2, đã xây dựng thuật ngữ Pesticide Paradox để mô tả một hiện tượng bạn kiểm thử phần mềm. Những điều tương tự với hiện tượng này đã xảy ra khi bạn dùng pesticides để diệt sâu bọ (mô tả trong hình 2.6). Nếu bạn liên tục dùng một loại pesticide giống nhau, sâu bọ sẽ kháng cự lại thuốc, và pesticide không còn hiệu quả nữa.



Hình 2.6: Phần mềm đã phải trải qua những phép thử lặp đi lặp lại tương tự nhau để chống lại các lỗi

Hãy nhớ về mô hình xoắn ốc của quy trình phát triển phần mềm được mô tả trong bài 2. Quá trình kiểm thử cũng phải lặp đi lặp lại mỗi lần quanh vòng lặp. Với mỗi lần lặp lại, tester nhận phần mềm để kiểm tra và chạy các trường hợp kiểm thử của họ. Cuối cùng, ngoài một vài trường hợp phần mềm chạy đúng yêu cầu, thì các trường hợp kiểm tra của tester sẽ tìm ra và phơi bày các lỗi. Nhưng ở lần lặp

sau, nếu tester vẫn tiếp tục chạy các trường hợp kiểm thử này, chúng sẽ không giúp tìm ra lỗi mới.

Để cách vượt qua pesticide paradox, tester phải viết thêm các trường hợp kiểm thử khác nữa, và tìm những cách tiếp cận mới để kiểm tra chương trình và tìm ra nhiều lỗi hơn.

f) Không phải tất cả các lỗi mà bạn phát hiện sẽ được sửa

Một trong những điều thật sự đáng buồn của kiểm thử phần mềm là sau tất cả những lỗ lực cố gắng làm việc của bạn, không phải tất cả các lỗi bạn phát hiện ra sẽ được sửa. Nhưng điều này cũng không gây thất vọng bởi vì nó không có nghĩa rằng bạn đã làm sai điều gì khi cố gắng thực hiện mục đích của mình, cũng không có nghĩa rằng bạn cùng với cả đội của bạn sẽ phải chấp nhận giao cho khách hàng một sản phẩm kém chất lượng. Tuy nhiên, nó có nghĩa rằng, bạn sẽ cần dựa trên những cặp tiêu chí về nghề tester đã được liệt kê trong bài 1. Bạn và đội của bạn cần mô tả lại những quyết định dựa trên sự rủi ro cho riêng từng lỗi và cho tất cả các lỗi, để đưa ra quyết định cái nào sẽ được sửa và cái nào thì không.

Có một số lý do khiến một số lỗi không được fix:

Không đủ thời gian: Trong mọi dự án luôn có rất nhiều feature của phần mềm, nhưng bạn lại có quá ít người để viết mã và kiểm thử chúng, và cũng không đủ khả năng để thay đổi kế hoạch làm việc cho đến khi kết thúc. Nếu bạn đang làm việc cho một chương trình đòi hỏi những thử thách lớn, mà thời hạn hoàn thành đã sắp đến thì bạn buộc phải bỏ qua một số lỗi

Nó không hẳn là một lỗi: Có lẽ bạn cũng đã từng nghe thấy thành ngữ sau: “it’s not a bug, it’s a feature!” Không phải là hiếm những trường hợp: hiểu sai, lỗi do quá trình kiểm tra, hoặc đặc tả thay đổi dẫn đến kết quả có thể phát sinh lỗi trong tương lai

Có quá nhiều rủi ro khi sửa lỗi: Thật không may điều này là quá thường xuyên xảy đến. Phần mềm có thể rất dễ hỏng, các bộ phận gắn kết chẽ với nhau, và đôi

khi chúng giống như “món mì Ý”. Có thể bạn sửa một lỗi lại khiến một lỗi khác xuất hiện. Dưới áp lực về thời gian hoàn thành sản phẩm, dưới một lịch trình kín đặc, thì có lẽ thay đổi phần mềm là một quyết định chừa quá nhiều rủi ro. Có lẽ tốt hơn hết là bỏ qua những lỗi có thể chấp nhận được để tránh phát sinh những lỗi mới, những rủi ro mới.

Nó không đáng để phải sửa: điều này nghe có vẻ bất hợp lý, nhưng nó là sự thật. Những lỗi hiếm khi xuất hiện hoặc những lỗi xuất hiện trong những feature ít sử dụng thì có thể bỏ qua được. Những lỗi “work-around”, tức là có cách để người sử dụng có thể ngăn chặn hoặc tránh được lỗi, thì thường không được sửa. Tất cả các quyết định phải mang tính chất thị trường dựa trên độ rủi ro.

Quá trình xử lý các quyết định này thường bao gồm các tester, các project manager, các coder. Mỗi người sẽ có những cách nhận định về một viễn cảnh xảy ra khi một số lỗi không được sửa. Nếu lỗi không được fix, tester hiểu khách hàng sẽ phải gánh chịu những hậu quả như thế nào. Project manager có tầm nhìn chiến lược và đoán nhận được những hậu quả có thể xảy ra với dự án khi lỗi không được giải quyết. Và coder hiểu được rằng nếu fix lỗi này thì chi phí cho việc đó sẽ lớn như thế nào. Dựa vào đó, họ sẽ đưa ra những lý do tại sao họ nên sửa hoặc không nên sửa các lỗi đó.

CHUYỆN GÌ SẼ XÂY ĐẾN KHI BẠN ĐƯA RA MỘT QUYẾT ĐỊNH SAI:

Hãy nhớ lại về lỗi mà hãng Intel Pentium mô tả trong bài 1. Hãng Intel đã tìm ra lỗi này trước khi sản phẩm được tung ra thị trường, nhưng đội phát triển sản phẩm đã quyết định rằng: nó là một lỗi quá nhỏ và không đáng để phải sửa. Họ có một lịch làm việc quá dày đặc và đã đến hạn hoàn thành sản phẩm. Vì vậy, họ đã quyết định việc sửa lỗi này sẽ được thực hiện trong phiên bản sau của chip.

Nhưng thật không may, lỗi này đã bị khách hàng phát hiện ra. Trong một số khía cạnh của phần mềm, có thể có tới hàng trăm lỗi không được sửa bởi vì họ nhận thấy rằng hiệu quả tích cực của nó là không lớn. Vậy rất khó để có thể nói rằng các quyết định này là đúng hay sai.

g) Một lỗi có tồn tại nhưng không ai phát hiện thì có phải là lỗi không? (When a Bug's a Bug Is Difficult to Say)

Nếu có một vấn đề trong phần mềm, nhưng không một ai phát hiện ra nó, không phải lập trình viên, không phải một tester, và thậm chí là một khách hàng nào đó, thì nó có được gọi là lỗi không?

Một nhóm các tester ở trong một phòng và hỏi chúng tôi câu hỏi này. Bạn sẽ phải thảo luận về vấn đề này. Ai cũng có ý kiến riêng của mình và có thể là bạn cũng thế. Vấn đề ở đây là không có câu trả lời xác định. Câu trả lời phụ thuộc vào bạn và đội phát triển của bạn với việc đưa ra những quyết định tốt nhất cho mình.

Với mục đích của cuốn sách này, bạn hãy tham khảo những quy tắc để xác định một lỗi trong bài 1:

Phần mềm không thực hiện một số thứ giống như mô tả trong bản đặc tả phần mềm

Phần mềm thực hiện một số việc mà bản đặc tả yêu cầu nó không được thực hiện

Phần mềm thực hiện một số chức năng mà bản đặc tả không đề cập tới

Phần mềm không thực hiện một số việc mà bản đặc tả không đề cập tới, nhưng là những việc nên làm

Trong con mắt của người kiểm thử phần mềm là khó hiểu, khó sử dụng, chậm đói với người sử dụng

Quy tắc này sẽ giúp chúng ta lọc ra những tình huống khó xử để đưa ra quyết định. Có một cách khác để suy xét nó. Không phải là hiếm những trường hợp mà 2 người sử dụng có những nhận xét hoàn toàn trái ngược nhau về vấn đề chất lượng phần mềm. Một người thì nói rằng chương trình như vậy là không thể chấp nhận và người còn lại thì quả quyết rằng chương trình này là hoàn hảo. Có thể cả hai đều đúng? Câu trả lời là một người sử dụng sản phẩm theo cách mà rất nhiều lỗi bị bộc lộ. Còn người kia thì không.

Chú ý: Lỗi không được khám phá hoặc chưa từng được chú ý thì thường được gọi là lỗi ngầm (*latent bug*)

Nếu điều này quá khó hiểu thì cũng đừng lo lắng. Hãy đem nó ra thảo luận với đồng nghiệp trong đội kiểm thử của bạn và cố gắng hiểu điều mà họ nghĩ. Hãy lắng nghe những ý kiến khác, kiểm tra ý tưởng của họ và định hình lại suy nghĩ của chính mình. Hãy nhớ lại một câu hỏi quen thuộc: “nếu một cái cây đổ trong rừng và không ai nghe thấy gì cả, vậy nó có tạo ra âm thanh không?” (*If a tree falls in the forest and there's no one there to hear it, does it make a sound?*”).

h) Xây dựng bản đặc tả phần mềm là công việc không bao giờ kết thúc

Phát triển phần mềm có một ván đề. Ngành công nghiệp này đang phát triển quá nhanh: năm ngoái nó có thể là một sản phẩm sắc bén, nhưng năm nay nó đã trở lên lỗi thời. Tại cùng một thời điểm, phần mềm có quy mô lớn hơn, có nhiều *feature* hơn và tương đối phức tạp, dẫn đến kế hoạch phát triển sẽ lâu hơn. Có 2 ván đề đối lập nhau, và kết quả là liên tục phải thay đổi bản đặc tả sản phẩm.

Không có cách nào khác để phản ứng lại sự thay đổi mau lẹ của bản đặc tả. Bạn cho rằng, sản phẩm của bạn đã bị khóa và chúng ta tuyệt đối không thể thay đổi bản đặc tả sản phẩm. Bạn đã đi được nửa chặng đường trong kế hoạch phát triển 2 năm của sản phẩm, đối thủ chính của bạn thì đã tung ra thị trường một sản phẩm hoàn toàn tương tự với sản phẩm của bạn. Mà thậm chí một vài *feature* rất tuyệt vời mà sản phẩm của bạn không có được. Liệu bạn có nên tiếp tục với bản đặc tả của mình và giao cho khách hàng một sản phẩm thua kém hơn không? Hay đội phát triển dự án của bạn nên tập hợp lại, suy ngẫm lại về những *feature* của sản phẩm, viết lại bản đặc tả và làm việc trên một sản phẩm được sửa lại? Trong hầu hết các trường hợp, những nhà kinh doanh sáng suốt sẽ tuyên bố sau cùng.

Là một *tester*, bạn phải thừa nhận rằng bản đặc tả sẽ thường xuyên thay đổi. Các *feature* sẽ được thêm vào mà nó không hề nằm trong kế hoạch kiểm thử. Các *feature* sẽ thay đổi và thậm chí là bị xóa hoàn toàn khi bạn đã kiểm tra và sẵn sàng báo cáo lỗi về nó. Điều này hoàn toàn có thể xảy ra. Bạn sẽ được tìm hiểu các

kỹ thuật linh hoạt để lập kế hoạch và thực thi việc kiểm thử trong phần còn lại của cuốn sách.

k) Tester không phải là thành viên được mọi người chờ đợi trong một dự án

Hãy nhớ lại mục đích của quá trình kiểm thử phần mềm là gì? *Mục đích của một tester là tìm ra lỗi, tìm thấy chúng sớm nhất có thể, và chắc chắn rằng chúng phải được sửa.*

Công việc của bạn là xem xét thật kỹ lưỡng và phê bình công việc của các đồng nghiệp của bạn, phát hiện những vấn đề của công việc, và phải thực hiện công khai những gì bạn tìm thấy. Và bạn sẽ phải cố gắng chiến thắng trong các cuộc tranh luận với các đồng nghiệp.

Hãy giữ thái độ hòa bình với những đồng nghiệp trong đội của bạn:

Phát hiện lỗi thật sớm: Dĩ nhiên, đây là công việc của bạn, và bạn phải kiên trì làm công việc này. Và dĩ nhiên, nếu bạn phát hiện một lỗi nguy hiểm trước 3 tháng thì tốt hơn là 1 ngày trước khi đến thời điểm tung sản phẩm ra thị trường.

Giữ thái độ hăng hái, nhiệt tình: Tốt thoi, bạn thật sự yêu công việc của mình. Bạn sẽ cảm thấy phấn khích khi bạn tìm được một lỗi khủng khiếp. Nhưng nếu bạn huênh hoang dồn ép lập trình viên và nói với anh ta rằng bạn vừa mới tìm được một lỗi kinh khủng nhất (*nastiest bug*) trong suốt quá trình làm việc của bạn, thì chắc hẳn rằng anh ta sẽ cảm thấy khó chịu.

Đừng chỉ báo cáo những thông tin xấu: Nếu bạn đã phát hiện ra một đoạn mã chứa đầy lỗi, hãy nói cho mọi người biết, dù bạn sẽ bị phản đối. Bởi vì nếu bạn chưa từng phát hiện ra lỗi của các lập trình viên, mọi người cũng sẽ tránh xa bạn.

i) Kiểm thử phần mềm là một công việc đòi hỏi tính kỷ luật

Kiểm thử phần mềm là công việc được thực hiện sau khi có sản phẩm. Các sản phẩm phần mềm và không phức tạp. Số lượng người với máy tính sử dụng phần mềm là bị giới hạn và một số ít lập trình viên trong đội dự án của bạn có khả năng gỡ lỗi cho mỗi đoạn mã của người khác. Các lỗi là một vấn đề không tốt.

Chúng xuất hiện và sớm được sửa chữa thì chi phí cho chúng sẽ không nhiều.

Thường thì các tester không được huấn luyện và họ vẫn phát huy khả năng của họ trong các dự án sau để làm thay đổi nhiều thứ.

Hãy nhìn xem sản phẩm phần mềm cần được giúp đỡ và bạn sẽ nhìn thấy một danh sách các tester. Ngành công nghiệp phần mềm đang phát triển vượt bậc với mũi nhọn là đội ngũ tester chuyên nghiệp. Bởi vì hiện tại, người ta đã mất quá nhiều chi phí để xây dựng lên những phần mềm kém chất lượng.

Thật là tội tệ, không phải mọi công ty đều thông nhất quan điểm đó. Nhiều *computer game* và những công ty phần mềm nhỏ vẫn thường xuyên sử dụng những mô hình phát triển lỏng lẻo như *big-bang* hoặc *code-and-fix*. Nhưng bây giờ, nhiều phần mềm được phát triển và luôn tuân thủ kỷ luật. Các tester trở thành lực lượng lõng cốt, những thành viên sống còn trong nhiệm vụ của họ.

Đây sẽ là một vấn đề lớn, nếu bạn là thợ săn thú với kiểm thử phần mềm. Nó đã trở thành một nghề nghiệp được nhiều người lựa chọn và cần phải được đào tạo, làm việc có kỷ luật và thúc đẩy sự tiến bộ.

2.2.2. Các định nghĩa và thuật ngữ kiểm thử phần mềm

Bài này bao gồm một danh sách các thuật ngữ và các định nghĩa. Các thuật ngữ này mô tả những khái niệm nền tảng về quá trình phát triển phần mềm và kiểm thử phần mềm. Bởi vì chúng thường rất lộn xộn và được sử dụng không hợp lý, chúng được định nghĩa ở đây như một cặp để giúp bạn hiểu ý nghĩa thật sự chúng và sự khác nhau giữa chúng. Hãy ý thức rằng nhiều người không bằng lòng về ngành công nghiệp phần mềm với những khái niệm của nhiều công ty, được phổ biến rộng rãi, (đó là các thuật ngữ). Là một tester, bạn nên thường xuyên làm rõ ràng ý nghĩa của các thuật ngữ mà đội của bạn sử dụng. Thường thì đây là cách tốt nhất để một khái niệm được đồng tình hơn là bạn phải cố gắng để mọi người chấp nhận rằng thuật ngữ đó là đúng.

Kiểm thử phần mềm (Software testing): Là một tiến trình thực hiện một chương trình hoặc một ứng dụng với mục đích tìm kiếm các lỗi (bugs) phần mềm.
Các mục tiêu chính của kiểm thử phần mềm :

- Phát hiện càng nhiều lỗi càng tốt trong thời gian kiểm thử xác định trước.
- Chứng minh rằng sản phẩm phần mềm phù hợp với các đặc yêu cầu của nó.
- Xác thực chất lượng kiểm thử phần mềm đã dùng chi phí và nỗ lực tối thiểu
- Tạo các testcase chất lượng cao, thực hiện kiểm thử hiệu quả và tạo ra các báo cáo vấn đề đúng và hữu dụng.

Kiểm thử phần mềm là 1 thành phần trong lĩnh vực rộng hơn, đó là Verification & Validation (V & V), ta tạm dịch là Thanh kiểm tra và kiểm định phần mềm.

Verification (sự kiểm tra) và Validation (sự xác nhận)

Verification và *Validation* thường được sử dụng thay thế cho nhau nhưng thực chất chúng là các khái niệm khác nhau. Sự khác nhau này rất quan trọng trong kiểm thử phần mềm.

Verification là quy trình xác nhận rằng một số khía cạnh của phần mềm là phù hợp với bản đặc tả của nó. *Validation* là quy trình xác nhận rằng phần mềm phù hợp với yêu cầu của người sử dụng.

Thanh kiểm tra (Verification) phần mềm là qui trình xác định xem sản phẩm của 1 công đoạn trong qui trình phát triển phần mềm có thỏa mãn các yêu cầu đặt ra trong công đoạn trước không (Ta có đang xây dựng đúng đắn sản phẩm không?).

Thanh kiểm tra phần mềm thường là hoạt động kỹ thuật vì nó dùng các kiến thức về các artifacts, các yêu cầu, các đặc tả rời rạc của phần mềm.

Các hoạt động Thanh kiểm tra phần mềm bao gồm kiểm thử (testing) và xem lại (reviews).

Kiểm định (Validation) phần mềm là qui trình đánh giá phần mềm ở cuối chu kỳ phát triển để đảm bảo sự bằng lòng sử dụng của khách hàng (Ta có xây dựng phần mềm đúng theo yêu cầu khách hàng?).

Các hoạt động kiểm định được dùng để đánh giá xem các tính chất được hiện thực trong phần mềm có thỏa mãn các yêu cầu khách hàng và có thể theo dõi với các yêu cầu khách hàng không ?

Kiểm định phần mềm thường phụ thuộc vào kiến thức của lĩnh vực mà phần mềm xử lý.

Ví dụ kính thiên văn không gian Hubble (*Hubble space telescope*).

Vào tháng 4 năm 1990, Kính thiên văn không gian *Hubble* (*Hubble space telescope*) được đưa vào quỹ đạo quanh trái đất. Là một thiết bị phản chiếu, *Hubble* sử dụng một 1 tấm gương lớn như một phương tiện chính để khuếch đại đối tượng mà nó nhắm tới. Quá trình chế tạo tấm gương này là đòi sự chính xác và tập trung

tuyệt đối. Kiểm tra tấm gương rất khó, từ khi chiếc kính thiên văn này được thiết kế để sử dụng trong không gian và người ta không thể xác định được vị trí, thậm chí là nó có tầm nhìn xuyên suốt ngay cả khi nó vẫn ở trên trái đất. Với những lý do này thì chỉ có một cách kiểm tra tốt nhất là đo đạc cẩn thận tất cả các thuộc tính của nó và so sánh với những tiêu chuẩn đã được chỉ ra. Quá trình kiểm tra này đã được thực thi và *Hubble* được tuyên bố là đã sẵn sàng.

Nhưng thật không may, ngay sau khi nó được đưa vào quỹ đạo hoạt động, các bức ảnh nó gửi về không hề có trung tâm. Tổ chức điều tra đã khám phá ra rằng tấm gương đã được chế tạo không hợp lý. Khi ở trên mặt đất, tấm gương này đã được sản xuất theo đúng bản đặc tả, nhưng bản đặc tả này lại sai. Tấm gương vô cùng *precise* nhưng nó không *accurate*. Quá trình kiểm tra đã xác nhận rằng tấm gương được sản xuất đã đáp ứng được sự kiểm tra của bản đặc tả (*spec verification*), nhưng nó không xác nhận được rằng nó đáp ứng được yêu cầu cơ bản (*original requirement validation*).

Năm 1993, một phái đoàn trên con thoi đã sửa kính thiên văn *Hubble* bằng cách cài đặt một “*corrective lens*” để lấy lại trung tâm của những bức ảnh được chụp bởi *Hubble*.

Mặc dù không có một ví dụ về phần mềm, *verification* và *validation* áp dụng tốt như nhau với quá trình kiểm thử. Chứa từng có bản đặc tả nào là đúng. Nếu bạn thay đổi bản đặc tả và thông qua sản phẩm cuối cùng, thì bạn sẽ tránh được những vấn đề như với chiếc kính thiên văn *Hubble*.

Sai sót (error): Là một sự nhầm lẫn hau một sự hiểu sai trong quá trình phát triển phần mềm của người phát triển

Khiếm khuyết, lỗi (bug, defect, faults): Xuất hiện trong phần mềm như là một kết quả của một sai sót khi lập trình viên thiết kế và xây dựng chương trình.

Hỗn hối (failure): là kết quả của khiếm khuyết có trong ứng dụng hoặc sản phẩm, làm cho chương trình không hoạt động được hay hoạt động nhưng cho kết quả không như mong đợi.

Kiểm thử viên (Tester): Là người thực hiện kiểm thử

Ca kiểm thử (Test Case):

- Theo chuẩn IEEE 610 (1990) định nghĩa test case là một các dữ liệu đầu vào kiểm thử, các điều kiện thực hiện, và các kết quả mong đợi được phát triển cho một mục tiêu cụ thể như một đường thực thi chương trình hoặc xác minh sự tuân thủ với một yêu cầu xác định.
- Một test case là một tài liệu gồm có một tập dữ liệu test, các tiền điều kiện, các kết quả mong đợi và các hậu điều kiện, được phát triển cho một kịch bản test (Test scenario) cụ thể để xác minh sự tuân thủ một yêu cầu cụ thể.

Test case đóng vai trò như là điểm khởi đầu cho việc thực hiện kiểm tra và sau khi áp dụng một tập các giá trị đầu vào, ứng dụng có một kết quả nhất định và rời khỏi hệ thống tại một số điểm kết thúc hoặc được xem như hậu điều kiện thực hiện.

Các tham số điển hình của test case:

- Test Case ID
- Test Scenario
- Test Case Description
- Test Steps
- Prerequisite
- Test Data
- Expected Result
- Test Parameters
- Actual Result
- Environment Information
- Comments

Ví dụ:

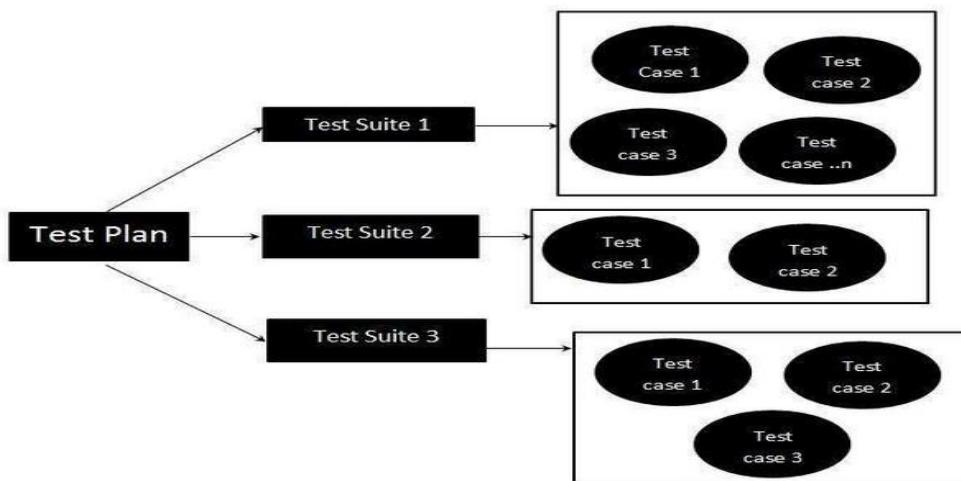
Chúng ta nói rằng chúng ta cần kiểm tra một trường dữ liệu vào mà chấp nhận mười ký tự. Trong khi phát triển các test case cho kịch bản ở trên, các test case được tài liệu hóa như sau. Trong ví dụ dưới, trường hợp đầu là kịch bản qua (pass), kịch bản thứ hai là không qua (fail).

Scenario	Test Step	Expected Result	Actual Outcome
Verify that the input field that can accept maximum of 10 characters	Login to application and key in 10 characters	Application should be able to accept all 10 characters.	Application accepts all 10 characters.
Verify that the input field that can accept maximum of 11 characters	Login to application and key in 11 characters	Application should NOT accept all 11 characters.	Application accepts all 10 characters.

Nếu kết quả mong đợi không phù hợp như kết quả thực tế, thì chúng ta ghi nhận là một defect. Defect đó đi qua chu kỳ sống của defect.

Một bộ test (Test Suite): Bộ test gồm có nhiều test. Nó có thể có một trong ba trạng thái Active, Inprogress và completed. Một test case có thể được thêm vào nhiều bộ test và kế hoạch test. Sau khi tạo test plan, test suit được tạo, trong test suit chứa nhiều test case.

Test Suite - Diagram:



Hình 2.7. Quan hệ giữa ba khái niệm

Quality (chất lượng) và reliability (sự đáng tin cậy)

Trong cuốn từ điển của trường cao đẳng Merriam-Webster đã định nghĩa rằng *quality* là “độ đo sự hoàn hảo” hoặc “sự vượt chội về thứ hạng”. Nếu sản phẩm phần mềm có chất lượng cao, nó sẽ đáp ứng được nhu cầu của khách hàng. Khách hàng sẽ cảm thấy sản phẩm hoàn hảo và nó sẽ được xếp thứ hạng cao hơn trong danh sách lựa chọn của khách hàng.

Tester có thể cảm thấy 2 khái niệm *quality* và *reliability* là gần nhau. Họ cảm thấy rằng nếu như họ có thể kiểm tra một chương trình cho đến khi nó chạy ổn định và có thể tin tưởng được (*reliability*). Khi đó, họ có thể quả quyết rằng sản phẩm đã đạt chất lượng tốt. Nhưng thật không may, điều này không hẳn đã đúng.

Reliability chỉ là một khía cạnh của *quality*.

Quan niệm về *quality* của người sử dụng phần mềm có thể bao gồm cả sự thoải mái của các *feature*, sản phẩm có khả năng chạy trên cả những PC cũ, dịch vụ hậu mãi của các công ty phần mềm, và thường bao gồm cả giá cả của sản phẩm. Sự tin tưởng hoặc cách thức mà phần mềm thâm nhập vào dữ liệu của khách hàng, có thể là rất quan trọng, nhưng không phải lúc nào cũng thế.

Chắc rằng, với một chương trình có chất lượng cao và đáng tin cậy, thì *tester* phải kiểm tra và thông qua trong suốt quá trình phát triển sản phẩm.

Testing (Kiểm thử) và Quality Assurance (đảm bảo chất lượng) (QA)

Cặp khái niệm cuối cùng là *testing* và *quality assurance* (có thể viết tắt là QA). Hai thuật ngữ này, một cái thường được sử dụng để mô tả nhóm hoặc quá trình kiểm tra và xác nhận chất lượng phần mềm. Bạn sẽ được tìm hiểu nhiều hơn về thước đo chất lượng phần mềm, nhưng trước tiên hãy xem xét những khái niệm sau:

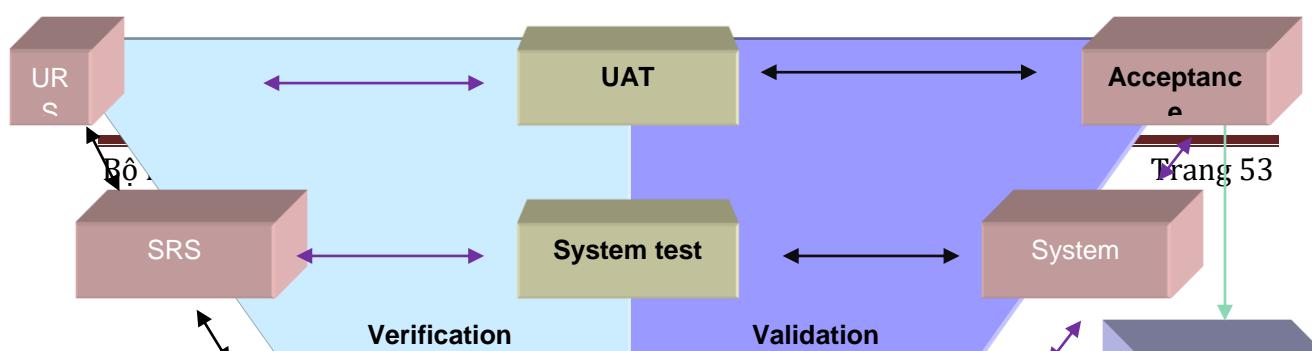
Mục đích của *testing* là tìm ra lỗi, tìm thấy chúng sớm nhất có thể, và đảm bảo rằng chúng đã được sửa.

Trách nhiệm chính của người *QA* là tạo và bắt phần mềm phải tuân theo các chuẩn để cải tiến quy trình phát triển phần mềm và ngăn chặn các lỗi xuất hiện bất cứ lúc nào

Tuy nhiên, 2 khái niệm này vẫn có sự chồng chéo nhau. Một số *tester* sẽ làm nhiệm vụ *QA*, một số thì thực thi việc kiểm tra. Hai công việc này cùng các nhiệm vụ của nó có quan hệ chặt chẽ với nhau. Tuy nhiên khó mà tránh khỏi sự lộn xộn giữa các thành viên làm nhiệm vụ kiểm thử (*testing*) và các thành viên đảm bảo chất lượng phần mềm (*QA*).

2.2.3. Mô hình chữ V

Mô hình chữ V sẽ giúp chúng ta hình dung về quy trình test trong toàn bộ kế hoạch thực hiện dự án



Hình 2.8. V - Model

2.3. Quá trình nghiên cứu bản đặc tả phần mềm

Trong phần này chúng ta sẽ tìm hiểu:

- Khởi động
- Thực hiện duyệt ở mức cao của bản đặc tả
- Kỹ thuật kiểm thử bản đặc tả mức thấp

Bài này sẽ giúp bạn bắt tay kiểm thử một phần mềm thật sự đầu tiên nhưng đó không phải là điều mà chúng ta mong chờ. Bạn sẽ không cài đặt và chạy phần mềm và bạn cũng không đập thình thịch vào bàn phím để hi vọng sẽ thấy phần mềm chạy sai. Trong bài này, bạn sẽ học cách làm thế nào để kiểm thử bản đặc tả của sản phẩm để tìm lỗi trước khi chúng được chuyển thành một phần mềm.

Kiểm tra bản đặc tả không phải kiểm tra những thứ mà tất cả các tester cho rằng đó là công việc xa xỉ, không cần thiết. Đôi khi bạn đang ở giữa quá trình phát triển một dự án, sau khi bản đặc tả đã được viết và việc viết mã cũng đã bắt đầu, người ta phát hiện bản đặc tả không ổn. Nếu gặp hoàn cảnh này thì cũng đừng lo

lắng, bạn vẫn có thể sử dụng các kỹ thuật đã được mô tả ở đây để kiểm tra bản đặc tả cuối cùng.

Nếu bạn không có đủ may mắn và lâm vào cảnh rắc rối với dự án ngay từ đầu và phải xem xét tới bản đặc tả sơ bộ, bài này sẽ dành cho bạn. Các lỗi được tìm kiếm tại giai đoạn này là tiềm năng giúp bạn tiết kiệm được một số lượng lớn tiền của và thời gian cho dự án của bạn.

Trọng tâm của bài này bao gồm:

- Thế nào kiểm thử *black-box* và *white-box*
- Kiểm thử *static* và *dynamic* khác nhau như thế nào
- Kỹ thuật mức cao nào có thể được sử dụng để duyệt lại một bản đặc tả phần mềm
- Vấn đề đặc biệt nào bạn nên tìm kiếm khi duyệt lại bản đặc tả chi tiết

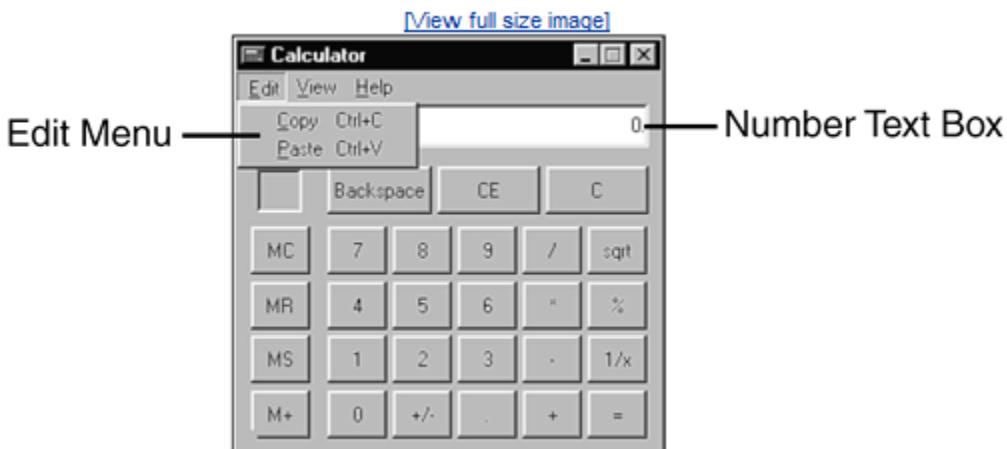
2.3.1. Khởi đầu

Hãy nghĩ về 4 mô hình phát triển phần mềm được mô tả trong bài 2 “Quy trình phát triển phần mềm”: *big-bang*, *code-and-fix*, *waterfall* và *spiral*. Trong mỗi mô hình, ngoại trừ *big-bang*, đội phát triển phần mềm tạo ra một bản đặc tả (*product specification*) từ tài liệu yêu cầu của khách hàng (*requirement document*) để định nghĩa những cái mà phần mềm sẽ làm.

Điển hình, bản đặc tả sản phẩm là tài liệu được viết bằng *word* và có các bức tranh để mô tả sản phẩm dự định. Một trích dẫn từ bản đặc tả phần mềm *Windows Calculator* (trong hình 2.9) có thể đọc một số thứ giống như dưới đây:

Menu Edit sẽ có 2 lựa chọn: *Copy* và *paste*. Có thể chọn bằng một trong 3 cách: trỏ chuột và click vào các item của menu, sử dụng phím truy cập (*alt+E* và sau đó là *C* để *Copy* và *P* để *Paste*), hoặc sử dụng các phím tắt chuẩn của windows như *Ctrl+C* để *Copy* và *Ctrl+V* để *Paste*.

Chức năng Copy sẽ sao chép toàn bộ dữ liệu hiện tại được hiển thị trên ô textbox và đưa vào Windows Clipboard. Chức năng Paste sẽ dán những dữ liệu được lưu trữ trong Windows Clipboard thành dữ liệu trong ô textbox.



Hình 2.9: Phần mềm Windows Calculator chuẩn hiển thị menu drop-down Edit

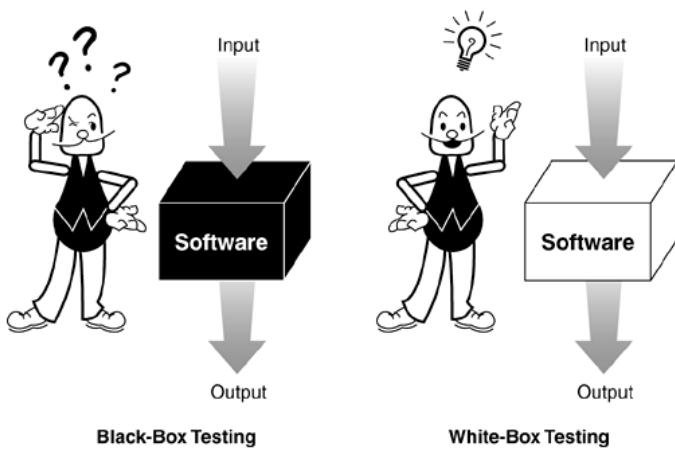
Bạn có thể nhìn thấy rằng, phải mất một đoạn văn bản ngắn để mô tả về việc điều khiển 2 item menu trong một chương trình *Calculator* đơn giản. Một bản đặc tả chi tiết và triệt để cho toàn bộ ứng dụng có thể phải dài hàng trăm trang.

Dường như với cách viết như vậy, chúng ta đã tạo ra một văn bản quá tì mỉ cho một phần mềm đơn giản. Tại sao lại không để cho lập trình viên viết một phần mềm *Calculator* theo ý của anh ta? Vấn đề là bạn sẽ không thể có ý tưởng nào là cuối cùng. Ý tưởng của các lập trình viên sẽ hình thành nên phần mềm, chức năng nào nên có, và người sử dụng sẽ sử dụng nó như thế nào. Chỉ có cách đảm bảo rằng sản phẩm cuối cùng là cái mà khách hàng yêu cầu với kế hoạch hợp lý về những lỗ lực kiểm thử để mô tả thấu đáo sản phẩm trong một bản đặc tả.

Những thuận lợi khác có trong một bản đặc tả, và là nội dung cơ bản của bài này, một tester sẽ có một văn bản về các item để thực hiện kiểm tra. Bạn có thể tìm các lỗi trong chính văn bản đó trước khi dòng mã đầu tiên được viết.

a) Kiểm thử black-box và white-box

Hai thuật ngữ mà các *tester* sử dụng để mô tả cách mà họ tiếp cận để kiểm thử phần mềm là kiểm thử *black-box* và kiểm thử *white-box*. Hình 2.10 biểu diễn sự khác nhau giữa 2 hướng tiếp cận này. Trong kiểm thử *black-box*, *tester* chỉ biết cái mà phần mềm giả định là thực hiện được và anh ta cũng không hề biết cách thức phần mềm hoạt động như thế nào. Nếu anh ta đưa vào một dữ liệu chính xác, anh ta cần nhận được dữ liệu chính xác. Anh ta không hề biết làm cách nào và tại sao điều đó lại xảy ra.



Hình 2.10: Với kiểm thử *black-box*, *tester* không biết chi tiết về cách thức mà phần mềm làm việc

Lời khuyên: Đôi khi, kiểm thử *black-box* được quy về kiểm thử chức năng (*function testing*) hoặc kiểm thử về cách hoạt động (*behavioral testing*). Đừng có cố chạy theo những thuật ngữ được dùng trong thực tế. Đôi của bạn có thể sử dụng những thuật ngữ khác đi. Công việc của bạn là phải hiểu những thuật ngữ này và cách mà chúng được sử dụng trong đội của bạn.

Hãy suy nghĩ về phần mềm *Calculator* được mô tả trong hình 4.1. Nếu bạn đưa dữ liệu vào là 3.14159 và nhấn nút *Sqrt*, bạn sẽ nhận được 1.772453102341. Với kiểm thử *black-box*, không có vấn đề gì khi phần mềm thực hiện tính toán $\sqrt{\pi}$. Nó vừa thực hiện điều này. Là một tester, bạn có thể xác minh kết quả trên một *Calculator* đã được chứng thực khác và xác nhận rằng *Windows calculator* thực hiện chức năng này đúng.

Trong kiểm thử *white-box* (đôi khi còn được gọi là kiểm thử *clear-box*), *tester* có khả năng truy cập vào mã nguồn và có thể kiểm tra nó với các manh mối để giúp ích quá trình kiểm thử. Dựa trên những gì mà *tester* nhìn thấy, anh ta có thể xác định rằng những con số chính xác nào có thể gây lỗi và từ đó anh ta có thể thiết kế bài kiểm tra của mình dựa trên những thông tin này.

Chú ý: Kiểm thử *white-box* có nhiều rủi ro. Rất dễ dàng để phát sinh lỗi khi kiểm thử phần mềm 1 cách khách quan bởi vì bạn có thể thiết kế bài kiểm tra kết nối với điều khiển của mã.

b) Kiểm thử static và dynamic

Có 2 thuật ngữ khác cũng được sử dụng để mô tả cách phần mềm được kiểm tra là kiểm thử *static* và kiểm thử *dynamic*. Kiểm thử *static* quy về việc kiểm tra một số thứ mà nó không phải đang được chạy, đang kiểm tra và đang duyệt lại nó. Kiểm thử *dynamic* là cái mà thông thường bạn nghĩ về cách để kiểm tra, cách chạy và sử dụng phần mềm.

Sự giống nhau nhiều nhất của các thuật ngữ này là quy trình bạn đi xuyên qua khi kiểm tra một chiếc xe ô tô đã được sử dụng. Đá vào vỏ xe, kiểm tra lớp sơn và nhìn xuống dưới mui xe là các kỹ thuật kiểm tra *static*. Nào hãy bắt đầu, lắng nghe tiếng động cơ, và đánh xe trên đường là các kỹ thuật kiểm thử *dynamic*.

c) Kiểm thử static black-box: Kiểm tra bản đặc tả

Kiểm tra bản đặc tả là kiểm thử *static black-box*. Bản đặc tả là một tài liệu, không phải là một chương trình đang chạy, vì vậy mà nó được xem xét tĩnh (*static*). Nó cũng có thể là một số thứ được tạo ra sử dụng dữ liệu từ nhiều đề tài nghiên cứu tiện dụng, các nhóm trọng tâm, đầu vào mang tính thị trường... Bạn không cần phải biết làm cách nào và tại sao các thông tin này lại được sử dụng hoặc các chi tiết của quá trình xử lý đã sử dụng nó. Chúng được tóm tắt lại trong một bản đặc tả sản phẩm. Sau đó, bạn có thể nắm bắt được tài liệu này, thực thi việc kiểm thử *static black-box*, và nghiên cứu cẩn thận về các lỗi.

Ở ngay phần đầu, bạn đã quan sát một ví dụ về bản đặc tả sản phẩm cho phần mềm *Windows Calculator*. Ví dụ này đã sử dụng một tài liệu được viết rất chuẩn với một bức tranh mô tả về cách thức hoạt động của phần mềm. Mặc dù đây là cách thức rất phổ biến cho việc viết một bản đặc tả, có nhiều sự thay đổi. Đội phát triển dự án của bạn có thể làm nổi bật biểu đồ dựa trên các từ hoặc nó có thể sử dụng một ngôn ngữ máy *self-document* ví dụ như Ada. Với bất kể lựa chọn nào của họ, bạn vẫn có thể áp dụng tất cả các kỹ thuật biểu diễn trong bài này. Bạn sẽ phải thiết kế chúng dựa trên định dạng của bản đặc tả mà bạn có, nhưng ý tưởng vẫn giống như vậy.

Bạn phải làm gì nếu dự án của bạn không có một bản đặc tả? Có thể đội của bạn đang sử dụng mô hình *big-bang* hoặc một mô hình *code-and-fix* không chặt chẽ lắm. Là một tester, đây là một vị trí khác. Mục đích của bạn là tìm ra các lỗi sớm nhất có thể, trước khi phần mềm được viết code. Nhưng nếu như sản phẩm của bạn không có một bản đặc tả, dường như điều này là không thể được. Mặc dù, bản đặc tả có thể không được viết ra, một ai đó, hoặc một vài người, biết cái thứ mà họ đang cố gắng để xây dựng. Có thể đó là người phát triển phần mềm, là một quản trị dự án, hoặc là một thương gia. Sử dụng chúng như khi đi dạo, nói chuyện, đặc tả sản phẩm và áp dụng các kỹ thuật giống như đánh giá đánh giá bản đặc tả tinh thần này (*mental specification*), mặc dù nó được viết trên giấy. Thậm chí, bạn có thể từng bước ghi lại các thông tin và tụ họp lại để tính toán lại nó.

Hãy nói với đội dự án của bạn, “Đây là cái mà tôi lập kế hoạch để kiểm tra và đệ trình lại các lỗi”. Bạn sẽ hết sức ngạc nhiên rằng có bao nhiêu chi tiết, họ sẽ ngay lập tức điền đầy nó.

Lời khuyên: Bạn có thể kiểm tra một bản đặc tả với các kỹ thuật *static black-box* mà không có vấn đề gì về định dạng của bản đặc tả. Nó có thể là văn bản viết hoặc tài liệu mô tả hoặc cả hai. Thậm chí, bạn có thể kiểm tra một bản đặc tả không được viết, mà chỉ gồm các câu hỏi của những người đang thiết kế và viết phần mềm

2.3.2. Thực thi quá trình xem xét bản đặc tả ở mức cao

Định nghĩa một sản phẩm phần mềm là một quy trình rất khó. Bản đặc tả phải đề cập đến rất nhiều thứ chưa được biết tới, nó bao gồm vô số dữ liệu đầu vào thay đổi liên tục, và cố gắng lôi kéo tất cả chúng vào một tài liệu mô tả một sản phẩm mới. Quá trình này là một hệ thống kiến thức không mang tính khuôn mẫu và áp ủi nhiều vấn đề.

Trong bước đầu tiên của việc kiểm thử là xem xét bản đặc tả là không được phép bỏ qua và tìm kiếm các lỗi cụ thể. Bước đầu tiên là đứng sau và xem xét nó ở mức cao. Kiểm tra bản đặc tả với những vấn đề lớn, giám sát và bỏ quên một số thứ. Bạn cũng có thể xem xét và nghiên cứu kỹ hơn về kiểm thử, nhưng cuối cùng thì sự nghiên cứu này cũng chỉ là một công cụ để hiểu kỹ hơn về cái mà phần mềm nên làm. Nếu bạn hiểu rõ hơn là tại sao và như thế nào về bản đặc tả, bạn sẽ thấy tốt hơn về việc kiểm thử một cách chi tiết.

a) Giả sử bạn là một khách hàng

Điều dễ chịu nhất dành cho một tester là khi anh ta nhận một bản đặc tả để duyệt và anh ta sẽ được đóng vai trò như một khách hàng. Anh ta sẽ phải làm một số nghiên cứu về những khách hàng tiềm năng. Hãy nói chuyện với những người quảng bá sản phẩm của bạn và những người bán hàng để nhận được từ họ những gợi ý về nhu cầu của khách hàng. Nếu sản phẩm là một phần mềm nội bộ, hãy xác định xem ai sẽ sử dụng nó và nói chuyện với họ.

Điều quan trọng là bạn phải hiểu được những mong muốn của khách hàng. Hãy nhớ rằng chất lượng (*quality*) có nghĩa là “đáp ứng được những nhu cầu của khách hàng”. Là một tester, bạn phải hiểu những nhu cầu đó để kiểm tra xem phần mềm đã có những gì. Để thực hiện hiệu quả không có nghĩa rằng bạn phải bạn phải là một chuyên gia trong lĩnh vực vật lý hạt nhân nếu bạn đang kiểm thử cho một *power plant*. Hoặc bạn phải là một phi công chuyên nghiệp nếu bạn đang kiểm tra một phần mềm mô phỏng các chuyến bay. Nhưng nếu bạn có những hiểu biết nhất định về lĩnh vực của phần mềm bạn đang kiểm thử thì cũng sẽ giúp bạn rất nhiều.

Với tất cả những cái khác thì không có gì cả. Nếu bạn phải xem xét một phần của bản đặc tả và không hiểu gì về nó, thì đừng cho rằng nó là một bản đặc tả tốt và tiếp tục làm việc. Cuối cùng, bạn vẫn phải sử dụng bản đặc tả này để thiết kế các trường hợp kiểm thử của bạn. Vì vậy mà rút cuộc thì bạn vẫn phải hiểu nó. Không có thời điểm nào tốt hơn để tìm hiểu về nó như bây giờ. Nếu bạn tìm thấy lỗi theo cách này, tất cả sẽ ổn.

Lời khuyên: Đừng quên chú ý tới khả năng bảo mật của phần mềm khi bạn đóng vai trò là một khách hàng. Khách hàng sẽ tin rằng phần mềm là đảm bảo, nhưng bạn có thể biết rằng lập trình viên sẽ phải bàn về vấn đề bảo mật một cách hợp lý.

b) Nghiên cứu về những chuẩn đã có sẵn và đường lối chỉ đạo

Hãy trở về với thời kỳ đầu của *Microsoft Windows* và *Apple Macintosh*, những sản phẩm phần mềm đầu tiên có một giao diện sử dụng rất khác biệt. Chúng có các màu sắc khác nhau, cấu trúc menu khác nhau, các cách không giới hạn để mở một file và vô số những lệnh khó hiểu để thực hiện các nhiệm vụ giống nhau. Để chuyển từ việc sử dụng phần mềm này sang phần mềm khác đòi hỏi bạn phải được đào tạo lại hoàn toàn.

Nhưng thật may mắn, người ta đã cố gắng để chuẩn hóa phần cứng và phần mềm. Họ đã tổ chức một cuộc nghiên cứu trên phạm vi lớn về cách mọi người sử dụng máy tính. Kết quả là ngày nay chúng ta có những sản phẩm có cách dùng rất gần gũi với nhau và thân thiện với người dùng. Bạn có thể thảo luận về những *standard* đã được thông qua và *guideline* không phải là hoàn hảo, có những cách rất tốt để hoàn thành nhiệm vụ một cách xuất sắc, nhưng hiệu quả được cải tiến rất nhiều vì sự tương đồng này. Bây giờ bạn nên tự xem xét xem những chuẩn (*standard*) và đường nối (*guideline*) nào có thể áp dụng vào sản phẩm của bạn.

Chú ý: Sự khác nhau giữa các chuẩn (*standards*) và đường lối (*guidelines*) là về mức độ quan trọng. Một *standard* có tính bền vững hơn một *guideline*. Nên tuân theo các *standard* một cách chặt chẽ nếu như đội của bạn được chỉ thị rằng: điều

quan trọng là phải tuân theo chúng một cách đầy đủ. *Guideline* thì không bắt buộc nhưng cũng nên tuân theo. Cũng không ít trường hợp các đội quyết định dùng các *standard* lâu dài như một *guideline*. Mọi người cho rằng đó là một kế hoạch.

Có một vài ví dụ về các *standard* và các *guideline* được xem xét. Đây không phải là danh sách cuối cùng. Bạn nên nghiên cứu những cái có thể bạn sử dụng cho phần mềm của bạn:

- Các thuật ngữ và các quy ước của các tổ chức (*Corporate Terminology and Conventions*): Nếu phần mềm này là để đáp ứng nhu cầu cho một công ty cụ thể, nó sẽ trở thành các quy ước và thuật ngữ chung được các nhân viên trong công ty đó sử dụng.
- Nhu cầu của ngành công nghiệp (*Industry Requirements*): Các ngành tài chính, công nghiệp, dược và y học có những tiêu chuẩn rất nghiêm ngặt mà buộc các phần mềm của họ phải tuân theo.
- Chuẩn về chính quyền (*Government Standards*): Chính phủ, đặc biệt là lực lượng vũ trang, có những tiêu chuẩn rất nghiêm ngặt (*strict standards*)
- Giao diện đồ họa người dùng (*Graphical User Interface - GUI*): Nếu như phần mềm của bạn chạy dưới hệ điều hành *Microsoft Windows* hoặc *Apple Macintosh* thì sẽ có những *standard* và *guideline* được công bố để đạt được những yêu cầu về sự nhìn nhận và cảm nghĩ (*look and feel*) của người dùng về phần mềm.
- Tiêu chuẩn về sự bảo mật (*Security Standards*): Phần mềm của bạn, giao diện và những giao thức của nó có thể cần đảm bảo đạt đến những chuẩn về bảo mật hoặc đạt các mức độ (*levels*). Cũng có thể nó cần được chứng nhận một cách độc lập những thứ mà nó có thể làm được, và những tiêu chuẩn mà nó đạt được.

Là một *tester*, công việc của bạn không phải là vạch rõ những *guideline* nào và *standard* nào nên áp dụng cho phần mềm của bạn. Việc đó là của những người quản lý dự án hoặc thậm chí là của những người đang viết bản đặc tả. Tuy nhiên,

bạn nên tự điều tra để phục vụ việc kiểm thử của mình sao cho phần mềm đảm bảo đạt chuẩn và mọi người đều nhận thấy điều đó. Bạn cũng phải có những kiến thức về *standard* này và kiểm tra lại chúng khi bạn xác minh và công nhận phần mềm. Xem chúng như một phần của bản đặc tả.

c) Xem xét và kiểm tra những phần mềm tương tự

Một trong những cách thức tốt nhất để tìm hiểu cái mà phần mềm của bạn cần đạt đến là nghiên cứu những phần mềm tương tự. Đó cũng có thể là một phần mềm của công ty đang cạnh tranh với công ty bạn hoặc là một số sản phẩm tương tự của chính công ty bạn. Một số điều cần lưu ý khi xem xét các sản phẩm đang cạnh tranh với phần mềm của bạn:

- **Tỷ lệ (scale):** Các *feature* sẽ nhiều hơn hay ít hơn? *Code* sẽ ít đi hay nhiều hơn? Những sự thay đổi này sẽ là nội dung trong công việc kiểm thử của bạn?
- **Sự phức tạp (complexity):** Phần mềm của bạn sẽ phức tạp hơn hay đơn giản hơn? Điều này cũng sẽ ảnh hưởng tới công việc kiểm thử của bạn chứ?
- **Khả năng kiểm thử (testability):** Bạn sẽ có tài nguyên, thời gian, và chuyên môn để kiểm thử phần mềm như vậy không?
- **Chất lượng / tính tin cậy (quality / reliability):** Đây có phải là phần mềm tiêu biểu cho chất lượng các phần mềm của bạn. Liệu phần mềm của bạn có đáng tin cậy hơn trước không?
- **Bảo mật (security):** Độ bảo mật của phần mềm cạnh tranh với công ty bạn như thế nào, cả những lời quảng cáo và sự thật, so sánh để thấy được mức mà bạn đề nghị?

Không gì có thể thay thế được kinh nghiệm truyền tay (*hands-on experience*), vì vậy khi làm bất sản phẩm nào bạn nên xem xét những phần mềm tương tự, sử dụng nó, đánh giá những phần hoàn hảo nhất của nó, và đặt nó vào trong hoàn cảnh của chính nó. Bạn sẽ thu thập được rất nhiều kinh nghiệm. Nó sẽ giúp ích rất nhiều cho bạn khi bạn kiểm thử một cách chi tiết bản đặc tả của mình.

Lời khuyên: Đừng quên tìm kiếm những tài liệu trên mạng, xem xét các tài liệu về phần mềm và các bài báo về đối thủ của bạn. Điều này có thể đặc biệt có ích cho vấn đề bảo mật. Bạn không muốn thấy những lỗ hổng bảo mật như ở những phần mềm bạn đã từng sử dụng.

2.3.3. Kỹ thuật kiểm thử đặc tả mức thấp

Sau khi bạn hoàn thành thao tác kiểm tra mức cao của bản đặc tả sản phẩm, bạn sẽ hiểu kỹ hơn về cái mà sản phẩm của bạn có được và những vấn đề gây ảnh hưởng cuối cùng tới thiết kế của bạn. Ngụy trang với các thông tin này, bạn có thể tiếp tục kiểm thử ở mức thấp hơn. Phần còn lại của bài này giải thích bản đặc tả làm gì.

a) Danh mục những thuộc tính của bản đặc tả

Thật là tốt, bản đặc tả được cân nhắc kỹ, với "all its t's crossed and its i's dotted", có 8 thuộc tính quan trọng:

- Hoàn thiện (*complete*): Có chỗ nào còn lỗi hoặc bị bỏ quên không? Phần mềm đã triệt để chưa? Có phải nó bao gồm mọi thứ cần thiết để nó có thể đứng độc lập không?
- Chính xác (*accurate*): Giải pháp được đề xuất có đúng không? Nó có xác định được mục đích một cách hợp lý không? Có còn tồn tại lỗi nào không?
- Rõ ràng, chính xác, không mập mờ và trong sáng (*Precise, Unambiguous, and Clear*): Mô tả có chính xác, rõ ràng không? Những lời giải thích có làm sáng tỏ vấn đề không? Có dễ đọc, dễ hiểu không?
- Nhất quán (*consistent*): Lời mô tả cho những *feature* đã được viết không được mâu thuẫn với chính nó hoặc những mục khác trong bản đặc tả.
- Mối quan hệ (*relevant*): Có những mô tả cần thiết để chỉ định ra các đặc trưng không? Nó có phải là những thông tin mở rộng nên bỏ bớt

đi không? Có phải *feature* là những cái mô tả nhu cầu cơ bản của khách hàng không?

- **Khả thi (*Feasible*):** có phải *feature* có thể được thực thi với nguồn nhân lực đã có sẵn, các công cụ, và tài nguyên trong lịch trình và nguồn ngân sách đã được chỉ định
- **Mã nguồn mở (*code - free*):** Có phải bản đặc tả chỉ định sản phẩm và những thứ không nằm phía dưới bản thiết kế, kiến trúc và mã của phần mềm. Bản đặc tả chỉ rõ những phần dùng mã nguồn mở, nguồn gốc của những code này
- **Khả năng kiểm thử (*testable*):** Các *feature* có cần được kiểm thử không? Thông tin liệu có đủ để cung cấp cho *tester* để họ thực hiện quá trình xác minh những điều khiển của phần mềm không?

Khi bạn đang kiểm tra một bản đặc tả sản phẩm, đọc những đoạn văn bản, kiểm tra những hình vẽ của nó, xem xét cận thận từng đặc điểm một. Bạn hãy tự hỏi chính bản thân mình về từ ngữ và những bức hình mà bạn đang kiểm tra. Nếu chúng không đáp ứng được yêu cầu thì bạn đã tìm thấy lỗi

b) Danh mục những thuật ngữ đặc tả

Bổ sung cho danh sách những thuộc tính là một loạt các từ ngữ về các vấn đề cần được quan tâm trong khi xem xét bản đặc tả. Sự xuất hiện của những từ này báo hiệu rằng *feature* không được trọng視. Hãy tìm kiếm những từ này trong bản đặc tả và cẩn thận xem xét chúng được sử dụng trong hoàn cảnh nào. Bản đặc tả có thể lọc ra hoặc trau chuốt cho chúng, hoặc có thể loại những từ ngữ không rõ ràng, mơ hồ ra khỏi bản đặc tả.

- Luôn luôn, mọi thứ, tất cả, không một ai, không bao giờ (*Always, every, all, none, never*): Nếu bạn nhìn thấy những từ kiểu như vậy là bản đặc tả muốn nói đến những khẳng định chắc chắn. Là một tester, bạn hãy đặt ra những tình huống không đúng với những khẳng định đó.

- Tất nhiên, bởi vậy, chắc hẳn rồi, hiển nhiên, rõ ràng (*Certainly, Therefore, Clearly, Obviously, Evidently*): Những từ này có khuynh hướng làm cho bạn tin tưởng và chấp nhận những điều mà họ đưa ra. Đừng rơi vào những tình trạng đó.
- Vân vân, và cứ tiếp tục ở phía trước, và cứ tiếp tục như vậy, ví dụ (*Etc., And So Forth, And So On, Such As*): Đây là danh sách những từ mà chúng ta không có khả năng kiểm tra được. Những từ ngữ này cần được khẳng định và giải thích để tránh sự lôn xộn, để kiểm soát xem chuỗi tiếp theo được sinh ra như thế nào và cài gì xuất hiện tiếp theo.
- Tốt, nhanh, rẻ, hiệu quả, nhỏ gọn, ổn định (*Good, Fast, Cheap, Efficient, Small, Stable*): Có một số thuật ngữ không đủ điều kiện để được sử dụng. Không thể kiểm tra được độ chính xác của chúng. Nếu thấy chúng xuất hiện trong một bản đặc tả bạn phải chỉ định thêm những lời giải thích chính xác xem chúng là gì.
- Danh hiệu, quy trình, loại bỏ, bỏ qua, loại trừ (*Handled, Processed, Rejected, Skipped, Eliminated*): Những thuật ngữ này ẩn dấu một số lượng lớn các chức năng mà cần phải được chỉ rõ.
- Nếu ... thì ... nhưng không có trường hợp còn lại (*If...Then...but missing Else*): hãy tìm những lời thông báo dạng mệnh đề “nếu... thì...” nhưng không có vé còn lại. Hãy tự đặt câu hỏi: cái gì sẽ xảy đến nếu như trường hợp “nếu...” không xảy ra.

Tổng kết:

Bây giờ bạn phải hiểu sản phẩm phần mềm tạo ra như thế nào, cái gì bên trong chúng và quy trình được sử dụng để liên kết chúng với nhau. Cũng như bạn có thể nhìn thấy, không có một hướng tiếp cận xác định. Bốn mô hình mô tả ở đây chỉ là ví dụ. Có nhiều mô hình khác nữa và biến thể của chúng. Mỗi công ty, mỗi dự án và mỗi đội sẽ chọn mô hình phù hợp với họ. Đôi khi, họ lựa chọn đúng, nhưng thỉnh thoảng họ cũng sẽ lựa chọn sai. Công việc của bạn là kiểm tra phần mềm để nó làm việc tốt nhất trong mô hình phát triển của nó, áp dụng các kỹ năng kiểm thử trong phần còn lại của cuốn sách để tạo ra những phần mềm tốt nhất có thể được.

BÀI 3. QUY TRÌNH KIỂM THỬ PHẦN MỀM

3.1. Quy trình kiểm thử phần mềm tổng quát

Quy trình kiểm thử phần mềm là gì?

- Chế độ kiểm thử được định nghĩa bởi tổ chức phát triển phần mềm là gì.
- Cần có chiến lược kiểm thử và nó sẽ giải tại sao tổ chức phần mềm kiểm thử các thành phần mà mình tạo ra.
- Cần nhận dạng cái gì là quan trọng đối với tổ chức (chi phí, chất lượng, thời gian, phạm vi,...) và cách nào, bởi ai và khi nào việc kiểm thử sẽ được thực hiện.

f Tất cả các thông tin trên sẽ được lập thành tài liệu cho hoạt động kiểm thử và ta có thể gọi qui trình tạo lập tài liệu này là qui trình kiểm thử phần mềm (Test Process).

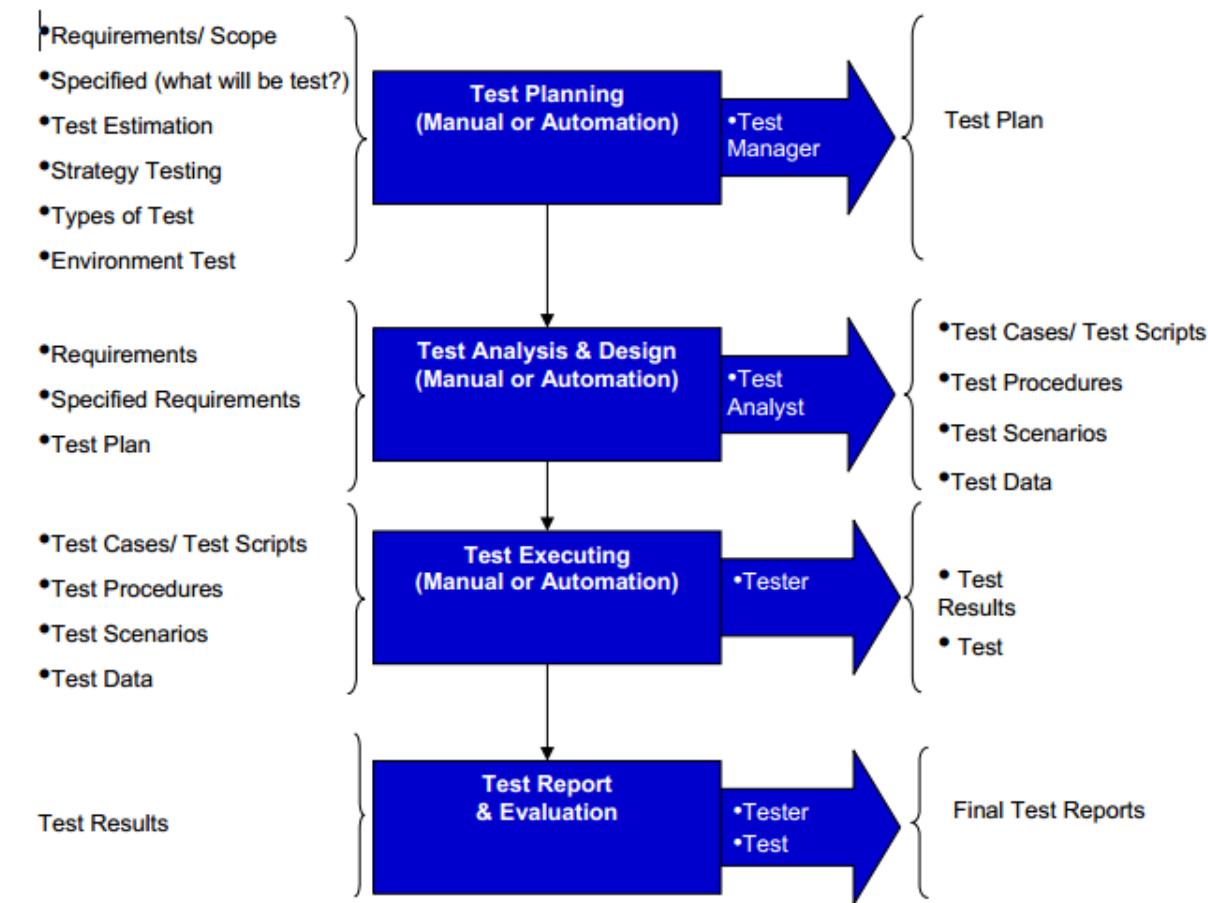
Tạo sao cần phải thực hiện qui trình kiểm thử phần mềm ?

- Cần làm rõ vai trò và trách nhiệm của việc kiểm thử phần mềm.
- Cần làm rõ các công đoạn, các bước kiểm thử.
- Cần phải hiểu và phân biệt các tính chất kiểm thử (tạo sao phải kiểm thử), các bước kiểm thử (khi nào kiểm thử), và các kỹ thuật kiểm thử (kiểm thử bằng cách nào).

Các tính chất của qui trình kiểm thử tốt :

- Cần có 1 mức độ kiểm thử cho mỗi công đoạn phát triển phần mềm.
- Các mục tiêu kiểm thử sẽ bị thay đổi, mỗi mức kiểm thử nên có các mục tiêu đặc thù của mình.
- Việc phân tích và thiết kế testcase cho 1 mức độ kiểm thử nên bắt đầu sớm nhất như có thể có.
- Các tester nên xem xét các tài liệu sớm như có thể có, ngay sau khi các tài liệu này được tạo ra trong chu kỳ phát triển phần mềm.
- Số lượng và cường độ của các mức kiểm thử được điều khiển theo các yêu cầu đặc thù của project phần mềm đó

Hình dưới đây là quy trình kiểm thử phần mềm tổng quát.



Hình 3.1. Quy trình kiểm thử phần mềm

3.2. Lập kế hoạch kiểm tra (Test Plan)

a) Đầu vào

- Kế hoạch dự án
- Tài liệu SRS, SRD, HLD
- Các tài liệu mô tả nghiệp vụ

b) Mô tả

Tìm hiểu các yêu cầu là bước đầu tiên của quy trình kiểm thử phần mềm. Trong giai đoạn này, kiểm thử viên phải có trách nhiệm tìm hiểu thông tin về sản phẩm, về các tính năng của sản phẩm sẽ kiểm tra, các yêu cầu và tiêu chí chấp nhận sản phẩm của khách hàng. Từ đó sẽ xác định kiểm tra những cái gì và phạm vi kiểm tra, cũng như tiến độ, thời gian và nguồn nhân lực tham gia vào quy trình kiểm thử. Trong quá trình tìm hiểu, ngoài việc tiếp nhận và khảo sát yêu cầu của khách hàng còn có thêm công việc tư vấn cho khách hàng về hệ thống đang cần xây dựng.

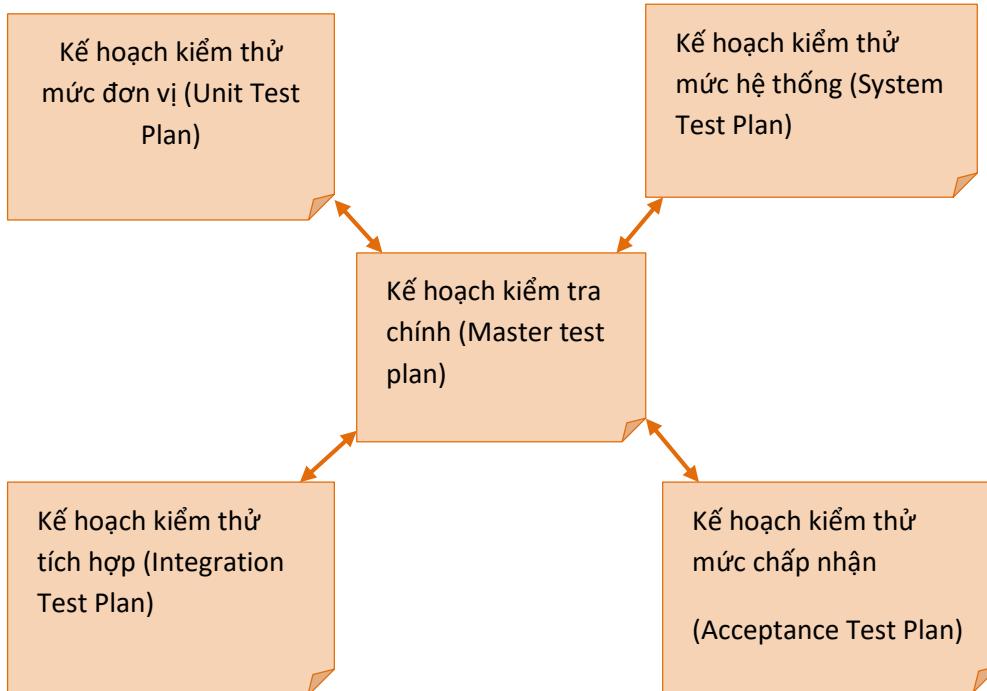
Sau khi tiếp nhận yêu cầu kiểm tra của dự án phần mềm, nhóm đảm bảo chất lượng phần mềm PQA (Product Quality Assurance) tiến hành lập kế hoạch kiểm tra. Việc lập kế hoạch kiểm tra hệ thống phần mềm nhằm mục đích xác định các yêu cầu, mục đích, phương thức kiểm tra cũng như xác định nguồn lực và thời gian, công cụ kiểm tra cần thiết cho toàn bộ quá trình kiểm tra, trước khi tiến hành kiểm tra hệ thống phần mềm. Giai đoạn này tiến hành song song với thời gian phát triển dự án phần mềm. Khi đã có kế hoạch dự án, tài liệu SRS và HLD thì có thể tiến hành xây dựng kế hoạch kiểm tra. Đồng thời, nhóm PQA thống nhất phương thức giao tiếp với nhóm phát triển.

- ✓ Nhóm dự án cung cấp các tài liệu liên quan của phần mềm cho nhóm PQA.
- ✓ Nhóm PQA nghiên cứu các yêu cầu, thời gian và nội dung các yêu cầu nhận được để quyết định các bước tiếp theo.
- ✓ Nhóm PQA chuẩn bị nhân lực và đào tạo nghiệp vụ cần thiết phục vụ cho nhu cầu dự án.
- ✓ Cán bộ PQA lập kế hoạch kiểm tra và trình Giám đốc dự án phê duyệt. Bản kế hoạch kiểm tra đã phê duyệt sẽ được phổ biến đến từng thành viên tham gia quá trình kiểm tra và là cơ sở cho việc đánh giá mức độ hoàn thành công việc và quản lý nhóm làm việc, tiến độ công việc.

Kết quả của bước lập kế hoạch là bản tài liệu kế hoạch KTPM, bao gồm nhiều chi tiết từ các loại kiểm tra, chiến lược kiểm tra, cho đến thời gian và phân định lực lượng kiểm tra viên. Giai đoạn này được tiến hành song song với thời gian phát triển dự án phần mềm. Khi đã có kế hoạch dự án, tài liệu SRS thì có thể tiến hành xây dựng kế hoạch kiểm tra.

Bản kế hoạch kiểm tra đầu tiên được phát triển rất sớm trong chu trình phát triển phần mềm (PTPM), ngay từ khi các yêu cầu đã tương đối đầy đủ, các chức năng và luồng dữ liệu chính đã được mô tả. Bản kế hoạch này có thể được coi là bản kế hoạch chính (master test plan), trong đó tất cả các kế hoạch chi tiết cho các mức kiểm tra và loại kiểm tra khác nhau đều được đề cập.

Lưu ý, tùy theo đặc trưng và độ phức tạp của mỗi dự án, các kế hoạch kiểm tra chi tiết có thể được gom chung vào bản kế hoạch chính hoặc được phát triển riêng.



Hình 2.2: Kế hoạch kiểm thử phần mềm

Sau khi bản kế hoạch chính được phát triển, các bản kế hoạch chi tiết lần lượt được thiết kế theo trình tự thời gian phát triển của dự án. (Hình 3.1 minh họa thời điểm phù hợp để thiết lập các kế hoạch kiểm tra, gắn liền với quá trình phát triển của dự án). Quá trình phát triển các kế hoạch kiểm tra không dừng lại tại một thời điểm, mà liên tục được cập nhật chỉnh sửa cho phù hợp đến tận cuối dự án.

c) Các bước lập kế hoạch

- ✓ *Xác định yêu cầu kiểm tra:* chỉ định bộ phận, thành phần của phần mềm sẽ được kiểm tra, phạm vi hoặc giới hạn của việc kiểm tra. Yêu cầu kiểm tra cũng được dùng để xác định nhu cầu nhân lực.
- ✓ *Khảo sát rủi ro:* Các rủi ro có khả năng xảy ra làm chậm hoặc cản trở quá trình cũng như chất lượng kiểm thử. Ví dụ : Kỹ năng và kinh nghiệm của kiểm thử viên quá yếu, không hiểu rõ yêu cầu.
- ✓ *Xác định chiến lược kiểm thử:* Chỉ định phương pháp tiếp cận để thực hiện việc kiểm thử trên phần mềm. Chỉ định các kỹ thuật và công cụ hỗ trợ kiểm

thử, xác định các phương pháp dùng để đánh giá chất lượng kiểm tra cũng như điều kiện để xác định thời gian kiểm tra.

- ✓ *Xác định nhân lực, vật lực:* Kỹ năng, kinh nghiệm của kiểm tra viên; phần cứng, phần mềm, công cụ, thiết bị giả lập... cần thiết cho việc kiểm tra.
- ✓ *Lập kế hoạch chi tiết:* Ước lượng thời gian, khối lượng công việc, xác định chi tiết các phần công việc, người thực hiện, thời gian tất cả các điểm mốc của quá trình kiểm tra.
- ✓ *Tổng hợp và tạo các bản kế hoạch kiểm tra:* Kế hoạch chung và kế hoạch chi tiết.
- ✓ *Xem xét các kế hoạch kiểm tra:* Phải có sự tham gia của tất cả những người có liên quan, kể cả trưởng dự án và có thể cả khách hàng. Việc xem xét nhằm bảo đảm các kế hoạch là khả thi, cũng như để phát hiện (và sửa chữa sau đó) các sai sót trong các bản kế hoạch.

d) Kết quả

- Kế hoạch kiểm tra được phê duyệt.

e) Người thực hiện

- Cán bộ PQA
- Trưởng nhóm PQA
- Giám đốc dự án

3.3. Chuẩn bị môi trường kiểm tra

a) Đầu vào

- Kế hoạch kiểm tra
- Tài liệu SRS, SRD, HLD
- Tài liệu nghiệp vụ.

b) Mô tả

Chuẩn bị dữ liệu và môi trường kiểm tra là tiền đề để đảm bảo cho việc kiểm tra có chất lượng và năng suất cao. Tiến hành xây dựng dữ liệu, thiết lập môi trường, kiểm tra thiết bị cần thiết, công cụ kiểm tra đã đáp ứng kế hoạch kiểm tra đề ra. Mục đích của giai đoạn này là chuẩn bị môi trường kiểm tra cho hệ thống phần mềm.

Cán bộ kiểm tra có trách nhiệm phối hợp cùng với bên hỗ trợ kỹ thuật để chuẩn bị đầy đủ môi trường thực tế hay giả lập cho chương trình, dự án phần mềm trước khi tiến hành kiểm tra.

c) Các công việc cần thực hiện

- Xây dựng các công cụ tạo dữ liệu, kiểm tra cơ sở dữ liệu.
- Tạo dữ liệu kiểm tra hệ thống cho chương trình/công cụ kiểm tra.
- Xây dựng môi trường phần cứng
- Xây dựng môi trường phần mềm

d) Kết quả

- Môi trường kiểm tra sẵn sàng

e) Người thực hiện

- Cán bộ PQA.
- Trưởng nhóm PQA

3.4. Thiết kế kiểm tra (Test Design)

a) Đầu vào

- Tài liệu SRS, SRD, HLD.
- Prototype (nếu có)

b) Mô tả

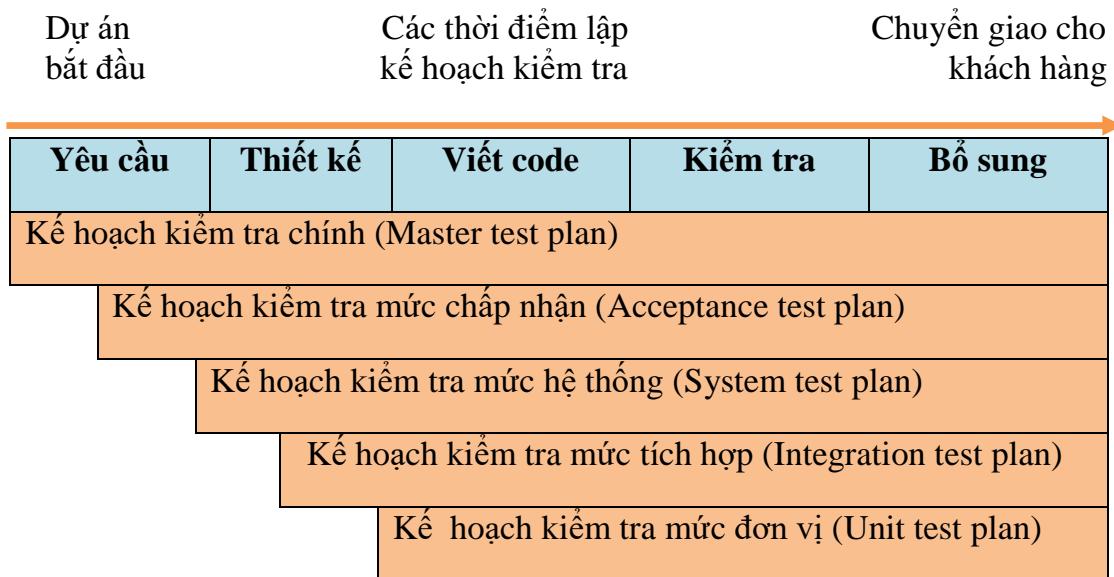
Giai đoạn thiết kế kiểm tra nhằm đưa ra các tình huống kiểm tra. Đây là tài liệu cụ thể hoá các bước kiểm tra sẽ phải tiến hành trong quá trình thực hiện tình huống kiểm tra. Các tình huống kiểm tra cần chỉ rõ phần mềm có thể đáp ứng được các yêu cầu nào, có những khía cạnh nào. Các khía cạnh cần tập trung là:

Các yêu cầu chức năng bắt buộc, các yêu cầu chức năng tùy chọn hay phi chức năng

- Các tính năng bảo mật mà hệ thống cần đáp ứng
- Tính ổn định của chương trình
- Khả năng phục hồi của hệ thống
- Các giai đoạn tiến hành kiểm tra

Việc lập các tình huống kiểm tra sẽ do cán bộ kiểm tra đảm nhiệm, quá trình lập tình huống sẽ được Trưởng nhóm kiểm tra thường xuyên xem xét và sửa đổi, cập nhật nếu cần.

Hình dưới đây cho thấy việc thiết kế test không chỉ làm một lần, nó sẽ được sửa chữa, cập nhật, thêm hoặc bớt trong suốt chu trình phát triển phần mềm, vào bất cứ lúc nào có sự thay đổi yêu cầu hoặc sau khi phân tích thấy cần sửa chữa, bổ sung.



Hình 2.3: Thiết kế kiểm tra

c) Các bước thiết kế kiểm tra

- ✓ *Xác định và mô tả Test Case:* xác định các điều kiện cần thiết lập trước và trong lúc kiểm tra. Mô tả đối tượng hoặc dữ liệu đầu vào, mô tả các kết quả mong chờ sau khi kiểm tra.
- ✓ *Mô tả các bước chi tiết để kiểm tra:* các bước này mô tả chi tiết để hoàn thành một Test Case khi thực hiện kiểm tra. Các Test Case như đã nói ở trên thường chỉ mô tả đầu vào, đầu ra, còn cách thức tiến hành như thế nào thì không được định nghĩa. Thao tác này nhằm chi tiết hóa các bước của một Test Case, cũng như chỉ định các loại dữ liệu nào cần có để thực thi các Test Case, chúng bao gồm các loại dữ liệu trực tiếp, gián tiếp, trung gian, hệ thống...
- ✓ *Xem xét và khảo sát độ bao phủ của việc kiểm tra:* mô tả các chỉ số và cách thức xác định việc kiểm tra đã hoàn thành hay chưa. Bao nhiêu phần trăm phần mềm đã được kiểm tra. Để xác định điều này có hai phương pháp: căn cứ trên yêu cầu của phần mềm hoặc căn cứ trên số lượng code đã viết

- ✓ *Xem xét Test Case và các bước kiểm tra:* Việc xem xét cần có sự tham gia của tất cả những người có liên quan, kể cả trưởng dự án nhằm bảo đảm các Test Case và dữ liệu yêu cầu là đủ và phản ánh đúng các yêu cầu cần kiểm tra, độ bao phủ đạt yêu cầu, cũng như để phát hiện (và sửa chữa) các sai sót.

1. **Test Case**

- *Mục đích*
- Một tình huống kiểm tra (test case - TC) được thiết kế để kiểm tra một đối tượng có thỏa mãn yêu cầu đặt ra hay không.
- *Một Test case thường có 4 phần cơ bản:*
 - Điều kiện (Condition): Đặc tả các điều kiện cần có để tiến hành kiểm tra.
 - Dữ liệu đầu vào (Input Data): Đặc tả đối tượng hay dữ liệu cần thiết, được sử dụng làm đầu vào để thực hiện việc kiểm tra.
 - Kết quả mong chờ (Expected Result): Kết quả mong đợi trả về từ đối tượng kiểm tra.
 - Kết quả thực tế (Actual Result): kết quả thực tế trả về từ đối tượng kiểm tra
- *Kỹ thuật thiết kế Test Case:*
 - Kỹ thuật dựa trên đặc tả - hộp đen (Black box testing): Tìm kiếm lỗi theo cách hệ thống thực hiện.
 - Kỹ thuật dựa trên cấu trúc – hộp trắng (White box testing): Tìm kiếm lỗi theo cách hệ thống được xây dựng
 - Kỹ thuật dựa trên kinh nghiệm: Tạo dữ liệu kiểm thử dựa vào sự hiểu biết về hệ thống, kinh nghiệm trong quá khứ, phương pháp phỏng đoán về lỗi.

2. **Test Script**

- *Định nghĩa*

Một kịch bản (Test Script) 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.

- *Mục đích*

Bước này thường không bắt buộc trong các loại và mức kiểm tra, chỉ yêu cầu trong những trường hợp đặc thù cần thiết kế, tạo ra các Test Script có khả năng chạy trên máy tính giúp tự động hóa việc thực thi các bước kiểm tra đã định nghĩa ở bước thiết kế test

➤ *Các bước phát triển Test Script*

- ✓ Tạo Test Script: Thủ công hoặc dùng công cụ hỗ trợ để phát sinh script một cách tự động (tuy nhiên trong hầu hết mọi trường hợp, ta vẫn phải chỉnh sửa ít hoặc nhiều trên các script được sinh tự động). Thông thường, mỗi bước kiểm tra được thiết kế trong phần thiết kế test, đòi hỏi ít nhất một Test Script. Các Test Script có khả năng tái sử dụng càng nhiều càng tốt để tối ưu hóa công việc.
- ✓ Kiểm tra Test script: Xem có “chạy” tốt không nhằm bảo đảm các Test Script hoạt động đúng yêu cầu, thể hiện đúng ý đồ của các bước kiểm tra
- ✓ Thành lập các bộ dữ liệu ngoài dành cho các Test Script: Bộ dữ liệu này sẽ được các Test Script sử dụng khi thực hiện kiểm tra tự động. Gọi là “ngoài” vì chúng được lưu độc lập với các Test Script, tránh trường hợp vì dễ dãi, một số kiểm tra viên “tích hợp” luôn phần dữ liệu vào bên trong code của các script (thuật ngữ chuyên môn gọi là “hard-code”). Việc tách riêng dữ liệu cho phép dễ dàng thay đổi dữ liệu khi kiểm tra, cũng như giúp việc chỉnh sửa hoặc tái sử dụng các script sau này.
- ✓ Xem xét và khảo sát độ bao phủ của việc kiểm tra: Bảo đảm các Test Script được tạo ra bao phủ toàn bộ các bước kiểm tra theo yêu cầu.

3. *Test Data*

➤ *Định nghĩa*

Test Data là bộ dữ liệu được xây dựng để chạy thử các test case. Dữ liệu trong Test Data gồm có hai loại là dữ liệu thường (normal data) và dữ liệu bắt buộc (Initial Data). Xây dựng Test Data là một khâu rất quan trọng trong tiến trình test, vì kết quả test phụ thuộc rất lớn vào dữ liệu trong Test Data.

Initial Data là các trường dữ liệu dùng để khởi tạo chương trình, bắt buộc cần phải có để hệ thống có thể làm việc được. Initial Data là một bộ phận của Test Data.

➤ *Các bước phát triển Test Data*

- ✓ Kiểm thử với tất cả các dữ liệu vào là cần thiết, nhưng chúng ta không thể thực hiện kiểm thử “vết cạn” được.
- ✓ Chọn tập các dữ liệu thử đại diện từ miền dữ liệu vào dựa trên các tiêu chuẩn chọn dữ liệu thử. Yêu cầu đối với bộ dữ liệu kiểm thử là phải phủ kín được mọi trường hợp cần đánh giá

➤ *Người thực hiện*

- ✓ Test Leader và Developer xây dựng, sau khi có tài liệu phân tích thiết kế mức chi tiết.
- ✓ Dữ liệu khởi tạo (Initial Data) là phần bắt buộc cần phải có để hệ thống có thể hoạt động được, thường do lập trình viên tạo lập sau khi hoàn chỉnh từng bộ phận của hệ thống.
- ✓ Test Leader chịu trách nhiệm xây dựng bộ dữ liệu Test Data thông thường.
- ✓ Tester sẽ dùng các dữ liệu này để thực hiện Test Case

➤ *Đầu vào*

Để xây dựng Test Data cần sử dụng các tài liệu mô tả dữ liệu trong phần Data_Model (Analysis_Design). Trong đó, quan trọng nhất là 2 tài liệu:

- Physical_ER_Diagram: Lược đồ quan hệ thực thể; lược đồ này mô tả quan hệ giữa các bảng trong dự án.
- Physical_Table_Definition: Mô tả định nghĩa các bảng, bao gồm Danh sách toàn bộ tên bảng sử dụng trong dự án; Mô tả chi tiết của từng bảng (tên cột, kiểu giá trị của từng cột, kích thước dữ liệu, ...).

d) Kết quả

- Tình huống kiểm tra được phê duyệt.

e) Người thực hiện

- Cán bộ kiểm tra
- Trưởng nhóm PQA
- Quản trị dự án

3.5. Thực hiện kiểm tra (Test Execute)

a) Đầu vào

- Kế hoạch kiểm tra
- Tình huống kiểm tra
- Môi trường kiểm tra đã sẵn sàng

b) Mô tả

Giai đoạn thực hiện tình huống kiểm tra là giai đoạn quan trọng nhất trong toàn bộ quy trình kiểm tra hệ thống phần mềm, vì nó là quá trình thực hiện toàn bộ kế hoạch, tình huống kiểm tra và các hoạt động khác; nhằm bảo đảm phần mềm được phát hành và chuyển giao đáp ứng các yêu cầu hoặc các tiêu chuẩn được xác lập. Các công việc kiểm tra cần chỉ rõ phần mềm đã đáp ứng hoặc chưa đáp ứng các yêu cầu nào, có những lỗi nào. Các khía cạnh cần tập trung là:

- Khẳng định các module riêng biệt và các giao dịch không có lỗi.
- Hệ thống đáp ứng các yêu cầu về giao diện.
- Hệ thống thực hiện đúng quy trình, thao tác nghiệp vụ.
- Sự ổn định, tốc độ yêu cầu với khối lượng giao dịch/dữ liệu trong phạm vi cho phép.
- Sự ổn định, tốc độ yêu cầu với môi trường tối thiểu.
- Hệ thống đáp ứng các yêu cầu về tính thông nhất (ngôn ngữ, phím nóng, thuật ngữ,...)

Các lỗi được tìm thấy trong quá trình kiểm tra được ghi nhận và theo dõi trạng thái lỗi, tình trạng lỗi ...đảm bảo rằng lỗi không được ở trạng thái Open quá lâu, các lỗi nghiêm trọng phải được ưu tiên sửa trước...Kiểm tra lại những lỗi đã được sửa và các chức năng liên quan đảm bảo lỗi được Đóng triệt để và không ảnh hưởng đến các lỗi khác, các chức năng khác... Việc thực hiện tình huống kiểm tra sẽ do cán bộ kiểm tra đảm nhiệm.Trong quá trình thực hiện tình huống kiểm tra, Trưởng nhóm kiểm tra thường xuyên xem xét (nếu cần).

c) Các bước thực hiện kiểm tra

Quá trình thực hiện kiểm tra thường thông qua các bước sau:

- *Thực hiện các bước kiểm tra:* thủ công hoặc thi hành các Test Script nếu là quy trình kiểm thử tự động. Để thực hiện kiểm tra, tao tác đầu tiên cần làm là xác lập, khởi động môi trường và điều kiện kiểm tra. Việc kiểm tra nhằm bảo đảm tất cả các bộ phận liên quan (như phần cứng, phần mềm, máy chủ, mạng, dữ liệu...) đã được cài đặt sẵn sàng trước khi chính thức bắt đầu kiểm tra
- *Dánh giá quá trình kiểm tra:* giám sát quá trình kiểm tra suôn sẻ đến khi hoàn thành hay bị treo và dừng giữa chừng, cần bổ sung hay sửa chữa gì không để quá trình kiểm tra được tốt hơn. Nếu quá trình kiểm tra diễn ra tron tru, kiểm tra viên hoàn thành chu kỳ kiểm tra và chuyên qua bước “thẩm định kết quả kiểm tra”. Nếu quá trình bị treo hoặc dừng giữa chừng, kiểm tra viên cần phân tích để xác định nguyên nhân lỗi khắc phục và lập lại quá trình kiểm tra.
- *Thẩm định kết quả kiểm tra:* sau khi kết thúc, kết quả kiểm tra cần được xem xét để đảm bảo kết quả nhận được là đáng tin cậy, cũng như biết được những lỗi xảy ra không phải do phần mềm mà do dữ liệu dùng để kiểm tra, môi trường kiểm tra hoặc các bước kiểm tra (hoặc Test Script) gây ra. Nếu thực sự lỗi xảy ra do quá trình kiểm tra cần sửa chữa và kiểm tra lại từ đầu.

Quá trình thực thi test thông thường bao gồm 4 giai đoạn: Unit test, Integration test, System test, Acceptance test ,

- Unit testing – Kiểm thử mức đơn vị
 - Tập trung vào việc xác minh trên đơn vị nhỏ nhất của thiết kế phần mềm, thành phần phần mềm, module (hàm, thủ tục, đoạn code, component...)
 - Dựa vào tài liệu thiết kế chi tiết
 - Unit testing được thực hiện bởi lập trình viên
 - Sử dụng phương pháp test hộp trắng(White box testing).
- Integration test – Kiểm thử mức tích hợp
 - Dựa và tài liệu thiết kế tổng thể của dự án
 - Kiểm tra sự tương tác giữa các module
 - Kiểm tra tính đúng đắn so với bản đặc tả

- Kiểm tra tính hiệu quả
- Được thực hiện bởi kiểm thử viên
- Sử dụng các kỹ thuật kiểm tra như: Tích hợp từ trên xuống (Top-Down), Tích hợp từ dưới lên (Bottom -Up), Tích hợp kẹp (Sandwich)

➤ System test – Kiểm thử mức hệ thống

- Dựa vào tài liệu đặc tả yêu cầu phần mềm
- Kiểm tra tất cả các thành phần của hệ thống đã được tích hợp có thực hiện đúng các chức năng đã xác định hay chưa.
- Sử dụng phương pháp test hộp đen(Black box testing).
- Người thực hiện: Kiểm thử viên, Lập trình viên, Nhóm dự án phát triển phần mềm

➤ Acceptance test – Kiểm thử chấp nhận sản phẩm

- Dựa vào yêu cầu nghiệp vụ của khách hàng và tài liệu hướng dẫn sử dụng phần mềm
- Kiểm tra xem phần mềm đã thỏa mãn tất cả các yêu cầu của khách hàng hay chưa
- Người thực hiện là khách hàng
- Người hỗ trợ: Kiểm thử viên và nhóm phát triển phần mềm

➤ Regression Test - Kiểm tra hồi quy

- Thực hiện kiểm tra khi thay đổi bất kỳ chức năng nào của phần mềm
- Có thể thực hiện tại mọi mức kiểm tra và thường sử dụng lại Test Case và bộ dữ liệu thử đã sử dụng trong các giai đoạn trước đó.
- Người thực hiện là kiểm thử viên và nhóm phát triển dự án phần mềm

d) Kết quả

- Báo cáo kết quả kiểm tra
- Tình huống kiểm tra được cập nhật (nếu có)

e) Người thực hiện

- Cán bộ kiểm tra
- Trưởng nhóm PQA
- Nhóm phát triển

- Quản trị dự án

3.6. Thẩm tra và đánh giá kết quả kiểm tra

a) Đầu vào

- Kế hoạch kiểm tra
- Tình huống kiểm tra
- Báo cáo ghi nhận lỗi và xử lý lỗi.

b) Mô tả

Sau khi kết thúc, kết quả kiểm tra cần được xem xét để bảo đảm kết quả nhận được là đáng tin cậy, cũng như nhận biết được những lỗi xảy ra không phải do PM mà do dữ liệu dùng để kiểm tra, môi trường kiểm tra hoặc các bước kiểm tra (hoặc Test Script) gây ra. Nếu thực sự lỗi xảy ra do quá trình kiểm tra, cần phải sửa chữa và kiểm tra lại từ đầu.

Đánh giá toàn bộ quá trình kiểm tra, bao gồm xem xét và đánh giá kết quả kiểm tra, liệt kê lỗi, chỉ định các yêu cầu thay đổi, và tính toán các số liệu liên quan đến quá trình kiểm tra (chẳng hạn số giờ, thời gian kiểm tra, số lượng lỗi, phân loại lỗi...).

Lưu ý, mục đích của việc đánh giá kết quả kiểm tra ở bước này hoàn toàn khác với bước thẩm định kết quả kiểm tra sau khi hoàn tất một vòng kiểm tra. Đánh giá kết quả kiểm tra ở giai đoạn này mang tính toàn cục và nhầm vào bản thân giá trị của các kết quả kiểm tra. Việc đánh giá quá trình và kết quả kiểm tra được thực hiện song song với bất kỳ lần kiểm tra nào và chỉ chấm dứt khi quá trình kiểm tra đã hoàn tất.

c) Các bước thẩm tra và đánh giá kết quả kiểm tra

Đánh giá quá trình kiểm tra thường thông qua các bước sau:

- ✓ *Phân tích kết quả kiểm tra và đề xuất yêu cầu sửa chữa:* Chỉ định và đánh giá sự khác biệt giữa kết quả mong chờ và kết quả kiểm tra thực tế, tổng hợp và gửi thông tin yêu cầu sửa chữa đến những người có trách nhiệm trong dự án, lưu trữ để kiểm tra sau đó.
- ✓ *Đánh giá độ bao phủ:* Xác định quá trình kiểm tra có đạt được độ bao phủ yêu cầu hay không, tỷ lệ yêu cầu đã được kiểm tra (tính trên các yêu cầu của phần mềm và số lượng code đã viết).

- ✓ *Phân tích lỗi:* Đưa ra số liệu phục vụ cho việc cải tiến các qui trình phát triển, giảm sai sót cho các chu kỳ phát triển và kiểm tra sau đó. Ví dụ, tính toán tỷ lệ phát sinh lỗi, xu hướng gây ra lỗi, những lỗi “ngohan cố” hoặc thường xuyên tái xuất hiện.
- ✓ *Xác định quá trình kiểm tra có đạt yêu cầu hay không:* Phân tích đánh giá để xem các Test Case và chiến lược kiểm tra đã thiết kế có bao phủ hết những điểm cần kiểm tra hay không? Kiểm tra có đạt yêu cầu dự án không? Từ những kết quả này, kiểm tra viên có thể sẽ phải thay đổi chiến lược hoặc cách thức kiểm tra.
- ✓ *Báo cáo tổng hợp:* Tổng hợp kết quả các bước ở trên và phải được gửi cho tất cả những người có liên quan.

d) Đầu ra

- Thông kê lỗi, lỗi phát sinh
- Báo cáo kết quả kiểm tra hệ thống phần mềm

e) Người thực hiện

- Tester: Thu thập kết quả Test.
- Test Manager: Tổng hợp và phân tích kết quả test.
- Test Manager : Thông báo với Project Manager về Test Result.

3.7. Ghi nhận và xử lý lỗi

a) Đầu vào

- Báo cáo kết quả kiểm tra
- Tình huống kiểm tra

b) Mô tả

Theo dõi xử lý lỗi nhằm phân tích, tổng hợp các lỗi mới nhất để gửi tới nhóm phát triển tiến hành sửa đổi cũng như cập nhật các tình huống kiểm tra mới vào tài liệu tình huống kiểm tra khi có các lỗi phát sinh mới, nhằm đảm bảo:

- Lỗi được sửa đúng tiến độ.
- Thực hiện kiểm tra lại các lỗi đã được sửa

Việc xử lý lỗi cũng phải được tập hợp vào công cụ kiểm tra, dữ liệu kiểm tra. Các công việc theo dõi và xử lý sẽ do Trưởng nhóm kiểm tra đảm nhiệm phối hợp cùng với bên phát triển

c) Các bước ghi nhận và xử lý lỗi

➤ *Ghi nhận lỗi*

- Trong quá trình triển khai phần mềm, khi nhà phát triển nhận phản hồi về lỗi phần mềm từ phía khách hàng, tester sẽ có nhiệm vụ ghi nhận lỗi trên với các thông tin rõ ràng: dự án, module, giai đoạn, mức độ nghiêm trọng của lỗi, kiểu lỗi, ngày kiểm tra, ngày tạo...
- Trạng thái lỗi lúc này là ERROR

➤ *Phân tích lỗi*

Lúc này, trưởng nhóm test (Test Leader) phải tiến hành phân tích lỗi nhằm mục đích cải thiện tình trạng lỗi của phần mềm đang xây dựng.

- Xác định lỗi do yếu tố nào mang lại (do hiểu sai requirement của khách hàng, hiểu sai thiết kế, do yếu tố khách quan, do yếu tố chủ quan..).
- Xác định trọng số cho lỗi, lỗi mang trọng số cao sẽ được ưu tiên fix trước
- Xác định lỗi xảy ra thuộc giai đoạn test nào
- Theo dõi, xác định tình trạng lỗi đó (đã fix, đang fail, đang pending hay đang skip)
- Ai sẽ là người chịu trách nhiệm sửa lỗi này
- Trạng thái lỗi lúc này là ASSIGNED

➤ *Sửa lỗi*

- Tùy vào các thuộc tính của lỗi (lỗi rất nghiêm trọng, lỗi nghiêm trọng, lỗi nhẹ) mà nhóm phát triển đưa ra bản kế hoạch và xử lý lỗi phù hợp không ảnh hưởng đến uy tín và chất lượng của Nhà phát triển. Ví dụ với lỗi là lỗi nghiêm trọng buộc phải sửa chữa để có thể sử dụng được các yêu cầu đặt ra. Lập trình viên có trách nhiệm giải quyết lại xem các lỗi đó xảy ra từ đâu và chỉnh sửa lại code sao cho lỗi đó không xảy ra nữa và đảm bảo chương trình vẫn chạy đúng.
- Trạng thái lỗi lúc này là PENDING

➤ *Kiểm tra lại lỗi*

- Việc thực hiện xử lý lỗi phải tiến hành song song với việc lập kế hoạch và triển khai test lại. Nhằm mục đích kiểm tra lại các chức năng có chứa lỗi và đảm bảo rằng lỗi đã được sửa chữa hoàn toàn. Công việc này sẽ do tester thực hiện.
- Trạng thái lỗi lúc này là TESTED
- Sau khi các lỗi đã được kiểm tra lại, nhóm phát triển phải cập nhật sản phẩm sau khi xử lý lỗi, Update các trạng thái lỗi.

d) Đầu ra

- Có được phiên bản mới nhất sau khi được sửa lỗi
- Cập nhật tình huống kiểm tra nếu có lỗi mới phát sinh
- Cập nhật dữ liệu mới vào công cụ kiểm tra
- Kết quả xử lý lỗi phát hiện

e) Người thực hiện

- Cán bộ kiểm tra
- Trưởng nhóm PQA
- Nhóm phát triển
- Quản trị dự án

3.8. Lập kế hoạch và triển khai kiểm thử hồi quy

a) Đầu vào

- Kế hoạch dự án
- Báo cáo kết quả kiểm tra
- Tình huống kiểm tra
- Phiên bản phần mềm mới nhất trước khi được sửa lỗi
- Công cụ hỗ trợ

b) Mô tả

Trong thực tế, lỗi là phần “luôn hiện diện” trong mọi PM từ giai đoạn phát triển sơ khởi đến khi nó không còn được sử dụng. Lập kế hoạch và triển khai test lại được tiến hành sau khi nhà phát triển nhận tiếp nhận lỗi từ phía khách hàng và được

nhóm phát triển thực hiện xử lý lỗi. Việc lập kế hoạch và triển khai test lại nhằm mục đích là để chắc chắn rằng chúng ta đang phát triển một thứ có khả năng đúng và hữu ích đối với khách hàng. Chuẩn bị và triển khai test lại: Môi trường kiểm thử, test case, test script... Thực hiện test dựa trên kế hoạch test lại.

c) Các bước lập kế hoạch và triển khai test lại

- *Lập kế hoạch thực hiện test lại:* được thực hiện bởi Test Manager nhằm đưa ra kế hoạch thực hiện test lại các lỗi sau khi đã được sửa. Lập kế hoạch cho test lại tương tự như lập kế hoạch kiểm tra cho một dự án. Căn cứ vào số lượng lỗi và thời gian bàn giao sản phẩm cho khách hàng mà Test Manager cần phải đưa ra kế hoạch test lại cho hợp lý.
- *Chuẩn bị và triển khai test lại:* bao gồm các tình huống kiểm tra, thủ tục kiểm tra, tài nguyên hệ thống cần thiết
- *Thực hiện test dựa trên kế hoạch test lại:* Cần ưu tiên kiểm tra những lỗi nghiêm trọng trước tiên sau đó đến các lỗi khác... Đánh giá, tổng hợp lại kết quả kiểm tra sau khi đã test lại

d) Kết quả

- Báo cáo kết quả kiểm tra lỗi được test lại
- Báo cáo kết toàn bộ hệ thống sau khi test lại
- Có được phiên bản mới nhất sau khi được test lại

e) Người thực hiện

- Cán bộ kiểm tra
- Trưởng nhóm PQA
- Nhóm phát triển
- Quản trị dự án

3.9. Thông báo phát hành sản phẩm

a) Đầu vào

- Tài liệu SRS, SRD
- Kế hoạch kiểm tra
- Báo cáo kết quả kiểm tra hệ thống phần mềm

- Tình huống kiểm tra đã cập nhập

b) Mô tả

Đây là bước cuối cùng của quá trình kiểm tra nhằm mục đích thông báo sản phẩm phần mềm mới đã đạt yêu cầu và sẵn sang chuyển giao cho khách hàng.

c) Quá trình thực hiện

- Làm thủ tục thông báo phát hành

d) Kết quả

- Biên bản thông báo phát hành sản phẩm
- Tài liệu hướng dẫn sử dụng

e) Người thực hiện

- Trưởng nhóm PQA

3.10. Xây dựng kế hoạch kiểm thử

1- Định nghĩa: Kế hoạch kiểm thử thường được để trong 1 file và chứa các kết quả của các hoạt động sau :

- Nhận dạng các chiến lược được dùng để kiểm tra và đảm bảo rằng sản phẩm thỏa mãn đặc tả thiết kế phần mềm và các yêu cầu khác về phần mềm.
- Định nghĩa các mục tiêu và phạm vi của nỗ lực kiểm thử.
- Nhận dạng phương pháp luận mà đội kiểm thử sẽ dùng để thực hiện công việc kiểm thử.
- Nhận dạng phần cứng, phần mềm và các tiện ích cần cho kiểm thử.
- Nhận dạng các tính chất và chức năng sẽ được kiểm thử.
- Xác định các hệ số rủi ro gây nguy hại cho việc kiểm thử.
- Lập lịch kiểm thử và phân phối công việc cho mỗi thành viên tham gia.

Test Manager hoặc Test Leader sẽ xây dựng kế hoạch kiểm thử.

2- Nhu cầu cần phải có kế hoạch kiểm thử:

Kế hoạch kiểm thử cần phải được xây dựng sớm nhục có thể có trong mỗi chu kỳ phát triển phần mềm để:

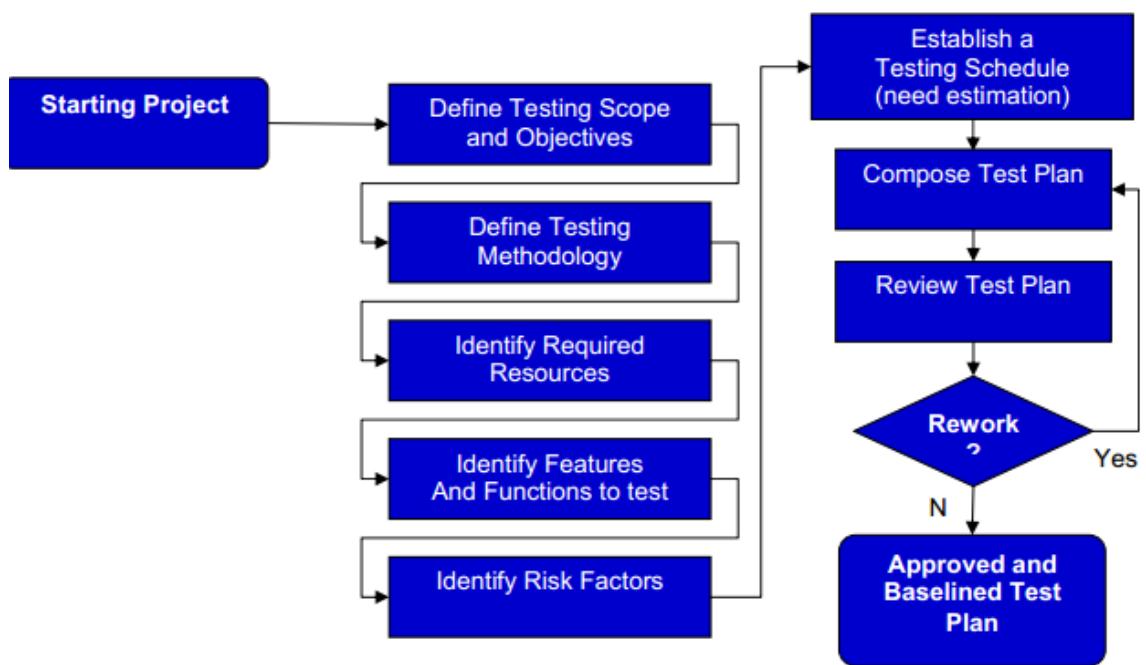
- Tập hợp và tổ chức các thông tin kiểm thử cần thiết.
- Cung cấp thông tin về qui trình kiểm thử sẽ xảy ra trong tổ chức kiểm thử.
- Cho mỗi thành viên trong đội kiểm thử có hướng đi đúng.

- Gán các trách nhiệm rõ ràng cụ thể cho mỗi thành viên đội kiểm thử.
- Có lịch biểu làm việc rõ ràng và các thành viên có thể làm việc với nhau tốt.

3- Kế hoạch kiểm thử cần chứa các thông tin sau đây :

- Phạm vi/mục tiêu kiểm thử
- Các chiến lược được dùng
- Các tài nguyên phần cứng và phần mềm phục vụ kiểm thử.
- *f* Các nhu cầu về nhân viên và huấn luyện nhân viên.
- *f* Các tính chất cần được kiểm thử.
- *f* Các tính chất không cần kiểm thử.
- *f* Các rủi ro & sự cố bất ngờ.
- *f* Lịch kiểm thử cụ thể.
- *f* Các kênh thông tin liên lạc.
- *f* Cấu hình cho từng phần tử như kế hoạch kiểm thử, testcase, thủ tục kiểm thử,...
- Môi trường kiểm thử(Test bed)
- Tiêu chí đầu vào và tiêu chí dùng kiểm thử.
- Các kết quả phân phối

4- Quy trình xây dựng kế hoạch kiểm thử



Sau khi xây dựng kế hoạch kiểm thử xong ta có thể thay đổi nó, nhưng cần tuân thủ quy trình yêu cầu thay đổi

5- Các hoạt động chính trong việc xây dựng kế hoạch kiểm thử

- Định nghĩa mục đích, phạm vi, chiến lược, cách tiếp cận, các điều kiện chuyên, các rủi ro, kế hoạch giảm nhẹ và tiêu chí chấp thuận.
- Định nghĩa cách thức thiết lập môi trường và các tài nguyên được dùng cho việc kiểm thử.
- Thiết lập cơ chế theo dõi lỗi phát hiện.
- Chuẩn bị ma trận theo dõi bao phủ mọi yêu cầu phần mềm.
- Báo cáo trạng thái kiểm thử.
- Phát hành leo thang (Escalating Issues)
- Raising Testing related PIR (Process Improvement Request) / PCR (Process Change Request)

3.11. Các thành phần chính trong kế hoạch kiểm thử

1- Mục đích và phạm vi kiểm thử:

- Đặc tả mục đích của tài liệu về kế hoạch kiểm thử.
- Cung cấp vắn tắt về phạm vi mà project được hỗ trợ như platform, loại database, hay danh sách vắn tắt về các loại project con in project kiểm thử.

□ Ví dụ:

Purpose

The purpose of this document is to give stakeholders, as well as team members, a detailed quality assurance plan covering test requirements for the release. Test plan objectives include, but are not limited to:

- Identify existing project information and the software components that should be tested
- List the recommended test requirements
- Recommend and describe the testing strategies to be employed
- Include traceability from Requirement to Test Case
- List the deliverable elements of the test project

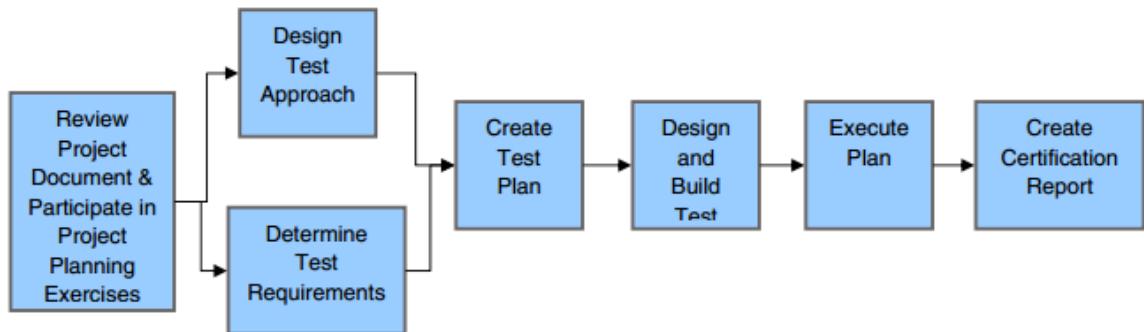
Testing scope

This section to provide test requirements, strategies as below:

- Operation will be tested: Windows XP SP2, SP3 + Latest security updated from Microsoft.
- Database type: Microsoft SQL Server 2005
- Browsers: Internet Explorer 7
- The sub-products will be tested as below:
 - Quality Monitoring 9.0 SP3
 - Agent Capture
 - UST/BUIT
 - Media Testing
 - Documents verification
 - Installation/Upgrade testing

2- Cách tiếp cận & các chiến lược được dùng :

- Đặc tả về phương pháp luận kiểm thử sẽ được dùng để thực hiện kiểm thử.



- Thí dụ: General Testing Process Approach for Project ABC

Đề cập các cấp độ kiểm thử cần thực hiện

Các kỹ thuật được dùng cho mỗi kiểu kiểm thử trong project :

- Kiểm thử tích hợp (Integration Testing)
- Kiểm thử hệ thống (System Testing)
- Kiểm thử độ chấp thuận (Acceptance Testing)
- Kiểm thử chức năng của người dùng (Functionality Testing)
- Kiểm thử hồi qui (Regression Testing)
- Kiểm thử việc phục hồi sau lỗi (Failover and Recovery Testing)
- Kiểm thử việc kiểm soát an ninh và truy xuất (Security and Access Control Testing)
- Kiểm thử việc cấu hình và cài đặt (Configuration and Installation Testing)
- Kiểm thử đặc biệt (Ad-hoc Testing)
- Kiểm thử hiệu suất (Performance Testing)

3- Các tính chất cần được kiểm thử

Danh sách các tính chất của phần mềm cần được kiểm thử, đây là 1 catalog chứa tất cả các testcase (bao gồm chỉ số testcase, tiêu đề testcase) cũng như tất cả trạng thái cơ bản

Ví dụ:

New Feature and Requirements

List all features in scope. Requirement to Test Case Traceability will be updated in Quality Center.

Planned Iteration	Req ID	Requirement	Test Case(s) (should map to QC)	Reviewer
1	ABC-2001	Support ABC Client in 64-bit Citrix/TS	ABC-2001 - Support ABC Client in 64-bit Citrix/TS	
1	ABC-2002	Enhance Supervisor Installation for JRE Selection	ABC-2002 - Enhance Supervisor Installation for JRE Selection	
2	ABC-1065	Dynamic Workspaces	ABC-1065 - Dynamic Workspaces	
3	ABC-1108	Configurable JRE Versioning	ABC-1108 - Configurable JRE Versioning	
3	ABC-1093	Remove DCOM Dependency	ABC-1093 - Remove DCOM Dependency	
3	ABC-1106	DSE (VTG) Card Replacement	ABC-1106 - DSE (VTG) Card Replacement	

Regression

ID	Test Case Name (Should Map to QC)	Automated?
123	__FTC - Abandoned Call	N
124	__FTC - Basic Call	N
125	__FTC - Conference Call	N
126	__FTC - Consultation Call	N
127	__FTC - Race Condition	N
128	__FTC - Transfer Call	N
454	__Inters - Playback	N
455	__Inters - Agent List Security	N

4- Các tính chất không cần kiểm thử

- Danh sách các vùng phần mềm được loại trừ khỏi kiểm thử, cũng như các testcase đã được định nghĩa nhưng không cần kiểm thử.

De-scoped Test Cases / Out of Scope Testing

+

ID	Test Case	Reason
	General	Low priority
	Install	Low priority
	Error Handling	Low priority
	JRE	Low priority
	Native Player	Low priority
	Mini Regression	Low priority

5- Rủi ro và sự cố bất ngờ

- Danh sách tất cả rủi ro có thể xảy ra trong chu kỳ kiểm thử.
- Phương pháp mà ta cần thực hiện để tối thiểu hóa hay sống chung với rủi ro
- Ví dụ

#	Risks	Contingency Plan	Level
1	Requirement changes affect to human resources and test strategy.	Need to do re-planning with a realistic schedule to deal with changed requirement: either extend schedule or/and increase resource if need to execute testing for new requirement or previous requirement if changing requirements effect on previous requirements	High
2	Build is not ready on time as project schedule.	Development team must follow schedule to create build strictly. If not, the escalation will be issued by Quality Control Lead	High
3	Requirements are not base-lined on time.	Onshore team must follow-up to get requirements baselined on time.	High
4	Late response on issues	The feedback must be provided in 5 working days when an issue is raised.	Medium
5	Virus effectively	Highlight this to everyone in team, update the newest version for anti virus program. Turn on the auto-update for Windows service patch/	High

6- Tiêu chí định chỉ & phục hồi kiểm thử:

- Tiêu chí định chỉ kiểm thử là các điều kiện mà nếu thỏa mãn thì kiểm thử sẽ dừng lại.
- Tiêu chí phục hồi là những điều kiện được đòi hỏi để tiếp tục việc kiểm thử đã bị ngưng trước đó.

1.6 Các nguyên tắc cơ bản về kiểm thử

Suspension Criteria

The testing will be halted if these criteria below happen:

- No build notes or it is not clear
- There are some Fatal errors in smoke test build without work around solutions

Resumption Criteria

The testing will be resumed if the build has:

- Build notes clearly
- Any fatal errors with work around solutions
- Test cases had been baselined

7- Môi trường kiểm thử

- Đặc tả đầy đủ về các môi trường kiểm thử, bao gồm đặc điểm cứng, mang, database, phần mềm, hệ điều hành và các thuộc tính môi trường khác ảnh hưởng đến kiểm thử.
- Ví dụ:

Web Server

Covered	Not Covered
Tomcat 5.5.9 with JRE 1.5.0 Update 17	

JRE

Covered	Not Covered
JRE 1.6.0 Update 07	
JRE 1.5.0 Update 17 (Default JRE installed)	
JRE 1.4.2 _Update 19	

Web Browsers

Covered	Not Covered
Microsoft Internet Explorer 7.0	
Microsoft Internet Explorer 6.0, Service Pack 3 for Windows XP	Microsoft Internet Explorer 6.0, Service Pack 1 for Windows 2003
Microsoft Internet Explorer 8.0 (if available)	Note: Popup blockers are not supported

Server OS

Covered	Not Covered
Microsoft Windows 2003 Enterprise Edition 32 bit, Service Pack 2	Microsoft Windows 2003 Standard Edition 32 bit, Service Pack 2

Client OS (Supervisor, Agent)

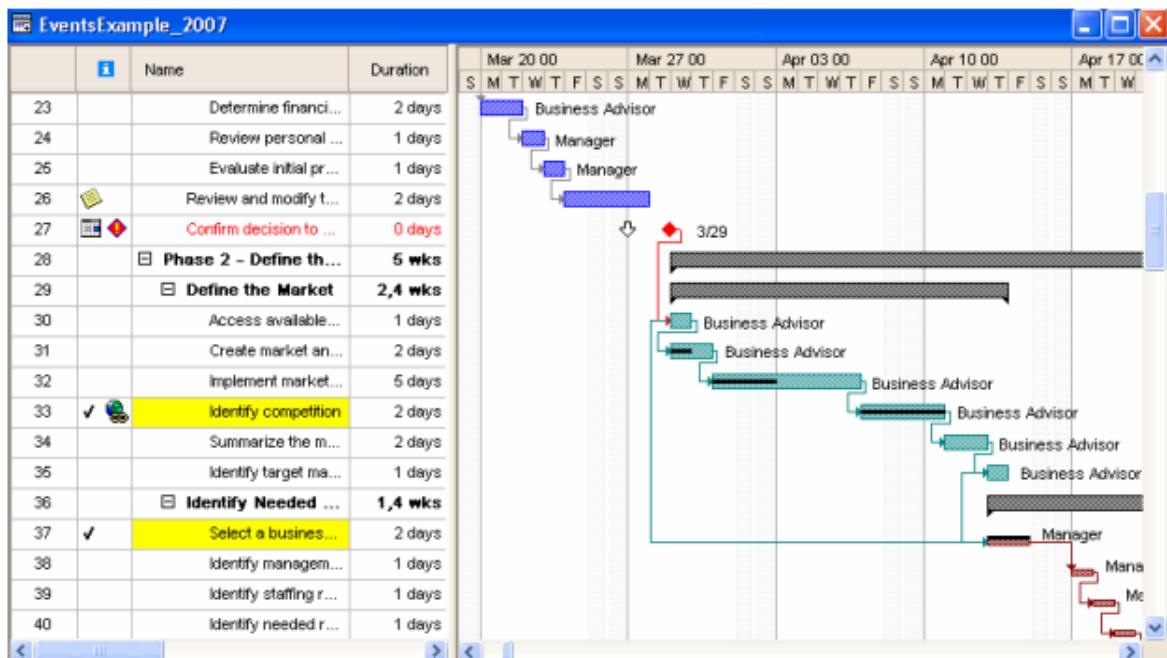
Covered	Not Covered
Microsoft Windows XP Professional 32 bit, Service Pack 2 (Use SP3 if available) (25%)	Microsoft Vista Enterprise 32 bit, Service Pack 1
Microsoft Vista Business 32 bit, Service Pack 1 (25%)	Microsoft Vista Ultimate 32 bit, Service Pack 1
MS 2000 Professional 32 bit, SP 4 (25%)	Microsoft Windows 2003 Standard Edition 32 bit, Service Pack 2

Database

Covered	Not Covered
Microsoft SQL Server 2005 Service Pack 3 Enterprise Edition	Microsoft SQL Server 2005 Service Pack 2 Standard Edition
Oracle 10g Release 2 (10.2.0.1.0) Enterprise Edition	Microsoft SQL Server 2005 Service Pack 2 Enterprise Edition
	Support for SQL Server Database Cluster environment - active-passive

8- Lịch kiểm thử

- Lịch kiểm thử ódạng ước lượng, nên chứa các thông tin : các cột mốc với ngày xác định + Kết quả phân phối của từng cột mốc.
- Ví dụ:



f

9- Tiêu chí dừng kiểm thử và chấp thuận

Bất kỳ chuẩn chất lượng mong muốn nào mà phần mềm phải thỏa mãn yêu cầu sẵn sàng cho việc phân phối đến khách hàng. Có thể bao gồm các thứ sau :

- Các yêu cầu mà phần mềm phải được kiểm thử dưới các môi trường xác định.
- Số lỗi tối thiểu ở cấp an ninh và ưu tiên khác nhau, số phủ kiểm thử tối thiểu,..
- Stakeholder sign-off and consensus
- Ví dụ:

Exit Criteria

- 100% of test cases have been executed and passed.
- Stability requirements have been met.
- Defect criteria has been met as follows
 - Critical - 0
 - High - 0
 - Medium - 20
 - Low - 50

10-Nhân sự:

Vai trò và trách nhiệm từng người :

- Danh sách các vai trò xác định của các thành viên đội kiểm thử trong hoạt động kiểm thử.
- Các trách nhiệm của từng vai trò.
- Công tác huấn luyện.
- Danh sách các huấn luyện cần thiết cho các QC
- Ví dụ:

Worker	Minimum Resources Recommended	Specific Responsibilities/Comments
Test Leader A Nguyen	20%	Provides management oversight Responsibilities: <ul style="list-style-type: none"> • Provide technical direction • Acquire appropriate resources • Management reporting • Ensures test environment and assets are managed and maintained.
Test Designer A Nguyen B Tran C Ngo D Tran	40% 50% 50% 50%	Identifies, prioritizes, and implements test cases Responsibilities: <ul style="list-style-type: none"> • Generate test cases • Evaluate effectiveness of test effort • Prepare test data
Tester A Nguyen B Tran C Ngo D Tran	40% 50% 50% 50%	Executes the tests Responsibilities: <ul style="list-style-type: none"> • Execute tests • Log results • Recover from errors • Document defects

11-Các tool phục vụ kiểm thử

- Danh sách tất cả các tiện ích cần dùng trong suốt chu kỳ kiểm thử.
- Với project kiểm thử tự động, các tiện ích cần được liệt kê với chỉ số version cùng thông tin license.
- Ví dụ:

Test Tools

During the test cycle, following tools will be used with its purposes:

- HP Quality Center: for storing Test cases, Defects and Test cases status
- Guru site: repository for the latest requirements, test metrics

Automation Test tool

Because there is requested for performance test, tool below will be used with license had been payment fully.

- HP LoadRunner 9.2

12-Các kết quả phân phối

- Danh sách tất cả tài liệu hay artifacts dự định phân phối nội bộ sau khi mỗi cột mốc kết thúc hay sau khi project kết thúc.
- Ví dụ:

Test Deliverables

Following is documents, artifacts that will be delivered at the end of testing life cycle

- Test cases
- Status for each Test cases under tested
- Defect reports
- Defect metric will be delivered weekly along with weekly report of project
- Test Scripts for performance test with its results
- Certification test

BÀI 4. CÁC MỨC ĐỘ KIỂM THỬ PHẦN MỀM – TEST LEVELS

Thực tế, KTPM không đơn giản như nhiều người thường nghĩ, công việc này có nhiều mức độ khác nhau và có mối tương quan với các chặng phát triển trong dự án PTPM. Hình 1 cho thấy 4 mức độ cơ bản của KTPM và hình 2 cho thấy mối tương quan với các chặng PTPM trong mô hình V-model.

Phần sau sẽ làm rõ chi tiết về các mức độ KTPM, do một số thuật ngữ không có từ tương đương sát nghĩa trong tiếng Việt, mặt khác để các bạn tiện tham khảo sau này, chúng tôi xin giữ nguyên một số thuật ngữ gốc tiếng Anh.

Quá trình kiểm thử có thể chia làm các giai đoạn :

- Kiểm thử đơn vị
- Kiểm thử tích hợp
- Kiểm thử hệ thống
- Kiểm thử chấp nhận
- Kiểm thử hồi quy

4.1. Unit Test – Kiểm thử mức độ đơn vị

Để có thể hiểu rõ về Unit Test, khái niệm trước tiên ta cần làm rõ: thế nào là một đơn vị PM (Unit)?

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

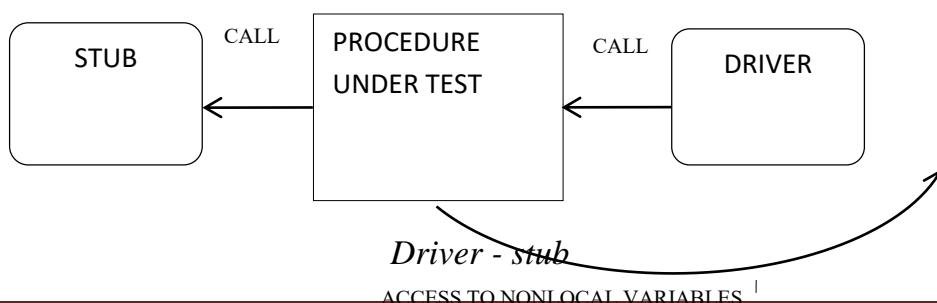
Vì Unit được chọn để kiểm tra thường có kích thước nhỏ và chức năng hoạt động đơn giản, chúng ta không khó khăn gì trong việc tổ chức, kiểm tra, ghi nhận và phân tích kết quả kiểm tra. Nếu phát hiện lỗi, việc xác định nguyên nhân và khắc phục cũng tương đối dễ dàng vì chỉ khoanh vùng trong một đơn vị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

việc tiết kiệm rất nhiều thời gian và chi phí cho việc kiểm tra và sửa lỗi ở các mức kiểm tra sau đó.

Unit Test thường do lập trình viên thực hiện. Công đoạn này cần được thực hiện càng sớm càng tốt trong giai đoạn viết code và xuyên suốt chu kỳ PTPM. Thông thường, Unit Test đòi hỏi kiểm tra viên có kiến thức về thiết kế và code của chương trình. Mục đích của Unit Test là bảo đảm thông tin được xử lý và xuất (khỏi Unit) là chính xác, trong mối tương quan với dữ liệu nhập và chức năng của Unit. Điều này thường đòi hỏi tất cả các nhánh bên trong Unit đều phải được kiểm tra để phát hiện nhánh phát sinh lỗi. Một nhánh thường là một chuỗi các lệnh được thực thi trong một Unit, ví dụ: chuỗi các lệnh sau điều kiện If và nằm giữa then ... else là một nhánh. Thực tế việc chọn lựa các nhánh để đơn giản hóa việc kiểm tra và quét hết Unit đòi hỏi phải có kỹ thuật, đôi khi phải dùng thuật toán để chọn lựa.

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

Module không phải là chương trình cô lập, nên cần phát triển các phần mềm driver (bộ lái) và/hoặc stub (cuống) cho kiểm thử mỗi unit. Driver là module gọi thực thi làm cho module cần kiểm tra hoạt động, nó giả lập các module khác sẽ sử dụng module này. Các tập dữ liệu chia sẻ mà các module khác thiết lập trong thực tế cũng được thiết lập ở driver. Stub là module giả lập các module được module đang kiểm tra sử dụng.



Sử dụng hai kỹ thuật Black – Box và White – Box để xây dựng các tình huống (TC) hoặc kịch bản (script), trong đó chỉ định rõ dữ liệu vào, các bước thực hiện và dữ liệu mong chờ sẽ xuất ra. Các TC và script này nên được giữ lại để tái sử dụng.

Nội dung kiểm thử

- Giao diện
- Lỗi vào ra
- Cấu trúc dữ liệu sử dụng cục bộ
- Dòng điều khiển
- Điều kiện logic
- Phép toán xử lý
- Những lỗi tiềm ẩn

Kiểm thử dữ liệu qua giao diện

Kiểm thử này liên quan đến định lượng và định dạng của các biến và các module sử dụng trên giao diện. Các đặc trưng qua giao diện:

- Số tham số = Số đối số?
- Tính chất của tham số ≡ Tính chất đối số?
- Đơn vị của tham số ≡ Đơn vị đối số ?
- Số đối số được truyền gọi module = Số các tham số đầu vào của module?
- Thuộc tính các đối số được truyền gọi module ≡ Thuộc tính các tham số?
- Đơn vị của đối số được truyền để gọi module ≡ Đơn vị các tham số module?
- Thuộc tính và các thành phần khác của các đối số định sẵn có đúng không?
- Các đối số chỉ đọc đã được đổi chưa?
- Định nghĩa biến toàn cục có trong suốt module?
- Các ràng buộc lên biến đã chuyển qua như tham số chưa?

Kiểm thử liên quan đến vào ra

Khi thực hiện I/O bên ngoài cần tiến hành thêm:

- Tính chất của các file có đúng đắn không?
- Các câu lệnh mở/đóng có đúng không?
- Đặc tả hình thức có đúng với các câu lệnh I/O ?

- Cỡ của bộ đệm có phù hợp với cỡ của bản ghi?
- Các file có mở trước khi sử dụng ?
- Các điều kiện end – of –file có được xử lý không?
- Các ngoại lệ I/O có được xử lý không?
- Có sai văn bản nào trong thông tin ra?

Kiểm thử cấu trúc dữ liệu cục bộ

Tìm ra các loại lỗi sau:

- Giá trị ngầm định hoặc giá trị khởi tạo sai
- Tên các biến, hàm không đúng (sai chữ hoặc mất chữ)
- Kiểu dữ liệu không nhất quán
- Ngoại lệ về địa chỉ, sự thiếu hay tràn bộ nhớ.

Kiểm thử dòng điều khiển

Một số lỗi như:

- Vòng lặp không kết thúc hoặc kết thúc không chính xác
- Sự lặp phân kỳ khó thoát được
- Các biến lặp bị cải biến không chính xác

Điều khiển logic

Tìm ra các lỗi về kiểu, toán tử, ngữ nghĩa:

- So sánh các kiểu dữ liệu khác nhau
- Ưu tiên hoặc toán tử logic không đúng đắn
- Dự đoán một biểu thức so sánh, trong khi sai số là cho đẳng thức không chắc có thực.
- Các giá trị so sánh không đúng đắn

Phép toán xử lý

Các sai về trình tự, độ chính xác gồm có:

- Thứ tự ưu tiên các phép tính số học
- Sự nhất quán của các phép toán trộn kiểu
- Khởi tạo hay kết thúc không đúng đắn
- Sự chính xác của kết quả

Lỗi tiềm ẩn

Các lỗi tiềm ẩn cần được đánh giá:

- Mô tả sai hay khó hiểu
- Dữ liệu ghi không tương ứng với sai đã gặp
- Điều kiện sai có trước khi xử lý sai
- Xử lý ngoại lệ không đúng đắn
- Mô tả sai không cung cấp đủ thông tin để trợ giúp định vị nguyên nhân sai

Các kỹ thuật Unit test

Có các kỹ thuật Unit test sau:

- Bao phủ câu lệnh: Statement Coverage
- Bao phủ nhánh/quyết định – Branch/Decision Coverage
- Bao phủ đường đi – Path Coverage

Bao phủ câu lệnh: Trong trường hợp test case được thực thi theo cách mà mỗi câu lệnh của code là thực thi ít nhất một lần

Bao phủ nhánh/quyết định: Điều kiện bao phủ test yêu cầu đủ các test case như là mỗi điều kiện trong nhánh quyết định lấy tất cả các nhánh có thể ít nhất một lần. Theo đó, mỗi nhánh (quyết định) cần lấy theo mỗi đường true và false. Nó giúp thẩm tra lại tất cả các nhánh trong code để chắc chắn rằng không nhánh nào bị bỏ qua thường trong ứng dụng.

Bao phủ đường đi: test case được thực thi theo cách mà mỗi đường đi được thực thi ít nhất một lần. Tất cả khả năng điều khiển đường đi đều lấy, bao gồm tất cả đường đi có thể lấy được.

Cách tính toán số lượng test case cho bao phủ câu lệnh, bao phủ nhánh và bao phủ đường đi

Vẽ lưu đồ thuật toán từ đó:

- Bao phủ câu lệnh: phủ qua các node trên lưu đồ ít nhất một lần
- Bao phủ nhánh/quyết định: phủ qua ít nhất mỗi cạnh một lần

Ví dụ:

Read P

Read Q

If $P + Q > 100$ then Print “Large”

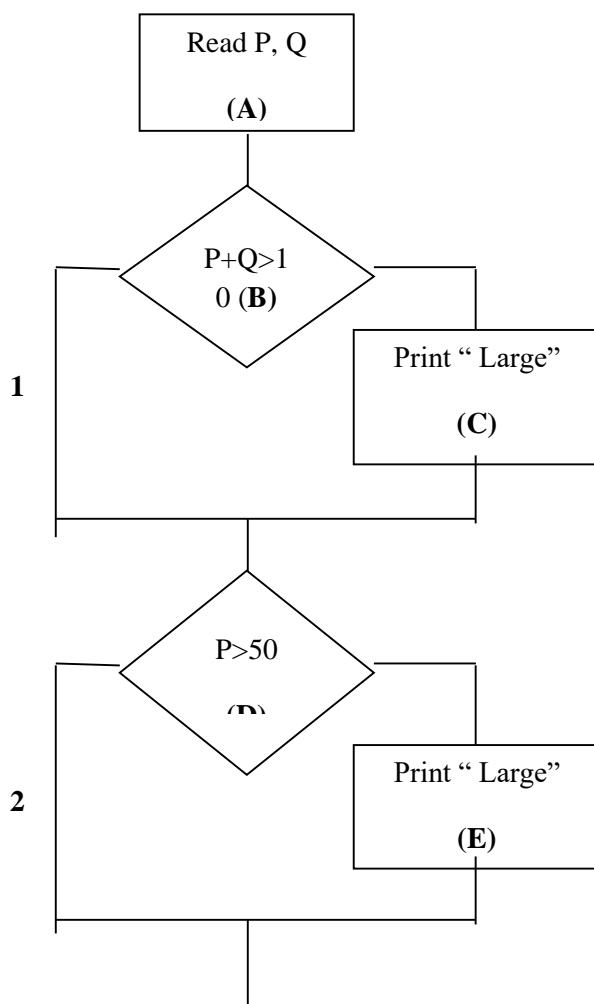
End if

If $P > 50$ then Print “P Large”

End if

Tính toán số lượng test case bao phủ câu lệnh, bao phủ nhánh và bao phủ đường đi.

Hướng dẫn: Ta có lưu đồ thuật toán:



Statement coverage	A - B - C - D - E
Branch/Decision	A - B - C - D - E

coverage	A – B – 1 – D – 2
Path coverage	A - B – C – D – E
	A – B – 1 – D – 2
	A – B – C – D – 2
	A- B – 1 – D – E

Như vậy, bao phủ câu lệnh có tối thiểu 1 test case, bao phủ nhánh/quyết định có 2 test case và bao phủ đường đi có 4 test case cho bài toán trên.

Kết luận:

100% bao phủ đường đi sẽ bao phủ 100% nhánh/quyết định

100% bao phủ nhánh sẽ bao phủ 100 % câu lệnh

100% bao phủ đường đi sẽ bao phủ 100% câu lệnh

100% bao phủ nhánh sẽ bao phủ 100% quyết định

100% bao phủ quyết định sẽ bao phủ 100% nhánh

4.2. Integration Test – Kiểm thử mức tích hợp

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

Integration Test có 2 mục tiêu chính:

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

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

được kiểm tra đầy đủ khi các Unit tích hợp với nhau trong khi thực hiện Integration Test.

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

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

Có 4 loại kiểm tra trong Integration Test:

- Kiểm tra cấu trúc (structure): Tương tự White Box Test (kiểm tra nhằm bảo đảm các thành phần bên trong của một chương trình chạy đúng), chú trọng đến hoạt động của các thành phần cấu trúc nội tại của chương trình chẳng hạn các lệnh và nhánh bên trong.
- Kiểm tra chức năng (functional): Tương tự Black Box Test (kiểm tra chỉ chú trọng đến chức năng của chương trình, không quan tâm đến cấu trúc bên trong), chỉ khảo sát chức năng của chương trình theo yêu cầu kỹ thuật.
- Kiểm tra hiệu năng (performance): Kiểm tra việc vận hành của hệ thống.
- Kiểm tra khả năng chịu tải (stress): Kiểm tra các giới hạn của hệ thống.

Là kiểm tra sự giao tiếp giữa các Unit khi các Unit này được tích hợp với nhau.

Điều kiện tiên quyết: Các Unit đã được kiểm tra thành công tại mức Unit test (trừ trường hợp có yêu cầu đặc biệt)

Mục tiêu:

- Phát hiện lỗi giao tiếp xảy ra giữa các Unit
- Tích hợp các Unit đơn lẻ thành hệ thống nhỏ - SubSystem và thành hệ thống hoàn chỉnh – System chuẩn bị cho bước kiểm tra mức hệ thống.

Các lỗi có thể gặp khi tích hợp:

- Dữ liệu bị mất khi đi qua một giao diện/ Chức năng.
- Hiệu ứng bất lợi: một modul vô tình gây ra đối với các modul khác.
- Sự kết hợp các chức năng phụ: có thể không sinh ra chức năng chính mong muốn.
- Sự phỏng đại: các sai sót riêng rẽ có thể tới mức không chấp nhận được.
- Vấn đề của cấu trúc dữ liệu toàn cục có thể để lộ ra.

Gồm có:

- Kiểm thử cấu trúc: tương tự White-Box Test
- Kiểm thử chức năng: tương tự Black – Box Test
- Kiểm tra hiệu năng
- Kiểm tra khả năng chịu tải

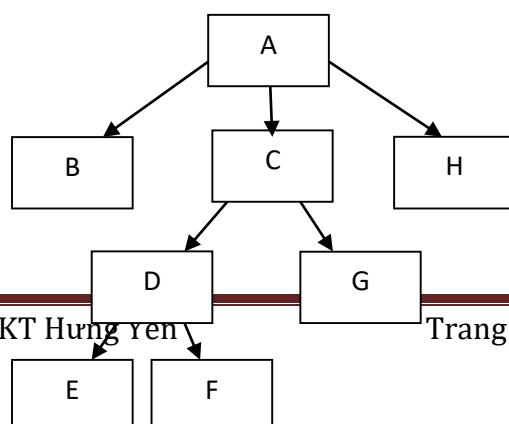
Kỹ thuật:

- Tích hợp từ trên xuống (Top-Down)
- Tích hợp từ dưới lên (Bottom -Up)
- Tích hợp kẹp (Sandwich).

Tích hợp từ trên xuống

Gộp dàn các module từ trên xuống theo trật tự dòng điều khiển, bắt đầu từ module điều khiển “main”, gắn với từng module phụ trợ vào module điều khiển thượng cấp.

Điều kiện tiên quyết: Modul ở mức cao đã được test và tích hợp đầu tiên



Top - down

Thực hiện theo các bước:

1. Module điều khiển chính được dùng như test driver và tất cả các module phụ trợ trực tiếp được thay thế bởi các stub.
2. Thay thế dần từng stub bởi module thực thi tương ứng.
3. Sau khi tích hợp module đó tiến hành các kiểm thử tương ứng.
4. Khi hoàn thành các kiểm thử trên thì thay một stub khác bằng module thực (tức là quay lại bước 2).
5. Có thể kiểm thử lại (kiểm thử hồi quy) để đảm bảo không có sai sót nào được sinh ra.
6. Tiếp tục lặp lại từ bước 2 cho tới khi toàn bộ cấu trúc chương trình được xây dựng.

Thuật tiện

- Không cầm có driver kiểm thử.
- Lỗi giao diện được phát hiện sớm.

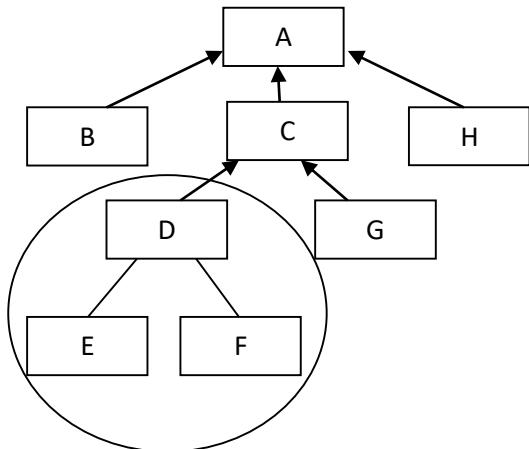
Bất tiện

- Cần cuống (stubs).
- Làm chậm tiến trình kiểm thử.
- Lỗi ẩn trong các modul ở mức thấp khó tìm ra.

Tích hợp từ dưới lên

Bắt đầu xây dựng và kiểm thử từ các module nguyên tố ở mức thấp trước. Thông thường, người ta thường nhóm các module tầng dưới thành các nhóm chức năng, tích hợp và kiểm thử chúng theo từng nhóm.

Điều kiện tiên quyết: Modul ở mức thấp đã được test và tích hợp đầu tiên



Bottom – Up

Thực hiện các bước:

1. Các module mức thấp được tổ hợp vào trong các cụm (cluster) thực hiện một chức năng phụ trợ đặc biệt.
2. Một driver được viết để phôi hợp đầu vào và đầu ra của ca kiểm thử.
3. Kiểm thử cụm đó
4. Tháo bỏ các driver và các cụm tổ hợp ngược lên trong cấu trúc chương trình.

Thuận tiện

- Không cần đến gốc
- Rất dễ điều chỉnh số lượng người cần thiết
- Lỗi quyết định sớm được tìm thấy

Bất tiện

- Các driver kiểm thử là cần thiết
- Rất nhiều modul phải được tích hợp trước khi làm việc
- Lỗi giao diện khám phá muộn

Tích hợp kẹp

Là một phương pháp kiểm thử kết hợp cả Top-Down và Bottom-Up.

Tất cả các modul và giao diện đều phải kiểm thử bằng phương pháp Top-Down.

Cả driver và stub đều được sử dụng khi cần thiết.

Tất cả các modul đều được xây dựng và kiểm thử unit bắt đầu từ mức thấp nhất, sử dụng chiến thuật Bottom-Up.

4.3. System Test - Kiểm thử mức hệ thống

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

System Test bắt đầu khi tất cả các bộ phận của PM đã được tích hợp thành công. Thông thường loại kiểm tra này tốn rất nhiều công sức và thời gian. Trong nhiều trường hợp, việc kiểm tra đòi hỏi một số thiết bị phụ trợ, phần mềm hoặc phần cứng đặc thù, đặc biệt là các ứng dụng thời gian thực, hệ thống phân bố, hoặc hệ thống nhúng. Ở mức độ hệ thống, người kiểm tra cũng tìm kiếm các lỗi, nhưng trọng tâm là đánh giá về hoạt động, thao tác, sự tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống.

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

Sau khi hoàn thành Integration Test, một hệ thống PM đã được hình thành cùng với các thành phần đã được kiểm tra đầy đủ. Tại thời điểm này, lập trình viên hoặc kiểm tra viên (tester) bắt đầu kiểm tra PM như một hệ thống hoàn chỉnh. Việc lập kế hoạch cho System Test nên bắt đầu từ giai đoạn hình thành và phân tích các yêu cầu. Phần sau ta sẽ nói rõ hơn về một quy trình System Test cơ bản và điển hình.

System Test kiểm tra cả các hành vi chức năng của phần mềm lẫn các yêu cầu về chất lượng như độ tin cậy, tính tiện lợi khi sử dụng, hiệu năng và bảo mật. Mức kiểm tra này đặc biệt thích hợp cho việc phát hiện lỗi giao tiếp với PM hoặc phần

cứng bên ngoài, chẳng hạn các lỗi "tắc nghẽn" (deadlock) hoặc chiếm dụng bộ nhớ. Sau giai đoạn System Test, PM thường đã sẵn sàng cho khách hàng hoặc người dùng cuối cùng kiểm tra để chấp nhận (Acceptance Test) hoặc dùng thử (Alpha/Beta Test).

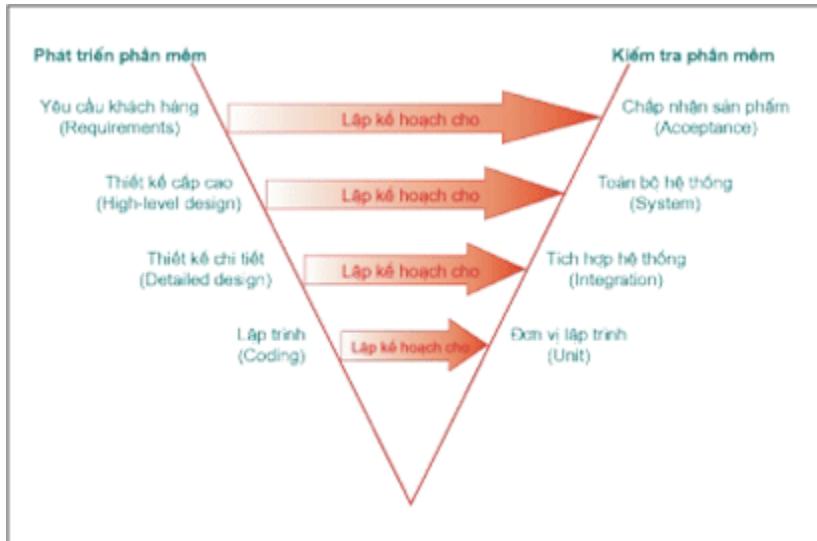
Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan, System Test thường được thực hiện bởi một nhóm kiểm tra viên hoàn toàn độc lập với nhóm phát triển dự án.

Bản thân System Test lại gồm nhiều loại kiểm tra khác nhau (xem hình 6.2), phổ biến nhất gồm:

- Kiểm tra chức năng (Functional Test): bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
- Kiểm tra khả năng vận hành (Performance Test): bảo đảm tối ưu việc phân bổ tài nguyên hệ thống (ví dụ bộ nhớ) nhằm đạt các chỉ tiêu như thời gian xử lý hay đáp ứng câu truy vấn...
- Kiểm tra khả năng chịu tải (Stress Test hay Load Test): bảo đảm hệ thống vận hành đúng dưới áp lực cao (ví dụ nhiều người truy xuất cùng lúc). Stress Test tập trung vào các trạng thái tới hạn, các "điểm chết", các tình huống bất thường...
- Kiểm tra cấu hình (Configuration Test)
- Kiểm tra khả năng bảo mật (Security Test): bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.
- Kiểm tra khả năng phục hồi (Recovery Test): bảo đảm hệ thống có khả năng khôi phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến.

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

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



Hình 6.1: Mối tương quan giữa phát triển và kiểm tra phần mềm

4.4. Acceptance Test - Kiểm thử chấp nhận sản phẩm

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

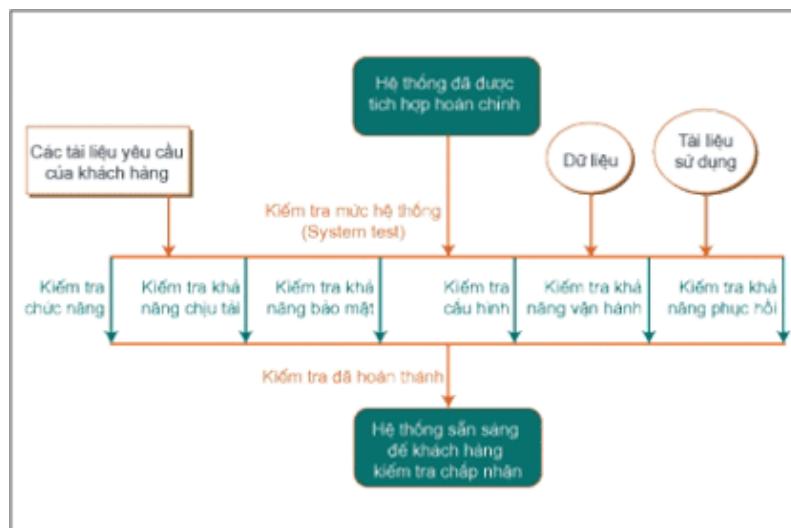
Acceptance Test có ý nghĩa hết sức quan trọng, mặc dù trong hầu hết mọi trường hợp, các phép kiểm tra của System Test và Acceptance Test gần như tương tự, nhưng bản chất và cách thức thực hiện lại rất khác biệt.

Đối với những sản phẩm dành bán rộng rãi trên thị trường cho nhiều người sử dụng, thông thường sẽ thông qua hai loại kiểm tra gọi là Alpha Test và Beta Test. Với Alpha Test, người sử dụng (tiềm năng) kiểm tra PM ngay tại nơi PTPM, lập trình viên sẽ nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa. Với Beta Test,

PM sẽ được gửi tới cho người sử dụng (tiềm năng) để kiểm tra ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

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

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



Hình 6.2: Các loại kiểm tra khác nhau trong System Test

4.5. Regression Test - Kiểm thử hồi quy

Regression Test không phải là một mức kiểm tra, như các mức khác đã nói ở trên.

Nó đơn thuần kiểm tra lại sau khi phần mềm có một sự thay đổi xảy ra, để bảo đảm 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 tra và thường sử dụng lại TC và bộ dữ liệu thử đã sử dụng trong các giai đoạn trước đó.

Ví dụ: một phần mềm đang phát triển khi kiểm tra cho thấy nó chạy tốt các chức năng A, B và C. Khi có thay đổi code của chức năng C, nếu chỉ kiểm tra chức năng C thì chưa đủ, cần phải kiểm tra lại tất cả các chức năng khác liên quan đến chức năng C, trong ví dụ này là A và B. Lý do là khi C thay đổi, nó có thể sẽ làm A và B không còn làm việc đúng nữa.

Mặc dù không là một mức kiểm tra, thực tế lại cho thấy Regression Test là một trong những loại kiểm tra tốn nhiều thời gian và công sức nhất. Tuy thế, việc bỏ qua Regression Test là "không được phép" vì có thể dẫn đến tình trạng phát sinh hoặc tái xuất hiện những lỗi nghiêm trọng, dù ta "tưởng rằng" những lỗi đó không có hoặc đã được kiểm tra và sửa chữa rồi! Với những dự án bảo trì, khởi động cho kiểm thử hồi quy phải được xác định trong kế hoạch kiểm thử. Người lãnh đạo quản lý kiểm thử phải xác định khi nào đội dự án kiểm soát kiểm thử hồi quy và phạm vi kiểm thử hồi quy. Lập biểu của kiểm thử hồi quy phải được xác định trong lập biểu của dự án.

TÓM TẮT

Trên đây là tổng quan về các mức và loại kiểm tra PM cơ bản. Thực tế nếu đi sâu vào từng mức và loại kiểm tra, còn có rất nhiều kiểm tra đặc thù khác nữa, mang tính chuyên biệt cho từng vấn đề hoặc từng loại ứng dụng. Tuy nhiên, những mức độ và loại kiểm tra nêu trên là cơ bản nhất và có thể áp dụng trong hầu hết các loại ứng dụng PM khác nhau.

BÀI 5. THIẾT KẾ TEST – TEST DESIGN

5.1. Tổng quan về test design

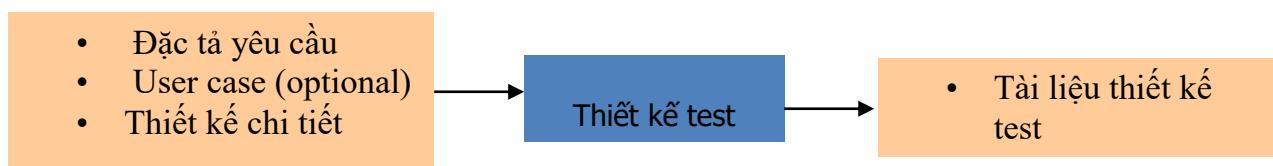
Là tài liệu được thiết kế nhằm liệt kê các tình huống có thể được kiểm tra. Nó được sử dụng nhằm xây dựng test case chi tiết về sau. Đôi khi nó giống như tài liệu HLD (High Level Design) nhưng lại được dùng cho việc test.

Test design được thực hiện trong giai đoạn solution của dự án SDLC (Software Development Life-Cycle)

Mục tiêu:

- * Để tester hiểu về yêu cầu sản phẩm từ đó tester hiểu sớm hơn về tìm lỗi phần mềm
- * Rà soát nhanh -> đảm bảo bao phủ sớm test case ở mức cao
- * Để phân chia thành nhiều test case phân công viết các test case cho các tester
- * Giảm chỉnh sửa test case và tạo các test case có chất lượng cao

Test design process:



5.2. Cấu trúc test design

Gồm 4 phần:

- ✓ Thông tin chung
- ✓ Xác định yêu cầu test
- ✓ Định nghĩa các test case
- ✓ Xác định các test suites

Thông tin chung

- * Trang bìa:

- * Thông tin dự án
- * Thông tin dòng thay đổi
- * Giới thiệu về tất cả các trang của tài liệu
- * Trang tài liệu tham khảo: liệt kê các tài liệu mà test design tham chiếu

Xác định các yêu cầu test:

- * Liệt kê các danh mục (các yêu cầu chức năng, phi chức năng) mà được xác định để kiểm thử
- * Danh sách đó sẽ được kiểm thử
- * Có rất nhiều yêu cầu sẽ được đánh số và quản lý
- * Function requirement:
 - * Là tất cả mọi thứ mà hệ thống phải làm:
 - * Để tính toán
 - * Để ra một quyết định
 - * Để tạo một thông báo
 - * Non-Function requirement:
 - * Chất lượng mà hệ thống cần phải có:
 - * Hiệu năng
 - * Bảo mật
 - * Tính khả dụng
 - * Khả năng bảo trì được
 - * Ràng buộc với qui trình phát triển PM
 - * Thông tin:
 - * Mã yêu cầu:
 - * R1: module
 - * R1.1: các chức năng thuộc về module R1
 - * 1.1.1: các hàm con hoặc nghiệp vụ (tùy chọn)

- * Tên yêu cầu
- * Tham chiếu yêu cầu: Tham chiếu tới tài liệu hoặc bất kỳ nguồn tài nguyên để lấy thông tin chi tiết các yêu cầu
- * Danh mục các yêu cầu:
 - * Functional
 - * Non-Functional

Xác định test case:

- * Liệt kê tất cả các test case có thể cho mỗi yêu cầu để kiểm thử:
 - * Dựa trên đặc tả yêu cầu
 - * Dựa trên thiết kế chi tiết
 - * Dựa trên kỹ thuật kiểm thử
- * Các test case được đánh số và quản lý
- * Thiết kế test case:
 - * Kiểm thử giao diện người dùng – GUI Test case
 - * Giao diện: size, position, menu, look and feel, field alignment
 - * Các điều khiển: buttons, checkbox, text box, list box, links, combo box
 - * Các phương thức truy cập/ sự kiện: tab keys, mouse movements, accelerator key
 - * Test nghiệp vụ - Function test case
 - * Xác định đầu vào
 - * Xác định quy tắc nghiệp vụ
 - * Luồng cơ bản: Dữ liệu hợp lệ
 - * Các luồng khác: hợp lệ cho các dữ liệu khác
 - * **Trường dữ liệu hợp lệ khi nhập dữ liệu**
 - * Kiểm tra tính bắt buộc nhập dữ liệu

- * Kiểm tra độ dài lớn nhất
 - * Trường nhập dữ liệu cho phép nhập các ký tự đặc biệt như name, địa chỉ email, password, nội dung...
 - * Các khoảng trống ở đầu hoặc cuối xâu sẽ được chuẩn hóa
- * **Trường dữ liệu hợp lệ khi tìm kiếm**
- * Dữ liệu nhạy cảm hoặc không
 - * Các kí hiệu trống ở đầu hoặc cuối xâu đã được chuẩn hóa
 - * Chú ý: nên cẩn thận bạn có thể bỏ qua kiểm thử nếu như hệ thống yêu cầu hỗ trợ Unicode
- * **Trường ngày tháng khi nhập dữ liệu**
- * Kiểm tra tính bắt buộc nhập dữ liệu
 - * Kiểm tra định dạng (inputted date, chuyển đổi kiểu dữ liệu)
 - * So sánh giá trị nhập vào với ngày hiện tại (if required)
 - * So sánh ngày dựa trên quy tắc nghiệp vụ
- * **Trường ngày tháng khi tìm kiếm dữ liệu:**
- * Kiểm tra định dạng ngày
 - * So sánh ngày đi và ngày đến
 - * Tìm kiếm trong các trường lưu trữ giá trị kiểu datetime
- Chú ý: Kết quả tìm kiếm là đúng/ bản ghi mới kiểm tra nếu định dạng ngày tháng là đúng và độc lập với thiết lập ngày tháng ở regional
- * **Hợp lệ dữ liệu là số khi nhập dữ liệu:**
- * Kiểm tra các trường phải nhập liệu
 - * Kiểm tra giá trị lớn nhất/nhỏ nhất

- * Kiểm tra số nguyên/số thực
- * Kiểm tra số âm/số dương
- * Kiểm tra chuyển đổi đanh dạng (kí hiệu số thực, ký hiệu nhóm số, biểu diễn số 0)
- * Hợp lệ dữ liệu là số khi tìm kiếm:
- * Hiển thị kí hiệu số thực, kí hiệu nhóm số, kí hiệu số 0 ở đầu

Xác định các test suites

- * Luồng test
 - * Xác định tiền điều kiện (pre-condition)
 - * Xác định phụ thuộc inter-case
 - * Xác định các test case thuộc bộ

5.3. Ví dụ về test design

Mẫu trang bìa – thông tin chung

TEST DESIGN					
Project Name	Tên dự án	Project Code	Mã dự án	Version	1
Creator	Người tạo bản test design	Position	Tester	Date	
Reviewer	Tên Test Leader	Position	Test Leader	Date	
Approver	tên PM	Position	PM	Date	
Record of change					
Effective Date	Version	Changed Sheet	*A, D, M	Change description	Author
30/03/2012	1.0	Cover	A	Thay đổi thông tin trang bìa	HuongLTT
Introduction					
Sheet Name		Description		Note	
Cover		Title page and Introduction - Trang bìa		This page	

<u>Reference</u>	List of reference documents that the Test design is based on. - Liệt kê các tài liệu mà TD sẽ thiết kế dựa trên nó	
<u>Requirement for Test</u>	List all requirement for test (both functional requirement and non-functional requirement) - Liệt kê các yêu cầu cho việc kiểm thử (bao gồm cả yêu cầu chức năng và phi chức năng)	
<u>List of TC</u>	List of test cases- Liệt kê các tình huống kiểm thử	
<u>Test Suites</u>	List of test suites - Liệt kê các bộ kiểm thử	

Mẫu trang tham khảo

Reference Document

#	Title/File name	Author	Version	Effective Date	Note
1	SRS	Kỹ sư phân tích	V1.0	yyyy/mm/dd	Bản đặc tả yêu cầu chức năng
2	User case				Đặc tả ca sử dụng
3	LLD	Designer	V1.0		Bản thiết kế chi tiết
4	MS Excel	Microsoft			Dùng để thiết kế tài liệu TD
5					

Mẫu trang tài liệu thiết kế test

Requirement			Requirement reference	Requirement Category	Test design status
Req. ID		Requirement Name			
R1			SRS....	Function	Yes
	R1.1			Non - function	No
		1.1.1			
		1.2			
2					
	2.1				
	2.2				
	2.3				
3					
	3.1				

	3.2					
4						

Mẫu liệt kê các test case

TC ID	TC Name	Priority	Test type	Note
TC001				
TC002				
TC003				
TC004				

Mẫu liệt kê các test suite

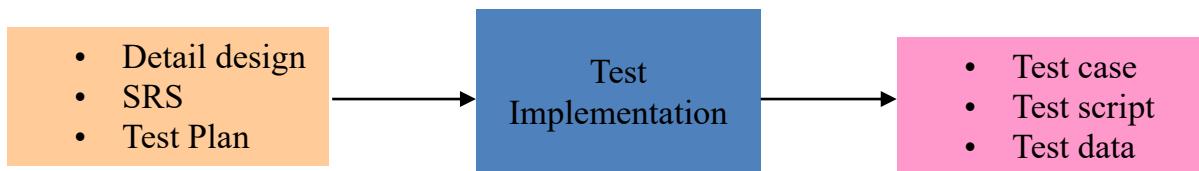
Test Suite ID	Test Suite Description	TC Name	Note
TS_001		TC001 TC003 TC004	Precondition: ...

5.4. Test Case

5.4.1. Khái niệm test case

Là một tập hợp gồm đầu vào, điều kiện thực thi và kết quả mong đợi xây dựng trong mục tiêu cụ thể để kiểm chứng theo các yêu cầu khách hàng

Quy trình tạo test case



Trong đó:

- ✓ Test Script: các kịch bản được sử dụng cho việc kiểm thử hoặc kiểm tra kết quả đầu ra cho việc kiểm thử tự động
- ✓ Test Data: Dữ liệu được sử dụng cho quá trình kiểm thử

Tại sao cần thiết kế test case:

- ✓ Bản test design chi tiết mô tả cách triển khai thiết kế test
- ✓ Dự đoán kết quả mong đợi
- ✓ Giúp các thành viên tester mới có thể làm quen với hệ thống/ ứng dụng đã tồn tại mà không cần đọc yêu cầu

5.4.2. Những tiêu chí tạo nên test case tốt

- ✓ Giúp tìm ra nhiều lỗi phần mềm
- ✓ Mục tiêu TC rõ ràng
- ✓ Cách bố trí TC tốt
- ✓ Có khả năng rà soát
- ✓ Có khả năng bảo trì và nâng cấp TC
- ✓ Hữu ích cho các tester khác khi họ không thiết kế

5.4.3 Cấu trúc Test case

Gồm các phần:

- Thông tin chung: cover sheet
 - Thông tin dự án
 - Thông tin thay đổi
- GUI: Liệt kê tất cả các cửa sổ giao diện để dễ dàng kiểm tra
- [Module Name] test cases

- Pictures: liệt kê tất cả các màn hình thiết kế mà sẽ được kiểm thử. Các màn hình đó phải được chấp nhận bởi khách hàng.
- Test Report

Trong đó:

Thông tin chung: mô tả chung về dự án và tài liệu test case

GUI test case:

- Screen Name
- Field Name
- Expected Output:
 - Type
 - Mandatory
 - Editable
 - Default value
 - Max Length
 - Range Value
- Test Status
- Test Date
- Notes

[Module] test cases:

- ID
- Mô tả Test case :
- Liệt kê các tình huống/kịch bản sẽ được kiểm thử
- Liệt kê các tình huống, thiết kế trong tài liệu thiết kế test cho các hàm/mô đun xác định
- Các thủ tục Test case :
 - Các hoạt động được thực hiện khi thực thi test
 - Test input: dữ liệu thực tế được test tại mỗi bước

- Expected output: kết quả mong đợi từ ứng dụng khi thực hiện qua mỗi bước
- Inter-test case dependence: Liệt kê tất cả các test case được test trước hiệu khi thực thi tình huống này.
- Actual output: kết quả thực tế test được từ ứng dụng khi thực thi các bước
- Result: Pass; Fail; Untested; N/A
- Note

Test script:

- Kiểm tra thông tin đầu ra test
- Ghi tự động
- Coding: Sử dụng các công cụ test hoặc các ngôn ngữ lập trình chuẩn như VB, C/C++, Java hoặc SQL
- Test stub: temporary implementation of part of a program for unit test purposes
- Test driver: program which sets up an environment to call a module (or function) for testing

Các biểu mẫu của test case:

Mẫu thông tin chung :

TEST CASE

Project Name		Author	Tester
Project Code		Reviewer/Approver	TL
Document Name		Issue Date	
		Version	

Record of change:

Effective Date	Version	Change Item	*A,D, M	Change description	Reference
----------------	---------	-------------	---------	--------------------	-----------

Mẫu GUI – test case

Screen Name	Field Name	Expected result						Test Status	Test Date	Note
		Type	Madatory	Editable	Default value	Max Length	Range/ Value			
Module Name										
	Precondition/Context:									
	[Dat ve may bay]									
[Screen Name]										
GUI_001	Name	Textbox	N	Y	N			P		
GUI_002	Address	Textbox	N	Y	N			P		
GUI_003	ID card No	Textbox	N	Y	N			P		
GUI_004	Date of Birth	Textbox	N	Y	N			P		
GUI_005	Tyte of class	Combobox	N	Y	N		Bussiness/ Economy	P		
GUI_006	Payment	Textbox	N	N	N			P		
GUI_007	Save	Button						P		
GUI_008	Exit	Button						P		
Common case										
	Note: Common cases apply for all new/edit/display pages of all modules.									
Audit trail										

Mẫu module test case:

TEST CASE

Module Code	<i>Subject Management</i>					Pass Fail Untested N/A	
Test requirement							
Tester							
Pass	Fail	Untested	N/A	Number of Test cases			
0	0	9	0	9			

ID	Test Case Description	Test Case Procedure	Expected Output	Inter-test case Dependence	Actual Output	Result	Note
Subject Management							
[Subject Management-1]	Aaa						
[Subject Management-2]	Bbb		-	-			
[Subject Management-3]	Ccc		-	-			

5.4.5. Ví dụ test case

Ví dụ 1:

- If an internal telephone system for a company with 200 telephones has 3-digit extension numbers from 100 to 699, we can identify the following partitions and boundaries

Ví dụ 2: Xác định các test case cho bài toán sau?

Figure 1: Calculate Payment application

Payment is calculated based on age and type of the patient. The result will be displayed on the screen.

Male

Age	Payment
18 - 35	100 euro
36 - 50	120 euro
51 - 145	140 euro

Female (* Payments in this example are not same than in a real world)

Age	Payment
18 - 30	80 euro
31 - 50	110 euro
51-145	140 euro

Child

Age	Payment
0 - 17	50 euro

BÀI 6. KỸ THUẬT KIỂM THỬ HỘP TRẮNG (WHITE BOX TESTING) (I)

6.1. Tổng quan về kiểm thử hộp trắng

Kiểm thử hộp trắng là một kỹ thuật kiểm thử, mà kiểm tra cấu trúc chương trình và lấy dữ liệu kiểm thử từ mã chương trình. Kiểm thử hộp trắng còn có các tên khác glass box testing, open box testing, logic driven testing hoặc path driven testing hoặc structural testing.

Đối tượng được kiểm thử là 1 thành phần phần mềm (TPPM). TPPM có thể là 1 hàm chức năng, 1 module chức năng, 1 phân hệ chức năng...

Kiểm thử hộp trắng dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu bên trong của đơn vị phần mềm cần kiểm thử để xác định đơn vị phần mềm đó có thực hiện đúng không.

Do đó người kiểm thử hộp trắng phải có kỹ năng, kiến thức nhất định về ngôn ngữ lập trình được dùng, về thuật giải được dùng trong TPPM để có thể thông hiểu chi tiết về đoạn code cần kiểm thử.

Thường tốn rất nhiều thời gian và công sức nếu TPPM quá lớn (thí dụ trong kiểm thử tích hợp hay kiểm thử chức năng).

Do đó kỹ thuật này chủ yếu được dùng để kiểm thử đơn vị.

Trong lập trình hướng đối tượng, kiểm thử đơn vị là kiểm thử từng tác vụ của 1 class chức năng nào đó.

Có 2 kỹ thuật sử dụng trong kiểm thử hộp trắng :

- Kiểm thử luồng điều khiển: tập trung kiểm thử thuật giải chức năng.
- Kiểm thử dòng dữ liệu: tập trung kiểm thử đời sống của từng biến dữ liệu được dùng trong thuật giải.

6.2. Kiểm thử dựa trên luồng điều khiển (Control Flow)

6.2.1. Đường thi hành (Execution path)

Là 1 kịch bản thi hành đơn vị phần mềm tương ứng, cụ thể nó là danh sách có thứ tự các lệnh được thi hành ứng với 1 lần chạy cụ thể của đơn vị phần mềm, bắt đầu từ điểm nhập của đơn vị phần mềm đến điểm kết thúc của đơn vị phần mềm.

Mỗi TPPM có từ 1 đến n (có thể rất lớn) đường thi hành khác nhau. Mục tiêu của phương pháp kiểm thử luồng điều khiển là đảm bảo mọi đường thi hành của đơn vị phần mềm cần kiểm thử đều chạy đúng. Rất tiếc trong thực tế, công sức và thời gian để đạt mục tiêu trên đây là rất lớn, ngay cả trên những đơn vị phần mềm nhỏ.

Thí dụ đoạn code sau :

```
for (i=1; i<=1000; i++)
    for (j=1; j<=1000; j++)
        for (k=1; k<=1000; k++)
            doSomethingWith(i,j,k);
```

chỉ có 1 đường thi hành, nhưng rất dài : dài $1000 \times 1000 \times 1000 = 1$ tỉ lệnh gọi hàm doSomething(i,j,k) khác nhau.

Còn đoạn code gồm 32 lệnh if else độc lập sau :

```
if (c1) s11 else s12;
if (c2) s21 else s22;
if (c3) s31 else s32; ...
if (c32) s321 else s322;
```

có $2^{32} = 4$ tỉ đường thi hành khác nhau.

Mà cho dù có kiểm thử hết được toàn bộ các đường thi hành thì vẫn không thể phát hiện những đường thi hành cần có nhưng không (chưa) được hiện thực :

```
if (a>0) doIsGreater();
if (a==0) doIsEqual();
// thiếu việc xử lý trường hợp a < 0 - if (a<0) doIsLess();
```

Một đường thi hành đã kiểm tra là đúng nhưng vẫn có thể bị lỗi khi dùng thật (trong 1 vài trường hợp đặc biệt) :

```
int phanso (int a, int b)
```

{ return a/b; }

khi kiểm tra, ta chọn b <> 0 thì chạy đúng, nhưng khi dùng thật trong trường hợp b = 0 thì hàm phanso bị lỗi.

6.2.2. Các cấp phủ kiểm thử (Coverage)

Như đã trình bày ở trên, chúng ta không thể test thử hết tất cả các trường hợp thực thi của chương trình, ta nên kiểm thử 1 số test case tối thiểu mà kết quả độ tin cậy tối đa. Nhưng làm sao xác định được số test case tối thiểu nào có thể đem lại kết quả có độ tin cậy tối đa ?

Phủ kiểm thử (Coverage) : là tỉ lệ các thành phần thực sự được kiểm thử so với tổng thể sau khi đã kiểm thử các test case được chọn. Phủ càng lớn thì độ tin cậy càng cao.

Thành phần liên quan có thể là lệnh thực thi, điểm quyết định, điều kiện con hay là sự kết hợp của chúng.

- ❖ **Phủ cấp 0:** kiểm thử những gì có thể kiểm thử được, phần còn lại để người dùng phát hiện và báo lại sau. Đây là mức độ kiểm thử không thực sự có trách nhiệm.
- ❖ **Phủ cấp 1 (Statement Coverage):** kiểm thử sao cho mỗi một câu lệnh được thực hiện ít nhất một lần

Phân tích hàm foo sau đây :

```
1 float foo(int a, int b, int c, int d) {  
2     float e;  
3     if (a==0)  
4         return 0;  
5     int x = 0;  
6     if ((a==b) || ((c==d) && bug(a)))  
7         x = 1;  
8     e = 1/x;  
9     return e;  
10 }
```

Với hàm foo trên, ta chỉ cần 2 test case sau đây là đạt 100% phủ cấp 1 :

2. foo(1,1,1,1), trả về 1

1. foo(0,0,0,0), trả về 0

nhưng không phát hiện lỗi chia 0 ở hàng lệnh 8.

❖ **Phủ cấp 2 (Branch coverage)** : kiểm thử sao cho mỗi điểm quyết định luận lý đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các nhánh (Branch coverage). Phủ các nhánh đảm bảo phủ các lệnh.

Line	Predicate	True	False
3	(a == 0)	Test Case 1 foo(0, 0, 0, 0) return 0	Test Case 2 foo(1, 1, 1, 1) return 1
6	((a==b) OR ((c == d) AND bug(a)))	Test Case 2 foo(1, 1, 1, 1) return 1	Test Case 3 foo(1, 2, 1, 2) division by zero!

Với 2 test case xác định trong slide trước, ta chỉ đạt được $3/4 = 75\%$ phủ các nhánh. Nếu thêm test case 3 : 3. foo(1,2,1,2), thì mới đạt 100% phủ các nhánh.

❖ **Phủ cấp 3 (subcondition coverage)**: kiểm thử sao cho mỗi điều kiện luận lý con (subcondition) của từng điểm quyết định đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các điều kiện con (subcondition coverage). Phủ các điều kiện con chưa chắc đảm bảo phủ các nhánh & ngược lại.

Predicate	True	False
$a == 0$	TC 1 : foo(0, 0, 0, 0) return 0	TC 2 : foo(1, 1, 1, 1) return 1
$(a == b)$	TC 2 : foo(1, 1, 1, 1) return value 0	TC 3 : foo(1, 2, 1, 2) division by zero!
$(c == d)$	TC 4 : foo(1, 2, 1, 1) Return 1	TC 3 : foo(1, 2, 1, 2) division by zero!

❖ **Phủ cấp 4 :** kiểm thử sao cho mỗi điều kiện luận lý con (subcondition) của từng điểm quyết định đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE & điểm quyết định cũng được kiểm thử cho cả 2 nhánh TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các nhánh & các điều kiện con (branch & subcondition coverage). Đây là mức độ phủ kiểm thử tốt nhất trong thực tế. Phần còn lại của chương này sẽ giới thiệu qui trình kỹ thuật để định nghĩa các testcase sao cho nếu kiểm thử hết các testcase được định nghĩa này, ta sẽ đạt phủ kiểm thử cấp 4.

6.2.3. Đồ thị luồng điều khiển (Control Flow Graph)

Là một đồ thị có hướng, biểu diễn một chương trình, đồ thị gồm có các thành phần

- + Đỉnh (node): là một lệnh hoặc một khối lệnh
- + Cung (edge): biểu diễn đường đi giữa các đỉnh
- + Một đỉnh vào và một đỉnh ra được thêm vào để biểu diễn điểm vào và điểm ra của chương trình.

Các loại nút trong đồ thị dòng điều khiển :

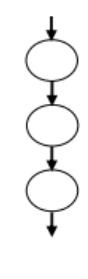


Các cấu trúc điều khiển

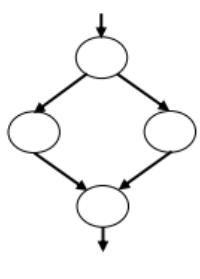
```

1. float
2. flo
3. if (
4. i
5. int
6. if (
7. x
8. e =
9. rel
10.}

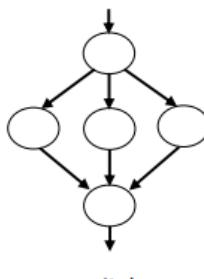
```



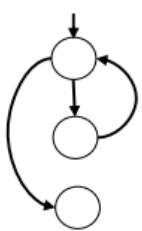
tuần tự



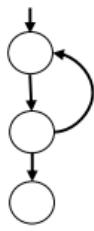
If



switch



while c do...



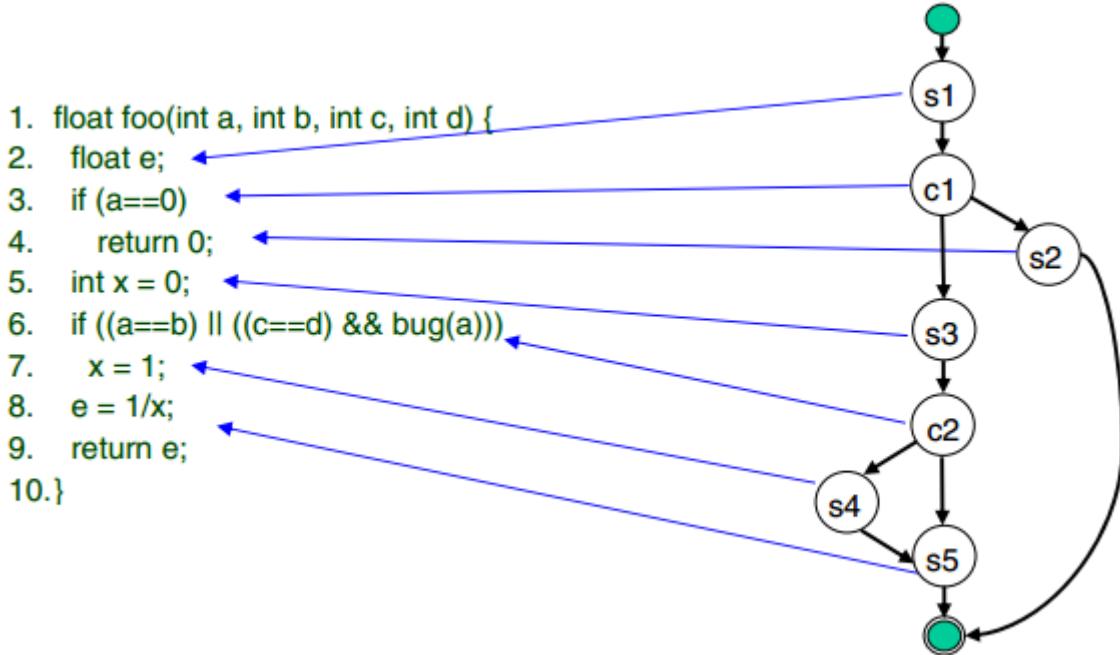
do ... while c

Ví dụ:

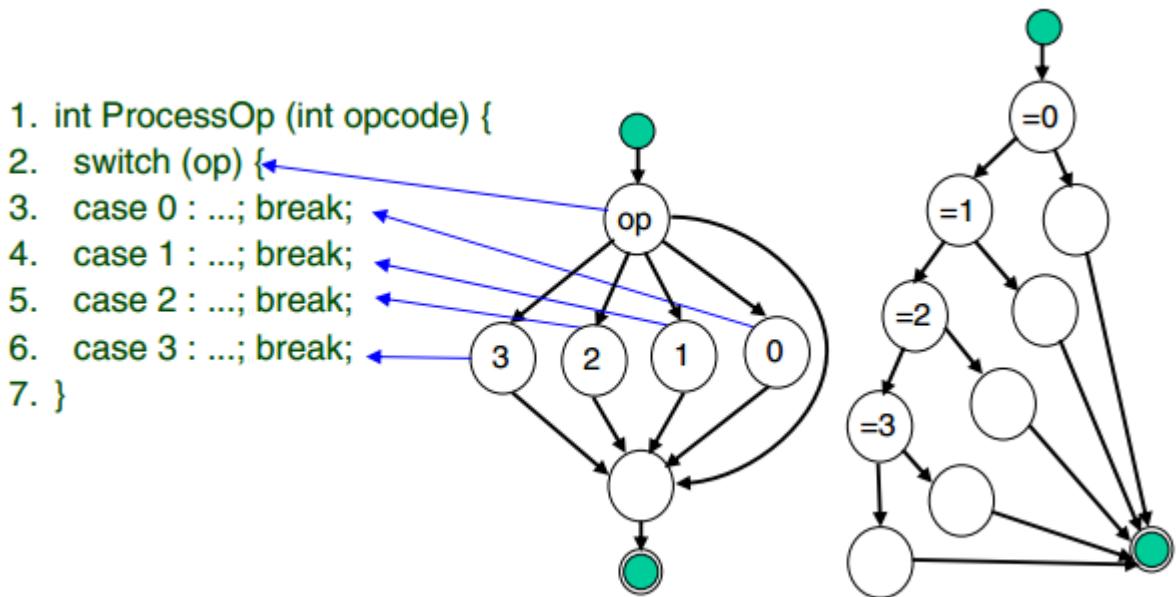
```

1. float foo(int a, int b, int c, int d) {
2.     float e;
3.     if (a==0)
4.         return 0;
5.     int x = 0;
6.     if ((a==b) || ((c==d) && bug(a)))
7.         x = 1;
8.     e = 1/x;
9.     return e;
10.}

```



Nếu đồ thị dòng điều khiển chỉ chứa các nút quyết định nhị phân thì ta gọi nó là đồ thị dòng điều khiển nhị phân. Ta luôn có thể chi tiết hóa 1 đồ thị dòng điều khiển bất kỳ



thành đồ thị dòng điều khiển nhị phân.

Độ phức tạp Cyclomatic C Độ phức tạp Cyclomatic C = $V(G)$ của đồ thị dòng điều khiển được tính bởi 1 trong các công thức sau :

$$f \quad V(G) = E - N + 2, \text{ trong đó } E \text{ là số cung, } N \text{ là số nút của đồ thị.}$$

- $V(G) = P + 1$, nếu là đồ thị dòng điều khiển nhị phân (chỉ chứa các nút quyết định luận lý - chỉ có 2 cung xuất True/False) và P số nút quyết định.

Độ phức tạp Cyclomatic C chính là số đường thi hành tuyến tính độc lập của TPPM cần kiểm thử.

Nếu chúng ta chọn lựa được đúng C đường thi hành tuyến tính độc lập của TPPM cần kiểm thử và kiểm thử tất cả các đường thi hành này thì sẽ đạt được phủ kiểm thử cấp 3 như đã trình bày ở trên.

6.2.4. Đồ thị dòng điều khiển cơ bản

Xét đồ thị dòng điều khiển nhị phân: nếu từng nút quyết định (nhị phân) đều miêu tả 1 điều kiện con luận lý thì ta nói đồ thị này là đồ thi dòng điều khiển cơ bản.

Ta luôn có thể chiết hóa 1 đồ thị dòng điều khiển bất kỳ thành đồ thị dòng điều khiển nhị phân. Tương tự, ta luôn có thể chiết hóa 1 đồ thị dòng điều khiển nhị phân bất kỳ thành đồ thị dòng điều khiển cơ bản.

Tóm lại, ta luôn có thể chiết hóa 1 đồ thị dòng điều khiển bất kỳ thành đồ thị dòng điều khiển cơ bản.

Độ phức tạp Cyclomatic C của đồ thị dòng điều khiển cơ bản chính là số đường thi hành tuyến tính độc lập cơ bản của TPPM cần kiểm thử.

Nếu chúng ta chọn lựa được đúng C đường thi hành tuyến tính độc lập cơ bản của TPPM cần kiểm thử và kiểm thử tất cả các đường thi hành này thì sẽ đạt được phủ kiểm thử cấp 4 như đã trình bày trong các slide trước.

6.2.5. Qui trình kiểm thử hộp trắng

Tom McCabe đề nghị qui trình kiểm thử TPPM gồm các bước công việc sau :

1. Từ TPPM cần kiểm thử, xây dựng đồ thị dòng điều khiển tương ứng, rồi chuyển thành đồ thị dòng điều khiển nhị phân, rồi chuyển thành đồ thị dòng điều khiển cơ bản.
2. Tính độ phức tạp Cyclomatic của đồ thị($C = P + 1$).
3. Xác định C đường thi hành tuyến tính độc lập cơ bản cần kiểm thử(theo thuật giải chi tiết ở phần kế tiếp).
4. Tạo từng test case cho từng đường thi hành tuyến tính độc lập cơ bản.
5. Thực hiện kiểm thử trên từng test case.
6. So sánh kết quả có được với kết quả được kỳ vọng.
7. Lập báo cáo kết quả để phản hồi cho những người có liên quan

a) Qui trình xác định các đường tuyến tính độc lập

Tom McCabe đề nghị qui trình xác định C đường tuyến tính độc lập gồm các bước :

1. Xác định đường tuyến tính đầu tiên bằng cách đi dọc theo nhánh bên trái nhất của các nút quyết định. Chọn đường

này là pilot.

2. Dựa trên đường pilot, thay đổi cung xuất của nút quyết định đầu tiên và cố gắng giữ lại maximum phần còn lại.

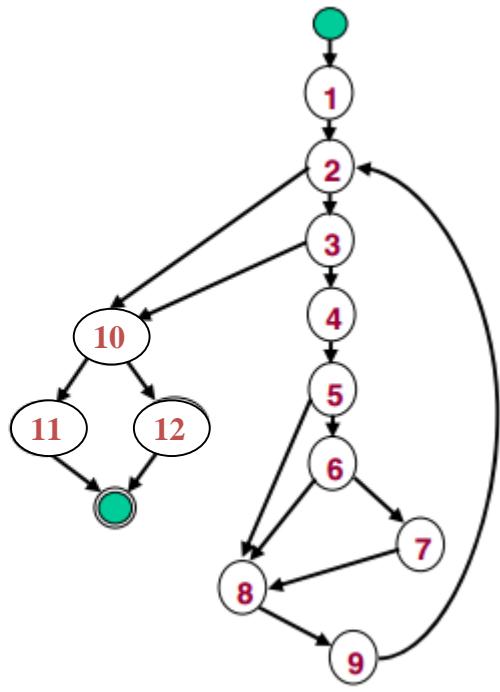
3. Dựa trên đường pilot, thay đổi cung xuất của nút quyết định thứ 2 và cố gắng giữ lại maximum phần còn lại.

4. Tiếp tục thay đổi cung xuất cho từng nút quyết định trên đường pilot để xác định đường thứ 4, 5,... cho đến khi không còn nút quyết định nào trong đường pilot nữa.

5. Lặp chọn tuần tự từng đường tìm được làm pilot để xác định các đường mới xung quanh nó y như các bước 2, 3, 4 cho đến khi không tìm được đường tuyén tính độc lập nào nữa (khi đủ số C).

Ví dụ:

```
double average(double value[], double min,
              double max, int& tcnt, int& vcnt) {
    double sum = 0;
    int i = 1;
    tcnt = vcnt = 0;
    while (value[i] > -999 && tcnt < 100) {
        tcnt++;
        if (min <= value[i] && value[i] <= max) {
            sum += value[i];
            vcnt++;
        }
        i++;
    }
    if (vcnt > 0) return sum/vcnt;
    return -999;
}
```



Đồ thị trên có 5 nút quyết định nhị phân nên có độ phức tạp $C = 5+1 = 6$.

6 đường thi hành tuyến tính độc lập cơ bản là :

1. $1 \rightarrow 2 \rightarrow 10 \rightarrow 11$

2. $1 \rightarrow 2 \rightarrow 3 \rightarrow 10 \rightarrow 11$
3. $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 9$
4. $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9$
5. $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$
6. $1 \rightarrow 2 \rightarrow 10 \rightarrow 12$

b) Thiết kế các test case

Phân tích mã nguồn của hàm average, ta định nghĩa 6 testcase kết hợp với 6 đường thi hành tuyến tính độc lập cơ bản như sau :

- Test case cho đường 1:

$\text{value}(k) <> -999$, với $1 < k < i$ $\text{value}(i) = -999$ với $2 \leq i \leq 100$

Kết quả mong đợi: (1) $\text{average} = \text{Giá trị trung bình của } i-1$ giá trị hợp lệ.

$$(2) \text{tcnt} = i-1. \quad (3) \text{vcnt} = i-1$$

Chú ý : không thể kiểm thử đường 1 này riêng biệt mà phải kiểm thử chung với đường 4 hay 5 hay 6.

- Test case cho đường 2 :

$\text{value}(k) <> -999$, với $\forall k < i, i > 100$

Kết quả mong đợi : (1) $\text{average} = \text{Giá trị trung bình của } 100$ giá trị hợp lệ.

$$(2) \text{tcnt} = 100. \quad (3) \text{vcnt} = 100$$

- Test case cho đường 3 :

$\text{value}(1) = -999$

Kết quả mong đợi : (1) $\text{average} = -999$. (2) $\text{tcnt} = 0$ (3) $\text{vcnt} = 0$

- Test case cho đường 4 :

$\text{value}(i) <> -999 \quad \forall i \leq 100$

và $\text{value}(k) < \text{min}$ với $k < i$

Kết quả mong đợi : (1) $\text{average} = \text{Giá trị trung bình của } n$ giá trị hợp lệ.

$$(2) \text{tcnt} = 100. \quad (3) \text{vcnt} = n \text{ (số lượng giá trị hợp lệ)}$$

- Test case cho đường 5 :

$\text{value}(i) <> -999$ với $\forall i \leq 100$

và $\text{value}(k) > \text{max}$ với $k \leq i$

Kết quả kỳ vọng : (1) average=Giá trị trung bình của n giá trị hợp lệ.

(2) tcnt = 100. (3) vcnt = n (số lượng giá trị hợp lệ)

- Test case cho đường 6 :

$\text{value}(i) <> -999$ và $\min \leq \text{value}(i) \leq \max$ với $\forall i \leq 100$

Kết quả kỳ vọng : (1) average=Giá trị trung bình của 100 giá trị hợp lệ.

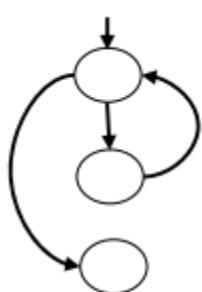
(2) tcnt = 100. (3) vcnt = 100

c) Kiểm thử vòng lặp

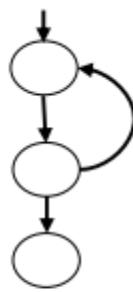
Thường thân của 1 lệnh lặp sẽ được thực hiện nhiều lần (có thể rất lớn). Chi phí kiểm thử đầy đủ rất tốn kém, nên chúng ta sẽ chỉ kiểm thử ở những lần lặp mà theo thống kê dễ gây lỗi nhất. Ta xét từng loại lệnh lặp, có 4 loại :

1. lệnh lặp đơn giản : thân của nó chỉ chứa các lệnh khác chứ không chứa lệnh lặp khác.
2. lệnh lặp lồng nhau : thân của nó có chứa ít nhất lệnh lặp khác...
3. lệnh lặp liền kề: 2 hay nhiều lệnh lặp kế tiếp nhau
4. lệnh lặp giao nhau : 2 hay nhiều lệnh lặp giao nhau.

1- Kiểm thử loại vòng lặp n lần đơn giản:



`while c do...`



`do ... while c`

Nên chọn các test case để kiểm thử thân lệnh lặp ở các vị trí sau :

f - chạy 0 bước.

- chạy 1 bước.

f - chạy 2 bước.

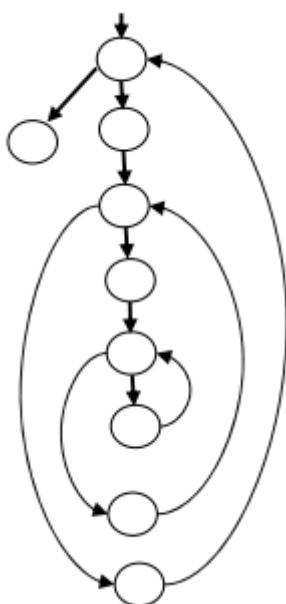
f - chạy k bước, k là giá trị nào đó thỏa $2 < k < n-1$.

f - chạy n-1 bước

f - chạy n bước

- chạy n+1 bước.

2- Kiểm thử vòng lặp chồng nhau :



Kiểm thử tuần tự từng vòng lặp từ trong ra ngoài theo đề nghị sau đây :

f - Kiểm thử vòng lặp trong cùng : cho các vòng ngoài chạy với giá trị min, kiểm thử vòng lặp trong cùng bằng 7 test case đã giới thiệu ở trên.

f - Kiểm thử từng vòng lặp còn lại : cho các vòng ngoài nó chạy với giá trị min, còn các vòng bên trong nó chạy với giá trị điển hình, kiểm thử nó bằng 7 test case đã giới thiệu ở trên.

3- Kiểm thử các vòng lặp liền kề:

Kiểm thử tuần tự từng vòng lặp từ trên xuống, mỗi vòng thực hiện kiểm thử bằng 7 test case đã giới thiệu.

4- Riêng các vòng lặp giao nhau: thì thường do việc viết code chưa tốt tạo ra ⇒nên cấu trúc lại đoạn code sao cho không chúa dạng giao nhau này.

BÀI 7. KỸ THUẬT KIỂM THỬ HỘP TRẮNG (II)

7.1. Kiểm thử luồng dữ liệu (Data Flow)

Phương pháp kiểm thử dòng dữ liệu sẽ kiểm thử đời sống của từng biến dữ liệu có "tốt lành" trong từng luồng thi hành của chương trình.

Phương pháp kiểm thử dòng dữ liệu là 1 công cụ mạnh để phát hiện việc dùng không hợp lý các biến do lỗi coding phần mềm gây ra :

- Phát biểu gán hay nhập dữ liệu vào biến không đúng.
- Thiếu định nghĩa biến trước khi dùng
- Tiên đề sai (do thi hành sai luồng thi hành).
- ...

Mỗi biến nên có chu kỳ sống tốt lành thông qua trình tự 3 bước : được tạo ra, được dùng và được xóa đi.

Chỉ có những lệnh nằm trong tầm vực truy xuất biến mới có thể truy xuất/xử lý được biến. Tầm vực truy xuất biến là tập các lệnh được phép truy xuất biến đó.

Thường các ngôn ngữ lập trình cho phép định nghĩa tầm vực cho mỗi biến thuộc 1 trong 3 mức chính yếu : toàn cục, cục bộ trong từng module, cục bộ trong từng hàm chức năng.

```
int x, y;  
void func1() { //thân hàm  
    int x; // định nghĩa biến x mới cục bộ trong hàm  
    ...; // mỗi lần truy xuất x là x cục bộ trong hàm  
    { // khôi lệnh bên trong bắt đầu  
        int y; // định nghĩa biến y mới cục bộ trong lệnh phức hợp  
        ...; // mỗi lần truy xuất y là y cục bộ trong lệnh phức hợp  
    } // y bên trong tự động bị xóa  
    ...; // truy xuất y ngoài cùng, x cục bộ trong hàm  
} // x cục bộ trong hàm bị xóa tự động
```

7.2. Phân tích đời sống của một biến

Các lệnh truy xuất 1 biến thông qua 1 trong 3 hành động sau :

- d(*definition*) : định nghĩa biến, gán giá trị xác định cho biến (nhập dữ liệu vào biến cũng là hoạt động gán trị cho biến).
- u(*use*) : tham khảo trị của biến (thường thông qua biểu thức).
- k : hủy (xóa bỏ) biến đi.

Như vậy nếu ký hiệu ~ là miêu tả trạng thái mà ở đó biến chưa tồn tại, ta có 3 khả năng xử lý đầu tiên trên 1 biến :

- ~d : biến chưa tồn tại rồi được định nghĩa với giá trị xác định.
- ~u : biến chưa tồn tại rồi được dùng ngay (trị nào ?)
- ~k : biến chưa tồn tại rồi bị hủy (lạ lùng).

3 hoạt động xử lý biến khác nhau kết hợp lại tạo ra 9 cặp đôi hoạt động xử lý biến theo thứ tự:

- dd : biến được định nghĩa rồi định nghĩa nữa : hơi lạ, có thể đúng và chấp nhận được, nhưng cũng có thể có lỗi lập trình.
- du : biến được định nghĩa rồi được dùng : trình tự đúng và bình thường.
- dk : biến được định nghĩa rồi bị xóa bỏ: hơi lạ, có thể đúng và chấp nhận được, nhưng cũng có thể có lỗi lập trình.
- ud : biến được dùng rồi định nghĩa giá trị mới : hợp lý.
- uu : biến được dùng rồi dùng tiếp : hợp lý.
- uk : biến được dùng rồi bị hủy : hợp lý.
- kd : biến bị xóa bỏ rồi được định nghĩa lại : chấp nhận được.
- ku : biến bị xóa bỏ rồi được dùng : đây luôn là lỗi.
- kk : biến bị xóa bỏ rồi bị xóa nữa : có lẽ là lỗi lập trình.

 Biến X tại câu lệnh S được cho là vẫn còn sống tại câu lệnh S' nếu như tồn tại một đường từ câu lệnh S đến câu lệnh S' không chứa bất kỳ định nghĩa nào của X.

⊕ DU pair (Definition-Use pair): Một cặp định nghĩa – sử dụng (DU pair) đối với biến X, ký hiệu $DU=[X, S, S']$ với X trong $DEF(S)$, $USE(S')$, và định nghĩa X trong câu lệnh S vẫn sống trong câu lệnh S' .

⊕ DU path: Là một chuỗi các lệnh $S \rightarrow S'$, trong đó $DEF(S)=USE(S')=\{X\}$

⊕ DC path (Definition-Clear path): Là một chuỗi DU path, và chỉ duy nhất S có $DES(S)=\{X\}$

7.3. Đồ thị dòng dữ liệu

Là một trong nhiều phương pháp miêu tả các kịch bản đổi sống khác nhau của các biến. Qui trình xây dựng đồ thị dòng dữ liệu dựa trên qui trình xây dựng đồ thị dòng điều khiển của TPPM cần kiểm thử.

Ta cũng dùng độ phức tạp Cyclomatic $C = V(G)$ của đồ thị dòng điều khiển của TPPM cần kiểm thử để xác định số đường thi hành tuyến tính độc lập của TPPM cần kiểm thử.

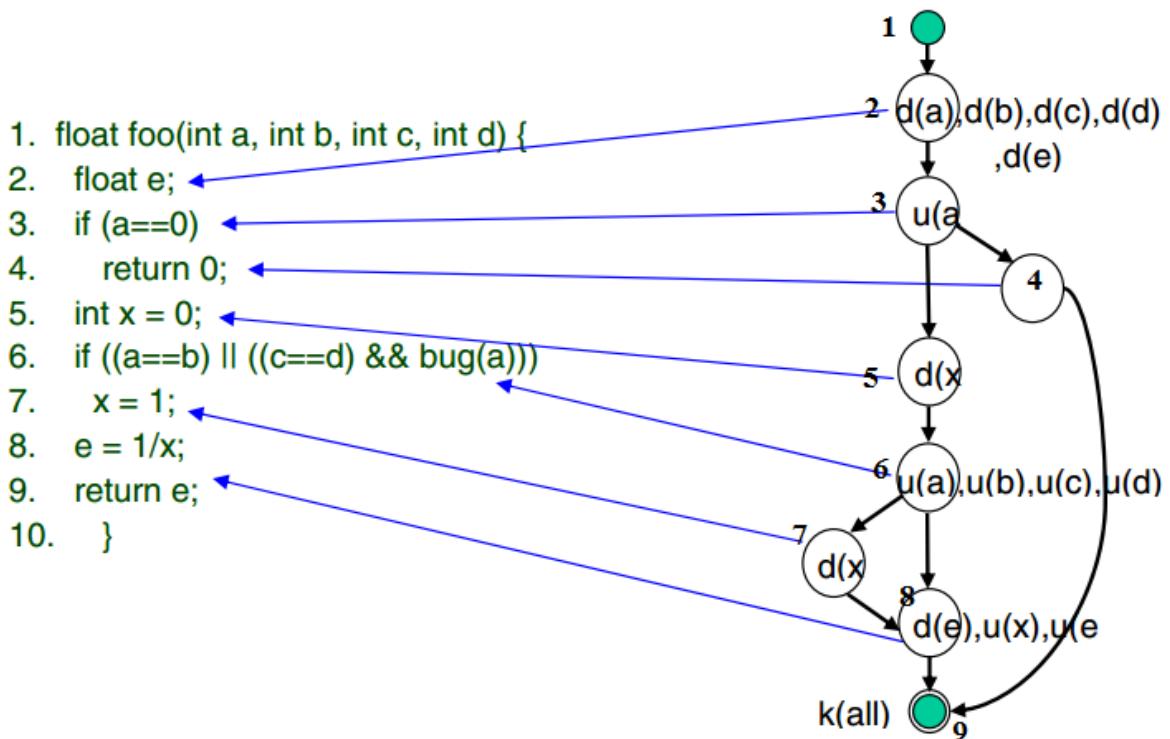
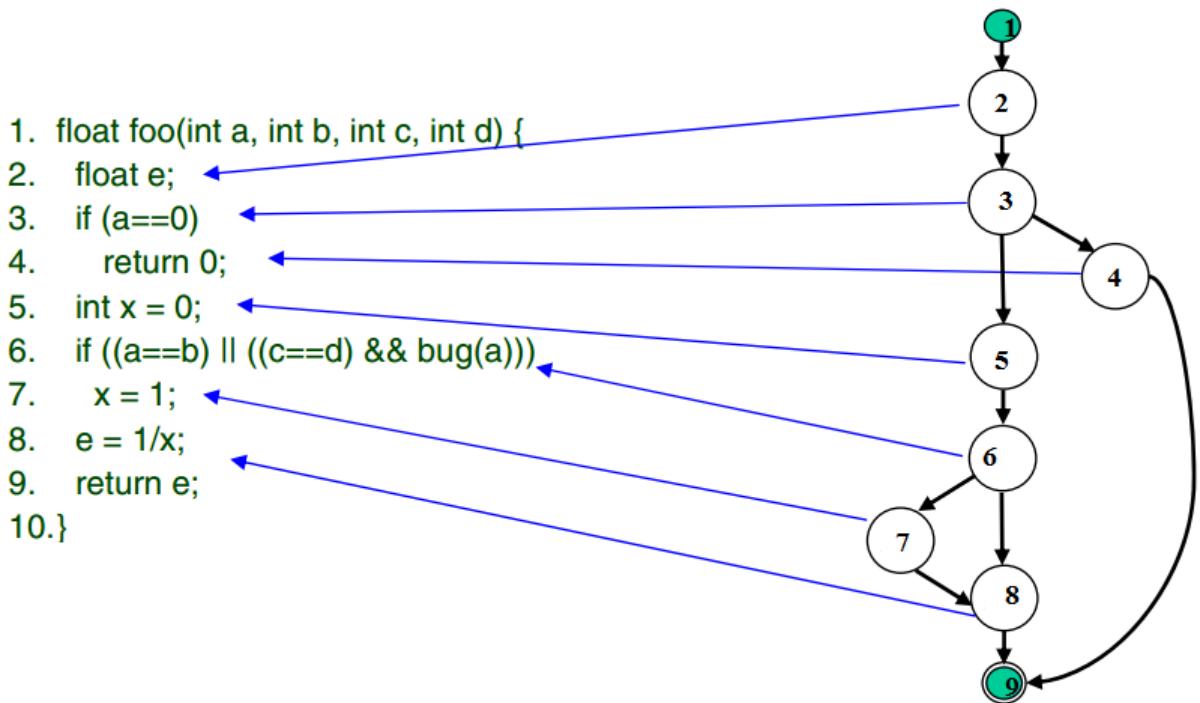
Mục tiêu của kiểm thử dòng dữ liệu là chọn lựa được đúng C đường thi hành tuyến tính độc lập của TPPM cần kiểm thử rồi kiểm thử đổi sống của từng biến trên từng đường thi hành này xem có lỗi gì không.

Quy trình kiểm thử dòng dữ liệu:

Qui trình kiểm thử dòng dữ liệu của 1 TPPM gồm các bước công việc sau :

- Từ TPPM cần kiểm thử, xây dựng đồ thị dòng điều khiển tương ứng, rồi chuyển thành đồ thị dòng điều khiển nhị phân, rồi chuyển thành đồ thị dòng dữ liệu.
- Tính độ phức tạp Cyclomatic của đồ thị ($C = P + 1$).
- Xác định C đường thi hành tuyến tính độc lập cơ bản cần kiểm thử
- Lặp kiểm thử đổi sống từng biến dữ liệu :
 - o mỗi biến có thể có tối đa C kịch bản đổi sống khác nhau.
 - o trong từng kịch bản đổi sống của 1 biến, kiểm thử xem có tồn tại cặp đôi hoạt động không bình thường nào không ? Nếu có hãy ghi nhận để lập báo cáo kết quả và phản hồi cho những người có liên quan.

Ví dụ:



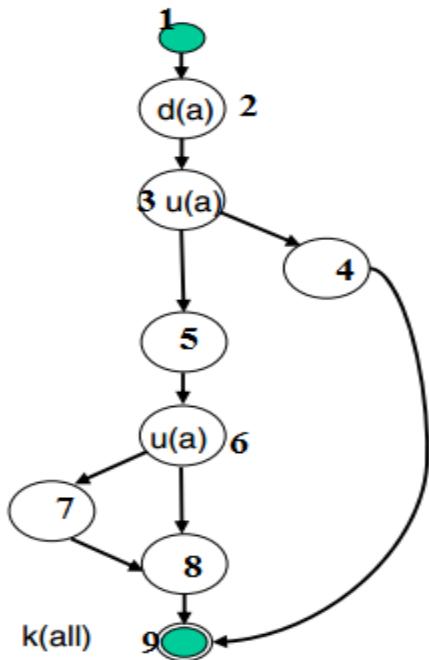
Đồ thị ở trên trước có 2 nút quyết định nhị phân nên có độ phức tạp $C = 2 + 1 = 3$.

Nó có 4 biến đầu vào (tham số) và 2 biến cục bộ.

Hãy lặp kiểm thử đòn sống từng biến a, b, c, d, e, x. trên các đường đi khác nhau

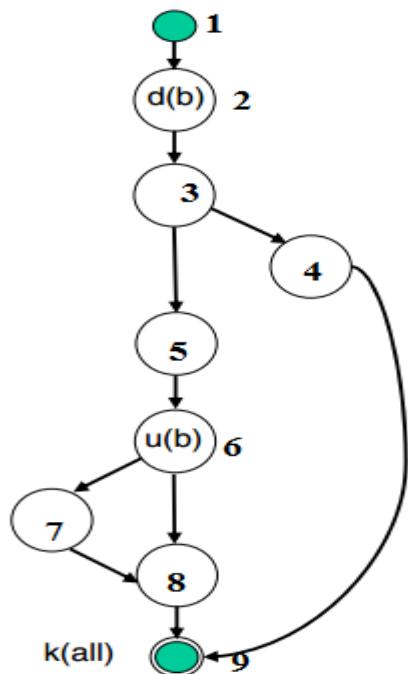
3 đường đi đó là: (1) : 1,2,3,5,6,7,8,9; (2): 1,2,3,5,6,8,9; (3): 1,2,3,4,9

Kiểm thử đòn sống biến a



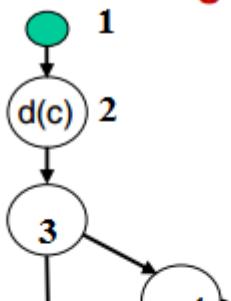
- Kịch bản 1 : ~duuk
 - Kịch bản 2 : ~duuk (giống kịch bản 1).
 - Kịch bản 3 : ~duk
- Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thường cả.

Kiểm thử đòn sống biến b



- Kịch bản 1 : ~duk
 - Kịch bản 2 : ~duk (giống kịch bản 1).
 - Kịch bản 3 : ~dk
- Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thường cả.

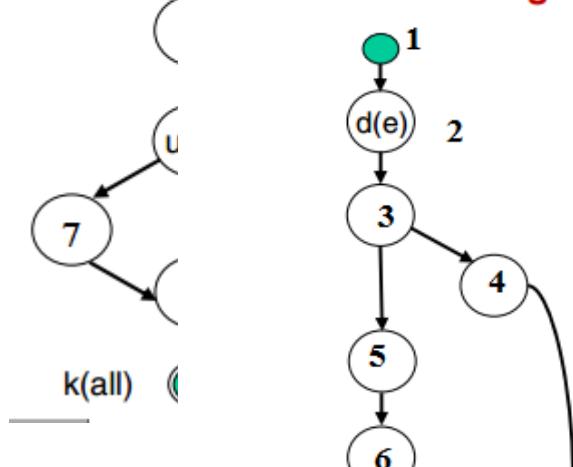
Kiểm thử đòn sóng biển c



- Kịch bản 1 : ~duk
- Kịch bản 2 : ~duk (giống kịch bản 1).
- Kịch bản 3 : ~dk

Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thường cả.

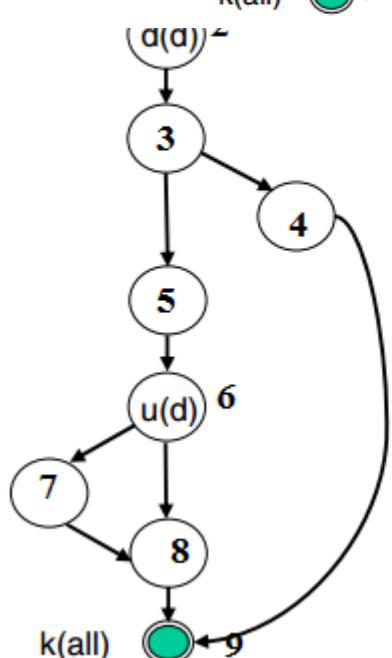
Kiểm thử đòn sóng biển e



- Kịch bản 1 : ~dduk
- Kịch bản 2 : ~dduk (giống kịch bản 1).
- Kịch bản 3 : ~dk

Trong 3 kịch bản trên, kịch bản 1 & 2 có chứa cặp đôi dd bất thường nên cần tập trung chú ý kiểm tra xem có phải là lỗi không.

Kiểm tra

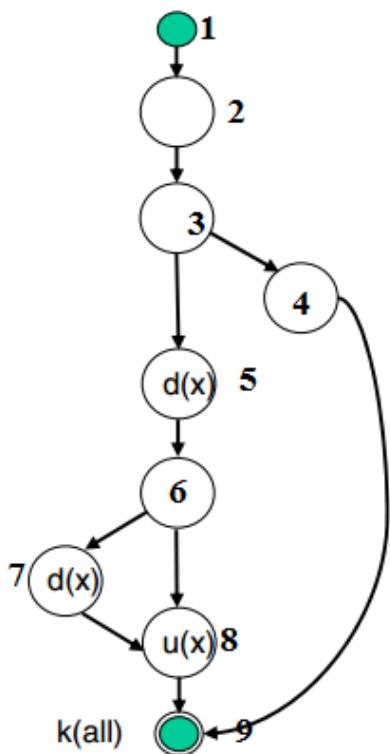


- Kịch bản 2 : ~duuk (giống kịch bản 1).

- Kịch bản 3 : ~dk

Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thường cả.

Kiểm thử đời sống của biến x



- Kịch bản 1 : ~dduk
- Kịch bản 2 : ~duk
- Kịch bản 3 : ~

Trong 3 kịch bản trên, chỉ có kịch bản 1 có chứa cặp đôi dd bất thường nên cần tập trung chú ý kiểm tra xem có phải là lỗi không.

Ca kiểm thử

Đầu vào					Đầu ra		
a	b	c	d	e	x	foo	
0						0	
5	5			1	1	1	
Bug(a)		5	5	1	1	1	
5	9	5	7	1/0	0	lỗi	

BÀI 8. KỸ THUẬT KIỂM THỬ HỘP ĐEN (BLACK - BOX) (I)

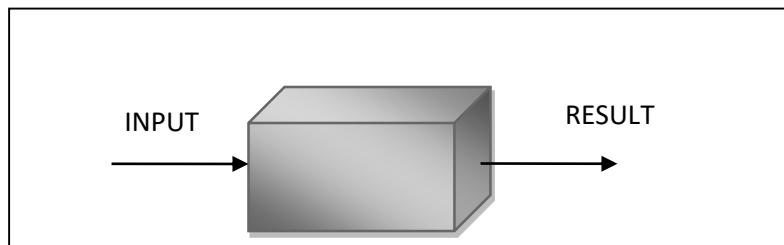
8.1. Giới thiệu

Đối tượng được kiểm thử là 1 thành phần phần mềm (TPPM). TPPM có thể là 1 hàm chức năng, 1 module chức năng, 1 phân hệ chức năng... Nói chung, chiến lược kiểm thử hộp đen thích hợp cho mọi cấp độ kiểm thử từ kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống, kiểm thử độ chấp nhận của người dùng.

Kiểm thử hộp đen (black-box testing) là chiến lược kiểm thử TPPM dựa vào thông tin duy nhất là các đặc điểm yêu cầu chức năng của TPPM tương ứng.

Đây là chiến lược kiểm thử theo góc nhìn từ ngoài vào, các người tham gia kiểm thử hộp đen không cần có kiến thức nào về thông tin hiện thực TPPM cần kiểm thử (mã nguồn của thành phần phần mềm, thuật giải được dùng, các dữ liệu được xử lý...).

Mô hình kiểm thử hộp đen



Đặc trưng:

- Nhằm thuyết minh: các chức năng phần mềm đủ và vận hành đúng
- Thực hiện các phép thử qua giao diện
- Cơ sở: đặc tả, các điều kiện vào/ra và cấu trúc dữ liệu
- Ít chú ý tới cấu trúc logic nội tại của nó

Việc kiểm thử hộp đen nhằm tìm các loại sai liên quan:

- Chức năng: đủ, đúng đắn
- Giao diện: vào, ra: đủ, phù hợp, đúng, tiện lợi
- Cấu trúc, truy cập dữ liệu: thông suốt, đúng đắn
- Thực thi: trôi chảy, kịp thời, chịu lỗi, phục hồi
- Khởi đầu hoặc kết thúc: mỗi tiến trình thông suốt.

Qui trình kiểm thử hộp đen tổng quát gồm các bước chính :

f - Phân tích đặc tả về các yêu cầu chức năng mà TPPM cần thực hiện.

f - Dùng 1 kỹ thuật định nghĩa các testcase xác định (sẽ giới thiệu sau) để định nghĩa các testcase. Định nghĩa mỗi testcase là xác định 3 thông tin sau :

- o Giá trị dữ liệu nhập để TPPM xử lý (hoặc hợp lệ hoặc không hợp lệ).
- o Trạng thái của TPPM cần có để thực hiện testcase.
- o Giá trị dữ liệu xuất mà TPPM phải tạo được.

f - Kiểm thử các testcase đã định nghĩa.

f - So sánh kết quả thu được với kết quả kỳ vọng trong từng testcase, từ đó lập báo cáo về kết quả kiểm thử.

Vì chiến lược kiểm thử hộp đen thích hợp cho mọi mức độ kiểm thử nên nhiều người đã nghiên cứu tìm hiểu và đưa ra nhiều kỹ thuật kiểm thử khác nhau như:

1. Kỹ thuật phân lớp tương đương (Equivalence Class Partitioning).
2. Kỹ thuật phân tích các giá trị biên (Boundary value analysis).
3. Kỹ thuật dùng các bảng quyết định (Decision Tables)
4. Kỹ thuật kiểm thử các bộn thân kỳ(Pairwise)
5. Kỹ thuật dùng bảng chuyển trạng thái (State Transition)
6. Kỹ thuật phân tích vùng miền (domain analysis)
7. Kỹ thuật dựa trên đặc tả Use Case (Use case)
8. Kỹ thuật dùng lược đồ quan hệ nhân quả(Cause-Effect Diagram)

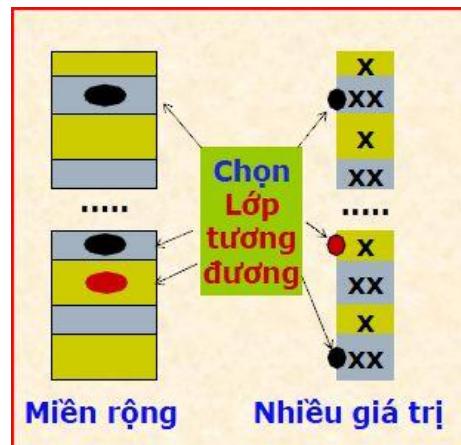
Trong phạm vi chương trình chúng ta xem xét một số kỹ thuật điển hình.

8.2. Kỹ thuật Phân chia lớp tương đương

Thực hiện chia miền vào của chương trình thành các lớp dữ liệu để lập ra các ca kiểm thử theo mỗi lớp đó.

Mỗi lớp dùng để kiểm thử một chức năng gọi là lớp tương đương.

Đối với mỗi dữ liệu vào, xác định các lớp tương đương. Chọn dữ liệu đại diện cho mỗi lớp tương đương đó. Cuối cùng là kết hợp các dữ liệu để tạo ra bộ dữ liệu kiểm thử.



Phân hoạch tương đương

Nguyên tắc:

- Nếu dữ liệu vào thuộc một khoảng, xây dựng:
 - Một lớp các giá trị lớn hơn
 - Một lớp các giá trị nhỏ hơn
 - N lớp các giá trị hợp lệ
- Nếu dữ liệu vào là tập hợp các giá trị, xây dựng:
 - Một lớp tập rỗng
 - Một lớp quá nhiều các giá trị
 - N lớp hợp lệ
- Nếu dữ liệu vào là điều kiện ràng buộc, xây dựng:
 - Một lớp với ràng buộc được thỏa mãn
 - Một lớp với ràng buộc không được thỏa mãn

Ví dụ: Bài toán nhập vào tháng: Kiểm tra tính hợp lệ của đầu vào có là tháng trong năm hay không ?

Condition (Điều kiện)	Valid(Các lớp tương đương hợp lệ)	Invalid(Các lớp tương đương không hợp lệ)
Số nhập vào: A	$1 \leq A \leq 12$	$A < 1, A > 12$
Loại dữ liệu	Integer	Not integer, ký tự

Định nghĩa

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

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

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

- Giảm thiểu số lượng các ca kiểm thử khác mà phải được phát triển để hoàn thành mục tiêu đã định của kiểm thử “hợp lý”.
- Bao phủ một tập rất lớn các ca kiểm thử có thể khác. Tức là, nó nói cho chúng ta một thứ gì đó về sự có mặt hay vắng mặt của những lỗi qua tập giá trị đầu vào cụ thể.

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

- Xác định các lớp tương đương.
- Xác định các ca kiểm thử.

Xác định các lớp tương đương.

Các lớp tương đương được xác định bằng cách lấy mỗi trạng thái đầu vào (thường là 1 câu hay 1 cụm từ trong đặc tả) và phân chia nó thành 2 hay nhiều nhóm.

Mẫu liệt kê các lớp tương đương:

Điều kiện đầu vào	Các lớp tương đương hợp lệ	Các lớp tương đương không hợp lệ

- Điều kiện đầu vào là một giá trị đặc biệt, mảng số hay chuỗi, tập hợp hay điều kiện đúng sai.
- Các lớp tương đương hợp lệ là mô tả các đầu vào hợp lệ của chương trình
- Các lớp tương đương không hợp lệ là mô tả các trạng thái khác của chương trình như: sai, thiếu, không đúng...

Nguyên tắc để xác định lớp tương đương.

- Nếu điều kiện đầu vào định rõ giới hạn của một mảng thì chia vùng tương đương thành 3 tình huống:
 - Xác định một lớp tương đương hợp lệ.
 - Xác định hai lớp tương đương không hợp lệ.
- Nếu điều kiện đầu vào là một giá trị xác định thì chia vùng tương đương thành 3 tình huống:
 - Một lớp tương đương hợp lệ.
 - Hai lớp tương đương không hợp lệ.
- Nếu điều kiện đầu vào chỉ định là một tập giá trị thì chia vùng tương đương thành 2 tình huống như sau:
 - Một lớp tương đương hợp lệ.
 - Một lớp tương đương không hợp lệ.
- Nếu điều kiện đầu vào xác định là một kiểu đúng sai thì chia vùng tương đương thành 2 tình huống:
 - Một lớp tương đương hợp lệ.
 - Một lớp tương đương không hợp lệ.

a. Xác định các ca kiểm thử.

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

1. Gán 1 số duy nhất cho mỗi lớp tương đương.
2. Cho đến khi tất cả các lớp tương đương hợp lệ được bao phủ bởi (hợp nhất thành) các ca kiểm thử. Viết 1 ca kiểm thử mới bao phủ càng nhiều các lớp tương đương đó chưa được bao phủ càng tốt.
3. Cho đến khi các ca kiểm thử của bạn đã bao phủ tất cả các lớp tương đương không hợp lệ. Viết 1 ca kiểm thử mà bao phủ một và chỉ một trong các lớp tương đương không hợp lệ chưa được bao phủ.
4. Lý do mà mỗi ca kiểm thử riêng bao phủ các trường hợp không hợp lệ là vì các kiểm tra đầu vào không đúng nào đó che giấu hoặc thay thế các kiểm tra đầu vào không đúng khác.

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

- b. Ví dụ về phân lớp tương đương.

Cho bài toán như sau:

User:

Password:

Yêu cầu: Thiết kế test case sao cho khi người dùng nhập user vào ô text thì chỉ cho nhập số ký tự [6 – 20].

Bài làm

Do yêu cầu của bài toán chỉ cho phép nhập số ký tự vào trong khi nhập của user nằm [6 - 20] nên ta có tình huống kiểm thử sau:

- Nhập vào một trường hợp hợp lệ: nhập 7 ký tự.
- Nhập vào trường hợp không hợp lệ thứ nhất: nhập 5 ký tự.
- Nhập vào trường hợp không hợp lệ thứ hai: nhập vào 21 ký tự.
- Trường hợp đặc biệt: không nhập gì vào ô text đó (để trống).

Lập bảng các lớp tương đương:

Điều kiện đầu vào	Các lớp tương đương hợp lệ	Các lớp tương đương không hợp lệ
Cho phép nhập số ký tự nằm [6 – 20]	<ul style="list-style-type: none"> - Nhập vào 7 ký tự 	<ul style="list-style-type: none"> - Nhập vào 5 ký tự - Nhập vào 21 ký tự - Để trống ô đó

8.3. Phân tích giá trị biên - BVA - Boundary Value Analysis

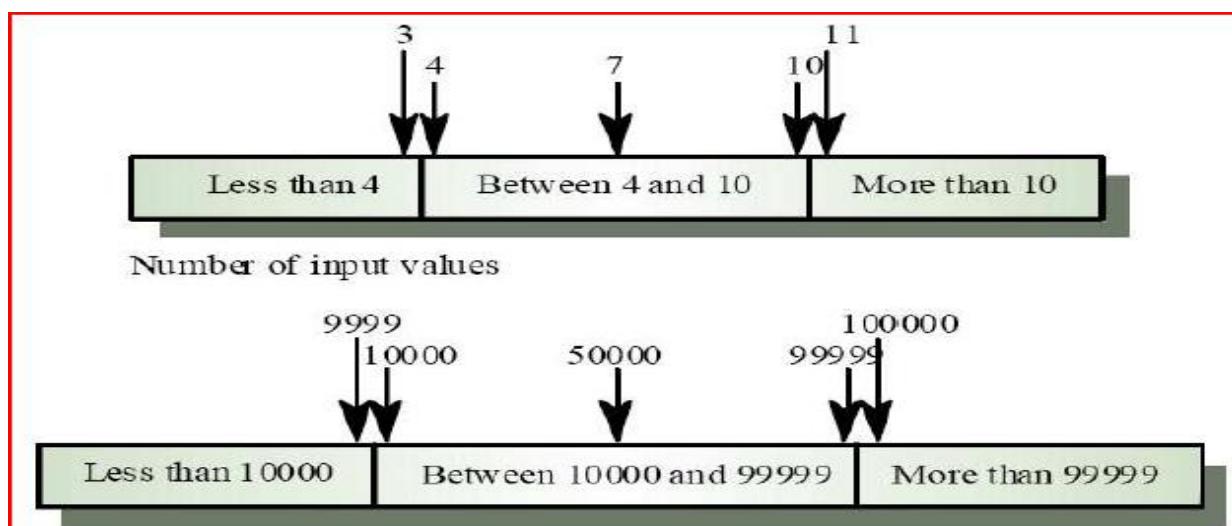
Định nghĩa.

Phân tích giá trị biên (*BVA*) là kỹ thuật thiết kế test case và hoàn thành phân vùng tương đương.

Mục tiêu là lựa chọn các test case để thực thi giá trị biên.

Kiểm thử các dữ liệu vào bao gồm:

- Giá trị nhỏ nhất.
- Giá trị gần kề lớn hơn giá trị nhỏ nhất.
- Giá trị bình thường.
- Giá trị gần kề bé hơn giá trị lớn nhất.
- Giá trị lớn nhất.



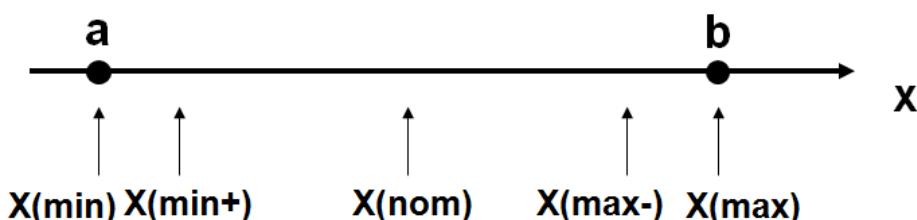
Hình 3: Ví dụ về biểu thị phân tích giá trị biên

a. Nguyên tắc chọn dữ liệu.

- Nếu giá trị đầu vào xác định là một mảng có biên là a và b($a < b$) thì có thể thiết kế được các test case như sau:

- Biên a
- Biên b
- Giá trị nhỏ hơn biên a
- Giá trị lớn hơn biên b
- Một giá trị nằm giữa a và b.

➤ *Kiểm thử theo giá trị biên với một biến*



Biến là: x

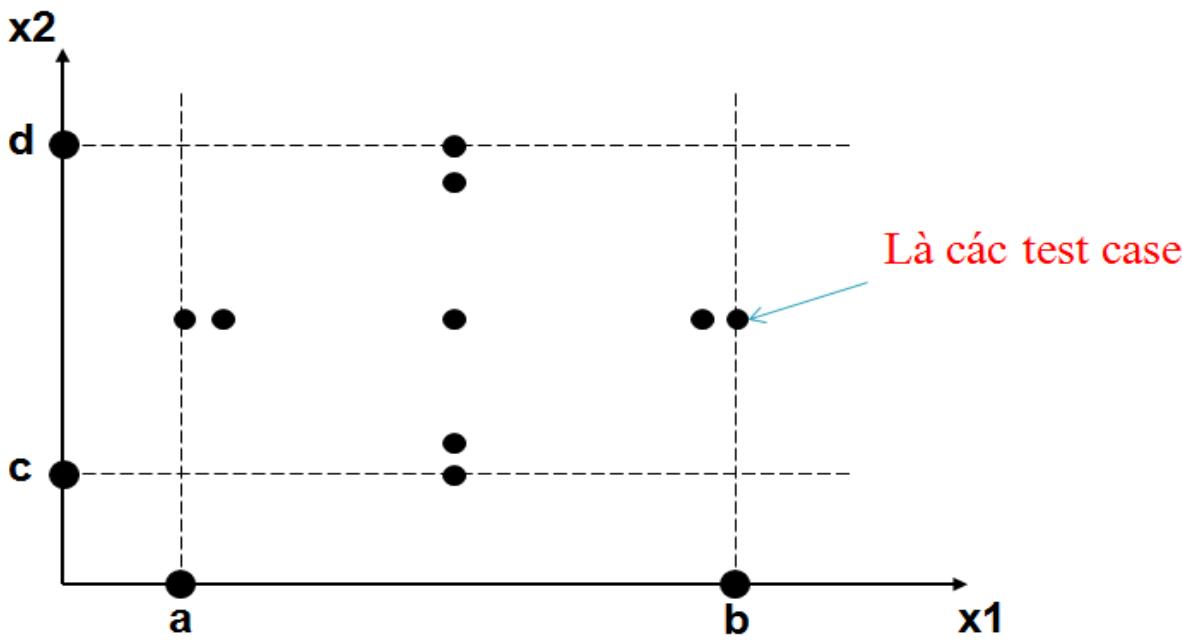
Điều kiện: $a \leq x \leq b$

Đồ thị kiểm thử giá trị biên với một biến

Ví dụ: Cho một mảng [-3 , 10] ta có thể thiết kế được các test case là:

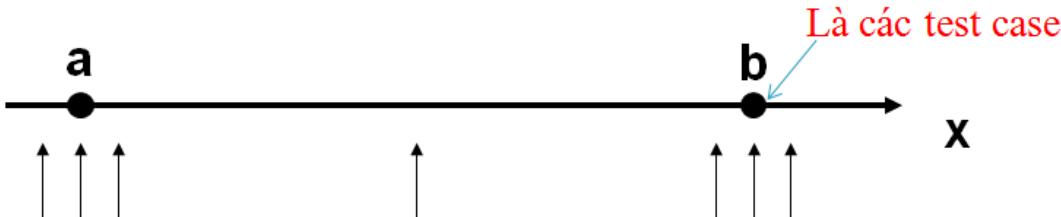
- Giá trị nhỏ nhất: -3
- Giá trị lớn nhất: 10
- Giá trị nhỏ hơn giá trị nhỏ nhất: -4
- Giá trị lớn hơn giá trị lớn nhất: 11
- Giá trị nằm trong -3 và 10: 0

➤ *Kiểm thử theo giá trị biên theo hai biến x1 và x2.*



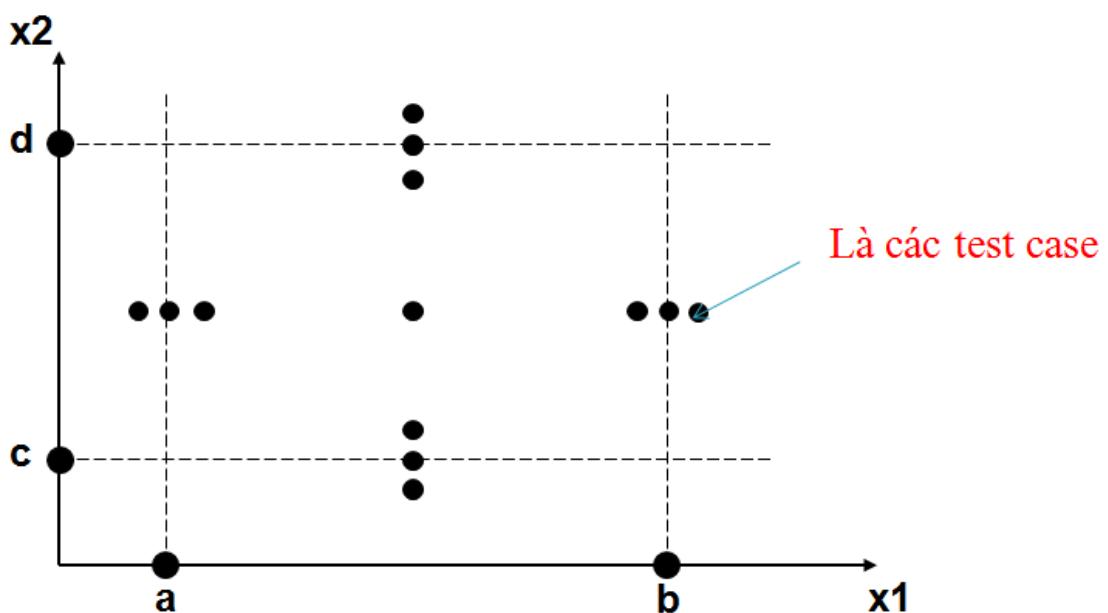
Đồ thị kiểm thử giá trị biên với hai biến

- *Kiểm thử theo giá trị biên đầy đủ với một biến x_1 .*



Hình 6: Đồ thị kiểm thử giá trị biên đầy đủ với một biến

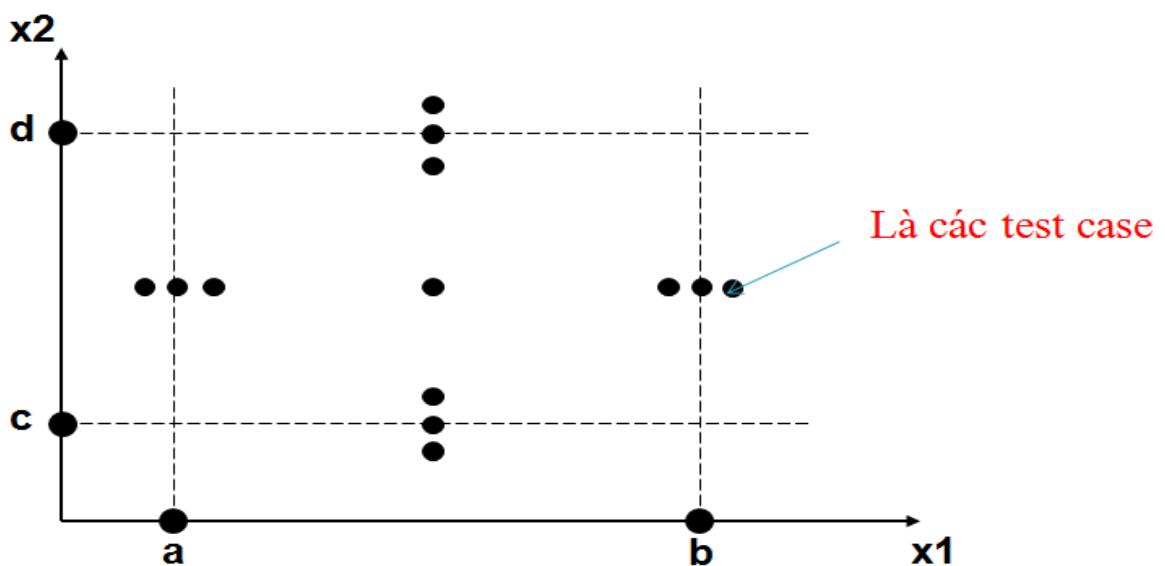
- *Kiểm thử theo giá trị biên đầy đủ với hai biến x_1 và x_2 .*



Test cases for variable x_1 and x_2

Hình 7: Đồ thị kiểm thử giá trị biên dày dặn với hai biến.

➤ *Kiểm thử theo giá trị biên xấu nhất với hai biến x_1 và x_2 .*



Test cases for variable x_1 and x_2

Hình 8: Đồ thị kiểm thử giá trị biên xấu nhất với hai biến.

Kết luận: Số lượng trường hợp kiểm thử phải có: (với n là các biến)

- Kiểm thử theo giá trị biên: $4n + 1$
- Kiểm thử theo giá trị biên đầy đủ: $6n + 1$
- Kiểm thử theo giá trị biên xấu nhất: 5^n
- Nếu miền giá trị đầu vào là một danh sách các giá trị thì có thể thiết kế các test case như sau:
 - Giá trị nhỏ nhất.
 - Giá trị lớn hơn giá trị nhỏ nhất.
 - Giá trị lớn nhất.
 - Giá trị nhỏ hơn giá trị lớn nhất.

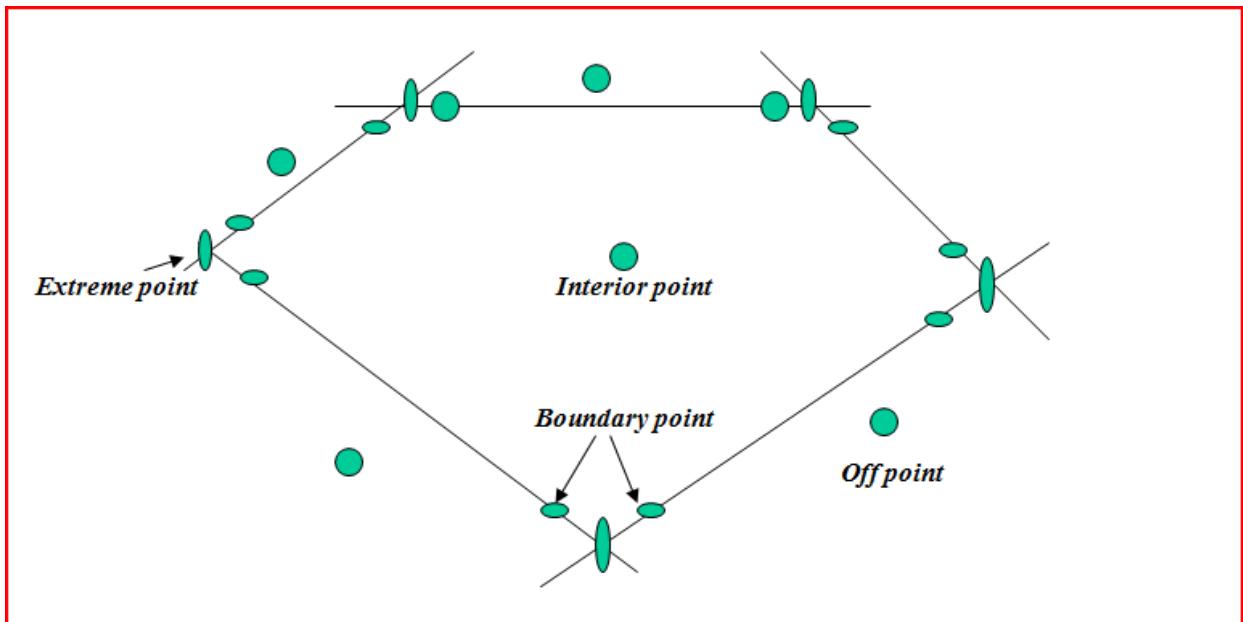
Ví dụ: Cho một danh sách như sau { 3,5,90,100,102} nên có thể thiết kế như sau:

- Giá trị nhỏ nhất: 3
- Giá trị lớn hơn giá trị nhỏ nhất: 5
- Giá trị lớn nhất: 102
- Giá trị nhỏ hơn giá trị lớn nhất: 100
- Nếu dữ liệu vào là điều kiện ràng buộc số giá trị thì có thể thiết kế được các test case như sau:
 - Số giá trị tối thiểu.
 - Số giá trị tối đa.
 - Và một số giá trị không hợp lệ.

b. Phân loại miền biên.

- Điểm trên biên (*Boundary point*).
- Điểm cực biên (*Extreme point*).

- Điểm ngoài biên (*Off point*).
- Điểm trong biên (*Interior point*.)



Hình 9: Phân loại miền biên

c. Ví dụ cho phân tích giá trị biên

Kiểm tra tính hợp lệ của tháng trong năm thì ta có thể thiết kế như sau:

- Giá trị biên: 1 và 12
- Giá trị cận biên ở bên trong: 2 và 11
- Giá trị cận biên ở bên ngoài: 0 và 13

BÀI 9. KỸ THUẬT KIỂM THỬ HỘP ĐEN (II)

9.1. Kỹ thuật dùng bảng quyết định (decision table)

Bảng quyết định là 1 công cụ rất hữu ích để đặc tả các yêu cầu phần mềm hoặc để đặc tả bảng thiết kế hệ thống phần mềm. Nó miêu tả các qui tắc nghiệp vụ phức tạp mà phần mềm phải thực hiện dưới dạng dễ đọc và dễ kiểm soát :

Conditions	Rule 1	Rule 2	...	Rule p
Condition-1				
Condition-2				
...				
Condition-m				
Actions	Action-1	Action-2	...	Action-n
Action-1				
Action-2				
...				
Action-n				

Condition-1 tới Condition-m miêu tả điều kiện dữ liệu nhập khác nhau có thể có. Action-1 tới Action-n miêu tả hoạt động khác nhau mà hệ thống có thể thực hiện phụ thuộc vào tổ hợp điều kiện dữ liệu nhập nào. Mỗi cột miêu tả 1 luật cụ thể: tổ hợp điều kiện nhập cụ thể và các hoạt động cụ thể cần thực hiện.

Lưu ý các hoạt động cần thực hiện không phụ thuộc vào thứ tự các điều kiện nhập, nó chỉ phụ thuộc vào giá trị các điều kiện nhập.

Tương tự, các hoạt động cần thực hiện không phụ thuộc vào trạng thái hiện hành của TPPM, chúng cũng không phụ thuộc vào các điều kiện nhập đã có trước đó.

Chúng ta sẽ lấy 1 thí dụ cụ thể để làm rõ bảng quyết định. Giả sử TPPM cần kiểm thử là phân hệ chức năng nhỏ của công ty bảo hiểm : nó sẽ khuyến mãi cho những chủ xe (cũng là tài xế) nếu họ thỏa ít nhất 1 trong 2 điều kiện : đã lập gia đình / là sinh viên giỏi.

Mỗi dữ liệu nhập là 1 giá trị luận lý, nên bảng quyết định chỉ cần có 4 cột, miêu tả 4 luật khác nhau :

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Married?	Yes	Yes	No	No
Good Student?	Yes	No	Yes	No
Actions				
Discount (\$)	60	25	50	0

Qui trình cụ thể để thực hiện kiểm thử dùng bảng quyết định :

1. Tìm bảng quyết định từ đặc tả về yêu cầu chức năng của TPPM hay từ bảng thiết kế TPPM. Nếu chưa có thì xây dựng nó dựa vào đặc tả về yêu cầu chức năng hay dựa vào bảng thiết kế TPPM.

2. Từ bảng quyết định chuyển thành bảng các testcase trong đó mỗi cột miêu tả 1 luật được chuyển thành 1 đến n cột miêu tả các testcase tương ứng với luật đó :

- Nếu điều kiện nhập là trị luận lý thì mỗi cột luật được chuyển thành 1 cột testcase.

- Nếu điều kiện nhập là 1 lớp tương đương (nhiều giá trị liên tục) thì mỗi cột luật được chuyển thành nhiều testcase dựa trên kỹ thuật lớp tương đương hay kỹ thuật giá trị biên.

Ví dụ:

- Trường hợp 1

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Cond 1	Yes	Yes	No	No
Cond 2	Yes	No	Yes	No
Actions				
Action-1	60	25	50	0



	TC1	TC2	TC3	TC4
Conditions				
Married?	Yes	Yes	No	No
Good Student?	Yes	No	Yes	No
Actions				
Discount (\$)	60	25	50	0

- Trường hợp 2

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Cond 1	0–1	1–10	10–100	100–1000
Cond 2	<5	5	6 or 7	>7
Actions				
Action-1	Do X	Do Y	Do X	Do Z



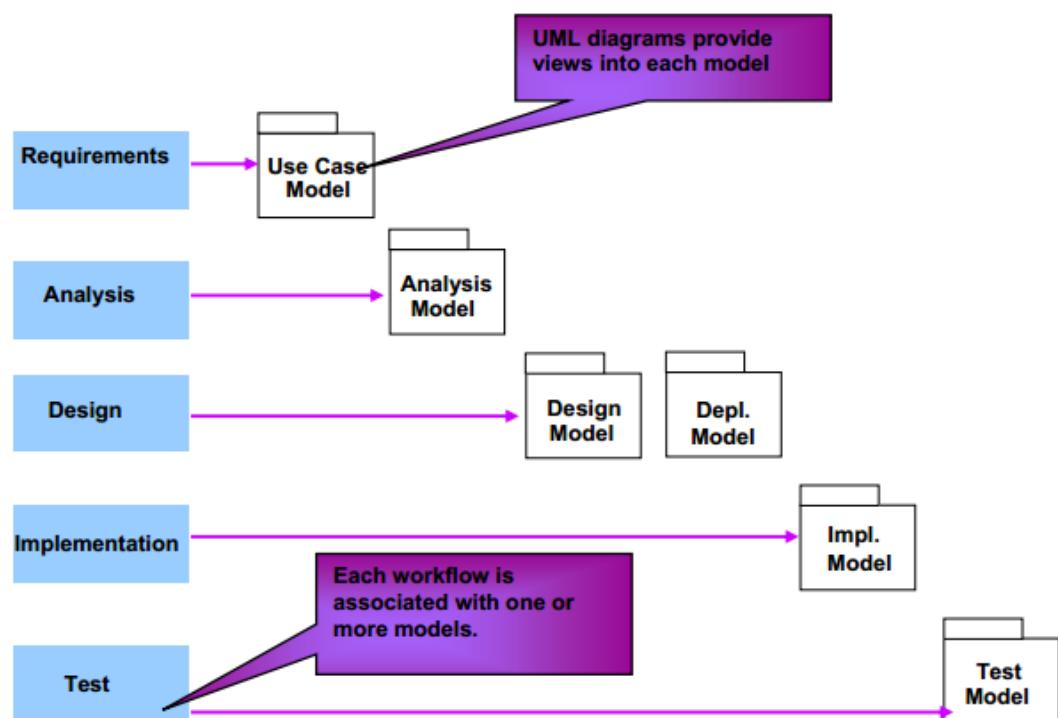
	TC1	TC2	TC3	TC4
Conditions				
Cond 1	0	5	50	500
Cond 2	3	5	7	10
Actions				
Action-1	Do X	Do Y	Do X	Do Z

9.2. Kỹ thuật dựa trên đặc tả Use Case (Use case)

Trong quá trình phát triển phần mềm hợp nhất, ta thực hiện nhiều workflows khác nhau : nắm bắt yêu cầu phần mềm, phân tích từng yêu cầu, thiết kế chi tiết để giải quyết từng yêu cầu, hiện thực từng phần bằng thiết kế, kiểm thử kết quả.

Mỗi workflows, thậm chí mỗi lần lập thực hiện 1 workflow, ta phải có kết quả, kết quả này phải được miêu tả ở dạng dễ đọc, dễ hiểu bởi nhiều người và phải có găng tay nghĩa để tránh nhầm lẫn.

Ta dùng khái niệm mô hình để miêu tả hệ thống phần mềm theo một góc nhìn nào đó.

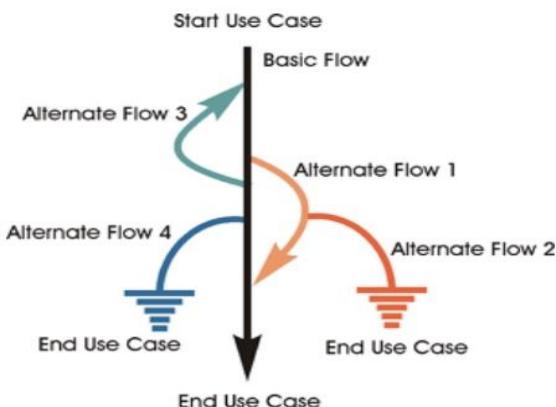


Về nguyên tắc, khi kiểm thử 1 thành phần phần mềm, ta có thể tận dụng bất kỳ thông tin nào trong bất kỳ mô hình nào. Kỹ thuật kiểm thử dùng thông tin trong use-case là kỹ thuật định nghĩa các testcase dựa vào các kịch bản thực hiện usecase.

Như chúng ta biết, mô hình usecase miêu tả hệ thống phần mềm theo góc nhìn bên ngoài : nó cung cấp các chức năng nào cho những “user” nào. Thành phần thiết yếu nhất của mô hình usecase là các lược đồ usecase.

Đi kèm với mỗi một Use case là đặc tả chi tiết của Use Case, ở dạng văn bản mô tả luồng sự kiện chính, và các luồng sự kiện rẽ nhánh và một số thông tin khác. Như trong tài liệu của Alistair Cockburn trong cuốn “Writing Effective Use Cases”.

Trong bảng là đặc tả của Use Case “Đăng ký khóa học”. Trong đó Pre-Condition là tiền điều kiện, là điều kiện cần đảm bảo trước khi thực hiện Use Case. Post-Condition là hậu điều kiện, là điều kiện đảm bảo sau khi thực hiện Use case. Trigger là sự kiện kích hoạt Use Case. Basic Flow là luồng chính, khi Use Case thực hiện tương ứng với trường hợp mọi điều kiện đều đúng đắn. Alternate Flow là các luồng rẽ nhánh khi thực hiện Use Case, tương ứng với các trường hợp rẽ nhánh và ngoại lệ xảy ra khi thực hiện Use Case. Một Use Case gồm một luồng chính và nhiều luồng rẽ nhánh như minh họa trong hình dưới đây.



Đặc tả có thể được mô tả bằng ngôn ngữ tự nhiên theo định dạng như trong bảng dưới

1. Pre-Condition

2. Pos-Condition

Nếu đăng ký khóa học thành công, danh sách các khóa học được ghi nhận vào hệ thống
nếu không thành công đưa ra các thông báo tương ứng

3. Trigger

Use Case này được thực hiện khi Sinh viên truy cập vào Website của trung tâm X

Basic flow

- 1 Hệ thống hiển thị giao diện đăng nhập

- 2 Sinh nhập User Name và PassWord
- 3 Sinh viên Click vào Button Đăng Nhập"
- 4 Hệ thống kiểm tra tính hợp lệ của UserName và PassWord
- 5 Hệ thống kiểm tra tài khoản của sinh viên có trong hệ thống không
- 6 Hệ thống hiển thị các chức năng mà sinh viên có quyền thực hiện
- 7 Sinh viên chọn chức năng "Tạo lịch học"
- 8 Hệ thống kiểm tra tình trạng đóng ký khóa học
- 9 Hệ thống lấy về danh sách các khóa học để xuất từ Danh mục khóa học và hiển thị danh sách sinh viên tham gia khóa học để xuất
- 10 Sinh viên chọn Khóa học từ danh sách các khóa học để xuất có thể
Bước này lặp lại nhưng tối đa là 4 khóa học được chọn trong một lần đăng ký
và còn khóa học để chọn
- 11 Sinh viên thực hiện "Cập nhật" lịch học
- 12 Hệ thống hiển thị lịch học gồm các khóa học đề nghị được chọn và lưu lịch học
vào hệ thống

Alternate flow

- 1 1.1. UserName và PassWord không hợp lệ
Hệ thống hiển thị thông báo "UserName trong khoản từ 8 đến 16 ký tự,
PassWord là 8 ký tự, yêu cầu nhập lại"
Nếu tài khoản của sinh viên không có trong hệ thống, số lần đăng nhập chưa
quá 3
 - 2.1. Hệ thống đếm số lần đăng nhập
 - 2.2. Hệ thống hiển thị thông báo "Tài khoản không tồn tại, đăng nhập lại"
 - 2.3 Hệ thống kiểm tra số lần đăng nhập nếu nhỏ hơn 3 thì quay về luồng chính,
ngược lại chuyển luồng phụ 3
- 3 3.1. Nếu tài khoản của sinh viên không có trong hệ thống và số lần đăng nhập
bằng 3
Hệ thống hiển thị thông báo: "Tài khoản không tồn tại, số lần đăng nhập
tối đa là 3" vô hiệu hóa chức năng đăng nhập và kết thúc
- 4 4. Hệ thống đóng không cho đăng ký
 - 4.1. Hệ thống hiển thị thông báo "Hệ thống đóng đăng ký khóa học, bạn vui
lòng đăng ký lần sau" và kết thúc Use Case
- 5 5. Tất cả các khóa học để xuất đã đủ sinh viên tối đa đăng ký
 - 5.1. Hệ thống hiển thị thông báo "Tất cả các khóa học đã đủ sinh viên tối đa
đăng ký, bạn vui lòng đăng ký sau" và kết thúc Use Case

Với phương pháp kiểm thử dựa trên đặc tả Use Case, kiểm thử viên tiến hành đọc đặc tả Use Case từ đó xác định các Test Case theo các bước như sau:

- + Xác định các kịch bản Use Case (Use Case Scenario), dựa theo các luồng điều khiển đã mô tả trong luồng chính và luồng phụ, chúng ta sẽ xác định được các đường đi khác nhau khi thực hiện một Use Case mỗi một đường đi là một kịch bản Use Case. Tương ứng với mỗi kịch bản Use Case là một kịch bản kiểm thử (Test Scenario). Một kịch bản Test gồm một chuỗi các hành động (hành động tác nhân, hành động của hệ thống) xảy ra theo một đường đi khi thực hiện Use Case.
- + Xác định Test Case, dựa trên mỗi kịch bản Test kiểm thử viên sẽ xác định các bộ giá trị thỏa mãn khác nhau của các dữ liệu đầu vào và dữ liệu đầu ra mong đợi. Các bộ giá trị thỏa mãn khác nhau của đầu vào được xác định thông qua đọc đặc tả luồng để nhận biết các giá trị theo phương pháp kết hợp các điều kiện đồng thời thỏa mãn (and), ít nhất một điều kiện thỏa mãn (or). Tương ứng với một kịch bản Test, kiểm thử viên có thể xác định được nhiều Test Case thỏa mãn một kịch bản Test.
- + Xác định các giá trị cụ thể cho từng Test Case, khi thực thi test kiểm thử viên sẽ nhập các giá trị cụ thể ứng với các trường hợp làm cho dữ liệu đầu vào thỏa mãn, không thỏa mãn theo miền giá trị.

Các kịch bản thực hiện Use Case

Kịch bản ID	Tên kịch bản	Luồng bắt đầu	Rẽ nhánh
SC1	Nhập UserName & Pass 1 lần hợp lệ và đăng ký khóa học thành công	Basic Flow	
SC2	Nhập UserName và PassWord không hợp lệ	Basic Flow	A1
SC3	Tài khoản đăng nhập không có trong hệ thống, số lần đăng nhập nhỏ hơn 3	Basic Flow	A2
SC4	Tài khoản đăng nhập không có trong hệ thống, số lần đăng nhập bằng 3	Basic Flow	A3
SC5	Hệ thống đóng, không cho đăng ký	Basic Flow	A4
SC6	Tất cả các khóa học đã đủ sinh viên tối đa đăng ký	Basic Flow	A5

Các Test Case của Use Case “Đăng Nhập”

Test Case	Kịch bản ID	User Nam	Pass Wor	Số lần	Tài khoản	Kết quả mong đợi
-----------	-------------	----------	----------	--------	-----------	------------------

ID		e	d	đăng nhập	có trong hệ thống	
TC001	SC1	V	V	N/A	True	Hiển thị các chức năng sinh viên được thực hiện
TC002	SC2	V	I	<3	N/A	message: "UserName trong khoảng từ 8 đến 16 ký tự, passWord là 8 ký tự, nhập lại"
TC003	SC2	I	V	<3	N/A	Message: "UserName trong khoảng từ 8 đến 16 ký tự, passWord là 8 ký tự, nhập lại"
TC004	SC3	V	V	<3	Fall	Message: "Tài khoản không tồn tại, đăng nhập lại"
TC005	SC4	V	V	>=3	Fall	Message: "Tài khoản không tồn tại, Số lần đăng nhập tối đa là 3", vô hiệu hóa chức năng đăng nhập

9.3. Đoán lỗi

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

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

Nhân xét về Black - Box

❖ Ưu điểm

- Mang lại nhiều hiệu quả hơn trên những đơn vị mã nguồn lớn.

- Người kiểm thử không cần có một kiến thức sâu rộng về đặc tả và ngôn ngữ lập trình
 - Người kiểm thử và các lập trình viên là độc lập với nhau.
 - Sẽ giúp cho việc hiển thị sự mơ hồ hoặc sự mâu thuẫn trong các đặc tả hệ thống.
 - Các ca kiểm thử có thể được xây dựng một cách sớm nhất ngay sau khi hoàn thành việc đặc tả hệ thống.
- ❖ Nhược điểm
- Chỉ có một số lượng nhỏ các yếu tố đầu vào được kiểm tra.
 - Nếu không đặc tả rõ ràng và chi tiết, các trường hợp kiểm thử khó có thể được thiết kế.
 - Có thể lặp lại, hoặc bỏ qua trường hợp
 - Có thể chấp nhận nhiều đường dẫn chương trình mà chưa được kiểm tra.
 - Không thể định hướng tới những đoạn mã cụ thể mà được cho là rất phức tạp...
 - Rất khó xác định các yếu tố đầu vào, nếu như qui trình kiểm thử không được phát triển dựa trên bản đặc tả chi tiết.

Nhận xét

Bảng 1.1 So sánh giữa Black – box và White - box

	Kiểm thử hộp trắng	Kiểm thử hộp đen
Ưu điểm	<ul style="list-style-type: none"> - Số lượng ca kiểm thử là hữu hạn. - Có khả năng tìm và phát hiện ra lỗi là rất cao. 	<ul style="list-style-type: none"> - Người kiểm thử phần mềm không cần có một kiến thức sâu rộng về đặc tả và ngôn ngữ lập trình. - Tính khách quan cao. - Sẽ giúp cho việc hiển thị sự mơ hồ hoặc sự mâu thuẫn trong các đặc tả hệ thống. - Việc kiểm thử được tiến hành một cách sớm nhất ngay sau khi hoàn thành

		việc đặc tả hệ thống. <ul style="list-style-type: none"> - Có hiệu quả cao khi được sử dụng trên hệ thống lớn. - Chi phí thấp.
Nhược điểm	<ul style="list-style-type: none"> - Yêu cầu kiến thức lập trình và khả năng tư duy của người kiểm thử. - Sử dụng phương pháp này rất dễ rơi vào “bẫy” của người viết mã. - Chi phí cao. 	<ul style="list-style-type: none"> - Nếu không có cách thức rõ ràng và chi tiết, các trường hợp kiểm thử khó có thể được thiết kế. - Có thể sẽ lặp lại hoặc thiếu sót trường hợp kiểm tra.

Kiểm thử hộp đen là phương pháp bổ sung cho hộp trắng, tuy nhiên vai trò của hai phương pháp kiểm thử này là tương đương nhau, có những lỗi mà chỉ phương pháp hộp trắng mới phát hiện được hay có lỗi chỉ có phương pháp hộp đen mới tìm ra.

Kiểm thử hộp đen thường được thực hiện sau khi kiểm thử hộp trắng hoàn thành. Vì kiểm thử hộp trắng được thực hiện ở giai đoạn viết mã (coding), còn kiểm thử hộp đen thì thường được áp dụng ở giai đoạn thiết kế và thẩm định, như vậy thì sẽ giảm bớt được số lượng các lỗi trùng lắp khi thực hiện cả hai phương pháp cùng một lúc.

Trong số báo cáo chúng tôi sẽ giới thiệu những bước cơ bản của một quy trình KTPM, làm thế nào để đánh giá và cải tiến năng lực KTPM của một tổ chức thông qua mô hình TMM (Testing Maturity Model), một mô hình được các chuyên gia đánh giá khá tốt dành cho hoạt động KTPM.

BÀI 10. MỘT SỐ VẤN ĐỀ CẦN KIỂM THỬ

10.1. Kiểm thử giao diện

10.1.1. Khái niệm kiểm thử giao diện

Kiểm thử giao diện chính là kiểm tra xem giao diện sử dụng có nhất quán, rõ ràng, có dễ sử dụng và đúng như mong đợi của người sử dụng hay không. Đảm bảo sự giao tiếp giữa người sử dụng và ứng dụng diễn ra ổn thỏa. Người dùng có thể giao tiếp với ứng dụng một cách dễ dàng mà không gặp bất cứ sự khó khăn nào về mặt giao diện.

10.1.2. Các vấn đề liên quan đến kiểm thử giao diện

10.1.2.1. Kiểm thử thiết kế giao diện người dùng

Kiểm thử thiết kế giao diện người dùng đánh giá mức độ thiết kế “quan tâm” đến người dùng, bởi cung cấp thông tin rõ ràng, phân phát thông tin phản hồi, và duy trì tính nhất quán của ngôn ngữ và phương pháp. Án tượng chủ quan về tính dễ sử dụng và nhìn và cảm nhận được xem xét cẩn thận trong kiểm thử thiết kế giao diện người dùng. Các vấn đề liên quan đến giao diện, luồng tự nhiên, các nút lệnh, khả năng sử dụng, khả năng truy cập được khảng định trong kiểm thử thiết kế giao diện. Trong kiểm thử thiết kế giao diện người dùng, bạn nên đặc biệt chú ý đến khả năng phù hợp của các mặt thiết kế.

Tính thẩm mỹ, khả năng phục hồi và khả năng tương tác nhất quán ảnh hưởng trực tiếp đến khả năng sử dụng của ứng dụng. Khi các gợi ý không rõ ràng, giao tiếp giữa người dùng và ứng dụng có thể bị phá vỡ.

Rất quan trọng để hiểu rõ mục đích của phần mềm cần được kiểm thử trước khi bắt đầu kiểm thử giao diện người dùng. Hai câu hỏi chính để trả lời là:

1. Ai là người dùng chính của ứng dụng?
2. Phương pháp thiết kế nào đã được sử dụng?

Hãy luôn ghi nhớ rằng giao diện người dùng phục vụ cho người dùng, chứ không phải cho người thiết kế hay lập trình viên. Với tư cách là kiểm thử viên, chúng ta đại diện cho người dùng, như thế chúng ta cần phải biết rõ yêu cầu của họ.

Kiểm thử giao diện người dùng liên quan đến hai loại người dùng chính: (1) người dùng phía trình chủ, và quan trọng hơn, (2) người dùng phía trình khách. Người

dùng phía trình khách thường tương tác với các ứng dụng Web qua trình duyệt Web. Thông thường, người dùng phía trình khách không có kiến thức về kỹ thuật và kiến trúc ứng dụng như là người dùng phía trình chủ trên cùng một hệ thống. Hơn nữa, các chức năng của ứng dụng dành cho người dùng phía trình khách thường khác với các chức năng dành cho người dùng phía trình chủ (thường là những người quản trị hệ thống). Vì thế kiểm thử giao diện phía trình chủ và kiểm thử giao diện phía trình khách nên được đánh giá bởi các chuẩn khác nhau.

Khi tạo ra mô tả sơ lược về người dùng, cần xem xét bốn loại tiêu chuẩn sau cho cả người dùng phía trình chủ và trình khách:

1. Kinh nghiệm về máy tính
2. Kinh nghiệm về Web.
3. Hiểu biết về lĩnh vực.
4. Kinh nghiệm về ứng dụng cụ thể đó.

Xem xét thiết kế

Bước tiếp theo trong chuẩn bị kiểm thử giao diện người dùng là nghiên cứu thiết kế dùng cho ứng dụng. Các loại ứng dụng và người dùng khác nhau yêu cầu các thiết kế khác nhau.

Các chủ đề cần xem xét khi đánh giá một thiết kế

- Phương pháp thiết kế.

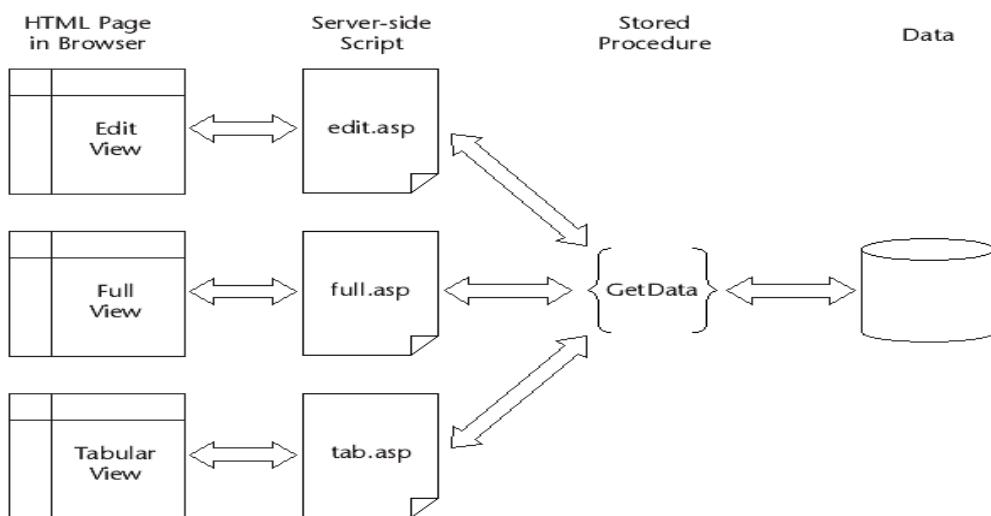
Khi thiết kế giao diện cho người dùng bạn nên đặt ra các câu hỏi cho chính bạn:

- Liệu rằng thiết kế của ứng dụng cần được kiểm thử có thích hợp với người dùng cuối?
 - Giao diện người dùng có trực quan?
 - Thiết kế giao diện có nhất quán trong toàn ứng dụng?
 - Giao diện có đảm bảo người dùng ở trạng thái kiểm soát được?
 - Giao diện có mang lại sự hứng thú?
 - Giao diện có đơn giản, dễ hiểu và dễ sử dụng?
 - Có trợ giúp cho mọi màn hình?
- Tương tác của người dùng.

Người dùng có thể thực thi các dữ liệu khác nhau thao tác bằng tay qua bàn phím và sự kiện chuột. Các cách thao tác dữ liệu bằng tay làm những cách khác nhau trên các điều khiển của màn hình và công nghệ khách nhau, như là cắt và dán và kéo và xóa.

➤ Biểu diễn dữ liệu (đầu ra dữ liệu)

Trong các ứng dụng web, thông tin có thể được truyền thông đến người dùng qua trạng thái khác nhau của các điều khiển UI (như là các menu, các button, check box,...) cái mà có thể được tạo ra trong trang HTML (frames, table, dialog box,...). Phân tích ứng dụng để sưu tầm việc thiết kế kiến trúc thông tin. Một trong hầu hết những cách mà có hiệu quả là làm việc cái này là gấp gỡ nhà phát triển của bạn.



Hình 2.1. Sơ đồ biểu diễn dữ liệu.

Dưới đây là liệt kê các danh sách check về GUI

GUI CHECK LIST

AESTHETIC CHECK (Kiểm tra về giao diện)

- 1 . Kiểm tra màu nền chung của toàn bộ màn hình có được set đúng theo yêu cầu không?
- 2 . Kiểm tra màu chữ, font, font size của tất cả các textbox có set đúng theo yêu cầu không?
- 3 . Kiểm tra background (màu nền) của tất cả các textbox có set đúng theo yêu cầu không?

- 4 . Kiểm tra màu chữ, font, font size của tất cả các label có set đúng theo yêu cầu không?
- 5 . Kiểm tra background (màu nền) của tất cả các label có set đúng theo yêu cầu không?
- 6 . Kiểm tra màu chữ và màu nền của các textbox trong chế độ read-only có được set đúng theo yêu cầu hay không?
- 7 . Kiểm tra tất cả các control trên màn hình có được canh đều hay không? (Label, textbox, checkbox, list , ...)
- 8 . Kiểm tra mặc định tất cả các ký tự chữ và ký tự số đều canh trái. Ngoại trừ trường hợp có yêu cầu cụ thể
- 9 . Kiểm tra mặc định tất cả các số đều canh phải. Ngoại trừ trường hợp có yêu cầu cụ thể.
- 10 . Kiểm tra tất cả các msg thông báo trên màn hình có được viết đúng chính tả hay không?
- 11 . Kiểm tra tất cả các giá trị input chữ hoa hay chữ thường có được hiển thị đúng hay không?
- 12 . Kiểm tra tất cả các textbox có yêu cầu set border hay không?
- 13 . Kiểm tra độ phân giải của màn hình có được set theo đúng chuẩn yêu cầu hay không? VD độ phân giải tối thiểu là 800x600

VALIDATION CHECK (Kiểm tra tính hợp lệ)

Datatype varchar, nvarchar, ntext

- 1 . Kiểm tra maxlen
- 2 . Phân biệt chữ hoa / chữ thường
- 3 . Phân biệt 全角/半角 (tùn giác/bán giác: chỉ áp dụng với Tiếng Nhật, tùng giác thì chữ mập, tròn hơn 2-3bytes; bán giác: chữ ôm 1byte)
- 4 . Phân biệt ký tự unicode
- 5 . Cho phép null hay không
- 6 . Cho phép nhập ký tự đặc biệt hay không?

Datatype (int, tinyint, float)

- 1 . Kiểm tra maxlen.

- 2 . Kiểm tra giá trị max, min
- 3 . Có cho phép nhập ký tự chữ hay không?
- 4 . Cho phép nhập ký tự đặc biệt hay không?
- 5 . Có cho phép nhập ký tự số 2 byte hay không?
- 6 . Cho phép null hay không?
- 7 . Không được phép nhập blank ở vị trí đầu tiên của field số.
- 9 . Không được phép nhập blank ở vị trí cuối cùng của field số.
- 11 . Kiểm tra lỗi chia cho 0
- 12 . Kiểm tra giá trị 0 cho tất cả các tính toán
- 13 . Kiểm tra giá trị trong giới hạn max,min
- 14 . Kiểm tra giá trị = giá trị max, min
- 15 . Kiểm tra giá trị vượt giới hạn giá trị max, min

Datatype (datetime)

- 1 . Kiểm tra maxlength
- 2 . Kiểm tra ngày hợp lệ
- 3 . Có cho phép nhập chữ hay không?
- 4 . Có cho phép nhập ký tự đặc biệt hay không?
- 5 . Có cho phép nhập ký tự số 2 byte hay không?
- 6 . Kiểm tra format theo kiểu nào?
- 7 . Kiểm tra đổi với trường hợp năm nhuận có được tính đúng không?
- 8 . Kiểm tra giá trị 00 và 13 đổi với tháng
- 9 . Kiểm tra giá trị 00 và 32 đổi với ngày
- 10 . Kiểm tra giá trị 28 , 29, 30 -Feb có được tính đúng không?

Datatype(bit)

- 1 . Chỉ được phép nhập 0 hoặc 1
- 2 . Có cho phép null hay không?
- 3 . Kiểm tra nhập ký tự số 2 byte 0 hoặc 1

NAVIGATION CHECK (Kiểm tra phương pháp di chuyển/đuyệt web)

- 1 . Tất cả các trang web/cửa sổ đều có thể truy cập từ menu.
- 2 . Tất cả các cửa sổ đều có thể truy cập từ toolbar
- 3 . Kiểm tra tất cả các màn hình được gọi từ button có được hiển thị đúng hay không?
- 4 . Khi chuyển page trên menu có hiển thị msg xác nhận chuyển trang hay không?
- 5 . Khi chuyển page trên menu có hiển thị msg xác nhận chuyển trang hay không?

USABILITY CHECK: (Kiểm tra tính thân thiện của chương trình)

- 1 . Tất cả các danh sách có được sort hay không? Mặc định là sort theo alphabel. Ngoại trừ trường hợp có yêu cầu sort cụ thể
- 2 . Tất cả các giá trị ngày tháng có được format theo đúng yêu cầu hay không?
- 3 . Tất cả các button trên màn hình có được gán với phím tắt tương ứng hay không?
- 4 . Tất cả các phím tắt được gán có hoạt động đúng hay không?
- 5 . Thứ tự Tab có theo đúng trình tự Top left bottom right hay không? Ngoại trừ trường hợp có yêu cầu set thứ tự riêng biệt.
- 6 . Kiểm tra tất cả các field read-only đều không có thứ tự tab
- 7 . Kiểm tra tất cả các field disable đều không có thứ tự tab
- 8 . Kiểm tra vị trí focus có được đặt ngay field đầu tiên hay control đầu tiên khi load màn hình hay không? . Ngoại trừ có trường hợp yêu cầu set vị trí focus cụ thể
- 9 . Trong trường hợp lỗi input, Khi hiển thị msg lỗi , có focus về vị trí lỗi sau khi đóng cửa sổ thông báo hay không?
- 10 . Trong trường hợp lỗi thao tác, khi hiển thị msg lỗi, có focus về vị trí trước đó sau khi đóng cửa sổ popup hay không?
- 11 . Trong trường hợp gọi cửa sổ popup , sau khi đóng cửa sổ có focus về vị trí trước đó hay không
- 12 . Trong trường hợp chưa đóng cửa sổ popup, thì không được phép focus xuống trang đang xử lý data

DATA INTEGRITY CONDITIONS (Kiểm tra tính ràng buộc dữ liệu)

- 1 . Data có được lưu khi đóng cửa sổ hay không?

2 . Kiểm tra chiều dài tối đa của tất cả các field, và đảm bảo các ký tự đều không bị cắt.

3 . Kiểm tra giá trị max/min đối với ký tự số

10.1.2.2. Kiểm thử thực thi giao diện người dùng

Kiểm thử thực thi giao diện người dùng xem xét ứng dụng về hoạt động của nó. Đánh giá xem các chức năng giao diện người dùng có hoạt động đúng đắn. Nếu một điều khiển giao diện không hoạt động như được thiết kế, nó có thể sẽ thất bại trong việc cho phép truy cập đến các chức năng ở bên trong, trong khi tồn tại độc lập các chức năng đó có thể hoạt động đúng đắn. Kiểm thử chức năng thường được thực hiện đồng thời với kiểm thử thiết kế giao diện người dùng, tuy nhiên nên xem xét hai loại kiểm thử này.

Ranh giới giữa tính nhất quán của thiết kế và chức năng của thiết kế không phải luôn luôn rõ ràng. Ví dụ, một liên kết siêu văn bản có một màu sắc nào đó được duy trì nhất quán từ màn hình này sang màn hình khác, trong khi nền mà trên đó liên kết văn bản được hiển thị thay đổi. Bởi vì nền thay đổi, liên kết văn bản trở nên rất khó đọc trên một số màn hình. Mặc dù liên kết văn bản là nhất quán trong ví dụ này, màu của liên kết văn bản nên được điều chỉnh để cải thiện khả năng dễ đọc.

Các phần tử giao diện người dùng

Bảng dưới đây liệt kê một số phần tử giao diện người dùng cần được kiểm thử

Loại phần tử	Các vấn đề cần xem xét
Kiểu chữ (fonts)	<ul style="list-style-type: none"> - Sự nhất quán của hình thức thể hiện - Tính dễ đọc của chữ - Khó đọc kiểu chữ nghiêng và có chân - Hiển thị lộn xộn do nhiều kiểu chữ
Màu sắc (color)	<ul style="list-style-type: none"> - Khả năng thích hợp của màu nền sau (background)

	<ul style="list-style-type: none"> - Khả năng thích hợp của màu nền trước (foreground) - Khả năng thích hợp của màu kiểu chữ. - Sử dụng màu lộn xộn.
Đường viền (border)	<ul style="list-style-type: none"> - Hiệu ứng ba chiều trên các nút lệnh có thể là những gợi ý trực quan hiệu quả đối với người dùng. - Sử dụng hiệu ứng ba chiều trên các phần tử không tương tác có thể dẫn đến sự khó hiểu.
Hình ảnh (imager)	<ul style="list-style-type: none"> - Các hình ảnh lớn có thể làm tăng thời gian tải. - Phù hợp với nền. - Tính dễ đọc các nhãn. - Tính dễ đọc các nút. - Kích thước hình ảnh phù hợp.
Khung (frames)	<ul style="list-style-type: none"> - Một số trình duyệt cũ không thể hiển thị các khung. - Các cài đặt hiển thị và loại trình duyệt có thể ảnh hưởng đến sự hiển thị các khung. - Sử dụng các nút quay lui thường dẫn đến những kết quả không mong muốn.
Bảng (tables)	<ul style="list-style-type: none"> - Các bảng lồng nhau làm chậm thời gian nạp HTML. - Kiểm thử nên thực hiện trên tất cả các trình duyệt, cài đặt màn hình, và kích thước cửa sổ trình duyệt.

Bảng 2.1. Các phần tử giao diện người dùng.

10.1.3. Một số chú ý khi kiểm thử giao diện

1. Khi kiểm thử giao diện bạn nên kiểm thử trên nhiều trình duyệt để đảm bảo sự tin cậy.

2. Kiểm thử giao diện cần và quan trọng nhất là dựa trên bản đặc tả.
3. Quan tâm đến yêu cầu mong muốn của khách hàng.

10.2. Kiểm thử chức năng

10.2.1. Khái niệm kiểm thử chức năng

Kiểm thử chức năng là kiểm tra xem các thành phần chức năng của hệ thống hoạt động có đúng đắn hay không, có đúng với mục đích của người sử dụng hay không, đảm bảo cho các thông số kỹ thuật, các dữ liệu đầu ra, đầu vào, chức năng trong hệ thống hoạt động đúng qui tắc nghiệp vụ và chính xác.

10.2.2. Mục đích của test chức năng

Đảm bảo mục tiêu đúng đắn của từng chức năng, của mỗi ứng dụng bao gồm định hướng, dữ liệu đầu vào, xử lý và dữ liệu nhận được.

10.2.3. Các vấn đề liên quan đến kiểm thử chức năng

10.2.3.1. Các phương pháp kiểm thử

- a. Kiểm thử đơn giản chấp nhận chức năng

Function acceptance simple tests (FASTs) tượng trưng cho mức thứ hai của kiểm thử chấp nhận. FAST thực thi mức thấp nhất của chức năng cho mỗi một lệnh của chương trình. Kết hợp các chức năng tuy nhiên không tích hợp không là trong phạm vi của *FAST*.

Một trong những mục tiêu của kiểm thử FAST là kiểm tra hành vi thích hợp của các điều khiển giao diện (ví dụ như textbox, pull-down list, radio button...) dựa trên thiết kế. Kiểm thử FAST đòi hỏi phải kiểm tra sự tồn tại của các điều khiển giao diện trên mỗi trang, mỗi cửa sổ hay mỗi hộp thoại; kiểm tra trạng thái mặc định (ví dụ như được phép chỉnh sửa (enable), không được phép chỉnh sửa (disable), được tô sáng (highliged)...) kiểm tra giá trị mặc định hay chế độ mặc định; kiểm tra thứ tự tab, kiểm tra hành vi của các phím tắt như (Ctrl -V, Ctrl-X...) và các phím truy cập khác như (Alt -O...). Một khía cạnh khác trong tiến trình này bạn hiểu được logic thực hiện của người phát triển khi xây dựng các chức năng cho người dùng cuối. Sau này, bạn sử dụng những thông tin này để thiết kế các kiểm thử nhằm phát hiện các sai sót về lô-gic thiết kế.

Trong môi trường web, thứ để kiểm tra cho FAST bao gồm:

- Các Link như nội dung link, thumbnail link, bitmap link và image map links
- Các điều khiển có bản như là các điều hướng backward(lùi về phía sau), forward(tiến về phía trước), zoom in(phóng to), zoom out(thu nhỏ), các điều khiển UI khác và các kiểm tra việc làm mới nội dung.
- Kiểm tra hành động của các lệnh như là thêm, gỡ bỏ, cập nhật và các loại khác của việc submit dữ liệu; kiểm tra tạo thuộc tính người dùng hay tài khoản người dùng bao gồm tài khoản email, tài khoản người dùng, và nhóm các tài khoản cơ bản và dữ liệu vào.
- Các chức năng quan trọng khác như là đăng nhập, đăng xuất, khai báo email, tìm kiếm, xác nhận thẻ tín dụng và sử lý, hay nắm giữ việc quên pass.

b. Kiểm thử chức năng hướng tác vụ

Kiểm thử chức năng hướng tác vụ (task-oriented functional testing -TOFT) kiểm tra xem ứng dụng có thể thực hiện các công việc hữu ích một cách đúng đắn hay không. Kiểm thử chức năng hướng tác vụ là những kiểm thử “tích cực” nhằm kiểm tra các chức năng của chương trình bằng cách so sánh kết quả của các công việc được thực hiện với tài liệu đặc tả sản phẩm và đặc tả yêu cầu, nếu có hoặc so sánh với mong đợi hợp lý của người dùng. Kiểm thử chức năng hướng tác vụ cũng kiểm tra độ chính xác và toàn vẹn của mỗi tác vụ riêng rẽ mà mỗi chương trình thực hiện. Nếu hành vi hay kết quả khác với đặc tả trong yêu cầu sản phẩm thì sẽ báo cáo lỗi.

c. Kiểm thử lỗi ép buộc

Kiểm thử lỗi ép buộc (forced-error tests -FET) cố ý tạo ra những điều kiện lỗi phần mềm . Mục tiêu của FET là tìm các điều kiện lỗi không được phát hiện hoặc bị xử lý sai. Các điều kiện lỗi cần được xử lý một cách hợp lý; nghĩa là ứng dụng phục hồi thành công, hệ thống phục hồi thành công, hay ứng dụng thoát mà không làm hỏng dữ liệu và vẫn có cơ hội lưu giữ các công việc đang làm.

Giả sử rằng bạn đang kiểm tra các trường hợp văn bản trong một mẫu đăng ký trực tuyến và đặc tả chương trình không cho phép nhập vào các ký hiệu không phải ký

tự vào trong trường Name. Một điều kiện lỗi sẽ được sinh ra trong trường hợp bạn nhập vào “123456” (hoặc các chuỗi ký tự bất kỳ không phải dạng ký tự) và chọn nút Submit. Với bất cứ điều kiện hợp lệ nào cũng luôn có một điều kiện không hợp lệ.

Một số cách tạo danh sách điều kiện lỗi:

- Thu thập danh sách các thông điệp lỗi từ các lập trình viên.
- Khảo sát dữ liệu chuỗi các ký tự trong tệp nguồn.
- Thu thập thông tin từ đặc tả.
- Sử dụng kinh nghiệm.
- Sử dụng ma trận kiểm thử đầu vào hợp lệ, không hợp lệ chuẩn

Khi bạn có được danh sách các điều kiện lỗi, mỗi điều kiện lỗi nên được thực thi theo các tiến trình kiểm thử sau:

1. Ép buộc điều kiện lỗi.
 2. Kiểm tra lô-gic phát hiện lỗi.
 3. Kiểm tra lô-gic xử lý
 4. Kiểm tra giao tiếp lỗi.
 5. Tìm kiếm các vấn đề liên quan.
- d. Kiểm tra điều kiện biên và phân tích lớp tương đương

Kiểm thử điều kiện biên (boundary condition test) tương tự như kiểm thử ép buộc lỗi (FET) trong đó các biên của mỗi biến được kiểm thử. Ví dụ khi kiểm thử một mẫu đăng ký trực tuyến, bạn cần kiểm tra một trường văn bản với một giới hạn từ hai đến bảy ký tự có hoạt động như mong đợi hay không. Trường đó có chấp nhận hai, ba sáu và bảy ký tự không? Nếu một hay tám ký tự có được không? Trường đó có thể để trống không.

- e. Kiểm thử dạng khám phá

Kiểm thử dạng khám phá (exploratory testing) là tiến trình đánh giá và quan sát hành vi của sản phẩm, cũng như giả thiết về hành vi của nó. Kiểm thử dạng khám phá bao gồm thực thi các ca kiểm thử và tạo ra các ca kiểm thử mới nhờ thông tin thu thập được từ kết quả của các kiểm thử trước đó. Thực thi kiểm thử bao gồm cài đặt môi trường, tạo ra các dữ liệu vào tấn công chương trình, quan sát kết quả của chương trình, đánh giá những gì học được và sau đó bắt đầu kiểm thử tiếp theo.

Bạn có thể thực hiện kiểm thử dạng khám phá bằng cách xem xét toàn bộ chương trình, tìm hiểu xem chương trình hoạt động như thế nào và kiểm thử nó. Dạng kiểm thử này được gọi là *kiểm thử dạng khám phá*, bởi vì bạn thực hiện khám phá một chương trình. Ngược lại với kiểm thử dạng khám phá là kiểm thử được chuẩn bị trước.

10.2.3.2. Các kỹ thuật kiểm thử chức năng

a. Kiểm thử hộp đen

➤ Phân lớp tương đương

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

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

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

1. Giảm thiểu số lượng các ca kiểm thử khác mà phải được phát triển để hoàn thành mục tiêu đã định của kiểm thử “hợp lý”.
2. Bao phủ một tập rất lớn các ca kiểm thử có thể khác. Tức là, nó nói cho chúng ta một thứ gì đó về sự có mặt hay vắng mặt của những lỗi qua tập giá trị đầu vào cụ thể.

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

- (1). Xác định các lớp tương đương và
- (2). Xác định các ca kiểm thử.

Xác định các lớp tương đương

Các lớp tương đương được xác định bằng cách lấy mỗi trạng thái đầu vào (thường là một câu hay một cụm từ trong đặc tả) và phân chia nó thành hai hay nhiều nhóm.

Điều kiện bên ngoài	Lớp tương đương hợp lệ	Lớp tương đương không hợp lệ
---------------------	------------------------	------------------------------

--	--	--

Hình 2.2. Một mẫu cho việc liệt kê các lớp tương đương

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

1. Nếu một trạng thái đầu vào định rõ giới hạn của các giá trị, xác định một lớp tương đương hợp lệ và hai lớp tương đương không hợp lệ.
2. Nếu một trạng thái đầu vào xác định số giá trị, xác định một lớp tương đương hợp lệ và hai lớp tương đương bất hợp lệ.
3. Nếu một trạng thái đầu vào chỉ định tập các giá trị đầu vào và chương trình sử dụng mỗi giá trị là khác nhau, xác định một lớp tương đương hợp lệ cho mỗi loại và một lớp tương đương không hợp lệ.
4. Nếu một trạng thái đầu vào chỉ định một tình huống “chắc chắn – must be”, xác định một lớp tương đương hợp lệ và một lớp tương đương không hợp lệ.

Xác định các ca kiểm thử

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

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

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

Mặc dù việc phân lớp tương đương là rất tốt khi lựa chọn ngẫu nhiên các ca kiểm thử, nhưng nó vẫn có những thiếu sót. Ví dụ, nó bỏ qua các kiểu testcase có lợi nào đó.

➤ Phân tích giá trị biên

Các điều kiện biên là những điều kiện mà các tình huống ngay tại, trên và dưới các cạnh của các lớp tương đương đầu vào và các lớp tương đương đầu ra. Phân tích các giá trị biên là phương pháp thiết kế ca kiểm thử bổ sung thêm cho phân lớp tương đương, nhưng khác với phân lớp tương đương ở hai khía cạnh:

- Từ mỗi lớp tương đương, phân hoạch tương đương sẽ chọn phần tử bất kỳ làm phần tử đại diện, trong khi việc phân tích giá trị biên sử dụng một hoặc một số phần tử. Như vậy, mỗi biên của lớp tương đương chính là đích kiểm thử.
- Không chỉ chú ý tập trung vào những điều kiện đầu vào, các trường hợp kiểm thử cũng được suy ra từ việc xem xét các kết quả ra (tức các lớp tương đương đầu ra).

Rất khó có thể có thể liệt kê hết các hướng dẫn cụ thể cho các trường hợp. Tuy nhiên, cũng có một số nguyên tắc phân tích giá trị biên như sau:

1. Nếu điều kiện đầu vào xác định một khoảng giá trị giữa a và b, các trường hợp kiểm thử sẽ được thiết kế với giá trị a và b, và các giá trị sát trên và sát dưới a và b.
2. Nếu một điều kiện đầu vào xác định một số các giá trị, các trường hợp kiểm thử sẽ được phát triển để thực hiện tại các giá trị cực đại, cực tiểu. Các giá trị sát trên và dưới giá trị cực đại, cực tiểu cũng được kiểm thử.
3. Nguyên tắc 1 và 2 được áp dụng cho các điều kiện đầu ra.
4. Nếu cấu trúc dữ liệu chương trình bên trong được qui định các biên (chẳng hạn, mảng được định nghĩa giới hạn 100 mục), tập trung thiết kế trường hợp kiểm thử để thực thi cấu trúc dữ liệu tại biên của nó.

Ngoài ra, người kiểm thử có thể sử dụng sự xét đoán và sáng tạo của mình để tìm các điều kiện biên.

Tóm lại, chúng ta phải kiểm thử mỗi biên của một lớp tương đương về tất cả các phía. Một chương trình nếu vượt qua những trường hợp kiểm thử đó có thể vượt qua các kiểm thử khác từ lớp đó.

b. Kiểm thử hộp xám

Kiểm thử hộp xám là sự kết hợp giữa kiểm thử hộp đen và kiểm thử hộp trắng. Mục đích của việc kiểm thử này là tìm ra những nhược điểm có liên quan tới lỗi thiết kế, hoặc lỗi thi hành của hệ thống.

Trong kiểm thử hộp xám, người kiểm thử cần phải có những hiểu biết về hệ thống và thiết kế các testcase hoặc dữ liệu kiểm thử dựa trên những kiến thức về hệ thống.

10.2.4. Một số công cụ kiểm thử

Giới thiệu về Quicktest Professional (QTP)

Phần mềm Quick Test Pro là phần mềm kiểm soát việc kiểm thử tự động những chức năng của một sản phẩm phần mềm khác. Phần mềm QuickTest Pro là một bộ phận (module) của hệ thống Mercury Quality Center bao gồm nhiều module phần mềm phối hợp với nhau để quản lý toàn bộ quy trình đảm bảo chất lượng sản phẩm phần mềm. Trước đây, do hãng Mercury (www.mercury.com) phát hành. Sau này, tập đoàn HP đã mua lại toàn bộ hãng Mercury, nên tên gọi của nó đầy đủ là: HP Mercury QuickTest Pro.

*Hình 2.3. Giao diện QTP*

QTP là công cụ kiểm thử dùng để kiểm tra chức năng (functional test) và cho phép thực hiện kiểm tra hồi qui (regression test) một cách tự động. Đây cũng là công cụ áp dụng phương pháp Keyword-Driven, một kỹ thuật scripting (lập trình trong kiểm thử tự động) hiện đại, cho phép kiểm thử viên bổ sung testcase bằng cách tạo file mô tả cho nó mà không cần phải chỉnh sửa hay bổ sung bất cứ script nào cả. Nó cũng phù hợp trong tình huống chuyển giao công việc mà người mới tiếp nhận chưa có thời gian hoặc không hiểu script vẫn có thể thực hiện kiểm tra phần mềm theo đúng yêu cầu.

10.3. Kiểm thử cấu hình và khả năng tương thích

10.3.1. Khái niệm kiểm thử cấu hình và khả năng tương thích

Kiểm thử cấu hình: Là kiểm tra hoạt động của phần mềm với tất cả các máy tính.

Kiểm thử khả năng tương thích: Là kiểm tra xem phần mềm có tương tác và chia sẻ thông tin chính xác với các phần mềm khác nhau hay không. Sự tương tác này có thể xảy ra giữa hai chương trình trên cùng máy tính, hoặc trên các máy tính khác nhau được kết nối Internet cách nhau tới hàng nghìn dặm.

Ví dụ: Nâng cấp một chương trình cơ sở dữ liệu mới và có tất cả các cơ sở dữ liệu đang tồn tại đã load về và làm việc như chúng đã làm việc với chương trình cũ.

10.3.2. Mục đích

Kiểm thử cấu hình hệ thống Web gồm kiểm thử các cài đặt phần mềm và phần cứng của các trình chủ được hỗ trợ khác nhau, các thiết lập cấu hình trình duyệt, các kết nối mạng, các cài đặt TCP/IP stack... Mục đích là để đảm bảo rằng ứng dụng có thể được thực thi với nhiều cấu hình nhất. Với mỗi cấu hình được hỗ trợ, hy vọng rằng ứng dụng hoạt động như người dùng mong đợi.

10.3.3. Các vấn đề liên quan đến kiểm thử cấu hình và khả năng tương thích

Kiểm thử cấu hình và khả năng tương thích đòi hỏi chi phí nhân lực rất cao đối với các hệ thống Web, so với các hệ thống trên máy tính độc lập, bởi vì các hệ thống web phát triển dựa trên kiến trúc các thành phần và kiến trúc phân tán. Kiểm thử cấu hình và khả năng tương thích của một hệ thống Web cần xem xét các trình chủ, cơ sở dữ liệu, các trình duyệt và các thiết bị kết nối. Kiểm thử cấu hình và khả năng tương thích thường đề cập đến các vấn đề:

Phía trình chủ bao gồm: trình chủ ứng dụng, trình chủ Web, trình chủ cơ sở dữ liệu, tường lửa, hệ điều hành, phần cứng.

Phía trình khách bao gồm: loại trình duyệt và phiên bản, hệ điều hành, tường lửa, phần mềm ngăn chặn truy cập các nội dung xấu, phần cứng phía trình khách như máy in cục bộ, video, thiết bị lưu trữ

Các thiết bị và kết nối mạng bao gồm: thiết bị cầu nối (bridges), thiết bị định tuyến (routers), cổng ra vào (gateway)..., Internet/intranet, 10/100 Base –T, modem, ISDN...

10.3.3.1. Khi nào thực thi kiểm thử cấu hình và khả năng tương thích

Kiểm thử cấu hình và khả năng tương thích nên được thực thi ngay sau khi bước đầu tiên của kiểm thử chức năng kết thúc, và tốt nhất là ngay sau khi nhiều lỗi chức năng được phát hiện. Nếu không có thể mất nhiều thời gian và khó khăn để phân biệt lỗi cấu hình và lỗi chức năng. Nếu có những vùng có vấn đề nghi ngờ, kiểm thử cấu hình có thể thực thi trong giai đoạn kiểm thử chức năng.

10.3.3.2. So sánh kiểm thử cấu hình với kiểm thử khả năng tương thích

Ranh giới giữa kiểm thử cấu hình và khả năng tương thích thường bị hiểu sai. Kiểm thử cấu hình được thiết kế để phát hiện các lỗi liên quan đến sự kết hợp phần cứng và phần mềm khác nhau, kiểm thử khả năng tương thích sau đó sẽ xác định một ứng dụng, với các cấu hình được hỗ trợ, hoạt động như mong đợi với các kết hợp khác nhau của phiên bản phần cứng và phần mềm.

Ví dụ, kiểm thử cấu hình có thể kiểm tra một hệ thống Web cài đặt trên máy tính có bộ vi xử lý kép (dual-processor) hoạt động đúng đắn; kiểm thử khả năng tương thích sau đó xác định với những nhà sản xuất nào và những nhãn hiệu máy chủ nào, với cùng một cấu hình, là tương thích với hệ thống Web.

10.3.3.3. Một số vấn đề có thể gặp trong quá trình kiểm thử

Các cấu hình trình chủ

Ở mức vĩ mô, một hệ thống Web có thể hoạt động đúng đắn với các cấu hình trình chủ trong khi không hoạt động đúng đắn với các cấu hình khác. Bất kỳ lỗi sinh ra bởi thay đổi sự phân tán các trình chủ nên được xem là lỗi cấu hình.

Một hệ thống điển hình cần được kiểm thử có thể sử dụng các thành phần sau:

- Trình chủ ứng dụng.
- Trình chủ cơ sở dữ liệu.
- Trình chủ email.
- Trình chủ Web.
- Trình chủ proxy.

Các lỗi phía trình khách:

Phía trình khách của các hệ thống Web có thể phải chịu trách nhiệm về một số loại lỗi cấu hình và khả năng tương thích. Để xác định các kiểm thử phía trình khách cần thiết cho một ứng dụng Web, cần xác định các thành phần xây dựng ứng dụng Web phía trình khách.

Hệ thống cần được kiểm thử có thể gồm các thành phần phía trình khách sau:

- *Các hệ điều hành:* Windows (98, 2000, NT), Linux, Unix, một số hệ điều hành khác cho thiết bị di động.
- *Các thành phần giao tiếp:* Trình duyệt, email, chat, ftp.

- *Plug -Ins:* QuickTime, RealPlayer, Flash, Windows Media Player.
- *Kết nối:* Thuê bao đường truyền, ISDN, DSL.
- *Phần cứng:* Nhà sản xuất, CPU, Ram, Card đồ họa, Card video, màn hình, thiết bị máy in, thiết bị vào (chuột, bàn phím...), Card mạng.

Ngoài ra còn có các ứng dụng thực thi đồng thời, phần mềm mạng, các dịch vụ trực tuyến, các chương trình mở rộng chạy thường trú.

Trình duyệt Web

Các trình duyệt Web là các thành phần trung tâm phía trình khách của hầu hết các ứng dụng Web. Một trình duyệt hoạt động như một lớp vỏ ứng dụng mà trong đó ứng dụng Web thực thi. Hành vi của các trình duyệt khác nhau có thể khác nhau, chúng có thể hỗ trợ các lệnh Java, các cài đặt thiết lập bảo mật, và các tính năng khác. Mỗi trình duyệt có thể có một hoặc nhiều trình thông dịch- mà có thể không tương thích với các trình duyệt khác.

Việc thiết lập cấu hình trình duyệt rất đa dạng cho mỗi trình duyệt càng làm phức tạp thêm vấn đề. Kiểm thử trình duyệt và thiết lập cấu hình trình duyệt nên đưa vào trong giai đoạn kiểm thử khả năng tương thích, thay vì đưa vào trong giai đoạn kiểm thử chức năng.

10.3.4. Kết luận

Ngày nay, nhiều chương trình được thiết kế cho nhiều hệ điều hành khác nhau, ví dụ, bạn đang test một chương trình, bạn nên test nó với tất cả các hệ điều hành mà nó được thiết kế để chạy trên đó. Các chương trình được thiết kế để hoạt động trên một web browser yêu cầu sự chú ý đặc biệt, vì có nhiều web browser hiện hữu khác nhau và chức năng hoạt động của chúng không giống nhau.Thêm vào đó, một web browser giống nhau lại hoạt động khác nhau trên các hệ điều hành khác nhau, và mỗi web browser lại có nhiều phiên bản khác nhau.

10.4. Kiểm thử hiệu năng

10.4.1. Khái niệm

Performance testing là một dạng kiểm tra hiệu suất trong đó thời gian phản hồi, tỷ lệ giao dịch và các yêu cầu phụ thuộc thời gian khác được đo đạc và đánh giá.

Performance testing là kiểm tra các yêu cầu về hiệu suất có đạt được hay không. Performance testing là tiến hành và thực hiện để mô tả sơ lược và điều chỉnh các hành vi hiệu suất của mục tiêu test như một hàm của các điều kiện. Ví dụ workload hoặc cấu hình phần cứng.

Load testing sử dụng kiểm thử tải để xác minh hành vi của ứng dụng trong điều kiện tải trọng bình thường và cao điểm. Điều này cho phép bạn kiểm tra xem ứng dụng của bạn có thể đáp ứng mục tiêu thực hiện mong muốn của bạn. Nó cho phép bạn đo thời gian đáp ứng, mức sử dụng tài nguyên, và để xác định điểm phá vỡ ứng dụng của bạn giả định rằng vi phạm điểm xảy ra dưới các điều kiện phụ tải định. Kiểm thử có thể thực hiện trên cả hai mặt về khối lượng hoạt động và thời gian hoạt động.

Stress testing sử dụng kiểm thử quá tải để ước lượng hành vi ứng dụng của bạn khi nó được đẩy vượt quá điều kiện bình thường hay tới việc tải cao điểm. Mục đích của kiểm thử quá tải là phát hiện lỗi của ứng dụng được phơi bày ra chỉ trong điều kiện cao điểm. Có thể bao gồm những thứ như các vấn đề về đồng bộ hóa, điều kiện tốc độ và sự rò rỉ về bộ nhớ.

Capacity testing hay Volume testing, kiểm thử công xuất là sự bổ sung cho kiểm thử tải và nó xác định điểm cuối cùng thất bại của máy chủ. Ví dụ để phù hợp với tải trọng trong tương lai bạn cần biết có bao nhiêu nguồn được thêm vào (như CPU, RAM, dung lượng ổ cứng hay băng thông mạng) là cần thiết để hỗ trợ mức độ sử dụng trong tương lai. Kiểm thử công suất giúp bạn xác định chiến lược sử dụng quy mô nên được thêm ra hay mở rộng ra.

10.4.2. Các yếu tố quan trọng của kiểm thử hiệu năng

Kiểm thử hiệu năng liên quan đến ba thành phần chính:

- Sức tải công việc (workload)
- Môi trường của hệ thống và nguồn tài nguyên có sẵn
- Thời gian đáp ứng của hệ thống

Sức tải của công việc là số lượng người sử dụng được dự đoán.

Không đơn giản để ước lượng số người sử dụng mà một hệ thống sẽ hỗ trợ vì các hoạt động của người sử dụng có thể thay đổi, cũng như có thể thay đổi thời gian

truy cập và tần suất của các hoạt động. Biểu diễn sức tải công việc bởi số người sử dụng dự đoán cần phải được tính toán cẩn thận, cũng như sự mô phỏng những người sử dụng thực sự.

Xác định mức hiệu năng có thể chấp nhận được đối với một hệ thống cần thông tin đầu vào từ các nhóm tiếp thị và quản lý. Hiệu năng sẽ được lường như thế nào, nó sẽ cần chi phí bao nhiêu, các công cụ nào sẽ được sử dụng, và các đơn vị đo lường được sử dụng là những yếu tố cần được xem xét.

Có ba thành phần cơ bản biểu diễn nguồn tài nguyên liên quan trong bất kỳ giao tác trực tuyến: (1) trình duyệt phía trình khách, (2) mạng và (3) trình chủ. Bởi vì các ứng dụng web điển hình gồm nhiều thành phần phần cứng, phần mềm tương tác, nếu một trong những thành phần này không hoạt động hoặc hoạt động không tốt đều có thể ảnh hưởng đến hiệu năng, do đó cần định nghĩa rõ ràng môi trường và nguồn tài nguyên cần được kiểm tra thử. Môi trường và nguồn tài nguyên liên quan trong một ứng dụng web bao gồm một số hoặc tất cả các thành phần sau:

- Các biến truy cập mạng. Ví dụ 28,2 kilôbit/giây, DSL, T1,...
- Các biến về nhân khẩu học. ví dụ một web site được sử dụng chủ yếu bởi những người sử dụng có kinh nghiệm có thể được sử dụng thường xuyên hơn chức năng tìm kiếm so với những người sử dụng chưa có kinh nghiệm.

Trình duyệt	Mạng	Trình chủ
Nguồn tài nguyên làm trì trệ hiệu năng	Nguồn tài nguyên điển hình làm trì trệ hiệu năng	Nguồn tài nguyên điển hình làm trì trệ hiệu năng
Thời gian CPU	Độ trễ do các thiết bị và hàng đợi dữ liệu của mạng	Thời gian CPU
	Băng thông hay tốc độ truyền dữ liệu	Thời gian truy cập vào ra, bus vào ra điều khiển đĩa, truy cập đĩa.
Các hoạt động điển hình	Các hoạt động điển hình	Các hoạt động điển hình
Nhận/ gửi dữ liệu	Các gói dữ liệu đi từ trình khách đến trình chủ	Nhận các yêu cầu

Định dạng dữ liệu	Các gói dữ liệu đi từ trình chủ đến trình chủ	Chạy các script hàm thư viện, thủ tục lưu trữ, các tệp thực thi.
Hiển thị dữ liệu	Các gói dữ liệu đi từ trình chủ đến trình khách.	
Thực thi các script và nội dung động.		

Bảng 2.2. Môi trường hệ thống và nguồn tài nguyên

- Các biến về địa lý. Ví dụ, nếu trình duyệt ở California và các yêu cầu của người sử dụng đến từ London, tốc độ đến của các yêu cầu sẽ chậm hơn cho dù dữ liệu dữ liệu được yêu cầu giống nhau.
- Các biến hạ tầng ISP. Ví dụ, tốc độ truyền dữ liệu của mỗi ISP.
- Cấu hình của trình khách. Ví dụ, hệ điều hành, trình duyệt,...
- Cấu hình của trình chủ. Ví dụ tốc độ CPU, bộ nhớ, phần mềm,..

10.4.3. Ba giai đoạn của kiểm thử hiệu năng

Tiến trình kiểm thử hiệu năng có thể chia làm bao giai đoạn: lập kế hoạch, kiểm thử và phân tích.

1) Giai đoạn lập kế hoạch:

- Định nghĩa các mục tiêu và kết quả, xác định mong đợi.
- Thu thập yêu cầu kiểm thử và hệ thống.
- Định nghĩa sức tải.
- Chọn đơn vị đo hiệu năng
- Xác định các kiểm thử cần thực thi và thiết kế kịch bản người sử dụng và tạo các script kiểm thử

2) Giai đoạn kiểm thử:

- Chuẩn bị sẵn sàng (nghĩa là cài đặt môi trường kiểm thử (test bed) và các trình giám sát hiệu năng trên mạng và trình chủ)
- Thực thi các kiểm thử
- Thu thập dữ liệu

3) Giai đoạn phân tích

- Phân tích kết quả
- Thay đổi hệ thống để tối ưu hiệu năng
- Thiết kế các kiểm thử mới

Thiết lập các mục tiêu, các mong đợi và định nghĩa các sản phẩm kết quả

Dưới góc độ của nhà quản lý, các vấn đề và xem xét phổ biến trước, trong và sau khi lập kế hoạch và thực thi kiểm thử hiệu năng gồm:

- Ứng dụng Web sẽ có thể hỗ trợ số lượng người sử dụng dự kiến trong khi vẫn giữ được hiệu năng chấp nhận được và ở mức chi phí nào?
- Ở điểm nào khả năng xử lý sức tải của ứng dụng web sẽ bắt đầu suy giảm?
- Sẽ có những ảnh hưởng tài chính nào kéo theo do sự suy giảm hiệu năng?
- Có thể làm gì để tăng xử lý sức tải của ứng dụng web vào ở mức chi phí nào?
- Hệ thống có thể được giám sát như thế nào sau khi triển khai sao cho các hành động thích hợp có thể được thực hiện trước khi nguồn tài nguyên hệ thống bị bão hòa.

Để cho tiến trình kiểm thử thành công, rất quan trọng để chuẩn bị các mong đợi của tổ chức và những nhà quản lý. Bạn cần hiểu yêu cầu kiểm thử và phạm vi của những gì bạn được yêu cầu kiểm thử. Để thực hiện điều đó, cần định nghĩa hay ít nhất đáp ứng các câu hỏi sau:

- Mục đích của kiểm thử hiệu năng là gì?
- Ai cấp cho bạn ngân sách để lập kế hoạch và thực hiện kiểm thử hiệu năng?
- Tại sao đo lường? Bạn đã dự thảo ngân sách về nguồn tài nguyên và thời gian phù hợp cho phép nhóm phát triển hướng đến các vấn đề về hiệu năng và cho phép bạn kiểm thử lại hệ thống?

10.4.4. Tìm hiểu các loại kiểm thử hiệu năng thường sử dụng

Để đánh giá khả năng hỗ trợ của người sử dụng, ba loại kiểm thử thường được thực hiện: (1) kiểm thử hiệu năng (perfomance), (2) kiểm thử tải (load), và (3) kiểm thử quá tải (stress). Mặc dù ba thuật ngữ này thường được sử dụng thay thế lẫn nhau, mỗi loại biểu diễn một kiểm thử được thiết kế nhằm hướng đến mục tiêu khác nhau:

	Performance Test (PT)	Load Test (LT)	Stress Test (ST)
Bản chất	<ul style="list-style-type: none"> - Được thực hiện nhằm xác định tốc độ, khả năng phân tái và mức độ tin tưởng của PM trong môi trường nhiều người dùng, có nhiều hoạt động khác nhau. - Dùng công cụ KTTĐ để kiểm tra hiệu năng PM ở điều kiện có sự điều chỉnh về mức độ tải. 	<ul style="list-style-type: none"> - Là một phần trong qui trình thực hiện PT. Load Test đôi khi còn gọi là Volume Test. - Dùng công cụ KTTĐ để kiểm tra PM ở điều kiện liên tục tăng mức độ chịu tải. Tuy nhiên mức độ chịu tải cao nhất vẫn ở mức chức năng PM hoạt động đúng chức năng. 	<ul style="list-style-type: none"> - Đây là cách thực hiện nhằm làm cho PM không còn chạy được nữa. - Phương pháp này rất hay áp dụng để kiểm tra PM và phản ứng.
Mục tiêu	<ul style="list-style-type: none"> - Tìm ra điểm “thắt cổ chai” của PM và từ đó có những cải tiến để tăng khả năng hoạt động của PM. - Đề ra các thông số, tiêu chuẩn về hiệu năng thực thi của PM. Một số thông số đó là: số phiên làm việc, thời gian xử lý của phiên làm việc, và các tài nguyên khác bị chiếm giữ. 	<p>Khi PM không còn khả năng cải tiến. Ở mức chịu tải cao nhất:</p> <ul style="list-style-type: none"> - Giám sát việc PM quản lý tài nguyên khi chạy trong thời gian dài. - Giám sát thông số đề ra trong PT như thời gian xử lý,... khi chạy trong thời gian dài. 	<ul style="list-style-type: none"> - Kiểm tra khả năng phục hồi của PM sau khi có sự cố. - Kiểm tra khả năng chịu tải cho một máy khác khi máy đó gặp sự cố do không có khả năng chịu tải được nữa.
Ví dụ	<p>Ví dụ 1: Có ứng dụng web, yêu cầu cần tìm thông số về hiệu năng thực thi của ứng dụng.</p> <p>Dùng LoadRunner tạo tình huống khởi đầu có 10 người dùng, cứ 2 phút tăng</p>	<p>Ví dụ 2: Ứng dụng web chỉ hoạt động tốt với tối đa là 1000 người dùng. Yêu cầu kiểm tra khả năng của web khi hoạt động ở ngưỡng đáp ứng với thời gian dài 2 ngày.</p> <p>Dùng LoadRunner tạo</p>	<p>Ví dụ 3: Ứng dụng web chỉ có thể quản lý, đáp ứng tối đa 1000 yêu cầu. Yêu cầu cần kiểm tra ứng xử của web khi gặp sự cố quá ngưỡng số người sử dụng.</p> <p>Dùng LoadRunner tạo</p>

<p>thêm 10 người, tăng tối đa là 2000 người.</p> <p>Quan sát: Biểu đồ thời gian đáp ứng với kết quả xử lý đúng và kết quả sai, có bao nhiêu yêu cầu không được xử lý, tài nguyên sử dụng như RAM, CPU,...</p> <p>Thông qua đó giúp xác định ứng dụng hoạt động tốt nhất trong điều kiện nào.</p>	<p>tình huống khởi đầu có 800 người dùng, cứ 1 phút tăng 2 người, tăng tối đa 1000 người, và giữ ở mức đó tiếp tục chạy trong vòng 2 ngày.</p> <p>Quan sát: Tài nguyên sử dụng như RAM, CPU, thời gian đáp ứng với kết quả đúng và sai khi ứng dụng chịu tải ở mức cao nhất và hoạt động trong thời gian dài.</p>	<p>tình huống có 1100 người truy cập, khi đạt đến ngưỡng đó LoadRunner ngừng tải.</p> <p>Quan sát: Kết quả xử lý 1000 yêu cầu đầu, 100 yêu cầu sau đó bị từ chối ra sao, webserver có khả năng bị khởi động lại hay không,...</p> <p>Từ đó giúp đưa ra kết luật ứng dụng sẽ ứng xử như thế nào khi đạt ngưỡng chịu tải tối đa.</p>
--	---	--

Bảng 2.3. Phân biệt ba loại kiểm thử Performance Test, Load Test, Stress test.

Một phương pháp đo lường hiệu năng phổ biến là thời gian giao tác (transaction time). Thời gian giao tác là tổng thời gian yêu cầu bởi trình khách, mạng và trình chủ để hoàn thành một giao tác. Trong một ứng dụng web, thời gian của giao tác có thể được đo lường bởi khoảng thời gian khi người sử dụng nhấp vào một nút hay một liên kết đến khi trình duyệt đã kết thúc hiển thị trang kết quả (bao gồm thực thi các script phía trình khách, java applet....).

Độ trễ là thời gian yêu cầu để hoàn thành một yêu cầu. Thời gian cần để dữ liệu truyền từ một máy tính đến một máy tính khác được gọi là độ trễ mạng. Thời gian để một trình chủ hoàn thành xử lý một yêu cầu được gọi là độ trễ trình chủ.

Độ trễ cũng có thể biểu diễn sự chậm trễ của một phần cứng đặc biệt. Ví dụ một thiết bị định tuyến có thể xử lý một số giới hạn các gói dữ liệu mỗi giây. Nếu tốc độ đến của các gói dữ liệu vượt quá khả năng xử lý của thiết bị định tuyến, các gói không được xử lý sẽ được xử lý một khi thiết bị định tuyến có thể xử lý chúng.

Mục đích chính của kiểm thử hiệu năng là để xác định như thế nào ứng dụng của bạn thực thi tốt liên quan đến mục tiêu hiệu năng của bạn.

- Xác định tắc nghẽn và các nguyên nhân của chúng

- Tối ưu hóa và điều chỉnh cấu hình nền tảng (cả phần cứng và phần mềm) cho hiệu suất tối đa.
- Xác minh độ tin cậy của ứng dụng của bạn bị căng thẳng.

Một số đặc tính ứng dụng thực hiện thử nghiệm sẽ giúp bạn xác định:

- Thời gian đáp ứng.
- Thông lượng.
- Tối đa người dùng đồng thời hỗ trợ. Đối với một định nghĩa của người sử dụng đồng thời, xem "Kiểm tra cân nhắc", sau đó trong chương này.

Nguồn sử dụng về số lượng CPU, RAM, mạng I / O, và đĩa I / O tiêu thụ tài nguyên ứng dụng của bạn trong thời gian thử nghiệm.

Ứng dụng vi phạm điểm, việc áp dụng điểm có nghĩa là phá vỡ một điều kiện mà các ứng dụng ngừng đáp ứng các yêu cầu.

Các triệu chứng và nguyên nhân của thất bại ứng dụng trong điều kiện căng thẳng.

Mục tiêu

Hầu hết kiểm thử hiệu năng phụ thuộc vào thiết lập được xác định trước, tài liệu và thỏa thuận các mục tiêu hiệu xuất. Biết được mục tiêu ngay từ đầu sẽ giúp làm cho quá trình thử nghiệm hiệu quả hơn. Có thể đánh giá hiệu suất ứng dụng bằng cách so sánh nó với những mục tiêu hiệu suất.

Các mục tiêu hiệu xuất thường bao gồm dưới đây:

- Thời gian đáp ứng hoặc độ trễ
- Thông lượng
- Sử dụng tài nguyên (CPU, mạng I/O, ổ cứng I/O và bộ nhớ)
- Khối lượng tải công việc
- Sức tải công việc

Thời gian đáp ứng hay độ trễ

Thời gian đáp ứng là lượng thời gian thực hiện để đáp ứng yêu cầu. Bạn có thể đo lường thời gian phản hồi của máy chủ hay máy khách như sau:

Đo độ trễ tại máy chủ: đây là thời gian thực hiện bởi máy chủ để hoàn thành thực thi một yêu cầu. Điều này không bao gồm thời gian chờ khách hàng đến máy chủ, trong đó bao gồm thời gian bổ sung cho các yêu cầu và trả lời qua mạng.

Độ trễ đo lường tại máy khách: độ trễ đo lường tại máy khách bao gồm hàng đợi các yêu cầu, cộng với thời gian thực hiện tại máy chủ để hoàn thành việc thực hiện các yêu cầu và độ trễ mạng. Có thể đo lường độ trễ bằng nhiều cách khác nhau. Hai cách tiếp cận thông thường là thời gian thực hiện các byte ban đầu tới khách hàng hay thời gian bởi các byte cuối cùng của phản ứng để đạt được của khách hàng. Nên kiểm tra điều này bằng cách sử dụng mạng băng thông khác nhau giữa máy chủ và máy khách.

Thông lượng

Thông lượng là số yêu cầu mà có thể phục vụ bởi ứng dụng của bạn trên đơn vị thời gian. Nó có thể khác nhau tùy thuộc vào tải (số người dùng) và các loại hoạt động người dùng áp dụng cho máy chủ. Ví dụ, tải các tập tin đòi hỏi thông lượng cao hơn so với trình duyệt văn bản dựa trên các trang web. Thông thường được đo trong điều khoản của yêu cầu mỗi giây. Có những đơn vị khác để đo lường, chẳng hạn như giao dịch / giây hoặc đơn đặt hàng mỗi giây.

Sử dụng tài nguyên

Xác định chi phí sử dụng nguồn lực về tài nguyên máy chủ và mạng. Các nguồn tài nguyên chính là:

- CPU
- Bộ nhớ RAM
- Disk I/O
- Network I/O

Có thể xác định được chi phí tài nguyên trên mỗi cơ sở hoạt động. Hoạt động có thể bao gồm trình duyệt một danh mục sản phẩm, thêm các mục vào một giỏ mua hàng, hoặc đặt hàng. Bạn có thể đo lường chi phí tài nguyên cho một tải người dùng, hoặc bạn có thể trung bình chi phí tài nguyên khi ứng dụng được kiểm thử bằng cách sử dụng một hồ sơ cho khối lượng công việc.

Một hồ sơ khói lượng công việc bao gồm một hỗn hợp tổng hợp của người dùng thực hiện các hoạt động khác nhau. Ví dụ, đối với một tải của 200 người dùng đồng thời (như được định nghĩa dưới đây), hồ sơ có thể chỉ ra rằng 20% người dùng thực hiện đặt hàng, 30% thêm các mục vào một giỏ mua hàng, trong khi 50% duyệt danh mục sản phẩm. Điều này giúp bạn xác định và các khu vực tối ưu hóa có thể tiêu thụ một tỷ lệ lớn bất thường của các tài nguyên máy chủ và time.ofile phản ứng.

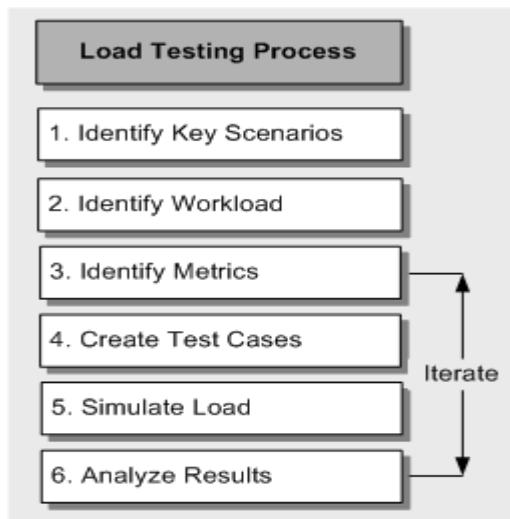
Sức tải công việc

Người dùng đồng thời (Simultaneous users) cùng kết nối đến trang web, trong khi người sử dụng đồng thời (concurrent users) nhấn trang web tại thời điểm chính xác như nhau. Đồng thời truy cập là có thể xảy ra tại các khoảng không thường xuyên. Trang web của bạn có thể có 100-150 người dùng đồng thời nhưng cũng có thể có 1.000 đến 1.500 người sử dụng đồng thời.

Quy trình kiểm thử tải (load testing process)

Sử dụng kiểm thử tải để xác minh hành vi của ứng dụng trong điều kiện trọng tải bình thường và cao điểm. Từng bước tăng tải từ bình thường lên đến cao điểm tải để xem ứng dụng của bạn thực thi với các điều kiện trọng tải khác nhau. Tiếp tục tăng tải lên cho đến khi bạn vượt qua ngưỡng giới hạn cho mục tiêu hiệu suất của bạn. Ví dụ, bạn có thể tiếp tục tăng tải cho đến khi máy chủ CPU đạt mức khoảng 75% là mức quy định của bạn. Quá trình kiểm tra tải trọng cho phép bạn xác định tắc nghẽn ứng dụng và vận hành tối đa năng lực của ứng dụng.

Quy trình kiểm thử tải gồm 6 bước:



Hình 2.4. Quy trình kiểm thử tải

- 1) **Identify key scenarios** (xác định các kịch bản): Xác định các kịch bản ứng dụng rất quan trọng để thực hiện.
- 2) **Identify workload** (xác định sức tải công việc): Phân phối tổng tải ứng dụng trong các kịch bản chính xác định trong bước 1.
- 3) **Identify metrics** (xác định các số liệu): Xác định các số liệu mà bạn muốn thu thập về các ứng dụng khi chạy thử nghiệm.
- 4) **Create test cases** (tạo ra các test case): Tạo các trường hợp kiểm tra nơi bạn xác định các bước để thực hiện một thử nghiệm duy nhất cùng với các kết quả mong đợi.
- 5) **Simulate load** (mô phỏng tải): Sử dụng các công cụ để kiểm tra mô phỏng trọng tải trong các trường hợp thử nghiệm và để nắm bắt được các số liệu.
- 6) **Analyze the results** (phân tích kết quả): Phân tích được các số liệu nắm bắt được trong quá trình kiểm tra.

Các kết quả khác nhau của quá trình thử tải như sau:

- Cập nhật kế hoạch thử nghiệm.
- Hành vi của các ứng dụng của bạn ở mức tải khác nhau.
- Vận hành tối đa công suất.
- Khả năng tắc nghẽn.
- Đề xuất sửa chữa các tắc nghẽn.

Quy trình kiểm thử quá tải (Stress-Testing Process)

Kiểm thử Stress ứng dụng chịu các tải trọng rất cao, vượt quá khả năng của ứng dụng. Ví dụ, bạn có thể triển khai ứng dụng của bạn trên một máy chủ đang chạy một ứng dụng chuyên sâu đã được xử lý. Bằng cách này, ứng dụng của bạn là ngay lập tức bị thiếu tài nguyên bộ xử lý và phải cạnh tranh với các ứng dụng khác cho các chu kỳ CPU.

Mục tiêu của kiểm thử Stress là lỗi ứng dụng phát hiện ra chỉ trong điều kiện tải cao. Những lỗi có thể bao gồm:

- Đồng bộ hóa các vấn đề
- Điều kiện tốc độ
- Bộ nhớ rò rỉ
- Mất dữ liệu khi tắt nghẽn mạng

Workload Modeling

Mỗi hiện trạng khối lượng công việc miêu tả sự thay đổi của thuộc tính cho phép:

- Kịch bản chính
- Số người sử dụng đồng thời
- Tốc độ của yêu cầu
- Mô hình yêu cầu

Mẫu khối lượng công việc được định nghĩa xác định mỗi kịch bản như thế nào được thực thi. Nó cũng định nghĩa gần đúng dữ liệu truy cập mẫu và xác định kiểu số người sử dụng và đặc điểm.

Một số câu hỏi quan trọng giúp bạn xác định khối lượng công việc cho ứng dụng của bạn bao gồm:

1) Kịch bản chính là gì?

Khi xác định kịch bản để tạo ra ứng dụng của bạn từ hiệu năng triển vọng, bạn nên lưu ý các kịch bản một phần yêu cầu phân tích thực thi trong giai đoạn đơn giản của vòng đời phát triển ứng dụng của bạn.

2) Số người sử dụng mong đợi lớn nhất đăng nhập trong ứng dụng

Số người sử dụng đồng thời là số người sử dụng mà có hành động kết nối tới website. Người miêu tả lưu lượng hệ thống cho ứng dụng của bạn. Cho mẫu ứng dụng

bán hàng điện tử thừa nhận 1000 người sử dụng đồng thời. Nếu bạn phát triển một ứng dụng mới, bạn có thể chứa số người sử dụng đang làm việc với đội thăm dò và sử dụng kết quả phân tích của đội. Cho ứng dụng tồn tại, bạn có thể xác định số người dùng tương tự khi phân tích bạn đăng nhập IIS Internet Information Services.

3) Người dùng có thể thực thi được những hành động đã đặt trước

Điều này phụ thuộc trên hành động người dùng có thể thực thi khi anh ấy hoặc cô ấy đăng nhập vào ứng dụng. Cho ứng dụng mẫu, sẽ cho phép hành động:

- Kết nối tới trang chủ
- Đăng nhập vào ứng dụng
- Duyệt danh mục sản phẩm
- Tìm kiếm sản phẩm cụ thể
- Thêm sản phẩm trong giỏ hàng
- Xác nhận và đặt hàng
- Đăng xuất ra khỏi ứng dụng.

4) Các hồ sơ người dùng khác nhau đối với ứng dụng

Bạn có thể nhóm thành các nhóm khác nhau về người sử dụng và các hành động họ thực hiện. Ví dụ: có những nhóm người dùng chỉ cần kết nối và duyệt danh mục sản phẩm, và có những người dùng khác đăng nhập, tìm kiếm các sản phẩm cụ thể, và sau đó đăng xuất. Tổng số người dùng đăng nhập vào là một tập hợp con của những người dùng thực sự đặt hàng các sản phẩm.

5) Các hoạt động thực hiện bởi người dùng trung bình cho mỗi lần truy cập

Các hoạt động thực hiện bởi một người dùng trung bình cho mỗi cấu hình dựa trên dữ liệu thăm dò cho một ứng dụng mới và phân tích bản ghi IIS cho một ứng dụng hiện có. Ví dụ, nếu bạn đang phát triển một trang web doanh nghiệp với người tiêu dùng và thương mại điện tử, nghiên cứu dựa trên dữ liệu thăm dò của bạn cho biết rằng trên mức trung bình, một người dùng mua nhiều sản phẩm sẽ mua ít nhất là ba sản phẩm từ trang web của bạn trong một phiên duy nhất.

Các hành động thực hiện đối với các hồ sơ để đặt cho các ứng dụng mẫu được xác định trong bảng sau:

Hoạt động	Số thời gian thực hiện
-----------	------------------------

	trong một phiên duy nhất
Kết nối tới trang chủ	1
Đăng nhập vào ứng dụng	1
Duyệt danh mục sản phẩm	4
Tìm kiếm sản phẩm cụ thể	2
Thêm sản phẩm vào giỏ mua hàng	3
Xác nhận và đặt lệnh	1
Đăng xuất khỏi ứng dụng	1

Bảng 2.4. Hoạt động của người dùng đối với ứng dụng mẫu

Một người sử dụng điển hình cho cấu hình tự đặt có thể không tìm kiếm trang web của bạn, nhưng bạn có thể giả định hành động như vậy bởi trung bình trong các hoạt động thực hiện bởi người sử dụng khác nhau cho một cấu hình cụ thể. Bạn có thể tạo ra một bảng các hành động thực hiện bởi một người dùng trung bình cho tất cả các cấu hình cho các ứng dụng của bạn.

Thời gian thao tác giữa các yêu cầu

Thời gian thao tác là thời gian sử dụng của người dùng giữa hai yêu cầu liên tiếp. Trong thời gian này, người sử dụng xem các thông tin hiển thị trên một trang web hoặc đi vào chi tiết như số thẻ tín dụng, tùy theo tính chất của hoạt động được thực hiện.

Thời gian thao tác có thể khác nhau tùy thuộc vào sự phức tạp của dữ liệu trên một trang. Ví dụ, thời gian thao tác cho các trang đăng nhập sẽ thấp hơn thời gian thao tác cho một trang đặt hàng nơi người sử dụng phải nhập chi tiết thẻ tín dụng. Bạn có thể đưa ra thời gian thao tác trung bình cho tất cả các yêu cầu.

Dự kiến kết hợp hồ sơ người sử dụng

Các mô hình sử dụng cho mỗi kịch bản mang một ý tưởng trong một khung thời gian nhất định của sự pha trộn tỷ lệ phần trăm của các hành động kinh doanh được thực hiện bởi người sử dụng. Một ví dụ là thể hiện trong bảng sau:

Hồ sơ người sử dụng	Tỷ lệ %	Số người sử dụng đồng thời
Duyệt	50%	500
Tìm kiếm	30%	300
Đặt hàng	20%	200
Tổng	100%	1000

Bảng 2.5. Kết hợp hồ sơ người dùng với kịch bản mẫu

Thời gian cần cho việc kiểm tra một thao tác là gì?

Thời hạn kiểm tra phụ thuộc vào khối lượng công việc cho các mô hình ứng dụng của bạn và có thể từ 20 phút đến khi một tuần. Một trang web tìm ra được sự đều đặn của hồ sơ người dùng trong một ngày. Ví dụ, lưu trữ một trang web thương mại điện tử trực tuyến, các hoạt động thực hiện bởi một người dùng trung bình ghé thăm trang web không có khả năng thay đổi trong thời gian 8 đến 10 giờ kinh doanh trong ngày. Các thử nghiệm được thực hiện đơn giản bằng cách thay đổi số lượng người dùng cho mỗi chu kỳ kiểm tra. Đối với kịch bản này, một thời gian thử nghiệm trong khoảng 20 phút đến 1 giờ có thể là đủ.

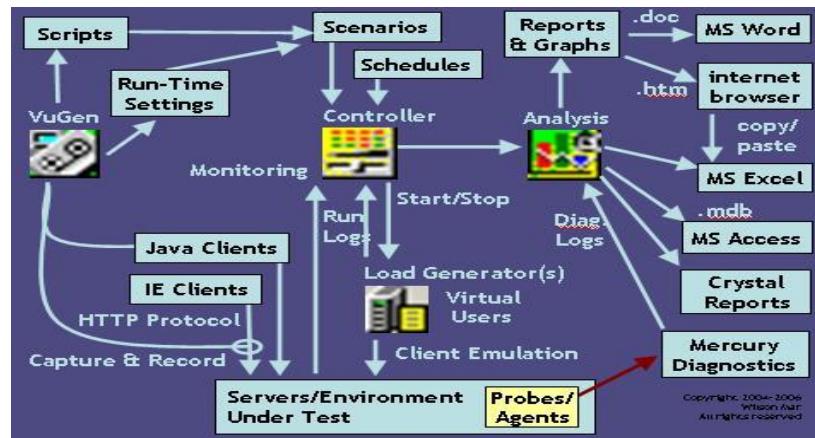
10.4.5. Một số công cụ kiểm thử

10.4.5.1. Giới thiệu công cụ kiểm thử loadrunner

Kiểm thử hiệu năng là một chiến lược phổ biến cho chất lượng cũng như giảm bớt nguy cơ rủi ro cho một doanh nghiệp.

LoadRunner là một công nghệ hiệu quả hàng đầu và là một sản phẩm kiểm thử tải của Hewlett- để kiểm tra hành vi và hiệu suất của hệ thống trong khi tạo tải thực tế. LoadRunner có thể cho phép hàng trăm hoặc hàng ngàn người dùng sử dụng chạy các ứng dụng của mình cùng một lúc thông qua các điều kiện khắt khe trong thực tế của người sử dụng tải khi thu thập thông tin từ các thành phần chính trong một hệ thống như: webserver, database server,... Các kết quả sẽ được phân tích một cách chi tiết. LoadRunner mô phỏng số lượng lớn người sử dụng cùng một lúc truy cập tới server.

10.4.5.2. Kiến trúc tổng quan của LoadRunner



Hình 2.5. Kiến trúc tổng quan của LoadRunner

LoadRunner có ba thành phần chính nằm ở trung tâm đó là: Controller, Analysis và Virtual Users

1) Virtual Users:

LoadRunner hoạt động bằng cách tạo ra người dùng ảo, nơi mà người dùng thực hiện phần mềm khách như: Internet Explorer để gửi những yêu cầu bằng cách sử dụng giao thức HTTP đến IIS (Internet Information Services) hoặc Apache web servers.

2) Controller:

Một khi một kịch bản được chuẩn bị trong VuGen, nó chạy qua Controller. Mỗi lần chạy được cấu hình với một chuỗi sự kiện, trong đó mô tả chi tiết cái kịch bản sẽ chạy, khi họ chạy thì bao nhiêu người dùng ảo sẽ chạy. LoadRunner sử dụng màn hình trong thời gian thử tải để giám sát hiệu suất của các thành phần tải riêng biệt. Một chuỗi sự kiện được thiết lập và chạy xong, kết quả của chuỗi sự kiện đó được hiển thị qua công cụ Analysis.

3) Analysis:

Công cụ này sẽ hiển thị các đồ thị cần thiết cho người kiểm thử xem. Ngoài ra các đồ thị còn có thể được hợp nhất để có được một hình ảnh tốt về hiệu suất. Các kiểm thử sau đó có thể được hiệu chỉnh cần thiết để vẽ đồ thị và chuẩn bị một bản báo

cáo. Bản báo cáo bao gồm tất cả các đồ thị cần thiết có thể được lưu trong một số định dạng, bao gồm cả HTML và định dạng Microsoft Word.

10.4.5.3. Các thuật ngữ liên quan

Virtual Users (Vuser): LR thay thế người dùng thật bằng người dùng ảo. Khi bạn chạy một kịch bản, người dùng ảo cạnh tranh các hoạt động với người dùng thật. Trong khi một phiên làm việc chỉ cung cấp cho một người dùng thật thì có rất nhiều người dùng ảo có thể hoạt động trong một phiên làm việc như thế. Trong thực tế, một kịch bản có thể chứa hàng chục, hàng trăm hoặc hàng nghìn người dùng ảo.

Virtual User Scripts (Vuser Scripts): Các hoạt động mà một người dùng ảo thực hiện trong khi kịch bản được mô tả trong một Vuser script. Khi bạn chạy kịch bản, từng người dùng ảo sẽ thực thi một Vuser script. Chức năng của một Vuser script bao gồm: Đo lường và ghi lại hiệu năng hoạt động của các thành phần trong ứng dụng của bạn.

Transactions: Một transaction (phiên giao dịch) là một hay một tập các hoạt động mà bạn muốn xác định khi client và server trao đổi với nhau.

Ví dụ: bạn có thể chỉ ra một phiên giao dịch để đo thời gian mà server xử lý một yêu cầu vấn tin số dư tài khoản và thông tin hiển thị trên ATM của khách hàng.

Rendezvous points: Tester có thể thêm Rendezvous point vào Vuser Script để điều khiển nhiều người dùng ảo đến và thực hiện đồng thời một công việc nào đó.

Scenario: Chúng ta thiết lập các thông số đầu vào để thực hiện kiểm thử trong kịch bản. Một kịch bản chỉ rõ các sự kiện xảy ra (Vuser Scripts) trong khi một ca test được thực hiện. Trong kịch bản, rất nhiều người dùng ảo được tạo ra cùng thực hiện một nhiệm vụ nào đó.

VuGen (Virtual User Generator): Là một thành phần của LoadRunner có thể tự sinh ra Vuser Script hoặc sử dụng Vuser Script có sẵn để phục vụ cho việc kiểm thử.

10.4.5.4. Các phiên bản của LoadRunner

Trong quá trình hình thành và phát triển, LoadRunner có các phiên bản: V6.0 sử dụng riêng biệt cho mỗi người dùng, cái mà yêu cầu gần 10 thời gian I/O và chu kỳ CPU hơn 6,5.

Phiên bản 7.8 ra tháng 9 năm 2003 được hỗ trợ cho Windows XP.

Phiên bản 8.0 ra tháng 8 năm 2004. Nó thêm "Thuộc tính bổ sung" để Cài đặt Thời gian chạy. Nó cũng cho biết thêm (cho thêm phí) chẩn đoán và điều chỉnh các khả năng, cho phép giao dịch Breakdown đến sự cố lần giao dịch trên các máy chủ khác nhau phục vụ các tầng giao dịch nhiều máy chủ web (Oracle 11i & Peoplesoft 8 ứng dụng máy chủ, cơ sở dữ liệu)

Phiên bản 8.1 Feature Pack 3 cài đặt LR81FP3.exe, tại 116.601.240 byte, được ký ngày 18 tháng 6 2006 là phiên bản 8.1.3.0 file (Build 2085). Cài đặt Microsoft WSE (Web Services Enhancements) 2.0 SP3 để triển khai các chính sách an ninh cho hệ thống chạy.

Phiên bản 8.1 Feature Pack 4 cài đặt LR81FP4.exe, tại 194.644.720 byte, đã được ký kết vào ngày 15 tháng 12 năm 2006 như là phiên bản 8.1.4.0 file (Build: 2249) là ghi âm Phiên bản: 1289. Điều này đòi hỏi một sự nâng cấp lên 2,0 MS.NET khách hàng.

Phiên bản 9.10 trình cài đặt, có tháng hai năm 2008, là 2,31 GB sau khi mở rộng. Tuy nhiên, các thư mục sau khi tạo ra một ngôn ngữ tiếng Anh cài đặt là 931MB.

Phiên bản 9.51 là bản sửa lỗi của 9.50 (tập KM750376 tập LR_03009.zip, ngày 06 tháng bảy năm 2009, là một 201 MB zip. AJAX cho phép công nhận tốt hơn và đặc điểm kỹ thuật của các thuộc tính nguyên tố DOM này chạy GACSetup.exe và Magentconfig.exe, đòi hỏi người dùng Vista cho phép. Tính năng chính của LR trong việc kiểm thử phần mềm.

10.4.5.5. Ưu điểm của load runner so với phần mềm khác

Hiện nay có rất nhiều phần mềm hỗ trợ việc kiểm thử hiệu năng hoạt động của một trang web như : Dotcom- Monitor's Web Load Stress Test, Web2Test của Quality First Software, Rational IBM... Song theo đánh giá của nhiều người sử dụng thì LoadRunner là một phần mềm có nhiều tính năng và đem lại hiệu quả cao nhất.

LoadRunner giả lập môi trường hoạt động của ứng dụng client-server giống như thực tế chỉ trên một máy tính (hoặc một vài máy tính) với một người là Tester điều khiển. Vì vậy sử dụng LoadRunner sẽ tiết kiệm được chi phí về tài nguyên và nhân lực.

LoadRunner giảm bớt nhu cầu về nhân lực bằng cách thay thế người dùng thật bằng người dùng ảo, người dùng ảo cạnh tranh hoạt động với người dùng thật.

LoadRunner tự động sinh ra hàng nghìn người dùng ảo cùng hoạt động trên một máy tính và cho phép bạn dễ dàng điều khiển hoạt động của người dùng ảo nên giảm chi phí về tài nguyên phần cứng.

LoadRunner cho phép bạn giám sát và tự động ghi lại mọi hoạt động của ứng dụng khi đang thực hiện kiểm thử.

LoadRunner tự động ghi lại hiệu năng hoạt động của ứng dụng trong khi đang kiểm tra. Bạn có thể lựa chọn và quan sát các biểu đồ khác nhau như: Tài nguyên phần cứng sử dụng, thời gian đáp trả, thời gian phục hồi...

LoadRunner giúp bạn phân tích các thông số thu được để từ đó có giải pháp cải thiện hiệu suất hoạt động của hệ thống.

Giúp giảm thời gian viết test script đến 80% do nó cung cấp chức năng tự động phát sinh script mô tả lại các tình huống muốn kiểm tra.

LoadRunner giúp bạn thực hiện test lại (tái tạo) một cách dễ dàng.

10.5. Kiểm thử bảo mật

10.5.1. Khái niệm

Security Testing là một tiến trình nhằm xác định rằng một hệ thống thông tin bảo mật dữ liệu cần phải đảm bảo tính đồng nhất và duy trì những chức năng như đã chỉ định.

Là kiểm tra xem chức năng, dữ liệu *private* có bị *public* không.

Kiểm thử bảo mật và kiểm soát truy cập tập trung vào hai lĩnh vực bảo mật:

- Bảo mật ở mức ứng dụng, bao gồm truy cập dữ liệu và các chức năng nghiệp vụ
- Bảo mật ở mức hệ thống, bao gồm truy cập vào hệ thống hoặc truy cập từ xa

Bảo mật mức ứng dụng đảm bảo rằng, dựa trên bảo mật đã yêu cầu, người dùng bị hạn chế sử dụng một số chức năng hoặc tình huống sử dụng, hoặc bị hạn chế trong giới hạn dữ liệu phù hợp với họ. Ví dụ, mọi người có thể được phép nhập dữ liệu để tạo account nhưng chỉ có người quản lý có thể xóa chúng. Nếu là bảo mật ở mức dữ liệu, việc test đảm bảo rằng “người dùng nhóm 1” có thể nhìn thấy các thông tin khách hàng, bao gồm dữ liệu tài chính, tuy nhiên “người dùng nhóm 2” chỉ nhìn thấy các thông tin chung chung cho cùng một khách hàng.

Bảo mật mức hệ thống đảm bảo rằng chỉ những người dùng được cho quyền truy cập vào hệ thống mới có khả năng truy cập vào ứng dụng và chỉ bằng các công thức hợp

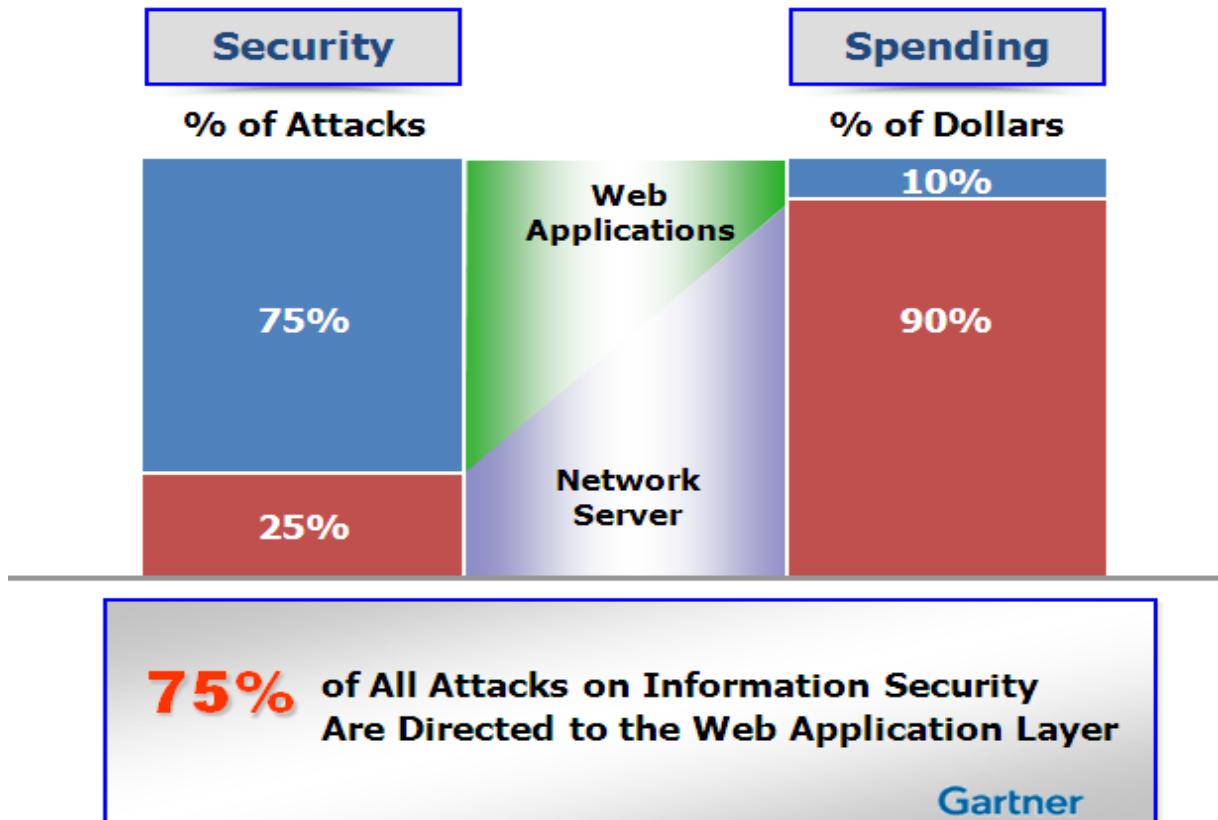
10.5.2. Mục đích của kiểm thử bảo mật

Bảo mật mức ứng dụng: Đảm bảo rằng một người dùng chỉ có thể truy cập vào những chức năng hoặc dữ liệu mà nhóm người dùng đó được phép.

Bảo mật mức hệ thống: Đảm bảo rằng chỉ những người được phép truy cập hệ thống và ứng dụng được phép truy cập chúng.

10.5.3. Các vấn đề liên quan đến kiểm thử bảo mật

10.5.3.1. Tấn công ứng dụng web

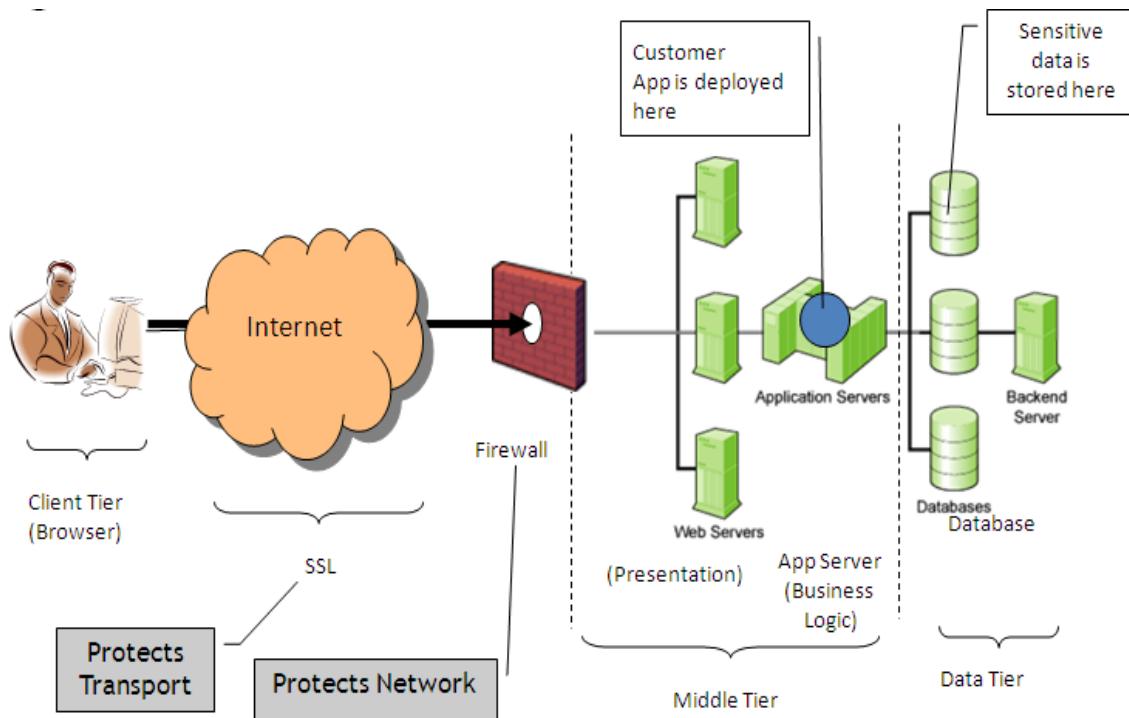


Hình 2.6. lý do quan tâm đến các tấn công vào ứng dụng Web

Lý do bảo mật ứng dụng có độ ưu tiên cao vì: Hacker tập trung tấn công vào các ứng dụng web, với 75% tấn công vào lớp ứng dụng (Gartner) và XSS và SQL Injection theo báo cáo thì có độ nguy hiểm cao (Mitre). Hầu hết các website đều có độ nguy hiểm; đồng thời Hacker luôn tập trung vào các thông tin như dữ liệu khách hàng, credit card, id... để thu được kết quả cao.

Nguyên nhân bảo mật ứng dụng vẫn còn tồn tại một số vấn đề: nguyên nhân chủ yếu là do người lập trình không được đào tạo để viết hoặc test một code đảm bảo an toàn, bảo mật network (firewall, IDS...) không được bảo vệ trong lớp ứng dụng Web (Web Application Layer). Tình trạng hiện tại, khoảng cách trao đổi thông tin giữa bảo mật và phát triển vẫn còn tồn tại một số vấn đề chưa được giải quyết và kiểm thử tổng quan chưa được hoàn chỉnh. Vậy mục đích của kiểm thử bảo mật nhằm phát triển nhiều ứng dụng Web, Website hoàn chỉnh hơn và an toàn hơn.

10.5.3.2. Các kỹ thuật tấn công vào ứng dụng web



Hình 2.7. Kiến trúc ứng dụng web ở mức cao

Tấn công vào các lỗ hổng- điểm yếu (Injection vulnerabilities)

Loại tấn công này thường được sử dụng khi dữ liệu đầu vào chưa được xác nhận (validated). Những cuộc tấn công này sẽ cung cấp một số dữ liệu đầu vào và đính kèm thêm dữ liệu độc hại (những dữ liệu đính kèm này có thể dưới dạng các dòng lệnh để lấy được thêm các thông tin bảo mật). Tấn công vào các lỗ hổng là một cách tấn công phổ biến nhất. Và những lỗi trong việc kiểm tra đầu vào (Input validation) là một điểm yếu mà hầu hết các ứng dụng gặp phải.

Trong kỹ thuật tấn công vào các lỗ hổng lại chia thành nhiều phương pháp khác nhau mà điển hình bao gồm:

CRLF Injection

SQ L injection

SQL injection là một kỹ thuật cho phép những kẻ tấn công lợi dụng lỗ hổng trong việc kiểm tra dữ liệu nhập trong các ứng dụng web và các thông báo lỗi của hệ quản trị cơ sở dữ liệu để “tiêm vào” (inject) và thi hành các câu lệnh SQL bất hợp

pháp (không được người phát triển ứng dụng lường trước). Hậu quả của nó rất tai hại vì nó cho phép những kẻ tấn công có thể thực hiện các thao tác xóa, hiệu chỉnh, ... do có toàn quyền trên cơ sở dữ liệu của ứng dụng, thậm chí là server mà ứng dụng đó đang chạy. Lỗi này thường xảy ra trên các ứng dụng web có dữ liệu được quản lý bằng các hệ quản trị cơ sở dữ liệu như SQL Server, MySQL, Oracle, DB2, Sysbase.

Có bốn dạng thông thường bao gồm: vượt qua kiểm tra lúc đăng nhập (authorization bypass), sử dụng câu lệnh SELECT, sử dụng câu lệnh INSERT, sử dụng các stored-procedures.

Dạng tấn công vượt qua kiểm tra đăng nhập, với dạng tấn công này, tin tặc có thể dễ dàng vượt qua các trang đăng nhập nhờ vào lỗi khi dùng các câu lệnh SQL thao tác trên cơ sở dữ liệu của ứng dụng web.

Dạng tấn công sử dụng câu lệnh SELECT, dạng tấn công này phức tạp hơn. Để thực hiện được kiểu tấn công này, kẻ tấn công phải có khả năng hiểu và lợi dụng các sơ hở trong các thông báo lỗi từ hệ thống để dò tìm các điểm yếu khởi đầu cho việc tấn công. Xét một ví dụ rất thường gặp trong các website về tin tức. Thông thường, sẽ có một trang nhận ID của tin cần hiển thị rồi sau đó truy vấn nội dung của tin có ID này. Ví dụ: <http://www.myhost.com/shownews.asp?ID=123>. Mã nguồn cho chức năng này thường được viết khá đơn giản theo dạng:

```
<%
Dim vNewsID, objRS, strSQL
vNewsID = Request("ID")
strSQL = "SELECT * FROM T_NEWS WHERE NEWS_ID =" & vNewsID

Set objRS = Server.CreateObject("ADODB.Recordset")
objRS.Open strSQL, "DSN=..."
Set objRS = Nothing
%>
```

Trong các tình huống thông thường, đoạn mã này hiển thị nội dung của tin có ID trùng với ID đã chỉ định và hầu như không thấy có lỗi. Tuy nhiên, giống như ví dụ đăng nhập ở trước, đoạn mã này để lộ sơ hở cho một lỗi SQL injection khác. Kẻ tấn công có thể thay thế một ID hợp lệ bằng cách gán ID cho một giá trị khác, và từ đó,

khởi đầu cho một cuộc tấn công bất hợp pháp, ví dụ như: 0 OR 1=1 (nghĩa là, <http://www.myhost.com/shownews.asp?ID=0 or 1=1>). Câu truy vấn SQL lúc này sẽ trả về tất cả các article từ bảng dữ liệu vì nó sẽ thực hiện câu lệnh:

```
SELECT * FROM T_NEWS WHERE NEWS_ID=0 or 1=1
```

Dạng tấn công sử dụng câu lệnh INSERT, thông thường các ứng dụng web cho phép người dùng đăng ký một tài khoản để tham gia. Chức năng không thể thiếu là sau khi đăng ký thành công, người dùng có thể xem và hiệu chỉnh thông tin của mình. SQL injection có thể được dùng khi hệ thống không kiểm tra tính hợp lệ của thông tin nhập vào. Ví dụ, một câu lệnh INSERT có thể có cú pháp dạng: INSERT INTO TableName VALUES('Value One', 'Value Two', 'Value Three'). Nếu đoạn mã xây dựng câu lệnh SQL có dạng:

```
<%  
strSQL = "INSERT INTO TableName VALUES(' " & strValueOne & " ', ' " _  
& strValueTwo & " ', ' " & strValueThree & " ')"  
Set objRS = Server.CreateObject("ADODB.Recordset")  
objRS.Open strSQL, "DSN=..."  
Set objRS = Nothing  
%>
```

Thì chắc chắn sẽ bị lỗi SQL injection, bởi vì nếu ta nhập vào trường thứ nhất ví dụ như: ' + (SELECT TOP 1 FieldName FROM TableName) + '. Lúc này câu truy vấn sẽ là: INSERT INTO TableName VALUES(' ' + (SELECT TOP 1 FieldName FROM TableName) + ' ', 'abc', 'def'). Khi đó, lúc thực hiện lệnh xem thông tin, xem như bạn đã yêu cầu thực hiện thêm một lệnh nữa đó là: SELECT TOP 1 FieldName FROM TableName.

Dạng tấn công sử dụng stored-procedures, việc tấn công bằng stored-procedures sẽ gây tác hại rất lớn nếu ứng dụng được thực thi với quyền quản trị hệ thống 'sa'. Ví dụ, nếu ta thay đoạn mã tiêm vào dạng: ' ; EXEC xp_cmdshell 'cmd.exe dir C: '. Lúc này hệ thống sẽ thực hiện lệnh liệt kê thư mục trên ổ đĩa C:\ cài đặt server. Việc phá hoại kiểu nào tuỳ thuộc vào câu lệnh đăng sau cmd.exe.

Javascript Injection

Javascript Injection là một kỹ thuật nhỏ tiện lợi cho phép thay đổi một số nội dung của trang web mà không thực sự rời khỏi trang web đó, bằng cách đưa vào một đoạn mã javascript lên thanh address bar.

Ví dụ: ta tiêm đoạn mã javascript:alert(document.cookie); lên thanh address bar của 1 trang web, chúng ta có thể nhìn thấy cookie của trang đó.

XSS Injection

Cross-Site Scripting (XSS) là một trong những kĩ thuật tấn công phổ biến nhất hiện nay, đồng thời nó cũng là một trong những vấn đề bảo mật quan trọng đối với các nhà phát triển web và cả những người sử dụng web. Bất kì một website nào cho phép người sử dụng đăng thông tin mà không có sự kiểm tra chặt chẽ các đoạn mã nguy hiểm thì đều có thể tiềm ẩn các lỗ XSS.

Cross-Site Scripting hay còn được gọi tắt là XSS (thay vì gọi tắt là CSS để tránh nhầm lẫn với CSS-Cascading Style Sheet của HTML) là một kĩ thuật tấn công bằng cách chèn vào các website động (ASP, PHP, CGI, JSP ...) những thẻ HTML hay những đoạn mã script nguy hiểm có thể gây nguy hại cho những người sử dụng khác. Trong đó, những đoạn mã nguy hiểm được chèn vào hầu hết được viết bằng các Client-Site Script như JavaScript, Jscript, DHTML và cũng có thể là cả các thẻ HTML.

Tấn công tràn bộ đệm (Buffer Overflow)

Tràn bộ đệm là những lỗ hổng bảo mật cổ điển mà đã có từ khi bắt đầu lập trình, và vẫn còn xảy ra ở hiện tại. Tràn bộ đệm là rất phổ biến và luôn làm đau đầu các chuyên gia bảo mật phần mềm. Một lỗ tràn bộ đệm là nơi mà bộ nhớ đã được ghi đè trên stack. Thông thường điều này có thể xảy ra bằng cách gửi một số lượng rất lớn dữ liệu đến ứng dụng và sau đó tiêm chính tấn công nguy hiểm vào cuối dòng dữ liệu.

Tấn công từ chối dịch vụ (DoS vulnerabilities)

Điều này có thể gây ra do sử dụng CPU quá mức, rò rỉ bộ nhớ, tìm kiếm ldap chậm hoặc dài... Một cuộc tấn công từ chối dịch vụ có thể mang xuống một hệ thống toàn bộ thông tin của hệ thống bảo mật. Có không nhiều cách để tester có thể kiểm tra cho loại tấn công này.

Tấn công vào thư mục qua trình duyệt. (Directory Traversal)

Trong loại tấn công này, Hacker sẽ nhập vào những dữ liệu không hợp lệ . Diễn hình của tấn công này là việc các ký tự ./ và \ không được kiểm tra một cách chặt chẽ. Hacker sẽ tận dụng lỗ hổng đó để xâm nhập và hệ thống các thư mục của ứng dụng qua dòng địa chỉ dù đã được mã hóa của trình duyệt.

Thông báo lỗi (Error message)

Ai đã từng làm việc với các ứng dụng java hẳn là không lạ với các lỗi này. Các lỗi Java Exception đôi khi có thể hiện ra cả các dòng code hay các thông tin quan trọng trong database của ứng dụng.

Từ việc phân tích các kỹ thuật cơ bản tấn công vào ứng dụng web ở trên, tester sẽ thiết kế được các tình huống kiểm thử bảo mật và tiến hành kiểm thử, đồng thời lựa chọn và sử dụng các công cụ quét lỗ hổng, kiểm thử bảo mật hiệu quả.

10.5.4. Các công cụ kiểm thử bảo mật

Hiện nay có rất nhiều các công cụ tự động hỗ trợ kiểm thử viên kiểm thử bảo mật các ứng dụng. Trong đó, Acunetix Web Vulnerability Scanner và WebSecurity là hai công cụ quét lỗ hổng bảo mật được đánh giá tốt, ngoài ra còn có Web Link Validator.

10.5.4.1. Acunetix Web Vulnerability Scanner

Acunetix Web Vulnerability Scanner là công cụ dụng để quét lỗi website, giúp tìm kiếm những lỗi có thể mắc phải trong quá trình code cho website như SSL, scan lỗi SQL Injection, tìm lỗi XSS,...

Chương trình có thể hỗ trợ:

Tìm kiếm lỗi của một website: SQL Injection, XSS...

Tìm kiếm cấu trúc của một website.

Tìm kiếm lỗi của server chứa website và các thông tin liên quan đến server của website.

Báo cáo cũng như gợi ý chỉnh sửa các lỗi của website. Lưu các kết quả báo cáo cho việc fix lỗi sau này.

Lập lịch tiến hành scan lỗi cho website.

10.5.4.2. WebSecurity

Websecurity là một công cụ kiểm thử tính bảo mật của một website trên môi trường Internet.

Websecurity sẽ được cài đặt và chạy trên client. Nhập đường dẫn của trang web cần kiểm thử bảo mật vào Websecurity và tiến hành test. Websecurity sẽ tự động tìm ra các lỗi liên quan đến vấn đề bảo mật cho website cần kiểm thử.

Phiên bản mới nhất của Websecurity là Websecurity 0.8Beta1(cho Windows), ngoài ra Websecurity còn chạy được trên các hệ điều hành khác như MAC, Linux...

Tính năng của Websecurity:

- Có thể chạy trên tất cả các hệ điều hành (Windows, Mac OS, Linux);
- Có giao diện đơn giản, dễ sử dụng.
- Hỗ trợ tích hợp theo chuẩn quốc tế.
- Dễ dàng mở rộng với việc hỗ trợ của các tiện ích và bổ sung.
- Xuất và tùy chỉnh báo cáo với các mức độ chi tiết khác nhau.
- Hỗ trợ Moduler và thiết kế tái sử dụng.
- Công cụ kiểm thử bằng tay và phương tiện trợ giúp hiệu quả.
- Kỹ thuật phân tích lỗi và quét lỗi hỏng hiệu quả.
- Có thể tạo ra các kịch bản kiểm thử, hỗ trợ JavaScript và Python.
- Có thể mở rộng thông qua nhiều ngôn ngữ bao gồm JavaScript, Python, C, C++ và Java.

10.5.4.3. Web Link Validator

Web Link validator là công cụ giúp các quản trị web kiểm tra độ chính xác và tính sẵn sàng của các liên kết, nó tìm ra các liên kết bị phá vỡ và các liên kết chứa lỗi cú pháp. Chương trình có thể dễ dàng xử lý các trang web có chứa hàng nghìn liên kết (bao gồm cả các liên kết được mã hoá với javascript và flash, hoặc được nhúng trong các biểu đồ, hình ảnh, các applet...); nó cũng cung cấp một số tính năng giúp quản trị web cải thiện đáng kể tốc độ xử lý trang web và sử dụng tài nguyên hệ thống một cách hiệu quả hơn.

Sau khi thực hiện phân tích chuyên sâu và kiểm tra các link trong trang web, Web Link Validator cho phép xuất ra các báo cáo quá trình test dưới dạng HTML hoặc tệp excel, hoặc có thể gửi báo cáo qua thư điện tử.

Các tính năng của Web Link Validator:

- Kiểm tra trên 10.000.000 liên kết cục bộ, HTTP, HTTPS và FTP;
- Hỗ trợ các siêu liên kết Javascript và Adobe Flash (tệp .SWF);
- Tích hợp kiểm tra lỗi chính tả trong tiếng anh và hơn 20 ngôn ngữ khác;
- Dò tìm các liên kết chuyển hướng không hợp lệ;
- Kiểm tra HTTPs, đảm bảo và bảo vệ mật khẩu của trang web;
- Có thể xuất bản sang HTML, CSV, RTF và định dạng Microsoft Excel;
- Hỗ trợ xác thực dựa trên web;
- Gửi báo cáo tự động qua email;
- Hỗ trợ dòng lệnh;

10.6. Kiểm thử khả năng tiện dụng

10.6.1. Khái niệm kiểm thử khả năng tiện dụng

Khả năng tiện dụng là độ đo hỗ trợ người thiết kế sản phẩm hay dịch vụ (bao gồm ứng dụng Web, ứng dụng phần mềm, và ứng dụng trên thiết bị di động) xác định sự hài lòng của những người dùng khi họ tương tác với sản phẩm hay dịch vụ qua giao diện, bao gồm giao diện người dùng. Một thiết kế giao diện người dùng hiệu quả là một giao diện cung cấp khả năng tiện dụng cao nhất cho người dùng.

10.6.2. Mục đích kiểm thử khả năng tiện dụng

Kiểm thử khả năng tiện dụng dùng kỹ thuật kiểm thử hộp đen. Mục đích là để quan sát những người sử dụng sản phẩm để phát hiện các lỗi và các vùng cần cải thiện. Kiểm thử khả năng tiện dụng sẽ đảm bảo cho người sử dụng có thể sử dụng sản phẩm một cách đơn giản, dễ hiểu nhất.

10.6.3. Các vấn đề trong kiểm thử khả năng tiện dụng

10.6.3.1. Kiểm thử giao diện người dùng

Một đặc tính quan trọng trong giao diện người dùng là phần mềm của bạn cho phép tồn tại chuẩn và hướng dẫn. Nếu phần mềm của bạn chạy trên nền tảng xác định như Mac hoặc Windows là những chuẩn được thiết lập. Mọi thứ đều được định nghĩa khi sử dụng các ô check thay cho các nút chọn cho tới các thuộc tính sử dụng như thông tin, cảnh báo, thông báo, phê bình.

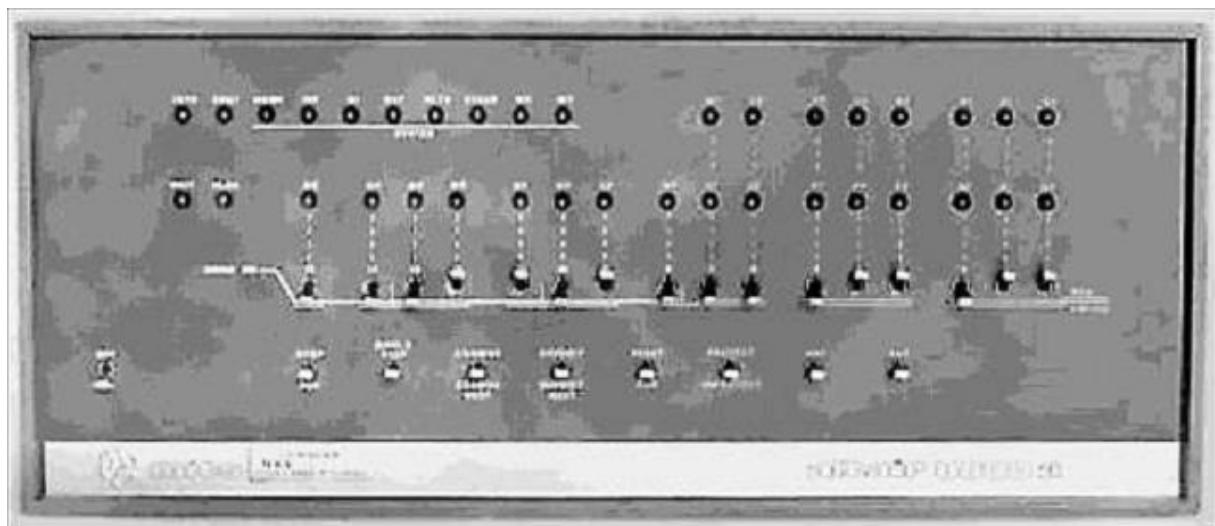


Hình 2.8. Ví dụ các thông báo

Các tiêu chuẩn và hướng dẫn

a. Trực giác:

Vào năm 1975 MITS Altair 8800 đã đưa ra bán máy tính cá nhân đầu tiên. Giao diện người sử dụng của nó không có cái gì khác ngoài những công tắc và bóng đèn không trực giác để sử dụng.



Hình 2.9. Hình ảnh chiếc máy tính cá nhân đầu tiên

b. Nhất quán:

Nhất quán với phần mềm của bạn và những phần mềm khác là một chìa khóa của thuộc tính. Tính nhất quán của phần mềm sẽ đảm bảo cho bạn không thực hiện lặp lại nhiều động tác.

c. Linh hoạt:

Phần mềm linh hoạt cung cấp cho ta nhiều lựa chọn và nhiều cách hoàn thành cùng một nhiệm vụ.

d. Thoải mái:

- *Sự thích hợp:* phần mềm có thể nhìn và cảm nhận thích hợp cho những cái nó làm và người sử dụng nó. Một ứng dụng tài chính sẽ điên rồ với những màu sắc sặc sỡ và hiệu ứng âm thanh. Một không gian game sẽ làm mất thời gian với những qui tắc.
- *Dùng lỗi:* Một chương trình cần phải cảnh báo người sử dụng và cho phép người sử dụng và cho phép họ khôi phục dữ liệu bị mất bởi lỗi.
- *Sự thực hiện:* Việc nhanh chóng thường xuyên không phải là tốt. Nhiều chương trình đã lóa sáng những thông báo lỗi quá nhanh để đọc. Nếu hệ thống chậm, nó ít nhất sẽ gửi những phản hồi đủ dài để nó chỉ ra nó vẫn làm việc hay không. Thanh Status bar là cách phổ biến trong trường hợp này.

e. Đúng đắn.

f. Hữu dụng.

10.6.3.2. Kiểm thử tính dễ tiếp cận phần mềm dành cho người khuyết tật

Dân số ngày càng già hóa và sự xâm nhập của công nghệ gần như vào mọi khía cạnh của cuộc sống của chúng ta. Chính vì vậy, tính tiện dụng của phần mềm trở nên quan trọng trong cuộc sống hằng ngày.

Mặc dù có rất nhiều kiểu tàn tật và dưới đây là một trong những cách rất khó khi sử dụng máy tính và phần mềm máy tính:

a. Sự suy giảm thị giác

Chứng mù màu, viễn thị, cận thị là những ví dụ điển hình của sự giới hạn trực quan. Rất nhiều người gặp khó khăn khi sử dụng phần mềm. Nếu bạn không thể nhìn thấy màn hình máy tính thì sao?

- b. Sự suy giảm thính giác

Những người như vậy thì không thể nghe thấy video hướng dẫn hay các âm thanh cảnh báo.

- c. Khó khăn khi di chuyển

Bị dị tật tay chân, họ không thể cầm chuột điều khiển, không thể nhấn bàn phím.

- d. Khuyết tật về ngôn ngữ và nhận thức

Nếu ai đó gặp vấn đề về trí nhớ kém hay khó đọc thì rất khó để sử dụng những giao diện phức tạp

10.6.4. Làm thế nào để thiết kế cho khả năng sử dụng cao

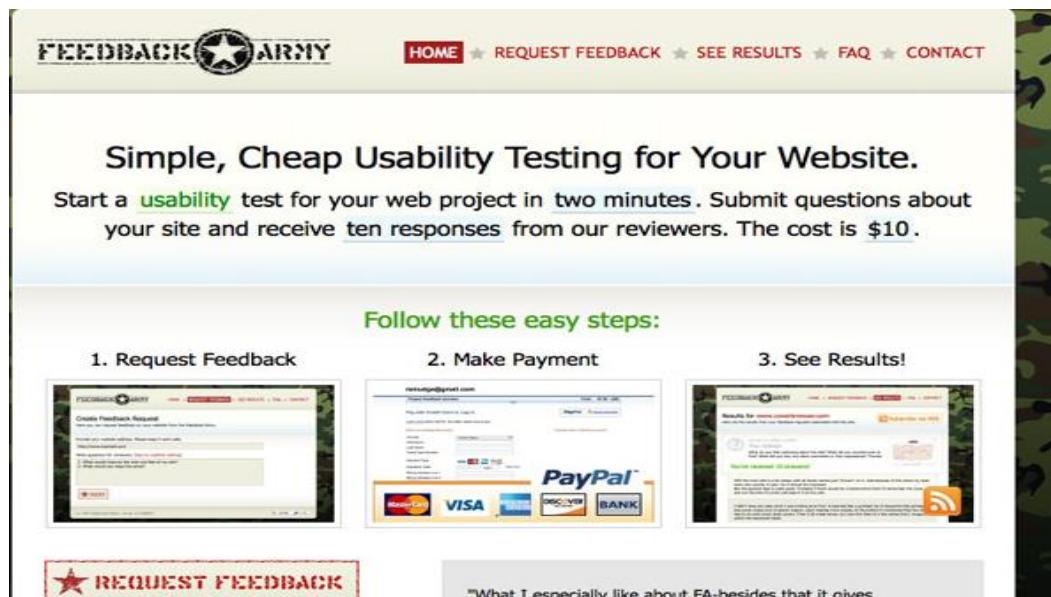
Để thiết kế cho khả năng sử dụng cao, các câu hỏi được đặt ra là:

- Mức độ dễ dàng đối với một người dùng chưa bao giờ nhìn thấy sản phẩm trước khi thực hiện các công việc cơ bản.
- Mức độ dễ dàng đối với một người đã từng sử dụng sản phẩm trước khi nhớ lại làm thế nào để thực hiện cùng các công việc đó.
- Mức độ hiệu quả đối với một người dùng đã từng sử dụng sản phẩm trước khi thực hiện nhanh chóng các công việc thường xuyên được sử dụng.
- Tần suất nào người sử dụng gặp lỗi khi sử dụng sản phẩm? Các lỗi này nghiêm trọng như thế nào?
- Mức độ kinh nghiệm của người dùng khi sử dụng sản phẩm.

10.6.5. Một số công cụ

- a. Feedback Army

Feedback Army có lẽ là cách nhanh nhất để có được thông tin phản hồi về trang web của bạn (ngoài trừ việc yêu cầu thông tin phản hồi từ các đồng nghiệp). Dịch vụ này được cung cấp bởi công cụ Mechanical Turk của Amazon.



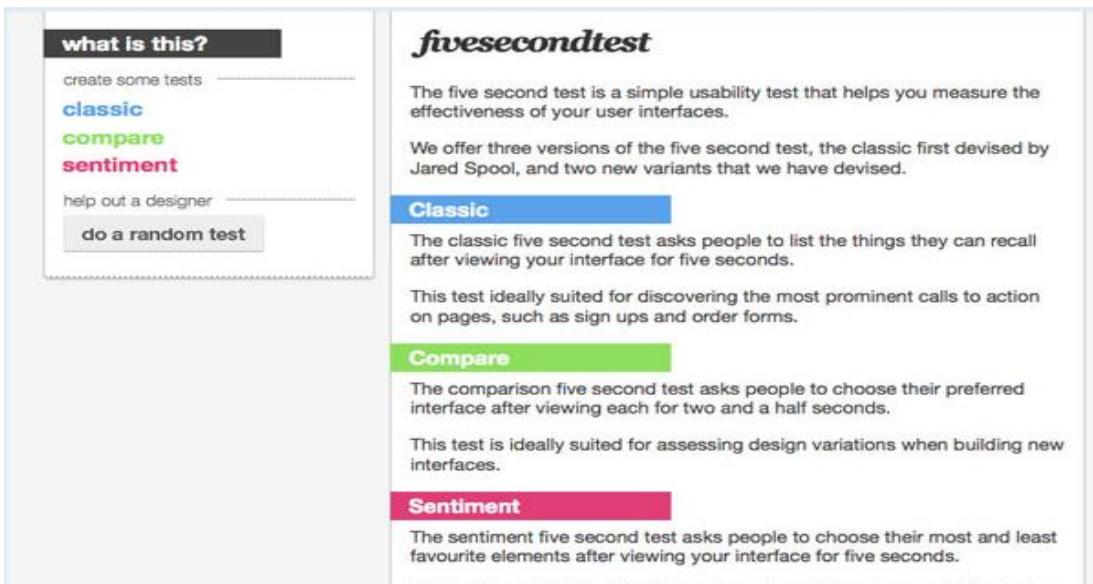
Hình 2.10. Giao diện chính phần mềm Feedback Army

Như tên của nó, Mechanical Turk là một “công cụ” mang tính nhân văn mạnh mẽ nhất được thiết kế để giải quyết các nhiệm vụ có thể được hoàn thành trong một thời gian ngắn. Feedback Army yêu cầu người dùng đặt ra các câu hỏi về trang web của bạn và nhanh chóng tập hợp thành thông tin phản hồi và hiển thị để bạn có thể cải thiện User Experience.

Chi phí tương đối rẻ: bạn chỉ mất 10 USD để mua 10 câu trả lời. Điều quan trọng là bạn phải xây dựng các câu hỏi của bạn một cách thật cẩn thận để nhận được thông tin phản hồi chất lượng cao nhất và hiệu quả nhất.

b. Five Second Test

Five Second Test là một dịch vụ thử nghiệm tính khả dụng miễn phí, nó sẽ cung cấp cho bạn 3 cách thử nghiệm khác nhau: “Classic – cổ điển”, “Compare – so sánh” và “Sentiment – tình cảm”. Cách thử nghiệm cổ điển sẽ hiển thị và sau đó ẩn màn hình chụp trang web của bạn và yêu cầu người dùng nhắc lại những gì họ nhớ. Cách thử nghiệm so sánh sẽ hiển thị hai màn hình chụp và yêu cầu người dùng tích vào màn những gì mà họ thích. Các thử nghiệm tình cảm hiển thị một trang web và yêu cầu người dùng chọn lọc ra những yếu tố họ ưa thích và những yếu tố họ không thích nhất. Những người kiểm tra chỉ mất 5 giây để cung cấp thông tin phản hồi sau khi xem câu hỏi.



Hình 2.11. Giao diện chính phần mềm Five Second Test

c. User testing

UserTesting cung cấp một phương pháp tiếp cận truyền thống để kiểm tra tính khả dụng. Hãy đưa UserTesting tới với người dùng trang web của bạn, và dịch vụ này sẽ chọn ra người dùng phù hợp để duyệt web của bạn.



Hình 2.12. Giao diện chính phần mềm User testing

10.7. Kiểm thử ngôn ngữ

10.7.1. Khái niệm

Kiểm thử ngôn ngữ có thể hiểu là kiểm thử để đảm bảo rằng phần mềm có thể xuất bản ra nhiều ngôn ngữ phổ biến trên thế giới.

10.7.2. Các vấn đề liên quan đến kiểm thử ngôn ngữ

Ngày nay hầu hết các phần mềm được tung ra trên toàn thế giới không phải một quốc gia nào đó hay một dạng ngôn ngữ nào đó. Microsoft đã tung ra Windows XP với sự hỗ trợ của 106 ngôn ngữ và thô âm từ Afrikaans tới Hungari hay Zulu. Hầu hết các công ty phần mềm đều làm như vậy, và thấy rằng thị trường Anh - Mĩ có lượng khách hàng tiềm năng chưa bằng một nửa. Nó làm cho việc kinh doanh có ý nghĩ hơn khi thiết kế và kiểm thử phần mềm có tính toàn cầu.

Phần này đề cập đến những vấn đề liên quan tới kiểm thử phần mềm được viết cho các quốc gia và ngôn ngữ khác nhau.

10.7.2.1. Ý nghĩa của từ ngữ và hình ảnh

Bạn đã bao giờ đọc một cuốn sổ về một thiết bị hay một thứ đồ chơi mà chỉ được chuyển đổi đơn giản sang một ngôn ngữ khác chưa? Có thể là dễ dàng khi chuyển đổi từng từ nhưng để tạo thành một khối tổng thể đúng nghĩa và dễ hiểu đòi hỏi đầu tư nhiều về thời gian và trí tuệ.

Những người phiên dịch tốt có thể làm được điều đó. Nếu họ thành thạo cả hai thứ tiếng, họ có thể biến bản text từ tiếng nước ngoài thành bản dịch có thể đọc được như bản gốc. Thật không may, điều bạn tìm thấy trong ngành công nghiệp phần mềm là thậm chí một bản dịch tốt cũng không đầy đủ.

10.7.2.2. Các vấn đề về dịch thuật

Quan trọng là bạn hay ai đó trong đội kiểm thử của bạn phải quen thuộc chút ít với ngôn ngữ các bạn đang kiểm thử. Dĩ nhiên, nếu bạn chuyển chương trình của mình sang 32 thứ tiếng khác nhau, điều này có thể rất khó khăn. Giải pháp là kí hợp đồng với một công ty kiểm thử địa phương hóa. Vô số các công ty như vậy trên thế giới có thể thực hiện kiểm thử gần như trên với bất kì ngoại ngữ nào.

- d) Sự mở rộng text

Ví dụ dễ thấy nhất về vấn đề dịch thuật có thể được xảy ra là do cái mà được gọi là sự mở rộng text. Dù tiếng Anh có vẻ dài dòng, nhưng thực sự khi được dịch ra các ngôn ngữ khác, chúng ta thường phải dùng nhiều kí tự hơn để nói về cùng một vấn đề.

Hình dưới cho thấy kích cỡ của một nút (button) cần mở rộng tới thế nào để lưu giữ text được dịch khi dịch hai từ rất phổ biến của ngôn ngữ máy tính.

Một quy tắc rất hay về dạng thumb là tăng 100% về kích cỡ đối với các từ. Ví dụ, trên một nút, hãy tăng 50% về kích cỡ đối với các câu và đoạn văn ngắn có các cấu trúc điển hình mà bạn có thể thấy ngay trong các hội thoại hoặc các message báo lỗi.



Hình 2.13. Ví dụ về sự mở rộng text

Khi được dịch sang ngôn ngữ khác từ Minimize hoặc Maximize có thể mở rộng về kích cỡ và buộc phải thiết kế lại để có thể lưu giữ được chúng

Bởi vì sự mở rộng này, bạn cần kiểm tra các vùng của phần mềm mà có thể bị ảnh hưởng bởi các text dài hơn. Hãy tìm kiếm text không được bao hoàn toàn, bị xén bớt hoặc bị gạch nối không đúng. Điều này có thể xảy ra bất kì ở đâu- trên màn hình, trong các cửa sổ, trong các boxes, các button,... Đồng thời hãy tìm kiếm các trường hợp nơi text có đủ chỗ để mở rộng nhưng vẫn bị như trên bởi đã đầy cái gì đó ra ngoài.

Để giải quyết vấn đề trên, một white – box tester có thể nắm bắt được vấn đề đó mà không cần hiểu rõ về ngôn ngữ đó.

e) ASCII, DBCS, và Unicode

Bảng mã ASCII sử dụng một byte gồm có 256 ký tự không đủ cho việc biểu diễn các ngôn ngữ khác nhau.

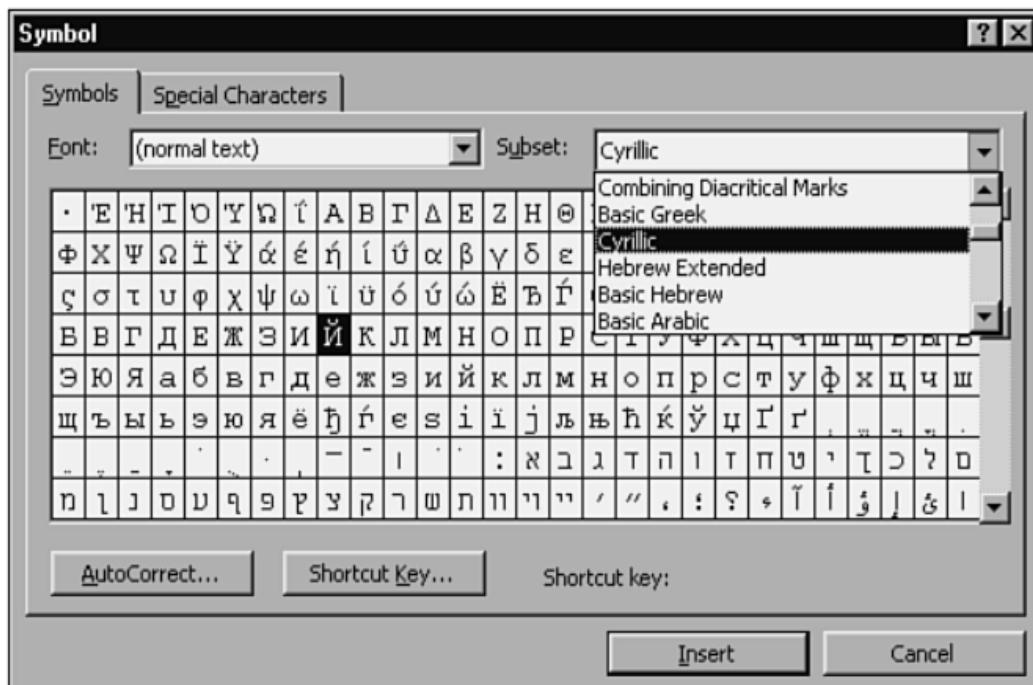
Ví dụ: như tiếng Nhật và tiếng Trung Quốc có hàng nghìn các ký tự đặc biệt.

Vì vậy hệ thống DBCS (Double Byte Character Set) sử dụng hai byte gồm 65.536 ký tự khác nhau để giải quyết cho vấn đề này. Nhưng những trang mã DBCS lại gặp một số vấn đề trở ngại trong khả năng tương thích.

Ví dụ: tài liệu của người Do Thái được tải trên máy tính của người Đức đang chạy một bộ xử lý văn bản tiếng Anh, kết quả là có thể gây ra sai về ngữ pháp.

Giải pháp chung cho những vấn đề này là chuẩn Unicode. Tìm hiểu “ Unicode là gì” từ Unicode Consortium website, www.unicode.org

Bởi vì Unicode là một chuẩn mở rộng hỗ trợ chính bởi những công ty phần mềm, những nhà sản xuất phần cứng, và những nhóm tiêu chuẩn khác, nó đang trở nên thông dụng hơn. Đa số những ứng dụng phần mềm hỗ trợ nó.



Hình 2.14. Hộp thoại hỗ trợ chuẩn Unicode

f) Phím nóng và phím tắt

Trong tiếng Anh, nó là “Search”. Trong tiếng Pháp, thì nó là “Réchercher”. Nếu phím nóng để lựa chọn cho sự tìm kiếm “Search” trong phiên bản tiếng Anh phần mềm của các bạn là Alt+S, mà nó cần sự thay đổi trong phiên bản tiếng Pháp.

Trong các phiên bản địa phương hoá phần mềm, bạn cần kiểm tra tất cả những phím nóng và công việc của những phím tắt sao cho dễ sử dụng và mang ý nghĩa gợi nhớ.

g) Mở rộng bảng ký tự

Bảng mã ASCII, ngoài 26 chữ cái hoa từ A..Z và 26 chữ cái thường từ a..z còn có những ký tự đặc biệt không có trên bàn phím, nhưng với sự hỗ trợ của bảng mã Unicode hoặc DBCS thì sự biểu diễn các ký tự đặc biệt trên không là vấn đề lớn.

Để kiểm tra bảng ký tự mở rộng đó bạn có thể kiểm tra việc nhập xuất của các ký tự đó có diễn ra như các ký tự thông thường không. Và kiểm tra xem chuyện gì sẽ xảy ra khi bạn thực hiện thao tác cut, copy và paste giữa các chương trình khác nhau?

10.7.2.3. Vấn đề địa phương hóa

Việc dịch chỉ là một vấn đề nhỏ trong quá trình kiểm thử ngôn ngữ. Với bất kỳ một văn bản dù có độ dài hay bao gồm nhiều kí tự khác nhau như thế nào đi chăng nữa thì đều có thể dịch một cách dễ dàng. Nhưng vấn đề ở đây là làm thế nào để các phiên bản dịch có thể phù hợp với tất cả mọi nơi.

Khi một phần mềm đã được dịch và kiểm tra cẩn thận thì thường được coi là chính xác và tin cậy. Nhưng nếu người lập trình viên không xem xét đến việc có thể xảy ra vấn đề địa phương hóa thì có lẽ phần mềm không còn chính xác, không còn được coi là chất lượng cao nữa.

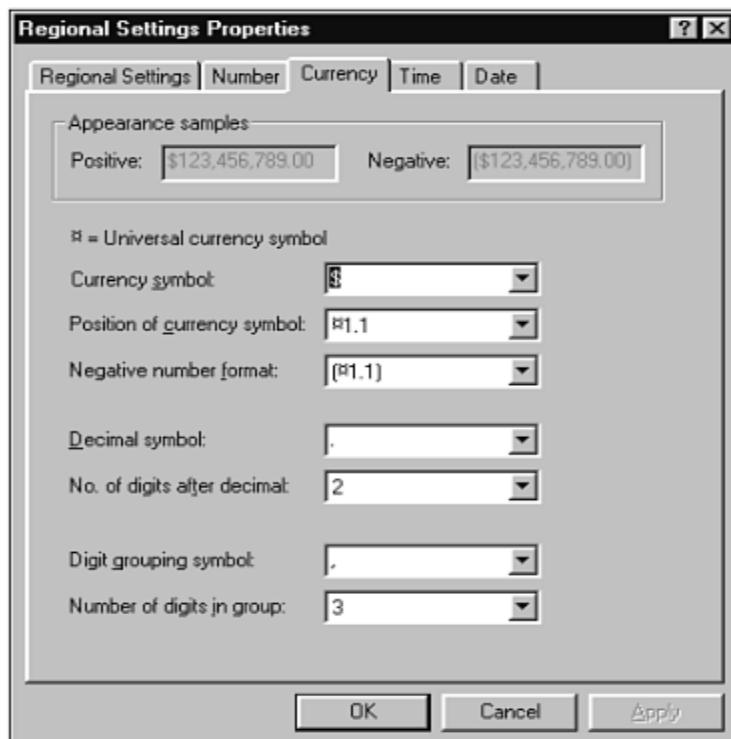
a) Một số ví dụ về sự địa phương hóa

Ví dụ về sự địa phương hóa: nếu bạn đang kiểm tra một sản phẩm mà sẽ được địa phương hóa, bạn cần xem xét cẩn thận nội dung để chắc chắn rằng nó sẽ thích hợp cho những nơi mà nó được sử dụng.

Danh sách các sản phẩm với nội dung mà chúng ta cần xem xét cho sự địa phương hóa:

Sample documents	Icons
Pictures	Sounds
Video	Help files
Maps with disputed boundaries	Marketing material
Packaging	Web links

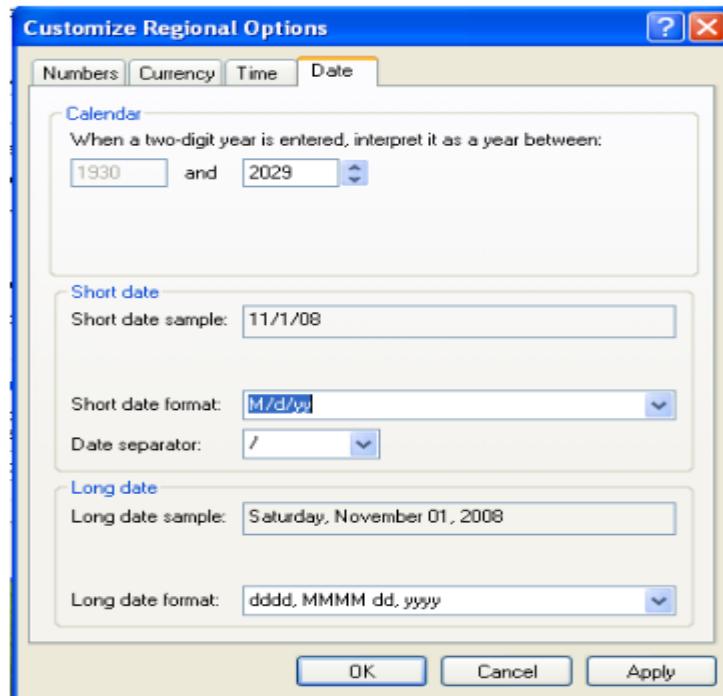
Ví dụ: Những tùy chọn và những thiết đặt trong windows cho phép một người sử dụng lựa chọn số, tiền tệ, thời gian, và ngày tháng sẽ được hiển thị:



Hình 2.15. Thiết đặt các thuộc tính địa phương hoá trong windows

Ví dụ: Bảng ngày tháng trên “Customize regional options” là kiểu ngày tháng ngắn.

Nếu bạn là tester phần mềm được địa phương hoá bạn sẽ phải quen thuộc sử dụng với tất cả các đơn vị đo lường để có thể đạt đến đích cuối cùng. Để kiểm tra phần mềm một cách đúng đắn, bạn cần tạo các lớp tương đương của việc phân chia dữ liệu.

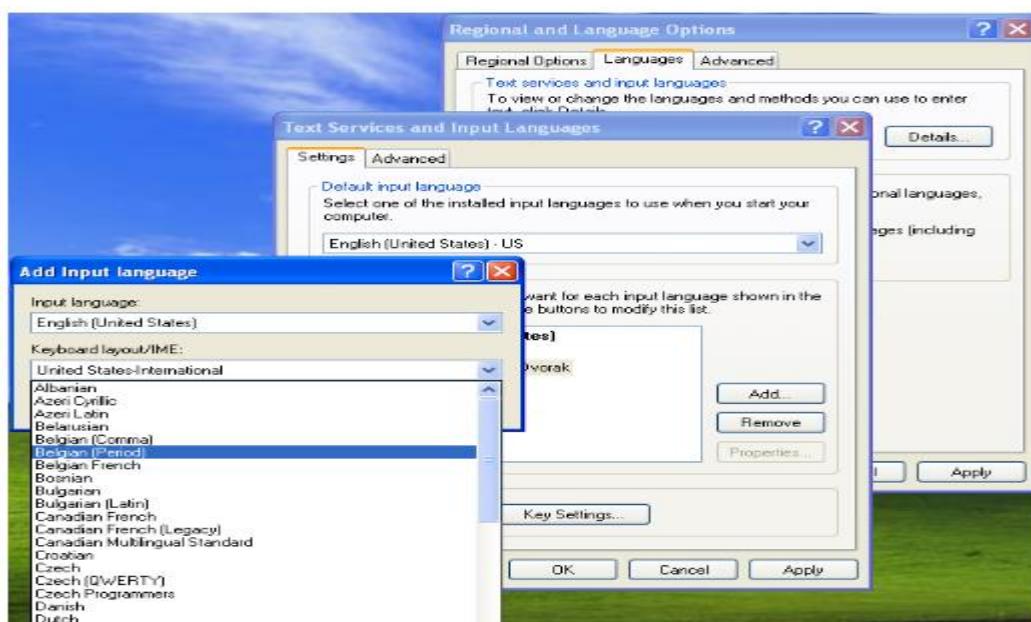


Hình 2.16. Tuỳ chọn địa phương hoá ngày, tháng, số, tiền tệ

10.7.2.4. Những vấn đề về cấu hình và sự tương thích

Việc kiểm thử về cấu hình và kiểm thử về tính tương thích là rất quan trọng khi kiểm thử địa phương hoá những phiên bản của phần mềm. Những vấn đề này có thể nảy sinh khi phần mềm tương tác với phần cứng và với các phần mềm khác.

a) Những vấn đề về cấu hình



Hình 2.17. Windows XP hỗ trợ 106 ngôn ngữ và 66 cách trình bày bàn phím khác nhau

Ví dụ về hai cách trình bày bàn phím khác nhau thiết kế cho những nước khác nhau:



Hình 2.18. Bàn phím \hat{A} rập



Hình 2.19. Bàn phím nước Pháp

Bạn sẽ thấy mỗi bàn phím biểu diễn bằng những từ khoá đặc biệt đối với ngôn ngữ của nước mình, nhưng vẫn lấy tiếng Anh làm đặc tính chung. Tiếng Anh được coi như là một ngôn ngữ thứ hai trong nhiều nước, và cho phép bàn phím sử dụng cả phần mềm bằng tiếng bản xứ của mình lẫn tiếng Anh.

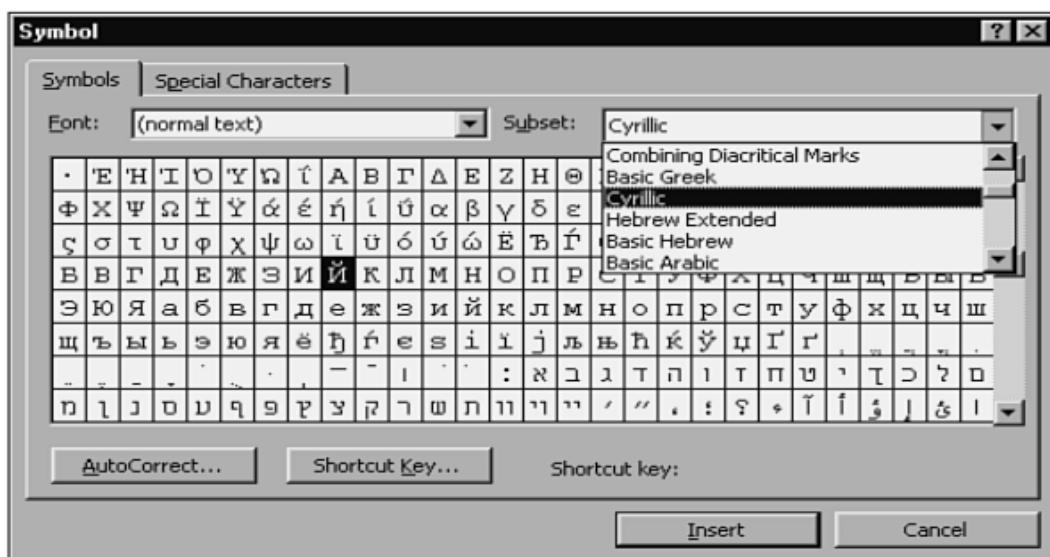
Các thiết bị khác như máy in, giả sử bạn cần in tất cả các đặc tính phần mềm của bạn gửi tới cho ai đó và đúng định dạng đầu ra trên nhiều kích thước giấy được dùng trong những nước khác nhau. Nếu phần mềm của bạn sử dụng Modem thì nó sẽ liên quan đến đường điện thoại hay những giao thức truyền thông khác nhau ...

Chú ý: Khi thiết kế những phân vùng tương đương, bạn đừng quên xem xét tất lại cả những phần cứng và phần mềm có liên quan đến. Bao gồm phần cứng, ổ cứng, và hệ điều hành.

b) Những vấn đề về sự tương thích

Cũng như sự thử cấu hình, sự thử tương thích dữ liệu đảm nhiệm một ý nghĩa hoàn toàn mới khi bạn thêm sự địa phương hóa vào chương trình.

Hình ảnh sau cho thấy sự thử tương thích dữ liệu của phần mềm được địa phương hoá có thể trở lên khá phức tạp thế nào.



Hình 2.20. Sơ phác tap về tính tương thích của phần mềm được địa phương hoá

Trong thời gian dữ liệu dịch chuyển vòng tròn, với tất cả sự chuyển biến và xử lý những đơn vị đo lường và kí tự mở rộng như vậy thì đã xuất hiện nhiều chỗ lỗi. Một số lỗi này có thể đúng như trong thiết kế.

Chú ý:

- Để tránh tốn nhiều thời gian, chi phí cho địa phương hoá phần mềm cho phù hợp với nhiều nơi thì chúng ta nên địa phương hoá ngay từ khi dự án bắt đầu.
 - Số lượng kiểm thử sự địa phương hoá là một việc khó xác định vì tuỳ thuộc vào vấn đề địa phương hoá và kinh nghiệm của tester về vấn đề địa phương hoá đó.

- Trong việc kiểm thử địa phương hoá, những người kiểm thử hộp trắng sẽ kiểm tra mã code, sử dụng các đơn vị đo lường, và các đặc tính mở rộng... Những người kiểm thử hộp đen cần thận xem xét đặc tả và sản phẩm để địa phương hóa những vấn đề như văn bản trong đồ họa và những vấn đề cấu hình.

10.8. Kiểm thử tài liệu

10.8.1. Khái niệm

Kiểm thử tài liệu được hiểu là đảm bảo độ chính xác của các tài liệu liên quan đến dự án như tài liệu đặc tả, tài liệu phân tích thiết kế,...

10.8.2. Các vấn đề về kiểm thử tài liệu

Ngày nay tài liệu phần mềm không chỉ đơn giản là tệp Readme nữa nó bao gồm nhiều loại tài liệu khác nữa. Vì vậy việc kiểm thử tài liệu cũng phức tạp hơn.

10.8.2.1. Các kiểu tài liệu phần mềm

Những tài liệu trong phần mềm bao gồm những gì:

Tài liệu của người sử dụng: sự xuất hiện của tài liệu trực tuyến với sự linh hoạt và hữu ích làm cho tài liệu trên giấy không còn được thông dụng như trước nữa. Ngày nay hầu hết các phần mềm đều hướng tới sự nhỏ gọn ngay từ những chi tiết đầu tiên. Những tài liệu trực tuyến có thể được phân bố trên các phương tiện truyền thông hoặc trên những website.

Trợ giúp trực tuyến: trợ giúp trực tuyến được lồng vào với tài liệu của người sử dụng, đôi khi trợ giúp trực tuyến còn được thay thế tài liệu người sử dụng. Trợ giúp trực tuyến giúp cho việc tìm kiếm trở nên dễ dàng hơn. Ví dụ, hãy nói với tôi làm thế nào để sao chép một đoạn văn bản giữa hai chương trình!

Các tài liệu hướng dẫn (Tutorials, wizards, and CBT-Computer Based Training): người sử dụng đặt ra câu hỏi sau đó phần mềm sẽ hướng dẫn người sử dụng thông qua từng bước để hoàn thành nhiệm vụ. Một trong những vấn đề đó chính là trợ giúp của Microsoft's Office “paper clip guy”



Hình 2.21. Trợ giúp của Microsoft Office

Thông báo lỗi (Error messages): vẫn đề về thông báo lỗi đã được đề cập đến trong cuốn sách này một vài lần nhưng sau đó nó lại bị lãng quên. Cuối cùng những vấn đề này chỉ nằm trong tài liệu.

10.8.2.2. Tầm quan trọng của việc kiểm thử tài liệu

Người sử dụng không quan tâm tới phần mềm được tạo ra như thế nào bởi ai. Cái mà người sử dụng quan tâm chính là chất lượng của sản phẩm sau khi đã được đóng gói.

- Nếu phần hướng dẫn cài đặt bị sai hoặc nếu những thông báo lỗi không đúng khi đó sẽ làm cho người sử dụng sang hiểu sai vấn đề.
- Người sử dụng sẽ gửi những lỗi đó cho nhà cung cấp sản phẩm, khi đó một vài lỗi phần mềm đó sẽ được tìm thấy bởi người kiểm thử phần mềm.

Một tài liệu phần mềm tốt nó sẽ làm cho chất lượng của toàn bộ phần mềm được tốt hơn:

- Tận dụng được tính khả dụng.
- Cải thiện sự tin cậy, nếu người sử dụng đọc tài liệu, sử dụng phần mềm và không tin cậy vào phần mềm đó. Kiểm thử phần mềm và tài liệu là một trong những cách tốt nhất để loại bỏ những lỗi đó.

- Nó giúp cho việc giảm chi phí: chi phí bỏ ra ít hay nhiều tùy thuộc vào quá trình phát hiện sớm hay muộn của khách hàng, nếu khách hàng dùng sản phẩm và phát hiện ra lỗi sớm thì chi phí bỏ ra sửa lại sẽ không cao, nếu khách hàng dùng sản phẩm và phát hiện ra lỗi muộn thì chi phí bỏ ra sửa lại là rất đắt.

Lưu ý:

- Là một Tester bạn nên quan tâm tới tài liệu phần mềm giống như Coder đang nỗ lực hoàn thành sản phẩm.
- Nếu bạn không coi trọng tài liệu thì toàn bộ phần mềm mà bạn kiểm thử sẽ không đạt kết quả cao.

10.8.2.3. Bạn tìm thấy gì khi xem xét tài liệu

Kiểm thử tài liệu có thể xét ở hai trường hợp:

- Nếu tài liệu không phải là mã, thì kiểm thử thường là một quá trình tĩnh
- Nếu tài liệu và mã có ràng buộc gần với nhau, kiểm thử trở thành quá trình động

Lưu ý:

- Nếu có đoạn mã đơn giản, bạn hãy gõ nó vào máy và thử như mô tả, với cách tiếp cận thực tế đơn giản này, bạn sẽ tìm ra những lỗi trong cả phần mềm và tài liệu.
- Dù cho tài liệu có mã hay không, thì hãy đọc tài liệu cẩn thận, thực hiện từng bước, khảo sát tất cả các hình, thử mọi ví dụ.

Danh sách các trường hợp kiểm thử (là cơ sở để xây dựng các trường hợp kiểm thử tài liệu):

Danh mục	Công việc thực hiện
Thuật ngữ	<ul style="list-style-type: none"> - Thuật ngữ có thích hợp với người sử dụng không? - Thuật ngữ có được sử dụng thường xuyên và chính xác không? - Một thuật ngữ được viết tắt có phải là thuật ngữ chuẩn hay không? - Nếu viết tắt tên công ty thì điều gì sẽ xảy ra?

Nội dung và đề tài	<ul style="list-style-type: none"> - Liệu đề tài bất kỳ nào cũng có thể sinh lỗi? - Những loại đề tài nào không thể khái quát nếu một đặc tính được lấy ra từ sản phẩm và người viết tài liệu không hề biết?
Thực tế	<ul style="list-style-type: none"> - Liệu mọi thông tin về mặt thực tế và kỹ thuật đều đúng? - Hãy kiểm tra nội dung, chỉ số, và các chương khác nhau của tài liệu. - Hãy thử với các website URL. Liệu các sản phẩm này có hỗ trợ số phone đúng hay không?
Thực hiện từng bước	Đọc tất cả văn bản cẩn thận và chậm rãi. Theo sau là những chỉ dẫn chính xác. Không đặt ra giả thiết gì! Và sửa những lỗi sai.
Màn hình	Kiểm tra các hình với độ chính xác cao. Bạn không được sao chép hình từ phần mềm đã có sự thay đổi. Và kiểm tra cùng với tên của hình đó.
Mẫu và ví dụ	Nhập vào và sử dụng từng mẫu như một khách hàng. Nếu nó là mã, có thể sao chép hoặc gõ vào máy và chạy nó.
Đánh vần và ngữ pháp	Đây không phải là một lỗi gây rắc rối lớn. Bạn đừng quên kiểm tra hoặc bỏ qua những thuật ngữ hay kỹ thuật chuyên ngành. Đó cũng có thể là những kỹ thuật kiểm tra bằng tay, như trong việc sao chép hình và vẽ hình.

Cuối cùng, nếu bạn phải xét nhiều trường hợp kiểm thử, hãy sử dụng kỹ thuật phân lớp tương đương để chọn ra các trường hợp điển hình để kiểm thử.

10.8.2.4. Thực tế của việc kiểm thử tài liệu

Một Tester sẽ gặp nhiều khó khăn trong quá trình kiểm thử bởi đôi khi người viết tài liệu cho phần mềm (writer) không phải là chuyên gia. Cách tốt nhất là nên làm việc gần gũi với writer để họ hiểu những khó khăn mà bạn gặp phải và giải thích rõ ràng hơn trong tài liệu.

Một tài liệu của sản phẩm phần mềm cần phải được cố định trước khi phần mềm hoàn thành. Nếu một chức năng nào đó của phần mềm bị thay đổi hoặc gây lỗi, tài liệu sẽ không kịp thời thay đổi theo. Đó là lý do tệp Readme ra đời.

Tệp Readme phản ánh những thay đổi cuối cùng của phần mềm cho người sử dụng biết mà chúng không được thay đổi trong tài liệu.

Để việc kiểm thử tài liệu có kết quả, hãy tiến hành trên các tệp đi kèm với phần mềm: Readme, hướng dẫn cài đặt... các tài liệu quảng cáo; các bản đăng ký, thậm chí là các tiêu chuẩn của phần mềm; cùng với các hình, mẫu, và các ví dụ...

Là một Tester, nếu bạn kiểm thử các tài liệu đúng cách, bạn sẽ tìm thấy những lỗi trước khi khách hàng của bạn phát hiện ra chúng.

10.9. Kiểm thử khả năng phục hồi

10.9.1. Khái niệm

Phục hồi là khả năng khởi động lại các hoạt động sau khi tính toán vẹn của ứng dụng bị mất.

Kiểm thử khả năng phục hồi là các kiểm thử được tiến hành nhằm làm hệ thống ngừng hoạt động và đánh giá khả năng phục hồi sau đó.

10.9.2. Mục đích của kiểm thử khả năng phục hồi

Để đảm bảo các hoạt động có thể được tiếp tục sau khi xảy ra thảm họa.

Kiểm thử khả năng phục hồi xác minh quá trình phục hồi và hiệu quả của quá trình phục hồi.

Bảo đảm hệ thống có khả năng khôi phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến...

10.9.3. Các vấn đề liên quan đến kiểm thử khả năng phục hồi

Nhiều hệ thống cần phải phục hồi sau lỗi, để lại tiếp tục xử lý trong một thời gian đã đặc tả trước.

Có trường hợp, hệ thống cần thử lỗi, nghĩa là việc xử lý lỗi bắt buộc không được làm ngưng hoạt động của toàn hệ thống.

Trong trường hợp khác, lỗi phải được khắc phục theo từng chu kỳ được đặc tả nếu không thiệt hại kinh tế nghiêm trọng sẽ xuất hiện.

Kiểm thử phục hồi là bắt buộc phần mềm hỏng theo nhiều cách để xem khả năng phục hồi của nó đến đâu.

Có hai cách phục hồi:

- Phục hồi tự động (được thực hiện bởi bản thân hệ thống): Sau khi phục hồi dữ liệu, hệ thống tự khởi động lại thì được đánh giá là đúng đắn.
- Phục hồi đòi hỏi sự can thiệp của con người: cần đánh giá thời gian trung bình để sửa chữa trong giới hạn cho phép có đúng hay không.

Với các hệ thống có khả năng phục hồi tự động, chúng ta cần đánh giá các công đoạn tái thiết lập thông số, khả năng khôi phục dữ liệu và tái khởi động.

Với các trường hợp đòi hỏi khởi động lại thủ công, chúng ta cần đánh giá thời gian ngừng để sửa chữa và trong một số trường hợp đánh giá cả chi phí cho việc khôi phục.

Kiểm thử khả năng phục hồi bao gồm việc quay trở lại điểm mà toàn vẹn hệ thống được biết đến sau đó xử lý lại cho đến điểm thất bại.

Thời gian để thực hiện phục hồi phụ thuộc vào:

- Số điểm khởi động lại.
- Khối lượng ứng dụng.
- Trình độ, kỹ năng của người thực hiện các hoạt động phục hồi cũng như các công cụ có sẵn để phục hồi.

Cách thực hiện kiểm thử khả năng phục hồi:

Dùng các thủ tục, phương pháp, công cụ và kỹ thuật để đánh giá được đầy đủ.

Sau khi hệ thống được phát triển, một thất bại có thể xuất hiện trong hệ thống và lúc này kiểm tra xem hệ thống có thể phục hồi được không.

Thay vì quan tâm đến toàn bộ kiểm thử phục hồi thì nên quan tâm đến từng phân đoạn và các kiểu cấu trúc khác.

Kiểm thử khả năng phục hồi thực hiện khi:

Khi người dùng yêu cầu sự liên tục của hệ thống là cần thiết để hệ thống thực hiện và hoạt động đúng chức năng.

Người sử dụng nên ước tính thiệt hại, khoảng thời gian để thực hiện kiểm thử phục hồi.

Ví dụ: kiểm thử việc mất mát thông tin, mất mát toàn vẹn dữ liệu hay đánh giá độ an toàn của quá trình sao lưu giữ liệu.

Như vậy từ sự phân tích ở trên, kiểm thử viên có thể dựa vào đó để thiết kế các tình huống kiểm thử và thực thi kiểm thử khả năng phục hồi một cách hiệu quả đồng thời lựa chọn được công cụ kiểm thử tự động phù hợp.

10.9.4. Công cụ kiểm thử khả năng phục hồi

Chúng ta có thể sử dụng công cụ LoadRunner để kiểm thử khả năng phục hồi.

BÀI 11. QUẢN LÝ LỖI PHẦN MỀM - BUG MANAGEMENT

11.1. Các thành phần của lỗi

11.1.1. Nội dung lỗi - Defect concept

Defect là gì?

Một defect là bất kỳ lỗi gì được tìm thấy bởi các hoạt động kiểm thử và rà soát (tất cả các lỗi được tìm thấy bởi người rà soát bên trong, bên ngoài và khách hàng)

Nếu bạn xem xét bất kỳ quan điểm để sửa lỗi, thì quan điểm là nên ghi nhận như là một defect trong công cụ xác định (ví dụ file excel, công cụ quản lý lỗi).

Bạn sẽ làm gì khi lỗi được tìm thấy?

- Các lỗi cần được ghi và phân công tới các lập trình viên mà có thể sửa được lỗi đó

- Sau khi vấn đề được giải quyết, các lỗi đã sửa lên test laik, và các xác định này nâng cấp các yêu cầu cho việc test hỏi qui để kiểm tra rằng việc sửa lỗi này sẽ không tạo ra vấn đề nào khác nữa.

- Quản lý lỗi nên đóng gói thành một qui trình

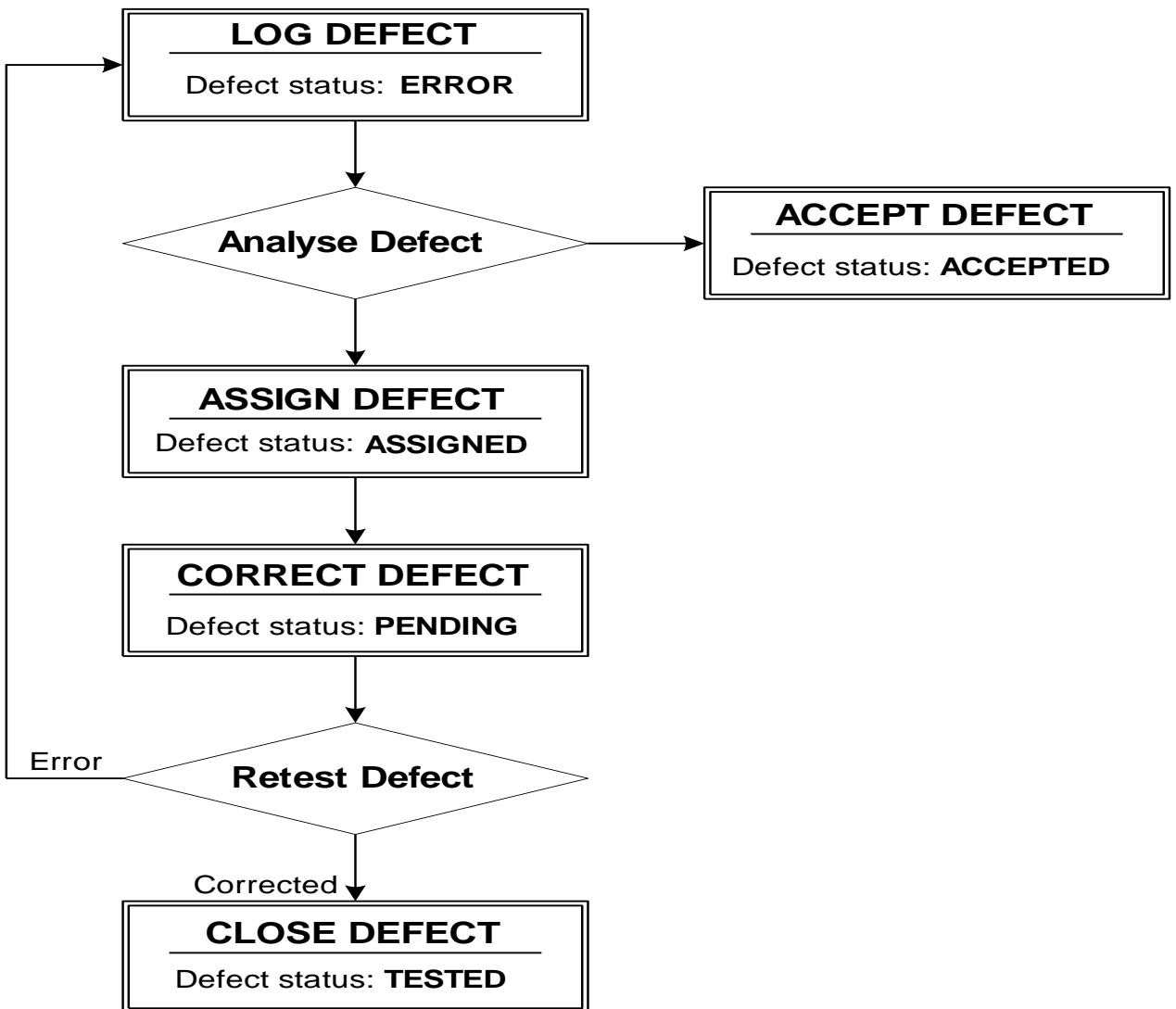
Các khái niệm

Defect logging: khởi tạo báo cáo và ghi nhận lại việc tìm và khám phá ra một lỗi

Defect tracking: điều khiển và ghi lại việc đã xảy ra với mỗi một lỗi sau khi các lỗi đó từ khi bắt đầu được tìm ra, cho đến lúc kết thúc giải quyết vấn đề của lỗi đó.
Việc dò thông qua trạng thái lỗi (defect status)

Defect status: mô tả trạng thái hiện thời của lỗi đã được ghi nhận

Vòng đời của một lỗi:



11.1.2. Trạng thái lỗi – Defect status

Có 6 trạng thái lỗi sau:

OPENED: Lỗi chưa được sửa, hoặc đã được sửa nhưng chưa thỏa mãn theo yêu cầu

ASSIGNED: Lỗi được xem xét lại và phân công để sửa nó

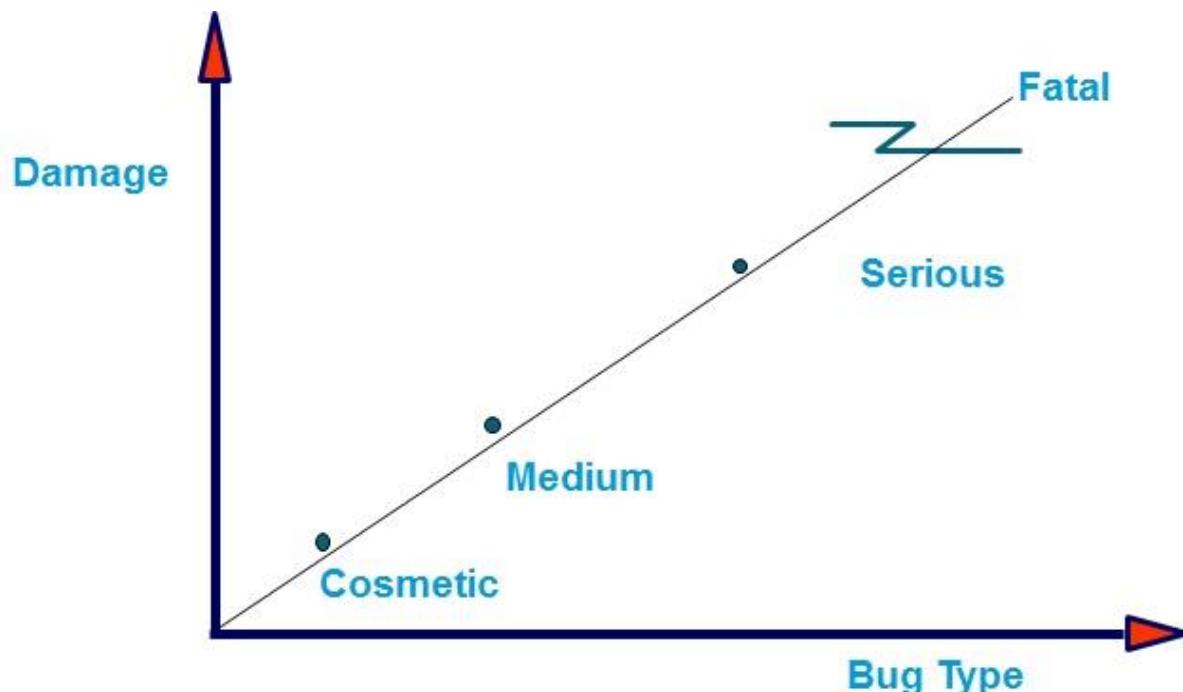
FIXED: Lỗi đã được sửa và đang đợi để test lại

CLOSED: Lỗi đã được sửa và thỏa mãn yêu cầu

ACCEPTED: Lỗi chưa được sửa thỏa mãn yêu cầu nhưng điều đó có thể chấp nhận được bởi tác giả hoặc khách hàng

CANCELLED: Nó không là một lỗi hoặc lỗi đã được loại bỏ bởi các hoạt động khác hơn là việc sửa lỗi

11.1.3. Mức độ nghiêm trọng của lỗi - Defect Severity



Mức độ nghiêm trọng - Fatal:

- Là một vấn đề lớn, nơi mà một phần lớn các chức năng hoặc các thành phần hệ thống quan trọng là hoàn toàn bị hỏng
- Hệ thống/ ứng dụng bị treo vĩnh viễn

- Không phục hồi CSDL bị ngắt
- Không có cách giải quyết và việc kiểm thử không thể tiếp tục
- Ngăn chặn người dùng sử dụng hệ thống tương lai

Mức độ nghiêm trọng - Serious:

- Chức năng chính không hoạt động hoặc hoạt động không chính xác
- Vấn đề lớn là nơi một phần lớn các chức năng hoặc các thành phần của hệ thống là làm việc không đúng cách
- Có cách giải quyết và việc kiểm thử có thể tiếp tục
- Bảo mật: các người sử dụng thuộc về vai trò xem/ thực thi các chức năng (menu) của vai trò B
- Mức độ nghiêm trọng được phân công phụ thuộc vào ưu tiên các chức năng của dự án

Mức độ nghiêm trọng - Medium:

- Chức năng nhỏ không hoạt động hoặc hoạt động không đúng
- Bất kỳ trường hợp đặc biệt bao gồm dữ liệu không hợp lệ, giá trị ranh giới, .. chức năng chính trả về không chính xác
- Dư thừa đầu ra
- Ảnh hưởng hiệu suất

Mức độ nghiêm trọng - Comestic:

Là các lỗi sai nhỏ như:

- Kiểu
- Lỗi chính tả/ ngữ pháp trên màn hình giao diện người dùng/ tin nhắn
- Cẩn lè, trường, định dạng, nhãñ
- Thứ tự tab, thiết lập nút mặc định, con trỏ mặc định, phím tắt...
- Tiêu chuẩn code: tên các đối tượng, biến, chú thích...

Defect priority:

Có 4 thứ tự ưu tiên sửa lỗi: ngay lập tức (Immediately), cao (High), trung bình (Medium), thấp (low)

11.1.4. Các kiểu lỗi - Defect type

Hiệu nhầm yêu cầu

Lỗi là do sự nhầm lẫn/ hiệu sai giữa toàn đội phát triển và khách hàng.

Ví dụ 1:

Khách hàng yêu cầu $C=A+ABS(B)$

Nhưng lập trình viên thì lại cài đặt: $C=A+B$

Ví dụ 2:

Yêu cầu: Trường năm nhận các giá trị từ 1990 đến 2020

Lập trình viên: trường năm nhận bất kỳ giá trị nào

Lỗi tính năng:

Lỗi là do thiếu tính năng hoặc tính năng chưa hoàn thiện

Ví dụ: khách hàng yêu cầu 3 chức năng: thêm mới, sửa đổi và xóa người dùng.

Nhưng lập trình viên chỉ thực hiện hai chức năng: Thêm mới và xóa người dùng

Logic nghiệp vụ

Lỗi có thể là do sự hiểu lầm giữa các thành viên trong đội khi được phân công công việc

Chức năng dư thừa có thể là nguyên nhân các lỗi về logic nghiệp vụ

Các yêu cầu trong tài liệu không rõ ràng, các giả định của nhóm phát triển không đáp ứng được kỳ vọng của khách hàng, nên sửa

Ví dụ: trong một hệ thống:

Trang 1: khi người dùng click nút “Back”, hệ thống sẽ trình diễn trang trước với trạng thái giống như người dùng đã sử dụng trước đó

Trang 2: Khi người dùng click nút Back thì hệ thống có hiển thị trang trước nhưng ở trạng thái đã bị refresh

Logic code

Do bất cẩn

Kỹ năng kỹ thuật

Xác nhận dữ liệu

Vấn đề về code

Ví dụ: Đối với thành phần Listview trong VB, ListItem được đánh chỉ mục từ 1..n (n=số ListItem trong Listview)

Nếu lập trình viên chỉ lấy chỉ số như sau: i=0 to n-1, thì chỉ số n được bỏ qua, vậy nên bắt đầu là 1

Đó là lỗi code logic, “chỉ số nằm ngoài biên”

Giao diện người dùng

Lỗi ở giao diện, layout

Ví dụ:

- Lỗi chính tả/ ngữ pháp trên màn hình giao diện người dùng/ tin nhắn
- Căn lề, trường, định dạng, nhẫn
- Thứ tự tab, thiết lập nút mặc định, con trỏ mặc định, phím tắt...

Hiệu năng

Tốc độ xử lý nghèo nàn

Hệ thống có thể bị sụp bởi kích thước đầu vào

Các vấn đề về tải hoặc bộ nhớ

Những vấn đề thường xuyên xảy ra với nhiều người dùng

Rà soát mã

Các vấn đề chính liên quan đến mã:

Tiêu chuẩn mã: căn lề, layout, chú thích, lịch sử thay đổi

Không theo mã qui ước

Không theo yêu cầu hoặc thiết kế chi tiết

Rà soát tài liệu

Tài liệu dự án: tài liệu test, kế hoạch, vv..

Lỗi được tìm thấy trong quá trình rà soát tài liệu: bản kế hoạch dự án, SRS, test plan, ... liên quan đến tiêu chuẩn tài liệu (mẫu, phiên bản, header/footer) ...

Lỗi ngữ pháp

Thiết kế

Các vấn đề có liên quan tới việc thiết kế

11.2. Mẫu Defect log

Sử dụng mẫu (Defect log) để ghi và dò lỗi. Mẫu Defect log bao gồm các thành phần:

Defect ID: chỉ mục lỗi để giúp quản lý một cách dễ dàng

Module: địa chỉ xác định lỗi (tên modul, tên chức năng)

Description: mô tả lỗi đã tìm thấy (một cách chi tiết)

Type: lựa chọn loại lỗi

Severity: lựa chọn mức độ nghiêm trọng của lỗi

Priority: lựa chọn mức độ ưu tiên để sửa lỗi

Status: lựa chọn trạng thái lỗi

Create date: Điện giá trị ngày tìm thấy lỗi

Assigned to: tên thành viên mà được phân công sửa lỗi đó

Corrective action: Các hoạt động sửa lỗi được giao

Hiệu quả của việc ghi lỗi:

- Kịp thời có thể ghi lại các lỗi
- Thể hiện tính nhất quán
- Ghi đầy đủ thông tin
- Dễ dàng tái sử dụng

Mẫu Defect log sau:

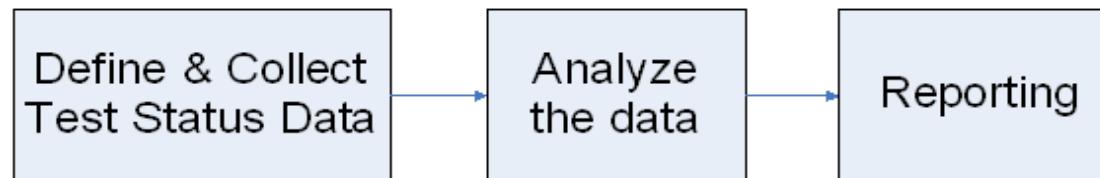
Bug list

Defect ID	Module	Description	Type	Severity	Priority	Status	Created Date	Assigned to	Corrective Action
1	Title section	<p><i><Describe defect and expected result></i></p> <p>Ex: Default value of Status field = blank is incorrect</p> <p>Expected result: when create new document, default value of [Status] field must be "Oppened"</p>	<p><i><Type of defect></i></p> <p>Ex: User Interface</p>	<p><i><Severity of defect></i></p> <p>Ex: Cosmetic</p>	<p><i><Priority of defect></i></p> <p>Ex: High</p>	<p><i><Status of defect></i></p> <p>Ex: Assigned</p>	<p><i><Created date of defect></i></p> <p>Ex: 21-Sep-2008</p>	<p><i><Person who is assigned to fix defect></i></p> <p>Ex: TungNT</p>	<p><i><Actions used to fix defect></i></p> <p>Ex: change code to update correct status</p>

11.3. Tổng quan về test report

- * Test report chỉ ra trạng thái của việc kiểm thử
 - * Các tài liệu thiết kế test đã thực sự được thực hiện
 - * Trạng thái của việc test trên
 - * Các phần còn lại (chưa được test)
 - * Tiến trình test chung
- * Test report nên được tài liệu hóa và trực quan với đội dự án

11.4. Quy trình test report



Khi nào test report được tạo:

- Theo mốc hoạch định (milestones)
- Kết thúc giai đoạn test
- Các lỗi trọng điểm đã được tìm thấy

Trạng thái test:

Các danh mục dữ liệu:

- Dữ liệu kết quả test
- Kết quả test và kết quả kiểm chứng test
- Defects

Dữ liệu kết quả test:

- Results pictures: Pass/fail result pictures
- Các thành phần phần mềm, ứng dụng cho test
- Hệ nền – Môi trường phần cứng và phần mềm với hệ thống phần mềm sẽ hoạt động

Test case và kết quả kiểm chứng test:

Các kết quả mô tả với sự tùy biến giữa đâu ra mong đợi và thực tế

Dưới đây là kết quả của kỹ thuật test được sử dụng bởi đội test với thực thi test

- ✓ Test cases – The type of tests that will be conducted during the execution of tests, which will be based on software requirements
- ✓ Inspections – A verification of process deliverables against deliverable specifications.
- ✓ Review checklists – Verification that the process deliverables/phases are meeting the user's true needs.

Defect:

The description of defects:

- ✓ Name of defect

- ✓ Status of the defect
- ✓ Severity of the defect
- ✓ Type of defect
- ✓ Module code which contains defect
- ✓ How the defect was discovered
- ✓ Data the defect uncovered
- ✓ Test case of defect

11.5. Cấu trúc của test report

Defect report:

- Defect Status Report: see current status of all defect in period of time
- Defect Distribution: see defect distribute to QC Activities
- Defect Re-Open: see the number defects are re-opened
- Defect Summary: see number defect based on Product Type
- Defect Trend: see trend of defect

11.6. Ví dụ test report

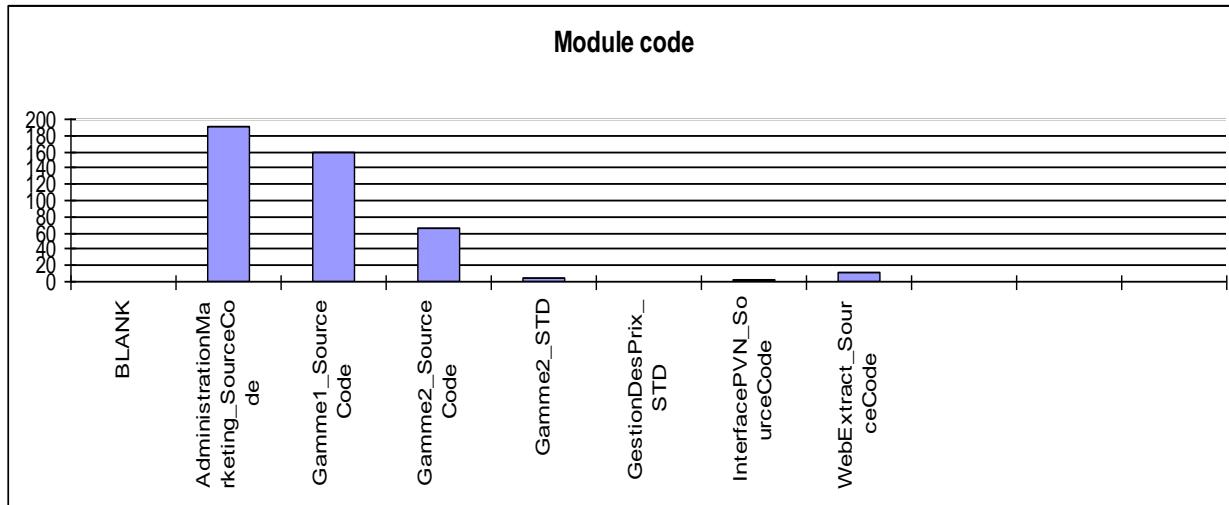
Ví dụ:

- General Status Report

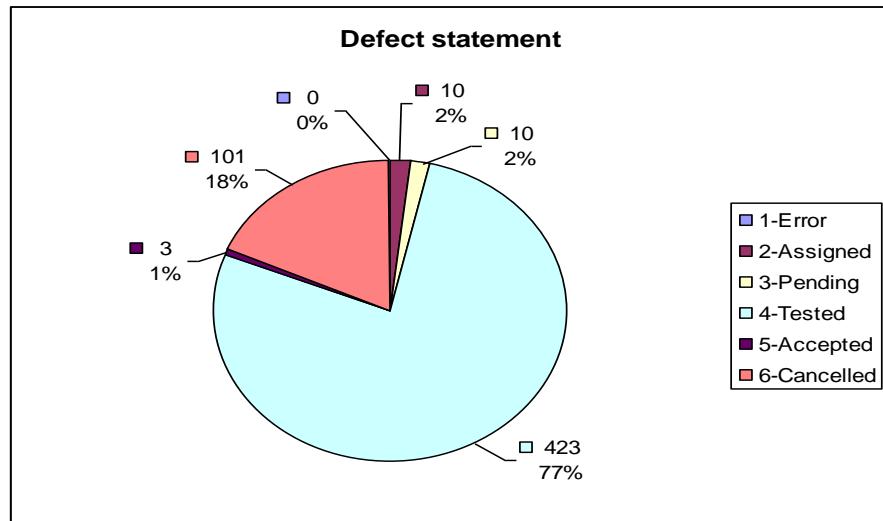
- Defect Distribution
- Defect Re-Open
- Defect Summary
- Defect Trend

Categories of the Test Report (examples)

Defect distribution report



Defect statement report



BÀI 12. THỰC HÀNH 1: KỸ THUẬT KIỂM THỬ HỘP TRẮNG

BÀI 13. THỰC HÀNH 2: KỸ THUẬT KIỂM THỬ HỘP ĐEN (I)

BÀI 14. THỰC HÀNH 3: KỸ THUẬT KIỂM THỬ HỘP ĐEN (II)

BÀI 15. THỰC HÀNH 4: THIẾT KẾ KIỂM THỬ PHẦN MỀM (TEST DESIGN)

BÀI 16. THỰC HÀNH 5: XÂY DỰNG CÁC TRƯỜNG HỢP KIỂM THỬ PHẦN MỀM (TESTCASE)

BÀI 17. THỰC HÀNH 6: QUẢN LÝ LỖI VÀ BÁO CÁO KIỂM THỬ

Mẫu tài liệu Test DesignMẫu 01: cover

Record of change					
Effective Date	Version	Changed Sheet	*A, D, M	Change description	Author
<Date when these changes are effective>					
Introduction					
Sheet Name	Description			Note	
Cover	Title page and Introduction			This page	
<u>Reference</u>	List of reference documents that the Test design is based on.				
<u>Requirement for Test</u>	List all requirement for test (both functional requirement and non-functional requirement)				
<u>List of TC</u>	List of test cases				
<u>Test Suites</u>	List of test suites				

Mẫu 02: Reference

Reference					
#	Title/File name	Author	Version	Effective Date	Note
1	<File name>		vx.x	yyyy/mm/dd	
2					

Mẫu 03: Test Requirement

Requirement view by test team

Test team can get below requirement list from project

Note team

Requirement			Requirement reference	Requirement Category	Test design status
Req. ID		Requirement Name			
R1					
	R1.1				
		1.1.1			
		1.2			
2					

	2.1					
	2.2					
	2.3					
3						
	3.1					
	3.2					
4						

Mẫu 04: List of test case

List of Test cases

TC ID	TC Name	Priority	Test type	Note
TC001				
TC002				
TC003				
TC004				
TC005				
TC006				
TC007				
TC008				
TC009				
TC010				
TC011				

TC012			
12			

Mẫu 05: Test suit

List of Test cases

Test Suite ID	Test Suite Description	TC Name	Note
TS_001		TC001 TC003 TC004	Precondition: ...

1			

Mẫu tài liệu Test Case

Mẫu 01: Cover

TEST CASE

Project Name		Author	
Project Code		Reviewer/Approvers	
Document Name		Issue Date	

Record of change:

Effective Date	Version	Change Item	*A,D,M	Change description	Reference

Mẫu 02: GUI

Graphic User Interface
[TOC](#)

Total:	0
Passed:	0
Failed:	0
Not yet tested:	0
Cancelled:	0

Screen Name	Field Name	Expected result						Test Status	Test Date	Note
		Type	Madatory	Editable	Default value	Max Length	Range/ Value			
Module Name										
	Precondition/Context:									
	[Module Name]									
[Screen Name]										
Common case										
	Note: Common cases apply for all new/edit/display pages of all modules.									
	Audit trail									

Mẫu 03: Modules

TEST CASE

Module Code					Pass Fail Untesed N/A	
Test requirement						
Tester						
Pass	Fail	Untested	N/A	Number of Test cases		
0	0	7	0	7		

ID	Test Case Description	Test Case Procedure	Expected Output	Inter-test case Dependence	Actual Output	Result	Note
[Module Name]							
Function Name							
[Function Name-1]	abc						
[Function Name-2]	def						
[Function Name-3]	dd		-	-			

TEST CASE

Module Code	<i>Subject Management</i>			
Test requirement				
Tester				
Pass	Fail	Untested	N/A	Number of Test cases
0	0	9	0	9

ID	Test Case Description	Test Case Procedure	Expected Output	Inter-test case Dependence	Actual Output	Result	Note
Subject Management							
[Subject Management-1]	Aaa						
[Subject Management-2]	Bbb		-	-			
[Subject Management-3]	Ccc		-	-			
[Subject Management-4]	Ddd						

Management-4]							
[Subject Management-5]	Eee						
[Subject Management-6]	Fff						
[Subject Management-7]	Ggg						

Mẫu 04: Picture

Chụp các form giao diện các chức năng của dự án cần test

Mẫu Bug Management

Defect ID	Module	Description	Type	Severity	Priority	Status	Created Date	Assigned to	Corrective Action
1	Title section	<p><i><Describe defect and expected result></i></p> <p>Ex: Default value of Status field = blank is incorrect</p> <p>Expected result: when create new document, default value of [Status] field must be "Oppened"</p>	<p><i><Type of defect></i></p> <p>Ex: User Interface</p>	<p><i><Severity of defect></i></p> <p>Ex: Cosmetic</p>	<p><i><Priority of defect></i></p> <p>Ex: High</p>	<p><i><Status of defect></i></p> <p>Ex: Assigned</p>	<p><i><Created date of defect></i></p> <p>Ex: 21-Sep-2008</p>	<p><i><Person who is assigned to fix defect></i></p> <p>Ex: TungNT</p>	<p><i><Actions used to fix defect></i></p> <p>Ex: change code to update correct status</p>

Mẫu 01: Test report**TEST REPORT**

Project Name	<i>Student Management</i>	Creator	<i>Ha Thi Thu Huong</i>
Project Code	<i>SM-2009</i>	Reviewer/Approver	<i>Van Kim Ngan</i>
Document Name	<i>SM-2009_Test Report_vx.x</i>	Issue Date	<i><Date when this test report is created></i>
Notes			

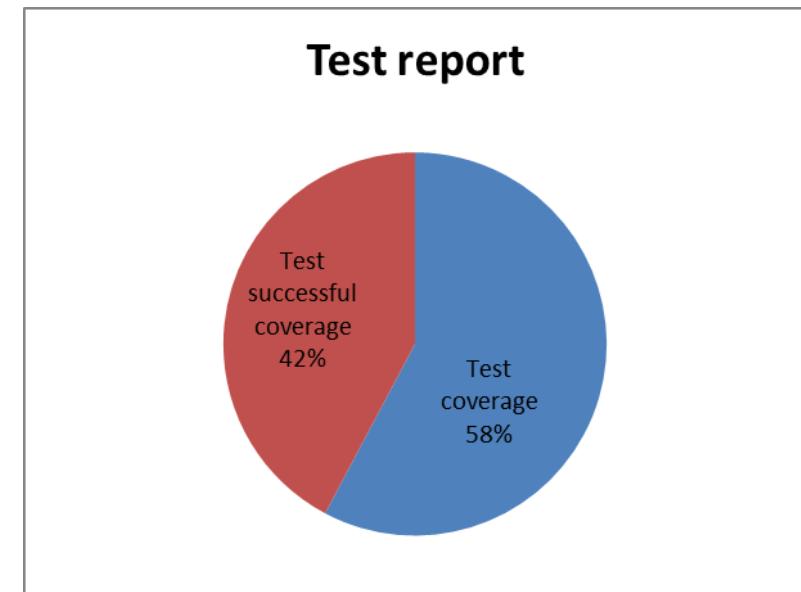
No	Module code	Pass	Fail	Untested	N/A	Number of test cases
1	Account Management	0	0	7	0	7
2	Subject Management	0	0	9	0	9
4	GUI	0	0	0	0	0
	Sub total	0	0	16	0	16

Test coverage **0.00** %

Test successful coverage **0.00** %

Ví dụ mẫu test report:**TEST REPORT**

Project Name		Creator	
Project Code		Review/Approve	
Document Code		Issue Date	
Notes			



No	Module Code	Pass	Fail	Untested	N/A	Number of Test case
1	Login	12	3	4	0	19
2	Student Management	33	3	2	0	38
3	Subject Management	12	4	5	0	21
Tổng		57	10	11	0	78

Test coverage **100 %**

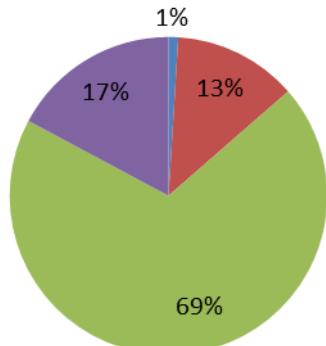
Test successful coverage **73.1 %**

Ví dụ mẫu: Defect report

Severity	Opened Defect						Fixed Defect						Total Serv	
	Error	Assigned	Fixing	Corrected	Confirmed	No	Delivered	Validated	Approved	Accepted	Cancelled	Closed	No	
Fatal	2	7				9					1	2	3	
Serious	3	136				139			1		2	12	15	
Medium	570	177	1	1		749	3	2	1	1	15	75	97	
Cosmetic		5	1			6				1	22	181	204	
Total Status	575	325	2	1		903	3	2	2	2	40	270	319	
Total W.def	1745	1286	4	3		3038	9	6	8	4	87	486	600	

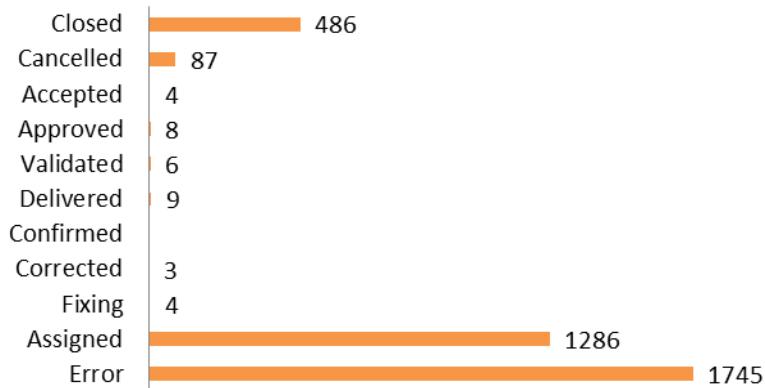
Total severity chart

■ Fatal ■ Serious ■ Medium ■ Cosmetic



Defect W.def chart

■ Series1



Ví dụ mẫu về Defect distribute:

QC Activity	Requirement					Design					Coding					Test					Other					W.def	
	F	S	M	C	W	F	S	M	C	W	F	S	M	C	W	F	S	M	C	W	F	S	M	C	W		
Acceptance test															1		1							1		3	10
After Release review									1	7	10					1		1			1	1			1	1	13
After Release test												4	4				2		2	2							26
Baseline Audit																									1	1	1
Code Review															3	9	28										28
Document Review			5	36	51		8	26	20	138					1		1			5	7	22				212	
Final inspection			6	6			1	1	4					8	1	1	35			2	2	8			1	1	54
Integration test											1	5	74	2	5	2307	4	8	11	1	114			2		6	2427
Other test																							1		10	10	
Prototype review																				3		9					9
System test																			1	1	4						4

Unit test				1	1			1	1	3	13	0	8	3	737		1	3	1	15			1	3	757		
Total				5	43	58		8	28	36	160	4	13	76	9	3112	6	9	26	3	196	1		4	3	25	3551

