

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1



BÀI GIẢNG
ĐẦM BẢO CHẤT LƯỢNG PHẦN MỀM

Biên soạn

: TS. ĐỖ THỊ BÍCH NGỌC
TS. PHAN HOÀI PHƯƠNG

HÀ NỘI, 12/2020

MỞ ĐẦU

Trước những thách thức trong quá trình phát triển phần mềm, việc đảm bảo chất lượng phần mềm (Software Quality Assurance-SQA) là hết sức quan trọng, đòi hỏi phải nghiên cứu một cách nghiêm túc để thực thi hiệu quả. Tài liệu này cung cấp những kiến thức cơ bản về chất lượng phần mềm, đảm bảo chất lượng trong một dự án phát triển phần mềm. Qui trình xây dựng hệ thống đảm bảo chất lượng phần mềm cũng được trình bày trong nội dung bài giảng. Qua đó, sinh viên hiểu được cách thức xây dựng một hệ thống đảm bảo chất lượng phần mềm và vai trò của những thành viên trong hệ thống. Một số chuẩn đảm bảo chất lượng cũng được giới thiệu trong chương cuối. Thông qua nội dung bài giảng sinh viên cũng sẽ nắm được kỹ năng rà soát và kiểm thử phần mềm.

Nội dung bài giảng được xây dựng trong bảy chương:

Chương 1. Giới thiệu đảm bảo chất lượng phần mềm

Những khái niệm mở đầu của tài liệu được giới thiệu trong Chương 1. Bắt đầu với khái niệm phần mềm, chất lượng phần mềm và đảm bảo chất lượng phần mềm, phần tiếp theo phân tích các tiêu chí chất lượng phần mềm.

Chương 2. Tích hợp các hoạt động đảm bảo chất lượng phần mềm vào vòng đời phát triển phần mềm

Chương 2 đề cập đến các thành phần đảm bảo chất lượng phần mềm trong vòng đời dự án phần mềm. Những nội dung được trình bày trong chương này bao gồm : phân tích một số mô hình phát triển phần mềm phổ biến. Sau đó, chương 2 đề cập đến các mức độ kiểm thử phần mềm.

Chương 3. Các hoạt động rà soát

Chương 3 trình bày về hoạt động rà soát cho các loại tài liệu được tạo trong quá trình phát triển phần mềm. Chương 3 trình bày các nguyên tắc và phương pháp thực hiện rà soát cũng như một số checklist rà soát mẫu.

Chương 4. Kiểm thử phần mềm

Chương 4 đề cập đến các khái niệm cơ bản trong kiểm thử phần mềm. Những nội dung được trình bày trong chương này bao gồm : khái niệm cơ bản, các mức kiểm thử, quá trình kiểm thử, cũng như ca kiểm thử.

Chương 5: Kỹ thuật kiểm thử hộp đen và hộp trắng

Chương này trình bày 2 kỹ thuật chính dùng trong thiết kế ca kiểm thử. Các kỹ thuật kiểm thử hộp đen để kiểm thử chức năng, nghiệp vụ của hệ thống. Các kỹ thuật kiểm thử hộp trắng để kiểm thử code, kiểm thử đơn vị.

Chương 6. Các công cụ hỗ trợ đảm bảo chất lượng phần mềm

Chương 6 đề cập đến các loại công cụ được dùng trong kiểm thử phần mềm. Những nội dung được trình bày trong chương này bao gồm : các phần mềm phục vụ quản lý kiểm thử, các công cụ hỗ trợ kiểm thử, và các công cụ hỗ trợ kiểm thử tự động cho cả kiểm thử chức năng và kiểm thử phi chức năng.

Chương 7. Các chuẩn, chứng chỉ và hoạt động đánh giá

Chương này đề cập tới các chuẩn quản lý chất lượng như ISO, CMM/CMMI, trong đó đi sâu vào CMM/CMMI.

Phụ lục

Gồm 3 phụ lục :

- Trình bày về các lỗi thường gặp khi viết chương trình.
- Trình bày một số hướng dẫn cho các loại kiểm thử
- Một test plan mẫu

CHƯƠNG 1: GIỚI THIỆU ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM 7

1.1	Khái niệm phần mềm	7
1.2	Các nguyên nhân gây ra lỗi phần mềm	8
1.2.1	Một số ví dụ điển hình về lỗi phần mềm	8
1.2.2	Lỗi phần mềm	13
1.2.3	Nguyên nhân gây ra lỗi phần mềm	14
1.3	Đảm bảo chất lượng phần mềm.....	17
1.3.1	Khái niệm	17
1.3.2	Mục tiêu đảm bảo chất lượng phần mềm	18
1.3.3	Xác minh, thẩm định và đánh giá chất lượng	18
1.4	Các tiêu chí chất lượng	19
1.5	Các tiêu chí chất lượng ảnh hưởng tới hoạt động đảm bảo chất lượng phần mềm như nào.	23

CHƯƠNG 2: TÍCH HỢP CÁC HOẠT ĐỘNG ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM VÀO VÒNG ĐỜI PHÁT TRIỂN PHẦN MỀM..... 25

2.1	Các phương pháp phát triển phần mềm.....	25
2.2	Các hoạt động đảm bảo chất lượng phần mềm.....	29
2.2.1	Đảm bảo chất lượng hợp đồng.....	29
2.2.2	Đảm bảo chất lượng đặc tả	30
2.2.3	Đảm bảo chất lượng phân tích, thiết kế	32
2.2.4	Đảm bảo chất lượng phát triển phần mềm (lập trình)	33
2.3	Các mức độ kiểm thử	34
2.3.1	Giới thiệu	34
2.3.2	Kiểm thử đơn vị	34
2.3.3	Kiểm thử tích hợp.....	35
2.3.4	Kiểm thử hệ thống.....	40
2.3.5	Kiểm thử chấp nhận	43

CHƯƠNG 3: CÁC HOẠT ĐỘNG RÀ SOÁT 44

3.1	Mục tiêu của rà soát	44
3.1.1	Định nghĩa.....	44
3.1.2	Mục tiêu.....	44
3.2	Các hình thức rà soát	44
3.2.1	Rà soát chính thức	44
3.2.2	Rà soát ngang hàng.....	47
3.2.3	Ý kiến chuyên gia	48
3.2.4	So sánh rà soát chính thức và rà soát ngang hàng	49
3.3	Thực hiện hoạt động rà soát trong dự án	50
3.3.1	Rà soát hợp đồng.....	50
3.3.2	Rà soát phân tích thiết kế	53
3.3.3	Các hoạt động rà soát khác.....	56
3.4	Đảm bảo chất lượng của các thành phần bảo trì phần mềm.....	63
3.4.1	Giới thiệu	63

3.4.2	Cơ sở cho chất lượng bảo trì cao.....	64
3.4.3	Các thành phần chất lượng phần mềm tiền bảo trì.....	66
3.4.4	Hỗ trợ đảm bảo chất lượng bảo trì phần mềm	70
3.5	Đảm bảo chất lượng phần mềm của các yếu tố bên ngoài cùng tham gia	78
3.5.1	Những thành phần bên ngoài đóng góp vào dự án phần mềm.....	78
3.5.2	Rủi ro và lợi ích của giới thiệu người tham dự ngoài.....	79
3.5.3	Những mục tiêu đảm bảo chất lượng về sự đóng góp người tham gia bên ngoài.....	80
3.5.4	Các công cụ đảm bảo chất lượng những đóng góp của các thành viên đóng góp bên ngoài.....	81
CHƯƠNG 4: KIỂM THỬ PHẦN MỀM		83
4.1	Định nghĩa và mục tiêu	83
4.1.1	Định nghĩa.....	83
4.1.2	Các mức độ kiểm thử	84
4.2	Quy trình kiểm thử	85
4.2.1	Quy trình.....	85
4.2.2	Input/Output cho test	87
4.2.3	Quản lý lỗi.....	88
4.3	Kế hoạch kiểm thử.....	90
4.4	Thiết kế test (test design).....	92
CHƯƠNG 5: KỸ THUẬT KIỂM THỬ HỘP ĐEN VÀ HỘP TRẮNG		95
5.1	Các kỹ thuật kiểm thử hộp đen	95
5.1.1	Phân lớp tương đương	95
5.1.2	Kiểm thử biên	99
5.1.3	Bảng quyết định.....	100
5.1.4	Lược đồ chuyển trạng thái.....	101
5.1.5	Kiểm thử theo cặp	103
5.2	Các kỹ thuật kiểm thử hộp trắng	106
5.2.1	Kiểm thử luồng điều khiển	106
5.2.2	Kiểm thử luồng dữ liệu	113
5.3	Kiểm thử đơn vị tự động	117
5.3.1	Giới thiệu chung	117
5.3.2	Tổng quan thư viện Junit	119
5.4	Bảng tóm tắt Testing Levels/ Techniques	122
CHƯƠNG 6: CÁC CÔNG CỤ HỖ TRỢ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM		123
6.1	Các công cụ quản lý thông tin trong Đảm bảo chất lượng phần mềm	123
6.1.1	Phần mềm hỗ trợ viết tài liệu	123
6.1.2	Phần mềm quản lý lỗi	123
6.2	Công cụ kiểm thử tự động là gì ?.....	125
6.2.1	Khái niệm	125
6.2.2	Quy trình kiểm thử tự động.....	125
6.3	Công cụ hỗ trợ kiểm thử đơn vị	126

6.4 Công cụ hỗ trợ kiểm thử chức năng tự động.....	126
6.4.1 Selenium WebDriver.....	129
6.4.2 Các câu lệnh sử dụng trong Selenium WebDriver	130
6.5 Công cụ hỗ trợ kiểm thử API.....	132
Giới thiệu công cụ kiểm thử API Postman	134
6.6 Công cụ hỗ trợ kiểm thử hiệu năng.....	135
6.7 Công cụ hỗ trợ kiểm thử bảo mật	135
CHƯƠNG 7: CÁC TIÊU CHUẨN TRONG QUẢN LÝ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM..	139
7.1 Giới thiệu	139
7.2 Đảm bảo chất lượng phần mềm trong các chuẩn của ISO	139
7.3 Đảm bảo chất lượng phần mềm trong các chuẩn CMM, CMMI	140
7.4 Cấu trúc và các level của CMMI :	144
7.4.1 Cấu trúc của CMMI :	144
7.4.2 Các level của CMMI:	144
7.4.3 Việt Nam áp dụng CMM/CMMI trong lĩnh vực phần mềm.	152
TÀI LIỆU THAM KHẢO.....	153
PHỤ LỤC	154
Phụ lục 1: Một số lỗi thường gặp trong phát triển phần mềm.....	154
Phụ lục 2: Một số hướng dẫn cho các loại kiểm thử.....	163
Phụ lục 3: Test plan mẫu.....	176

Chương 1: Giới thiệu đảm bảo chất lượng phần mềm

1.1 Khái niệm phần mềm

Trước khi tìm hiểu về đảm bảo về chất lượng phần mềm, mục này sẽ giới thiệu về khái niệm phần mềm.

Định nghĩa: Phần mềm bao gồm những thành phần sau đây:

- Chương trình máy tính
- Các thủ tục
- Tài liệu liên quan
- Dữ liệu cần thiết cho sự vận hành của hệ thống

Mỗi thành phần phần mềm đều có chức năng riêng và chất lượng của chúng đóng góp vào chất lượng chung của phần mềm và bảo trì phần mềm như sau:

- Chương trình máy tính được cần thiết là hiển nhiên vì chúng giúp máy tính vận hành thực thi các yêu cầu ứng dụng.
- Những thủ tục được yêu cầu để định nghĩa theo một thứ tự và lịch biểu của một chương trình khi thực thi, phương thức được triển khai và người chịu trách nhiệm cho thực thi các hoạt động cần thiết cho việc tác động vào phần mềm
- Nhiều kiểu tài liệu là cần thiết cho người phát triển, người sử dụng và người có nhiệm vụ duy trì. Tài liệu phát triển (báo cáo yêu cầu, báo cáo thiết kế, mô tả chương trình, v.v) cho phép sự phối hợp và cộng tác hiệu quả giữa các thành viên trong đội ngũ phát triển và hiệu quả trong việc xem lại và rà soát cá sản phẩm lập trình và thiết kế. Tài liệu sử dụng (thường là hướng dẫn sử dụng) cung cấp một sự miêu tả cho ứng dụng sẵn sàng và những phương pháp thích hợp cho họ sử dụng. Tài liệu bảo trì (tài liệu cho người phát triển) cung cấp cho đội bảo trì tất cả những thông tin yêu cầu về mã nguồn và công việc và cấu trúc cho từng module. Thông tin này được sử dụng để tìm nguyên nhân lỗi (bugs) hoặc thay đổi hoặc bổ sung thêm vào phần mềm có sẵn.
- Dữ liệu bao gồm các tham số đầu vào, mã nguồn và danh sách tên thích hợp với phần mềm để đặc tả những cái cần thiết cho người sử dụng thao tác với hệ thống. Một kiểu khác của dữ liệu cần thiết là chuẩn dữ liệu test, sử dụng để xác định rõ những thứ thay đổi không mong muốn trong mã nguồn

hoặc dữ liệu phần mềm đã từng xảy ra và những loại sự cố phần mềm nào có thể được lường trước.

1.2 Các nguyên nhân gây ra lỗi phần mềm

1.2.1 Một số ví dụ điển hình về lỗi phần mềm

Có rất nhiều trường hợp lỗi phần mềm đã gây thiệt hại hàng triệu đô la cũng như thiệt hại về người. Để thấy được mức độ nghiêm trọng cũng như sự đa dạng của lỗi phần mềm, mục này sẽ giới thiệu 10 lỗi nổi tiếng trong ngành phần mềm.

1. Ariane 5 Crash



Hình 1-0-1: Vụ nổ Ariane 5 do lỗi tràn số khi tính toán

Ariane 5 là chiếc thứ năm trong loạt tàu vũ trụ dân dụng Ariane của châu Âu dùng để phóng vệ tinh vào không gian. Vào ngày 4 tháng 6 năm 1996 ở Kourou, Guiana của Pháp, chiếc Ariane 5 không người lái đã phát nổ chỉ khoảng 40 giây sau khi phóng. Chiếc tên lửa trị giá 500 triệu đô la này đã phát nổ do một lỗi phần mềm phổ biến được biết đến dưới tên gọi *Integer Overflow*. Lỗi này đã xảy ra trong quá trình thực hiện chuyển đổi dữ liệu từ số floating point 64-bit sang giá trị số integer16-bit. Số floating point đã được chuyển đổi có giá trị lớn hơn số có thể được biểu diễn bởi một số integer16-bit.

2. Lỗi phần mềm tên lửa Patriot



Hình 1-0-2: Hệ thống lá chắn tên lửa phán đoán sai vị trí do lỗi làm tròn số

Ngày 25 tháng 2 năm 1991 trong Chiến tranh vùng Vịnh, hệ thống tên lửa Patriot bỗng dung đã không theo dõi và đánh chặn một tên lửa Scud đang tấn công một doanh trại của Mỹ. Theo Cơ quan Trách nhiệm Giải trình Chính phủ Hoa Kỳ, phần mềm đã bị trễ và đã không theo dõi việc phóng tên lửa trong thời gian thực, do đó nó đã để tên lửa của Iraq có cơ hội vượt qua và phát nổ trước khi có thể làm bát cú điều gì để ngăn chặn nó. Mỹ đã có 28 người thiệt mạng và 100 người bị thương sau sự cố này.

3. Lỗi Y2K

Y2K là viết tắt của Year 2000 và được gọi là “*lỗi thiên niên kỷ*”. Vào cuối những năm 90, lỗi Y2K là lỗi được đề cập nhiều nhất khi cả thế giới chờ đợi máy bay va chạm, tàu vũ trụ biến mất, thị trường chứng khoán sụp đổ như dự đoán của nhiều chuyên gia công nghệ. Lỗi này là một sai lầm đơn giản trong hệ thống quản lý thời gian của các máy tính chỉ sử dụng hai chữ số để biểu thị một năm. Theo đó, 1970 sẽ được biểu diễn là 70 và năm 1999 sẽ được biểu diễn là 99. Lý do của việc này là để tiết kiệm bộ nhớ.

Khi gần đến năm 2000, các lập trình viên máy tính nhận ra rằng máy tính sẽ không thể biểu diễn chính xác năm 2000, vì 00 được dùng để biểu diễn năm 1900. Các hoạt động được lập trình hàng ngày hoặc hàng năm sẽ bị hư hỏng hoặc thiếu sót. Vào ngày 31 tháng 12 năm 1999, chuyển sang ngày 1 tháng 1 năm 2000, máy tính sẽ hiểu là từ ngày 31 tháng 12 năm 1999, chuyển sang ngày 1 tháng 1 năm 1900.

Các ngân hàng, tính lãi suất hàng ngày, phải đổi mặt với những vấn đề thực sự. Lãi suất là số tiền mà người cho vay, ví dụ như ngân hàng, tính phí một khách hàng, chẳng hạn như một cá nhân hoặc một doanh nghiệp khi họ vay tiền. Thay vì tỷ lệ lãi suất cho một ngày, máy tính sẽ tính tỷ lệ lãi suất cho gần 100 năm !

Các trung tâm công nghệ, như các nhà máy điện, cũng bị đe dọa bởi lỗi Y2K. Nhà máy điện phụ thuộc máy tính để kiểm tra an toàn và bảo trì, chẳng hạn như áp lực nước hoặc mức độ bức xạ. Không có ngày chính xác sẽ làm mất những tính toán này và có thể đưa các cư dân gần đó đối mặt với nguy hiểm.

Giao thông vận tải cũng phụ thuộc vào thời gian và ngày tháng chính xác. Các hãng hàng không đặc biệt bị đe dọa vì máy tính lưu thông tin về tất cả các chuyến bay theo lịch trình sẽ bị đảo lộn hết cả.

Cuối cùng, có Y2K đã không gây ra hậu quả gì nghiêm trọng nhưng cũng phải mất một thời gian để các nhà phát triển phần mềm khắc phục triệt để lỗi này.

4. Khoản tiền gửi 92 nghìn triệu đô la trên PayPal

Vào một ngày trong tháng 6 năm 2013, *Chris Reynolds* đã cảm thấy giật mình hoảng hốt trước khoản tiền có trong tài khoản PayPal của mình. Số dư tài khoản của nhân viên của *PR Pennsylvania* này đã tăng lên thành *92.233.720.368.547.800 USD*.

Số tiền này đã được ghi có vào tài khoản PayPal của *Reynolds* do một lỗi phần mềm và làm anh trở thành người giàu nhất thế giới. PayPal thừa nhận rằng việc đó là do lỗi phần mềm của họ và PayPal đã đề nghị tặng một khoản tiền (không được công bố) cho *Reynolds*.

5. YouTube phải nâng cấp bộ đếm vì Gangnam Style

Năm 2014, YouTube bị lỗi bởi một video âm nhạc được gọi là *Gangnam Style* của *Psy*. Các nhà phát triển đã xây dựng YouTube trên nền tảng 32-bit, có nghĩa là YouTube có thể lưu và hiển thị số lượt xem của mỗi video bằng một con số nằm trong dải từ -2,147,483,648 đến 2,147,483,647. Tức là số lượt xem lớn nhất có thể biểu diễn được trên YouTube là khoảng 2.15 tỷ và YouTube nghĩ rằng khó có video nào có thể đạt được lượt xem kinh khủng như vậy. Tuy nhiên video *Gangnam Style* đã phá vỡ bộ đếm lượt xem của YouTube khi vượt qua con số 2.147.483.647. (có lẽ do các bố, các mẹ, các bà liên tục cho con, cho cháu xem để dụ chúng nó ăn cháo, ăn cơm nên số lượt xem mới khủng như vậy).

“Chúng tôi không bao giờ nghĩ rằng sẽ có video được xem với số lượng lớn hơn một số nguyên 32-bit” YouTube cho biết trong một bài đăng trên Google +.



Hình 1-0-3: lỗi tràn số do số lượt xem bài Gangnam style quá lớn

Google sau đó đã sửa lỗi YouTube này bằng cách sử dụng một số nguyên 64-bit để lưu số lượt xem của mỗi video.

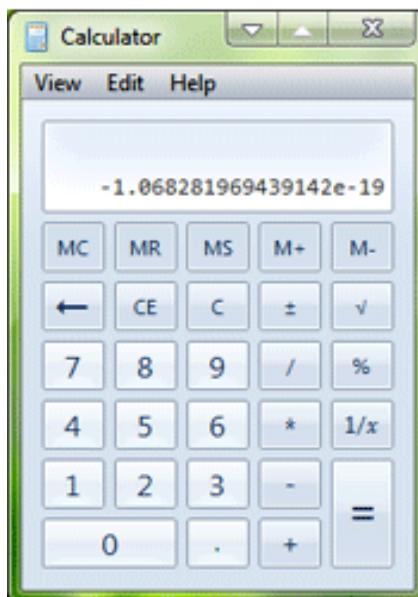
6. Lỗi tính toán của Windows Calculator

Đây là một lỗi trong Calculator của Windows, tồn tại ở các phiên bản Windows XP, Windows Vista, Windows 7 Windows 8. Lỗi xảy ra khi ta tính biểu thức: $\sqrt{4} - 2$ (lấy căn bậc hai của 4 rồi trừ đi 2). Câu trả lời đúng ra phải là 0 nhưng máy tính của Windows không cho ra kết quả 0.

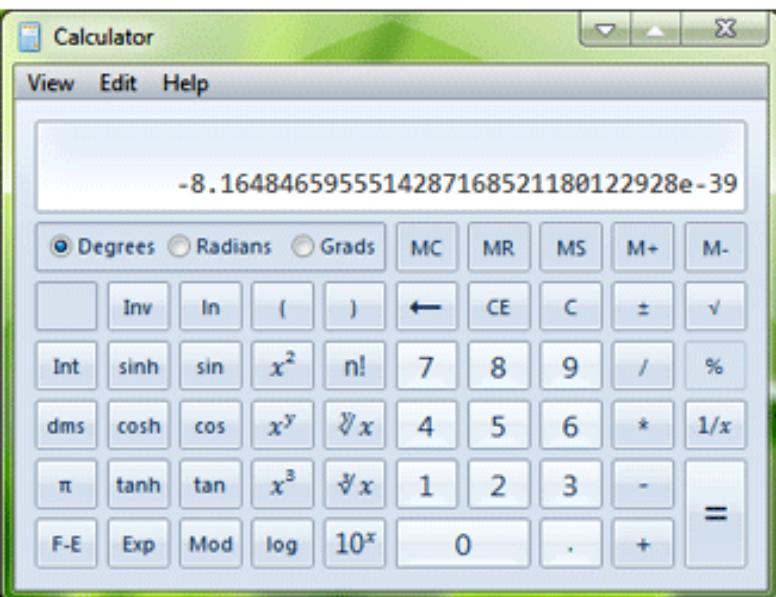
Ở chế độ *Standard*, kết quả sẽ là $-1.068281969439142e-19$ (hình bên dưới)

Ở chế độ *Scientific*, kết quả sẽ là $8.1648465955514287168521180122928e-39$ (hình bên dưới)

Standard Mode



Scientific Mode



Hình 1-0-4: Lỗi tính toán sai trên Calculator

Điều tương tự cũng xảy ra với các số khác như: $\sqrt{9} = -3$, $\sqrt{16} = -4$, ...

Tuy nhiên trên Windows 10 thì lỗi này đã được fix.

7. Year 2038

Year 2038 là một vấn đề lỗi thời gian tương tự như vấn đề của Y2K chúng ta đã đề cập ở trên. Các máy tính chạy các hệ điều hành 32-bit trên thế giới có thể dừng lại vào ngày 19 tháng 1 năm 2038. Vấn đề năm 2038 là một vấn đề về tính toán và lưu trữ dữ liệu, trong đó các giá trị thời gian được tính và lưu trữ như một số nguyên 32-bit có dấu và số này được hiểu là số giây kể từ 00:00:00 giờ UTC vào ngày 1 tháng 1 năm 1970. Điều này dẫn đến việc sẽ không thể mã hóa thời gian sau thời điểm 03:14:07 UTC ngày 19 tháng 1 năm 2038, bởi vì sau thời điểm đó số giây sẽ lớn hơn $231-1 = 2,147,483,647$ giây và không thể được biểu diễn chính xác trong 32-bit và dẫn đến lỗi *integer overflow*.

Hiện nay vẫn chưa có giải pháp chính thức được đưa ra cho vấn đề Year 2038. Hy vọng đến năm 2038 thì tất cả các máy tính không còn dùng 32-bit.

8. Lỗi “Race Condition” trên phần mềm làm mất điện ảnh hưởng đến 50 triệu người

Vào ngày 14 tháng 8 năm 2003, việc mất điện trên 8 tiểu bang Hoa Kỳ và Canada đã ảnh hưởng tới 50 triệu người. Cơ quan *PC Authority* đã công bố nguyên nhân, đó là một lỗi phần mềm được biết đến với thuật ngữ “race condition” – một lỗi xảy ra khi hai *thread* riêng biệt của cùng một chương trình sử dụng cùng tài nguyên mà không có xử lý đồng bộ đúng cách dẫn đến sụp đổ hệ thống.

Đó là những gì đã xảy ra với kết quả là 256 nhà máy điện không hoạt động, gây ra sự gián đoạn lớn về truyền thông di động.

9. Tàu vũ trụ Mars Climate Orbiter trị giá 327 triệu đô la nổ tung do lỗi phần mềm
Tàu vũ trụ Mars Climate Orbiter (dự án 327,6 triệu đô la Mỹ) đã được NASA phóng vào ngày 11 tháng 12 năm 1998 để săn tìm các hành tinh có thể hỗ trợ cuộc sống con người. Thật không may, do lỗi trong phần mềm máy tính trên mặt đất, tàu Orbiter đã nổ tung sau 286 ngày. Do tính toán sai lầm, tàu Orbiter đã đi vào vào bầu khí quyển của sao Hỏa nhưng vào không đúng chỗ dẫn đến bị nổ tung ngay sau đó.

10. Nhà mạng AT&T của Mỹ gián đoán 10 giờ đồng hồ vì lỗi phần mềm

Vào ngày 15 tháng 1 năm 1990, các khách hàng của nhà mạng AT&T không thể thực hiện được các cuộc gọi đường dài trong 9 giờ đồng hồ. Điều này làm tê liệt toàn bộ mạng điện thoại Hoa Kỳ. Tình trạng này xảy ra do một lỗi trong phần mềm kiểm soát các công tắc chuyển tiếp đường dài của AT&T. Khi đó, AT&T vừa trải qua một đợt cài đặt phần mềm mới, và phần mềm mới đã gây ra sự cố đáng tiếc. AT&T đã phải cài đặt lại phiên bản phần mềm cũ hơn nhưng đã tốn thất 60 triệu đô la cùng với sự túc giận của hàng triệu khách hàng.

1.2.2 Lỗi phần mềm

Có nhiều nguyên nhân gây ra lỗi phần mềm, biểu hiện của các lỗi cũng khác nhau ở mỗi giai đoạn phát triển phần mềm. Có ba loại lỗi phần mềm chính :

- **Error:** Là các phần của code mà không đúng một phần hoặc toàn bộ như là kết quả của lỗi ngữ pháp, logic hoặc lỗi khác được sinh ra bởi các nhà phân tích hệ thống, một lập trình viên hoặc các thành viên khác của đội phát triển phần mềm.
- **Fault:** Là các errors mà nó gây ra hoạt động không chính xác của phần mềm trong một ứng dụng cụ thể.

- **Failures:** Các faults trở thành failures chỉ khi chúng được “activated” đó là khi người dùng cố gắng áp dụng các phần mềm cụ thể đó bị faulty. Do đó, nguồn gốc của bất kì failure nào là một errors.

Ta hãy xem một ví dụ về bác sĩ chẩn đoán cho bệnh nhân. Bệnh nhân tới phòng khám với một danh sách failures (đó là các triệu chứng). Bác sĩ phải phát hiện ra các fault, hoặc nguồn gốc của các triệu chứng. Để trợ giúp quá trình chẩn đoán, bác sĩ có thể phải yêu cầu thực hiện các kiểm tra về sự bất thường của các trạng thái bên trong, như cao huyết áp, nhịp tim bất thường, lượng đường trong máu cao, cholesterol cao. Với thuật ngữ của chúng ta, các bất thường này gọi là errors.

Ta hãy xem xét một ví dụ về code Java sau:

```
public static int numZero (int[] x)
{ // Effects: if x == null throw NullPointerException
  // else return the number of occurrences of 0 in x
```

```

int count = 0;
for (int i = 1; i < x.length; i++) {
    if (x[i] == 0) {
        count++;
    }
}
return count;
}

```

fault ở đây là chương trình bắt đầu tìm kiếm các số bằng 0 từ chỉ số 1 thay vì chỉ số 0. Ví dụ numZero([2,7,0]) được tính chính xác là 1, còn với numZero([0,7,2]) sẽ được tính sai là 0. Trong cả 2 trường hợp, fault được thực thi. Mặc dù cả 2 trường hợp đều dẫn tới 1 lỗi, nhưng chỉ có trường hợp thứ 2 là gây ra failure. Để hiểu về trạng thái lỗi, ta cần định nghĩa trạng thái của chương trình. Trạng thái của numZero bao gồm giá trị của biến x, count, i và program counter (PC). Với ví dụ đầu tiên, trạng thái ở câu lệnh if cho vòng lặp đầu tiên là (x = [2, 7, 0], count = 0, i = 1, PC = if). Lưu ý là trạng thái này là lỗi vì đáng ra i = 0. Tuy nhiên, trạng thái lỗi này không được lan truyền tới output và do vậy phần mềm không fail. Nói cách khác, một trạng thái là lỗi nếu nó không phải là trạng thái như mong đợi dù cho tất cả các giá trị trong trạng thái là chấp nhận được. Tổng quát hơn, nếu chuỗi trạng thái mong đợi là s0,s1,s2... trong khi chuỗi trạng thái thực tế là s0,s2,s3... thì trạng thái s2 là lỗi.

Với trường hợp thứ 2, trạng thái tương ứng là (x = [0, 7, 2], count = 0, i = 1, PC = if). Trong trường hợp này, lỗi lan truyền tới kết quả trả về. Do vậy, ta thu được kết quả failure.

Định nghĩa về fault và failure cho phép ta phân biệt giữa testing và debugging.

1.2.3 Nguyên nhân gây ra lỗi phần mềm

Việc phát hiện ra lỗi là cần thiết, nhưng tìm ra nguyên nhân gây lỗi để tránh lỗi trong tương lai mới thực sự quan trọng. Chín nguyên nhân gây ra lỗi phần mềm thống kê sau đây đã được tổng kết sau nhiều năm nghiên cứu :

1. Định nghĩa yêu cầu lỗi
2. Lỗi giao tiếp giữa khách hàng và người phát triển
3. Sự thiếu rõ ràng của các yêu cầu phần mềm
4. Lỗi thiết kế logic
5. Lỗi coding
6. Không phù hợp với tài liệu và chỉ thị coding
7. Thiếu sót trong quá trình kiểm thử

8. Lỗi thủ tục

9. Lỗi tài liệu

Nội dung cụ thể mỗi nguyên nhân được xác định như sau:

1.2.3.1 Định nghĩa các yêu cầu bị lỗi

Việc xác định các lỗi yêu cầu, thường do khách hàng, là một trong những nguyên nhân chính của các lỗi phần mềm. Các lỗi phổ biến nhất loại này là:

- Sai sót trong định nghĩa các yêu cầu.
- Không có các yêu cầu quan trọng.
- Không hoàn chỉnh định nghĩa các yêu cầu.
- Bao gồm các yêu cầu không cần thiết, các chức năng mà không thực sự cần thiết trong tương lai gần.

1.2.3.2 Các lỗi trong giao tiếp giữa khách hàng và nhà phát triển

- Hiểu lầm trong giao tiếp giữa khách hàng và nhà phát triển là nguyên nhân bổ sung cho các lỗi ưu tiên áp dụng trong giai đoạn đầu của quá trình phát triển: Hiểu sai các chỉ dẫn của khách hàng như đã nêu trong các tài liệu yêu cầu. Hiểu sai các yêu cầu thay đổi của khách hàng được trình bày với nhà phát triển bằng văn bản trong giai đoạn phát triển.
- Hiểu sai của các yêu cầu thay đổi của khách hàng được trình bày bằng lời nói với nhà phát triển trong giai đoạn phát triển.
- Hiểu sai về phản ứng của khách hàng đối với các vấn đề thiết kế trình bày của nhà phát triển.

Thiếu quan tâm đến các đề nghị của khách hàng để cập đến yêu cầu thay đổi và khách hàng trả lời cho các câu hỏi nêu ra bởi nhà phát triển trên một phần của nhà phát triển.

1.2.3.3 Sai lệch có chủ ý từ các yêu cầu phần mềm

Trong một số trường hợp, các nhà phát triển có thể cố tình đi chệch khỏi các yêu cầu trong tài liệu, hành động thường gây ra lỗi phần mềm. Các lỗi trong những trường hợp này là sản phẩm phụ của các thay đổi. Các tình huống thường gặp nhất là: Phát triển các module phần mềm Các thành phần sử dụng lại lấy từ một dự án trước đó mà không cần phân tích đầy đủ về những thay đổi và thích nghi cần thiết để thực hiện một cách chính xác tất cả các yêu cầu mới.

Do thời gian hay áp lực ngân sách, nhà phát triển quyết định bỏ qua một phần của các yêu cầu các chức năng trong một nỗ lực để đối phó với những áp lực này. Nhà phát triển-khỏi xướng, không được chấp thuận các cải tiến cho phần mềm, mà

không có sự chấp thuận của khách hàng, thường xuyên bỏ qua các yêu cầu có vẻ nhỏ đồi với nhà phát triển. Như vậy những thay đổi "nhỏ" có thể, cuối cùng, gây ra lỗi phần mềm.

1.2.3.4 Các lỗi thiết kế logic

Lỗi phần mềm có thể đi vào hệ thống khi các chuyên gia thiết kế hệ thống-các kiến trúc sư hệ thống, kỹ sư phần mềm, các nhà phân tích, vv - Xây dựng phần mềm yêu cầu. Các lỗi điển hình bao gồm:

- + Định nghĩa các yêu cầu phần mềm bằng các thuật toán sai lầm.
- + Quy trình định nghĩa có chứa trình tự lỗi.
- + Sai sót trong các định nghĩa biên
- + Thiếu sót trong các trạng thái hệ thống phần mềm được yêu cầu
- + Thiếu sót trong định nghĩa các hoạt động trái pháp luật trong hệ thống phần mềm

1.2.3.5 Các lỗi coding

Một loạt các lý do các lập trình viên có thể gây ra các lỗi code. Những lý do này bao gồm sự hiểu lầm các tài liệu thiết kế, ngôn ngữ sai sót trong ngôn ngữ lập trình, sai sót trong việc áp dụng các CASE và các công cụ phát triển khác, sai sót trong lựa chọn dữ liệu...

1.2.3.6 Không tuân thủ theo các tài liệu hướng dẫn và mã hóa

Hầu hết các đơn vị phát triển có tài liệu hướng dẫn và tiêu chuẩn mã hóa riêng của mình để xác định nội dung, trình tự và định dạng của văn bản, và code tạo ra bởi các thành viên. Để hỗ trợ yêu cầu này, đơn vị phát triển và công khai các mẫu và hướng dẫn mã hóa. Các thành viên của nhóm phát triển, đơn vị được yêu cầu phải thực hiện theo các yêu cầu này.

1.2.3.7 Thiếu sót trong quá trình kiểm thử

Thiếu sót trong quá trình kiểm thử ảnh hưởng đến tỷ lệ lỗi bằng cách để lại một số lỗi lớn hơn không bị phát hiện hoặc không phát hiện đúng. Những kết quả yếu kém từ các nguyên nhân sau đây:

- Kế hoạch kiểm thử chưa hoàn chỉnh để lại phần không được điều chỉnh của phần mềm hoặc các chức năng ứng dụng và các trạng thái của hệ thống. Failures trong tài liệu và báo cáo phát hiện sai sót và lỗi lầm.
- Nếu không kịp thời phát hiện và sửa chữa lỗi phần mềm theo nhu hướng dẫn nhập nhằng cho lỗi này.
- Không hoàn chỉnh sửa chữa các lỗi được phát hiện do sơ suất hay giới hạn thời gian.

1.2.3.8 Các lỗi thủ tục

Các thủ tục trực tiếp cho người sử dụng đối với các hoạt động là cần thiết ở mỗi bước của quá trình. Chúng có tầm quan trọng đặc biệt trong các hệ thống phần mềm phức tạp, nơi các tiến trình được tiến hành một vài bước, mỗi bước trong số đó có thể có nhiều kiểu dữ liệu và cho phép kiểm tra các kết quả trung gian.

1.2.3.9 Các lỗi về tài liệu

Các lỗi về tài liệu là vấn đề của các đội phát triển và bảo trì đều có sai sót trong tài liệu thiết kế và trong tài liệu hướng dẫn tích hợp trong thân của phần mềm. Những lỗi này có thể là nguyên nhân gây ra lỗi bổ sung trong giai đoạn phát triển tiếp và trong thời gian bảo trì.

Cần nhấn mạnh rằng tất cả các nguyên nhân gây ra lỗi đều là con người, công việc của các nhà phân tích hệ thống, lập trình, kiểm thử phần mềm, các chuyên gia tài liệu, và thậm chí cả các khách hàng và đại diện của họ.

1.3 Đảm bảo chất lượng phần mềm

1.3.1 Khái niệm

Theo IEEE, chất lượng phần mềm được định nghĩa như sau :

Chất lượng phần mềm là:

- *Mức độ mà một hệ thống, thành phần hoặc một tiến trình đạt được yêu cầu đã đặc tả*
- *Mức độ mà một hệ thống, thành phần hoặc một tiến trình đạt được những nhu cầu hay mong đợi của khách hàng hoặc người sử dụng.*

Ban đầu đảm bảo chất lượng phần mềm có mục tiêu đạt được các yêu cầu đề ra, tuy nhiên thực tế phát triển phần mềm tồn tại rất nhiều ràng buộc đòi hỏi người phát triển cần tối ưu hóa công tác quản lý.

Khái niệm đảm bảo chất lượng phần mềm được xác định như sau :

Đảm bảo chất lượng phần mềm là một tập các hoạt động đã được lập kế hoạch và có hệ thống, cần thiết để cung cấp đầy đủ sự tin cậy vào quy trình phát triển phần mềm hay quy trình bảo trì phần mềm của sản phẩm hệ thống phần mềm phù hợp với các yêu cầu chức năng kỹ thuật cũng như với các yêu cầu quản lý mà giữ cho lịch biểu và hoạt động trong phạm vi ngân sách.

1.3.2 Mục tiêu đảm bảo chất lượng phần mềm

Phát triển phần mềm luôn đi đôi với bảo trì, vì vậy các hoạt động bảo đảm chất lượng phần mềm đều có mối liên quan chặt chẽ đến bảo trì. Những mục tiêu đảm bảo chất lượng phần mềm tương ứng với giai đoạn phát triển và bảo trì được xác định cụ thể như sau :

- Phát triển phần mềm (hướng tiến trình)

1. Đảm bảo một mức độ chấp nhận được rằng phần mềm sẽ thực hiện được các yêu cầu chức năng.
2. Đảm bảo một mức độ cấp nhận được rằng phần mềm sẽ đáp ứng được các yêu cầu về lịch biểu và ngân sách
3. Thiết lập và quản lý các hoạt động để cải thiện và nâng cao hiệu quả của phát triển phần mềm và các hoạt động SQA.

- Bảo trì phần mềm (hướng sản phẩm)

1. Đảm bảo một mức độ chấp nhận được rằng các hoạt động bảo trì phần mềm sẽ đáp ứng được các yêu cầu chức năng.
2. Đảm bảo một mức độ cấp nhận được rằng các hoạt động bảo trì phần mềm sẽ đáp ứng được các yêu cầu về lịch biểu và ngân sách
3. Thiết lập và quản lý các hoạt động để cải thiện và nâng cao hiệu quả của bảo trì phần mềm.

1.3.3 Xác minh, thẩm định và đánh giá chất lượng

Ba khía cạnh đảm bảo chất lượng của sản phẩm phần mềm gồm Verification, Validation, và Qualification.

- **Verification:** quá trình đánh giá một hệ thống hay một thành phần để xác định xem những sản phẩm của một pha phát triển xác định có thỏa mãn những điều kiện được đặt ra khi bắt đầu pha đó hay không.
- **Validation:** quá trình đánh giá một hệ thống hay một thành phần trong suốt hoặc khi kết thúc quá trình phát triển để xác định xem nó có thỏa mãn những yêu cầu đã đặc tả hay không.
- **Qualification:** quá trình xác định một hệ thống hoặc một thành phần có phù hợp với việc sử dụng hay không.

Theo những định nghĩa của IEEE, verification kiểm tra tính nhất quán của sản phẩm đang phát triển với những sản phẩm đã được phát triển ở pha trước. Khi thực hiện, người thẩm tra đi sau quy trình phát triển và giả sử rằng tất cả những pha phát triển

đãng trước đã được hoàn thành một cách chính xác hoặc là như kế hoặc gốc hoặc là sau khi đã sửa chữa những sai sót được phát hiện.

Validation miêu tả sự quan tâm của khách hàng, bằng cách thẩm tra những yêu cầu gốc của họ.

Qualification tập trung vào những khía cạnh hoạt động, ở đó việc bảo trì là vấn đề chính. Một thành phần phần mềm đã được phát triển và tài liệu hóa theo những chuẩn, kiểu, và cấu trúc chuyên nghiệp sẽ dễ dàng để bảo trì hơn những thành phần có code lạ.

Người lên kế hoạch cần phải xác định xem những khía cạnh nào nên được sát hạch trong mỗi hoạt động đảm bảo chất lượng phần mềm.

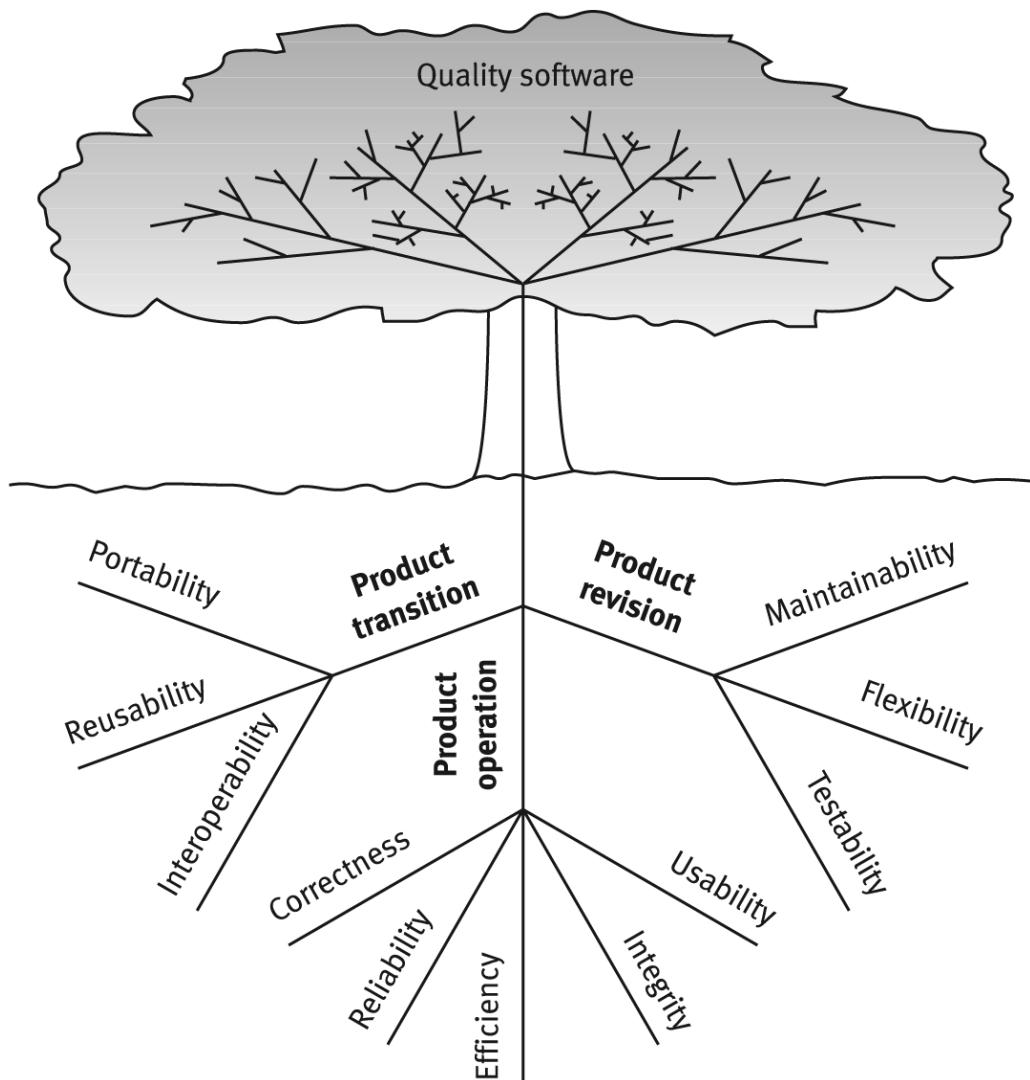
Xác minh thường là hoạt động có tính kỹ thuật cao hơn, sử dụng những tri thức về các yêu cầu, đặc tả phần mềm. Xác nhận thường phụ thuộc vào tri thức về lĩnh vực tương ứng. Cụ thể là, tri thức về ứng dụng của phần mềm được viết. Ví dụ, xác nhận của phần mềm về máy bay yêu cầu tri thức từ kỹ sư hàng không và phi công.

Cụm từ IV & V - independent verification and validation- nghĩa là xác nhận và xác minh độc lập. IV & V yêu cầu việc đánh giá phải được thực hiện bởi người không phải là lập trình viên. Một số trường hợp đội IV & V thuộc dự án, hoặc thuộc cùng công ty, cũng có thể thuộc một tổ chức độc lập. Vì sự độc lập của IV & V, tiến trình thường chưa bắt đầu cho tới khi dự án kết thúc và thường được thực hiện bởi chuyên gia trong lĩnh vực hơn là người phát triển phần mềm.

1.4 Các tiêu chí chất lượng

Đã có nhiều tác giả nghiên cứu về các yếu tố chất lượng phần mềm từ các yêu cầu cả nó. Theo thời gian có thể quan niệm về việc đảm bảo chất lượng phần mềm có phần thay đổi, tuy nhiên mô hình các yếu tố đảm bảo chất lượng phần mềm của McCall ra đời vào những năm 70 của thế kỷ trước vẫn còn được nhiều người nhắc đến như là cơ sở tham chiếu các yêu cầu phần mềm. Sau McCall cũng có một số mô hình được quan tâm như mô hình do Evans, Marcinak do hay mô hình của Deutsch và Willis, tuy nhiên những mô hình này chỉ bổ sung hay sửa đổi một vài yếu tố chất lượng. Theo McCall, các yếu tố chất lượng phần mềm được chia làm ba loại :

- Các yếu tố hoạt động của sản phẩm bao gồm tính chính xác, tin cậy, hiệu quả, tính toàn vẹn, sử dụng được
- Các yếu tố rà soát bao gồm tính bảo trì, linh hoạt, có thể test được
- Các yếu tố chuyển giao bao gồm tính khả chuyển, có khả năng sử dụng lại, có khả năng giao tác.



Hình 1-0-5: Cây mô hình tiêu chí chất lượng phần mềm theo MCCall

Chi tiết các thuộc tính được phân tích như sau :

(1) **Các yếu tố vận hành sản phẩm** : Sự chính xác, độ tin cậy, tính hiệu quả, tính toàn vẹn và khả năng sử dụng được :

• **Sự chính xác** : Các yêu cầu về độ chính xác được xác định trong một danh sách các đầu ra cần thiết của hệ thống phần mềm, như màn hình hiển thị truy vấn số dư của khách hàng trong một hệ thống thông tin kế toán bán hàng... Các đặc tả đầu ra thường là đa chiều, một số chiều thông dụng là :

- Nhiệm vụ đầu ra (ví dụ : bản in hóa đơn bán hàng, hay đèn báo động đỏ khi nhiệt độ tăng lên trên 250 độ F)
- Độ chính xác yêu cầu của các đầu ra này; chúng có thể bị ảnh hưởng bất lợi bởi các tính toán không chính xác hay các dữ liệu không chính xác.

- Tính đầy đủ của thông tin đầu ra; chúng có thể bị ảnh hưởng bất lợi bởi dữ liệu không đầy đủ.
 - Up-to-dateness của thông tin (xác định bằng thời gian giữa sự kiện và việc xem xét hệ thống phần mềm.
 - Độ sẵn sàng của thông tin (thời gian đáp ứng : được định nghĩa là thời gian cần thiết để có được các thông tin yêu cầu)
 - Các chuẩn cho việc code và viết tài liệu cho hệ thống phần mềm.
- **Độ tin cậy :** Các yêu cầu về độ tin cậy giải quyết các lỗi để cung cấp dịch vụ. Chúng xác định tỷ lệ lỗi hệ thống phần mềm tối đa cho phép, các lỗi này có thể là lỗi toàn bộ hệ thống hoặc một hay nhiều chức năng riêng biệt của nó.
 - **Tính hiệu quả :** Các yêu cầu về tính hiệu quả giải quyết vấn đề về các tài nguyên phần cứng cần thiết để thực hiện tất cả các chức năng của hệ thống phần mềm với sự phù hợp của tất cả các yêu cầu khác. Các tài nguyên phần cứng chính được xem xét ở đây là khả năng xử lý của máy tính (được đo bằng MIPS – triệu lệnh/giây; MHz – triệu chu kỳ/giây...); khả năng lưu trữ dữ liệu (dung lượng bộ nhớ, dung lượng đĩa – được đo bằng MBs, GBs, TBs...) và khả năng truyền dữ liệu (thường được đo bằng MBPS, GBPS). Các yêu cầu này có thể bao gồm cả các giá trị tối đa tài nguyên phần cứng được sử dụng trong hệ thống phần mềm. Một yêu cầu khác về tính hiệu quả đó là thời gian giữa các lần phải sạc điện đối với các hệ thống nằm trên các máy tính xách tay hay các thiết bị di động.
 - **Tính toàn vẹn:** các yêu cầu về tính toàn vẹn giải quyết các vấn đề về bảo mật hệ thống phần mềm, các yêu cầu này để ngăn chặn sự truy cập trái phép, để phân biệt giữa phần lớn nhân viên chỉ được phép xem thông tin với một nhóm hạn chế những người được phép thêm và thay đổi dữ liệu...
 - **Khả năng sử dụng:** Các yêu cầu về khả năng sử dụng được sẽ đưa ra phạm vi của tài nguyên nhân lực cần thiết để đào tạo một nhân viên mới và để vận hành hệ thống phần mềm.
 -

(2) Các yêu tố về rà soát sản phẩm :

- **Khả năng bảo trì được :** Các yêu cầu về khả năng bảo trì được sẽ xác định người dùng và nhân viên bảo trì phải nỗ lực thế nào để xác định được nguyên nhân của các lỗi phần mềm, để sửa lỗi và để xác nhận việc sửa lỗi thành công. Các yêu cầu của yếu tố này nói tới cấu trúc modul của phần mềm, tài liệu chương trình nội bộ và hướng dẫn sử dụng của lập trình viên...

- **Tính linh động :** Các yêu cầu về tính linh động cũng bao gồm cả các khả năng và nỗ lực cần thiết để hỗ trợ các hoạt động bảo trì. Chúng gồm các nguồn lực (man-day) cần thiết để thích nghi với một gói phần mềm, với các khách hàng trong cùng nghề, với các mức độ hoạt động khác nhau, với các loại sản phẩm khác nhau... Các yêu cầu về yếu tố này cũng hỗ trợ các hoạt động bảo trì trở nên hoàn hảo, như thay đổi và bổ sung vào phần mềm để tăng dịch vụ của nó và để thích nghi với các thay đổi trong môi trường thương mại và kỹ thuật của công ty.
- **Khả năng test được :** Các yêu cầu về khả năng kiểm tra được nói tới việc kiểm tra sự vận hành có tốt hay không của các hệ thống thông tin. Các yêu cầu về khả năng kiểm tra được liên quan tới các tính năng đặc biệt trong chương trình giúp người tester dễ dàng thực hiện công việc của mình hơn, ví dụ như đưa ra các kết quả trung gian. Các yêu cầu về khả năng kiểm tra được liên quan tới vận hành phần mềm bao gồm các chuẩn đoán tự động được thực hiện bởi hệ thống phần mềm trước khi bắt đầu hệ thống, để tìm hiểu xem có phải tất cả các thành phần của hệ thống phần mềm đều làm việc tốt hay không, và để có một bản báo cáo về các lỗi đã được phát hiện. Một loại khác của yêu cầu này là việc check các dự đoán tự động, được các kỹ thuật viên bảo trì sử dụng để phát hiện nguyên nhân gây lỗi phần mềm.

(3) **Các yếu tố về chuyển giao sản phẩm :** tính lưu động (khả năng thích nghi với môi trường), khả năng tái sử dụng và khả năng cộng tác được :

- **Tính lưu động :** các yêu cầu về tính lưu động nói tới khả năng thích nghi của hệ thống phần mềm với các môi trường khác, bao gồm phần cứng khác, các hệ điều hành khác... Các yêu cầu này đòi hỏi các phần mềm cơ bản có thể tiếp tục sử dụng độc lập hoặc đồng thời trong các trường hợp đa dạng.
- **Khả năng tái sử dụng :** Các yêu cầu về khả năng tái sử dụng nói tới việc sử dụng các modul phần mềm trong một dự án mới đang được phát triển mà các modul này ban đầu được thiết kế cho một dự án khác. Các yêu cầu này cũng cho phép các dự án tương lai có thể sử dụng một modul đã có hoặc một nhóm các modul hiện đang được phát triển. Tái sử dụng phần mềm sẽ tiết kiệm tài nguyên phát triển, rút ngắn thời gian phát triển và tạo ra các moduls chất lượng cao hơn. Chất lượng modul cao hơn là dựa trên giả định rằng hầu hết các lỗi phần mềm đều được phát hiện bởi các hoạt động đảm bảo chất lượng phần mềm thực hiện trên phần mềm ban đầu, bởi những người sử dụng phần mềm ban đầu và trong suốt những lần tái sử dụng trước của nó. Các vấn đề về tái sử dụng phần mềm đã trở thành một phần trong chuẩn công nghiệp phần mềm (IEEE,1999).

- **Khả năng cộng tác :** Các yêu cầu về khả năng cộng tác tập trung vào việc tạo ra các giao diện với các hệ thống phần mềm khác. Các yêu cầu về khả năng cộng tác có thể xác định tên của phần mềm với giao diện bắt buộc. Chúng cũng có thể xác định cấu trúc đầu ra được chấp nhận như một tiêu chuẩn trong một ngành công nghiệp cụ thể hoặc một lĩnh vực ứng dụng.

1.5 Các tiêu chí chất lượng ảnh hưởng tới hoạt động đảm bảo chất lượng phần mềm như nào.

Những hoạt động đảm bảo chất lượng là hướng quy trình, nói cách khác, có liên kết tới sự hoàn thành của một pha dự án, sự hoàn tất một mốc dự án, và nhiều hơn nữa. Những hoạt động đảm bảo chất lượng được tích hợp vào trong kế hoạch phát triển.

Những người lên kế hoạch đảm bảo chất lượng cho một dự án cần xác định:

- Danh sách những hoạt động đảm bảo chất lượng cần thiết cho dự án.
- Với mỗi hoạt động đảm bảo chất lượng:
 - Thời gian.
 - Loại hoạt động đảm bảo chất lượng áp dụng.
 - Người thực hiện hoạt động và tài nguyên yêu cầu.
 - Những tài nguyên yêu cầu cho việc khắc phục những nhược sai sót và những thay đổi.

Cường độ của những hoạt động đảm bảo chất lượng đã được lên kế hoạch được chỉ ra bởi số những hoạt động yêu cầu. Những yếu tố dự án và nhóm ảnh hưởng tới cường độ như sau:

Yếu tố dự án:

- Độ lớn của dự án.
- Sự phức tạp và khó của kỹ thuật.
- Phạm vi của những thành phần sử dụng lại.
- Hậu quả nếu dự án bị lỗi.

Yếu tố nhóm:

- Trình độ chuyên môn của những thành viên trong nhóm.
- Sự quen thuộc của nhóm với dự án và Kinh nghiệm của nhóm trong lĩnh vực.
- Tính sẵn sàng của những thành viên có thể trợ giúp nhóm.

- Sự hiểu biết giữa những thành viên trong nhóm, hay nói cách khác là số thành viên mới trong nhóm.

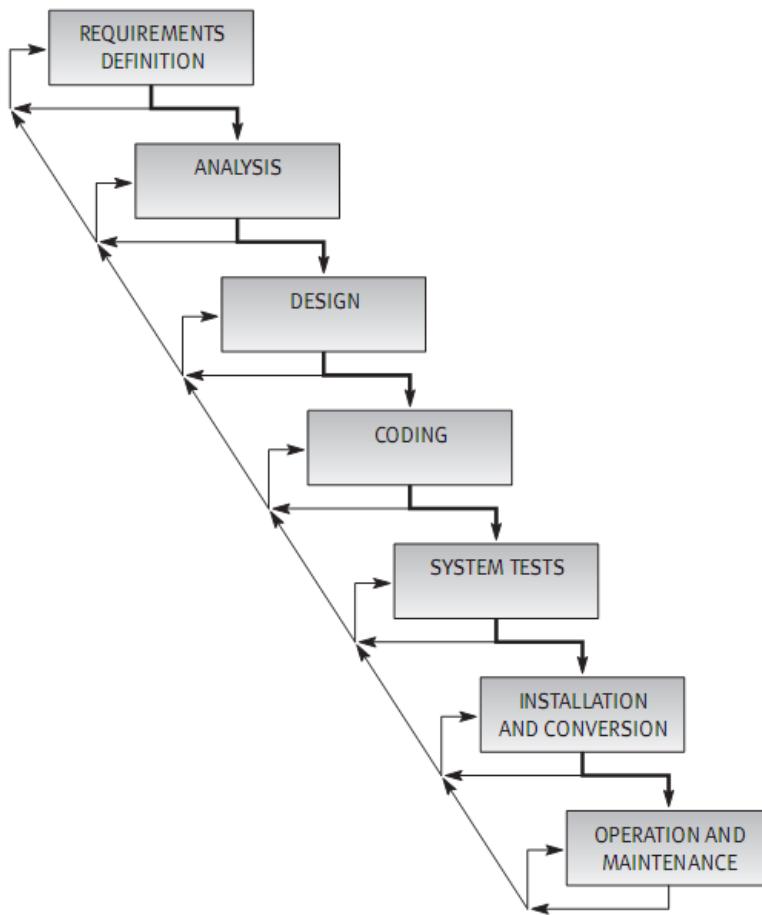
Chương 2: Tích hợp các hoạt động đảm bảo chất lượng phần mềm vào vòng đời phát triển phần mềm

2.1 Các phương pháp phát triển phần mềm

- Mô hình vòng đời phát triển phần mềm (SDLC)

Mô hình vòng đời phát triển phần mềm (SDLC) là một mô hình truyền thống (hiện nay vẫn có thể áp dụng được); nó cung cấp sự mô tả toàn diện nhất về quy trình phát triển phần mềm. Mô hình chỉ ra cần xây dựng những khối chính cho toàn bộ quá trình phát triển, được mô tả như là một chuỗi tuyến tính. Trong pha ban đầu của quy trình phát triển phần mềm, các tài liệu thiết kế sản phẩm được chuẩn bị, việc đánh giá phiên bản đầu tiên của chương trình máy tính chỉ diễn ra ở giai đoạn cuối của quy trình. Mô hình SDLC có thể đóng vai trò như một khung (framework) để biểu diễn các mô hình khác.

Thể hiện phổ biến nhất của SDLC là mô hình waterfall (thác nước). Mô hình này được minh họa trong hình dưới đây:



Hình 2-1: Mô hình vòng đời phát triển phần mềm Thác nước

Mô hình gồm có 7 pha: xác định yêu cầu, phân tích, thiết kế, coding, kiểm thử hệ thống, cài đặt và chuyển giao, vận hàng và bảo trì.

Xác định yêu cầu: Mục đích của pha xác định yêu cầu là xác định các chức năng của hệ thống cần xây dựng, khách hàng phải đưa ra và xác định các quyết định của họ. Trong nhiều trường hợp hệ thống phần mềm là một phần của hệ thống lớn hơn. Thông tin về các phần khác của hệ thống được mở rộng sẽ giúp thiết lập sự cộng tác giữa đội dự án và các giao diện thành phần phát triển.

Phân tích: Nỗ lực chính ở đây là phân tích sự ẩn ý của những yêu cầu thành mô hình khởi tạo của hệ thống phần mềm.

Thiết kế: Giai đoạn này bao gồm việc định nghĩa chi tiết đầu vào, đầu ra và các thủ tục xử lý, bao gồm cấu trúc dữ liệu, cơ sở dữ liệu, cấu trúc phần mềm, ...

Coding: Trong pha này, toàn bộ thiết kế được chuyển thành code. Việc coding bao gồm những hoạt động đảm bảo chất lượng phần mềm như inspection, unit test, và test tích hợp.

Test hệ thống (System test): Test hệ thống được thực hiện sau khi pha coding hoàn thiện. Mục đích chính của việc kiểm thử là càng tìm ra nhiều lỗi phần mềm càng tốt để đạt được mức độ chấp nhận của chất lượng phần mềm. Kiểm thử hệ thống được thực hiện bởi những người phát triển trước khi bàn giao cho khách hàng. Trong nhiều trường hợp, khách hàng thực hiện kiểm thử phần mềm một cách độc lập (còn gọi là test chấp nhận sản phẩm) để đảm bảo rằng những người phát triển đã thỏa mãn tất cả những hứa hẹn và những phản ứng phần mềm bất ngờ hoặc lỗi có thể được thấy trước. Đây là điều khá phổ biến, khách hàng yêu cầu người phát triển để họ được tham gia vào trong quá trình kiểm thử hệ thống, một thủ tục đã tiết kiệm được về mặt thời gian và tài nguyên cho việc phân tách riêng biệt giữa test hệ thống và test chấp nhận.

Cài đặt và chuyển giao: Sau khi hệ thống phần mềm được phê chuẩn, hệ thống được cài đặt để phục vụ như là một phần sụn, có nghĩa là, nó như là một phần của hệ thống thông tin, biểu diễn một thành phần cơ bản của hệ thống được mở rộng. Nếu hệ thống thông tin mới được xây dựng để thay thế hệ thống đang tồn tại, một quy trình chuyển giao phải được khởi tạo để đảm bảo rằng các hoạt động của tổ chức vẫn được tiếp tục mà không bị gián đoạn trong suốt pha chuyển giao.

Vận hành và bảo trì: Vận hành phần mềm bắt đầu sau khi pha cài đặt và chuyển giao đã được xong. Xuyên suốt thời kì vận hành, thường ít nhất là một vài năm hoặc cho đến khi xuất hiện phần mềm mới, việc bảo trì là cần thiết. Việc bảo trì kết hợp ba kiểu dịch vụ: thích ứng – sử dụng các đặc trưng của phần mềm đang tồn tại để thực hiện các yêu cầu mới; hoàn thiện – thêm một số đặc trưng để cải thiện hiệu năng của phần mềm; sửa lỗi – sửa các lỗi phần mềm được tìm thấy bởi người sử dụng trong quá trình vận hành.

Số lượng các pha có thể thay đổi tùy theo đặc trưng của từng dự án. Trong các dự án phức tạp, các mô hình phần mềm cỡ lớn, một số pha có thể bị chia ra, do đó, số lượng các pha có thể lên đến tám, chín hoặc thậm chí là nhiều pha hơn nữa. Trong các dự án nhỏ, một số pha lại có thể bị gộp lại, giảm số lượng pha của SDLC xuống còn sáu, năm, thậm chí là chỉ còn bốn pha.

Ở cuối mỗi pha, các đầu ra được xem xét và đánh giá bởi người phát triển, và trong một số trường hợp cũng có thể là khách hàng tham gia. Các kết quả có thể của quá trình xem xét lại bao gồm:

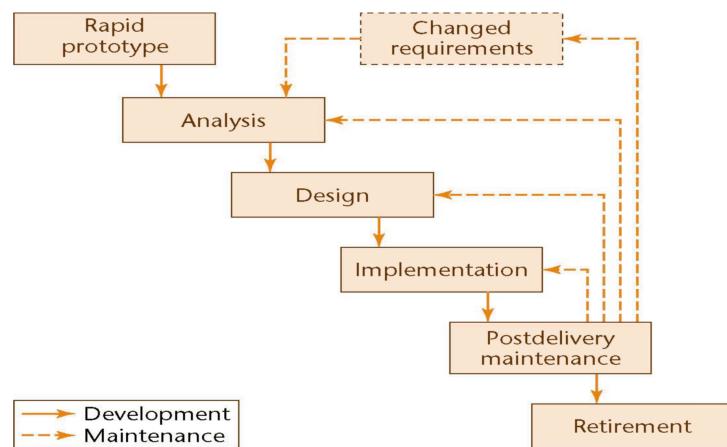
- Sự phê chuẩn của các đầu ra ở pha đó và quy trình của pha tiếp theo
- Các yêu cầu chỉnh sửa, làm lại hoặc thay đổi các phần của pha cuối cùng; trong trường hợp đảm bảo tính chắc chắn, có thể trả về các pha gần nhất theo yêu cầu.

Độ rộng của đường kết nối giữa các hình hộp chữ nhật trong hình minh họa phản ánh khả năng có những kết quả khác nhau. Do đó, hầu hết các quy trình được thực hiện như

là một dòng tuyến tính. Tuy nhiên, cần chú ý rằng, mô hình nhấn mạnh các hoạt động phát triển trực tiếp và không chỉ ra sự tham gia của khách hàng trong quy trình phát triển.

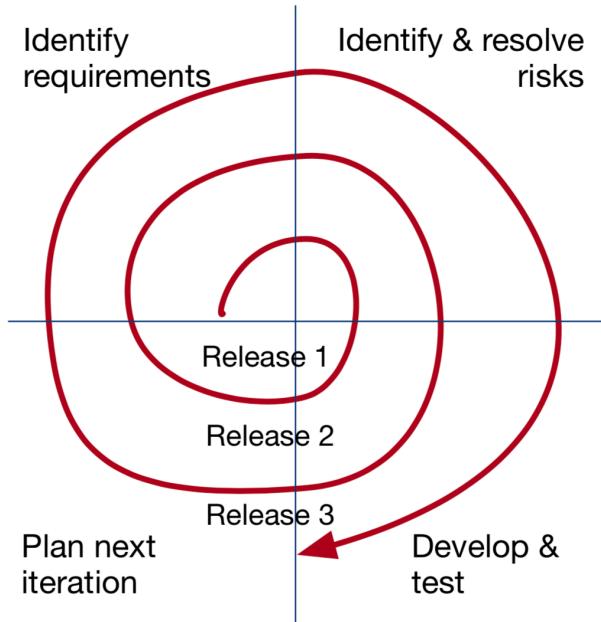
Mô hình Waterfall đã được đề xuất bởi Royce (1970) và sau đó đưa ra trong mô hình chung của nó được biết đến như là Boehm (1981). Mô hình này cung cấp các nền tảng cho phần lớn các chuẩn đảm bảo chất lượng phần mềm đã được triển khai, chẳng hạn như chuẩn IEEE 1012, IEEE 12207, ...

Mô hình bản mẫu dựa trên sự thay thế của một hoặc nhiều pha trong mô hình SDLC bằng một quy trình đánh giá, các bản mẫu phần mềm được sử dụng cho quá trình giao tiếp giữa người phát triển, người sử dụng cuối và khách hàng. Các bản mẫu được đưa ra để người sử dụng đánh giá. Sau đó, người phát triển tiếp tục phát triển một bản mẫu nâng cao, bản mẫu này cũng phải được đưa ra để đánh giá. Quá trình đánh giá này tiếp tục cho đến khi dự án phần mềm được hoàn thiện hoặc bản mẫu phần mềm đã đạt được những gì mà pha đó yêu cầu. Trong trường hợp này, phần còn lại của quy trình phát triển có thể được thực hiện theo một phương pháp luận khác, chẳng hạn như theo mô hình SDLC truyền thống.



Hình 2-2: Mô hình Phát triển phần mềm Bản mẫu nhanh

Mô hình xoắn ốc cung cấp một phương pháp luận nhằm đảm bảo hiệu quả về mặt hiệu năng của mỗi pha trong mô hình SDLC truyền thống. Mô hình này liên quan đến một quy trình lặp, tích hợp các yêu cầu thay đổi của khách hàng, phân tích và giải quyết các rủi ro, các hoạt động về mặt kỹ nghệ và kế hoạch cho phát triển phần mềm. Một trong những tương tác của mô hình xoắn ốc có thể là yêu cầu hoàn thiện ở mỗi pha của dự án trong mô hình SDLC.



Hình 2-3: Mô hình phát triển phần mềm Xoắn ốc

Mô hình hướng đối tượng kết hợp chặt chẽ việc sử dụng lại các phần mềm cũ lớn bằng cách kết hợp các mô đun có thể sử dụng lại được thành các hệ thống phần mềm mới. Trong trường hợp không có mô đun phần mềm nào có thể sử dụng được (theo thuật ngữ đối tượng hoặc thành phần) sẵn có, người phát triển có thể thực hiện một quy trình bản mẫu hoặc quy trình SDLC để hoàn thành việc phát triển một hệ thống mới.

2.2 Các hoạt động đảm bảo chất lượng phần mềm.

2.2.1 Đảm bảo chất lượng hợp đồng

Một hợp đồng tồi chắc chắn là khó có thể chấp nhận được. Từ quan điểm của SQA, một hợp đồng tồi – thường mô tả các yêu cầu không chặt chẽ và đưa ra kế hoạch cũng như ngân sách phi thực tế - thì sẽ dẫn đến một phần mềm có chất lượng tồi. Vì thế, một chương trình SQA cần được thực hiện để đảm bảo chất lượng phần mềm bằng cách rà soát lại những đề xuất ban đầu và sau đó là bản dự thảo hợp đồng (hoạt động “rà soát hợp đồng” bao gồm cả 2 hoạt động trên). Cả hai hoạt động rà soát trên là nhằm mục đích cải thiện ngân sách và thời gian biểu, là những cơ sở cho những đề nghị và hợp đồng sau này, đồng thời có thể biết được những rủi ro tiềm năng sớm (trong mục tiêu ban đầu và trong bản dự thảo hợp đồng).

Có khá nhiều tình huống có thể giúp một công ty phần mềm (“nhà cung cấp”) ký hợp đồng với một khách hàng. Phổ biến nhất là:

- Tham gia trong một cuộc đấu thầu

- Đưa ra bản phác thảo dựa trên yêu cầu đề xuất (RFP-Request For Proposal) của khách hàng
- Nhận một đặt hàng từ một khách hàng của công ty
- Nhận một yêu cầu từ bên trong hoặc từ phòng ban khác trong một tổ chức

Rà soát hợp đồng là một thành phần của SQA được nghĩ ra để hướng dẫn xem xét lại những bản dự thảo của những tài liệu đề xuất và hợp đồng. Nếu có thể, rà soát lại hợp đồng còn cung cấp sự giám sát những hợp đồng được thực hiện với những đối tác dự án tiềm năng và các nhà thầu phụ. Tiến trình rà soát có thể được chia thành hai giai đoạn:

- Giai đoạn 1: rà soát lại bản dự thảo đề xuất trước khi giao cho khách hàng tiềm năng (“rà soát bản dự thảo đề xuất”). Giai đoạn này rà soát lại bản dự thảo cuối cùng và những cơ sở đề xuất: những tài liệu yêu cầu của khách hàng, chi tiết yêu cầu thêm của khách hàng và dự diễn giải các yêu cầu, các ước lượng chi phí và tài nguyên, những hợp đồng hiện tại hoặc là những bản dự thảo hợp đồng của nhà cung cấp với các đối tác và nhà thầu phụ.
- Giai đoạn 2: rà soát lại bản dự thảo hợp đồng trước khi ký (“rà soát bản dự thảo hợp đồng”). Giai đoạn này rà soát lại bản dự thảo hợp đồng dựa trên đề xuất và sự hiểu biết (bao gồm cả những thay đổi) đã đạt được trong quá trình thương thảo hợp đồng.

Quá trình rà soát có thể bắt đầu khi những tài liệu dự thảo liên quan đã hoàn thành. Những cá nhân thực hiện rà soát phải xem xét kĩ lưỡng bản dự thảo trong khi đề cập đến một phạm vi toàn diện các đối tượng đang rà soát. Một danh sách kiểm tra là rất hữu ích cho việc đảm bảo xem xét hết các vấn đề liên quan.

Sau khi hoàn thành giai đoạn rà soát, việc cần thiết những sự thay đổi, cái thêm vào và sự hiệu chỉnh phải được thông báo bởi đội đề xuất (sau khi rà soát bản dự thảo đề xuất) và bởi ban phụ trách về luật pháp (sau khi rà soát lại bản dự thảo hợp đồng)

2.2.2 Đảm bảo chất lượng đặc tả

2.2.2.1 Chất lượng đặc tả

Đặc tả không có các hoạt động trước nó, và tất cả các hoạt động khác đều theo sau đặc tả. Vì vậy, nếu đặc tả không tốt thì phân tích, thiết kế cũng sẽ không tốt, và dẫn tới phát triển, bảo trì của 1 phần mềm kém hơn, hoặc thậm chí một phần mềm lỗi, và công sức bỏ ra để đảm bảo chất lượng thành lãng phí. Vì vậy, điều tối quan trọng là đặc tả phải đầy đủ và được định nghĩa tốt. Đặc tả thường gồm 6 nội dung sau:

- **Chức năng:** đặc tả các chức năng nào sẽ được phát triển

- **Phi chức năng:**

- (a) **Khả năng chịu tải:** đặc tả khả năng chịu tải của hệ thống (ví dụ: 100 người cùng thao tác)
- (b) **Intended use:** đặc tả yêu cầu hoặc các yêu cầu mà sản phẩm cần thoả mãn.
- (c) **Tính tin cậy:** đặc tả thời gian mà sản phẩm có thể hoạt động tốt trước khi cần bảo trì và phù hợp với yêu cầu của người dùng.
- (d) **Tính an toàn:** đặc tả ngưỡng an toàn cho con người và đặc tính khi sử dụng phần mềm
- (e) **Tính bảo mật:** đặc tả các mối đe doạ mà sản phẩm cần chuẩn bị

Làm thế nào để đảm bảo đặc tả là đầy đủ và chính xác? Với đặc tả chức năng, người phân tích nghiệp vụ, hoặc người phân tích hệ thống có thể đảm đương nhiệm vụ này. Với đặc tả phi chức năng, cần xây dựng các chuẩn nội bộ hoặc áp dụng các chuẩn chuyên nghiệp.

2.2.2.2 *Đảm bảo chất lượng đặc tả*

Trong công nghiệp phần mềm, đặc tả được xem như là đặc tả của người dùng. Nghĩa là, người dùng cuối coi đây như các yêu cầu của phần mềm mong muốn. Các tình huống sau có thể được dùng để lấy đặc tả người dùng:

- Người phân tích nghiệp vụ tìm hiểu, viết báo cáo, xây dựng đặc tả. Họ cần:
 - Gặp tất cả người dùng cuối để lấy các yêu cầu và lưu ý của họ
 - Gặp tất cả người đứng đầu các bộ phận để lấy yêu cầu và lưu ý của họ
 - Gặp người quản lý để lấy yêu cầu và lưu ý của họ
 - Tổng hợp các yêu cầu và trình bày để người dùng cuối, người đứng đầu các bộ phận và người quản lý phản hồi, nhận xét
 - Chính sửa theo phản hồi, nhận xét và xây dựng bản đặc tả chuẩn
- Có sẵn một bản yêu cầu người dùng như một phần của bản đề xuất
- Yêu cầu tham khảo một sản phẩm tương tự và cung cấp các tuỳ chỉnh riêng cho khách hàng.

Căn cứ vào các tình huống trên, khi một đặc tả đã sẵn sàng, các hoạt động đảm bảo chất lượng sẽ được thực hiện. Nhiệm vụ của bước đảm bảo chất lượng trong giai đoạn này là đảm bảo các đặc tả đã đầy đủ và phủ hết cả các yêu cầu chức năng và yêu cầu phi chức năng.

Các công cụ có thể sử dụng để đảm bảo chất lượng cho đặc tả là:

- Tài liệu quy trình: chi tiết phương pháp để lấy, phát triển, phân tích và tổng hợp đặc tả
- Các chuẩn (standard), hướng dẫn (guideline), các biểu mẫu (template): xác định tập tối thiểu các đặc tả cần phải xây dựng

- Danh sách kiểm tra (checklist): giúp phân tích để đảm bảo tính đầy đủ của đặc tả

Sử dụng các công cụ này, người phân tích có thể phát triển đặc tả đầy đủ và rõ ràng để sẵn sàng cho hoạt động tiếp theo (phân tích, thiết kế) và đảm bảo chất lượng cho đặc tả.

Các công cụ cũng có thể được dùng để đảm bảo chất lượng đặc tả là rà soát chính thức và rà soát ngang hàng. Các phương pháp rà soát này sẽ được trình bày ở Chương 3.

2.2.3 Đảm bảo chất lượng phân tích, thiết kế

2.2.3.1 Chất lượng phân tích, thiết kế

Nếu phân tích, thiết kế không tốt, sản phẩm sẽ lỗi thậm chí cả khi đặc tả được làm tốt. Phân tích gồm xác định kiến trúc, chức năng, ..., cách tiếp cận cho các yêu cầu về tính linh hoạt, tính di động, tính bảo trì được,... Thiết kế gồm thiết kế cơ sở dữ liệu, thiết kế giao diện, báo cáo... Phân tích, thiết kế gồm các thành phần sau:

1. Thiết kế chức năng
2. Kiến trúc phần mềm
3. Điều hướng (navigation)
4. Thiết kế cơ sở dữ liệu
5. Nền tảng phát triển
6. Nền tảng triển khai
7. Thiết kế giao diện người dùng
8. Thiết kế báo cáo
9. Bảo mật
10. Khả năng chịu lỗi
11. Khả năng chịu tải
12. Tính tin cậy
13. Tính bảo trì được
14. Tính hiệu quả và tính đồng thời
15. Kết nối
16. Đặc tả chương trình
17. Thiết kế kiểm thử

Ta cần người thiết kế có trình độ để đảm bảo thiết kế đúng đã được chọn và triển khai. Các chuẩn thiết kế phần mềm là cần thiết, có thể là được xây dựng nội bộ, hoặc lấy từ một tổ chức chuyên nghiệp. Các chuẩn này hỗ trợ người thiết kế để đạt được bản thiết kế tốt nhất có thể.

Thông thường, đầu pha phân tích cần lựa chọn cách thức thiết kế và quyết định các yếu tố chung như số lượng tầng, nền tảng kỹ thuật sử dụng, kết nối trong phần mềm... Bằng

cách này, người thiết kế có thể đưa ra giải pháp tốt nhất có thể cho dự án. Một bản mẫu thiết kế có thể được xây dựng và đánh giá.

Ở cuối pha phân tích, cần bước đánh giá dựa trên các chuẩn sẵn có để đảm bảo phân tích đạt yêu cầu của dự án. Bản phân tích cần được rà soát ngang hàng, rà soát từ chuyên gia, từ nhà quản lý trước khi chuyển sang thiết kế chi tiết.

2.2.3.2 Đảm bảo chất lượng phân tích, thiết kế

Pha phân tích, thiết kế nhận đầu vào là sản phẩm của pha đặc tả, từ đó xây dựng tài liệu phân tích thiết kế. Tài liệu này sẽ được lập trình viên sử dụng để lập trình ra phần mềm đặt yêu cầu.

Phân tích – còn gọi là thiết kế mức cao: đặc tả chức năng phần mềm, đặc tả yêu cầu phần mềm, kiến trúc phần mềm. Ở bước này, kiến trúc tổng thể của phần mềm, bao gồm số tầng, số module, cách tiếp cận để đạt được chức năng, thiết kế cơ sở dữ liệu, tính tin cậy, tính bảo mật, ... sẽ được xác định và tài liệu hoá. Tài liệu này được sử dụng cho pha thiết kế chi tiết.

Thiết kế - còn gọi là thiết kế chi tiết: chi tiết của từng đơn vị chương trình, màn hình, báo cáo, bảng,... được xây dựng và lập trình viên có thể sử dụng nó để lập trình cho phần mềm.

Các công cụ để đảm bảo chất lượng phân tích, thiết kế gồm:

- Tài liệu quy trình: chi tiết phương thức để thiết kế được xem xét, tiêu chí để lựa chọn các giải pháp cho dự án, cho pha phân tích
- Chuẩn, hướng dẫn, các biểu mẫu: xác định kiến trúc cùng với ưu, nhược của nó, phương thức để có danh sách các lựa chọn thay thế,...
- Danh sách kiểm tra (Checklist): giúp người thiết kế đảm bảo thiết kế là đầy đủ và chính xác

2.2.4 Đảm bảo chất lượng phát triển phần mềm (lập trình)

2.2.4.1 Chất lượng lập trình

Thông thường, pha phát triển thường có các hoạt động sau:

- Tạo cơ sở dữ liệu và các bảng (table)
- Phát triển các thư viện kết nối động cho các hoạt động chung
- Phát triển giao diện
- Phát triển báo cáo
- Phát triển kế hoạch test đơn vị
- Phát triển các thành phần với các khía cạnh khác như bảo mật, tính hiệu quả, khả năng chịu lỗi,..

Chất lượng phát triển tốt đạt được bằng cách cung cấp hướng dẫn lập trình cho ngôn ngữ sử dụng. Hướng dẫn này cần chứa cách đặt tên, cấu trúc code, cách

lập trình hiệu quả, cách ngăn ngừa lỗi và giúp lập trình viên viết code đáng tin cậy và không mắc lỗi. Tuy nhiên, cần có các lập trình viên tốt.

2.2.4.2 Đảm bảo chất lượng lập trình

Phát triển là bước xây dựng phần mềm tuân theo đặc tả. Trong pha này, mã nguồn được xây dựng và kết nối với các thư viện có sẵn, để hoàn thành các yêu cầu lập trình cho sản phẩm. Mã nguồn sẽ được dịch sang chương trình chạy được trên các phần cứng đã xác định trước. Đây cũng là bước cơ sở dữ liệu được xây dựng, từ đó dữ liệu có thể được nạp và sử dụng trong phần mềm.

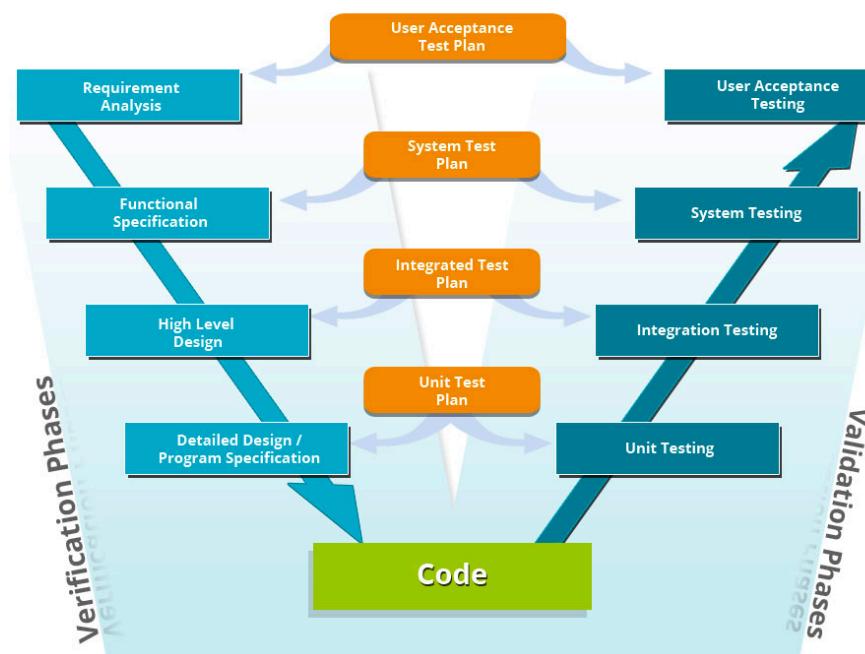
Làm thế nào để đảm bảo chất lượng pha phát triển? Ta có thể sử dụng các chuẩn nội bộ sẵn có về chất lượng code, cũng như các hướng dẫn lập trình cho ngôn ngữ lựa chọn. Ngoài ra, những thay đổi không được kiểm soát có thể dẫn tới lỗi. Vì vậy, quản lý thay đổi và quản lý cấu hình rất quan trọng trong đảm bảo chất lượng code.

Có 2 kỹ thuật có thể đảm bảo chất lượng trong quá trình xây dựng phần mềm là rà soát (walkthroughs) và lập trình. Chi tiết các kỹ thuật này sẽ được trình bày trong các chương tiếp theo.

2.3 Các mức độ kiểm thử

2.3.1 Giới thiệu

Các mức độ kiểm thử được xây dựng tương ứng với từng pha trong vòng đời phát triển phần mềm.



Hình 2-4: Các mức độ kiểm thử

2.3.2 Kiểm thử đơn vị

Kiểm thử đơn vị là hoạt động kiểm thử nhỏ nhất

Kiểm thử thực hiện trên các hàm hay thành phần riêng lẻ.

Cần hiểu biết về thiết kế chương trình và code.

Thực hiện bởi Lập trình viên (không phải kiểm thử viên)

Đơn vị: Là thành phần nhỏ nhất của phần mềm có thể kiểm thử được. Ví dụ: Các hàm, lớp, thủ tục, phương thức. Đơn vị thường có kích thước nhỏ, chức năng hoạt động đơn giản, không gây nhiều khó khăn trong việc kiểm thử, ghi nhận và phân tích kết quả do đó nếu phát hiện lỗi việc tìm kiếm nguyên nhân và sửa lỗi cũng đơn giản và tốn ít chi phí hơn.

Mục đích: Đảm bảo thông tin được xử lý đúng và có đầu ra chính xác trong mối tương quan giữa dữ liệu nhập và chức năng của đơn vị.

Người thực hiện: Do việc kiểm thử đơn vị đòi hỏi phải kiểm tra từng nhánh lệnh, nên đòi hỏi người kiểm thử có kiến thức về lập trình cũng như về thiết kế của hệ thống nên người thực hiện thường là lập trình viên.

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

2.3.3 Kiểm thử tích hợp

Kiểm thử tích hợp nhằm phát hiện lỗi giao tiếp xảy ra giữa các thành phần cũng như lỗi của bản thân từng thành phần (nếu có).

Thành phần có thể là:

- các module
- các ứng dụng riêng lẻ
- các ứng dụng client/server trên một mạng

Các loại kiểm thử tích hợp:

Big bang - Test toàn bộ phần mềm, một khi các gói đã hoàn thành có sẵn; có thể biết đến như là “big bang testing”.

Top-down – kiểm thử từ gốc của hệ thống phân cấp thành phần. Các thành phần được thêm theo thứ tự giảm dần của hệ thống phân cấp.

Bottom-up – kiểm thử từ đáy của hệ thống phân cấp. Các thành phần được thêm theo thứ tự tăng dần của hệ thống phân cấp.

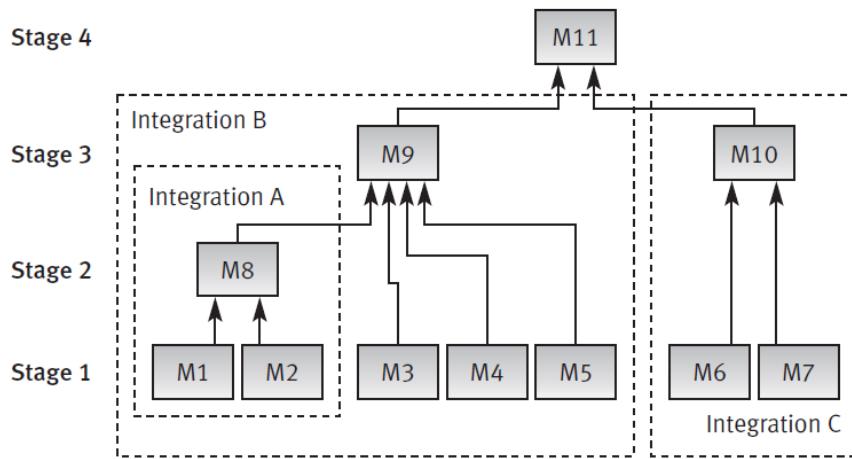
Hình sau minh họa test top-down và test bottom-up của cùng một dự án phát triển phần mềm gồm 11 mô-đun. Hình (a), quá trình phát triển phần mềm và sự thử nghiệm tiếp thep được thực hiện theo bottom-up, trong 4 giai đoạn sau:

Giai đoạn 1: test các mô-đun từ 1 đến 7

Giai đoạn 2: tích hợp test A của các mô-đun 1 và 2 ,đã phát triển và test ở giai đoạn 1, tích hợp với mô-đun 8, phát triển trong giai đoạn hiện thời.

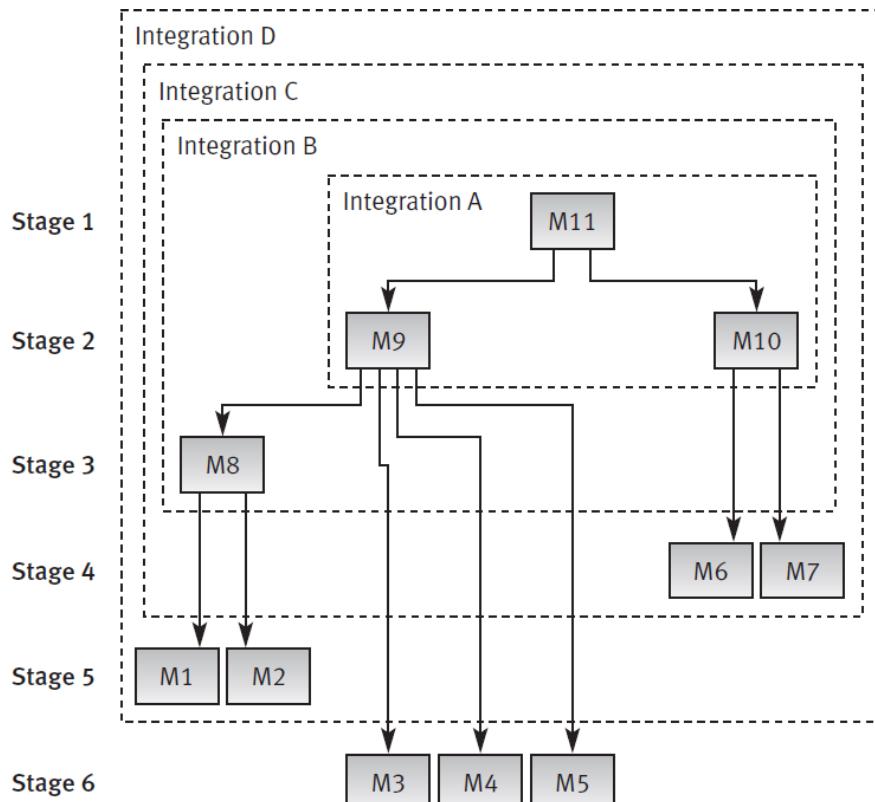
Giai đoạn 3: hai sự kiểm tra tích hợp riêng biệt, B, trên các mô-đun 3,4,5 và 8, tích hợp với mô-đun 9 , và C, mô đun 6 và 7 tích hợp với mô-đun 10

Giai đoạn 4: test hệ thống được thực hiện sau khi B và C đã tích hợp với mô-đun 11, đã phát triển trong giai đoạn hiện thời.



Hình 2-5: Ví dụ về tích hợp bottom up cho 1 mô hình hệ thống

1. Bottom-up testing



Hình 2-6: Ví dụ về tích hợp top down cho 1 mô hình hệ thống

(b) Top-down testing

Trong hình trên, phát triển phần mềm và kiểm thử được thực hiện từ trên xuống qua sáu giai đoạn :

Giai đoạn 1: test mô-đun 11(unit test)

Giai đoạn 2: tích hợp test A của mô-đun 11 tích hợp với mô-đun 9 và 10, phát triển trong giai đoạn hiện thời

Giai đoạn 3: tích hợp test B của A tích hợp với mô-đun 8, phát triển trong giai đoạn hiện thời.

Giai đoạn 4: tích hợp test C của B tích hợp với mô-đun 6 và 7 , phát triển trong giai đoạn hiện thời

Giai đoạn 5: tích hợp test D của C tích hợp với mô-đun 1 và 2, phát triển trong giai đoạn hiện thời

Giai đoạn 6: test hệ thống của D tích hợp với mô-đun 3,4,5, phát triển trong giai đoạn hiện thời

Việc gia tăng các đường trong hình trên chỉ có hai trong số nhiều các đường. Đường dẫn trong các ví dụ là “trình tự theo chiều ngang (horizontally sequenced)” (“breadth first”), mặc dù ta có thể chọn một đường dẫn là “trình tự theo chiều dọc(vertically

sequenced)" ("depth first"). Nếu ta thay đổi đường ngang của top-down trong hình (b), với một dãy dọc, kiểm thử có thể được thực hiện như sau :

Giai đoạn 1: test đơn vị mô-đun 11

Giai đoạn 2: tích hợp test A của tích hợp mô-đun 11 với mô-đun 9, phát triển trong giai đoạn hiện thời

Giai đoạn 3: tích hợp test B của A với mô-đun 8, phát triển trong giai đoạn hiện thời

Giai đoạn 4: tích hợp test C của B với các mô-đun 1 và 2, phát triển trong giai đoạn hiện thời

Giai đoạn 5: tích hợp test D của C với mô-đun 10, phát triển trong giai đoạn hiện thời

Giai đoạn 6: tích hợp test E của D với các mô-đun 6 và 7, phát triển trong giai đoạn hiện thời

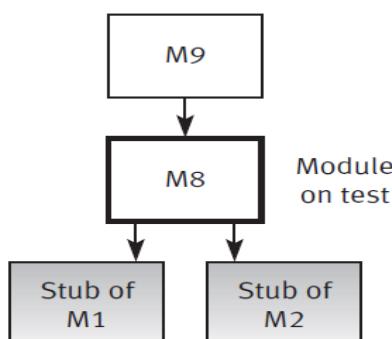
Giai đoạn 7: test hệ thống được thực hiện sau khi sau khi E đã được tích hợp với các mô-đun 3,4 và 5, phát triển trong giai đoạn hiện thời.

Các đường dẫn khác liên quan đến khả năng phân nhóm các mô-đun trong một giai đoạn kiểm thử. Ví dụ, đường dẫn trong top-down ở hình 9.1(b), một trong những nhóm mô-đun có thể là 8, 1 và 2 và/hoặc các mô-đun 10, 6 và 7.

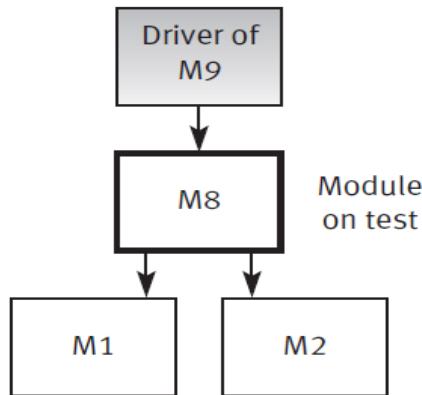
Stubs và drives để kiểm thử gia tăng

Stubs và drives là phần mềm mô phỏng thay thế cho các mô-đun không có sẵn khi thực hiện một đơn vị hoặc kiểm thử tích hợp.

Một stub (thường được cho là một "mô-đun giả (dummy module) ") thay thế một mô-đun không có sẵn ở mức thấp hơn, mô-đun phụ để kiểm thử. Các Stub được yêu cầu cho kiểm thử top-down cho các hệ thống không đầy đủ. Trong trường hợp này, stub cung cấp kết quả tính toán của mô-đun phụ, chưa được phát triển (mã hóa), được thiết kế để thực hiện. Ví dụ, ở giai đoạn 3 của ví dụ top-down trong hình (b), mô-đun 9 phía trên kích hoạt mô-đun 8, có sẵn; nó đã được kiểm thử và đúng ở giai đoạn 2. Các stub được yêu cầu để thay thế cho các mô-đun phụ 1 và 2 là những mô-đun chưa được hoàn thành. Sự thiết lập kiểm thử này được trình bày trong hình dưới đây



Giống như một Stub, nhưng một driver là một mô-đun thay thế cho các mô-đun mức trên để kích hoạt kiểm thử các mô-đun. Driver là đi từ kiểm tra dữ liệu đến kiểm thử mô-đun và chấp nhận kết quả tính toán của nó. Driver được yêu cầu kiểm thử bottom-up cho tới khi các mô-đun ở mức cao được phát triển (mã hóa). Ví dụ, ở giai đoạn kiểm thử 2 của ví dụ bottom-up trong hình (a), các mô-đun phụ 1 và 2 ở mức thấp hơn có sẵn; chúng đã được kiểm thử và đúng ở giai đoạn kiểm thử 1. Một driver được yêu cầu để thay thế cho mô-đun 9 chưa được hoàn thành. Sự thiết lập kiểm thử này được thể hiện trong hình (b).



- Các chiến lược Bottom-up so với Top-down

Lợi thế chính của chiến lược Bottom-up là thực hiện tương đối dễ, trong khi bất lợi của nó chính là sự chậm trễ trong trường hợp toàn bộ chương trình được theo dõi (nghĩa là, ở giai đoạn kiểm thử mô-đun cuối cùng). Lợi thế chính của chiến lược Top-down là khả năng nó cung cấp để chứng minh chức năng của toàn bộ chương trình một cách ngắn nhất ngay sau khi kích hoạt các mô-đun ở mức trên đã hoàn thành. Trong nhiều trường hợp, các đặc trưng này cho phép xác định các lỗi liên quan đến thuật toán, các yêu cầu chức năng trong phân tích và thiết kế một cách sớm nhất. Bất lợi chính của chiến lược này là tương đối khó khăn trong việc chuẩn bị các stub được yêu cầu, thường yêu cầu lập trình phức tạp. Bất lợi khác là khó khăn trong việc phân tích kết quả kiểm thử.

Big bang so với các chiến lược khác

Trừ khi chương trình quá nhỏ và đơn giản, ứng dụng của chiến lược kiểm thử big bang chỉ ra những bất lợi nghiêm trọng. Xác định các lỗi trở nên khá rườm rà đối với số lượng lớn phần mềm. Mặc dù có nhiều nguồn lực đã đầu tư, hiệu quả của phương pháp này là tương đối khiêm tốn. Tỷ lệ nhận dạng lỗi tương đối thấp của big bang chứng minh cho kết luận này. Hơn nữa, khi đối đầu với toàn bộ gói phần mềm, việc sửa lỗi thường là nhiệm vụ nặng nề, cần phải xem xét những ảnh hưởng có thể của vài mô-đun tại cùng một thời điểm. Những ràng buộc hiển nhiên này tạo ra một nỗ lực khá mờ nhạt của việc ước lượng các nguồn lực kiểm thử cần thiết và tiến độ kiểm thử. Điều này cũng ám chỉ

rằng triển vọng giữ đúng tiến độ và trong phạm vi ngân sách bị giảm đáng kể khi áp dụng chiến lược kiểm thử này.

2.3.4 Kiểm thử hệ thống

Kiểm thử hệ thống là một mức của tiến trình kiểm thử phần mềm khi các module và tích hợp các module đã được test.

Mục tiêu của kiểm thử hệ thống là để đánh giá phần mềm có tuân thủ theo các yêu cầu đã đưa ra không, gồm cả yêu cầu chức năng và yêu cầu phi chức năng. Vì vậy, System Test lại gồm nhiều loại kiểm thử khác nhau, phổ biến nhất gồm:

Functional testing:

Functional testing kiểm thử phần mềm để đảm bảo tất cả các chức năng hoạt động đúng. Có 2 loại kiểm thử chức năng: positive test và negative test

Positive test:

Positive test kiểm thử phần mềm xem chức năng hoạt động có đúng như đặc tả hay không, và không thử các hành động ngoại lệ. Positive test không nhằm để phát hiện lỗi phần mềm.

Negative test:

Negative test bao gồm sử dụng phần mềm theo nhiều cách khác nhau để phát hiện các lỗi ẩn không liên quan trực tiếp tới chức năng của phần mềm. Điều này đảm bảo các sai sót khi sử dụng phần mềm không gây ảnh hưởng tới phần mềm hoặc ảnh hưởng tới toàn bộ dữ liệu. Mỗi thành phần trên màn hình như text box, combo box, list box... có một lượng lớn sự kiện đi kèm. Ví dụ, click, double click, thay đổi, mouse up, mouse down, got focus, lost focus, key press, key down, key up,... liên quan tới 1 combo box.

End-to-End testing

Trong End-to-end testing, một thực thể trong phần mềm được theo dõi từ lúc tạo ra tới khi mất đi. Trong nhiều phần mềm, có thể mất nhiều năm để các chức năng thực thi trên 1 thực thể. Ví dụ trong phần mềm bảo hiểm nhân thọ, khi một chính sách được ban hành, nó có thể tồn tại 1 số năm. Một số người có thể đóng bảo hiểm và lấy lại tiền lãi trong đúng khoảng thời gian chính sách hiệu lực. Một số người khác có thể dừng việc nộp tiền trước khoảng thời gian chính sách hiệu lực, sau khoảng thời gian... Để đảm bảo các chức năng hoạt động đúng, loại kiểm thử này cần được xây dựng cho một thực thể. Ví dụ, trong phần mềm thanh toán lương, một nhân viên có thể được tăng chức, giảm chức, hay chuyển việc; lương tăng, giảm; nhân viên nghỉ hưu, chết, nghỉ việc,

hoặc dừng việc. End-to-end testing có thể có nhiều đường thực thi từ lúc thực thi khởi tạo cho tới lúc mất đi. Ta có thể dùng lược đồ chuyển trạng cho loại kiểm thử này.

Conccurent/Parallel testing:

Trong parallel testing, một lượng lớn người dùng truy cập cùng chức năng và cùng nhập/yêu cầu cùng dữ liệu. Parallel test kiểm tra khả năng phần mềm quản lý các yêu cầu xảy ra cùng lúc và đảm bảo tính toàn vẹn dữ liệu.

Có 2 phương pháp được sử dụng để thực hiện parallel testing: kiểm thử thủ công và sử dụng công cụ. Với kiểm thử thủ công, một số máy trạm sẽ chạy cùng bộ test cases. Người dùng sẽ chạy các test case và log kết quả lại. Với kiểm thử tự động, công cụ kiểm thử được lập trình với các test case cần thiết và công cụ mô phỏng một số người dùng bằng cách chạy các test case và log kết quả.

Parallel testing đảm bảo tính sẵn sàng/không sẵn sàng của phần mềm khi thực hiện một số yêu cầu cho cùng dịch vụ/chức năng.

Ví dụ, một hệ thống đặt vé. Giả sử chỉ còn 1 chỗ trên chuyến bay, và có 2 người cùng đang tìm kiếm. Khi cả 2 người cùng yêu cầu đặt chỗ ghế này, hệ thống chỉ được chấp nhận 1 người, và từ chối người còn lại. Hệ thống không được thu tiền của cả 2 người mà chỉ cung cấp chung 1 ghế. Kiểm thử loại này gọi là concurrent testing.

Một tình huống khác khi tập hợp các báo cáo nhất là báo cáo lấy thông tin từ 3 bảng trở lên. Khi có 2 hành động được thực hiện đồng thời (1) cập nhật dữ liệu và (2), việc xuất báo cáo có thể bị nhầm lẫn thông tin với dữ liệu đang được cập nhật. Để xử lý tình huống này, chúng ta cần các phương pháp điều khiển đồng thời, trong đó việc xuất báo cáo sẽ chỉ được thực thi sau khi hành động trên dữ liệu tương ứng được hoàn thành.

Performance Test (Kiểm thử hiệu năng):

Performance test đánh giá hiệu năng tổng thể của phần mềm, bao gồm thời gian phản hồi, thời gian xuất báo cáo... Việc kiểm thử này nhằm 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... Có một số loại kiểm thử hiệu năng:

Load test:

Trong phần mềm web nhiều người dùng, một số lượng lớn người dùng được phép đăng nhập và sử dụng phần mềm cùng lúc. Mục đích của load testing là để đánh giá phần mềm quản lý nhiều yêu cầu cùng lúc như nào và có thể đưa ra kết quả chính xác hay bị trộn lẫn nhau. Lỗi load testing thường liên quan tới bandwidth, database, RAM, không gian hard disk,...

Hai phương pháp để thực hiện load test: kiểm thử thủ công và áp dụng công cụ. Với kiểm thử thủ công, một lượng lớn máy trạm được cài đặt, mỗi người dùng sử dụng 1

máy. Người dùng được yêu cầu thực thi test cases và log kết quả ở máy trạm. Với kiểm thử tự động, công cụ kiểm thử sẽ được lập trình với các ca kiểm thử cần thiết và mô phỏng số lượng lớn người dùng lớn khi chạy các ca kiểm thử và log kết quả. Load test có thể phát hiện không chỉ lỗi phần mềm mà còn cả lỗi phần cứng và giới hạn khi hỗ trợ một lượng lớn người dùng.

Volume test:

Volume test thực hiện kiểm thử phần mềm khi kích thước dữ liệu phần mềm lớn để kiểm tra hiệu năng của phần mềm. Thông thường, khi phát triển phần mềm và test chức năng, ta chỉ dùng một lượng dữ liệu nhỏ. Tuy nhiên, khi sử dụng phần mềm trong thực tế, dữ liệu sẽ được thêm dần và trở lên lớn hơn rất nhiều. Hiệu năng của phần mềm thường bị giảm trong tình huống này, đặc biệt là với các chức năng xuất báo cáo hay khi các file dữ liệu và bảng chính được bảo trì. Vì vậy, volume test cần được xây dựng với bộ dữ liệu lớn.

Trong volume test, các chức năng có thể sử dụng lượng lớn dữ liệu sẽ được kiểm thử. Test cases được xây dựng từ tài liệu thiết kế. Dữ liệu với dung lượng lớn được sinh bởi một công cụ sinh dữ liệu test. Một công cụ sinh dữ liệu test sinh một lượng lớn dữ liệu test bằng cách sử dụng các giá trị quan trọng, và để các giá trị còn lại là ảo ở mọi bản ghi. Mục tiêu của dữ liệu test không phải để đảm bảo kết quả chính xác, mà để kiểm tra hiệu năng khi dữ liệu lớn. Sử dụng dữ liệu này, volume test sẽ được thực thi và kết quả sẽ được log lại để xem có đáp ứng yêu cầu không.

Stress testing:

Thực thi hệ thống với giả thiết là các tài nguyên hệ thống yêu cầu không đáp ứng được về chất lượng, ổn định và số lượng, ví dụ: tài nguyên không sẵn sàng, không giải phóng tài nguyên, kịch bản có thể dẫn tới deadlock, ngắt kết nối mạng... Stress test kiểm tra mức độ phản hồi của phần mềm với các sự kiện bất thường.

Trong ứng dụng web, loại kiểm thử này rất quan trọng, vì người dùng sử dụng hệ thống từ xa và không hiểu chuyện gì xảy ra, liệu hệ thống có phản hồi không, giao dịch có được thực hiện không?

Stress test thường được thực hiện thủ công bằng cách ngắt kết nối tài nguyên khi đang thực hiện test cases.

Kiểm thử cấu hình (Configuration Test): phân tích hệ thống với các thiết lập cấu hình khác nhau.

Kiểm thử bảo mật (Security Test): Bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.

Kiểm thử khả năng phục hồi (Recovery Test): Bảo đảm hệ thống có khả năng khôi

phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến...

Kiểm thử quá tải (overload testing): đánh giá hệ thống khi nó vượt qua giới hạn cho phép.

Kiểm thử chất lượng (quality testing): đánh giá sự tin tưởng, tính sẵn sàng của hệ thống. Bao gồm cả việc tính toán thời gian trung bình hệ thống sẽ bị hỏng và thời gian trung bình để khắc phục.

Kiểm thử cài đặt (Installation testing): Người dùng sử dụng các chức năng của hệ thống và ghi lại các lỗi tại vị trí sử dụng thật sự.

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

2.3.5 Kiểm thử chấp nhận

Kiểm thử chấp nhận là một cấp độ trong tiến trình kiểm thử phần mềm nhằm kiểm thử hệ thống về khả năng chấp nhận được.

Mục tiêu của kiểm thử này là để đánh giá sự tuân thủ của hệ thống với các yêu cầu nghiệp vụ và thẩm định xem đã có thể chấp nhận để bàn giao chưa.

Kiểm thử chấp nhận nhằm mục đích để chứng minh phần mềm thỏa mãn tất cả yêu cầu của khách hàng và khách hàng chấp nhận sản phẩm. Kiểm thử chấp nhận được khách hàng thực hiện (hoặc ủy quyền cho một nhóm thứ ba thực hiện).

Kiểm thử chấp nhận thông thường sẽ thông qua hai loại kiểm thử gọi là kiểm thử Alpha – **Alpha Test** và kiểm thử Beta – **Beta Test**. Với Alpha Test, người dùng kiểm thử phần mềm ngay tại nơi phát triển phần mềm, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa. Với Beta Test, phần mềm sẽ được gửi tới cho người dùng để kiểm thử ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

Thực tế cho thấy, nếu khách hàng không quan tâm và không tham gia vào quá trình phát triển phần mềm thì kết quả Acceptance Test sẽ sai lệch rất lớn, mặc dù phần mềm đã trải qua tất cả các kiểm thử trước đó. Sự sai lệch này liên quan đến việc hiểu sai yêu cầu cũng như sự mong chờ của khách hàng.

Chương 3: Các hoạt động rà soát

3.1 Mục tiêu của rà soát

3.1.1 Định nghĩa

Định nghĩa của IEEE (1990), *quá trình rà soát* là:

“Một quá trình hoặc một cuộc họp mà trong đó một sản phẩm công việc hoặc một tập các sản phẩm công việc được đưa ra tới toàn thể cá nhân tham gia vào dự án, các giám đốc, người dùng, khách hàng và các bên quan tâm đến dự án nhằm lấy ý kiến phê bình và phê chuẩn”

Một số phương pháp dùng để xem xét lại tài liệu:

- Xem xét lại thiết kế hình thức (Formal design reviews)
- Xem xét lại ngang hàng (Peer reviews)
- Ý kiến chuyên gia

3.1.2 Mục tiêu

Mục đích được chia làm 2 loại: mục đích trực tiếp và gián tiếp.

Mục đích trực tiếp:

- Phát hiện lỗi phân tích và thiết kế.
- Xác định các rủi ro mới.
- Xác định sự sai lệch so với mẫu, các kiểu thủ tục và qui ước.
- Đề phê chuẩn sản phẩm của phân tích hoặc thiết kế.

Mục đích gián tiếp:

- Nơi họp mặt không chính thức để trao đổi về những kiến thức chuyên môn.
- Ghi lại những lỗi phân tích và thiết kế sẽ hỗ trợ một cơ sở cho những hoạt động sửa chữa lỗi trong tương lai.

3.2 Các hình thức rà soát

3.2.1 Rà soát chính thức

Rà soát thiết kế chính thức(DRs-formal Design Reviews) là rà soát duy nhất cần thiết cho việc phê duyệt sản phẩm thiết kế

Rà soát thiết kế hình thức có thể được thực hiện tại bất cứ mốc phát triển nào yêu cầu sự hoàn thiện của tài liệu phân tích hay thiết kế.

Ví dụ các review trong quá trình phát triển phần mềm :

- DPR – Development Plan Review : Review kế hoạch phát triển
- SRSR – Software Requirement Specification Review : Review đặc tả yêu cầu phần mềm
- PDR – Preliminary Design Review : Review thiết kế sơ bộ
- DBDR- Detailed Design Review : Review thiết kế chi tiết
- TPR – Test Plan Review : Review kế hoạch kiểm thử
- STPR – Software Test Procedure Review : Review thủ tục kiểm thử phần mềm
- VDR- Version Description Review : Review mô tả phiên bản
- OMR- Operator Manual Review : Review vận hành thủ công
- SMR- Support Manual Review :Review trợ giúp thủ công
- TRR- Test Readiness Review : Review sự sẵn sàng kiểm thử
- PRR- Product Release Review : Review bản phát hành sản phẩm
- IPR-Installation Plan Review : Review kế hoạch cài đặt

Các nhân tố ảnh hưởng tới DRs

- Những người tham gia
- Sự chuẩn bị trước
- Phiên DR
- Các hoạt động sau DR được đề xuất

Những người tham gia rà soát thiết kế

Gồm có Review leader và review team.

Review Leader :

- Có kiến thức và kinh nghiệm trong việc phát triển kiểu dự án được review
- Có thâm niên ở mức độ bằng với hoặc cao hơn của project leader
- Có mối quan hệ tốt với project leader và đội dự án
- Có vị trí bên ngoài đội dự án

Review team:

- Phần lớn không thuộc đội dự án
- Kích thước từ 3-5 người
- Đa dạng về kinh nghiệm và phương pháp

Sự chuẩn bị cho một phiên làm việc DR

Được hoàn thành bởi 3 thành viên: review leader, review team và development team..

Chuẩn bị của Review leader

- Bổ nhiệm các thành viên nhóm
- Lập lịch các phiên review
- Phân chia tài liệu thiết kế cho các thành viên của nhóm

Chuẩn bị của Review team

- Xem lại tài liệu thiết kế
- Danh sách bình luận

Chuẩn bị của Development team

- Trình diễn ngắn tài liệu thiết kế
- Checklist các công việc review

Phiên DR

Một cuộc họp phiên DR thông thường gồm có :

1. Trình diễn ngắn gọn về tài liệu thiết kế
2. Các bình luận của các thành viên review team
3. Kiểm tra và xác nhận thảo luận mỗi bình luận
4. Các quyết định về tài liệu thiết kế để xác định tiến trình dự án. Các quyết định có thể có 3 loại :
 - Phê duyệt đầy đủ
 - Phê duyệt từng phần
 - Từ chối phê duyệt

Các hoạt động hậu review

- Báo cáo Review
- Review leader thực hiện sau phiên review
- Bao gồm :
 - Tổng kết các thảo luận review
 - Quyết định về sự tiếp tục của dự án
 - Danh sách các hoạt động cần thiết phải làm
 - Tên thành viên chịu trách nhiệm theo sát việc hiệu chỉnh
- Tiến trình theo dõi
 - Review leader thực hiện
 - Chắc rằng việc hiệu chỉnh được thực hiện đúng đắn

Việc theo dõi cần được ghi lại

3.2.2 Rà soát ngang hàng

Mục đích chính của peer review là xác định lỗi và độ lệch dựa vào các chuẩn.

Có hai phương pháp peer reviews:

- xét duyệt (inspection)
- kiểm tra từng bước (walkthrough).

Walkthrough phát hiện sai sót và ghi chú lên tài liệu.

Inspection phát hiện sai sót và kết hợp với nỗ lực để cải tiến.

Những người tham gia vào peer reviews

Một đội peer review tối ưu 3-5 người tham gia.

Tất cả những người tham gia nên là những người cùng địa vị của nhà thiết kế hệ thống phần mềm.

Một đội peer review để cử bao gồm:

- Một leader review.
- Một người thực thi (author).
- Các chuyên gia đặc biệt (specialized professionals).

Phân công trách nhiệm trong đội (team assignments)

Hai trong số các thành viên sẽ là:

- một người dẫn chương trình
- một người viết tài liệu trong cuộc thảo luận.

Chuẩn bị cho phiên peer review

- **Leader:**
 - Xác định những đoạn trong tài liệu thiết kế sẽ được review
 - Lựa chọn thành viên nhóm
 - Lập lịch cho những phiên review
 - Đưa tài liệu cho các thành viên trong đội trước phiên review
- **Đội peer review:** yêu cầu của inspection khá tinh vi, còn walkthrough chỉ yêu cầu đơn giản.
 - Inspection: Đọc & liệt kê chú thích của họ
 - Walkthrough: Đọc các đoạn sẽ được review

Phiên peer review

Inspection :

- Presenter đọc một đoạn tài liệu và thêm vào nếu cần thiết.

- Những người liên quan hoặc đưa ra chú thích, hoặc phản ứng với những lời chú thích trong tài liệu.

Walkthrough:

- Bắt đầu bằng sự trình bày ngắn của author (thường ko phải là presenter) hoặc tổng quan về dự án và những đoạn thiết kế sẽ được review.
- Ghi lại vị trí, mô tả, kiểu, đặc điểm (sai sót, những phần thiếu, những phần thêm vào) của mỗi lỗi được chấp nhận.

Quy tắc thời gian: phiên không nên vượt quá 2 giờ, hoặc lập lịch không nhiều hơn 2 ngày.

Tài liệu sau mỗi phiên review:

- Báo cáo những phát hiện trong phiên inspection
- Báo cáo tóm tắt của phiên inspection

Các hoạt động sau peer review (post-peer review activities)

Inspection:

- Nhắc nhở, sửa chữa hiệu quả, làm lại tất cả các lỗi
- Chuyển giao các bản báo cáo inspection tới CAB để phân tích.

Hiệu quả của peer review (the efficacy of peer reviews)

Một vài độ đo phổ biến ước lượng hiệu suất của peer review:

- Số giờ trung bình trên một lỗi.
- Mật độ phát hiện thiếu sót (Số thiếu sót trung bình trên một trang tài liệu thiết kế).
- Hiệu năng peer review bên trong.
- Peer review coverage: Là tỉ lệ nhỏ của tài liệu và toàn bộ code đã từng trải qua peer review

3.2.3 Ý kiến chuyên gia

Ý kiến của chuyên ra rất hữu ích trong những trường hợp sau:

- Thiếu sự hiểu biết đầy đủ về lĩnh vực nào đó.
- Tạm thời thiếu những người chuyên nghiệp để tham gia vào đội xem xét lại
- Các thành viên chuyên nghiệp cao cấp trong tổ chức không thống nhất được với nhau.
- Trong các tổ chức nhỏ số lượng ứng viên phù hợp cho đội xem xét lại là không đủ.

3.2.4 So sánh rà soát chính thức và rà soát ngang hàng

Để rõ hơn những đặc điểm của các phương pháp rà soát có thể tham khảo bảng so sánh giữa ba phương pháp rà soát sau đây:

Thuộc tính	Formal design reviews	Inspections	Walkthroughs
Mục đích chính trực tiếp	Phát hiện lỗi Xác định rủi ro mới Phê chuẩn tài liệu thiết kế	Phát hiện lỗi Xác định sự sai lệch so với tiêu chuẩn	Phát hiện lỗi
Mục đích chính gián tiếp	Trao đổi kiến thức	Trao đổi kiến thức Hỗ trợ hoạt động sửa chữa	Trao đổi kiến thức
Lãnh đạo việc xem xét lại	Người đứng đầu kỹ nghệ phần mềm hoặc là thành viên cao cấp	Người điều hành chỉ đạo (ngang hàng)	Người tổ chức
Người tham gia	Các thành viên cấp cao và đại diện khách hàng	Những người ngang hàng	Những người ngang hàng
Sự tham gia của trưởng dự án	Có	Có	Có, thông thường là khi bắt đầu xem xét lại
Thành viên chuyên gia trong đội		Người thiết kế Lập trình viên - Người kiểm thử	Người giám sát tiêu chuẩn - Chuyên gia bảo trì - Đại diện người dùng
Quá trình xem xét lại			
Hợp tổng quan	Không	Có	Có

Sự chuẩn bị của những người tham gia	Có – chuẩn bị kỹ lưỡng	Có – chuẩn bị kỹ lưỡng	Có – chuẩn bị tóm lược
Phiên xem xét lại	Có	Có	Có
Bám sát việc sửa chữa	Có	Có	Có
Cơ sở hạ tầng			
Đào tạo chính thức cho người tham gia	Không	Có	Không
Sử dụng checklist	Không	Có	Không
Lỗi liên quan đến thu thập dữ liệu	Không yêu cầu hình thức	Yêu cầu hình thức	Không yêu cầu hình thức
Tài liệu hóa việc xem xét lại	Báo cáo xem xét lại thiết kế hình thức	Báo cáo đánh giá phiên thanh tra Báo cáo tổng kết phiên thanh tra	Báo cáo đánh giá phiên walkthrough

3.3 Thực hiện hoạt động rà soát trong dự án

3.3.1 Rà soát hợp đồng

- Những mục đích của công việc rà soát bản dự thảo đề xuất

Mục đích của việc rà soát bản dự thảo đề xuất là để đảm bảo rằng những hoạt động sau được thực hiện một cách thỏa đáng.

1) Những yêu cầu của khách hàng đã được giải thích chi tiết và có chú giải

Những tài liệu yêu cầu đề xuất (RFP) và những tài liệu công nghệ tương tự có thể quá chung chung và mơ hồ cho những mục tiêu của dự án. Kết quả là có nhiều chi tiết cần được thêm vào từ khách hàng. Việc giải thích chi tiết những yêu cầu mập mờ và những cập nhật của chúng nên được ghi lại trong một tài liệu riêng biệt đã được sự chấp nhận của cả khách hàng và công ty phần mềm.

2) Lựa chọn những phương pháp thực hiện dự án đã được kiểm tra.

Thông thường, những lựa chọn có triển vọng và phù hợp mà trên đó thể hiện một đề xuất thì đã được xem xét đầy đủ (nếu tất cả) bởi đội đề xuất. Điều kiện này đặc biệt

muốn đề cập đến việc hoàn thành thay thế bao gồm tái sử dụng phần mềm, và những quan hệ đối tác hoặc là thầu lại với những công ty mà có hiểu biết chuyên môn hoặc nhân viên có chuyên môn có thể đảm bảo những điều khoản của đề xuất

3) Những khía cạnh hình thức của mối quan hệ giữa khách hàng và công ty phần mềm phải được ghi rõ.

Đề xuất nên định nghĩa những thủ tục bao gồm:

- Sự giao tiếp với khách hàng và những kênh giao diện
- Việc chuyển giao dự án và tiêu chuẩn được chấp nhận
- Tiến trình phê chuẩn pha hình thức
- Phương thức tiếp theo khách hàng thiết kế và kiểm tra
- Thủ tục khách hàng thay đổi yêu cầu

4) Xác định những rủi ro khi phát triển

Những rủi ro khi phát triển, như không đủ kiến thức chuyên môn liên quan đến lĩnh vực nghiệp vụ của dự án hoặc cách sử dụng những công cụ phát triển yêu cầu, cần được xác định và giải quyết.

5) Ước lượng đầy đủ những tài nguyên và thời gian biểu của dự án

Việc ước lượng tài nguyên để cập đến đội ngũ nhân viên chuyên nghiệp và ngân sách của dự án, bao gồm cả chi phí cho các nhà thầu con. Việc ước lượng thời gian nên đưa vào những yêu cầu về thời gian của tất cả những bên tham gia vào dự án.

Lưu ý

Trong một vài tình huống, một nhà cung cấp có ý đề nghị cung cấp một chi phí thấp, xem xét các yếu tố như tiềm năng bán hàng. Trong trường hợp này, khi mà đề xuất dựa trên ước lượng thực tế của thời gian, ngân sách và khả năng chuyên môn, những tồn thất phát sinh được coi là một miscalculation có thể tính được, không phải là một hợp đồng thắt bại

6) Kiểm tra năng lực của công ty đối với dự án.

Việc kiểm tra này nên xem xét đến năng lực chuyên môn cũng như là khả năng sẵn sàng của những thành viên trong đội được yêu cầu và những khả năng phát triển trong thời gian đã được lập lịch.

7) Kiểm tra năng lực của khách hàng để đáp ứng những yêu cầu của mình

Việc kiểm tra này đề cập đến khả năng tài chính và tổ chức của khách hàng, như tuyển dụng và đào tạo nhân sự, cài đặt phần cứng yêu cầu và nâng cấp các thiết bị liên lạc.

8) Định nghĩa đối tác và nhà thầu phụ tham gia

Điều này bao gồm các vấn đề bảo đảm chất lượng, lịch trình thanh toán, phân phối thu nhập, lợi nhuận của dự án, và hợp tác giữa quản lý dự án và các đội.

9) Định nghĩa và bảo vệ quyền sở hữu.

Yếu tố này có tầm quan trọng trong trường hợp tái sử dụng phần mềm, khi việc có thêm một gói mới vào hoặc có tái sử dụng phần mềm hiện nay trong tương lai hay không cần phải được quyết định. Nó cũng đề cập đến việc sử dụng các file độc quyền của các dữ liệu quan trọng cho hoạt động của hệ thống và các biện pháp an ninh.

Những mục tiêu của rà soát lại bản dự thảo đề xuất được tổng kết trong bảng sau:

Những mục tiêu của việc rà soát bản dự thảo đề xuất
9 mục tiêu của việc rà soát bản dự thảo đề xuất đảm bảo rằng những hành động sau đây được thực hiện một cách thỏa đáng:
<ol style="list-style-type: none">1. Những yêu cầu của khách hàng đã được giải thích chi tiết và có chú giải2. Lựa chọn những phương pháp thực hiện dự án đã được kiểm tra3. Những khía cạnh hình thức của mối quan hệ giữa khách hàng và công ty phần mềm phải được ghi rõ4. Xác định những rủi ro khi phát triển5. Ước lượng đầy đủ những tài nguyên và thời gian biểu của dự án6. Kiểm tra năng lực của công ty đối với dự án7. Kiểm tra năng lực của khách hàng để đáp ứng những yêu cầu của mình8. Định nghĩa đối tác và nhà thầu phụ tham gia9. Định nghĩa và bảo vệ quyền sở hữu

- **Những mục tiêu của rà soát dự thảo hợp đồng.**

Những mục tiêu của việc rà soát bản dự thảo hợp đồng để đảm bảo rằng những hoạt động sau đây được thực hiện một cách thỏa đáng:

1) Không có vấn đề chưa rõ ràng nào vẫn còn lại trong dự thảo hợp đồng

2) Tất cả những thỏa thuận đạt được giữa các khách hàng và công ty phải được giải thích đầy đủ và chính xác trong hợp đồng và phụ lục của nó. Những hiểu biết này được dùng để giải quyết tất cả các vấn đề chưa rõ ràng và khác biệt giữa khách hàng và công ty mà đã được đưa ra cho đến nay

3) Không có sự thay đổi, bổ sung, hoặc thiếu sót nào không được thảo luận và sự thỏa thuận nên được đưa vào dự thảo hợp đồng. Việc thay đổi, dù có ý hay không, có thể dẫn đến sự bổ sung đáng kể và những nhiệm vụ bất ngờ trong một bộ phận của nhà cung cấp.

Những mục tiêu của việc rà soát lại dự thảo hợp đồng có thể được tổng kết trong bảng 5.2

Những mục tiêu của việc rà soát dự thảo hợp đồng

Ba mục tiêu của việc rà soát dự thảo hợp đồng nhằm đảm bảo những hoạt động sau đây được thực hiện một cách thỏa đáng:

- 1) Không có vấn đề chưa rõ ràng nào vẫn còn lại trong dự thảo hợp đồng
- 2) Mọi thỏa thuận đã đạt được sau khi xem xét những đề xuất phải được chú giải một cách chính xác
- 3) Không có sự thay đổi, bổ sung, hoặc thiếu sót đưa vào bản dự thảo hợp đồng

3.3.2 Rà soát phân tích thiết kế

Danh mục rà soát thiết kế

Mã dự án:

Phiên bản sản phẩm:

Người rà soát:

Kích thước sản phẩm:

Công sức thực hiện rà soát (giờ công):

Câu hỏi	Yes	No	N/A	Ghi chú
Rà soát tài liệu				
Kiểm chứng xem các thủ tục rà soát tài liệu có được tuân thủ không bằng cách kiểm tra các tiêu chí dưới đây:				
Trang tiêu đề có tên tài liệu, mã phiên bản, ngày phát hành?				
Header và footer có ghi rõ tên và phiên bản tài liệu không?				
Phần đánh số trang có chỉ rõ tổng số trang của tài liệu không?				
Có thể theo dõi được lịch sử không?				
Tài liệu có gồm danh sách các tài liệu tham khảo không?				
Kiến trúc thiết kế				
Kiến trúc hệ thống có bao gồm các luồng dữ liệu, luồng điều khiển, các thành phần cao cấp và giao diện có được biểu diễn rõ ràng không?				
Các giao diện bên ngoài, bao gồm các giao diện người dùng có được định nghĩa và kiểm chứng không?				
Kiến trúc có được phân lớp thích hợp không?				
Chiến lược xử lý lỗi có được mô tả và kiểm chứng không?				
Các chiến lược vào/ra có được mô tả và kiểm chứng không?				
Thiết kế mức cao				
Tất cả các thành phần chính có được mô tả và kiểm chứng không?				
Luồng dữ liệu giữa các thành phần có được mô tả không?				
Các thuật toán chính có được mô tả và kiểm chứng không?				
Tất cả dữ liệu và tài nguyên được chia sẻ giữa các thành phần có được mô tả không?				
Có bắt được hết tất cả các trạng thái và sự kiện quan trọng của hệ thống không?				
Các cấu trúc dữ liệu chính có được mô tả và kiểm chứng không?				
Đầu vào các thủ tục có cần và đủ để thực hiện các hành động được yêu cầu không?				
Có chiến lược được mô tả và kiểm chứng cho:				
Việc xử lý các trạng thái đặc biệt ko? (Vd: kết thúc bất thường, khôi phục lỗi, mất kiểm soát)				
Việc xử lý hỏng hệ thống không? (Vd: kết thúc tiến trình, khôi phục hệ thống)				
Việc quản lý bộ nhớ không? Có ước lượng bộ nhớ đã sử dụng không?				
Việc quản lý các tài nguyên được chia sẻ không? Các module sử dụng tài nguyên được chia sẻ có được chỉ rõ không?				
Thiết kế chi tiết				
Mỗi đơn vị có một định danh duy nhất và tuân theo quy ước về kiểu và đặt tên không?				
Ngôn ngữ lập trình và các công nghệ được đề xuất có tương thích không?				

Có thiết kế cho từng đơn vị chứa các hiển thị thích hợp không? (Ví dụ: cấu trúc tĩnh, định nghĩa dữ liệu, luồng dữ liệu, kiểm soát luồng, trạng thái)			
Toàn bộ hàm và mục đích của mỗi đơn vị có được mô tả không?			
Các lời gọi, sự kiện và thông điệp giữa các đơn vị có được ghi trong tài liệu không?			
Thiết kế có phản ánh môi trường hoạt động thực tế không? Phản cứng? Phần mềm?			
Các ràng buộc như thời gian xử lý, thời gian chạy, sử dụng bộ nhớ, vào ra, truy cập cơ sở dữ liệu và thời gian phản hồi cho đơn vị này có được xem xét không?			
Lỗi, ngoại lệ và các xử lý và trạng thái bắt thường có được mô tả chi tiết không?			
Các thành phần dữ liệu đã được mô tả chi tiết chưa?			
Miền giá trị hợp lệ và các ràng buộc dữ liệu khác có được chỉ rõ không?			
Việc quản lý và sử dụng dữ liệu được chia sẻ và lưu trữ có được mô tả rõ ràng không?			
Các đầu vào và đầu ra của tất cả các giao diện đã đầy đủ và cần thiết chưa?			
Đã mô tả chi tiết các module được yêu cầu như sau chưa:			
Có mô tả các tác nhân và các trường hợp sử dụng của hệ thống không?			
Có mô tả sơ đồ luồng của tất cả các hàm public không?			
Có mô tả các biến public được sử dụng trong hệ thống không?			
Có mô tả định dạng dữ liệu bên ngoài được sử dụng trong hệ thống không?			
Có mô tả giao diện người dùng của hệ thống như sau không:			
Có mô tả/sắp xếp vị trí của tất cả các phần tử trong giao diện người dùng không?			
Có mô tả các tiến trình hoặc hành động cho giao diện người dùng không?			
Có mô tả quan hệ giữa các phần tử trong giao diện người dùng và các module hoặc các hàm public không?			

*** Nhận xét**

*** Kết luận**

- [] - Qua
- [] - Rà soát
- [] - Khác

3.3.3 Các hoạt động rà soát khác

Danh mục rà soát code

Mã dự án:

Phiên bản sản phẩm:

Người rà soát:

Ngày rà soát:

Kích thước sản phẩm:

Công sức thực hiện rà soát (giờ công):

Câu hỏi	Yes	No	N/A	Ghi chú
Chung				
Code có tuân thủ quy ước code (Coding convention) không?				
Chú thích (comment)				
Các chú thích (comment) có được cập nhật không?				
Chú thích (comment) có chính xác và rõ ràng không?				
Chú thích (comment) có tập trung vào việc giải thích tại sao thay vì giải thích như thế nào không?				
Những trường hợp ngoại lệ và các lỗi đã khắc phục có được chú thích (comment) lại không?				
Có chú thích (comment) mục đích của mỗi hành động không?				
Mã nguồn				
Tên của hành động có diễn tả mục đích của hành động đó không?				
Tên của đối số có ý nghĩa mô tả không?				
Luồng đúng của mỗi hành động có thể phân biệt được rõ ràng với các luồng ngoại lệ khác không?				
Hành động có quá dài, có thể rút gọn bằng cách chia ra thành các hành động nhỏ hơn không?				
Hành động có quá dài, có thể rút gọn bằng cách giảm bớt các điểm rẽ nhánh không? (Điểm rẽ nhánh là một câu lệnh mà tại đó code có thể đi theo các đường khác nhau, ví dụ như các câu lệnh if-, else-, while-, case-.)				
Các vòng lặp lồng có được giảm đến mức tối thiểu không?				
Các biến có được đặt tên đúng ý nghĩa và đúng chuẩn hay không?				

Code có dễ hiểu và tránh sử dụng giải pháp "thông minh" không?			
Mỗi lần đoạn code được cập nhật thì có thay đổi mô tả hay không?			
Những đoạn code phức tạp dễ gây nhầm lẫn có được chú thích (comment) và giải thích không?			
Với những đoạn code nhiều dòng thì có lùi đầu dòng hợp lý không?			
Có nhiều câu lệnh nằm cùng trên một dòng hay không?			
Có sử dụng ký hiệu xuống dòng khi câu lệnh quá dài không?			
Những câu lệnh con của câu lệnh trước có được lùi đầu dòng không?			
Các biến có được đặt tên khác với tên các đối tượng không?			
Các hàm có được đặt tên theo quy tắc chung không?			
Các hàm toàn cục và các hàm cục bộ có tên khác nhau không?			
Tên hàm có ý nghĩa không?			
Tên đối tượng có ý nghĩa và tuân theo chuẩn chung của công cụ phát triển không?			
Cách đặt tên thư mục và thư viện có được xác định trong tài liệu thiết kế không?			
Tên và kiểu của thư mục có tuân theo nội dung và chuẩn của công cụ phát triển không?			

*** Nhận xét**

*** Kết luận**

- [] - Qua
- [] - Rà soát lại
- [] - Khác

Danh mục rà soát thiết kế cơ sở dữ liệu

Mã dự án:

Phiên bản sản phẩm:

Người rà soát:

Ngày rà soát:

Kích thước sản phẩm:

Công sức thực hiện rà soát (giờ công):

Câu hỏi	Yes	No	N/A	Ghi chú
Thiết kế mức logic				
Cơ sở dữ liệu đã được chuẩn hóa thích hợp chưa?				
Tất cả các thuộc tính trong thiết kế đều được yêu cầu?				
Khóa chính của mỗi thực thể có chính xác không?				
Tất cả các thực thể đã được chuẩn hóa dựa trên khóa chính chưa?				
Quan hệ giữa các thực thể có chính xác không?				
Thiết kế mức vật lý				
Thiết kế mức vật lý có tương thích với thiết kế mức logic không?				
Kích thước và kiểu của các trường đã được xác định chính xác chưa?				
Các bảng đã được đánh chỉ số (index) cần thiết chưa?				
Thiết kế cơ sở dữ liệu có đủ linh hoạt để sau này có thể thêm các trường không?				
Kiểm tra các chuẩn cơ sở dữ liệu?				

* Nhận xét

* Kết luận

- Qua
- Rà soát lại
- Khác

Danh mục rà soát kế hoạch kiểm thử

Mã dự án:

Phiên bản sản phẩm:

Người rà soát:

Ngày rà soát:

Kích thước sản phẩm:

Công sức thực hiện rà soát (giờ công):

Câu hỏi	Yes	No	N/A	Ghi chú
KIỂM SOÁT TÀI LIỆU				
Kiểm tra xem các thủ tục kiểm soát tài liệu có tuân theo các điều sau đây không:				
<i>Trang tiêu đề có tên tài liệu, số phiên bản, ngày phát hành không?</i>				
<i>Phần đầu (header) hoặc phần chân (footer) tài liệu có ghi rõ tên và phiên bản tài liệu không?</i>				
<i>Phần đánh số trang có biểu thị tổng số trang của tài liệu không?</i>				
<i>Có thể theo dõi lịch sử không?</i>				
<i>Tài liệu có bao gồm danh sách các tài liệu tham khảo không?</i>				
Danh mục kế hoạch kiểm thử				
Các sản phẩm cần kiểm thử đã được xác định chưa?				
Kế hoạch có đưa ra phạm vi rõ ràng cho những mục sau đây không?				
<i>Kiểm thử đơn vị</i>				
<i>Kiểm thử tích hợp</i>				
<i>Kiểm thử hệ thống</i>				
<i>Kiểm thử chấp nhận</i>				
Kế hoạch có tham chiếu đến các yêu cầu như đã được chỉ rõ trong đặc tả yêu cầu hệ thống không?				
Các điều kiện làm dừng kiểm thử có được nêu ra không?				
Các kiểu kiểm thử có được nêu ra sau đây không?				
<i>Kiểm thử chức năng</i>				
<i>Kiểm thử giao diện người dùng</i>				
<i>Kiểm thử tích hợp dữ liệu và cơ sở dữ liệu</i>				
<i>Kiểm thử hiệu năng</i>				
<i>Kiểm thử kiểm soát bảo mật và truy cập</i>				
Tiêu chí chấp nhận đối với các yêu cầu đã được xác định chưa?				
Các kiểu kiểm thử có phản ánh được tất cả yêu cầu được chỉ rõ trong bản đặc tả yêu cầu hệ thống không?				
Phần mô tả mỗi loại kiểm thử có bao gồm những mục sau?				
<i>Mục đích kiểm thử</i>				
<i>Kỹ thuật</i>				
<i>Tiêu chí hoàn thành</i>				
<i>Xem xét đặc biệt</i>				
Môi trường kiểm thử, công cụ kiểm thử, phần cứng và phần mềm kiểm thử				
Tiêu chí qua/loại (Pass/Fail) của sản phẩm được kiểm thử có được định nghĩa với sự đánh giá đặc biệt tương thích với yêu cầu trong đặc tả yêu cầu hệ thống không?				

Người thực hiện và trách nhiệm của mỗi người có được xác định một cách chính xác không?			
Sản phẩm bàn giao kiểm thử có được định nghĩa không?			
Lịch kiểm thử có tương thích với lịch phát triển không?			
Các yêu cầu về bối cảnh của nhân viên và đào tạo có được xác định rõ không?			
Các rủi ro trong kiểm thử có được xác định không?			

*** Nhận xét**

*** Kết luận**

- Qua
- Rà soát lại
- Khác

Danh mục rà soát giao diện người dùng

Mã dự án:

Phiên bản sản phẩm:

Người rà soát:

Ngày rà soát:

Kích thước sản phẩm:

Công sức thực hiện rà soát (giờ công):

Câu hỏi	Yes	No	N/A	Ghi chú
Thiết kế đồ họa của giao diện người dùng được khách hàng duyệt hay chấp nhận chưa?				
Tất cả các nhãn (label) đã được căn lề đều nhau chưa?				
Các vùng nhập văn bản (text area) đã được căn lề đều nhau chưa?				
Có bị lỗi chính tả không?				
Các vùng không được phép nhập có được để cùng màu (xám) không?				
Các thành phần có được co/giãn khi thu nhỏ/phóng to cửa sổ không?				
Trình tự tab có chính xác không?				
Khi đang thực hiện tiêu trình ngầm thì con trỏ có biến thành đồng hồ cát không?				
Con trỏ có trở về mũi tên sau khi thực hiện tác vụ không?				
Độ dài tối đa của các trường nhập có phù hợp với độ dài dữ liệu không?				
Các trường được phép nhập chỉ chấp nhận nhập vào độ dài chuỗi cho trước không?				
Có sử dụng các từ viết tắt không?				
Người dùng cuối có thể hiểu được các từ đó không?				
Chúng có được sử dụng thống nhất trong tất cả các màn hình không?				
Cá chuột và phím bấm đã được xử lý chưa?				
Việc kết hợp màu sắc sử dụng trên các màn hình đã thống nhất chưa?				

Nếu có sử dụng các phím nóng hay tổ hợp phím, chúng có nhất quán không?				
Nếu câu trên đúng, thì điều này đã được kiểm thử chưa?				
Các thông báo lỗi có được hiển thị bằng thuật ngữ thông thường (không phải thuật ngữ kỹ thuật) và người dùng có thể hiểu được không?				
Các thông báo lỗi có cung cấp thông tin không?				
Trước khi một hành động làm ảnh hưởng đến hệ thống (ví dụ là hành động xóa) được thực hiện, người dùng có được hỏi xác nhận không?				
Định dạng ngày tháng và giờ trong hệ thống có nhất quán không?				
Định dạng số có nhất quán không?				
Nếu có sử dụng chú thích (tooltip) thì các chúng có hữu ích hoặc có ý nghĩa không?				
Các màn hình và thực đơn (menu) có ánh xạ chính xác không?				

* **Nhận xét**

* **Kết luận**

- [] - Qua
- [] - Rà soát lại
- [] - Khác

Danh mục rà soát ca kiểm thử

Mã dự án:

Phiên bản sản phẩm:

Người review:

Ngày review:

Công sức thực hiện review (giờ công):

Câu hỏi	Yes	No	N/A
Kiểm soát tài liệu			
Kiểm chứng xem các thủ tục kiểm soát tài liệu có tuân theo các kiểm tra sau không:			
<i>Trang tiêu đề có tên tài liệu, số phiên bản, ngày phát hành không?</i>			
<i>Phần đầu (header) hoặc phần chân (footer) tài liệu có ghi rõ tên và phiên bản tài liệu không?</i>			
<i>Phần đánh số trang có biểu thị tổng số trang của tài liệu không?</i>			
<i>Có thể theo dõi lịch sử không?</i>			
<i>Tài liệu có bao gồm danh sách các tài liệu tham khảo không?</i>			
Danh mục rà soát ca kiểm thử			
Các ca kiểm thử có phủ hết được tất cả yêu cầu không?			
Các ca kiểm thử có phủ hết được các kiểu test được mô tả trong kế hoạch kiểm thử (Test plan) không?			
Mỗi ca kiểm thử có được gán định danh không?			
Mỗi test case có chỉ rõ:			
<i>Hành động</i>			
<i>Điều kiện test</i>			
<i>Kết quả mong đợi</i>			
Các kết quả mong đợi có được ghi lại chi tiết không?			
Các ca kiểm thử để xác nhận trường, bản ghi và cập nhật cơ sở dữ liệu có bao gồm các điều sau không?			
<i>Các điều kiện hợp lệ</i>			
<i>Các điều kiện không hợp lệ</i>			
<i>Các điều kiện biên hoặc bất thường</i>			
Các mã code thông báo/thông báo lỗi có được chỉ rõ trong kết quả mong đợi cho các ca bất thường không?			
Sự độc lập giữa các ca kiểm thử có được mô tả không?			
Tất cả môi trường đã yêu cầu có được chỉ rõ không?			
Các điều kiện trước (tiền điều kiện) để kiểm thử có được chỉ rõ không?			

* Nhận xét

* Kết luận

- [] - Qua
- [] - Rà soát lại
- [] - Khác

3.4 Đảm bảo chất lượng của các thành phần bảo trì phần mềm

3.4.1 Giới thiệu

Giai đoạn chính của vòng đời phần mềm là giai đoạn hoạt động, thường kéo dài từ 5 đến 10 năm, mặc dù cũng có những trường hợp hoạt động kéo dài đến 15 năm và thậm chí là hơn thế nữa cũng không phải là hiếm. Vậy điều gì đã khiến cho gói phần mềm có thể đạt được “tuổi thọ cao” với nhóm người dùng mà đã hài lòng nó, trong khi gói phần mềm khác cũng phục vụ đối tượng đó lại “bị hủy sớm”? Yếu tố chính chịu trách nhiệm cho dịch vụ dài hay thành công là bảo trì chất lượng. Việc bảo trì phần mềm quan trọng như thế nào có thể được giả định thông qua việc chú ý đến chủ đề được đưa ra trong chuẩn ISO 9000 – 3 (xem ISO(1977), phiên bản 4.19 và ISO/IEC (2001), phiên bản 7.5), IEEE (1998) và Oskarsson và Glass (1996).

Dưới đây là 3 thành phần của dịch vụ bảo trì và nó được xem như là bản chất của sự thành công:

- Bảo trì sửa lỗi: những dịch vụ hỗ trợ người dùng và sửa lỗi phần mềm.
- Bảo trì thích ứng: làm cho các gói phần mềm thích ứng với những yêu cầu mới của khách hàng và điều kiện môi trường thay đổi.
- Bảo trì cải thiện chất lượng: kết hợp (1) bảo trì hoàn thiện những chức năng mới được thêm vào phần mềm cũng như nâng cao hiệu suất, cùng với (2) bảo trì phòng ngừa – những hoạt động cải thiện độ tin cậy và cơ sở hạ tầng hệ thống cho dễ dàng làm cho việc bảo trì trong tương lai hiệu quả hơn.

Những dịch vụ hỗ trợ người dùng (“những trung tâm hỗ trợ người dùng”) trong bảo trì sửa lỗi có thể cần phải rõ ràng (clarification). Những dịch vụ hỗ trợ giải quyết tất cả những khó khăn xuất hiện khi sử dụng hệ thống phần mềm của người dùng; những dịch vụ sửa lỗi phần mềm thường được tích hợp trong dịch vụ này. Những khó khăn của người dùng có thể được gây ra bởi nguyên nhân sau:

- Lỗi code (thường được dùng với thuật ngữ “thất bại phần mềm”-software failure”).
- Lỗi viết tài liệu thủ công, giúp những màn ảnh hoặc các hình thức tài liệu hướng dẫn được chuẩn bị sẵn sàng cho người dùng. Trong trường hợp này, dịch vụ hỗ trợ có thể cung cấp cho người dùng với những chỉ dẫn đúng (mặc dù không có sự chính xác trong chính tài liệu phần mềm được thực hiện).
- Không đầy đủ, mơ hồ hoặc tài liệu không chính xác.
- Thiếu kiến thức về hệ thống phần mềm của người dùng hoặc thất bại của họ khi sử dụng tài liệu được cung cấp. Trong những trường hợp này không có xét đến thất bại hệ thống phần mềm.

Ba nguyên nhân đầu tiên ở trên được xem như là những thất bại của hệ thống phần mềm.Thêm vào đó, sự tích hợp của những dịch vụ hỗ trợ người dùng và những dịch vụ hiệu chỉnh phần mềm nói chung là đã được hoàn thành trong sự kết hợp gần gũi, với nhiều thông tin chia sẻ. Những thành phần khác của dịch vụ bảo trì – cải thiện chất lượng và bảo trì thích ứng – khuynh

hướng không được bắt đầu bởi những dịch vụ hỗ trợ người dùng. Trong hầu hết các trường hợp, những công việc cải thiện chức năng và khả năng thích ứng sẽ trình bày những đặc tính của một dự án nhỏ hoặc lớn, điều đó phụ thuộc vào những mong muốn của khách hàng. Trong trường hợp này, những công việc trên có thể được thực thi như là một phần của quá trình phát triển phần mềm. Khi xem xét ở trên, việc bao gồm những dịch vụ hỗ trợ người dùng trong những hoạt động bảo trì sửa lỗi là hợp lý.

Nói chung, chúng ta có thể nói rằng việc bảo trì sửa lỗi bảo đảm những người dùng hiện thời có thể thao tác hệ thống như đã được vạch rõ, bảo trì thích ứng có thể mở rộng cho những đối tượng người dùng, và bảo trì cải thiện chất lượng đưa ra giai đoạn dịch vụ của gói.

Nhu đã đề cập trong mục trước, việc kết hợp 3 thành phần bảo trì phần mềm chiếm hơn 60% tổng số tài nguyên thiết kế và lập trình dành cho hệ thống phần mềm trong suốt vòng đời của nó (Pressman, 2000). Những ước lượng khác chia sẻ những tài nguyên bảo trì kéo dài từ trên 50% (Lientz và Swanson, 1980) đến 65 – 75 % tổng số những tài nguyên phát triển dự án.

Mục tiêu hoạt động QA bảo trì phần mềm:

- (1) Đảm bảo, với mức độ tin cậy được chấp nhận, rằng những hoạt động bảo trì phù hợp với những yêu cầu kỹ thuật chức năng.
- (2) Đảm bảo, với mức độ tin cậy được chấp nhận, rằng những hoạt động bảo trì phù hợp với những yêu cầu quản lý lập lịch và ngân sách.
- (3) Những hoạt động khởi đầu và quản lý nhằm cải thiện và tăng hiệu quả cho bảo trì phần mềm và những hoạt động SQA. Điều này liên quan đến việc cải thiện cái nhìn toàn cảnh để đạt được những yêu cầu về chức năng và quản lý trong khi giá thành giảm.

3.4.2 Cơ sở cho chất lượng bảo trì cao

Nhiều người cho rằng chất lượng của gói phần mềm được bảo trì có lẽ là cơ sở quan trọng nhất nằm dưới chất lượng của những dịch vụ bảo trì. Nhưng cũng có những nhà phê bình khác cho rằng đó là chính sách bảo trì. Dưới đây là những thảo luận về chủ đề này.

3.4.2.1 Cơ sở 1: Chất lượng gói phần mềm

Chất lượng của gói phần mềm được bảo trì rõ ràng bắt nguồn từ sự thành thạo và những nỗ lực của nhóm phát triển cũng như những hoạt động SQA được thực hiện xuyên suốt quá trình phát triển. Nếu chất lượng của gói phần mềm là nghèo nàn thì cũng dẫn đến việc bảo trì nghèo nàn hoặc không có hiệu quả. Khi mà hiểu sâu sắc cơ sở này, chúng ta sẽ lựa chọn việc nhấn mạnh 7 trong 11 nhân tố đảm bảo chất lượng ban đầu mà có tác động trực tiếp trong bảo trì phần mềm. Đặc biệt là, chúng ta sẽ thảo luận về 2 trong 5 nhân tố thao tác sản phẩm, 3 nhân tố xem xét lại sản phẩm và 2 trong 3 nhân tố chuyển giao sản phẩm.

Hai nhân tố thao tác sản phẩm:

- **Sự chính xác** (Correctness) – bao gồm:
 - Sự chính xác của đầu ra: Việc hoàn thành những đầu ra được chỉ rõ (nói cách khác là không có đầu ra mà đã được chỉ ra trước đó bị thiếu), sự đúng đắn của những đầu ra (những đầu ra của hệ thống được xử lý một cách chính xác), đầu ra hợp thời (thông tin được xử lý luôn được cập nhật như đã chỉ rõ) và đầu ra có tính sẵn sàng (Những lần tương tác không vượt những giá trị cực đại xác định, đặc biệt là những ứng dụng trực tuyến và thời gian thực).
 - Sự chính xác của tài liệu hướng dẫn: Chất lượng của tài liệu hướng dẫn: tính đầy đủ, sự đúng đắn, kiểu và cấu trúc tài liệu. Những hình thức tài liệu bao gồm bản sao trên giấy và những file máy tính – được in ấn thông thường cũng như những file “trợ giúp” điện tử - trong khi phạm vi của nó chứa đựng những tài liệu cài đặt, tài liệu hướng dẫn người dùng và tài liệu của người lập trình viên.
 - Viết mã đúng quy cách: phù hợp với những hướng dẫn mã, đặc biệt là những hạn chế và sự phức tạp trong mã biến đổi cũng như xác định kiểu viết mã chuẩn.
 - **Độ tin cậy**. Tần số của những thất bại của hệ thống cũng như những lần phục hồi.

Ba nhân tố xem xét lại sản phẩm:

- **Sự bảo trì**: Những yêu cầu này được thực hiện đầu tiên và tốt nhất bằng cách làm theo cấu trúc phần mềm và những yêu cầu kiểu cũng như là theo cài đặt những yêu cầu trong tài liệu của lập trình viên.
- **Tính linh động**: Đạt được thông qua việc thiết kế, lập kế hoạch thích hợp và những đặc tính cung cấp không gian ứng dụng lớn hơn nhu cầu của người dùng hiện tại.
- **Có thể test được**: Có thể test bao gồm tính sẵn sàng của những chuẩn đoán hệ thống được đưa ra bởi người dùng cũng như những chuẩn đoán thất bại được áp dụng bởi trung tâm hỗ trợ hoặc những nhân viên bảo trì tại vị trí người dùng.

Cuối cùng là 2 nhân tố chuyển phát sản phẩm:

- (1) **Tính khả chuyển**: Các ứng dụng tiềm tàng của phần mềm trong môi trường phần cứng và hệ điều hành khác nhau, nó gồm những hoạt động mà cho phép thực hiện những ứng dụng đó.
- (2) **Thao tác giữa các phần**: Khả năng của các gói giao tiếp với các gói hoặc thiết bị tính toán khác. Bằng việc cung cấp khả năng đáp ứng những chuẩn về giao diện và áp dụng những giao diện đó một cách phù hợp kết hợp với những hướng dẫn của thiết bị sản xuất và phần mềm, thao tác giữa các phần có thể đạt được ở mức cao.

3.4.2.2 Cơ sở 2: Chính sách bảo trì

Những thành phần của chính sách bảo trì chính ảnh hưởng đến sự thành công của việc bảo trì phần mềm là sự phát triển các phiên bản và thay đổi chính sách để được áp dụng trong suốt vòng đời phần mềm.

Chính sách phát triển phần mềm

Chính sách này có quan hệ mật thiết với câu hỏi có bao nhiêu phiên bản phần mềm được vận hành cùng một lúc. Rõ ràng rằng, đây không phải là vấn đề dịch vụ của một tổ chức về phần mềm được đặt hàng mà vấn đề chính ở đây là số lượng các phiên bản hay chính là giá thành (COSTs) của những gói phần mềm được lên kế hoạch để phục vụ cho nhiều khách hàng khác nhau. Chính sách phát triển phiên bản sau này có thể được thực hiện dưới dạng “tuần tự” hoặc “cây”. Khi áp dụng chính sách tuần tự, chỉ một phiên bản được tạo ra sẵn cho toàn bộ khách hàng. Phiên bản này gồm số lượng lớn những ứng dụng mà biểu lộ sự dư thừa cao, một thuộc tính mà cho phép phần mềm phục vụ tất cả những mong muốn của khách hàng. Phần mềm phải được xem lại một cách định kỳ nhưng một khi một phiên bản mới được hoàn thành, nó sẽ thay thế phiên bản đang được sử dụng hiện thời bởi toàn bộ đối tượng người dùng.

Khi áp dụng chính sách phiên bản “cây”, nhóm bảo trì phần mềm hỗ trợ những nỗ lực marketing (quảng cáo/tiếp thị) bằng việc phát triển một phiên bản chuyên dụng, nhằm tới những nhóm khách hàng hay khách hàng chính một khi nó được yêu cầu. Một phiên bản mới được bắt đầu bằng việc thêm những ứng dụng đặc biệt hoặc bỏ qua những ứng dụng, phụ thuộc vào những gì liên quan đến nhu cầu của khách hàng. Những phiên bản này thay đổi theo sự phức tạp và mức của ứng dụng – những ứng dụng hướng công nghiệp được hướng tới, vv ... Nếu chính sách này được chấp nhận, gói phần mềm có thể tiến triển thành một gói đa phiên bản sau vài năm của dịch vụ, có nghĩa rằng giống như một cái cây, với vài nhánh chính và nhiều nhánh phụ, từng phần nhánh đại diện cho một phiên bản đã được duyệt lại một cách chuyên dụng. Trái với phiên bản phần mềm dạng tuần tự, việc bảo trì và quản lý của phiên bản phần mềm dạng cây phức tạp và tốn thời gian hơn nhiều. Xem xét sự thiếu sót này, những tổ chức phát triển phần mềm cố gắng áp dụng chính sách phiên bản dạng cây một cách giới hạn, chỉ cho phép một số lượng nhỏ những phiên bản phần mềm được phát triển.

Chính sách thay đổi

Chính sách thay đổi đề cập đến phương thức để kiểm tra mỗi yêu cầu thay đổi và tiêu chuẩn sử dụng cho những phương pháp đó. Một chính sách là rõ ràng nếu nó được thực thi bởi CCB(The Change Control Board _ Ban điều khiển thay đổi) hoặc những người được ủy quyền để phê duyệt những thay đổi đó. Chính sách cần bằng đòi hỏi có một cuộc kiểm tra tỉ mỉ những yêu cầu thay đổi, điều này là rất phù hợp vì như vậy nó cho phép nhân viên tập trung vào những thay đổi quan trọng và có ích nhờ thế chúng ta mới có thể thực thi công việc trong thời gian hợp lý và theo những chuẩn chất lượng mong muốn.

3.4.3 Các thành phần chất lượng phần mềm tiền bảo trì

Giống như những thành phần SQA tiền dự án, những hoạt động SQA trước bảo trì cần được hoàn thành để khởi tạo các dịch vụ bảo trì cần thiết cũng rất quan trọng. Nó bao gồm thứ tự các bước sau:

- Xem xét lại hợp đồng bảo trì

- Xây dựng kế hoạch bảo trì.

3.4.3.1 Xem xét lại hợp đồng bảo trì

Khi xem xét hợp đồng bảo trì, quan điểm mở rộng nên được khái quát. Đặc biệt, những quyết định được yêu cầu về các loại hình dịch vụ cần được ký kết trong hợp đồng. Những quyết định này phụ thuộc vào các loại hình dịch vụ khách hàng: dành cho khách hàng mà có gói custom-made được phát triển; dành cho khách hàng mà mua gói phần mềm COST và những khách hàng bên trong. Vì vậy, trước khi bắt đầu cung cấp những dịch vụ bảo trì phần mềm tới từng nhóm khách hàng trên thì hợp đồng bảo trì tương ứng phải được hoàn thành để nó đánh giá tổng số những trách nhiệm bảo trì theo những điều kiện liên quan.

Giả định sự thực thi

Những dịch vụ bảo trì tới khách hàng bên trong thường không có hợp đồng. Trong một số trường hợp cụ thể, một vài dịch vụ được cung cấp trong quá trình thực hiện mà không có sự xác định rõ ràng cho những dịch vụ đó. Trong những trường hợp như vậy thì sự không hài lòng dễ xảy ra trong cả 2 phía: những khách hàng bên trong cảm thấy rằng họ mong muốn được hỏi ý kiến một cách có thiện ý thay vì nhận những dịch vụ một cách đều đặn mà họ không mong chờ, bên cạnh đó thì nhóm phát triển lại luôn yêu cầu thực thi việc bảo trì như là một việc bắt buộc một khi họ đã làm việc trong một dự án khác.

Để tránh tình trạng căng thẳng, một “hợp đồng dịch vụ bên trong” nên được viết. Trong tài liệu này, những dịch vụ này được cung cấp bởi nhóm bảo trì bên trong tới những khách hàng bên trong được xác định một cách rõ ràng. Với việc bỏ qua hầu hết những hiểu sai liên quan đến những dịch vụ áo, hợp đồng này có thể đáp ứng một cách cơ bản những bảo trì thỏa đáng tới những khách hàng bên trong.

Những hoạt động xem xét lại hợp đồng bảo trì bao gồm việc nhìn lại những bản nháp đề xuất cũng như là nhìn lại những bản hợp đồng nháp. Thực tế, việc nhìn lại mục tiêu và sự thực thi bản hợp đồng bảo trì là đi theo từng dòng trong việc xem xét lại bản hợp đồng tiền dự án.

Dưới đây là những mục tiêu chính trong việc xem xét lại hợp đồng bảo trì phần mềm:

- **Sự rõ ràng trong yêu cầu của khách hàng**

Những vấn đề sau đây đặc biệt được quan tâm:

- Loại dịch vụ bảo trì sửa đổi được yêu cầu: Danh sách những dịch vụ từ xa và những dịch vụ tại chỗ được cung cấp, giờ phục vụ, thời gian phản hồi,....
- Số lượng người sử dụng và kiểu ứng dụng được dùng.
- Những người dùng địa phương, ở khoảng cách xa (hoặc qua đại dương) và các kiểu dịch vụ được cài đặt tại mỗi nơi đó.
- Cung cấp bảo trì cải thiện chức năng, khả năng thích ứng và thủ tục cho những yêu cầu của dịch vụ cũng như đề xuất và phê duyệt việc thực thi cho những dịch vụ này.

- **Xem xét lại những phương pháp tiếp cận khác về các điều khoản trong văn bản bảo trì**

Xem xét những suy xét cụ thể dưới đây để đưa ra lựa chọn phù hợp:

- Những hợp đồng con cho những site (địa điểm) hoặc kiểu dịch vụ
- Việc thực thi một vài dịch vụ thông qua khách hàng cùng với sự hỗ trợ từ nhóm bảo trì của nhà cung cấp.

- **Nhìn lại sự ước lượng về tài nguyên bảo trì được yêu cầu**

Đầu tiên, những ước lượng nên được kiểm tra lại trên cơ sở của những dịch vụ được yêu cầu, sắp xếp theo đề xuất của nhóm. Sau đó, dựa vào năng lực của công ty để xem có đáp ứng được những khía cạnh chuyên môn cũng như là tính sẵn sàng của nhóm bảo trì đã phân tích hay không.

- **Xem xét lại những dịch vụ bảo trì được cung cấp bởi những hợp đồng con và/hoặc khách hàng.**

Việc nhìn lại này đề cập tới việc định danh lại những dịch vụ được cung cấp bởi mỗi người tham gia, số tiền trả cho những hợp đồng con, đảm bảo chất lượng và những thủ tục tiếp theo được đáp ứng.

- **Xem xét lại những ước lượng về giá thành bảo trì**

Những ước lượng này nên được nhìn lại dựa trên cơ sở của những tài nguyên được yêu cầu.

3.4.3.2 *Lập kế hoạch bảo trì*

Lập kế hoạch bảo trì phải được chuẩn bị cho tất cả các khách hàng trong và ngoài. Những kế hoạch này nên cung cấp framework trong việc tổ chức những điều khoản bảo trì.

Lập kế hoạch đó bao gồm những bước sau:

- **Danh sách những dịch vụ bảo trì được ký kết**
- Khách hàng bên trong và ngoài, số lượng người sử dụng, sự định vị vị trí cho mỗi site khách hàng.
- Đặc tính của những dịch vụ bảo trì sửa đổi (xa hoặc tại chỗ).
- Trách nhiệm của việc bảo trì cải thiện chức năng và khả năng thích ứng phục vụ điều khoản của mỗi khách hàng.
- **Mô tả tổ chức của nhóm bảo trì**

Kế hoạch tổ chức nhóm bảo trì tập trung vào những yêu cầu về nhân sự. Điều đó có thể được xem xét cẩn thận theo những tiêu chuẩn sau:

- Số lượng thành viên nhóm được yêu cầu.
- Chất lượng thành viên nhóm được yêu cầu theo công việc bảo trì, bao gồm những người quen với gói phần mềm cần được bảo trì.
- Cấu trúc tổ chức của những nhóm bảo trì, bao gồm tên của những người lãnh đạo nhóm.
- Định nghĩa các công việc (trách nhiệm của khách hàng, kiểu ứng dụng,...) cho mỗi nhóm.

- Đào tạo khi cần thiết
- **Danh sách điều kiện thuận lợi cho bảo trì**

Những điều kiện thuận lợi cho bảo trì – cơ sở hạ tầng mà có thể cung cấp dịch vụ bao gồm:

- Trung tâm hỗ trợ bảo trì với thiết bị phần cứng và phần mềm đã được cài đặt để cung cấp những dịch vụ hỗ trợ cho người dùng và sửa đổi phần mềm.

- Tài liệu chính thức chứa tập các tài liệu đã hoàn thành:

(1) Tài liệu phần mềm, bao gồm tài liệu phát triển.

(2) Những hợp đồng dịch vụ

(3) Cấu hình phần mềm cho mỗi khách hàng và phiên bản của những gói phần mềm được cài đặt tại mỗi địa điểm và được cung cấp bởi nhà quản lý cấu hình.

(4) Những thông tin bảo trì lịch sử được ghi lại cho mỗi người dùng và khách hàng.

- **Danh sách những rủi ro dịch vụ bảo trì được xác định**

Rủi ro dịch vụ bảo trì liên quan đến hoàn cảnh mà thất bại xảy ra để cung cấp bảo trì tương ứng được dự đoán trước. Những rủi ro này bao gồm:

- Thiếu nhân viên trong suốt những dịch vụ bảo trì của tổ chức, trong trung tâm hỗ trợ bảo trì cụ thể hoặc trong những ứng dụng cụ thể.
- Sự hiểu biết và trình độ chuyên môn không tương xứng với từng phần của các gói phần mềm liên quan để thực thi những dịch vụ hỗ trợ người dùng và/hoặc những công việc bảo trì sửa đổi.
- Những thành viên nhóm không đủ khả năng để thực hiện những công việc cải thiện chức năng cũng như khả năng thích ứng.

- **Danh sách những thủ tục và điều khiển quản lý phần mềm được yêu cầu**

Hầu hết những thủ tục được yêu cầu để cập nhật đến những tiến trình được thực thi bởi những nhóm bảo trì sửa đổi và trung tâm hỗ trợ người dùng. Những thủ tục này giải quyết:

- Vận hành những ứng dụng của khách hàng.
- Xử lý những bản ghi lỗi của phần mềm
- Ghi chép định kỳ và liên tục những dịch vụ hỗ trợ người dùng
- Ghi chép định kỳ và liên tục những dịch vụ bảo trì sửa đổi.
- Đào tạo và cấp giấy chứng thực cho những thành viên bảo trì.
- **Ngân sách bảo trì phần mềm**

Những ước lượng được sử dụng trong ngân sách bảo trì sửa đổi dựa trên kế hoạch tổ chức nhân sự, trình độ chuyên môn được yêu cầu và sự đầu tư vốn cần để tạo ra những điều kiện, những nhu cầu của nhóm và tác vụ khác. Họ phải được chuẩn bị mỗi khi nhân sự, trình độ chuyên môn và các thủ tục được xác định. Những ước lượng cho việc bảo trì cải thiện chất lượng và khả năng thích ứng được chuẩn bị theo sự mong đợi, và theo sự thay đổi cũng như là cải thiện khi chúng được thực hiện.

3.4.4 Hỗ trợ đảm bảo chất lượng bảo trì phần mềm

Sự đa dạng của các công cụ đảm bảo chất lượng phần mềm được sử dụng trong các chu kỳ thực hiện của vòng đời phần mềm. Bản chất riêng biệt của mỗi thành phần bảo trì phần mềm: bảo trì sửa lỗi, bảo trì thích ứng và bảo trì cải thiện chức năng, yêu cầu các tập công cụ SQA khác nhau cần phải sử dụng cho mỗi loại. Hơn nữa, chu kỳ thực hiện của phần mềm điển hình thì mở rộng việc sử dụng các công cụ SQA cơ sở và công cụ giám sát việc quản lý.

Ý tưởng về việc đưa SQA vào pha bảo trì được Perry đưa ra (1995). Trong 1 nghiên cứu ông thực hiện vào tháng 11 năm 1994, các bên tham gia được thông báo rằng: dựa trên kinh nghiệm của họ thì 31% kế hoạch bảo trì của họ đã được thực hiện để đảm bảo chất lượng.

Phần tiếp theo sẽ nói về các nội dung sau:

- 1) Công cụ SQA cho bảo trì sửa lỗi
- 2) Công cụ SQA cho bảo trì cải thiện chức năng
- 3) Công cụ SQA cơ sở cho bảo trì phần mềm
- 4) Công cụ SQA cho giám sát quản lý bảo trì phần mềm.

3.4.4.1 Công cụ SQA cho bảo trì sửa lỗi

Các hoạt động bảo trì sửa lỗi đưa ra đầu tiên: (a) các dịch vụ hỗ trợ người sử dụng và (b) sửa chữa phần mềm (sửa lỗi). Các dịch vụ hỗ trợ người sử dụng giải quyết các trường hợp lỗi về mã phần mềm và lỗi tài liệu, tài liệu không hoàn thành hoặc mập mờ. Chúng có thể chứa những câu lệnh mà người sử dụng không đủ kiến thức về phần mềm hoặc thất bại trong việc sử dụng tài liệu có sẵn. Các dịch vụ sửa chữa phần mềm – gỡ lỗi và sửa tài liệu được sử dụng trong trường hợp lỗi phần mềm (failure), và được cung cấp trong suốt chu kỳ khởi tạo của việc thực hiện (mặc dù sự nỗ lực này được đầu tư vào việc testing) và tiếp tục được yêu cầu mặc dù không thường xuyên lắm. Dù hai kiểu dịch vụ khác nhau nhưng tập các công cụ đảm bảo chất lượng đều tập trung vào cùng một mục đích là đảm bảo chất lượng dịch vụ. Trong nhiều trường hợp cùng một đội có thể thực hiện cả hai kiểu bảo trì sửa lỗi.

Thêm vào các công cụ SQA giám sát việc quản lý và cơ sở hạ tầng (được thảo luận ở phần sau của chương) thì hầu hết các công việc sửa lỗi yêu cầu sử dụng các công cụ SQA cho vòng đời nhỏ, mainly mini-testing (test những phần nhỏ quan trọng). Thủ tục mini-testing được yêu cầu cho việc xử lý các tác vụ về lỗi đường dẫn (repair patch) (small - scale), nó được đặc trưng bởi số lượng nhỏ dòng lệnh thay đổi cùng nhau để hoàn thành việc sửa lỗi nhanh chóng. Những vấn đề liên quan đến sửa chữa trì hoãn giống như một abridged-mini-form của testing thường được sử dụng. Tuy nhiên, sử dụng những công cụ mini testing vẫn cần được thực hiện để tránh những trường hợp không thể test được.

Để đảm bảo chất lượng của mini testing, những chỉ dẫn sau nên được giữ vững:

- Testing được thực hiện bởi những tester chất lượng, không phải bởi người lập trình thực hiện sửa chữa

- Tài liệu thủ tục test (dài nhất là 2-3 trang) nên được chuẩn bị. Tài liệu bao gồm một bản mô tả về những ảnh hưởng biết trước được của việc sửa chữa, phạm vi sửa lỗi và một danh sách các test case được thực hiện. Tài liệu thủ tục trước test, tương tự như tài liệu thủ tục testing cũng nên được chuẩn bị để thực hiện test việc sửa lỗi được phát hiện trong test trước.
- Một bản ghi test đầy đủ viết về các lỗi phát hiện được dưới dạng tài liệu trong mỗi giai đoạn test và re-test nên được hoàn thành.
- Nhóm test chính xem lại tài liệu test về phạm vi sửa lỗi, tính đúng đắn của các test case và các kết quả test.
- Việc sửa chữa được cho là “simple và trivial”, đặc biệt đối với việc thực hiện tại site của khách hàng, mini-testing có thể được tránh đi.

Các dịch vụ bảo trì hợp đồng con, đặc biệt là các dịch vụ hỗ trợ người sử dụng đã trở nên khá phổ biến bát cứ khi nào có quá nhiều vấn đề hoặc không kinh tế cho nhà thầu bảo trì cung cấp các dịch vụ đó một cách trực tiếp. Công cụ chính để đảm bảo chất lượng của các dịch vụ bảo trì của nhà thầu phụ và mở đường cho các quan hệ suôn sẻ là hợp đồng contractor-subcontractor. Công cụ SQA được tích hợp vào hợp đồng nhằm mục đích:

- Các thủ tục xử lý phân loại cụ thể các cuộc gọi bảo trì.
- Tài liệu đầy đủ về các thủ tục dịch vụ.
- Có sẵn các bản ghi được viết tài liệu kiểm thử chuyên nghiệp của thành viên đội bảo trì của nhà thầu phụ, cho nhà thầu xem xét lại.
- Cấp giấy phép cho nhà thầu thực hiện xem xét lại theo chu kỳ các dịch vụ bảo trì cũng như sự thỏa mãn của khách hàng.
- Các điều kiện liên quan đến chất lượng thì yêu cầu chặt chẽ về hình phạt và sự kết thúc của ràng buộc hợp đồng con trong trường hợp cực đoan.

Một khi bảo trì sẵn sàng thực hiện, nhà thầu nên quản lý đều đặn việc xem lại dịch vụ bảo trì và sự thỏa mãn của khách hàng.

3.4.4.2 Các công cụ SQA cho bảo trì cải thiện chức năng

Do sự giống nhau của việc bảo trì cải thiện chức năng và dự án phát triển phần mềm, các công cụ vòng đời dự án được áp dụng cho bảo trì cải thiện chức năng. Những công cụ này cũng được thực thi cho bảo trì thích ứng phạm vi rộng (large scale adaptive maintenance).

Các công cụ SQA mở rộng được thực thi cho bảo trì cải thiện chức năng là các công cụ điều khiển quản lý và cơ sở hạ tầng, được thảo luận thêm ở phần sau.

3.4.4.3 Các thành phần cơ sở hạ tầng SQA cho bảo trì phần mềm

Các công cụ cơ sở đảm bảo chất lượng phần mềm là các thành phần quan trọng của bảo trì phần mềm. Sự cân đối của mảng các công cụ SQA cơ sở là bản chất và được thực thi trong suốt vòng đời của hệ thống phần mềm. Hơn nữa, sự giống nhau của các tiến trình cải thiện chức năng phần mềm và phát triển phần mềm cho phép các tiến trình dùng chung các công cụ SQA

cơ sở và chỉ có thay đổi nhỏ. Các công cụ cơ sở chuyên dụng được yêu cầu cho các hoạt động bảo trì sửa lỗi do các đặc trưng đặc biệt của các hoạt động này. Các hoạt động bảo trì thích ứng được phục vụ bởi các công cụ SQA cơ sở, theo các đặc trưng của nó. Các công cụ được áp dụng thường xuyên nhất là các công cụ SQA cải thiện chức năng, được theo bởi các công cụ SQA bảo trì sửa lỗi.

Thực tế, sự đóng góp của các công cụ SQA cơ sở cho việc bảo trì không bắt đầu tại thời điểm ban đầu của các tiến trình bảo trì. Rõ ràng là việc áp dụng thích hợp các công cụ SQA cơ sở bởi đội phát triển phần mềm đóng góp căn bản vào hiệu quả và hiệu năng các hoạt động của đội bảo trì. Nói cách khác, những công cụ này góp phần vào việc bảo trì đảm bảo chất lượng theo 2 cách: đầu tiên, bằng cách hỗ trợ đội phát triển phần mềm khi tạo ra phần mềm chất lượng cao, và thứ hai, bằng cách hỗ trợ đội bảo trì đáp ứng việc bảo trì của sản phẩm phần mềm giống nhau.

Các công cụ SQA cơ sở chuyên dụng được yêu cầu cho các tiến trình bảo trì phần mềm, đặc biệt là bảo trì sửa lỗi, thể hiện các đặc trưng đặc biệt. Ở đây chúng ta tập trung vào các công cụ SQA cơ sở chuyên dụng của các lớp sau:

- Các thủ tục bảo trì và các chỉ dẫn làm việc
- Hỗ trợ các thiết bị chất lượng
- Đào tạo và cấp chứng chỉ cho đội bảo trì
- Các hoạt động ngăn ngừa và sửa lỗi
- Quản lý cấu hình
- Viết tài liệu và điều khiển bản ghi chất lượng

Các thủ tục bảo trì và chỉ dẫn làm việc

Hầu hết các thủ tục bảo trì và các chỉ dẫn làm việc chuyên dụng được áp dụng cho các hoạt động bảo trì sửa lỗi và hỗ trợ người sử dụng, ví dụ:

- Xử lý từ xa các yêu cầu cho dịch vụ trong trường hợp lỗi phần mềm
- Xử lý tại chỗ các yêu cầu của khách hàng cho dịch vụ trong trường hợp phần mềm lỗi.
- Dịch vụ hỗ trợ người sử dụng
- Điều khiển đảm bảo chất lượng cho các hoạt động sửa lỗi phần mềm và hỗ trợ người dùng
- Điều tra sự thỏa mãn khách hàng.
- Cấp chứng chỉ cho các thành viên đội bảo trì sửa lỗi và hỗ trợ người sử dụng.

Các thiết bị hỗ trợ chất lượng

Bảo trì được mong đợi để phát triển các thiết bị chuyên dụng hỗ trợ các hoạt động sửa lỗi phần mềm và hỗ trợ người dùng: templates(khuôn mẫu), checklists (danh sách kiểm tra) và những cái tương tự thế. Các thiết bị có thể bao gồm:

- Danh sách kiểm tra các phần có thể gây ra lỗi – được áp dụng bởi nhà chuyên môn về bảo trì
- Các khuôn mẫu báo cáo lỗi phần mềm được giải quyết như thế nào, bao gồm sự phát hiện của các tiến trình sửa lỗi
- Danh sách kiểm tra cho việc chuẩn bị tài liệu thủ tục testing nhỏ.

Đào tạo và cấp chứng chỉ cho đội bảo trì

Đào tạo đội bảo trì giải quyết các công việc cải thiện chức năng không khác mấy so với việc đào tạo đội phát triển phần mềm khác. Tuy nhiên, đào tạo đặc biệt và cấp chứng chỉ mang tính cốt yếu cho đội bảo trì sửa lỗi.

Việc đào tạo các chuyên gia bảo trì sửa lỗi được thúc đẩy bởi yêu cầu hỗ trợ các dịch vụ cụ thể trong hợp đồng bảo trì. Do vậy, kế hoạch đào tạo nên cung cấp các giải pháp cho các yêu cầu nhân sự trong suốt các chu kỳ trọng tải cao nhất và các yêu cầu của tổ chức để thay thế nhân sự bị sa thải. Trong nhiều trường hợp, việc đào tạo các nhân viên bảo trì dự trữ là không đủ, phải đào tạo thêm hệ thống riêng biệt. Nói cách khác, chương trình đào tạo nghiêm ngặt được yêu cầu để cho phép tổ chức xác định phạm vi với mức độ ràng buộc của các dịch vụ riêng biệt cho các chu kỳ trọng tải cao và trong trường hợp thay đổi nhân viên bảo trì hoặc bất kỳ lý do nào khác.

Yêu cầu cấp chứng chỉ cho các nhân viên sửa lỗi phần mềm và hỗ trợ người dùng là gốc rễ trong các đặc trưng của các dịch vụ này. Sự quan tâm đặc biệt nên dành cho việc cấp chứng chỉ cho các nhân viên sửa lỗi phần mềm, người mà thường xuyên thực hiện công việc của họ dưới áp lực về thời gian lớn, làm việc 1 mình, và trong nhiều trường hợp làm việc tại site của khác hàng, nơi mà sự hỗ trợ chuyên gia từ team leader hoặc những người khác bị giới hạn.

Các hoạt động ngăn ngừa và sửa lỗi

Pha hoạt động của vòng đời phần mềm tạo ra thông tin có giá trị cao: các bản ghi lỗi phần mềm, sửa chữa các lỗi đó cũng như bản ghi các yêu cầu hỗ trợ khách hàng có thể dẫn tới các hoạt động ngăn ngừa và sửa lỗi do đó góp phần cải thiện hệ thống phần mềm đã có và phần mềm mới.

Để các tiến trình hoạt động hiệu quả, cần có các tiến trình thích hợp cho việc biểu diễn thông tin thu thập được, xem lại và phân tích những phát hiện, và đưa ra những gợi ý cho việc cải thiện các tiến trình bảo trì và phát triển liên quan. Những hoạt động SQA này được chỉ dẫn và điều khiển bởi ủy ban nội bộ - CAB (Corrective Action Board), được sáng lập trong tổ chức phát triển phần mềm chính.

Những vấn đề điển hình ban đầu cần xem lại gồm:

- Những thay đổi về nội dung và tính thường xuyên của khách hàng yêu cầu về các dịch vụ hỗ trợ người dùng.

- Tăng thời gian trung bình được đầu tư để tuân theo yêu cầu hỗ trợ của khách hàng.
- Tăng thời gian trung bình được đầu tư để sửa chữa các lỗi phần mềm của khách hàng.
- Tăng phần trăm các lỗi hiệu chỉnh phần mềm.

Quản lý cấu hình

Nhóm bảo trì đưa ra hầu hết tập các phụ thuộc vào quản lý cấu hình. Sự phụ thuộc này là kết quả của các mối quan hệ dựa trên kinh nghiệm với các gói phần mềm dịch vụ qua nhiều năm, trong suốt các phiên bản mới được thêm vào, phiên bản cũ bị thay thế và nhiều sự cài đặt mới và các thay đổi phần mềm được thực hiện.

Hai ứng dụng chung dựa vào quản lý cấu hình là: (1) sửa lỗi và (2) sự thay thế nhóm (group replacement) của các phiên bản phần mềm đang được sử dụng bằng các phiên bản mới, được khởi tạo bởi tổ chức bảo trì.

1. Sửa lỗi (Failure repair): Trong vấn đề về sửa lỗi phần mềm, việc hỗ trợ cập nhật và độ tin cậy là cần thiết theo dạng của:

- Thông tin quan tâm tới phiên bản hệ thống phần mềm được cài đặt tại site của khách hàng.
- Bản copy code hiện thời và tài liệu của nó.

→ Sự đóng góp vào chất lượng phần mềm được thể hiện là lỗi ít hơn trong các thử nghiệm hiệu chỉnh lỗi và giảm bớt được tài nguyên đầu tư vào sửa lỗi.

2. Group replacement (Thay thế nhóm): Thuật ngữ group trong SQA đề cập đến tất cả khách hàng có cùng phiên bản phần mềm được cài đặt tại site của họ. Do vậy, group replacement chỉ rõ rằng tất cả khách hàng sử dụng phiên bản sẽ được nhận phiên bản mới được phát triển hoặc cập nhật tại cùng 1 thời gian. Quản lý cấu hình hỗ trợ cho vấn đề thay thế nhóm dựa trên thông tin về các thành viên của nhóm khách hàng

- Đưa ra quyết định về tính thích hợp của việc thực hiện thay thế nhóm dựa trên quy mô của việc thay thế và kiểu hợp đồng đã ký với khách hàng.
- Lên kế hoạch thay thế nhóm, phân phối tài nguyên và xác định thời gian.

→ Sự đóng góp vào chất lượng phần mềm thể hiện ở việc thay thế phiên bản phần mềm hiện thời bởi những phiên bản đã được cải thiện mà giảm lỗi phần mềm, yêu cầu ít sự hỗ trợ. Cải thiện chất lượng cũng góp phần bảo trì phần mềm hiệu quả, yêu cầu ít tài nguyên cho bảo trì sửa lỗi.

Tài liệu bảo trì và bản ghi chất lượng

Những yêu cầu cụ thể cho tài liệu và bản ghi chất lượng có quan hệ chặt chẽ nhất với các hoạt động sửa lỗi phần mềm và hỗ trợ người sử dụng. Tài liệu và bản ghi chất lượng được chuẩn bị để:

- Cung cấp dữ liệu cần thiết cho các hoạt động ngăn ngừa và sửa lỗi.
- Hỗ trợ việc xử lý các yêu cầu hỗ trợ người dùng và các báo cáo lỗi khách hàng trong tương lai.
- Cung cấp bằng chứng để đáp ứng những yêu sách từ phía khách hàng trong tương lai.

Những yêu cầu tài liệu được liệt kê trong các thủ tục bảo trì khác nhau nên đáp ứng với tất cả những yêu cầu tài liệu trên

3.4.4.4 Những công cụ SQA giám sát quản lý cho bảo trì phần mềm

Trong khi các công cụ SQA giám sát quản lý cụ thể được yêu cầu cho các hoạt động bảo trì sửa lỗi, thì sự giống nhau của các tiến trình phần mềm mô tả việc cải thiện chức năng và bảo trì thích ứng cũng như việc phát triển phần mềm cho phép những tiến trình này được sử dụng cùng một công cụ quản lý. Đặc biệt, các thành phần SQA quản lý có ý nghĩa giúp cải thiện việc điều khiển bảo trì bằng cách tạo ra những báo động sớm tín hiệu làm giảm chất lượng của các dịch vụ và tăng tỷ số thất bại dịch vụ.

Phần còn lại của mục này sẽ đưa ra những vấn đề điều khiển quản lý chất lượng, những phần chính này đạt tới dịch vụ sửa lỗi phần mềm và hỗ trợ người dùng ở trên:

- Điều khiển hiệu năng cho các dịch vụ bảo trì sửa lỗi.
- Đo chất lượng cho bảo trì sửa lỗi.
- Chi phí chất lượng bảo trì phần mềm.

Điều khiển hiệu năng cho các dịch vụ bảo trì sửa lỗi

Điều khiển hiệu năng quản lý cho các dịch vụ bảo trì sửa lỗi khác với khi áp dụng các dịch vụ sửa lỗi phần mềm và các dịch vụ hỗ trợ người dùng. Các công cụ điều khiển quản lý mang lại, bên cạnh thông tin hiệu năng theo chu kỳ, còn là *những báo động sự chú ý quản lý*, như sau:

- **Sửa lỗi phần mềm**
 - Tăng việc sử dụng tài nguyên
 - Giảm tỉ lệ sửa chữa lỗi từ xa so với sửa chữa on-site của khách hàng.
 - Tăng tỷ lệ sửa chữa on-site ở những vùng cục bộ ở xa so với các dịch vụ hải ngoại
 - Tăng phần trăm thất bại gấp phải khi sửa chữa yêu cầu lập lịch
 - Tăng tỉ lệ sửa lỗi, và liệt kê các trường hợp “model” cụ thể của các tình huống lỗi cực đoan.

- Giảm bớt sự thỏa mãn khách hàng dựa trên các điều tra sự thỏa mãn của khách hàng.
- **Hỗ trợ người sử dụng**
 - Tăng tỉ lệ các yêu cầu dịch vụ cho các hệ thống phần mềm cụ thể, cho các kiểu dịch vụ,...
 - Tăng tài nguyên được sử dụng cho các dịch vụ hỗ trợ người dùng
 - Tăng tỉ lệ thất bại của việc cung cấp các dịch vụ được yêu cầu
 - Tăng tỉ lệ lỗi, và trường hợp cụ thể của các thất bại đáng chú ý
 - Thông tin thỏa mãn khách hàng dựa trên các nghiên cứu sự thỏa mãn của khách hàng

Những điều khiển sửa chữa lỗi quản lý này (những cái được mong đợi để đưa ra những cảnh báo) được thực hiện qua báo cáo theo chu kỳ, qua buổi họp nhân viên đều đặn, qua việc xem xét các trung tâm hỗ trợ bảo trì cung cấp dịch vụ, qua phân tích báo cáo xử lý độ đo bảo trì phần mềm và giá thành chất lượng bảo trì. Thông tin được tích lũy hỗ trợ các quyết định quản lý quan tâm tới kế hoạch và việc thực hiện bảo trì sửa lỗi.

Đo chất lượng bảo trì phần mềm

Đo chất lượng bảo trì phần mềm được sử dụng chủ yếu để nhận biết khuynh hướng trong hiệu quả bảo trì, hiệu năng và sự thỏa mãn của khách hàng. Đơn vị đảm bảo chất lượng phần mềm thường thực hiện các tiến trình đo chất lượng. Thay đổi xu hướng tích cực hay tiêu cực, cung cấp cơ sở định lượng cho các quyết định quản lý, quan tâm tới:

- Uớc lượng các yêu cầu tài nguyên khi lên kế hoạch bảo trì sửa chữa cho chu kỳ tiếp theo
- So sánh các phương thức thực hiện
- Khởi tạo các hoạt động ngăn ngừa và sửa lỗi
- Uớc lượng các yêu cầu tài nguyên như một cơ sở cho các đề xuất các dịch vụ bảo trì mới hoặc đã được điều chỉnh.

Chi phí chất lượng phần mềm

Như trong những phần trước, ở đây chúng ta chỉ quan tâm tới vấn đề bảo trì sửa lỗi. Giá thành chất lượng của bảo trì sửa lỗi được phân loại thành 6 lớp. Sau đây là định nghĩa cho mỗi lớp và ví dụ:

- Chi phí của việc ngăn ngừa: Chi phí ngăn ngừa lỗi, ví dụ như chi phí của việc xây dựng và đào tạo đội bảo trì, chi phí của các hoạt động ngăn ngừa và sửa lỗi.
- Chi phí của việc đánh giá chất lượng: chi phí của việc phát hiện lỗi, ví dụ như chi phí của việc xem xét lại các dịch vụ bảo trì được thực hiện bởi nhóm SQA, đội bên trong và các điều tra sự thỏa mãn của khách hàng.

- Chi phí của việc chuẩn bị và giám sát quản lý: Chi phí của các hoạt động quản lý được thực hiện để ngăn ngừa lỗi, ví dụ như chi phí của việc chuẩn bị kế hoạch bảo trì, việc tuyển đội bảo trì và bám sát hiệu năng bảo trì (follow-up of maintenance performance)
- Chi phí lỗi bên trong: Chi phí của lỗi phần mềm được đề xướng bởi đội bảo trì (ưu tiên cho các yêu sách nhận được từ phía khách hàng)
- Chi phí lỗi bên ngoài: Chi phí của các lỗi phần mềm được đề xướng bởi các yêu sách của khách hàng.
- Chi phí của lỗi quản lý: Chi phí lỗi phần mềm gây ra bởi các hoạt động quản lý hoặc tương tác, ví dụ như chi phí của các thiệt hại từ việc thiếu nhân viên bảo trì hoặc bảo trì không chính xác.

Sau khi xem lại các lớp định giá thành chất lượng phần mềm, được định nghĩa theo các lớp cũng như mô hình mở rộng, sẽ được thảo luận kỹ hơn ở chương 22.

Giá thành lỗi bên trong của các hoạt động bảo trì sửa lỗi phần mềm

Để định nghĩa giá thành lỗi bên trong, có hai chu kỳ bảo trì phải được quan tâm riêng biệt. Đó là: (a) chu kỳ đảm bảo (warranty) (thường từ 3 đến 12 tháng sau khi cài đặt phần mềm) và (b) chu kỳ các dịch vụ bảo trì được ký hợp đồng, bắt đầu ở thời điểm kết thúc chu kỳ đảm bảo. Vẫn đề ở đây yêu cầu một quyết định trong trường hợp nào nên quan tâm đến lỗi bên ngoài, chỉ sau khi đưa ra quyết định này thì việc định giá chất lượng mới được xác định và ước lượng. Những định nghĩa được đề xuất về chi phí lỗi bên ngoài và các tham số của chúng cho các dịch vụ sửa lỗi phần mềm và hỗ trợ người dùng dưới đây:

- **Cho sửa lỗi phần mềm:**
 - Tất cả chi phí của sửa lỗi phần mềm được đưa ra bởi người sử dụng trong suốt chu kỳ đảm bảo là chi phí chất lượng bên ngoài bởi vì chúng đề cập tới kết quả trực tiếp từ lỗi phát triển phần mềm, hơn nữa người phát triển có trách nhiệm cho việc sửa lỗi của họ trong suốt chu kỳ này.
 - Sửa lỗi phần mềm được thực hiện trong suốt chu kỳ bảo trì được ký kết, nó được xem như là một phần chính của dịch vụ, khi trách nhiệm của người phát triển đối với việc sửa lỗi bị giới hạn ở chu kỳ đảm bảo. Chi phí của các dịch vụ này được xem là chi phí các dịch vụ chính thức chứ không phải là chi phí chất lượng..
 - Trong suốt chu kỳ bảo trì kí kết, sau những nỗ lực của việc sửa lỗi ban đầu thì chỉ có chi phí cho việc sửa lỗi lại (re-correction) mới được xem là chi phí của lỗi bên ngoài.
- **Cho các dịch vụ hỗ trợ người dùng:**

- Trong suốt chu kỳ đảm bảo, các dịch vụ hỗ trợ người dùng được xem như là một phần của nỗ lực làm theo chỉ dẫn (instruction effort), và do vậy không nên quan tâm tới chi phí lỗi bên ngoài.
- Suốt chu kỳ bảo trì được kí kết, tất cả các kiểu dịch vụ hỗ trợ người dùng, dù là giải quyết một lỗi phần mềm được xác định hay bàn (consultation) về các tùy chọn ứng dụng, là tất cả các phần của dịch vụ chính thức, và chi phí của chúng không quan tâm tới chi phí lỗi bên ngoài.
- Trong cả hai chu kỳ bảo trì, một lỗi bên ngoài được định nghĩa như một trường hợp nơi sự bàn bạc thứ hai (second consultation) được yêu cầu sau khi sự bàn bạc ban đầu chứng minh là chính xác. Chi phí cung cấp cho sự bàn bạc thứ hai và kỹ hơn cho cùng một trường hợp được xem như là chi phí lỗi bên ngoài.

Như trường hợp tổng quát, thông tin chi phí bảo trì chất lượng, cùng với thông tin giám sát quản lý khác được mong đợi nhằm giúp đỡ cho việc quản lý đưa ra quyết định, được quan tâm như sau:

- Chỉ dẫn cho việc đầu tư trong việc cải thiện các dịch vụ bảo trì bằng cách đưa ra điểm yếu của chi phí chất lượng cực cao và điểm mạnh của chi phí chất lượng cực thấp.
- Phát triển phiên bản được cải tiến của phần mềm (trong trường hợp phần mềm được làm cho khách hàng) hoặc việc thay thế gói phần mềm đã được mua.

3.5 Đảm bảo chất lượng phần mềm của các yếu tố bên ngoài cùng tham gia

3.5.1 Những thành phần bên ngoài đóng góp vào dự án phần mềm

Những người tham gia một dự án phát triển phần mềm – tổ chức quan tâm đến hệ thống phần mềm (khách hàng) và tổ chức cam kết tiên hành phát triển (nhà thầu) – ngày nay thường không chỉ là những người tham dự trong dự án. Những người tham gia bên ngoài có liên quan đến dự án phát triển phần mềm đóng góp cho dự án nhưng không phải là nhà thầu, mà cũng không phải là những thành viên của nhà thầu. Sự đóng góp của họ cho dự án được cấu trúc thông qua những thỏa thuận với nhà thầu (nhà thầu phụ hay những người cung cấp của COTS software) hoặc thông qua những điều khoản của hợp đồng dự án, các phần của dự án sẽ được thực thi bởi chính khách hàng của họ. Dự án lớn hơn và phức tạp hơn, có ý nghĩa hơn có thể đúng mà những người tham gia bên ngoài được yêu cầu, phần lớn công việc được chuyển giao hoặc chia ra. Mục đích để chuyển hướng những người tham gia bên ngoài dựa vào một số nhân tố, xác định khoảng cách từ kinh tế đến kỹ thuật và đến những cá nhân liên quan (to personnel – related interests), và mang lại một sự gia tăng mối quan tâm trong sự phân phối của công việc liên quan trong việc hoàn thành những dự án phức tạp.

Những người tham gia bên ngoài có thể được phân chia thành 3 nhóm chính:

- Subcontractors(những nhà thầu phụ, hiện nay được gọi là những tổ chức “outsourcing”) họ cam kết thực hiện các thành phần của 1 dự án, lớn hay nhỏ, tùy theo từng trường hợp. Những nhà thầu phụ thường đưa ra hợp đồng ở mức tối thiểu một trong những lợi ích: khả năng huy động nhân viên, ý kiến về mặt chuyên môn đặc biệt hoặc giá thấp.
- Những nhà cung cấp COTS software và những module phần mềm sử dụng lại. Lợi ích của sự tích hợp các thành phần đó là rất rõ ràng, sự xác định khoảnh cách từ kế hoạch làm việc và giảm bớt giá thành đến chất lượng. Một sự mong đợi mà sự tích hợp của những thành phần sẵn sàng để dùng sẽ được lưu trữ trong những mã nguồn phát triển, một kế hoạch làm việc ngắn hơn và phần mềm chất lượng cao hơn. Phần mềm chất lượng cao hơn thì được chờ đợi nhưng những thành phần đã được kiểm tra và được sửa chữa bởi những người phát triển, tốt như được sửa chữa theo những thiếu sót được xác định bởi khách hàng xem trước. Các tính chất của COTS software và các vấn đề chất lượng liên quan đến sử dụng chúng đã được nhận định bởi Basili và Boehm (2001).
- Khách hàng, bản thân họ như là người tham gia thực hiện dự án. Điều này khá chung để mỗi khách hàng thực hiện các phần của dự án: để áp dụng những chuyên môn đặc biệt của những khách hàng, đáp ứng cho giao dịch hoặc những yêu cầu bảo mật khác, giữ lại những nhân viên phát triển nội bộ đang sử dụng, ngăn ngừa những vấn đề bảo mật trong tương lai.v.v. Trường hợp này có những hạn chế trong những điều khoản của quan hệ khách hàng-người cung cấp cần thiết để thực hiện thành công một dự án. Vì thế, chắc chắn trường hợp này đã trở thành 1 thành phần chuẩn của nhiều dự án phát triển phần mềm và những quan hệ bằng hợp đồng.

3.5.2 Rủi ro và lợi ích của giới thiệu người tham dự ngoài.

Rủi ro chủ yếu tới chất lượng dự án liên quan với người giới thiệu tham gia từ bên ngoài trong cơ cấu của dự án là như sau:

- Sự trì hoãn hoàn thành của dự án

Trong đó trường hợp người tham gia ở bên ngoài cung cấp chậm những thành phần cho hệ thống phần mềm, dự án nói chung sẽ bị hoãn lại. Sự chậm trễ này chủ yếu là do nhà thầu phụ và khách hàng gây ra chứ không phải do các nhà cung cấp phần mềm có sẵn (COTS).

Trong nhiều trường hợp, trách nhiệm kiểm soát sự phát triển phần mềm của nhà thầu phụ và khách hàng không cao, do đó gây ra tình trạng chậm chạp, trì hoãn và không có thời gian cho sự thay đổi cũng như thời gian cần thiết để tổ chức lại gây ảnh hưởng tiêu cực với dự án.

- Các phần dự án chất lượng thấp được cung cấp bởi các thành viên bên ngoài

Các vấn đề về chất lượng có thể phân loại như

(a) sai sót: cao hơn so với số lượng sai sót mong đợi, thường cao hơn nhiều so với mong đợi

(b) Coding và tài liệu không chuẩn : sự vi phạm của phong cách và cấu trúc trong việc xây dựng và các thủ tục(theo giả thuyết như đã án định trong bát cứ hợp đồng nào). Phần mềm chất lượng thấp và không chuẩn sẽ gây ra khó khăn trong giai đoạn kiểm thử và sau đó trong giai đoạn bảo trì. Việc yêu cầu thêm thời gian để kiểm tra và chỉnh sửa chất lượng phần mềm chất lượng thấp có thể gây ra sự chậm trễ trong dự án đặc biệt trong trường hợp khi các thành viên bên ngoài hoàn thành nhiệm vụ của họ đúng thời hạn.

- Khó khăn về bảo trì trong tương lai.

Thực tế một số tổ chức tham gia việc phát triển nhưng chỉ một trong số họ, nhà thầu, là người trực tiếp gây nên 2 khó khăn trong việc bảo trì:

- Nhà thầu có thể đối mặt với việc các coding và tài liệu không hoàn thành và/hoặc không đúng tiêu chuẩn từ các thành viên bên ngoài, gây ra sự kém chất lượng trong dịch vụ bảo trì của nhóm bảo trì và nhà thầu sẽ tốn chi phí cao hơn.
- Các dịch vụ bảo trì được cung cấp bởi nhiều hơn một tổ chức, có thể nhà thầu phụ, nhà cung cấp phần mềm có sẵn COTS và các bộ phận phát triển của khách hàng.Khi mỗi phần này bị hạn chế khả năng đáp ứng, khách hàng buộc phải tìm kiếm người chịu trách nhiệm cho các lỗi cụ thể của phần mềm mỗi khi các lỗi này được phát hiện.

- Mất sự kiểm soát các bộ phận của dự án

Dù có ý hay không có ý,sự kiểm soát việc phát triển phần mềm của thành viên bên ngoài có thể tạo ra một bức tranh lạc quan không thực tế về tình trạng của dự án. Sự trao đổi với nhóm các thành viên bên ngoài có thể làm gián đoạn tới một vài tuần,gây cản trở việc đánh giá tiến độ của dự án..Kết quả là,cảnh báo về khó khăn trong phát triển, thiếu đội ngũ nhân viên và nhiều vấn đề khác đến muộn với các nhà thầu.

Nhận thức được trước các khó khăn này, nhà thầu phải xem xét kết hợp lợi ích và rủi ro được đưa ra bởi các thành viên bên ngoài trong một dự án.

3.5.3 Những mục tiêu đảm bảo chất lượng về sự đóng góp người tham gia bên ngoài

Những mục tiêu nào thu được bằng việc áp dụng công cụ SQA được cung cấp bởi các thành viên bên ngoài? Những mục tiêu dưới đây có thể thu được trực tiếp từ việc liệt kê các rủi ro đã đề cập ở trên:

- Để tránh trì hoãn hoàn thành nhiệm vụ và để đảm bảo cảnh báo sớm để tính trước sự trì hoãn.
- Để đảm bảo mức độ chất lượng có thể chấp nhận được của bộ phận triển khai và nhận cảnh báo sớm của phạm vi chất lượng yêu cầu.
- Để đảm bảo đủ tài liệu phục vụ cho nhóm bảo trì.

- Để đảm bảo liên tục, toàn diện và đáng tin cậy kiểm soát việc thực hiện người tham gia bên ngoài.

3.5.4 Các công cụ đảm bảo chất lượng những đóng góp của các thành viên đóng góp bên ngoài.

Chúng ta mong muốn các thành viên đóng góp bên ngoài thực hiện các phương thức SQA của chính họ, bao gồm các công cụ cần thiết để các sản phẩm phần mềm và các dịch vụ của họ đạt được tới mức chất lượng có thể chấp nhận được. Các công cụ được đề cập tới ở đây là những thứ mà các nhà thầu có thể áp dụng cho các thành viên đóng góp bên ngoài. Trong mục đích này, vấn đề chất lượng và thời gian là quan trọng nhất được xác định.

Các công cụ chính được áp dụng trước và trong suốt quá trình kết hợp các thành viên đóng góp bên ngoài trong một dự án phát triển phần mềm được liệt kê bên dưới.

- xem xét lại tài liệu yêu cầu.
- Đánh giá các tiêu chuẩn chọn lựa liên quan đến các thành viên đóng góp bên ngoài.
- Thành lập ủy ban điều khiển gia nhập và kết hợp của dự án.
- Sự đóng góp trong sự xem xét thiết kế.
- Sự đóng góp trong kiểm tra phần mềm.
- Cách trình bày các thủ tục đặc biệt
- Xác định các team leader của các nhà cung cấp và các thành viên.
- Chuẩn bị các báo cáo tiến trình phát triển của các hoạt động phát triển dự án.
- Xem xét lại các giao phẩm (các tài liệu) và acceptance tests

Chương 4: Kiểm thử phần mềm

4.1 Định nghĩa và mục tiêu

4.1.1 Định nghĩa

Kiểm thử phần mềm có nhiều định nghĩa khác nhau để xuất bởi nhiều tổ chức hay cá nhân khác nhau. Một số định nghĩa nổi bật:

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

Kiểm thử phần mềm là quá trình thực thi một chương trình với mục đích tìm lỗi. (Theo “*The Art of Software Testing*” – *Nghệ thuật kiểm thử phần mềm*).

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

Có thể định nghĩa một cách dễ hiểu như sau: *Kiểm thử phần mềm là quá trình thực thi một hệ thống phần mềm để xác định xem phần mềm có đúng với đặc tả không và môi trường hoạt động có đúng yêu cầu không.*

Mục tiêu của kiểm thử:

Các mục tiêu trực tiếp

- Xác định và phát hiện nhiều lỗi nhất có thể trong phần mềm được kiểm thử
- Sau khi sửa chữa các lỗi đã xác định và kiểm tra lại, làm cho phần mềm đã được kiểm thử đến một mức độ chấp nhận được về chất lượng.
- Thực hiện các yêu cầu kiểm thử cần thiết một cách hiệu quả và có hiệu quả, trong phạm vi ngân sách và thời gian cho phép.

Các mục tiêu gián tiếp

Biên dịch một bản ghi về các lỗi phần mềm để sử dụng trong công tác phòng chống lỗi (bằng các hành động khắc phục và ngăn ngừa).

4.1.2 Các mức độ kiểm thử

Tuỳ thuộc vào mục tiêu của kiểm thử, ta chia ra thành các mức như sau:

Mức 0: testing và debuging là giống nhau

Mức 1: Mục tiêu của kiểm thử là để chỉ ra phần mềm hoạt động

Mức 2: Mục tiêu của kiểm thử là để chỉ ra phần mềm không hoạt động

Mức 3: Mục tiêu của kiểm thử là để giảm các rủi ro khi sử dụng phần mềm

Mức 4: Nhằm trợ giúp các chuyên gia CNTT phát triển các phần mềm có chất lượng cao hơn.

a. Mức 0

Testing và debuging là một

Mức này thường được thực hiện bởi các sinh viên trong các môn học lập trình. Sinh viên viết chương trình, chạy với vài đầu vào, và debug lỗi nếu có.

Mức này không phân biệt giữa hành vi không đúng và lỗi bên trong chương trình.

Mức này chỉ giúp ích đôi chút trong việc phát triển phần mềm chính xác

b. Mức 1

Nhằm để chứng minh tính đúng đắn.

Một cách phát triển tự nhiên từ mức 0, tuy nhiên ta không thể chứng minh tính đúng đắn của phần mềm. Giả sử ta chạy một tập test và không phát hiện ra lỗi nào. Vậy, kết luận là gì? Liệu ta có thể giả thiết là phần mềm chạy tốt hay tập test kém? Vì không thể chứng minh tính đúng đắn, việc kiểm thử không có giới hạn dừng cố định, cũng như không có một kỹ thuật test hình thức (formal) nào. Nếu người quản lý hỏi: còn phải thực thi bao nhiêu test nữa? Ta không có cách nào trả lời chính xác câu hỏi này.

c. Mức 2

Nhằm để chỉ ra lỗi. Tìm lỗi là một mục tiêu rõ ràng, nhưng mang tính tiêu cực. Tester có thể vui vẻ khi tìm ra lỗi, nhưng developers thì không muốn vậy - họ muốn phần mềm chạy (mức 1 là suy nghĩ tự nhiên của developers). Do đó, mức 2 đặt tester và developers vào quan hệ đối đầu. Điều này có thể ảnh hưởng xấu tới cả nhóm. Ngoài ra, câu hỏi đặt ra là nếu không tìm thấy lỗi nào thì sao? Ta có thể kết luận phần mềm chạy tốt? hoặc việc kiểm thử còn yếu?

d. Mức 3

Kiểm thử có thể chỉ ra lỗi khi nó xuất hiện, nhưng không thể chứng tỏ phần mềm không có lỗi. Nghĩa là, ta phải chấp nhận mỗi khi sử dụng phần mềm, ta có nguy cơ gặp lỗi. Nguy cơ này có thể nhỏ, và không gây hậu quả gì, hoặc nguy cơ có thể lớn và gây ra

thảm họa. Điều này chỉ ra rằng, toàn đội phát triển phần mềm có chung mục tiêu - giảm nguy cơ gặp lỗi khi sử dụng phần mềm. Ở mức 3, cả tester và developer làm việc cùng nhau để giảm nguy cơ gặp lỗi.

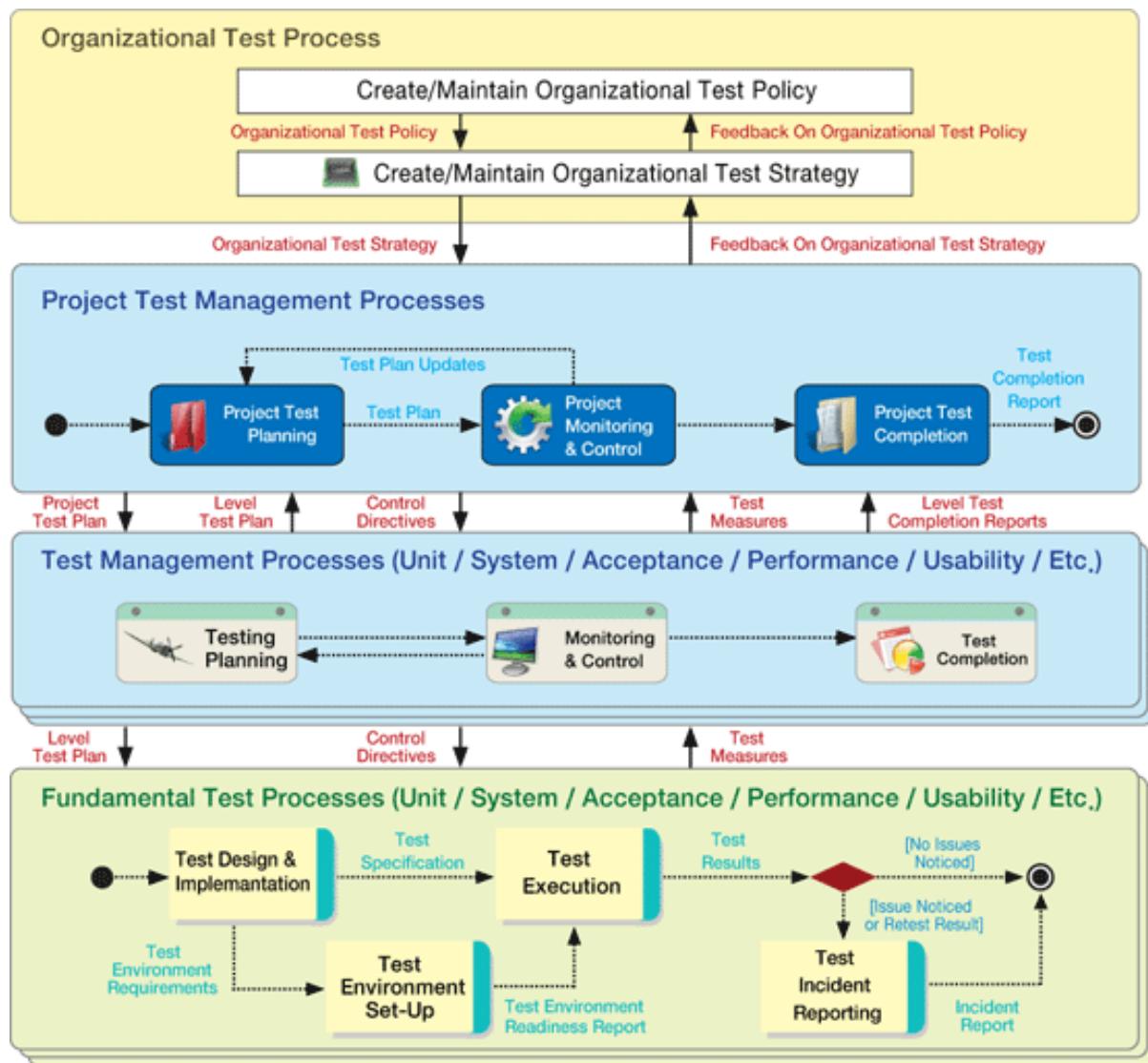
e. Mức 4

Khi tester và developers có chung mục tiêu (mức 3), tổ chức có thể chuyển sang mức 4. Kiểm thử nhằm mục tiêu tăng chất lượng. Có nhiều cách để tăng chất lượng, trong đó tạo ra test có thể dẫn tới lỗi chỉ là một. Ở mức này, kỹ sư kiểm thử có thể trở thành trưởng nhóm kỹ thuật của dự án. Họ có nhiệm vụ đánh giá, cải thiện chất lượng phần mềm, và sự thẩm định của họ sẽ trợ giúp cho developers. Ví dụ như ta có 1 chương trình spell checker. Ta thường nghĩ nhiệm vụ chính của nó là để tìm những từ sai chính tả (đánh vần sai), nhưng thực tế, mục tiêu cao nhất của nó là để tăng khả năng viết chính tả của chúng ta. Mỗi khi spell checker tìm ra một từ sai chính tả, ta có cơ hội để học cách viết đúng. Do vậy, spell checker là “chuyên gia” về chất lượng viết chính tả. Một cách tương tự, mức 4 hướng tới mục tiêu kiểm thử để tăng khả năng của developers trong việc phát triển các phần mềm chất lượng cao. Testers có thể đào tạo developers.

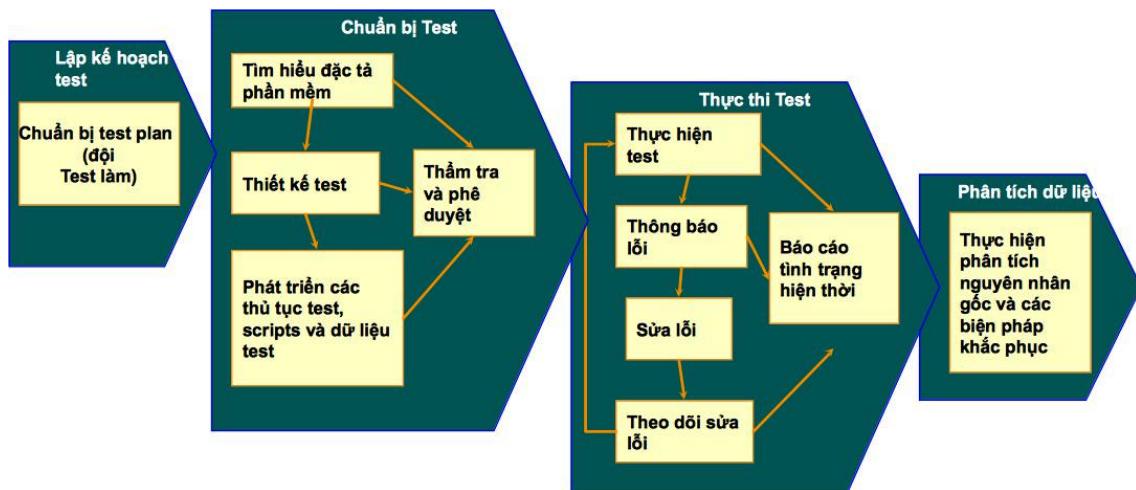
4.2 Quy trình kiểm thử

4.2.1 Quy trình

Tùy vào từng tổ chức, hệ thống, ngữ cảnh, mức độ rủi ro của phần mềm mà quy trình kiểm thử phần mềm có thể gồm nhiều bước khác nhau. Nhưng nhìn chung mọi quy trình kiểm thử đều có những bước cơ bản như dưới đây:



Hình 4-1: Quy trình kiểm thử theo ISO/IEC 29119



Hình 4-2: Quy trình kiểm thử chung

Theo đó một quy trình kiểm thử phần mềm cơ bản gồm 4 giai đoạn:

- **Lập kế hoạch kiểm thử:** Nhiệm vụ quan trọng trong phần lập kế hoạch kiểm thử là xác định được các thành phần sau:
 - Tìm hiểu nghiệp vụ của hệ thống phải kiểm thử.
 - Xây dựng kịch bản kiểm thử.
 - Chuẩn bị dữ liệu kiểm thử.
 - Xem xét phê duyệt các tài liệu kiểm thử.
- **Chuẩn bị kiểm thử:**
 - Thực hiện kiểm thử dựa trên các kịch bản kiểm thử, test case, thủ tục, dữ liệu có sẵn từ bước chuẩn bị kiểm thử.
 - Thực hiện quá trình quản lý lỗi: báo lỗi, sửa lỗi.
- **Thực thi kiểm thử:**
 - Lập báo cáo kiểm thử.
 - Phân tích nguyên nhân và đề xuất các hành động khắc phục.

4.2.2 Input/Output cho test

Input:

- Yêu cầu của khách hàng và tiêu chí chấp nhận

- Yêu cầu về thay đổi
- Đặc tả yêu cầu phần mềm (Software Requirement Specification SRS)
- Tài liệu thiết kế
- Chương trình (Modules)

Output:

- Tài liệu test: kế hoạch test, test cases và procedures, Test script, dữ liệu Test
- Danh sách lỗi
- Mô tả thực hiện test
- Báo cáo phân tích lỗi

4.2.3 Quản lý lỗi

Thông thường, lỗi phát hiện sẽ được quản lý trong các hệ thống quản lý lỗi (ví dụ RedMine, Jira,...). Một số lưu ý khi thực hiện ghi nhận lỗi:

- Lưu ý là 1 dự án có nhiều pha phát triển khác nhau, có nhiều chức năng trong hệ thống khác nhau => lúc log lỗi phải nói rõ là lỗi gì, thuộc chức năng nào?
- Mục đích viết lỗi là để cho Developer/Leader đọc và hiểu được. Một ngày dev có thể nhận được nhiều bugs, và phải thực hiện nhiều việc => cần viết ngắn gọn, rõ ràng, và có thể dễ dàng theo dõi chỉ từ subjects.
- Phải dùng từ ngữ thông nhất tránh gây hiểu lầm và dễ tìm kiếm nếu cần. Subject: cần chỉ rõ chức năng có lỗi, nội dung cụ thể của lỗi, Description: Mô tả rõ lỗi và đường đi tới lỗi

Ví dụ 1: lỗi đơn giản

The screenshot shows a Redmine issue tracking interface. The URL in the address bar is <http://demo.redmine.org/projects/mair/Issues/new>. The page title is "New issue - Mair - Redmine demo". The form fields are as follows:

- Tracker ***: Bug
- Subject ***: Trang tìm kiếm - Lỗi chính tả trường "Về"
- Description**: Trong trang tìm kiếm. Theo UI, trường "Đến" phải là "Về"
- Status ***: New
- Priority ***: Low
- Assignee**: Do Ngoc

Ví dụ 2: lỗi điển hình và có liên quan tới dữ liệu

- Cần nhập rõ trình tự thực hiện và dữ liệu dẫn tới kết quả sai
- Viết rõ kết quả thực hiện của chương trình
- Viết rõ kết quả mong muốn theo đặc tả

Tracker * Bug

Subject * Trang tìm kiếm - Chưa kiểm tra ngày về sớm hơn ngày đi

Description

Step 1: Nhập ngày đi là Tháng 12
Step 2: Nhập ngày về là Tháng 12 hoặc Tháng 6 (để ngày về sớm hơn ngày đi)
Output:
Màn hình Kết quả tìm kiếm hiện thị "Xin lỗi đã hết chỗ!"
Expected output:
Màn hình Kết quả tìm kiếm hiển thị "Ngày nhập không hợp lệ, ngày về phải cách ngày đi ít nhất 1 năm. Hãy tìm lại!"

Status * New

Priority * Normal

Assignee Do Ngoc

Parent task

Start date 2015-03-24

Due date

Estimated time Hours

Ví dụ 3: cách log lỗi chưa tốt

Subject: không chỉ rõ điểm gây lỗi, mô tả lỗi chung chung

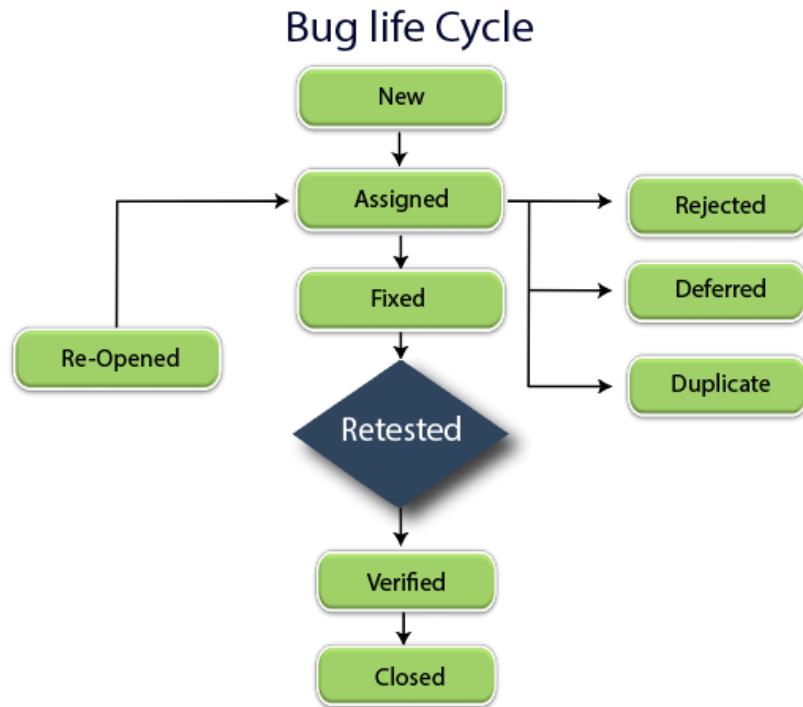
Description: không mô tả rõ ràng vị trí gây lỗi

Hậu quả:

- developer không xác định được lỗi ở đâu (nhất là khi họ đang phải làm nhiều việc cùng lúc)
- các mô tả lỗi có subject giống giống nhau (trong khi có thể có hàng trăm, hàng nghìn lỗi trong hệ thống) nên dễ bị hiểu lầm là log trùng

Vòng đời lỗi

Sau khi tester tạo mới 1 lỗi, các thành viên trong đội dự án có thể thực hiện chỉnh sửa/tù chối và đổi trạng thái của lỗi. Một vòng trạng thái lỗi điển hình cho ở hình dưới, trong đó lập trình viên được gán lỗi có thể chấp nhận sửa lỗi hoặc từ chối nếu thấy đó không phải là lỗi; kiểm thử viên kiểm tra lại lỗi sau khi sửa và có thể chấp nhận để đóng lỗi hoặc yêu cầu sửa lại nếu việc sửa lỗi của lập trình viên không đáp ứng yêu cầu.



○ Hình 4-3: Vòng đời quản lý lỗi

4.3 Kế hoạch kiểm thử

Một số câu hỏi cần trả lời trong quá trình lập kế hoạch:

- Test cái gì ?
- Ai thực hiện test?
- Thực hiện test ở chỗ nào ?
- Khi nào kết thúc test?

-Test cái gì?

Quá trình kiểm thử sẽ giới thiệu đầy đủ kế hoạch kiểm thử, từ test đơn vị (unit test) đến test tích hợp (integration test) và test hệ thống (system test). Các yếu tố quyết định sẽ liên quan đến:

Các module nào sẽ được unit test

Cách tích hợp cần được test.

Xác định rõ mức độ ưu tiên để phân bổ tài nguyên test tới ứng dụng hệ thống phần mềm riêng rẽ.

+ *Đánh giá các unit, integration và các application.*

Các phương thức đánh giá cho các unit(module), integration và các ứng dụng để xác định rõ mức độ ưu tiên của chúng trong kế hoạch testing được dựa trên 2 yếu tố :

Yếu tố A: Mức thiệt hại nghiêm trọng. Độ nghiêm trọng trong trường hợp module hoặc ứng dụng thất bại.

Yếu tố B: Mức rủi ro của phần mềm. Mức độ rủi ro là xác suất xuất hiện thất bại. Để xác định mức rủi ro của một module, unit, integration hoặc ứng dụng, các vấn đề ảnh hưởng của rủi ro cần thẩm tra kỹ. Các vấn đề này có thể được phân loại như các vấn đề module/application và vấn đề người lập trình.

Các vấn đề ảnh hưởng tới mức rủi ro của phần mềm.

Các vấn đề module/application

- Độ lớn
- Độ phức tạp và độ khó.
- Tỉ lệ của phần mềm gốc (với tỉ lệ phần mềm được dùng lại)

Vấn đề người lập trình

- Tính chuyên nghiệp
- Kinh nghiệm với các vấn đề của module cụ thể
- Tính khả dụng của hỗ trợ chuyên nghiệp (việc sao lưu các kiến thức và kinh nghiệm)
- Sự hiểu biết của người lập trình và năng lực về khả năng đánh giá .

- Test được thực hiện bởi ai ?

Những người thực thi kiểm thử được xác định ở giai đoạn lập kế hoạch.

- Integration tests, đặc biệt là unit tests, thường được thực hiện bởi nhóm phát triển phần mềm.
- System test thường xuyên được thực hiện bởi một nhóm test độc lập (trong nội bộ nhóm test hoặc có vấn đề bên ngoài nhóm test).
- Trong trường hợp hệ thống phần mềm lớn, nhiều hơn một nhóm testing có thể được dùng để thực hiện các test hệ thống.
- Testing bởi nhóm phát triển khác. Mỗi nhóm phát triển sẽ coi như nhóm testing của dự án được phát triển cho các nhóm khác.

- Test được thực hiện ở đâu ?

Test đơn vị và test tích hợp được thực hiện ở phía người phát triển phần mềm. Khi test hệ thống: chúng nên thực hiện ở phía người phát triển hay phía khách hàng? Nếu test hệ thống là để được thực hiện bởi các tư vấn test bên ngoài, một lựa chọn thứ 3 phát sinh: phía người tư vấn. Lựa chọn phụ thuộc vào môi trường test hay môi trường hệ thống.

Tuy nhiên, môi trường máy tính ở phía khác hàng khác có thể khác với phía nhà phát triển. Trong trường hợp này, để giảm những lo ngại của khách hàng, hệ thống sẽ được cài đặt và kiểm thử ở phía khách hàng và không cần test chấp nhận nữa.

-Khi nào dùng test?

Quyết định khi nào quá trình kiểm thử kết thúc có ý nghĩa quan trọng nhất là với test hệ thống. Có nhiều hướng lựa chọn để có thể dùng test, mỗi lựa chọn dựa trên tiêu chí khác nhau, ví dụ như:

- Hướng hoàn tất thực thi . Theo hướng này, testing được hoàn thành khi toàn bộ kế hoạch test đã được thực hiện xong và sạch lỗi, kết quả đạt được cho tất cả các yêu cầu test. Đây là hướng lý tưởng, khi mà ta không bị giới hạn bởi ngân sách và thời gian.

- Hướng áp dụng mô hình toán học. Ở hướng này, mô hình toán học được áp dụng để ước lượng tỉ lệ lỗi không bị phát hiện trên tỉ lệ lỗi được phát hiện. Testing sẽ hoàn thành khi tỉ lệ phát hiện lỗi giảm dưới mức chấp nhận được theo một chuẩn nào đó. Những bất lợi của hướng này là mô hình tính toán được lựa chọn có thể không đại diện hoàn toàn cho đặc trưng của dự án.

- Hướng các đội test độc lập. Hai đội test thực hiện test một cách độc lập. Bằng cách so sánh danh sách các lỗi được phát hiện của mỗi đội có thể đưa ra quyết định dừng quá trình test.

- Dừng khi hết tài nguyên. Với kiểu này việc dừng test sẽ xảy ra khi ngân sách hoặc thời gian được phân bổ cho testing đã hết.

4.4 Thiết kế test (test design)

Các sản phẩm của giai đoạn thiết kế test là :

- Thiết kế chi tiết và các Thủ tục cho mỗi việc test.
- Các tệp hoặc cơ sở dữ liệu của các test case.

Quá trình thiết kế test được tiến hành trên cơ sở của kế hoạch test phần mềm, lập tài liệu bởi STP (software test procedure). Các thủ tục test và cơ sở dữ liệu hay tệp test case có lập tài liệu trong một tài liệu “thủ tục test phần mềm” và “test case file” hoặc trong một tài liệu đơn gọi là “mô tả test phần mềm” (STD). Một mẫu (template) của STD được trình bày như sau:

Mẫu mô tả test phần mềm (STD)

- Phạm vi của test

1.1 Các gói phần mềm được test (name, version, và revision)

1.2 Các tài liệu cơ bản được cung cấp cho việc thiết kế test (tên và phiên bản của từng tài liệu)

- Môi trường test (của mỗi test)

2.1 Định danh test (các chi tiết test được viết tài liệu trong STP)

2.2 Mô tả chi tiết về hoạt động hệ thống và cấu hình phần cứng và yêu cầu chuyển đổi thiết lập test.

2.3 Hướng dẫn tải phần mềm.

- Tiến trình test

3.1 Hướng dẫn cho đầu vào, mô tả chi tiết mỗi bước của tiến trình đầu vào.

3.2 Dữ liệu được ghi chép trong suốt quá trình test.

- Test cases (với mỗi case)

4.1 Chi tiết định danh test case

4.2 Dữ liệu Đầu vào và thiết lập hệ thống.

4.3 Kết quả trung gian mong đợi (nếu có)

4.4 Các kết quả mong đợi (Số, thông điệp, sự kích hoạt thiết bị,...)

Hành động trong trường hợp chương trình lỗi / kết thúc

Thủ tục được áp dụng theo bản tóm tắt kết quả test.

Ví dụ một bản test cases:

Mã module		QLTKH		
Yêu cầu kiểm thử		Kiểm thử các chức năng trên màn hình quản lý thông tin khách hàng		
Người kiểm thử		Số ca kiểm thử		
Pass	Fail	Untested		
19	7	0		26
Mã	Mô tả	Quy trình ca kiểm thử	Dữ liệu kiểm thử	Kết quả mong muốn
Giao diện người dùng		Dữ liệu kiểm thử		
1	Kiểm thử giao diện màn hình quản lý danh mục cho vay	1. Chạy ứng dụng 2. Nhấp chuột vào mục "Quản Lý Khách Hàng"		<ul style="list-style-type: none"> Hiển thị màn hình "QUẢN LÝ THÔNG TIN KHÁCH HÀNG" với các thành phần sau: <ul style="list-style-type: none"> Bảng danh mục thông tin gồm các trường thông tin: ô tích chọn, Số lưu ký, Họ và tên, Giới tính, Ngày sinh, Số CMND, Địa chỉ, Số điện thoại, Email, Trang thái 4 nút chức năng: Thêm, Sửa, Tra cứu, Phong tỏa/Giải tỏa. Thứ tự tab: nút "Thêm" → "Sửa" → "Tra cứu" → Phong tỏa/Giải tỏa → datagridview danh mục thông tin khách hàng
2	Kiểm thử sự di chuyển của các nút chức năng	1. Tại màn hình "Quản lý thông tin khách hàng", nhấp chuột vào nút "Thêm"		<ul style="list-style-type: none"> Hiển thị màn hình "Thêm mới tài khoản".
3	Kiểm thử sự di chuyển của các nút chức năng	1. Tại màn hình "Thêm mới tài khoản", nhấp chuột vào nút "Hủy"		<ul style="list-style-type: none"> Đóng màn hình "Thêm mới tài khoản". Trở về màn hình "Quản lý thông tin khách hàng".
4	Kiểm thử sự di chuyển của các nút chức năng	1. Tại màn hình "Quản lý thông tin khách hàng", tích chọn một dòng trong danh mục quản lý thông tin khách hàng 2. Nhấp chuột vào nút "Sửa"		<ul style="list-style-type: none"> Các trường thông tin Họ và tên, Giới tính, Ngày sinh, Số CMND, Địa chỉ, Số điện thoại, Email có thể sửa được

Chương 5: Kỹ thuật kiểm thử hộp đen và hộp trắng

5.1 Các kỹ thuật kiểm thử hộp đen

Black-box testing là phương pháp kiểm thử mà không cần biết cài đặt của chương trình.

Cần có một bản chương trình chạy được và đặc tả

Test cases được công thức hóa là một cặp

ví dụ, (input, output mong muốn)

Một số kỹ thuật thiết kế test:

phân lớp tương đương, test biên, phân loại, kiểm thử theo cặp

Quan trọng trong công nghiệp

5.1.1 Phân lớp tương đương

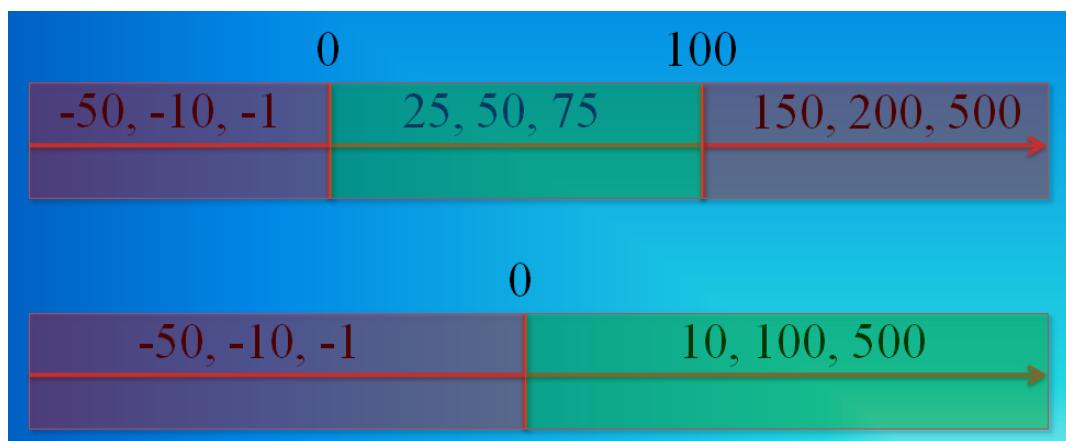
Phân lớp tương đương là một phương pháp kiểm thử hộp đen chia miền đầu vào của chương trình thành các lớp dữ liệu, từ đó suy dẫn ra các ca kiểm thử. Lớp tương đương biểu thị cho tập các trạng thái hợp lệ hay không hợp lệ đối với điều kiện vào.

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

Bước 1: 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 2 hay nhiều nhóm.

Ví dụ về các lớp tương đương:



Từ các lớp tương đương trên ta có bảng liệt kê các lớp tương đương tương ứng:

Điều kiện đầu vào	Các lớp tương đương hợp lệ	Các lớp tương đương không hợp lệ
x lớn hơn 0 và nhỏ hơn 100.	25, 50, 75	-50, -10, -1, 150, 200, 500
x lớn hơn 0.	10, 100, 500	-50, -10, -1

Để xác định các lớp tương đương có thể áp dụng các nguyên tắc dưới đây:

- **Một vùng giá trị**

Điều kiện đầu vào là **một vùng giá trị**.

Ví dụ: “Giá trị x chỉ có thể dao động từ 0 đến 100”.

Chúng ta sẽ xác định được 1 lớp tương đương hợp lệ là: **0 <=x <= 100**

và 2 lớp tương đương không hợp lệ là: **x < 0** và **x > 100**.

- **Một số giá trị**

Điều kiện đầu vào được mô tả là **một số giá trị**.

Ví dụ: “Chỉ một đến sáu người có thể được đăng ký”.

Chúng ta sẽ xác định 1 lớp tương đương hợp lệ là: “**Có từ một đến sáu người đăng ký**”

Và 2 lớp tương đương không hợp lệ là: “**không người nào đăng ký**” và “**nhiều hơn sáu người đăng ký**”.

- **Một tập hợp các giá trị**

Điều kiện đầu vào được mô tả là **một tập các giá trị**. Ta sẽ xác định mỗi giá trị trong tập đó là một lớp tương đương hợp lệ.

Ví dụ: “Các loại xe được đăng ký là xe bus, xe khách, xe tải, xe taxi và xe máy”.

Đối với mỗi loại xe ở trên chúng ta sẽ xác định là một lớp tương đương hợp lệ (ở đây chúng ta sẽ có 5 lớp tương đương hợp lệ) và một lớp tương đương không hợp lệ là một loại xe khác các loại xe nêu trên ví dụ như “xe đạp”.

- **Điều kiện đặc biệt**

Điều kiện đầu vào được mô tả là **một điều kiện đặc biệt**.

Ví dụ: “Ký tự đầu tiên phải là ký tự chữ”

Chúng ta sẽ xác định được một lớp tương đương hợp lệ là “*ký tự đầu tiên là ký tự chữ*” và một lớp tương đương không hợp lệ là “*không phải là ký tự chữ (có thể là số hoặc ký tự đặc biệt)*”.

Bước 2: Xác định các ca kiểm thử

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

- ✓ Gán số thứ tự cho mỗi lớp tương đương đã xác định.
- ✓ Viết test case cho các giá trị nằm trong các lớp tương đương hợp lệ.

Viết test case cho các lớp tương đương không hợp lệ.

Ví dụ:

Cho một chức năng đăng ký đăng nhập gồm 2 trường dạng text là User và Password. Trong trường Password chỉ cho nhập số ký tự trong khoảng 8 đến 30 ký tự với các mức bảo mật như trong bảng

Số ký tự	Mức độ bảo mật
Từ 8 đến 12	Yếu
Từ 13 đến 18	Trung bình
Từ 19 đến 24	Khá
Từ 25 đến 30	Tốt

Ta sẽ chia:

- 4 vùng hợp lệ tương đương với 4 mức độ bảo mật
- 3 vùng không hợp lệ là số ký tự lớn hơn 30, số ký tự nhỏ hơn 8 và trường Password để trống.

Gọi x là số ký tự ta có bảng phân vùng:

Vùng tương đương	x

Không hợp lệ	$x = 0$
Không hợp lệ	$0 \leq x \leq 7$
Hợp lệ	$8 \leq x \leq 12$
Hợp lệ	$13 \leq x \leq 18$
Hợp lệ	$19 \leq x \leq 24$
Hợp lệ	$25 \leq x \leq 30$
Không hợp lệ	$x > 30$

Các ca kiểm thử được sinh ra từ các vùng tương đương

STT	Mô tả	Dữ liệu kiểm thử	Đầu ra mong muốn
1	Trường Password để trống	Không nhập gì	Hiển thông báo “Mật khẩu không được để trống”
2	Số kí tự trong [Password] từ 1 đến 7	Password: Sqa	Hiển thông báo “Mật khẩu dưới 8 ký tự, mời nhập lại”
3	Số kí tự trong [Password] từ 8 đến 12	Password: Sqa12345	Hiển ra dòng chữ “Mức độ bảo mật: yếu”
4	Số kí tự trong [Password] từ 13 đến 18	Password: SQA12345789	Hiển ra dòng chữ “Mức độ bảo mật: trung bình”
5	Số kí tự trong [Password] từ 19 đến 24	Password: Sqa12345678987654321	Hiển ra dòng chữ “Mức độ bảo mật: khá”
6	Số kí tự trong [Password] từ 25 đến 30 Ấn nút đăng ký	Password: Sqa01234567899876543210sqA	Hiển ra dòng chữ “Mức độ bảo mật: tốt” ngay dưới [Password]
7	Số kí tự trong [Password] > 30 Ấn nút đăng ký	Password: Sqa012345678998765432100123456789	Hiển thông báo “Mật khẩu trên 30 ký tự, mời nhập lại” khi ấn nút đăng ký

5.1.2 Kiểm thử biên

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

- Phân tích giá trị biên không lựa chọn phần tử bất kỳ nào trong 1 lớp tương đương là điển hình, mà nó yêu cầu là 1 hay nhiều phần tử được lựa chọn như vậy mà mỗi cạnh của lớp tương đương đó chính là đối tượng kiểm tra.
- Ngoài việc chỉ tập trung chú ý vào các trạng thái đầu vào (không gian đầu vào), các ca kiểm thử cũng nhận được bằng việc xem xét không gian kết quả (các lớp tương đương đầu ra).

Những cách phân tích giá trị biên

- **Một vùng giá trị**

Điều kiện đầu vào được mô tả là **một vùng giá trị**.

Ta viết test case cho giá trị hợp lệ là điểm bắt đầu, kết thúc của vùng giá trị này. Test case cho giá trị không hợp lệ là giá trị ở phía ngoài của 2 điểm này.

Ví dụ: “Giá trị x dao động từ 0 đến 100”

Ta sẽ viết test case cho các trường hợp: **0, 100, -1, 100**.

- **Một số giá trị**

Điều kiện đầu vào được mô tả là **một số giá trị**.

Viết test case cho giá trị hợp lệ là số nhỏ nhất, lớn nhất của các giá trị này. Test case cho giá trị không hợp lệ là giá trị ở phía ngoài của 2 số này.

Ví dụ: “Chỉ một đến sáu người có thể đăng ký”

Ta cần viết test case cho các trường hợp: **1, 6, 0** và **7**.

- **Quan tâm đến điều kiện xuất (kết quả)**

Sử dụng cách 1 và 2 ở trên áp dụng cho điều kiện xuất.

Ví dụ: “Màn hình hiển thị tóm tắt các tin tức mới nhất và hiển thị được nhiều nhất 4 tin”.

Ta viết test case cho các kết quả hợp lệ là: **0, 1 và 4 tin**.

Test case cho kết quả không hợp lệ là **5 tin**.

- **Danh sách có thứ tự**

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

Cuối cùng, tùy vào các trường hợp khác nữa, chúng ta cũng cần sự tư duy và kinh nghiệm của mình để tìm ra các biên cần test.

5.1.3 Bảng quyết định

- Miêu tả các qui tắc nghiệp vụ phức tạp mà phần mềm phải thực hiện dưới dạng dễ đọc và dễ kiểm soát
- Ví dụ 1 chức năng nhỏ của công ty bảo hiểm : khuyến mãi cho những chủ xe 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	60	25	50	0
Discount (\$)				

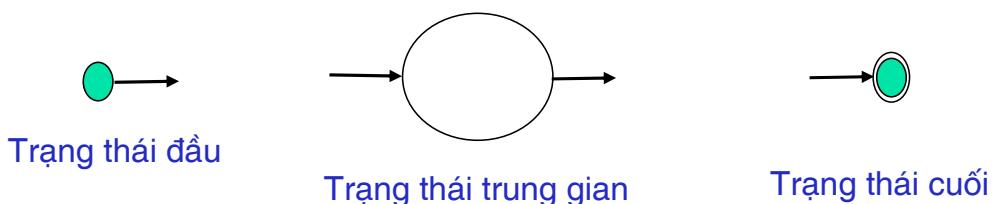
Từ bảng quyết định chuyển thành bảng các testcase trong đó mỗi cột miêu tả 1 luật được chuyển thành 1 đến n cột miêu tả các testcase tương ứng với luật đó :

- 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.

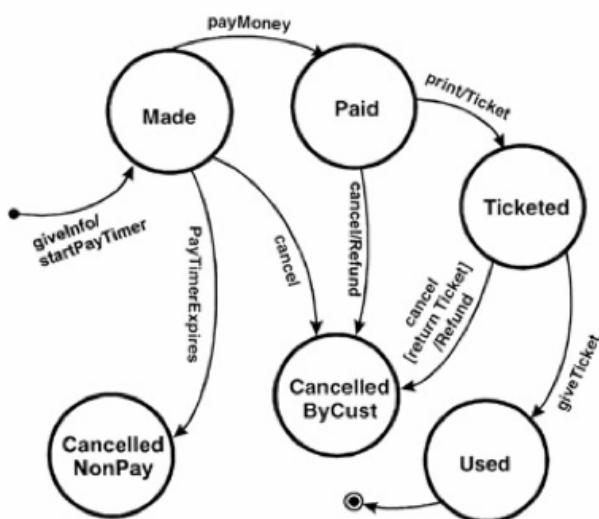
5.1.4 Lược đồ chuyển trạng thái

- 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.
- Lược đồ chuyển trạng thái ghi nhận các sự kiện xảy ra, rồi được hệ thống xử lý cũng như những đáp ứng của hệ thống.
- Khi hệ thống phải nhớ trạng thái trước đó của mình, hay phải biết trình tự các hoạt động nào là hợp lệ, trình tự nào là không hợp lệ thì lược đồ chuyển trạng thái là rất thích hợp.



Ví dụ:

Module đặt mua vé máy bay có 6 trạng thái:

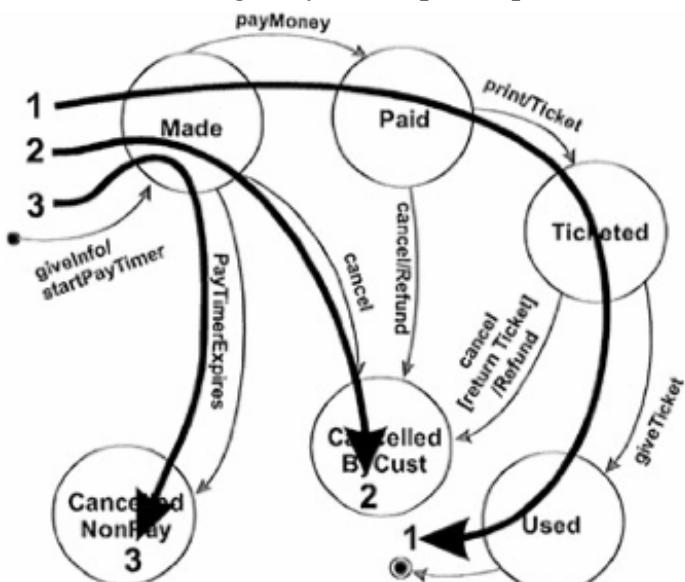


Trạng thái	Điều kiện chuyển đến	Hành động cần thực hiện tiếp
made	sau khi người dùng đã nhập thông tin khách hàng.	khởi động timer T0 đếm thời gian giữ trạng thái

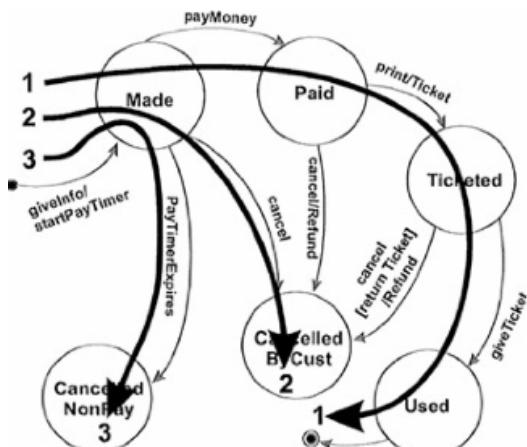
Cancelled (NonPay)	sau khi timer T0 đã hết.	null
3. Paid	sau khi người dùng đã thanh toán tiền.	null
Cancelled(byCust)	sau khi người dùng đã cancel	null
Ticketed	sau khi in vé xong.	null
6. Used	sau khi người dùng đã dùng	null

Dựa vào lược đồ chuyển trạng thái, ta có thể dễ dàng định nghĩa các testcase.

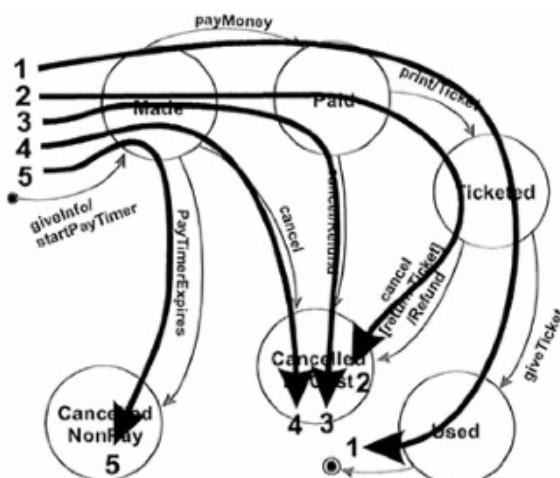
- **Phủ cấp 1 :** tạo các testcase sao cho mỗi trạng thái đều xảy ra ít nhất 1 lần. Thí dụ 3 đường chạy sẽ đạt phủ cấp 1



- Phủ cấp 2 : tạo các testcase sao cho mỗi sự kiện đều xảy ra ít nhất 1 lần. Thí dụ 3 đường chạy đạt phủ cấp 2



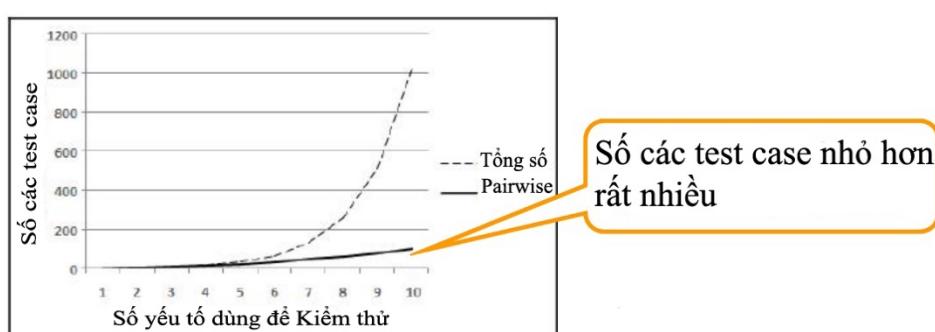
- Phủ cấp 3 : tạo các đường chạy sao cho tất cả các path chuyển đều được kiểm thử. 1 path chuyển là 1 đường chuyển trạng thái xác định, bắt đầu từ trạng thái nhập và kết thúc ở trạng thái kết thúc.
 - Đây là phủ tốt nhất vì đã vét cạn mọi khả năng hoạt động, tuy nhiên không khả thi khi path chuyển có vòng lặp.



- Phủ cấp 4 : tạo các đường chạy sao cho mỗi path chuyển *tuyến tính* đều xảy ra ít nhất 1 lần.

5.1.5 Kiểm thử theo cặp

Thực tế cho thấy hầu hết các lỗi đều được sinh ra từ sự kết hợp giá trị của các cặp tham số đầu vào.



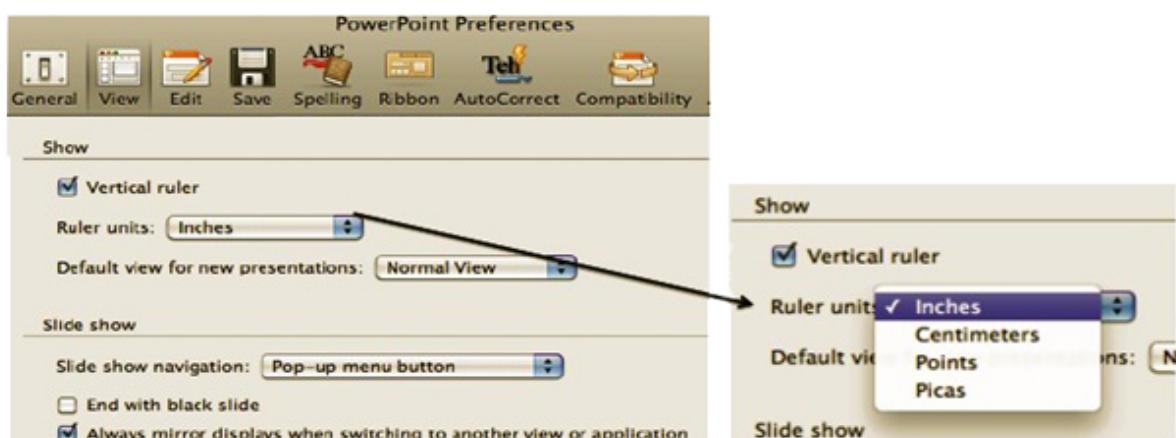
Lựa chọn tham số đầu vào và các giá trị tương ứng

Lấy tổ hợp (pairwise) của các giá trị giữa 2 tham số

Xây dựng bộ test sao cho bao phủ được tất cả các cặp xác định ở trên

Ví dụ

Xét tab tùy chọn View từ một phiên bản của phần mềm Powerpoint Microsoft (xem Hình). Dữ liệu trong hình.



The screenshot shows the 'PowerPoint Preferences' window with the 'View' tab selected. The 'Show' section contains the following settings:

- Vertical ruler
- Ruler units: **Inches**
- Default view for new presentations: **Normal View**

Below this, the 'Slide show' section includes:

- Slide show navigation: **Pop-up menu button**
- End with black slide
- Always mirror displays when switching to another view or application
- Warn before a linked file, application, or macro is opened

Panel (a) shows the 'Ruler units' dropdown with 'Inches' selected. Panel (b) shows the dropdown expanded to include 'Inches', 'Centimeters', 'Points', and 'Picas'.

(a) **(b)**

Vertical Ruler	Ruler Units	Default View	SS Navigation	End with Black	Always Mirror	Warn Before
Visible	Inches	Normal	Pop-up	Yes	Yes	Yes
Invisible	Centimeters	Slide	None	No	No	No
	Points	Outline				
	Picas					

(c)

Tab View_preference gồm bảy thuộc tính, mỗi thuộc tính lại bao gồm một trong các giá trị con khác nhau: Vertical_Ruler (Visible, InVisible), Ruler_units (Inches, Centimeters, Points, Picas), Default_View (Normal, Slide, Outline), Ss_Navigator (Popup, None), End_With_Black (Yes1, No1), Always_Mirror (Yes2, No2), Warn_Before (Yes3, No3).

Ban đầu, chúng ta có tổng số $2*4*3*2*2*2 = 384$ test case.

Các cặp có thể có: (Visible, Inches), (Visible, Centimeters), ..., (No2, No3)

Test case (Visible, Centimeters, Nomal, Non, No1, Yes2, Yes3) bao gồm 21 cặp (Visible, Centimeters), (Visible, Nomal), ..., (Yes2, Yes3)

Một bộ test case bao phủ được tất cả các cặp được cho ở bảng dưới (được tạo bởi công cụ sinh pairwise PICT ((<http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi>))

Vertical Ruler	Ruler units	Default View	Ss Navigator	End With Black	Always Mirror	Warn Before
Visible	Centimetes	Normal	None	No1	Yes2	Yes3
InVisible	Picas	Normal	Popup	Yes1	No2	No3
Visible	Picas	Slide	Popup	No1	No2	Yes3
InVisible	Inches	Normal	None	Yes1	Yes2	No3
InVisible	Points	Outline	None	No1	No2	No3
Visible	Centimetes	Slide	Popup	Yes1	Yes2	No3
Visible	Picas	Outline	None	Yes1	Yes2	Yes3
InVisible	Inches	Outline	Popup	No1	No2	Yes3
Visible	Points	Slide	None	Yes1	Yes2	Yes3
InVisible	Inches	Slide	None	No1	Yes2	Yes3
Visible	Inches	Normal	Popup	No1	Yes2	Yes3
Visible	Points	Normal	Popup	Yes1	Yes2	No3
InVisible	Centimetes	Outline	Popup	Yes1	No2	Yes3

5.2 Các kỹ thuật kiểm thử hộp trống

- Kiểm thử hộp trống dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu bên trong của module cần kiểm thử để xác định module đó có thực hiện đúng không.
- Do đó người Kiểm thử hộp trống phải có kỹ năng, kiến thức để có thể thông hiểu chi tiết về đoạn code cần kiểm thử.
- Thường tốn rất nhiều thời gian và công sức
- Với các module quan trọng, thực thi việc tính toán chính của hệ thống, phương pháp này là cần thiết.
 - Cần các kỹ thuật cho kiểm thử hộp trống
- Các phương pháp kiểm thử hộp trống:
 - Kiểm thử luồng điều khiển
 - Kiểm thử luồng dữ liệu

5.2.1 Kiểm thử luồng điều khiển

5.2.1.1 Khái niệm cơ bản

- **Đường thi hành (Execution path)** : là 1 kịch bản thi hành đơn vị phần mềm tương ứng : 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.
- 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.

Ví dụ:

```
1: WHILE NOT EOF LOOP
2:     Read Record;
3:     IF field1 equals 0 THEN
4:         Add field1 to Total
5:         Increment Counter
6:     ELSE
7:         IF field2 equals 0 THEN
8:             Print Total, Counter
9:             Reset Counter
10:        ELSE
```

```

6:           Subtract field2 from Total
7:           END IF
8:           END IF
8:           Print "End Record"
9:           END LOOP
9:           Print Counter

```

Ví dụ các đường thi hành:

1, 9

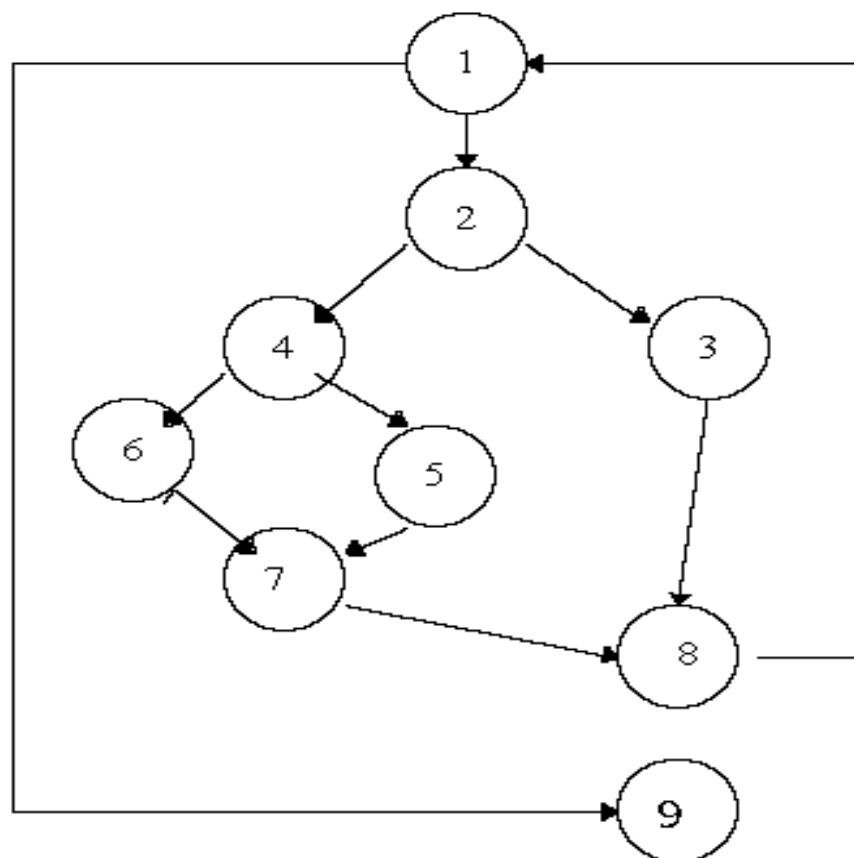
1, 2, 3, 8, 1, 9

1, 2, 4, 5, 7, 8, 1, 9

1, 2, 4, 6, 7, 8, 1, 9

Đồ thị luồng điều khiển (Control flow graph): một đồ thị luồng điều khiển (Control flow graph – CFG) là một biểu diễn đồ thị của luồng điều khiển/hay tính toán trong quá trình thực thi chương trình.

Ví dụ tương ứng với đoạn code ở trên, ta có control flow graph:



Một số vấn đề của kiểm thử luồng điều khiển:

- 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);

```

có 1 đường thi hành dài $1000*1000*1000 = 1$ tỉ lệnh gọi `doSomethingWith(i,j,k)` khác nhau.

- Thí dụ đoạn code gồm 32 lệnh if else 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 thiệt (trong 1 vài trường hợp đặc biệt) :

```
int blech (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 blech bị lỗi.

5.2.1.2 Phủ kiểm thử

Ta nên kiểm thử 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 đảm 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, điểm quyết định, điều kiện con, đường thi hành hay là sự kết hợp của chúng.

Các mức phủ kiểm thử:

- 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 : kiểm thử sao cho mỗi lệnh được thực thi ít nhất 1 lần.

```

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 :

1. foo(0,0,0,0), trả về 0
 2. foo(1,1,1,1), trả về 1
- nhưng không phát hiện lỗi
chia 0 ở hàng lệnh 8

Phủ cấp 2 : kiểm thử sao cho mỗi 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 nhánh (Branch coverage). Phủ các nhánh đảm bảo phủ các lệnh.

Với 2 test case xác định trong mức phủ trước, ta chỉ đạt được $3/4 \times 75\% = 56.25\%$ phủ các nhánh.
Nếu thêm test case 3 :

3. foo(1,2,1,2), thì mới đạt 100% phủ các nhánh.

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) return 1

- Phủ cấp 3 : 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.

Predicate	True	False
-----------	------	-------

a==0	Test Case 1 foo(0, 0, 0, 0) return 0	Test Case 2 foo(1, 1, 1, 1) return value 0
a==b	Test Case 2 foo(1, 1, 1, 1) return 1	Test Case 3 foo(1, 2, 1, 2) division by zero!
c==d		Test Case 3 foo(1, 2, 1, 2) division by zero!
bug(a)		

- 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. Ta gọi mức kiểm thử này là phủ các nhánh & điều kiện con (branch & subcondition coverage).

5.2.1.3 Phương pháp kiểm thử các đường thi hành cơ bản - Basis Path Testing (by Tom McCabe)

Các bước:

- Tù module cần kiểm thử, xây dựng đồ thị luồng điều khiển G tương ứng.
- Tính độ phức tạp Cyclomatic của đồ thị (=C).
 - $V(G) = E - N + 2$, trong đó E là số cung, N là số nút của đồ thị.
 - $V(G) = P + 1$, với P là số nút quyết định luận lý
 - Lưu ý: thông thường, nếu $V(G) > 10$, ta nên chia module thành các module nhỏ hơn để giảm xác suất gây lỗi
 - Xác định C đường thi hành tuyến tính cơ bản cần kiểm thử.

Tạo từng test case cho từng đường thi hành tuyến tính cơ bản.

Các bước xác định đường tuyến tính độc lập:

- Xác định đường cơ bản, đường này nên là đường thi hành phổ biến nhất.
- Để chọn đường thứ 2, thay đổi cung xuất của nút quyết định đầu tiên và cố gắng giữ lại maximum phần còn lại.
- Để chọn đường thứ 3, dùng đường cơ bản nhưng thay đổi cung xuất của nút quyết định thứ 2 và cố gắng giữ lại maximum phần còn lại.

- Tiếp tục thay đổi cung xuất cho từng nút quyết định trên đường cơ bản để xác định đường thứ 4, 5,... cho đến khi không còn nút quyết định nào trong đường cơ bản nữa.
- Lặp dùng tuần tự các đường tìm được làm đường cơ bản để xác định các đường mới xung quanh nó y như các bước 2, 3, 4 cho đến khi không tìm được đường tuyến tính độc lập nào nữa (khi đủ số C).

Ví dụ:

Với CFG cho ở ví dụ trên, ta có $C = 3 + 1 = 4$, đường độc lập:

1, 9

1, 2, 3, 8, 1, 9

1, 2, 4, 5, 7, 8, 1, 9

1, 2, 4, 6, 7, 8, 1, 9

Tương ứng, các test case là:

Đường chạy	Test cases	
	Input	Expected output
1, 9	File rỗng	Không xác định
1, 2, 3, 8, 1, 9	File có 1 bản ghi: field1 = 0, field2 = 2	Không xác định
1, 2, 4, 5, 7, 8, 1, 9	File có 1 bản ghi: field1 = 3, field2 = 0	Không xác định
1, 2, 4, 6, 7, 8, 1, 9	File có 1 bản ghi: field1 = 3, field2 = 6	Không xác định

Lưu ý:

- do không có đặc tả, nên ta không thể xác định expected output trong trường hợp này.
- Chỉ cần căn cứ vào test case, ta có thể khoanh vùng các câu lệnh gây ra lỗi. Ví dụ test case với đường chạy 1-9 (file rỗng) lỗi thì lỗi chỉ xảy ra ở câu lệnh 1 hoặc 9.

5.2.1.4 Kiểm thử vòng lặp

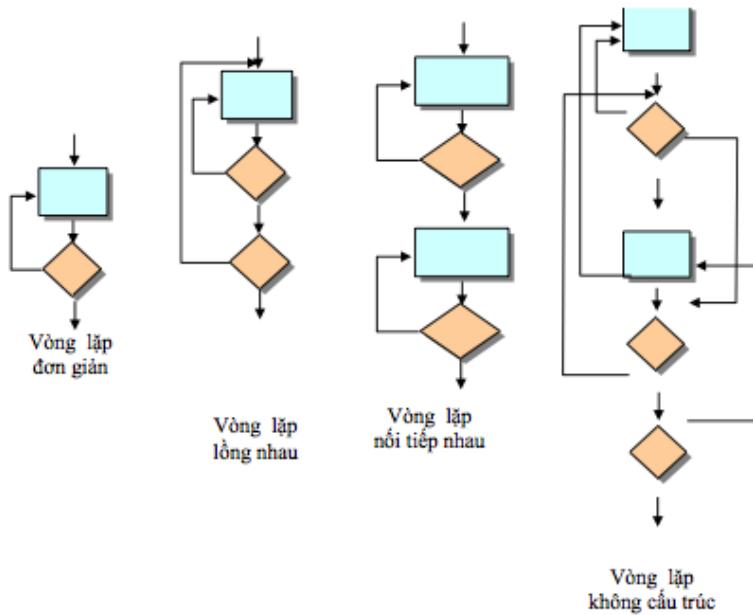
Tập trung vào kiểm tra tính hợp lệ của cấu trúc vòng lặp.

Với vòng lặp đơn:

- bỏ qua vòng lặp
- lặp 1 lần
- lặp 2 lần
- lặp k lần ($k < n$)

- lặp $n-1$, n , $n + 1$ lần

Với n là số lần lặp tối đa.



Với vòng lặp lồng nhau:

- khởi đầu với vòng lặp trong cùng. Thiết lập các tham số lặp cho các vòng lặp bên ngoài về giá trị nhỏ nhất
- kiểm tra tham số $min+1$, một giá trị tiêu biểu, $max - 1$, max cho vòng lặp trong cùng khi các tham số lặp của các vòng lặp bên ngoài là nhỏ nhất
- tiếp tục với các vòng lặp bên ngoài cho tới khi tất cả các vòng lặp được kiểm tra

Ví dụ

```
// LOOP TESTING EXAMPLE PROGRAM import java.io.*;
class LoopTestExampleApp {
    // ----- FIELDS -----
    public static BufferedReader keyboardInput = new BufferedReader(new
    InputStreamReader(System.in));
    private static final int MINIMUM = 1;
    private static final int MAXIMUM = 10;
    // ----- METHODS -----
    /* Main method */
    public static void main(String[] args) throws IOException {
        System.out.println("Input an integer value:");
    }
}
```

```

int input = new Integer(keyboardInput.readLine()).intValue();
int numberofIterations=0;
for(int index=input;index >= MINIMUM && index <= MAXIMUM;index++)
{ numberofIterations++;
} // Output and end System.out.println("Number of iterations = " +
numberofIterations); }
}

```

Ta có test case là

Input (input)	Expexted output (number of iteration)
11	0 (bỏ qua vòng lặp)
10	1 (lặp 1 lần)
9	2 (lặp 2 lần)
5	6 (lặp k lần)
2	9 (lặp n – 1 lần)
1	10 (lặp n lần)
0	0 (bỏ qua vòng lặp)

5.2.2 Kiểm thử luồng 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 thông qua trình tự 3 bước : được tạo ra, được dùng và được xóa đi.
- Chỉ có những lệnh nằm trong phạm vi truy xuất biến mới có thể truy xuất/xử lý được biến. Phạm vi: toàn cục, cục bộ

Phân tích đời sống của 1 biến

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

d (define): định nghĩa biến, gán giá trị xác định cho biến (nhập dữ liệu vào biến cũng là hoạt động gán trị cho biến).

r(reference) : tham khảo giá trị của biến (thường thông qua biểu thức).

u(undefine) : 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.

~r : biến chưa tồn tại rồi được dùng ngay (trị nào ?)

~u : biến chưa tồn tại rồi bị hủy (không bình thường).

Phân tích đời sống của 1 biến (tiếp)

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

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.

dr : biến được định nghĩa rồi được dùng : trình tự đúng và bình thường.

du : 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.

rd : biến được dùng rồi định nghĩa giá trị mới : hợp lý.

rr : biến được dùng rồi dùng tiếp : hợp lý.

ru : biến được dùng rồi bị hủy : hợp lý.

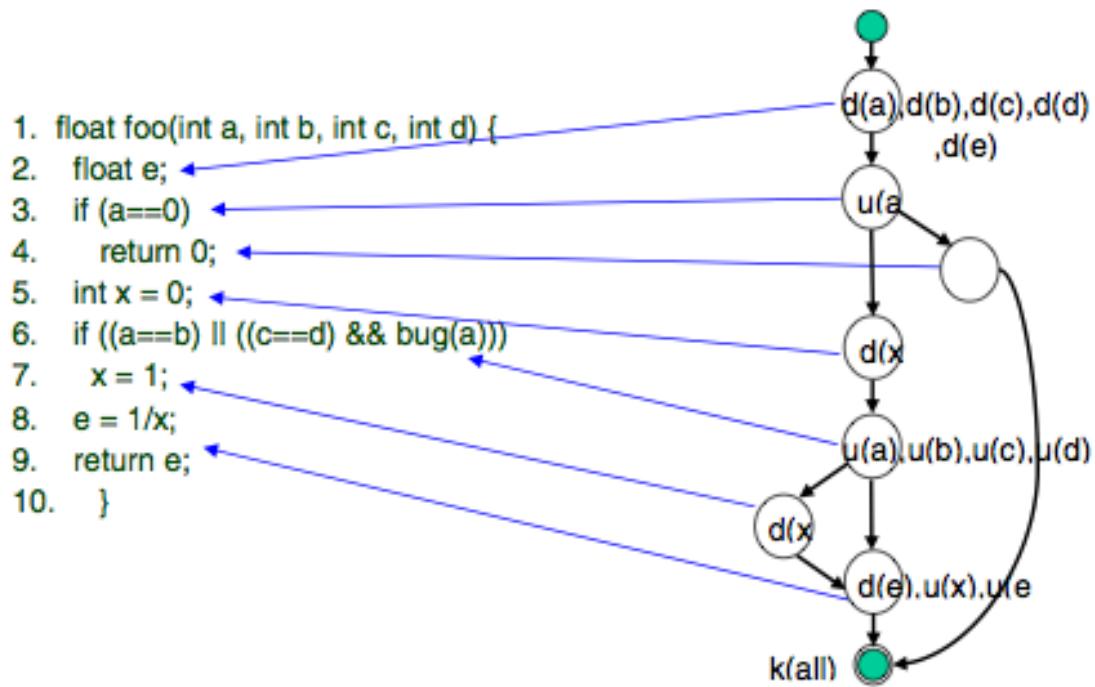
ud : biến bị xóa bỏ rồi được định nghĩa lại : chấp nhận được.

ur : biến bị xóa bỏ rồi được dùng : lỗi.

uu : biến bị xóa bỏ rồi bị xóa nữa : có lẽ là lỗi lập trình.

Đồ thị dòng dữ liệu

- Là một trong nhiều phương pháp miêu tả các kịch bản đời 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 module cần kiểm thử.



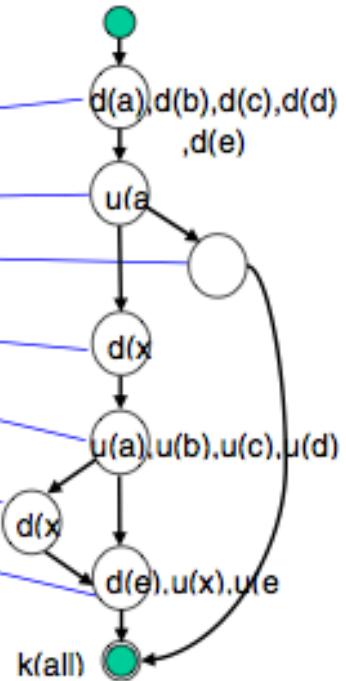
Qui trình kiểm thử dòng dữ liệu của 1 module gồm các bước:

- 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 (theo thuật giải chi tiết ở chương 3).
- Lắp kiểm thử đòn súng từng biến dữ liệu :
 - mỗi biến có thể có tối đa C kịch bản đòn súng khác nhau.
 - trong từng kịch bản đòn súng 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
- Ví dụ:
Đồ thị sau 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 lắp kiểm thử đòn súng từng biến a, b, c, d, e, x.

```

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. }

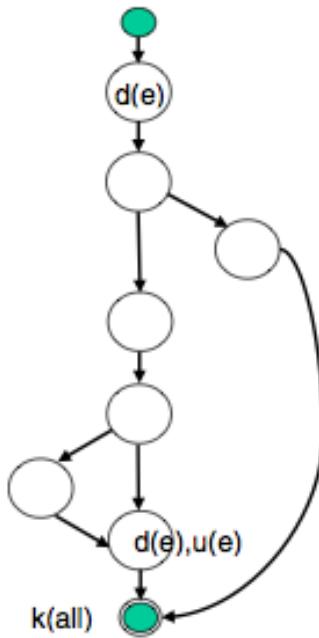
```



Kiểm thử đòn sống của biến 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



5.3 Kiểm thử đơn vị tự động

5.3.1 Giới thiệu chung

Một **unit** là phần nhỏ nhất có thể test được của chương trình. Trong lập trình hướng đối tượng, một **unit** có thể là một method của một class.

Unit Test là các đoạn mã có cấu trúc giống như các đối tượng được xây dựng để kiểm tra từng bộ phận trong hệ thống. Mỗi Unit Test sẽ gửi đi một thông điệp và kiểm tra câu trả lời có đúng hay không, bao gồm:

- Các kết quả trả về mong muốn.
- Các ngoại lệ mong muốn.

Unit Test kiểm tra tính đúng đắn của các hoạt động của các thành phần đơn vị với một quy trình tách biệt với quy trình phát triển phần mềm, giúp phát hiện sai sót kịp thời. Unit Test còn có thể giúp phát hiện các vấn đề tiềm ẩn và các lỗi thời gian thực ngay cả trước khi chuyên viên kiểm định chất lượng (QA - Quality Assurance) tìm ra, thậm chí có thể sửa lỗi ngay từ ý tưởng thiết kế.

b. Đặc điểm của một Unit Test

- Đóng vai trò như là người sử dụng đầu tiên của hệ thống.
- Chỉ có giá trị khi chúng phát hiện ra các lỗi tiềm ẩn hoặc lỗi kỹ thuật.
- Các đoạn mã Unit Test hoạt động liên tục hoặc định kỳ để thăm dò và phát hiện các lỗi kỹ thuật trong suốt quá trình phát triển, do đó Unit Test còn được gọi là kỹ thuật kiểm nghiệm tự động.
- Unit Test có 3 trạng thái cơ bản:
 - Fail (trạng thái lỗi).

- Ignore (tạm ngừng thực hiện).
- Pass (trạng thái làm việc).

Do việc kiểm thử đơn vị đòi hỏi phải kiểm tra từng nhánh lệnh, nên đòi hỏi người kiểm thử có kiến thức về lập trình cũng như về thiết kế của hệ thống nên người thực hiện thường là lập trình viên.

c. Quy trình thiết kế hoạt động của Unit Test

Mỗi một Unit Test đều được thiết kế theo trình tự sau:

- Chuẩn bị các ca kiểm thử (*Test case*) hoặc kịch bản kiểm thử (*Test script*), trong đó chỉ định rõ dữ liệu đầu vào, các bước thực hiện và dữ liệu đầu ra mong muốn. Các *Test case* và *Test script* này nên được giữ lại để tái sử dụng.
- Thiết lập các điều kiện cần thiết: khởi tạo các đối tượng, xác định tài nguyên cần thiết, xây dựng các dữ liệu giả lập...
- Triệu gọi các phương thức cần kiểm tra.
- Kiểm tra sự hoạt động đúng đắn của các phương thức.
- Dọn dẹp tài nguyên sau khi kết thúc kiểm tra.

Unit Test chỉ hoạt động hiệu quả khi:

- Được vận hành lặp lại nhiều lần.
- Tự động hoàn toàn.
- Độc lập với các Unit Test khác.

d. Lợi ích của Unit Test

- Tách biệt mã kiểm thử ra khỏi mã chương trình.
- Tạo ra môi trường lý tưởng để kiểm tra bất kỳ đoạn mã nào, có khả năng thăm dò và phát hiện lỗi chính xác, duy trì sự ổn định của toàn bộ phần mềm và giúp tiết kiệm thời gian so với công việc gõ rối truyền thống.
- Tự động hóa việc tổ chức và thi hành các bộ số liệu kiểm thử, giải phóng chuyên viên QA khỏi các công việc kiểm tra phức tạp.
- Cố lập từng phần của chương trình và đảm bảo những phần đó được test chạy đúng như yêu cầu, phát hiện các lỗi nghiêm trọng có thể xảy ra trong những tình huống rất hẹp.
- Phát hiện các thuật toán thực thi không hiệu quả, các thủ tục chạy vượt quá giới hạn thời gian
- Khi làm việc team, nếu mỗi phần do từng thành viên viết được test unit kỹ càng thì khi kết hợp lại sẽ suôn sẻ, ít gặp lỗi hơn.
- Là công cụ đánh giá năng lực của lập trình viên. Số lượng các tình huống kiểm tra (*test case*) chuyển trạng thái "pass" sẽ thể hiện tốc độ làm việc, năng suất của

bạn.

Bên cạnh các lợi ích trên, Unit Test cũng có một số điểm hạn chế:

- Unit Test không thể bắt được tất cả các lỗi của chương trình, những test case chỉ kiểm lỗi những unit nhỏ nhất của chương trình, cho nên không lường trước những vấn đề có thể xảy ra khi kết hợp các module lại với nhau.
- Có nhiều trường hợp không thể sử dụng Unit Test được, ví dụ với private class, private method, ...

e. Chiến lược viết mã Unit Test hiệu quả

- Phân tích các tình huống có thể xảy ra đối với mã. Không được bỏ qua các tình huống tồi tệ nhất có thể xảy ra, ví dụ dữ liệu nhập làm một kết nối cơ sở dữ liệu thất bại, ứng dụng bị treo vì một phép toán chia cho không, các thủ tục đưa ra lỗi ngoại lệ sai có thể phá hỏng ứng dụng một cách bí ẩn...
- Mọi Unit Test phải bắt đầu với trạng thái "fail" và chuyển trạng thái "pass" sau một số thay đổi hợp lý đối với mã chính.
- Nhập một số lượng đủ lớn các giá trị đầu vào để phát hiện điểm yếu của mã theo nguyên tắc:
 - Nếu nhập giá trị đầu vào hợp lệ thì kết quả trả về cũng phải hợp lệ.
 - Nếu nhập giá trị đầu vào không hợp lệ thì kết quả trả về phải không hợp lệ.
- Sớm nhận biết các đoạn mã không ổn định và có nguy cơ gây lỗi cao, viết Unit Test tương ứng để không chế.
- Đòi hỏi sự nỗ lực, kinh nghiệm và sự sáng tạo như viết phần mềm.

5.3.2 Tổng quan thư viện Junit

a. Junit là gì?

- Junit là một framework đơn giản dùng cho việc tạo các Unit Testing tự động và chạy các test case có thể lặp đi lặp lại.
- Junit được xây dựng bởi Erich Gamma và Kent Beck, hai người nổi tiếng nhất về lập trình XP.
- Junit là một phần của họ kiến trúc Xunit cho việc tạo các Unit Testing và cũng là một chuẩn trên thực tế cho Unit Testing trong Java.
- b. Đặc điểm của Junit
- **Junit** là công cụ giúp ta thử nghiệm, gỡ rối chương trình Java. Với Junit bạn dễ dàng theo dõi diễn biến của chương trình, nhanh chóng dàn dựng hàng loạt phép thử (test case) để kiểm tra mọi việc có xảy ra đúng như dự định hay không.
- Các test case của JUnit là các lớp của Java, các lớp này bao gồm một hay nhiều

các phương thức unit testing, và những test này lại được nhóm thành các Test Suite.

- JUnit có những đặc điểm quan trọng như sau:
- Xác nhận (assert) việc kiểm tra kết quả được mong đợi.
- Các Test Suite cho phép chúng ta dễ dàng tổ chức hay chạy các test.
- Hỗ trợ giao diện đồ họa hay giao diện dòng lệnh.

Các test trong JUnit có thể là các test được chấp nhận hay thất bại, các test này được thiết kế để khi chạy mà không cần có sự can thiệp của con người. Từ những thiết kế như thế, bạn có thể thêm các bộ test vào quá trình tích hợp và xây dựng phần mềm một cách liên tục và để cho các test chạy một cách tự động.

c. Lợi ích của Junit

- JUnit tránh cho người lập trình phải làm đi làm lại những việc kiểm thử nhằm chán bằng cách tách biệt mã kiểm thử ra khỏi mã chương trình.
- Tự động hóa việc tổ chức và thi hành các bộ số liệu kiểm thử.

Thoạt tiên, khi sử dụng JUnit, ta đều có thể có cảm giác là JUnit chỉ làm mất thêm thời gian cho việc kiểm thử. Thay vì phải viết thêm các lớp và phương thức mới phục vụ cho công tác kiểm thử, ta có thể soạn nhanh một bộ số liệu rồi viết ngay vào trong phương thức **main()** và quan sát ngay kết quả kiểm thử. Nhưng khi tổ chức lại chương trình cho hợp lý hơn hoặc khi phải thay đổi chương trình để phục vụ cho nhu cầu mới, các bộ số liệu kiểm thử trước đây sẽ cần được sử dụng lại để chắc chắn rằng những thay đổi trong chương trình không làm phương hại đến những thành quả trước đó, lúc này ta sẽ phải mất thời gian để tìm hiểu lại xem bộ số liệu trước đây sẽ tương ứng với kết xuất gì vì ta không thể nhớ hết mọi hoạt động kiểm thử đã diễn ra, nếu dữ liệu test lớn thì việc nhớ đó hoàn toàn bất khả thi. Và việc sử dụng Junit sẽ hoàn toàn khắc phục được những yếu điểm đó.

d. Một số phương thức trong Junit

○ **Các phương thức assertXXX()**

Các phương thức dạng assertXXX() được dùng để kiểm tra các điều kiện khác nhau. Dưới đây là mô tả các phương thức assertXXX() khác nhau có trong lớp junit.framework.Assert:

- **Boolean assertEquals():** So sánh hai giá trị để kiểm tra bằng nhau. Phép thử thất bại nếu hai giá trị không bằng nhau.
- **Boolean assertFalse():** Đánh giá biểu thức logic. Phép thử thất bại nếu biểu thức đúng.
- **Boolean assertNotNull():** So sánh tham chiếu của một đối tượng với Null. Phép

thử thất bại nếu tham chiếu đối tượng Null.

- **Boolean assertNotSame()**: So sánh địa chỉ vùng nhớ của hai tham chiếu hai đối tượng bằng cách sử dụng toán tử `==`. Phép thử thất bại trả về nếu cả hai đều tham chiếu đến cùng một đối tượng.
- **Boolean assertNull()**: So sánh tham chiếu của một đối tượng với giá trị Null. Phép thử thất bại nếu đối tượng không là Null.
- **Boolean assertSame()**: So sánh địa chỉ vùng nhớ của hai tham chiếu đối tượng bằng cách sử dụng toán tử `==`. Phép thử thất bại nếu cả hai không tham chiếu đến cùng một đối tượng.
- **Boolean assertTrue()**: Đánh giá một biểu thức logic. Phép thử thất bại nếu biểu thức sai.
- **fail()**: Phương thức này làm cho test hiện tại thất bại, phương thức này thường được sử dụng khi xử lý các ngoại lệ.

Tất cả các phương thức trên đều có thể nhận vào một String không bắt buộc làm tham số đầu tiên. Khi được xác định, tham số này cung cấp như một thông điệp thất bại giúp cho việc sửa lỗi được dễ dàng hơn.

Ví dụ phương thức test hai xâu có trùng nhau không:

```
@Test
public void Test(){
    String s1 = " xâu 1";
    String s2 = " xâu 2";

    assertEquals(s1, s2);
}
```

Hoặc ta thêm thông điệp “Hai xâu khác nhau” làm tham số đầu tiên của phương thức assertEquals() nhằm cung cấp thông điệp lỗi sai rõ ràng hơn.

```
@Test
public void Test(){
    String s1 = " xâu 1";
    String s2 = " xâu 2";
    assertEquals("Hai xâu khác nhau :", s1,
s2);
}
```

- **setUp() và tearDown()**

Hai phương thức này là một phần của lớp junit.framework.TestCase. Khi sử dụng hai phương thức này sẽ giúp chúng ta tránh được việc trùng mã khi nhiều test cùng chia sẻ nhau ở phần khởi tạo và dọn dẹp các biến.

JUnit tuân thủ theo một dãy có thứ tự các sự kiện khi chạy các test. Đầu tiên, nó tạo ra một thể hiện mới của Test Case ứng với mỗi phương thức thử. Từ đó, nếu bạn có 5 phương thức thử thì JUnit sẽ tạo ra 5 thể hiện của Test Case. Vì lý do đó, các biến thể hiện không thể được sử dụng để chia sẻ trạng thái giữa các phương thức test. Sau khi tạo xong tất cả các đối tượng test case, JUnit tuân theo các bước sau cho mỗi phương thức test:

1. Gọi phương thức setUp() của test case.
2. Gọi phương thức thử.
3. Gọi phương thức tearDown() của test case.

Quá trình này được lặp lại đối với mỗi phương thức thử trong Test Case.

Thông thường chúng ta có thể bỏ qua phương thức tearDown() vì mỗi phương thức thử riêng không phải là những tiến trình chạy tốn nhiều thời gian.

5.4 Bảng tóm tắt Testing Levels/ Techniques

Ngoài các kỹ thuật kiểm thử hộp trắng, hộp đen, có 2 kỹ thuật kiểm thử khác:

Kiểm thử gia tăng - Incremental Testing: Test các nhóm tích hợp các mô-đun đã được test với các mô-đun vừa hoàn thành. Được thực hiện bằng cách thêm từng thành phần một, rồi kiểm thử kết quả của hành động thêm này.

Thread Testing: Sử dụng trong kiểm thử tích hợp, để xác minh khả năng của các chức năng chính bằng việc kiểm thử một chuỗi các units mà thực hiện một chức năng của hệ thống.

Một bảng thống kê việc áp dụng kỹ thuật kiểm thử cho các mức kiểm thử khác nhau cho ở bảng dưới đây:

Testing Levels/ Techniques	White-box	Black-box	Incremental	Thread
Unit Testing	X			
Integration Testing	X		X	X
System Testing		X		
Acceptance Testing		X		

Chương 6: Các công cụ hỗ trợ đảm bảo chất lượng phần mềm

6.1 Các công cụ quản lý thông tin trong Đảm bảo chất lượng phần mềm

6.1.1 Phần mềm hỗ trợ viết tài liệu

Các phần mềm hỗ trợ viết tài liệu thường được sử dụng trong việc lên kế hoạch kiểm thử - *Test Plan*, thiết kế và tạo ra các ca kiểm thử - *Test Case*, viết các báo cáo, các hướng dẫn sử dụng cho người dùng ...

Các phần mềm thường được sử dụng là :

- Microsoft Office Word.
- Microsoft Office Exel.
- Microsoft Office Project.
- Microsoft Office Power Point.

6.1.2 Phần mềm quản lý lỗi

Một lỗi phần mềm là một lỗi, thiếu sót, thất bại, hoặc lỗi trong một chương trình máy tính hoặc hệ thống sản xuất một kết quả không chính xác hoặc không mong muốn, hoặc làm cho nó hành xử theo những cách không mong muốn. Hầu hết các lỗi phát sinh từ những sai lầm và lỗi của con người trong các đoạn mã nguồn của một chương trình hoặc trong các thiết kế, và một số được gây ra bởi các trình biên dịch mã không chính xác. Chính vì thế, trong các dự án Công nghệ thông tin cần có một hệ thống theo dõi lỗi để giúp theo dõi và báo cáo các lỗi trong quá trình phát triển phần mềm.

Các thành phần chính của một hệ thống theo dõi lỗi là một cơ sở dữ liệu ghi lại những thông tin về lỗi được phát hiện như : thời gian phát hiện lỗi, mức độ nghiêm trọng của lỗi, cách gỡ lỗi ...

Dưới đây là một số loại phần mềm theo dõi lỗi phổ biến:

6.1.2.1 Bugzilla

Bugzilla là một phần mềm máy chủ cho phép quản lý các lỗi phát sinh trong quá trình phát triển dự án phần mềm được phát triển bởi tổ chức Mozilla. Các tính năng:

- Cho phép khai báo các lỗi mới phát hiện.
- Phân loại các lỗi theo thành phần hệ thống, độ phức tạp, mức độ ưu tiên.
- Hệ thống quản lý cho phép một người khai báo lỗi và giao trách nhiệm sửa lỗi cho một người khác.

- Cho phép quản lý quá trình hoạt động cũng như tiến độ test lỗi từng dự án.
- Cho phép nhiều user làm việc cùng lúc, dễ tìm kiếm và phân bổ công việc cho từng thành viên.
- Cập nhật thông tin cho thành viên tham gia dự án thông qua chức năng gửi thư điện tử.

Link trang chủ: <http://www.bugzilla.org/>

6.1.2.2 *Redmine*

Redmine là phần mềm nguồn mở hữu ích cho việc quản trị dự án với rất nhiều tiện ích hỗ trợ. Các tính năng:

- Quản lý dự án bao gồm cả biểu đồ Gantt.
- Hỗ trợ đồng thời nhiều dự án cùng với các dự án nhỏ bên trong.
- Kiểm soát truy cập linh hoạt theo quyền hạn.
- Quản lý theo từng thời gian cụ thể.
- Cho phép tùy chỉnh theo từng lĩnh vực.
- Quản lý file, tin tức tài liệu.
- Hỗ trợ nhiều database.

Link trang chủ: <http://www.redmine.org/>

6.1.2.3 *Atlassian JIRA*

JIRA là một ứng dụng theo dõi và quản lý lỗi, vấn đề và dự án, được phát triển để làm quy trình này trở nên dễ dàng hơn cho mọi tổ chức. JIRA đã được thiết kế với trọng tâm vào kết quả công việc, có thể sử dụng ngay và linh hoạt khi sử dụng. Các tính năng:

- Quản lý lỗi, tính năng, công việc, những cải tiến hoặc bất kỳ vấn đề gì.
- Theo dõi các tệp gắn, những thay đổi, các cấu phần và các phiên bản.
- Bảng phân tích đồ họa có thể tùy biến và các số liệu thống kê thời gian thực.
- Dễ dàng mở rộng và tích hợp với các hệ thống khác.
- Có thể chạy trên hầu hết các nền tảng phần cứng, hệ điều hành và cơ sở dữ liệu.
- Dịch vụ Web cho phép kiểm soát hệ thống.

Link trang chủ: <http://www.atlassian.com>

6.2 Công cụ kiểm thử tự động là gì ?

6.2.1 Khái niệm

Công cụ kiểm thử tự động là các công cụ giúp thực hiện việc kiểm thử phần mềm một cách tự động.

Ý nghĩa của các công cụ kiểm thử tự động:

1. Giảm bớt công sức và thời gian thực hiện quá trình kiểm thử .
2. Tăng độ tin cậy.
3. Giảm sự nhảm chán cho con người.
4. Rèn luyện kỹ năng lập trình cho kiểm thử viên.
5. Giảm chi phí cho tổng quá trình kiểm thử .

Thuận lợi và khó khăn cơ bản khi áp dụng công cụ kiểm thử tự động:

1. Không cần can thiệp của kỹ thuật viên.
2. Giảm chi phí khi thực hiện kiểm tra số lượng lớn test case hoặc test case lặp lại nhiều lần.
3. Giả lập tình huống khó có thể thực hiện bằng tay.
4. Mất chi phí tạo các script để thực hiện kiểm thử tự động.
5. Tốn chi phí dành cho bảo trì các script.
6. Đòi hỏi kỹ thuật viên phải có kỹ năng tạo script kiểm thử tự động.
7. Không áp dụng được trong việc tìm lỗi mới của phần mềm.

6.2.2 Quy trình kiểm thử tự động

1. **Tạo kịch bản kiểm thử**

Giai đoạn này chúng ta sẽ dùng test tool để ghi lại các thao tác lên phần mềm cần kiểm tra và tự động sinh ra kịch bản kiểm thử - *test script*.

2. **Chỉnh sửa kịch bản kiểm thử**

Chỉnh sửa để test script thực hiện kiểm tra theo đúng yêu cầu đặt ra, cụ thể là làm theo test case cần thực hiện.

3. **Chạy test script để kiểm thử tự động**

Chạy kịch bản kiểm thử để kiểm tra phần mềm có đưa ra đúng như kết quả mong muốn không.

4. **Đánh giá kết quả**

Kiểm tra kết quả thông báo sau khi thực hiện kiểm thử tự động. Sau đó bổ sung, chỉnh sửa những sai sót.

6.3 Công cụ hỗ trợ kiểm thử đơn vị

Kiểm thử đơn vị - *Unit Testing* là một cách tiếp cận kiểm tra các đơn vị cá nhân của mã nguồn và kiểm tra nếu nó phù hợp với mục đích. Kiểm thử đơn vị cho phép các lập trình viên để sửa đổi và duy trì mã hiện có và vẫn đảm bảo chắc chắn rằng các mô-đun hoạt động chính xác và đáp ứng các yêu cầu chức năng và không có chức năng dự kiến. Nó cũng giúp làm giảm sự không chắc chắn trong các đơn vị và đặc biệt hữu ích trong một cách tiếp cận từ dưới lên phong cách thử nghiệm.

Một số loại công cụ kiểm tra đơn vị:

1. **JUnit.**
2. **Jtest:** là một thử nghiệm java tự động và mã tĩnh phân tích sản phẩm được thực hiện bởi Parasoft. Nó nhằm mục đích nâng cao độ tin cậy, tính năng, bảo mật, hiệu suất, và bảo trì. Chức năng cơ bản bao gồm kiểm tra mức đơn vị, phân tích tĩnh, kiểm tra hồi quy, phát hiện lỗi thời gian chạy, và xem xét mã.

Link trang chủ: <http://www.parasoft.com>.

1. **Sonar:** là một nền tảng quản lý chất lượng nguồn mở Java, dành riêng cho liên tục phân tích và đo lường chất lượng mã nguồn từ các danh mục đầu tư dự án với phương thức llop.

Link trang chủ: <http://www.sonatype.com/>

6.4 Công cụ hỗ trợ kiểm thử chức năng tự động

Có nhiều công cụ kiểm thử chức năng tự động, cả có phí và miễn phí, ví dụ như:

- Selenium công cụ miễn phí, kiểm thử website
- Unified Functional Testing (UFT) công cụ có phí, kiểm thử website và phần mềm chạy trên máy tính
- Appium công cụ miễn phí, kiểm thử ứng dụng di động

Trong các công cụ trên, Selenium được sử dụng nhiều do là công cụ mạnh mẽ, miễn phí và cộng đồng hỗ trợ mạnh mẽ,

Khái quát về Selenium

Selenium là một công cụ hỗ trợ kiểm thử tự động cho các ứng dụng Web. Selenium hỗ

trợ kiểm thử trên hầu hết các trình duyệt phổ biến hiện nay như Firefox, Internet Explorer, Safari, ... cũng như các hệ điều hành chủ yếu như Windows, Linux, Mac,... Selenium cũng hỗ trợ một số lớn các ngôn ngữ lập trình Web phổ biến hiện nay như C#, Java, Perl, PHP, Python, Ruby,... Công cụ này có thể kết hợp thêm với một số công cụ khác như Junit và TestNG nhưng với người dùng thông thường chỉ cần chạy tự động mà không cần cài thêm các công cụ hỗ trợ.

a. Đặc điểm của Selenium

- Mã nguồn mở: Đây là điểm mạnh nhất của Selenium khi so sánh với các test tool khác. Vì là mã nguồn mở nên chúng ta có thể sử dụng mà không phải lo lắng về phí bản quyền hay thời hạn sử dụng.
- Cộng đồng hỗ trợ: vì là mã nguồn mở nên Selenium có một cộng đồng hỗ trợ khá mạnh mẽ. Bên cạnh đó, Google là nơi phát triển Selenium nên chúng ta hoàn toàn có thể yên tâm về sự hỗ trợ miễn phí khi có vấn đề về Selenium. Tuy nhiên, đây cũng là một điểm yếu của Selenium. Vì công cụ này hoàn toàn miễn phí, cộng đồng lại đông nên một vấn đề có thể nhiều giải pháp, và có thể một số giải pháp là không hữu ích. Mặc khác, chúng ta không thể hối thúc hay ra deadline cho sự hỗ trợ.
- Selenium hỗ trợ nhiều ngôn ngữ lập trình.
- Selenium hỗ trợ chạy trên nhiều hệ điều hành khác nhau với mức độ chỉnh sửa script hầu như là không có. Thực sự thì điều này phụ thuộc phần lớn vào khả năng viết script của người dùng.
- Chạy test case ở background. Khi chúng ta thực thi một test script, chúng ta hoàn toàn có thể làm việc khác trên cùng một máy tính. Điều này hỗ trợ chúng ta không cần tốn quá nhiều tài nguyên máy móc khi chạy test script.
- Không hỗ trợ Win app. Selenium thực sự chỉ hỗ trợ chúng ta tương tác với Browser mà không hỗ trợ chúng ta làm việc với các Win app, kể cả Win dialog như Download/Upload. Vậy nên, để xử lý các trường hợp cần tương tác với hệ thống hay một app thứ ba, chúng ta cần một hay nhiều thư viện khác như AutoIt hay Coded UI.

Là một công cụ hỗ trợ kiểm tra tính năng nên Selenium không có khả năng giả lập nhiều người dùng ảo cùng một lúc. Công việc của nó là chạy kiểm thử tự động dựa trên một kịch bản đã được thiết kế từ trước. Qua đó chúng ta có thể chắc chắn rằng đối tượng kiểm thử có hoạt động đúng như mong đợi hay không.

b. Các thành phần của Selenium

Selenium là một bộ công cụ hỗ trợ kiểm thử tự động các tính năng của ứng dụng trên nền Web, bao gồm 4 thành phần: Selenium IDE, Selenium Grid, Selenium 1.0 (hay Selenium Remote Control – Selenium RC) và Selenium 2.0 (hay Selenium

WebDriver). Mỗi loại có một vai trò cụ thể trong việc hỗ trợ sự phát triển của tự động hóa kiểm thử ứng dụng web.

❖ **Selenium IDE**

Selenium IDE(*Intergated Development Environment*) là được phát triển dưới hình thức add-on của Firefox. Chúng ta chỉ có thể Record trên trình duyệt FireFox, nhưng bù lại, chúng ta có thể Playback trên các trình duyệt khác như là IE, Chrome....

Selenium có thể sinh code tự động hoặc nạp các đoạn mã viết tay. Công cụ này cung cấp chức năng “thu và chạy lại” – Record and Playback. Sau đó chạy lại các câu lệnh này để kiểm thử. Chức năng này rất hữu dụng giúp tiết kiệm thời gian viết kịch bản kiểm thử. Selenium IDE còn cho phép lưu kịch bản đã thu dưới nhiều loại ngôn ngữ lập trình khác nhau như Java, PHP, C#, Ruby...[3].

Selenium Core: Đã được tích hợp trong Selenium IDE. Selenium Core là một công cụ chạy các test script viết bằng Selenese. Thế mạnh của công cụ này là có thể chạy test script trên gần như tất cả các trình duyệt, nhưng lại yêu cầu được cài đặt trên máy chủ của website cần kiểm tra. Điều này là không thể khi Tester không có quyền truy cập đến máy chủ đó.

❖ **Selenium RC**

Selenium RC (Remote Control) là một framework kiểm thử cho phép thực hiện nhiều hơn và tuyển tính các hành động trên trình duyệt. Nó cho phép cho phép các nhà phát triển tự động hóa kiểm thử sử dụng một ngôn ngữ lập trình cho tính linh hoạt tối đa và mở rộng trong việc phát triển logic thử nghiệm.

Công cụ này có thể nhận các test script được thu bởi Selenium IDE, cho phép chỉnh sửa, cải tiến linh động bằng nhiều ngôn ngữ lập trình khác nhau. Sau đó khởi động một trong các trình duyệt Web được chỉ định để thực thi kiểm thử trực tiếp trên trình duyệt đó. Selenium RC còn cung cấp khả năng lưu lại kết quả kiểm thử; cung cấp một API (Application Programming Interface) và thư viện cho mỗi ngôn ngữ được hỗ trợ: HTML, Java, C#, Perl, PHP, Python và Ruby. Khả năng sử dụng Selenium RC với một ngôn ngữ lập trình bậc cao để phát triển các trường hợp kiểm thử cũng cho phép kiểm thử tự động được tích hợp với một dự án xây dựng môi trường tự động.

❖ **Selenium WebDriver**

Selenium WebDriver là sự kế thừa từ Selenium Remote Control, làm việc trực tiếp với trình duyệt ở mức hệ điều hành, cho phép gửi lệnh trực tiếp đến trình duyệt và xuất ra kết quả.

❖ **Selenium-Grid**

Selenium – Grid Là một hệ thống hỗ trợ người dùng thực thi test script trên nhiều trình duyệt một cách song song mà không cần phải chỉnh sửa test script.

Thực hiện phương pháp kiểm tra phân bô, phối hợp nhiều Selenium RC để có thể thực thi trên nhiều trình duyệt Web khác nhau trong cùng một lúc nhằm giảm thiểu thời gian thực hiện.

6.4.1 Selenium WebDriver

❖ Tiền thân của Selenium WebDriver

Trước khi Selenium WebDriver ra đời và phát triển thì Selenium RC là công cụ chính trong suốt một thời gian dài. Hiện nay, Selenium RC không được sử dụng nhiều như Selenium WebDriver nữa, tuy nhiên người dùng vẫn có thể tiếp tục phát triển các kịch bản kiểm thử với Selenium RC.

Selenium RC là công cụ phục vụ cho các công việc kiểm thử đòi hỏi nhiều hơn việc thao tác với các website trên giao diện. Nó cho phép viết các kịch bản kiểm thử tự động ứng dụng Web với sự hỗ trợ của các ngôn ngữ lập trình như Java, C#, Python, Perl, PHP để tạo ra các trường hợp kiểm thử phức tạp hơn như đọc và viết các tệp tin, truy vấn cơ sở dữ liệu, gửi mail kết quả kiểm thử.

Các thành phần của Selenium RC gồm:

- **Máy chủ Selenium:** Thực hiện phân tích và chạy các lệnh được gửi đến từ ứng dụng cần kiểm thử và các thao tác như HTTP proxy, phân tích và xác minh các thông điệp HTTP, giữa trình duyệt và ứng dụng cần kiểm tra.
- **Các thư viện máy khách:** cung cấp sự hỗ trợ lập trình cho phép chạy lệnh Selenium từ chương trình. Các thư viện máy khách hỗ trợ cho các ngôn ngữ lập trình khác nhau thì khác nhau. Giao diện lập trình là tập các chức năng chạy lệnh Selenium, trong mỗi giao diện có một chức năng lập trình hỗ trợ Selenium.

❖ Đặc trưng của Selenium WebDriver

Selenium WebDriver (hay còn gọi là Selenium 2.0) kế thừa và phát triển từ Selenium IDE, Selenium RC, Selenium Grid. Selenium WebDriver tương tác trực tiếp với trình duyệt mà không cần thông qua bất kỳ trung gian, không giống như Selenium RC phụ thuộc vào một máy chủ.

Tính năng chính trong Selenium 2.0 là việc tích hợp WebDriver API. Ngoài việc giải quyết một số hạn chế trong Selenium RC API, Selenium WebDriver còn được thiết kế để mang đến một giao diện lập trình đơn giản hơn. Nó cho phép sử dụng một trong số các ngôn ngữ lập trình như HTML, Java, .Net, Perl, Ruby... để tạo kịch bản kiểm thử kết hợp với sử dụng các điều kiện, vòng lặp... khiến cho test script trở nên chính xác hơn.

Selenium WebDriver còn được phát triển tốt hơn để hỗ trợ cho các trang Web động do các phần tử trong một trang Web động có thể thay đổi bất cứ lúc nào, ngay cả khi trang

đó không được tải lại. Vì vậy, Selenium WebDriver được phát triển để hỗ trợ quá trình kiểm thử mà không cần phải thực hiện lại khi có thay đổi xảy ra.

6.4.2 Các câu lệnh sử dụng trong Selenium WebDriver

a. Các câu lệnh trình duyệt

Selenium WebDriver có một số các câu lệnh thao tác với trình duyệt như mở, đóng, lấy tiêu đề của trang Web như dưới đây:

- Câu lệnh get

Mục đích: Câu lệnh này sử dụng để mở một trang Web mới trong trình duyệt hiện tại.

Cú pháp: `driver.get(URL);`

Trong đó: URL: Là url để tải trang, nên sử dụng một url đầy đủ

- Câu lệnh lấy tiêu đề getTitle

Mục đích: Câu lệnh này sử dụng để lấy tiêu đề của trang Web hiện tại

Cú pháp: `driver.getTitle();`

- Câu lệnh lấy URL hiện tại `getCurrentUrl`

Mục đích: Câu lệnh này dùng để lấy URL của trang hiện tại đã được tải trên trình duyệt.

Cú pháp: `driver.getCurrentUrl();`

- Câu lệnh lấy source của trang Web `getPageSource`

Mục đích: Câu lệnh này dùng để lấy source của trang được tải cuối cùng.

Cú pháp: `driver.getPageSource();`

b. Các câu lệnh WebElement

Để tương tác với một trang Web, cần phải xác định vị trí của phần tử trên trang Web, WebDriver cung cấp 2 phương thức “Find Element” and “Find Elements” để xác định vị trí của phần tử trên trang Web.

➤ Phương thức “Find Element” và “Find Elements”

Sự khác nhau giữa phương thức “Find Element” và “Find Elements” là trả về đối tượng WebElement, nếu không ném một ngoại lệ và trả về một danh sách WebElement, có thể sẽ trả về danh sách rỗng nếu không có phần tử DOM phù hợp với truy vấn. Phương thức “Find” lấy vị trí hoặc đối tượng truy vấn gọi bằng phương thức “By”

- Tìm phần tử bằng ID: By ID

Mục đích: Tìm vị trí của phần tử bằng ID, nếu tìm được id phù hợp sẽ trả về vị trí của phần tử, nếu không có phần tử phù hợp với id sẽ xuất hiện NoSuchElementException

Cú pháp: `driver.findElement(By.id(""));`

- Tìm phần tử bằng Name (By Name)

Mục đích: Tìm vị trí của phần tử bằng name, nếu tìm được giá trị thuộc tính name phù hợp sẽ trả về vị trí của phần tử, nếu không có phần tử phù hợp với thuộc tính name sẽ xuất hiện NoSuchElementException.

Cú pháp: `driver.findElement(By.name(""));`

- Tìm phần tử bằng Class Name (className)

Mục đích: Tìm phần tử dựa trên giá trị của thuộc tính “class”.

Cú pháp: `driver.findElement(By.className(""));`

- Tìm phần tử bằng Link Text

Mục đích: Tìm phần tử của thẻ a bằng tên của link ,

Cú pháp: `driver.findElement(By.tagName(""));`

c. Các câu lệnh điều hướng trình duyệt

- Câu lệnh forward

Mục đích: Lệnh này dùng để đi đến trang tiếp theo, giống với nút forward trên trình duyệt.

Cú pháp: `driver.navigate().forward();`

- Câu lệnh back

Mục đích: Lệnh này dùng để quay về trang trước, giống với nút back trên trình duyệt.

Cú pháp: `driver.navigate().back();`

- Câu lệnh refresh

Mục đích: Lệnh này dùng để làm mới trang hiện tại .

Cú pháp: `driver.navigate().refresh();`

d. Các lệnh switch

Một số trang Web có nhiều frames hoặc nhiều cửa sổ. Selenium WebDriver gán id cho mỗi cửa sổ ngay khi đối tượng WebDriver được khởi tạo. Id này được gọi là cửa sổ xử lý.

Selenium sử dụng id duy nhất này để điều khiển nhiều cửa sổ. Trong đó, mỗi cửa sổ có một id duy nhất, do đó Selenium có thể phân biệt được khi nó được chuyển sang điều khiển một cửa sổ khác.

Dưới đây là một số câu lệnh switch:

- Câu lệnh getWindowHandle

Mục đích: Lệnh này dùng để lấy cửa sổ xử lý (window handle) của cửa sổ hiện tại.

Cú pháp: `driver.getWindowHandle();`

- Câu lệnh getWindowHandles

Mục đích: Lệnh này dùng để lấy cửa sổ xử lý (window handle) của tất cả các cửa sổ hiện tại.

Cú pháp: `driver.getWindowHandles();`

- Câu lệnh Switch To Window

Mục đích: Lệnh này dùng hỗ trợ di chuyển giữa các cửa sổ khác nhau thông qua tên của chúng bằng cách sử dụng phương thức “switchTo”.

Cú pháp: `driver.switchTo().window("windowName");`

e. Các câu lệnh wait

- Câu lệnh implicitlyWait

Mục đích: Đợi một thời gian nhất định trước khi ném một ngoại lệ khi không thể tìm thấy các phần tử trên trang web.

Cú pháp: `driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);`

- Câu lệnh pageLoadTimeout

Mục đích: Thiết lập giá trị thời gian để chờ đợi cho trang Web hoàn thành tải(loadding) trước khi ném một lỗi.

Cú pháp: `driver.manage().timeouts().pageLoadTimeout(100, SECONDS);`

- Câu lệnh setScriptTimeout

Mục đích: Thiết lập giá trị thời gian chờ đợi một kịch bản(script) không đồng bộ để kết thúc việc thực hiện trước khi ném một lỗi. Nếu thời gian chờ là tiêu cực, sau đó kịch bản sẽ chạy vô hạn.

Cú pháp: `driver.manage().timeouts().setScriptTimeout(100,SECONDS);`

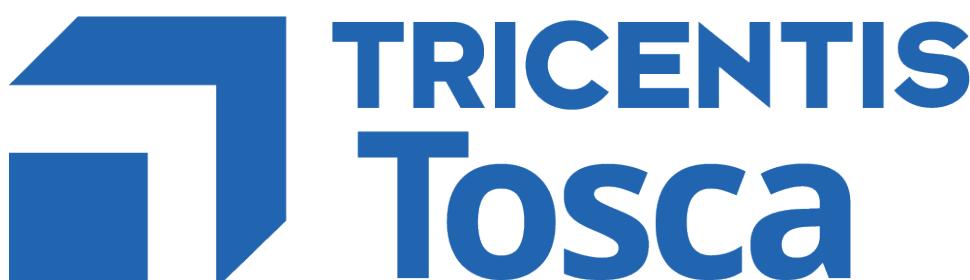
- Câu lệnh sleep

Mục đích: Câu lệnh này hiếm khi được sử dụng vì nó luôn luôn buộc các trình duyệt chờ đợi một thời gian cụ thể.

Cú pháp: `thread.sleep(1000);`

6.5 Công cụ hỗ trợ kiểm thử API

Tricentis Tosca



- Là nền tảng thử nghiệm liên tục cho Agile và DevOps
- Đặc điểm nổi bật
 - Hỗ trợ nhiều giao thức như HTTPS JMS, AMQP, Rabbit MQ, TIBCO, REST, IBM MQ, NET TCP.
 - Hỗ trợ tích hợp vào chu trình Agile và DevOps.
 - Tối ưu hóa việc sử dụng và bảo trì với tự động hóa thử nghiệm dựa trên mô hình.
 - API có thể được sử dụng trên thiết bị di động, đa nền tảng.

- Giảm thiểu thời gian kiểm thử hồi quy
- Mất phí

Apigee



- Là một công cụ kiểm thử API trên đám mây, cho phép người dùng đo lường và kiểm tra hiệu suất API, hỗ trợ và xây dựng API bằng các trình soạn thảo khác như Swagger
- Đặc điểm nổi bật
 - Được viết bằng Javascript.
 - Cho phép kiểm soát thiết kế, triển khai, và chia tỷ lệ API
 - Xác định các vấn đề về hiệu suất bằng cách theo dõi lượng API, tỷ lệ, thời gian phản hồi.
 - Dễ dàng tạo ra các proxy API từ đặc tả API mở và triển khai chúng trong đám mây
 - Mô hình triển khai đám mây, tại chỗ hoặc kết hợp trên một mã duy nhất.
 - Được xây dựng với mục đích kinh doanh kỹ thuật số và các ứng dụng dựa trên thiết bị di động nhiều tài nguyên.
 - Mất phí hàng tháng, có bản dùng thử miễn phí

Postman

- Dễ dàng sử dụng REST client.
- Giao diện sử dụng đơn giản, dễ thao tác.
- Hỗ trợ chạy đa nền tảng (MacOS, Windows, Linux,...)
- Cho phép thay đổi header của request
- Hỗ trợ nhiều định dạng kết quả nhận về như text, hình ảnh, xml, json,...

Giới thiệu công cụ kiểm thử API Postman

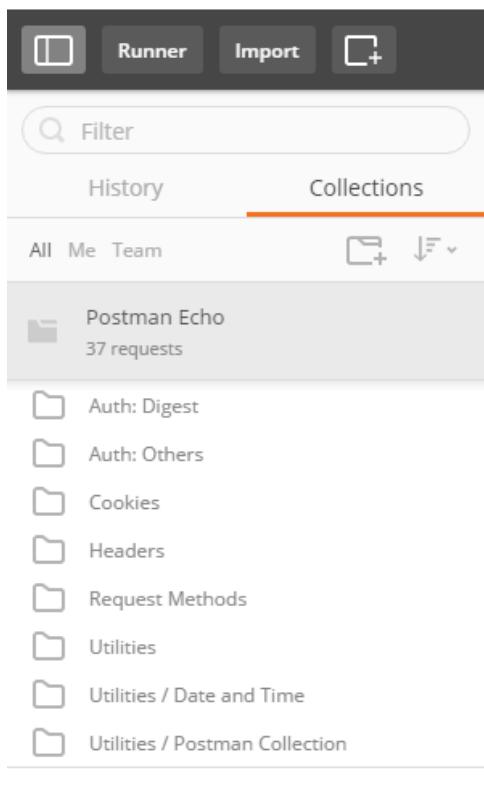


Postman là gì?

- Postman là 1 công cụ nền tảng cho việc phát triển API. Postman có thể sử dụng để thiết kế, xây dựng, và kiểm thử các giao diện lập trình ứng dụng (API).
- Postman hỗ trợ toàn bộ các phương thức HTTP (GET, POST, PUT, DELETE, ...)

Các thành phần chính trong giao diện của Postman

- Bộ sưu tập (Collections): Lưu trữ thông tin các API theo folder hoặc theo thời gian



- Nội dung API: Hiển thị nội dung chi tiết API và các phần hỗ trợ giúp thực hiện kiểm thử API.

- Environments (Môi trường): Chứa các thông tin về môi trường
- Request: Phần chứa thông tin của chính của 1 API request, bao gồm URL, Method, Headers, Body, ...
- Response: Phần chứa thông tin trả về của API

6.6 Công cụ hỗ trợ kiểm thử hiệu năng

Kiểm thử hiệu năng được thực hiện để xác định hệ thống thực hiện một khối lượng công việc cụ thể nhanh thế nào. Nó cũng có thể dùng để xác nhận và xác minh những thuộc tính chất lượng khác của hệ thống như: khả năng mở rộng, độ tin cậy, sử dụng tài nguyên.

Load testing là khái niệm chủ yếu của việc kiểm thử mà có thể tiếp tục hoạt động ở một mức tải cụ thể, cho dù đó là một lượng lớn dữ liệu hoặc lượng lớn người sử dụng.

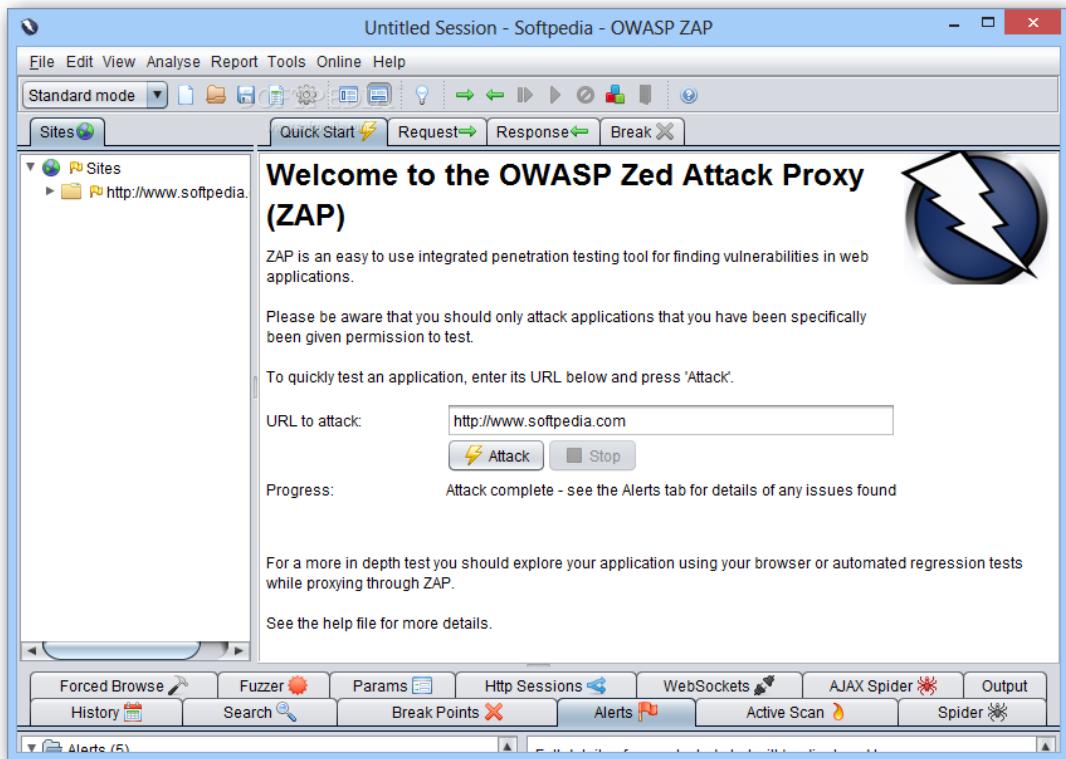
Volume testing là một cách kiểm tra chức năng. Stress testing là một cách để kiểm tra tính tin cậy. Load testing là một cách để kiểm tra hiệu năng. Đây là một số thỏa thuận về các mục tiêu cụ thể của load testing. Những thuật ngữ như load testing, performance testing, reliability testing, và volume testing thường sử dụng thay thế cho nhau.

Một số công cụ kiểm tra hiệu năng:

1. **Loadrunner:** là công cụ kiểm tra tải / căng thẳng cho các trang web và các ứng dụng khác, hỗ trợ một loạt các môi trường ứng dụng, nền tảng, và cơ sở dữ liệu. Link trang chủ: <http://www8.hp.com>
2. **Pylot:** là một công cụ mã nguồn mở được chạy thử nghiệm tải HTTP để thử nghiệm hiệu suất và khả năng mở rộng các dịch vụ web. Nó tạo ra đồng thời tải, xác minh trả lời máy chủ, và tạo ra các báo cáo với số liệu.
3. **Apache JMeter :** có thể được sử dụng để thử nghiệm hiệu suất cả về tài nguyên tĩnh và động. Link trang chủ: <http://jmeter.apache.org/>
-

6.7 Công cụ hỗ trợ kiểm thử bảo mật

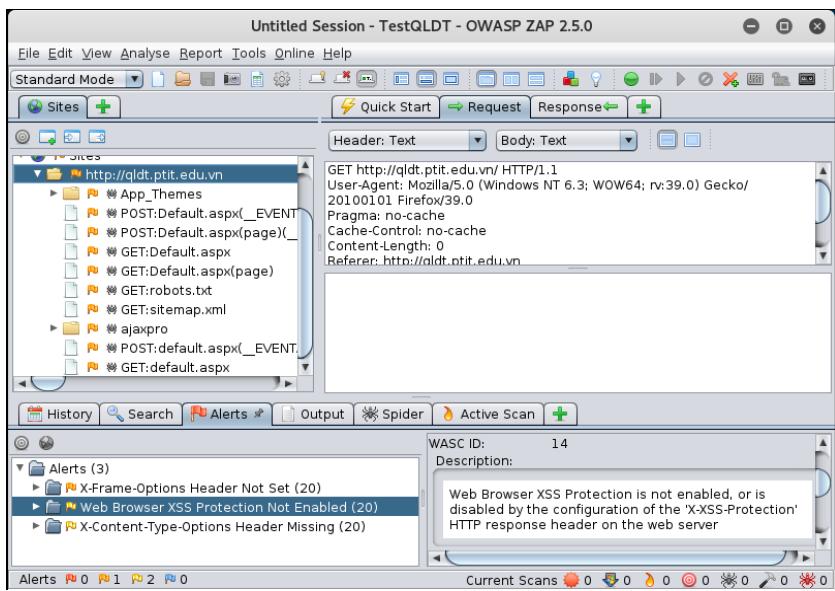
Công cụ kiểm thử bảo mật Zed Attack Proxy (ZAP) là công cụ được tích hợp bởi nhiều công cụ pentest có chức năng khác nhau để kiểm thử bảo mật do tổ chức Owasp phát triển. ZAP có chức năng tự động quét lỗi và cho phép người sử dụng có thể thực hiện nhiều tác vụ nhằm dò tìm điểm yếu trên ứng dụng web.



Owasp - Open Web Application Security Project là 1 dự án mở về bảo mật ứng dụng web, dự án là sự cố gắng chung của cộng đồng với mục đích giúp các doanh nghiệp có thể phát triển, mua và bảo trì các ứng dụng web một cách an toàn

❖ Những tiện ích nổi bật của ZAP

- Intercepting Proxy : Bản chất ZAP là một proxy chặn giữa người dùng và ứng dụng web
- Automated scanner: Quét lỗi tự động
- Passive scanner: Quét lỗi thụ động
- Brute Force scanner: Cho phép tìm cây thư mục, tính năng này là sự tích hợp của công cụ Dir Buster mà mình đã giới thiệu trước đó.
- Spider: Tìm kiếm các url trong website
- Fuzzer: Tích hợp tiện ích JbroFuz.
- Port scanner: Quét lỗi cổng
- Dynamic SSL certificates: tự tạo một chứng chỉ gốc, thực hiện MiTM để decrypt các traffic mã hóa bởi SSL.
- Report: cung cấp output ra report dạng HTML và XML



❖ Các lỗi bảo mật ZAP có thể phát hiện được

Do ZAP là công cụ được Owasp phát triển nên nó có thể tìm được các lỗi mà tổ chức này thống kê được. Và dưới đây là 10 lỗi bảo mật mới nhất dựa theo số liệu mới nhất của OWASP năm 2013 và 2017

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

Ngoài ra còn một số các nhóm công cụ kiểm thử tự động khác như:

- Công cụ quản lý sự cố – *Incident Management Tools*.
- Công cụ quản lý cấu hình – *Configuration Management Tools*.
- Công cụ bảo hiểm đo lường – *Coverage Measurement Tools*.
- Công cụ giám sát – *Monitoring Tools*.
- Công cụ kiểm thử cơ sở dữ liệu – *Database Testing Tools*.
- Công cụ kiểm thử bảo mật – *Security Testing Tools*.

Công cụ kiểm thử tĩnh – *Static Testing Tools*.

Chương 7: Các tiêu chuẩn trong quản lý Đảm bảo chất lượng phần mềm

7.1 Giới thiệu

Các tiêu chuẩn chứng chỉ phần mềm lớn dựa trên sự đánh giá nội dung hay mức độ quan trọng. Phạm vi của các tiêu chuẩn chứng chỉ dựa trên mục đích của các chứng chỉ .Đó là:

- Cho phép tổ chức phát triển phần mềm chứng minh năng lực phù hợp để đảm bảo sản phẩm phần mềm của mình hay các dịch vụ bảo trì tuân theo đúng yêu cầu chất lượng.
- Giống như sự giao ước cơ bản của khách hàng và nhà cung cấp nhằm đánh giá hệ thống quản lý chất lượng của nhà cung cấp , điều này có thể được hoàn thành bởi việc kiểm toán chất lượng của khách hàng .Việc kiểm toán này dựa trên các yêu cầu chứng chỉ tiêu chuẩn .
- Hỗ trợ tổ chức phát triển phần mềm để cải thiện hiệu năng của hệ thống quản lý chất lượng và nâng cao sự hài lòng của khách hàng thông qua việc tuân theo các tiêu chuẩn yêu cầu

Phạm vi của các tiêu chuẩn đánh giá cũng xác định trên mục đích của việc đánh giá, đó là :

- Các tổ chức phát triển và duy trì phần mềm sẽ sử dụng như 1 công cụ cho việc đánh giá bản thân về kỹ năng của họ trong đảm nhận các dự án phát triển phần mềm
- Như 1 công cụ để cải thiện quá trình phát triển và duy trì , các tiêu chuẩn có thể đưa ra các chỉ dẫn cho việc cải thiện tiến trình .
- Giúp các tổ chức chi trả hay đầu tư xác định được tiềm năng của nhà cung cấp
- Hướng dẫn tập luyện đánh giá bằng cách mô tả chất lượng và các khóa học lập trình.

7.2 Đảm bảo chất lượng phần mềm trong các chuẩn của ISO

a. ISO là gì?

- ISO (International Organization for Standardization) là tổ chức tiêu chuẩn hóa quốc tế, là cơ quan thiết lập tiêu chuẩn quốc tế bao gồm các đại diện từ các tổ chức quốc gia. Tổ chức được thành lập vào năm 1947, tổ chức này đã đưa ra các tiêu chuẩn thương mại, sản xuất và thông tin trên phạm vi toàn thế giới. ISO có trụ sở ở Geneva (Thụy Sĩ) và là một tổ chức Quốc tế chuyên ngành có các thành

viên là các cơ quan tiêu chuẩn Quốc gia của 111 nước. Tuỳ theo từng nước, mức độ tham gia xây dựng các tiêu chuẩn ISO có khác nhau

b. Một số bộ tiêu chuẩn của ISO

- ISO 9000 Hệ thống quản lý chất lượng - Nguyên tắc cơ bản và từ vựng, bao gồm những tiêu chuẩn cơ bản về hệ thống quản lý chất lượng đang chứa đựng những ngôn ngữ cốt lõi của bộ tiêu chuẩn ISO 9000.
- ISO 9001 Hệ thống quản lý chất lượng - Các yêu cầu dự kiến cho sử dụng ở bất kì tổ chức mà thiết kế, phát triển, sản xuất, lắp đặt hay phục vụ cho bất kì 1 sản phẩm nào hoặc cung cấp bất kì kiểu dịch vụ nào. Nó đem lại số lượng yêu cầu mà các tổ chức cần phải hoàn thành nếu như nó làm vừa lòng khách hàng thông qua những sản phẩm và dịch vụ hoàn chỉnh mà làm thỏa mãn mong chờ của khách.

Một số phiên bản của ISO 9001:

- ISO 9001:1987 : Quản lý chất lượng - Mô hình đảm bảo chất lượng trong thiết kế triển khai, sản xuất, lắp đặt và dịch vụ kỹ thuật.
- ISO 9001:1994 : Tiêu chuẩn Việt Nam tương đương: TCVN ISO 9001:1996 Quản lý chất lượng - Mô hình đảm bảo chất lượng trong thiết kế, triển khai, sản xuất, lắp đặt và dịch vụ kỹ thuật.
- ISO 9001:2000 : Tiêu chuẩn Việt Nam tương đương: TCVN ISO 9001:2000 Quản lý chất lượng - Các yêu cầu.
- ISO 9001:2008 : Tiêu chuẩn Việt Nam tương đương: TCVN ISO 9001:2008 Quản lý chất lượng - Các yêu cầu. Đây là phiên bản hiện hành của ISO 9001
- ISO 9004 Hệ thống quản lý chất lượng - Hướng dẫn cải tiến. Nó đem lại cho bạn nhiều lời khuyên về việc bạn có thể làm gì để nổi bật hệ thống đã hoàn thiện. Tiêu chuẩn này đã được tuyên bố một cách cụ thể rằng nó sẽ dẫn đường cho việc thực thi một cách đầy đủ.

7.3 Đảm bảo chất lượng phần mềm trong các chuẩn CMM, CMMI

Giới thiệu CMM/CMMI

- CMMI (Capability Maturity Model Integration) là phiên bản kế tiếp của CMM. Cá CMM và CMMI đều được nghiên cứu bởi viện phần mềm (SEI: Software Engineering Institute) Mỹ, trường đại học Carnegie Mellon. CMM được phát triển từ năm 1987 đến năm 1997. Năm 2000 CMMI phiên bản 1.02 được đưa ra thị trường. Phiên bản CMMI 1.2 ra đời vào 8/2006. Hiện tại phiên bản mới nhất là phiên bản CMMI 1.3 ra đời vào năm 2010. Đến 1/2013 toàn bộ sản phẩm của CMMI được chuyển từ SEI tới tổ chức CMMI. 1 tổ chức mới được thành lập tại đại học Carnegie Mellon.
- Mô hình CMMI được sử dụng như 1 công cụ để đánh giá khả năng của các tổ chức khi họ tiến hành 1 dự án phần mềm theo hợp đồng. Tuy CMMI được thiết kế để đánh giá quá trình phát triển phần mềm nhưng nó đã và đang được sử dụng như 1 mô hình chung cho các công ty về công nghệ thông tin.

7.3.1.1 CMM/CMMI là gì?

- Theo viện nghiên cứu SEI của Mỹ, CMM là một phương pháp tiếp cận cải tiến quy trình cung cấp cho các tổ chức với yếu tố thiết yếu của quá trình mà hiệu quả cuối cùng là cải thiện hiệu suất. Có thể nói, CMM là 1 bộ khung những chuẩn đề ra cho 1 tiến trình sản xuất phần mềm hiệu quả, bao gồm việc mô tả các nguyên tắc, các thực tiễn, lịch trình ... cho 1 dự án phần mềm.
- Năm 2000, mô hình CMMI (Capability Maturity Intergration Integration) là phiên bản mới của CMM thay thế CMM. Đây là mô hình được phát triển nhằm cung cấp những tiêu chuẩn hướng dẫn, kinh nghiệm thực tế để phát triển, cải tiến các quy trình của 1 tổ chức phần mềm. Hay nói cách khác CMMI là chuẩn quản lý quy trình chất lượng phần mềm, là khuôn khổ cho các quy trình quản lý chất lượng phần mềm.
- Mô hình CMM/CMMI gồm 5 levels và 18 KPAs (vùng quy trình quan trọng – Key Process Area):

Level 1: Khởi đầu (initial) : không có KPAs nào
Đây là điểm khởi đầu để sử dụng 1 quy trình mới.

Level 2 : Lặp (Repeatable) : có 6 KPAs.
Quy trình được lặp lại nhiều lần.

Level 3: Xác lập (Defined) : có 7 KPAs.

Quy trình được xác lập / xác nhận như 1 quy trình doanh nghiệp chuẩn.

Level 4 : Kiểm soát (Quantitatively managed) : có 2 KPAs.

Tiến hành kiểm soát và đo lường quy trình sản xuất phần mềm.

Level 5 : Tối ưu (Optimizing) : có 3 KPAs.

Kiểm soát quy trình bao gồm việc nhắc kỹ để cải tiến/ tối ưu hóa quy trình.

- CMMI đưa ra các mô hình khác nhau cho từng mục đích sử dụng có đặc điểm riêng:
 - CMMI – SW (CMMI - Software Engineering): dành cho công nghệ phần mềm.
 - CMMI – SE/SW (Capability Maturity Model Integration for Systems Engineering and Software Engineering) : dành cho công nghệ hệ thống và phần mềm.
 - CMMI – SE/SW/IPPD (CMMI for Systems Engineering and Software Engineering, Integrated Product and Process Development): dành cho công nghệ hệ thống và công nghệ phần mềm với việc phát triển sản phẩm và quy trình tích hợp.

7.3.1.2 CMMI được dùng để làm gì?

- Các công ty, doanh nghiệp sử dụng mô hình CMMI để hỗ trợ việc xác định cải tiến quy trình để xây dựng hệ thống và phát triển quy trình chất lượng phần mềm và phát triển quy trình.
- Công ty sử dụng quy trình này để phát triển, duy trì các sản phẩm và dịch vụ để làm chuẩn cho chính họ để chống lại các công ty khác.
- CMMI không thể nhanh chóng phù hợp với tất cả các công ty mà cũng không ảnh hưởng đến sự phát triển của công ty đó. Và quá trình cải thiện dự án đòi hỏi có nhiều kiến thức và nguồn lực nên các công ty lớn có thể có được kết quả tốt hơn từ CMMI.

- SEI không cấp giấy chứng nhận cho bất cứ CMMI nào. SEI chỉ cấp giấy phép hoạt động và cho phép các nhà thẩm định hàng đầu tiến hành đánh giá.

Lợi ích của CMMI:

- Lợi ích chung:
 - Cải tiến năng lực bằng cách nâng cao kiến thức và kỹ năng của lực lượng lao động.
 - Hướng động lực của cá nhân vào với mục tiêu của tổ chức.
 - Giảm bớt các công việc lặp lại.
 - Năng lực phát triển phần mềm là thuộc tính của tổ chức chứ không phải của 1 vài cá thể.
- *Đối với doanh nghiệp :*
 - Có thêm những quyết định rút khoát rõ ràng trong việc quản lý.
 - Có tầm nhìn về phạm vi trong vòng đời phát triển phần mềm cũng như các hoạt động đảm bảo sản phẩm hoặc dịch vụ đáp ứng được nhu cầu của khách hàng.
 - Thực hiện đầy đủ và thuần thục với cách thức làm việc.
 - Kết hợp những gì đã có được và thêm vào được những thực hành tốt nhất.
- *Đối với người quản lý (người thực hiện):*
 - Môi trường làm việc tốt.
 - Vạch rõ vai trò, trách nhiệm của từng vị trí công việc.
 - Đánh giá đúng năng lực.
 - Liên tục phát triển những kỹ năng cốt yếu.
- Đạt được giá trị lớn nhất thông qua các quy trình của mô hình CMMI liên quan đến ba yếu tố then chốt :
 - Hiểu được các giải pháp mới
 - Không sử dụng mô hình này một cách máy móc, dập khuôn mà hãy làm cho chúng phù hợp với môi trường.
 - Đi liền với những thay đổi lâu dài để các mô hình CMMI có thể tạo ra sự khác biệt

7.4 Cấu trúc và các level của CMMI :

7.4.1 Cấu trúc của CMMI :

Cấu trúc của CMMI gồm các thành phần :

- Maturity Levels (mức độ trưởng thành): Là các lớp cơ cấu tổ chức với điều kiện là một chuỗi các quy tắc được định ra cần thiết để liên kết trong quy trình phát triển phần mềm. Nó rất quan trọng đối với các tổ chức, công ty... cần phát triển năng lực làm việc, khối lượng công việc, công nghệ hoặc công cụ trong hoạt động. Nó giúp các dự án, các nhóm, các công ty định hướng được việc trình bày hợp lý trong các lựa chọn.
- Key process areas – KPA (miền quy trình): mỗi 1 mức độ trưởng thành bao gồm 1 vài miền quy trình. Miền quy trình là 1 nhóm các hoạt động có liên quan với nhau khi thực hiện trung để hoàn tất mục tiêu.
- Goals: Mục tiêu của một phạm vi miền quy trình (KPA) tóm tắt tình trạng phải tồn tại của phạm vi miền quy trình thực hiện đầy đủ với kết quả và quá trình thực hiện dài lâu. Quy mô của mục tiêu đã hoàn thành là cho biết năng lực của tổ chức thiết lập trên các cấp bậc của sự thuận thực.
- Common Features (các tính năng chung) : Những điểm đặc trưng bao gồm thực hành và thể chế hóa phạm vi miền quy trình KPA. Nó bao gồm 5 dạng là Commitment to Perform (giao phó đến thực hiện), Ability to Perform (khả năng đến thực hiện), Activities performed (các hoạt động đã thực hiện), Measurement and Analysis (đo lường và phân tích), và Verifying Implementation (thực hiện thẩm tra).
- Key Practices: Khóa thực hiện mô tả các yếu tố của cơ sở hạ tầng và thực hiện góp phần tạo những hiệu quả thực sự trong việc thực hiện và thể chế hóa miền quy trình KPA.

7.4.2 Các level của CMMI:

- CMMI gồm 5 levels và 18 KPAs (vùng quy trình quan trọng – Key Process Area). Trong đó, mỗi level đều tuân theo 1 chuẩn ở mức độ cao hơn. Muốn đạt được chuẩn cao hơn thì các chuẩn của level trước phải thỏa mãn. Mỗi level đều có đặc điểm chú ý quan trọng của nó mà các doanh nghiệp phải đáp ứng được.

Level 1 : khởi đầu (Initial) không có KPAs nào.

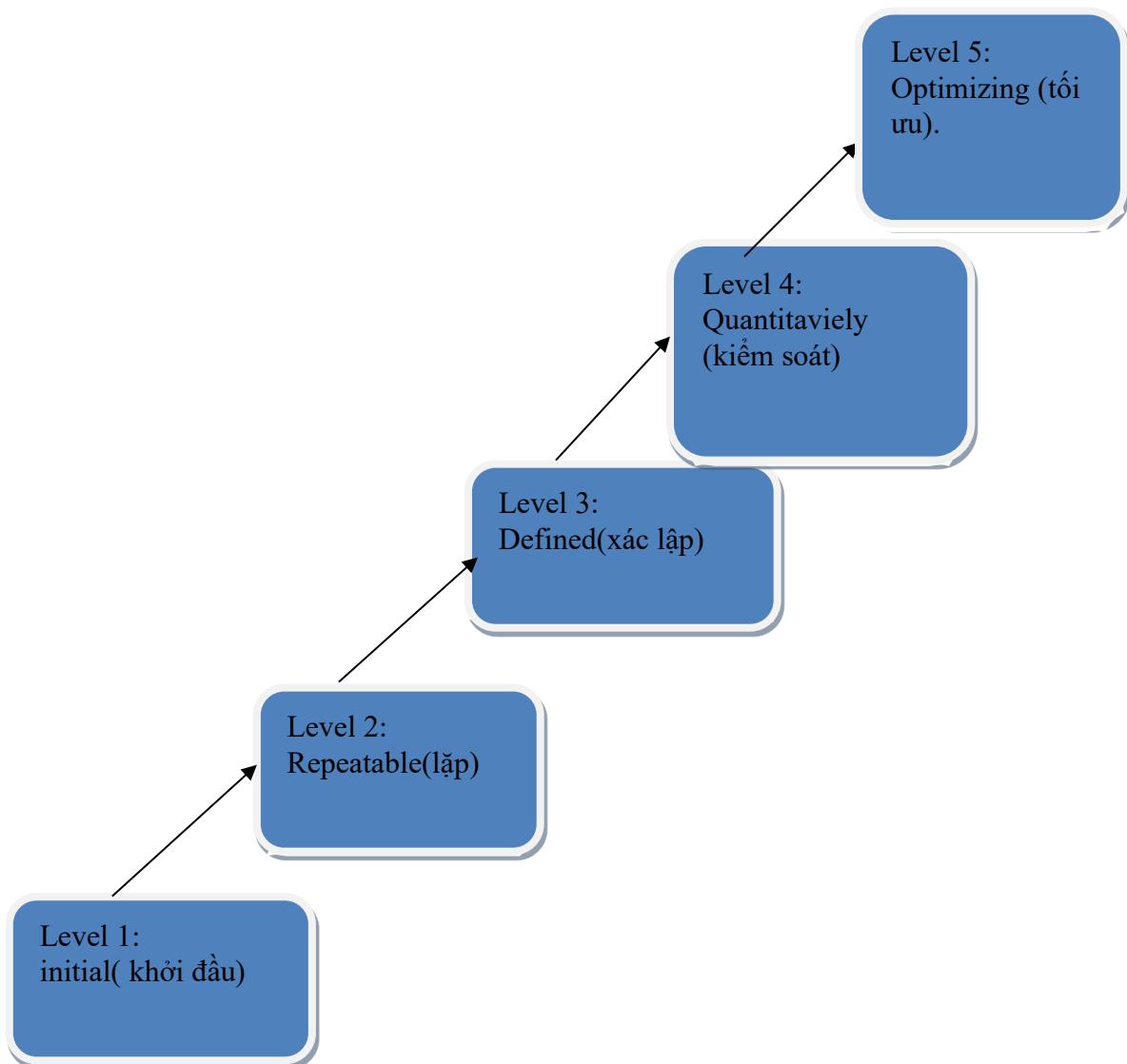
Level 2 : Lặp (Repeatable) có 6 KPAs.

Level 3 : Xác lập (Defined) có 7 KPAs.

Level 4: Kiểm soát (Quantitatively Managed) có 2 KPAs.

Level 5 : Tối ưu (Optimizing) có 3 KPAs.

- 18 KPAs của CMMI đều có 5 thuộc tính (chức năng) chung, trong đó có các quy tắc về Key Practice (khóa thực hiện) là những hướng dẫn về các thủ tục (Procedure), quy tắc (Polities) và hoạt động (Activites) của từng KPA.



Level 1 : khởi đầu (initial)

- Đây là bước khởi đầu của CMMI, quá trình này thường hỗn độn và hỗn loạn. Tổ chức thường không cung cấp một môi trường ổn định. Vì vậy, mọi công ty hay doanh nghiệp, cá thể đều có thể đạt được vì level này CMMI chưa yêu cầu bắt kì tính năng nào.
- Đặc điểm :
 - Các hoạt động của lực lượng lao động được quan tâm hàng đầu nhưng được thực hiện một cách haphazard.
 - Đào tạo quản lý nhân lực nhỏ bé chủ yếu dựa vào kinh nghiệm cá nhân.
 - Người quản lý mong bộ phận nhân sự điều hành và kiểm soát các hoạt động của lực lượng lao động.
 - Các hoạt động của lực lượng lao động không đáp ứng ngay mà không cần phân tích ảnh hưởng.
 - Doanh số thường xuyên thay đổi : nhân viên không trung thành với tổ chức.

Đây là điểm khởi đầu cho một quy trình mới. Nên các tổ chức không có các tiến hành cấu trúc. Quy trình phần mềm có thể tự phát triển rất hỗn độn và không dự đoán trước được. Chất lượng của phần mềm cũng không thể biết được. Các doanh nghiệp sản xuất ra phần mềm hoặc dịch vụ thường bị vượt quá ngân sách dự thảo và kế hoạch của dự án.

Level 2 : Lặp (Repeatable)

- Ở level 2 thì người quản lý phải thiết lập được các nguyên tắc cơ bản và quản lý tất cả các hoạt động diễn ra. Họ phải có trách nhiệm phải quản lý đội ngũ của mình. Chúng gồm có :
 - Requirement Management (Lấy yêu cầu khách hàng, quản lý các yêu cầu đó)
 - Software Project Planning (Lập các kế hoạch cho dự án)
 - Project Monitoring and control (Theo dõi giám sát và kiểm soát dự án)
 - Measurement and analysis (Đo lường và phân tích)
 - Software Configuration Management (Quản trị cấu hình sản phẩm => đúng yêu cầu của khách hàng không)

- Process and product quality assurance (đảm bảo quá trình và chất lượng của sản phẩm)
- Mức 2, sẽ có những đặc điểm đặc trưng sau :
 - Goal (mục tiêu): các hoạt động và những đề xuất của một dự án phần mềm phải được lên kế hoạch và viết tài liệu đầy đủ.
 - Commitment (đề xuất/xem xét): dự án phải tuân thủ theo các qui tắc của tổ chức khi hoạch định.
 - Ability (Khả năng): Việc thực hiện lập kế hoạch cho dự án phần mềm phải là bước thực hiện từ rất sớm khi dự án được bắt đầu.
 - Measument (Đo lường): Sự đo lường luôn được thực thi và sử dụng chúng ta luôn có thể xác định và kiểm soát được tình trạng các hoạt động trong tiến trình thực hiện dự án.
 - Verification (Kiểm chứng): Các hoạt động khi lập kế hoạch dự án phải được kiểm tra lại của các quản lý cấp cao.
- Để đạt được level 2, một tổ chức phải đạt được tất cả các mục tiêu cụ thể của level 2. Dự án của tổ chức phải đảm bảo rằng các yêu cầu được quản lý và các quy trình được lên kế hoạch, thực hiện, đánh giá và kiểm soát.

Ở level này, tất cả các yêu cầu, quy trình, sản phẩm đều được quản lý. Tại mọi thời điểm được xác định đều có thể thấy được tình trạng của dự án.

Các quy trình quản lý dự án đã được thiết lập được giá thành, kế hoạch cũng như chức năng. Sản phẩm phải đáp ứng được yêu cầu, quy định, tiêu chuẩn và mục tiêu.

Các KPAs của nó cần chú trọng tới các thành phần:

- Chế độ đài ngộ
- Đào tạo
- Quản lý thành tích
- Phân công lao động
- Thông tin giao tiếp
- Môi trường làm việc

Để từ level 1 tiến tới level 2 ta cần có những điều kiện sau :

Trước hết ta cần phải thỏa mãn các điều kiện ở level 1. Cần chú ý tới các yếu tố sau :

- Môi trường làm việc:
 - Đảm bảo điều kiện làm việc
 - Tạo hứng thú trong công việc
 - Không bị ảnh hưởng, gây mất tập trung bởi các yếu tố khác
- Thông tin: Xây dựng cơ chế truyền thông tin suốt từ trên xuống và ngược lại nhằm giúp các cá nhân trong tổ chức chia sẻ thông tin, kiến thức, kinh nghiệm, các kỹ năng giao tiếp phối hợp và làm việc hiệu quả.
- Xây dựng đội ngũ nhân viên: Ngay từ khâu tuyển dụng, cần lựa chọn kỹ càng, định hướng, thể chế hóa quy trình tuyển dụng.
- Quản lý thành tích: Cần đẩy mạnh thành tích, công nhận năng lực, thành tích bằng cách thiết lập các tiêu chí khách quan để đánh giá và liên tục khuyến khích khả năng làm việc, tập trung phát triển sự nghiệp, xây dựng các mục tiêu tiếp theo.
- Đào tạo : Không chỉ đào tạo các kiến thức chuyên môn phục vụ cho dự án mà còn mở rộng đào tạo các kỹ năng then chốt, cần thiết như kỹ năng làm việc nhóm, kỹ năng quản lý... nhằm tạo cơ hội cho nhân viên phát huy hết khả năng của mình và có cơ hội hỏi, phát triển bản thân.
- Chế độ đai ngộ: Hoạch định chiến lược đai ngộ, thu thập ý kiến lực lượng lao động và công bố công khai. Chế độ đai ngộ cần tập trung vào việc trả lương cho công nhân viên dựa vào vai trò, vị trí của họ (Position), Con người (Person) – thái độ và tác phong làm việc và thành tích (Performance) mà họ đạt được, công hiến cho tổ chức. Đưa ra được chính sách lương, thưởng, phụ cấp các quyền lợi khác để khuyến khích các cá nhân dựa trên sự đóng góp của họ và cấp độ phát triển của toàn tổ chức.

Level 3 : Xác lập (Defined)

- Các vùng tiên trình chủ chốt ở mức 3 nhằm vào cả hai vấn đề về dự án và tổ chức. Các tiêu chuẩn, quy trình, thủ tục cho 1 dự án được thiết kế riêng từ các tổ chức với các quy trình tiêu chuẩn cho phù hợp với 1 dự án cụ thể. Quy trình phần mềm cho các hoạt động quản lý cũng như sản xuất được tài liệu hóa và tích hợp vào quy trình phần mềm chuẩn của nhà sản xuất. Level 3 tập trung vào các yếu tố sau:
 - Requirements Development (Phát triển yêu cầu)
 - Technical Solution (Giải pháp kỹ thuật)
 - Product Integration (Tích hợp sản phẩm)
 - Organizational Process Focus (Tập trung tiến trình tổ chức)
 - Organizational Process Definition (Phân định tiến trình tổ chức)
 - Organizational Training (Tổ chức đào tạo)
 - Integrated Project Management (Quản lý tích hợp dự án)
 - Risk Management (Quản lý rủi ro)
 - Decision Analysis and Resolution (Phân tích quyết định và giải pháp)
- Để đạt được level 3 thì tổ chức phải nắm rõ và hiểu các quy trình được miêu tả trong các tiêu chuẩn, phương pháp. Người quản lý phải biến đổi cải tiến các hoạt động đang diễn ra, cải tiến môi trường làm việc.
- Ở level 3, tổ chức phải đạt được tất cả các mục tiêu cụ thể, yêu cầu cụ thể của cả mức 2 và 3.
- KPA chú trọng tới các yếu tố sau :
 - Văn hóa cá thể
 - Công việc dựa vào kỹ năng
 - Phát triển sự nghiệp
 - Hoạch định nhân sự
 - Phân tích kiến thức và kỹ năng

Từ level 2 đến level 3 các KPA cần thực hiện :

- Phân tích kiến thức và kỹ năng :

Xác định những kỹ năng và kiến thức cần thiết để làm nền tảng cho hoạt động nhân sự. Lĩnh vực phân tích này bao gồm: xác định quy trình cần thiết để duy trì năng lực tổ chức, phát triển và duy trì các kỹ năng và kiến thức phục vụ công việc, dự báo nhu cầu kiến thức và kỹ năng trong tương lai.

- Hoạch định nguồn nhân lực :

Đây là lĩnh vực phối hợp hoạt động nhân sự với nhu cầu hiện tại và trong tương lai ở cả các cấp và toàn tổ chức. Hoạch định nguồn nhân lực có tính chiến lược cùng với quy trình theo dõi chặt chẽ việc tuyển dụng và các hoạt động phát triển kỹ năng sẽ tạo nên thành công trong việc hình thành đội ngũ.

- Phát triển sự nghiệp :

Tạo điều kiện cho mỗi cá nhân phát triển nghề nghiệp và có cơ hội thăng tiến trong nghề nghiệp, bao gồm: xác định các cơ hội, theo dõi sự tiến bộ trong công việc, khuyến khích đạt mục tiêu công việc.

- Các hoạt động dựa trên năng lực :

Ngoài các kỹ năng, kiến thức cốt lõi còn có hoạch định nhân lực, tuyển dụng dựa vào khả năng làm việc, đánh giá hiệu quả qua mỗi công việc và vị trí, xây dựng các chế độ ưu đãi dựa trên hiệu quả đạt được ... đảm bảo các hoạt động của tổ chức đều xuất phát từ mục đích phục vụ cho phát triển nhân lực.

- Văn hóa cá thể :

Tạo lập các chế độ thông tin liên lạc thông suốt, kênh thông tin hiệu quả ở mọi cấp độ trong tổ chức, phối hợp được kinh nghiệm, kiến thức của mỗi người để hỗ trợ lẫn nhau, thúc đẩy nhân viên tham gia ý kiến...

Level 4 : Kiểm soát (Quantitatively Managed)

- Thực hiện đo lường chi tiết quy trình phần mềm và chất lượng sản phẩm. Mục tiêu định lượng về chất lượng và hiệu suất của quá trình được sử dụng như là tiêu chuẩn trong quá trình quản lý.

- Mục tiêu định lượng được dựa trên nhu cầu của khách hàng, người dùng, tổ chức và quá trình dự án. Chất lượng và hiệu suất của dự án sẽ được thống kê và quản lý suốt quá trình của dự án.
- Các vùng tiến trình chủ yếu ở mức 4 tập trung vào thiết lập hiểu biết định lượng của cả quá trình sản xuất phần mềm và các sản phẩm phần mềm đang được xây dựng. Đó là:
 - Quantitative Project Management (Quản lý định lượng dự án)
 - Organizational Process Performance (Tổ chức quá trình hiệu suất)
- Lực lượng lao động làm việc theo đội, nhóm và được quản lý một cách định lượng.
- Các KPA của level 4 chú trọng tới:
 - Chuẩn hóa thành tích trong tổ chức
 - Quản lý năng lực tổ chức
 - Công việc dựa vào cách làm việc theo nhóm
 - Xây dựng đội ngũ chuyên nghiệp
 - Cố vấn
- Để đạt được level 4 thì phải đo lường và chuẩn hóa. Đo lường hiệu quả đáp ứng công việc, chuẩn hóa phát triển các kỹ năng, năng lực cốt lõi.
- Sự khác nhau giữa level 3 và level 4 là khả năng đoán trước việc thực hiện của quy trình. Tại level 4, việc thực hiện của quy trình là có thể kiểm soát bởi tiêu chuẩn và định lượng kỹ thuật và là định lượng có thể đoán trước được. Tại level 3, các quy trình chỉ đoán được chất lượng.

Level 5 : Tối ưu (Optimizing)

- Các quy trình ở mức 5 liên tục được cải tiến dựa trên những ý kiến phản hồi từ việc sử dụng quy trình, thí điểm những ý tưởng quản lý và công nghệ mới. Đó là:
 - Causal Analysis and Resolution (Phân tích nguyên nhân và giải pháp)
 - Organizational Innovation and Deployment (Tổ chức cải tiến và triển khai)

- Level 5 tập trung vào cải thiện liên tục các thực hiện của quá trình trong suốt quá trình phát triển, đổi mới công nghệ. Quá trình phát triển chất lượng đối tượng của tổ chức được thiếp lập, tiếp tục duyệt lại để phản ánh những thay đổi trong đối tượng kinh doanh và sử dụng như tiêu chuẩn trong quá trình phát triển quản lý. Tối ưu quá trình là nhanh nhẹn, thích hợp và đổi mới quyết định dựa trên sự tham gia của lực lượng lao động sắp theo giá trị kinh doanh và đối tượng của tổ chức.
- Ngoài ra, level 5 còn chú trọng vào việc quản lý, phát triển năng lực của nhân viên. Huấn luyện nhân viên trở thành các chuyên gia.
- Khoảng cách giữa level 4 và level 5 là ở dạng của sự biến đổi được trong quá trình. Tại level 4, các quy trình có liên quan tới 1 địa chỉ đặc biệt bởi vì sự biến đổi của quy trình và điều kiện tiêu chuẩn có thể đoán được trước kết quả trong suốt quá trình, có thể thủ tục đoán được kết quả, kết quả này có thể không chỉ đạt được việc thiết lập đối tượng. tại level5, các quy trình liên quan đến nhiều địa chỉ chia sẻ bởi quá trình và thay đổi này để phát triển thực hiện quá trình giành được thiết lập định lượng cho quá trình phát triển đối tượng.

7.4.3 Việt Nam áp dụng CMM/CMMI trong lĩnh vực phần mềm.

Hiện nay, các công ty, doanh nghiệp ở Việt Nam đua nhau lấy được các chứng nhận của ISO và CMMI. Bởi vì họ muốn cải tiến quá trình quản lý trong việc phát triển phần mềm và còn là đi theo nhu cầu của khách hàng hoặc là để quảng cáo. Tất cả cũng đều nhận thức được vấn đề là ISO và CMMI thực sự sẽ đem lại lợi ích dài lâu cho tổ chức, công ty của họ.

Về cơ bản, quản lý chất lượng phần mềm là vấn đề không mới, nhưng nó lại là vấn đề còn yếu của các công ty phần mềm Việt Nam. Một số công ty đã đạt chuẩn quốc tế CMM/CMMI nâng cao năng lực và quản lý chất lượng phần mềm, song chỉ gói gọn trong vài công ty gia công cho nước ngoài.

Và cho đến thời điểm hiện nay, có nhiều doanh nghiệp phần mềm đang áp dụng mô hình đánh giá năng lực sản xuất phần mềm CMM/CMMI ở Việt Nam. Đây là 1 số công ty, doanh nghiệp đã đạt được chứng chỉ này. Ví dụ như FPT Software (CMM ở mức 5), CMC soft đạt yêu cầu tiêu chuẩn Việt Nam ISO 9001 : 2008 và CMMI ở mức 3, PSV (CMMI mức 5), GCS (CMMI mức 4, Global CyberSoft (Việt Nam) đạt chuẩn CMMI Mức 5.

Tài liệu tham khảo

- [1] Murali Chemuturi. *Mastering Software Quality Assurance: Best Practices, Tools and Techniques for Software Developers*. J. Ross Publication Inc., 2011.
- [2]. Neil Walkinshaw, *Software Quality Assurance Consistency in the Face of Complexity and Change*, Springer Nature, 2017.
- [3] Stephen Vance. *Quality Code: Software Testing Principles, Practices, and Patterns*. Addison-Wesley Professional, 2013.
- [4]. Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2016.
- [5] Daniel Galin, *Software Quality Assurance – From Theory to Implementation*, Addison Wesley, 2004.
- [6] Glenford J. Myers. *The Art of Software Testing - Second Edition*. John Wiley & Sons, Inc, Publication, 2004.
- [7] <http://cmmiinstitute.com>
- [8] <http://www.tutorialspoint.com/cmmi/cmmi-maturity-levels.htm>
- [9] [Bộ chuẩn CMMI5 của FPT](#)

Phụ lục

Phụ lục 1: Một số lỗi thường gặp trong phát triển phần mềm

Một số lỗi thường mắc phải trong quá trình phát triển phần mềm, đặc biệt là pha cài đặt được trích từ cuốn sách “The art of software testing” cho ở dưới đây.

Lỗi tham chiếu - Data reference Error

Biến tham chiếu được gán giá trị trước khi được khởi tạo? Đây là lỗi thường xảy ra.

Với các truy cập mảng, chỉ số có nằm trong biên cho phép hay không?

Với các truy cập mảng, chỉ số có nhận giá trị nguyên hay không?

Với các tham chiếu dùng con trỏ hoặc biến tham chiếu, ô nhớ tham chiếu đã được xác định chưa? Đây được gọi là “dangling reference”. Nó xảy ra khi thời gian tồn tại của con trỏ dài hơn thời gian tồn tại của ô nhớ tham chiếu. Trường hợp khác là khi con trỏ tham chiếu tới biến địa phương trong hàm, giá trị của con trỏ được gán cho một tham số bên ngoài, hoặc biến toàn cục, hàm kết thúc (giải phóng ô nhớ được tham chiếu), và sau đó chương trình dùng giá trị con trỏ.

Khi đặt nhiều tên (alias) cho một vùng nhớ với các thuộc tính khác nhau, liệu giá trị dữ liệu trong vùng nhớ này có đúng thuộc tính khi nó được tham chiếu theo một trong các tên? Ví dụ, một chương trình FORTRAN chưa một biến thực A và nguyên B; cả 2 biến đều là alias cho cùng một vùng nhớ sử dụng câu lệnh EQUIVALENCE. Nếu chương trình lưu một giá trị vào A và sau đó truy cập tới biến B; lỗi sẽ xuất hiện vì máy tính sẽ dùng biểu diễn số floating point trong bộ nhớ như một số nguyên.

Liệu giá trị của biến có kiểu hoặc thuộc tính khác với những gì trình biên dịch đặt ra? Tình huống này có thể xảy ra khi một chương trình C, C++ đọc một bản ghi vào bộ nhớ và truy cập tới nó thông qua cấu trúc, tuy nhiên biểu diễn vật lý của bản ghi khác với định nghĩa cấu trúc.

Vấn đề về địa chỉ? liệu đơn vị của vùng nhớ được cấp phát trên máy tính đang chạy có nhỏ hơn đơn vị của vùng nhớ được định nghĩa. Ví dụ, với một số môi trường, xâu bit có độ dài cố định không cần bắt đầu bằng byte biên, nhưng địa chỉ chỉ được chỉ tới byte biên. Nếu một chương trình tính địa chỉ của xâu bit và sau đó tham chiếu tới xâu thông qua địa chỉ này, ta có thể truy cập vào một vùng nhớ không chính xác. Tình huống này cũng có thể xảy ra khi truyền đối số xâu bit cho một chương trình con.

Nếu biến con trỏ hoặc biến tham chiếu được dùng, liệu vùng nhớ được tham chiếu có thuộc tính như trình biên dịch mong muốn? Một ví dụ về lỗi này là khi một con trỏ C++ trỏ tới địa chỉ của một cấu trúc dữ liệu khác với cấu trúc dữ liệu của nó.

Nếu một cấu trúc dữ liệu được tham chiếu ở nhiều hàm, thủ tục, liệu cấu trúc có được định nghĩa thống nhất trong các thủ tục không?

Khi đánh chỉ số cho một xâu/mảng, liệu có giới hạn chỉ số kết thúc xâu/mảng trong phạm vi cho phép không?

Với ngôn ngữ lập trình hướng đối tượng, liệu tất cả yêu cầu thừa kế có thoả mãn cài đặt class?

Lỗi khai báo dữ liệu (Data-Declaration Errors)

Liệu các biến có được khai báo chính xác? Một failure trong trường hợp này có thể không phải là lỗi, nhưng nó cũng khởi nguồn cho nhiều vấn đề. Ví dụ, nếu một hàm nhận một tham số là mảng, và không định nghĩa tham số như một mảng thành công, một tham chiếu tới mảng (ví dụ $C=A(I)$ được dịch thành một lời gọi hàm, dẫn tới máy tính cố gắng thực hiện mảng như một chương trình. Thêm nữa, nếu một biến không được định nghĩa một cách rõ ràng trong thủ tục, hoặc khỏi lệnh, liệu ta có thể hiểu là biến đó sẽ được dùng chung hay không?)

Nếu tất cả các thuộc tính của một biến không được quy định rõ ràng lúc định nghĩa, liệu giá trị mặc định có được hiểu rõ? Ví dụ, thuộc tính mặc định nhận được trong Java thường là một nguồn ngạc nhiên lớn. Khi một biến được khởi tạo bằng câu lệnh khai báo, liệu nó có được khởi tạo đúng? Trong nhiều ngôn ngữ, khởi tạo mảng và xâu đôi khi khá phức tạp và có xu hướng gây lỗi.

Liệu mỗi biến có được gán với đúng độ dài và kiểu?

Liệu khởi tạo của biến có đồng nhất với kiểu memory? Ví dụ, một biến trong hàm FORTRAN cần được khởi tạo lại mỗi lần gọi hàm, nó phải được khửoi tạo bằng một câu lệnh gán hơn là một câu lệnh DATA

Liệu có tên biến tương tự nhau (ví dụ VOLT và VOLTS)? Đây không nhất thiết là lỗi, nhưng ta nên đặt tên khác nhau phòng khi nhầm lẫn.

Lỗi tính toán (Computation Errors)

Liệu có tính toán nào sử dụng biến với kiểu dữ liệu không phù hợp?

Liệu có tính toán mixed-mode (kết hợp kiểu) nào không? Ví dụ cộng biến floating point cho biến integer. Cần hiểu rõ các luật chuyển đổi kiểu để tránh sai sót. Ví dụ, đoạn mã Java sau sẽ có lỗi rouding (làm tròn) khi thực hiện với số integers.

```
int x=1;  
int y=2;
```

```
int z=0;  
z=x/y;  
System.out.println ( "z=" +z );  
OUTPUT: z = 0
```

Liệu có tính toán nào sử dụng biến có cùng kiểu nhưng độ dài khác nhau?

Liệu kiểu của biến đích trong phép gán nhỏ hơn kiểu của kết quả của biểu thức ở vé phải?

Liệu có xảy ra lỗi overflow hoặc underflow (tràn) khi tính toán biểu thức? Chẳng hạn, kết quả là giá trị hợp lệ, nhưng kết quả trung gian có thể quá lớn hoặc quá nhỏ so với kiểu cho phép của ngôn ngữ lập trình.

Liệu có được phép thực hiện chia cho 0?

Nếu máy biểu diễn biến bằng số nhị phân, liệu có chuỗi tính toán nào cho kết quả sai hay không? Chẳng hạn, 10×0.1 không bằng 1.0 khi thực hiện trên số nhị phân.

Khi đưa vào ứng dụng, liệu giá trị của biến có ra ngoài khoảng có nghĩa.

Với biểu thức chứa nhiều hơn 1 phép toán, liệu giả thiết về thứ tự ưu tiên các phép toán có chính xác?

Có phép toán không hợp lệ nào liên quan tới số integer không, đặc biệt là phép chia? Ví dụ, nếu i là biến integer, biểu thức $2*i/2 == i$ sẽ phụ thuộc vào i là số chẵn hay lẻ và phép nhân hay chia được thực hiện trước?

Lỗi so sánh (Comparison Errors)

Có phép so sánh nào giữa các biến với kiểu khác nhau không, ví dụ so sánh xâu kí tự với địa chỉ, ngày, số?

Liệu có so sánh mixed-mode nào không hoặc là so sánh giữa các biến với độ dài khác nhau? Nếu có, cần đảm bảo là luật biến đổi phải được hiểu chính xác.

Các phép so sánh có chính xác? Lập trình viên thường bị nhầm các quan hệ như *at most, at least, greater than, not less than, less than or equal*.

Liệu các biểu thức Boolean có giá trị đúng như mong đợi? Lập trình viên thường mắc lỗi khi viết các biểu thức logic *and, or, not*.

Liệu các toán hạng của phép toán Boolean có là giá trị Boolean? Phép so sánh và phép toán Booleans thường được dùng xen kẽ. Hành động này dễ dẫn tới lỗi. Ví dụ, ta cần xác định i có nằm trong khoảng từ 2 tới 10 hay không, biểu thức $2 < i < 10$ là sai; biểu

thức đúng là $(2 < i) \&\& (i < 10)$. Nếu ta muốn xác định xem i có lớn hơn x hoặc y, $i > x \parallel y$ là sai, biểu thức đúng phải là $(i > x) \parallel (i > y)$. Nếu ta muốn so sánh 3 số có bằng nhau hay không, biểu thức $(a == b == c)$ cho kết quả khác với mong đợi. Nếu muốn kiểm tra quan hệ $x > y > z$, biểu thức đúng phải là $(x > y) \&\& (y > z)$.

Liệu có phép so sánh nào giữa số phẩy tĩnh và số phẩy động (floating point) ở hệ cơ số 2? Đây có thể là một nguồn lỗi vì lỗi làm tròn và biểu diễn xấp xỉ cơ số 2 của số hệ cơ số 10.

Với biểu thức chứa nhiều hơn 1 phép toán boolean, quy ước về thứ tự thực hiện các phép toán có chính xác? Ví dụ, với biểu thức $(if((a == 2) \&\& (b == 2) \parallel (c == 3))$, phép and hay or sẽ được thực hiện trước cần được quy định rõ ràng.

Liệu cách thức trình biên dịch tính toán giá trị của biểu thức Boolean có ảnh hưởng tới chương trình? Ví dụ, câu lệnh :

`if((x == 0) \&\& (x/y) > z)`

có thể được chấp nhận với trình biên dịch kết thúc kiểm tra điều kiện ngay khi một vé của and là false, nhưng có thể dẫn tới lỗi division-by-zero (chia cho 0) với trình biên dịch khác.

Lỗi luồng điều khiển (Control-Flow Errors)

Nếu chương trình chứa nhánh nhiều hướng, chẳng hạn phép tính GO TO, liệu biến chỉ số có ra ngoài phạm vi của nhánh? Ví dụ câu lệnh

`GO TO (200, 300, 400), i`

liệu i có nằm trong số giá trị 1, 2, 3?

Liệu vòng lặp có kết thúc? Chứng minh các vòng lặp sẽ kết thúc

Chương trình, module, hàm con có kết thúc?

4. Với một vòng lặp được điều khiển bởi cả biểu thức điều kiện (ví dụ vòng lặp tìm kiếm), và iteration, what are the consequences of loop fall-through? Ví dụ, với giả mã:

`DO I=1 to TABLESIZE WHILE (NOTFOUND)`

chuyện gì xảy ra nếu NOTFOUND không bao giờ bằng false?

Lỗi giao diện (Interface Errors)

Liệu số tham số của module có bằng với số đối số trong các lời gọi hàm. Liệu thứ tự

đối số có chính xác?

Liệu thuộc tính (chẳng hạn kiểu dữ liệu, kích thước) của từng tham số có tương thích với thuộc tính của đối số tương ứng?

Liệu đơn vị của từng tham số có tương thích với đơn vị của đối số tương ứng? Ví dụ, nếu tham số dùng đơn vị degrees, nhưng đối số lại dùng đơn vị radians?

Liệu số đối số truyền bởi module này cho module khác có bằng số tham số expected by that module?

Nếu hàm built-in được gọi, liệu số, thuộc tính và thứ tự của đối số có chính xác?

Lỗi Input/Output

Nếu files đã được định nghĩa, liệu các thuộc tính của nó có chính xác?

Liệu các thuộc tính của câu lệnh OPEN file có chính xác?

Liệu định dạng file có khớp với các câu lệnh I/O?

Có đủ bộ nhớ để chứa file mà chương trình sẽ đọc?

Các files đã được mở trước khi dùng?

Các files đã được đóng sau khi dùng?

Điều kiện end-of-file có được phát hiện và quản lý chính xác?

Các điều kiện lỗi I/O đã được quản lý chính xác?

Các xuất hiện lỗi chính tả, cú pháp ở các đoạn văn bản có được in/hiển thị ?

Các kiểm tra khác

Nếu trình biên dịch đưa ra một danh sách thuộc tính, kiểm tra các thuộc tính của từng biến để đảm bảo là không có thuộc tính mặc định không mong muốn nào đã được gán giá trị.

Nếu chương trình được dịch thành công, nhưng máy tính đưa ra cảnh báo (“warning”), ta cần kiểm tra chúng cẩn thận. Cảnh báo ám chỉ là trình biên dịch phát hiện ra bạn đang làm việc gì đó gây nghi vấn. Các nghi vấn này cần được kiểm tra lại.

Liệu chương trình hay module có kiểm tra validity cho đầu vào.

Bảng B.1. Gợi ý một số loại lỗi

STT	Loại lỗi	Nguồn gốc lỗi	Mức độ ảnh hưởng
-----	----------	---------------	------------------

1	Truy cập hoặc lưu trữ dữ liệu không chính xác	Lập trình	Lớn
2	Thêm chức năng mới	Yêu cầu	Lớn
3	Câu lệnh không rõ ràng	Lập trình	Nhỏ
4	Tiêu chuẩn áp dụng không đồng thuận	Thiết kế	Nghiêm trọng
5	Thay đổi yêu cầu của phần mềm	Yêu cầu	Lớn
6	Kiểm tra sai biến	Lập trình	Lớn
7	Vấn đề trong tính toán	Lập trình	Lớn
8	Mẫu thuẫn giữa các mục	Thiết kế	Lớn
9	Lỗi lộn giữa các mục	Thiết kế	Nhỏ
10	Vấn đề xử lý dữ liệu	Lập trình	Nghiêm trọng
11	Vấn đề dữ liệu	Lập trình	Nhỏ
12	Tham chiếu dữ liệu vượt quá giới hạn	Lập trình	Lớn
13	Kích cỡ dữ liệu không chính xác	Lập trình	Nhỏ
14	Vấn đề về tài liệu	Lập trình	Nhỏ
15	Vấn đề về chất lượng tài liệu	Tài liệu	Nhỏ
16	Logic bị lặp lại	Lập trình	Nhỏ
17	Dữ liệu nhúng trong bảng không chính xác hoặc thiếu	Lập trình	Nghiêm trọng
18	Sự cải tiến	Thiết kế	Lớn
19	Phương trình thiếu hoặc không chính xác	Lập trình	Nghiêm trọng
20	Dữ liệu ngoài không chính xác hoặc thiếu	Lập trình	Lớn
21	Bỏ qua điều kiện cực trị	Lập trình	Nghiêm trọng
22	Lỗi do sửa chữa trước đó	Lập trình	Nghiêm trọng
23	Còn hoặc chỉ số được đặt không chính xác	Lập trình	Lớn
24	Trường hợp hoặc bước bị quên	Lập trình	Nghiêm trọng
25	Mục phi logic	Thiết kế	Lớn

26	Thay đổi trong thực hiện biên tập	Tài liệu	Nhỏ
27	Cải thiện hiệu quả code	Lập trình	Lớn
28	Cải thiện comment	Thiết kế	Nhỏ
29	Cải thiện tính khả dụng	Lập trình	Lớn
30	Các mục chưa hoàn thiện	Lập trình	Lớn
31	Câu lệnh chưa hoàn thiện	Lập trình	Nghiêm trọng
32	Mâu thuẫn	Lập trình	Nghiêm trọng
33	Đối số chương trình con không nhất quán	Lập trình	Lớn
34	Mục không chính xác	Lập trình	Nghiêm trọng
35	Gọi chương trình con được đặt không chính xác	Lập trình	Lớn
36	Khởi tạo dữ liệu không chính xác	Lập trình	Lớn
37	Đầu vào dữ liệu không đúng hoặc thiếu	Lập trình	Nghiêm trọng
38	Thời gian đầu vào hoặc ra không chính xác	Lập trình	Nghiêm trọng
39	Vấn đề giao diện hoặc thời gian	Lập trình	Nghiêm trọng
40	Xử lý gián đoạn không chính xác	Lập trình	Nghiêm trọng
41	Lặp lại vòng lặp không chính xác	Lập trình	Lớn
42	Vấn đề logic	Lập trình	Nghiêm trọng
44	Sự giải thích sai	Lập trình	Lớn
45	Tính toán thiếu	Lập trình	Nghiêm trọng
46	Kiểm tra thiếu điều kiện	Lập trình	Nghiêm trọng
47	Mục bị thiếu	Thiết kế	Nghiêm trọng
48	Cách thức hợp	Lập trình	Lớn
49	Không định danh	Lập trình	Nhỏ
50	Gọi chương trình con không tồn tại	Lập trình	Nghiêm trọng
51	Mục không được xác thực	Thiết kế	Lớn
52	Không phải lỗi	Báo cáo	Nhỏ

53	Not current	Lập trình	Lớn
54	Không thể truy vết	Yêu cầu	Nghiêm trọng
	Toán hạng trong phương trình không chính xác	Lập trình	Lớn
55	Dữ liệu toán tử không chính xác hoặc thiếu	Lập trình	Nghiêm trọng
56	Toán tử trong phương trình không chính xác	Lập trình	Lớn
57	Các sự cải tiến khác	Lập trình	Lớn
59	Vấn đề khác (Không thể phân loại)	Yêu cầu	Lớn
60	Dữ liệu đầu ra không chính xác hoặc bị thiếu	Lập trình	Nghiêm trọng
61	Đóng gói hoặc mở gói dữ liệu không chính xác	Lập trình	Lớn
62	Dùng dấu ngoặc không chính xác	Lập trình	Nghiêm trọng
63	Giảm độ chính xác	Lập trình	Nhỏ
64	Mục dư thừa	Thiết kế	Nhỏ
65	Tham chiếu sai biến dữ liệu	Lập trình	Nhỏ
66	Loại bỏ chức năng năng không cần thiết	Thiết kế	Nhỏ
67	Lỗi làm tròn	Lập trình	Nhỏ
68	Chia tỉ lệ hoặc chọn sai đơn vị dữ liệu	Lập trình	Nhỏ
69	Phạm vi dữ liệu không chính xác	Lập trình	Lớn
70	Dữ liệu cảm biến không chính xác hoặc thiếu	Lập trình	Nghiêm trọng
71	Lỗi trong việc quy ước dấu hiệu	Lập trình	Lớn
72	Sửa chữa phần mềm hoặc vấn đề phần cứng	Lập trình	Lớn
73	Chương trình con hoặc mô-đun không khớp	Lập trình	Nghiêm trọng
74	Biến đăng ký không chính xác	Lập trình	Nhỏ

75	Lỗi thời gian gây mất mát dữ liệu	Lập trình	Nghiêm trọng
76	Mục không thể thực hiện	Thiết kế	Nghiêm trọng
77	Chức năng không cần thiết	Lập trình	Nhỏ
78	Cập nhật chức năng hiện tại	Thiết kế	Lớn
79	Kiểu biến không chính xác	Lập trình	Nhỏ
80	Gọi hàm con không sai	Lập trình	Lớn

Phụ lục 2: Một số hướng dẫn cho các loại kiểm thử

Một số hướng dẫn cho các loại kiểm thử

Các điểm cần được xem xét để đảm bảo chất lượng của GUI

Phương diện	Mô tả	Kết quả mong đợi
Chính tả	Kiểm tra chính tả. Sử dụng từ điển của Merriam-Webster là tập mẫu tự cho tiếng Anh-Mỹ và từ điển Oxford là tập mẫu tự cho tiếng Anh-Anh	Tất cả các từ đúng chính tả
Chính tả	So sánh từ được viết hoa với hướng dẫn viết hoa. Kiểm tra tính nhất quán trong việc viết hoa giữa các từ	Tuân theo quy định viết hoa cho trước.
Định vị nhãn	Kiểm tra các nhãn có ở gần các điều khiển (controls) mà nó hiện giải thích (explanation) không	Tất cả các nhãn gần với điều khiển, với 1 khoảng trống ở giữa kí tự cuối cùng và cạnh của điều khiển
Định vị nhãn	Kiểm tra các nhãn được đặt thông nhất ở bên trái hay bên trên các điều khiển	<ol style="list-style-type: none"> Tất cả các nhãn được đặt ở bên trái hoặc bên trên điều khiển Nếu nhãn ở phía trên điều khiển, nó sẽ được căn chỉnh về góc trái dưới hoặc ở trung tâm Nếu nhãn được căn chỉnh ở bên phải hoặc dưới điều khiển thì phải có lý do hợp lệ
Màu font chữ	Để có thể đọc được, ta kiểm tra độ tương phản của màu font chữ cần tương thích với nền. Kiểm tra tất cả các màn hình để đảm bảo tất cả các kí tự hiển thị rõ ràng	Tất cả kí tự trên tất cả các màn hình đều dễ đọc
Màu font chữ	Font chữ màu đỏ nên được dùng cho thông báo lỗi và những cảnh báo (warning)	<ol style="list-style-type: none"> Font chữ đỏ không dùng cho tiêu đề, header hoặc văn bản thông thường Thông báo lỗi và cảnh báo được dùng với font chữ đỏ

Màu font chữ	Font chữ màu xanh (blue) thường sẽ gây khó đọc, nên tránh màu font này	Nếu xuất hiện chỗ có font chữ màu xanh, thì nó phải rõ ràng và dễ đọc
Màu font chữ	Font màu tối thì dễ đọc hơn. Font màu sáng trên nền font màu tối sẽ gây khó đọc	Màu font chữ nên tối hơn màu nền
Kiểu chữ	Kiểu chữ được chọn cần phù hợp với màn hình hiển thị và đồng nhất qua các ứng dụng khác nhau	Tất cả kí tự trên cùng một màn hình có cùng font. Nếu không thì cần có lí do hợp lệ, và vẫn cần phải tương thích với màn hình và mục đích sử dụng
Kích cỡ chữ	Kích cỡ chữ cần tương thích với văn bản thường, tiêu đề, header, cảnh báo, ...	<ol style="list-style-type: none"> Tất cả tiêu đề có cùng kích cỡ Tất cả các header có cùng kích cỡ Tất cả văn bản thường có cùng kích cỡ Tất cả các cảnh báo có cùng kích cỡ Kích cỡ của tiêu đề, header văn bản thường, cảnh báo có thể khác nhau để phân biệt với nhau cho rõ ràng
Tông màu	Tông màu nên tương thích xuyên suốt các ứng dụng trên tất cả các màn hình. Nghĩa là các nền(background), menu, font chữ, hình ảnh, và thông báo có sự tương thích về màu sắc	Tất cả các màn hình sử dụng tông màu tương thích
Tông màu	Tông màu nên có sự khác nhau giữa các lớp thông tin khác nhau trên màn hình. Menu nên có màu khác so với văn bản thường, tiêu đề, header nên có màu khác với văn bản thường,... Điều này giúp người dùng phân biệt được các mức thông tin	Tông màu khác nhau cho các lớp thông tin khác nhau trên một màn hình

Tông màu	Những thông tin như yêu cầu đặc biệt, giảm giá hay thông báo cảnh báo (vd : hết hạn thẻ thành viên) nên có tông màu thu hút sự chú ý người dùng	Tất cả thông tin đặc biệt trên màn hình có tông màu khác với phần còn lại, và có khả năng thu hút sự chú ý ngay lập tức
Tông màu	Các màu khác nhau sẽ bao hàm các ý nghĩa khác nhau cho người dùng. Việc bao hàm này có thể khác nhau giữa các quốc gia, nhưng có 1 số màu phổ biến như sau: 1. Đỏ: nguy hiểm, dừng lại, nóng, mất tiền/tài chính 2. Xanh lá cây: ổn, đi 3. Vàng: cảnh báo, gắn với kí hiệu dừng lại hoặc nguy hiểm 4. Xanh dương: mát lạnh 5. Đen: lợi nhuận về mặt tài chính 6. Xám: nặng, mờ 7. Cam: năng lượng 8. Trắng: nguyên chất, tinh khiết	Tông màu sẽ tương thích với hàm ý ngầm của nó. Chú ý tránh sử dụng sai màu, đặc biệt với các thông tin đặc biệt
Giao diện đồ họa	Đồ họa nên chỉ ra được mục đích rõ ràng. Kiểm tra xem đồ họa có truyền tải được ý nghĩa mà nó mang hay không	Giao diện đồ họa khớp với mục đích mà nó được gán
Giao diện đồ họa	Giao diện đồ họa trên website không nên gây phản cảm giữa các nền văn hóa. Ví dụ như hiển thị ngón tay giữa hay ngón trỏ sẽ gây phản cảm ở nhiều nơi. Đảm bảo rằng giao diện đồ họa không xúc phạm tôn giáo hoặc tín ngưỡng ở bất kì đâu	Chỉ sử dụng đồ họa tiêu chuẩn hoặc trung lập tôn giáo
Giao diện đồ họa	Thi thoảng thì đồ họa sẽ không tự động được hiển thị. Trong những trường hợp này thì những văn bản thay thế	Mọi giao diện đồ họa có văn bản thay thế

	hiển thị. Do đó hãy đảm bảo mọi giao diện đồ họa có văn bản thay thế	
Biểu tượng	Các biểu tượng nên phản ánh đúng chức năng của nó. Bên cạnh đó mọi biểu tượng nên có công cụ gợi ý(tool tip)	Tất cả biểu tượng phản ánh đúng chức năng và hiển thị phần gợi ý giải thích chức năng
Biểu tượng	Mỗi biểu tượng nên có phím tắt để chức năng của nó có thể được sử dụng thông qua bàn phím	Mỗi biểu tượng có thể click(clickable icon) có thể được truy cập bằng phím tắt thông qua bàn phím

Một số hướng dẫn kiểm thử để đảm bảo chất lượng GUI

Khía cạnh	Mô tả	Kết quả mong đợi
Đa nhiệm	Nếu ứng dụng cho phép thực thi đa nhiệm thì cần thử cho chạy hai hoặc nhiều yêu cầu đồng thời	Không lỗi
Đa nhiệm	Nếu ứng dụng không cho phép thực thi đa nhiệm thì không cho chạy yêu cầu thứ hai	Yêu cầu thứ hai này không được phép chạy. Có thông báo một yêu cầu khác đang chạy sẽ được hiển thị, và yêu cầu đó vẫn đang tiếp tục chạy
Thay đổi kích cỡ cửa sổ	Nếu ứng dụng cho phép thay đổi kích cỡ cửa sổ, ta thay đổi, thu nhỏ, phóng to cửa sổ	Cửa sổ được thay đổi, thu nhỏ, phóng to
Kích hoạt và vô hiệu hóa	Các nút có thể được kích hoạt hoặc vô hiệu hóa, phụ thuộc vào ứng dụng. Nút “Xóa” thông thường nên để vô hiệu hóa khi không có gì được chọn để xóa hoặc ở trong những trường hợp tương tự	Các nút có thể được kích hoạt hoặc vô hiệu hóa khi cần thiết
Kích hoạt và vô hiệu hóa	Các mục menu có thể được kích hoạt hoặc vô hiệu hóa dựa vào cài đặt bảo mật của ứng dụng. Chạy	Những mục menu không tương thích với quyền bảo mật sẽ bị vô hiệu hóa và ngược lại

	ứng dụng dưới nhiều quyền bảo mật(security authorization)	
Hộp thoại phương thức(modal dialog)	Khi một hộp thoại phương thức được hiển thị, cửa sổ chính bị vô hiệu hóa. Hiển thị một hộp thoại phương thức xem xem có vô hiệu hóa được cửa sổ chính không	Cửa sổ chính không thể truy cập được
Hộp thoại không chế độ(modeless dialog)	Hộp thoại này nên được gắn với cửa sổ chính. Hiển thị một hộp thoại không chế độ và thử phóng to, thu nhỏ cửa sổ chính	Hộp thoại không chế độ cũng phóng to và thu nhỏ theo cửa sổ chính
Tiêu điểm con trỏ	Khi màn hình vừa được tải lại, con trỏ nên được đặt ở một trường mặc định, thi thoảng sẽ không có trường mặc định	Có một trường mặc định cho mọi màn hình và con trỏ sẽ được đặt ở đó mỗi khi màn hình tải lại
Tiêu điểm con trỏ	Khi con trỏ được chuyển đến một vùng điều khiển có chứa dữ liệu, tất cả dữ liệu phải được chọn. Điều này sẽ ngăn việc bổ sung thêm dữ liệu vào trường điều khiển đã sẵn có. Nhập dữ liệu vào trường điều khiển vào chuyển tiêu điểm con trỏ sang đó	Tất cả dữ liệu được chọn
Thông báo	Sau khi một thông báo lỗi được hiển thị và được đóng, con trỏ nên được chuyển tới trường lỗi. Thi thoảng con trỏ không được trả tới trường nào hết	Con trỏ trả tới trường lỗi
Thông báo	Sau khi một thông báo lỗi được hiển thị và được đóng, con trỏ nên trả trở lại trường mà trước đó đã trả vừa trả. Thi thoảng tiêu điểm trả không được đặt	Con trỏ được đặt trả lại trường mà trước đó đã trả
Refresh màn hình	Việc tải lại màn hình là cần thiết sau các tác vụ như lưu trữ hoặc xóa. Trong khi màn hình đang tải	Màn hình refresh đúng

lại, việc cần thiết là bỏ trống các trường có thể để trống trong quá trình sử dụng, cũng cần phải tải các giá trị trong các trường “có thể chọn”(selectable field), ví dụ: combo box, list box, list view, etc. Thực hiện các tác vụ lưu hoặc xóa và quan sát màn hình khi refresh

Các sự kiện nhấp chuột	Các khu vực không mong đợi nhấp chuột sẽ không kích hoạt hành động. Ví dụ: khung, nhãn và danh sách trống không được kích hoạt bất kỳ hành động nào khi nhấp vào. Nhấp và nhấp đúp vào các khu vực như vậy và quan sát phản hồi	Không có phản hồi
Thanh điều hướng	Thanh điều hướng khi sử dụng phím tab phải đi từ trái qua phải từ trên xuống dưới. Tab từ điều khiển này đến điều khiển khác	Tiêu điểm (focus) chuyển từ điều khiển bên trái sang điều khiển bên phải, sau đó xuống hàng tiếp theo. Khi tab đến điều khiển cuối cùng thì quay trở lại với điều khiển đầu tiên.
Thanh điều hướng	Những người khuyết tật – đặc biệt là về thị giác – có thể bắt buộc phải dùng bàn phím bên cạnh chuột.	Tất cả cá chức năng đều có thể truy cập bởi bàn phím.
Thanh điều hướng	Cuộn màn hình theo cả chiều dọc và chiều ngang là rất tốn công	Không cần thiết phải cuộn màn hình theo cả 2 hướng trừ khi có lý do bắt buộc
Các mặc định	Phải có một ô được chọn sẵn khi tải bảng điều khiển có các nút chọn (radio button)	Có một nút chọn được chọn sẵn khi tải xong
Các mặc định	Các ô đánh dấu (checkbox) phải được đánh dấu trước hoặc không tùy theo các giá trị mặc định	Các ô đánh dấu (checkbox) phải được đánh dấu trước hoặc không

tùy theo các giá trị mặc định đối với lần đầu tiên tải

Các mặc định	Một nút lệnh phải được mặc định cho màn hình.	1. Mặc định một nút lệnh nào đó phải được kích hoạt khi ấn phím enter. 2. Nút mặc định thường là những nút mang tính tích cực như “Save” hoặc “Ok”. Các nút mang tính tiêu cực như “Cancel” hoặc “Delete” không nên là các nút mặc định 3. Phải có lý do hợp lý thì các nút mang tính tiêu cực mới có thể là nút mặc định
Phản hồi hành động	Đối với hành động “lưu” nên biểu thị rằng bản ghi đã được lưu	Sự xác nhận hành động “lưu” có thể được hiển thị trên một thanh tiến trình, hay sự thay đổi hình dáng con trỏ, tin nhắn thông báo xác nhận hoặc các thứ gì đó tương tự. Hành động “lưu” không thể bỏ qua nếu không có sự biểu thị xác nhận
Tính nhất quán	Cần có sự nhất quán giữa các nút trên thanh công cụ và các item trong menu. Khởi tạo hành động (bắt sự kiện) đối với từng lựa chọn trên thanh công cụ rồi đến menu (cũng có thể chấp nhận ngược lại)	Các hành động tương tự nhau được kích hoạt khi đã khởi tạo trong thanh công cụ hoặc menu
Thanh công cụ	Mỗi nút trên thanh công cụ đều phải có một menu item tương ứng với nó trong menu	Mỗi nút trên thanh công cụ đều phải có một menu item tương ứng với nó trong menu
Thanh công cụ	Các nút trên thanh công cụ đều phải có gợi ý công cụ để mô tả hành động khi được nhấn nút. Chắc chắn rằng gợi ý công cụ	Mỗi nút trên thanh công cụ đều phải có gợi ý công cụ

hiện lên khi đưa chuột qua mỗi nút

Thứ tự (khi ấn) tab	Các điều khiển đầu vào đè có thể truy cập được bằng nút tab. Thứ tự của chúng là từ trái qua phải, từ trên xuống dưới	Con trỏ chuyển từ điều khiển đầu vào này đến điều khiển đầu vào khác với phím tab, từ trái qua phải và từ trên xuống dưới
---------------------	---	---

Kiểm thử ngoại lệ cho màn hình.

Loại dữ liệu	Mô tả ca kiểm thử	Chú ý
Các giá trị số	<ol style="list-style-type: none">Không được nhập kí tự không phải là số.Trường hợp chỉ nhận số dương, không được nhập dấu âm.Trường hợp số thập phân, không được nhập 2 dấu chấm thập phân.Trường hợp là số nguyên, dấu chấm thập phân và dấu phân số không được chấp nhận.Bởi vì kiểu dữ liệu cho phép số có kích thước tối đa, phần mềm nên kiểm tra kích cỡ của giá trị nhập vào và ngăn người dùng nhập số lớn hơn giá trị tối đa.Kiểm tra nếu trường bên trái là khoảng trắng, phần mềm sẽ lưu số ‘0’ trong cơ sở dữ liệu hoặc có thủ tục để xử lý trắng.	<p>Thử nhập 1 giá trị không phải là số.</p> <p>Nhập số với dấu âm.</p> <p>Nhập số với 2 dấu chấm thập phân.</p> <p>Nhập phân số ở vị trí số nguyên.</p> <p>Nhập 1 số nhiều hơn chữ số hoặc 1 số rất lớn.</p> <p>Để các trường trống và thử lưu dữ liệu</p>

	7. Các giá trị xung quanh biên có thể bị lỗi.	Nhập các giá trị bên trên và bên dưới biên.
	8. Các giá trị không được khởi tạo cho lần lặp thứ hai trở đi, đặc biệt là tác vụ “Save” . Luôn thực hiện lại tác vụ “Save” với một bộ giá trị khác.	Kiểm tra giá trị mặc định phù hợp khi màn hình được làm mới.
	9. Các lỗi phổ biến nhất là khi tính toán kết quả.	Kiểm tra kết quả thường xuyên.
	10. Kiểm tra phép chia có trả về lỗi khi mẫu số bằng không hoặc khi cả tử số và mẫu số bằng không.	Thử làm mẫu số bằng không.
Các giá trị chữ số	1. Phần mềm nên kiểm tra kích cỡ và giới hạn của giá trị mà nó cho phép.	Nhập dữ liệu dài hơn mức cho phép.
	2. Trường hợp là tên của một người, phần mềm phải đảm bảo rằng các giá trị số sẽ không được nhập.	Nhập vài số vào trường tên.
	3. Các trường không được phép chứa các kí tự đặc biệt, chẳng hạn như tổ hợp với các phím Ctrl hoặc Alt. Kiểm tra những tổ hợp trên nếu chúng gây ra bất kì vấn đề gì.	Nhập một vài kí tự đặc biệt trong trường chữ số.
Các giá trị ngày	1. Kiểm tra tính nhất quán giữa tháng và ngày. Ví dụ : Tháng Hai có 28 hoặc 29 ngày, tháng Một có 31 ngày, tháng Tư có 30 ngày, v.v.. Đôi khi các lập trình viên không cho số ngày thích hợp trong các tháng.	Nhập số ngày không tương ứng hoặc không hợp lệ.

	<p>2. Nếu các tháng được cho phép nhập dưới dạng chữ số, thì các chữ số lớn hơn 12 sẽ bị từ chối.</p> <p>3. Nếu có “from date” và “to date”, thì “to date” sẽ muộn hơn “from date” .</p> <p>4. Kiểm tra tính hợp lệ của năm.Ví dụ, năm “9999” sẽ bị từ chối. Dựa vào ứng dụng, nhập một vài giá trị năm không hợp lệ và xem ứng dụng có từ chối nó không.</p>	Nhập tháng không hợp lệ. Nhập “to date” muộn hơn “from date”.	
Tệp và tác vụ bảng	<p>1. Hầu hết các lập trình viên không xử lý trường hợp bảng rỗng trong các tác động bảng chính.</p> <p>2. Một lỗi thường gặp khác là do các lập trình viên để bảng hay tệp còn mở sau khi dùng.Điều này có thể gây lỗi nếu cùng bảng hay tệp đó được mở ra lần nữa.</p>	Làm rõ ràng tất cả dữ liệu của bảng chính và chạy ứng dụng. Thử chạy cùng dữ liệu trên màn hình tuần tự, kiểm tra dữ liệu có được lưu đúng cách và ứng dụng không bị crash.	
Sự kiện “Save” và “Submit”	Đôi khi các lập trình viên thực hiện xác thực dữ liệu trong sự kiện “lost focus”, ví dụ như text box và combo box.Có thể kiểm tra điều này bằng cách giữ focus trong trình điều khiển nhưng nhấn vào nút Save hoặc Submit để xem việc xác thực dữ liệu có diễn ra hay không	Sau khi nhập dữ liệu sai ở một trường, nhấn vào nút Save.Nên xác thực lại dữ liệu một lần nữa.	
Nhấn	Các nhấn không nên được nhấn vào trừ khi chúng kết nối tới một trang web.	Nhấn vào tất cả các nhấn và xem nếu có kết quả xảy ra.	
Các nút Delete	<p>1. Bất cứ khi nào một hành động “Delete” được khởi tạo, hệ thống nên đưa ra một thông báo xác nhận việc xóa.</p> <p>2. Một hành động “Delete” không được cho phép khi không mục nào được chọn.Hệ thống cũng nên hiển thị một tin nhắn lỗi hoặc ngăn chặn việc “Delete” ấy.</p>	Thử xóa vài giá trị và đảm bảo rằng tin nhắn xác nhận được hiển thị.	Thiết lập tiêu điểm trống và nhấn vào nút “Delete”.

Nút Submit và Save	1. Khi bắt kì một trường bắt buộc nào để trống, hệ thống nên trả về một tin nhắn lỗi. 2. Khi một trường có độ dài lớn hơn mức cho phép, hệ thống nên trả về một tin nhắn lỗi.	Để một vài trường bắt buộc trống và nhấn vào nút Save.
Danh sách	Khi một danh sách rỗng, hệ thống không nên phản hồi hoặc không được rời vào trường hợp lỗi nếu danh sách được nhấn hoặc nhấn 2 lần.	Nhấn và nhấn đúp vào các danh sách trống.
Nhấn hoặc nhấn đúp	Khi nhấn vào các khu vực không được dự kiến, hệ thống không nên thực hiện bất kì hoạt động nào hoặc bị lỗi.	Thử nhấn ngẫu nhiên vào vài vị trí không mong muốn trên màn hình, và kiểm tra nếu có gì xảy ra hoặc ứng dụng bị lỗi không.
Đa màn hình	Khi ứng dụng cho phép mở nhiều màn hình, hãy thử mở một vài cái và chuyển đổi chúng với nhau. xem nó có bị lỗi không.	Mở nhiều màn hình và
Sử dụng sai phím	1. Nhấn vài phím tổ hợp với Ctrl và Alt, và kiểm tra xem có bị lỗi không 2. Nhấn nhiều phím cùng lúc và kiểm tra có lỗi gì xảy ra không	Nhấn phím C, Z và D cùng với Crt và vài phím với Alt. Nhấn nhiều phím cùng thời điểm.

Kiểm thử ngoại lệ cho kết quả trả về.

Các khía cạnh báo cáo	Bản mô tả ca kiểm thử	Chú ý
Tham số báo cáo	1. Trong vài trường hợp, báo cáo cần các tham số như ngày, tên, mã	Thử tạo báo cáo mà không đưa kèm các tham số cần

, v.v ... Trong trường hợp đó, nếu tham số trả về là trống, thì hệ thống nên phản hồi một tin nhắn lỗi hoặc đưa ra trả về trống, nhưng không được bị lỗi.

2. Trong trường hợp sai tham số trả về, hệ thống nên phản hồi một tin nhắn lỗi hoặc đưa ra báo các rỗng, nhưng không được bị lỗi.

3. Trong trường hợp các tham số không hợp lí, như “from date” lại sau “to date” hoặc 1 giá trị không tồn tại, hệ thống nên trả về một tin nhắn lỗi.

Căn chỉnh	1. Dữ liệu trong các cột và các tiêu đề cột phải được căn chỉnh chính xác. Chắc chắn rằng các mục chữ số được căn trái và giá trị số được căn phải. 2. Tiêu đề, ngày báo cáo, số trang, v.v ... có thể được cắt bớt nếu không đủ không gian cho chúng.	Kiểm tra tính nhất quán của dữ liệu và tiêu đề cho các cột của báo cáo.	Thử tạo báo cáo cùng với các tham số sai.
-----------	---	---	---

Quá trình tổng quát	Trong các báo có kích thước lớn, kiểm tra xem tất cả các bản ghi có được bao gồm trong quá trình không. Những báo cáo này thông thường đi kèm với thống kê kiểm soát như : số lượng bản ghi được xem xét và số lượng thực trong báo cáo.	Kiểm tra báo cáo có thống kê kiểm soát. Kiểm tra tất cả các bản ghi được xử lý và các bản ghi liên quan có trong báo cáo.
---------------------	--	---

Kiểm thử ngoại lệ cho các trang web.

Các khía cạnh	Bản mô tả ca kiểm thử	Chú ý
Đường dẫn	<ol style="list-style-type: none">Nhấn vào tất cả các đường dẫn trên trang và đảm bảo rằng nó dẫn tới đúng trang web. Cùng với đó là đảm bảo không đường dẫn nào bị hỏng.Kiểm tra các đường dẫn tới các trang web khác được mở ở một cửa sổ mới.Không cho phép sử dụng nút Back sau khi đăng xuất khỏi trang. Hệ thống nên hiển thị một thông báo yêu cầu người dùng đăng nhập.	Kiểm tra tất cả đường dẫn. Nhấn vào các đường dẫn đến trang khác. Đăng xuất khỏi ứng dụng và sau đó ấn vào nút Back.
Dữ liệu đầu vào sai	Khi dữ liệu đầu vào sai, ngay lập tức hiển thị thông báo lỗi ngay sau đầu vào hoặc sau khi nhấn nút Send.	Nhập sai dữ liệu vài trường và xem nó có bị báo lỗi không.

Phụ lục 3: Test plan mẫu