

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC CÔNG NGHỆ TP.HCM

KIỂM THỬ VÀ ĐẢM BẢO
CHẤT LƯỢNG PHẦN MỀM

Biên Soạn:

ThS. Dương Thành Phết

ThS. Trịnh Công Nhựt

ThS. Nguyễn Đông Phương

ThS Nguyễn Thị Thanh Trúc

KIỂM THỬ VÀ ĐÀM BẢO CHẤT LƯỢNG PHẦN MỀM



★ 1 . 2 0 2 1 . C M P 1 7 9 ★

Các ý kiến đóng góp về tài liệu học tập này, xin gửi về e-mail của ban biên tập:
tailieuhoctap@hutech.edu.vn

MỤC LỤC

MỤC LỤC	I
HƯỚNG DẪN	V
BÀI 1: TỔNG QUAN VỀ KIỂM THỬ VÀ ĐÀM BẢO CHẤT LƯỢNG PHẦN MỀM.....	1
1.1 CÁC MÔ HÌNH PHÁT TRIỂN PHẦN MỀM.....	1
1.1.1 Water-fall Model.....	1
1.1.2 Prototype Model	3
1.1.3 Spiral Model	3
1.1.4 SDLC	5
1.1.5 Agile Model.....	5
1.1.6 Qui trình Hợp nhất (RUP)	5
1.1.7 V model	7
1.2 KIỂM THỬ VÀ ĐÀM BẢO CHẤT LƯỢNG PHẦN MỀM.....	8
1.2.1 Phần mềm và chất lượng phần mềm.....	8
1.2.2 Các yếu tố ảnh hưởng chất lượng phần mềm	9
1.2.3 Khái niệm kiểm thử và đảm bảo chất lượng phần mềm	11
1.2.4 Mục tiêu của kiểm thử và đảm bảo chất lượng phần mềm	12
1.2.5 Tầm quan trọng của kiểm thử và đảm bảo chất lượng phần mềm.....	12
1.2.6 Các nguyên tắc trong kiểm thử và đảm bảo chất lượng phần mềm	14
1.2.7 Các khái niệm liên quan kiểm thử và đảm bảo chất lượng phần mềm.....	15
1.2.8 Các đối tượng thực hiện kiểm thử và đảm bảo chất lượng phần mềm.....	18
1.2.9 Các điểm cần lưu ý khi kiểm thử và đảm bảo chất lượng phần mềm	19
1.2.10 Các hạn chế của kiểm thử và đảm bảo chất lượng phần mềm	20
1.3 VAI TRÒ CỦA KIỂM THỬ VÀ ĐÀM BẢO CHẤT LƯỢNG PHẦN MỀM.....	20
1.4 TÀI LIỆU VỀ SẢN PHẨM PHẦN MỀM	21
1.4.1 Tài liệu dự án.....	21
1.4.2 Tài liệu đặc tả	21
1.4.3 Tài liệu đặc tả thiết kế	22
1.4.4 Tài liệu kiểm thử	23
1.5 YÊU CẦU KIỂM THỬ VÀ ĐÀM BẢO CHẤT LƯỢNG	24
1.5.1 Quy trình kiểm thử và đảm bảo chất lượng phần mềm	24
1.5.2 Thuộc tính yêu cầu phần mềm	29
1.5.3 Các thành phần chính trong bản kế hoạch kiểm thử.....	29
1.6 CÁC YÊU TỐ CÂN THIẾT CỦA KIỂM THỬ VÀ ĐÀM BẢO CHẤT LƯỢNG PHẦN MỀM ..	31
1.7 CÁCH VIẾT YÊU CẦU KIỂM THỬ	32
TÓM TẮT	34
BÀI 2: KỸ THUẬT THIẾT KẾ TEST CASE.....	35
2.1 CÁC KHAI NIÊM CHÍNH	35

2.2 THIẾT KẾ TEST-CASE: WHITE-BOX.....	37
2.2.1 Tổng quan về kiểm thử hộp trắng	37
2.2.2 Ưu điểm/Nhược điểm kiểm thử hộp trắng	37
2.2.3 Các mức độ áp dụng	38
2.3 THIẾT KẾ TEST-CASE: BLACK-BOX	38
2.3.1 Tổng quan kiểm thử hộp đen	38
2.3.2 Ưu điểm/Nhược điểm của kiểm thử hộp đen	39
2.3.3 Qui trình kiểm thử hộp đen.....	40
2.4 CÁC HỆ THỐNG QUẢN LÝ TEST-CASE.....	42
2.4.1 Giới thiệu	42
2.4.2 Testcase	43
2.4.3 Các thành phần testcase	44
TÓM TẮT	45
BÀI 3: THIẾT KẾ TEST CASE - BLACK-BOX.....	46
3.1 KỸ THUẬT LỚP TƯƠNG ĐƯƠNG	46
3.2 KỸ THUẬT GIÁ TRỊ BIÊN	49
3.3 KỸ THUẬT BẢNG QUYẾT ĐỊNH.....	53
3.4 KỸ THUẬT DỊCH CHUYÊN TRẠNG THÁI.....	56
3.5 BÀI TẬP THIẾT KẾ TEST CASE – BLACK BOX	59
TÓM TẮT	63
BÀI 4: THIẾT KẾ TEST CASE – WHITE - BOX.....	64
4.1 TỔNG QUAN KIỂM THỬ HỘP TRẮNG	64
4.2 KIỂM THỬ ĐƯƠNG CƠ BẢN	65
4.3 KIỂM THỬ ĐỘ BAO PHÙ.....	68
4.3.1 Một số khái niệm.....	68
4.3.2 Phù câu lệnh.....	69
4.3.3 Phù nhánh.....	70
4.3.4 Phù đường dẫn	71
4.3.5 Phù Điều kiện	72
4.3.6 Phù Nhánh và Điều kiện	72
4.4 KIỂM THỬ VONG LẶP	74
4.5 KIỂM THỬ LUÔNG DỰ LIỆU	75
4.6 BÀI TẬP THIẾT KẾ TEST CASE – WHITE BOX.....	78
TÓM TẮT	80
BÀI 5: LÔI PHẦN MỀM VÀ HỆ THỐNG QUẢN LÝ LÔI	81
5.1 TỔNG QUAN VỀ LÔI PHẦN MỀM	81
5.2 NGUYÊN NHÂN GÂY RA LÔI THƯƠNG GẶP.....	82
5.3 CÁC LÔI THƯƠNG GẶP TRONG PHẦN MỀM.....	85
5.4 TIM LÔI VÀ PHÂN TÍCH LÔI	86
5.5 GIỚI THIỆU VỀ HỆ THỐNG QUẢN LÝ BUG	94
5.6 VONG ĐƠI CỦA BUG TRÊN HỆ THỐNG QUẢN LÝ BUG	94

5.6.1 Những thông số cần thiết để đo lỗi	95
5.6.2 Kiểm soát thực hiện sửa lỗi (Defect tracking).....	95
5.6.3 Các thông số lỗi	96
5.6.4 Môi trường kiểm thử	96
5.7 THỰC HÀNH VỚI HỆ THÔNG QUẢN LÝ BUGZILLA	96
TÓM TẮT	107
BÀI 6: KIỂM THỬ TỰ ĐỘNG VÀ CÔNG CỤ	108
6.1 GIỚI THIỆU VỀ AUTOMATION SOFTWARE TESTING	108
6.1.1 Quy trình kiểm thử tự động	109
6.1.2 Ưu và nhược điểm kiểm thử tự động.....	109
6.1.3 Phân loại công cụ kiểm thử.....	110
6.2 GIỚI THIỆU VỀ ACTION-BASED TESTING.....	112
6.3 KIỂM THỬ TỰ ĐỘNG VỚI KATALON STUDIO	117
6.3.1 Giới thiệu	117
6.3.2 Các tính năng chính của Katalon Studio.....	117
6.3.3 Cài đặt Katalon Studio	118
6.3.4 Viết kịch bản test với Katalon Studio.....	121
6.4 KIỂM THỬ TỰ ĐỘNG VỚI SELENIUM	123
6.4.1 Giới thiệu	123
6.4.2 Thành phần của Selenium	125
TÓM LƯỢC.....	127
BÀI 7: KIỂM THỬ ĐƠN VỊ (UNIT TEST)	128
7.1 TỔNG QUAN VỀ KIỂM THỬ ĐƠN VỊ	128
7.1.1 Kiểm thử đơn vị là gì?.....	128
7.1.2 Tại sao Kiểm thử đơn vị lại quan trọng?	129
7.1.3 Lợi ích của Kiểm thử Đơn vị	129
7.1.4 Các loại Kiểm thử Đơn vị	130
7.1.5 Ai thực hiện Kiểm thử đơn vị?	130
7.1.6 Làm thế nào để làm Unit Testing?	130
7.1.7 Quy trình kiểm thử đơn vị.....	131
7.1.8 Quá trình kiểm tra đơn vị bao gồm:	131
7.1.9 Các thực hành tốt nhất về Unit Testing	131
7.1.10 Một số framework Unit Test	133
7.2 KIỂM THỬ JUNIT	133
7.2.1 JUnit là gì?	133
7.2.2 Các tính năng của JUnit.....	133
7.2.3 Một số khái niệm trong JUnit	134
7.2.4 Cài đặt JUnit.....	134
7.2.5 Ví dụ sử dụng JUnit trên Eclipse	135
7.3 KIỂM THỬ NUNIT.....	138
7.3.1 NUnit là gì?	138

7.3.2 Cài đặt NUnit	138
7.3.3 Tao dự án kiểm thử	141
7.3.4 Các thuộc tính <i>TestFixture</i> , <i>Test</i> và lớp <i>Assert</i>	143
7.3.5 Thực hiện kiểm thử với công cụ <i>Test Explorer</i>	145
TÓM TẮT	147
BÀI 8: QUẢN LÝ CHẤT LƯỢNG PHẦN MỀM	148
8.1 CHẤT LƯỢNG QUÁ TRÌNH VÀ CHẤT LƯỢNG SẢN PHẨM	148
8.2 ĐÀM BẢO CHẤT LƯỢNG VÀ CÁC CHUẨN CHẤT LƯỢNG	150
8.3 LẬP KẾ HOẠCH VÀ KIỂM SOÁT CHẤT LƯỢNG	153
8.3.1 Lập kế hoạch chất lượng	153
8.3.2 Kiểm soát chất lượng	155
8.4 MÔ HÌNH CMM/CMMI.....	156
8.4.1 CMM và CMMI là gì?.....	156
8.4.2 Cấu trúc của CMM	157
8.4.3 So sánh giữa CMM và CMMI	164
8.4.4 Lợi ích của CMM đem lại cho doanh nghiệp	165
TÓM LƯỢC.....	167
BÀI 9: QUẢN LÝ CẤU HÌNH.....	168
9.1 KẾ HOẠCH QUẢN LÝ CẤU HÌNH	168
9.2 QUẢN LÝ VIỆC THAY ĐỔI	169
9.3 QUẢN LÝ PHIÊN BẢN VÀ BẢN PHÁT HÀNH.....	170
9.3.1 Quản lý phiên bản	170
9.3.2 Quản lý phát hành.....	172
9.4 XÂY DỰNG HỆ THỐNG	173
9.5 CÁC CÔNG CỤ CASE CHO QUẢN TRỊ CẤU HÌNH	173
9.5.1 GitHub.....	173
9.5.2 Team Foundation Server.....	176
TÓM LƯỢC.....	184
PHỤ LỤC – ĐỒ ÁN MÔN HỌC.....	185
TÀI LIỆU THAM KHẢO.....	194

HƯỚNG DẪN

MÔ TẢ HỌC PHẦN

Bài giảng Kiểm thử phần mềm và đảm bảo chất lượng phần mềm trang bị cho sinh viên kiến thức về kiểm thử phần mềm, quy trình kiểm thử phần mềm và đảm bảo chất lượng phần mềm. Học phần được xây dựng với các nội dung: kiến thức về phương pháp kiểm thử phần mềm và các quy trình kiểm thử phần mềm; kỹ thuật và kỹ năng cơ bản trong thiết kế và cài đặt kiểm thử; cung cấp kiến thức về các công cụ hỗ trợ quản lý quá trình kiểm thử phần mềm; cung cấp kiến thức về kiểm thử tự động, các phần mềm hỗ trợ kiểm thử tự động, các khái niệm về đảm bảo chất lượng phần mềm và quản lý cấu hình. Qua học phần này cung cấp kiến thức và kỹ năng cho sinh viên để áp dụng trong quy trình kiểm thử phần mềm và đảm bảo chất lượng phần mềm. Với kiến thức của học phần này sinh viên có thể áp dụng để thực thi việc kiểm thử một phần mềm cụ thể để một phần mềm đảm bảo đạt chất lượng. Sinh viên có khả năng sử dụng thành thạo các công cụ phục vụ cho việc kiểm thử phần mềm và đảm bảo chất lượng phần mềm.

NỘI DUNG HỌC PHẦN

- BÀI 1. Tổng quan và yêu cầu về kiểm thử phần mềm và đảm bảo chất lượng phần mềm: Các mô hình phát triển phần mềm; Kiểm thử phần mềm và đảm bảo chất lượng phần mềm; Vai trò của kiểm thử và đảm bảo chất lượng phần mềm; Tài liệu về sản phẩm phần mềm; Yêu cầu kiểm thử và đảm bảo chất lượng; Các yếu tố cần thiết của kiểm thử và đảm bảo chất lượng phần mềm
- BÀI 2. Kỹ thuật thiết kế test-case. Giới thiệu về các khái niệm chính, thiết kế test case theo kỹ thuật kiểm thử black-box, white-box, các hệ thống quản lý test-case.
- BÀI 3. Thiết kế test case Black-box. Giới thiệu kỹ thuật phân lớp tương đương, phân tích biên, Phân tích ràng buộc, Mối quan hệ giữa hàm và dữ liệu, Chuyển trạng thái, Tổ hợp điều kiện. Bài tập thiết kế các test-case
- BÀI 4. Thiết kế test-case White-box. Giới thiệu bài tập thực hành black-box, bài tập thực hành white-box. Bài tập thiết kế các test-case

- BÀI 5. Lỗi phần mềm và Hệ thống quản lý bug. Giới thiệu tổng quan về lỗi phần mềm, những nguyên nhân gây ra lỗi thường gặp, các lỗi thường gặp trong phần mềm, tìm lỗi và phân tích lỗi. Giới thiệu về hệ thống quản lý bug, Vòng đời của bug trên hệ thống quản lý bug, Thực hành hệ thống quản lý bug Bugzilla.
- BÀI 6. Kiểm thử tự động. Giới thiệu về Automation testing, giới thiệu về action based testing. Các công cụ hỗ trợ kiểm thử tự động. Giới thiệu về các công cụ kiểm thử tự động Katalon Studio, Selenium.
- Bài 7. Kiểm thử đơn vị (Unit Test). Giới thiệu về kiểm thử đơn vị và các phương pháp kiểm thử đơn vị. Kiểm thử JUnit, NUnit và tạo dự án kiểm thử.
- Bài 8. Quản lý chất lượng phần mềm. Giới thiệu về chất lượng quá trình và chất lượng sản phẩm. Đảm bảo chất lượng và các chuẩn chất lượng. Lập kế hoạch và kiểm soát chất lượng.
- Bài 9. Quản lý cấu hình. Giới thiệu kế hoạch quản lý cấu hình, quản lý việc thay đổi. Quản lý phiên bản và bản phát hành. Xây dựng hệ thống. Các công cụ CASE cho quản trị cấu hình.

KIẾN THỨC TIỀN ĐỀ

Sinh viên cần hoàn thành các học phần: Cơ sở dữ liệu, Phân tích thiết kế hệ thống thông tin, Công nghệ Phần mềm trước khi học học phần này.

YÊU CẦU HỌC PHẦN

Người học cần đi học đầy đủ, đọc các nội dung sẽ được học trước khi đến lớp, làm các bài tập về nhà và đảm bảo thời gian tự học ở nhà.

CÁCH TIẾP NHẬN NỘI DUNG HỌC PHẦN

Để học tốt môn này, sinh viên cần ôn tập các bài đã học, trả lời câu hỏi và làm đầy đủ bài tập; đọc trước bài mới và tìm thêm các thông tin liên quan đến bài học.

Đối với mỗi bài học, người học đọc trước mục tiêu và tóm tắt bài học, sau đó đọc nội dung bài học. Kết thúc mỗi ý của bài học, người đọc trả lời câu hỏi ôn tập và kết thúc toàn bộ bài học, người đọc làm các bài tập.

PHƯƠNG PHÁP ĐÁNH GIÁ HỌC PHẦN

Môn học được đánh giá gồm hai thành phần.

- Điểm quá trình: 50%. Hình thức và nội dung do giáo viên quyết định, phù hợp với quy chế đào tạo và tình hình thực tế tại nơi tổ chức học tập.

Điểm thi: 50%. Lấy từ đồ án môn học.

BÀI 1: TỔNG QUAN VỀ KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

Nội dung gồm các phần sau:

- Giới thiệu tổng quan, các khái niệm cơ bản về kiểm thử và đảm bảo chất lượng phần mềm
- Các mô hình phát triển phần mềm
- Vai trò kiểm thử và đảm bảo chất lượng phần mềm
- Tài liệu sản phẩm phần mềm
- Qui trình kiểm thử và đảm bảo chất lượng phần mềm
- Các yếu tố cần thiết kiểm thử và đảm bảo phần mềm

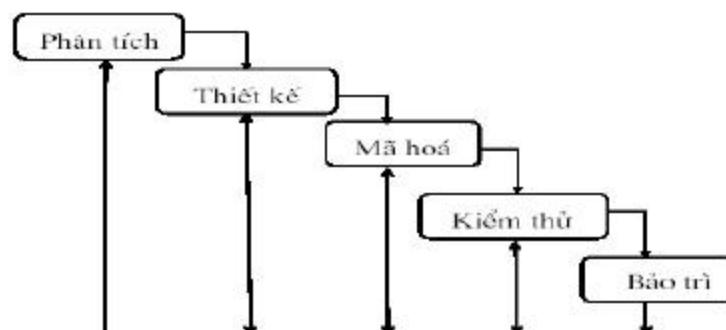
1.1 CÁC MÔ HÌNH PHÁT TRIỂN PHẦN MỀM

1.1.1 Water-fall Model

Các Mô hình thác nước là một trong những mô hình đầu tiên và phổ biến được áp dụng trong quá trình phát triển phần mềm. Mô hình này chia quá trình phát triển phần mềm thành những giai đoạn tuần tự. Mỗi giai đoạn sẽ có một mục đích nhất định. Kết quả của giai đoạn trước là thông tin đầu vào cho giai đoạn tiếp theo sau. Tùy theo qui mô của phần mềm cần phát triển mà mô hình thác nước sẽ có những biến thể khác nhau như sau:

- Qui trình 5 giai đoạn: Là qui trình cải tiến của qui trình phía trước bằng cách bổ sung thêm một giai đoạn mới sau giai đoạn lập trình nhằm tăng cường độ tin cậy của phần mềm.

- Xác định yêu cầu: Được tiến hành ngay khi có nhu cầu xây dựng phần mềm, xác định chính xác các yêu cầu phần mềm sẽ xây dựng.
- Phân tích: Tiến hành ngay sau khi kết thúc việc xác định yêu cầu, mô tả lại thế giới thực thông qua các mô hình trước khi thiết kế.
- Thiết kế: Tiến hành ngay sau khi kết thúc việc phân tích, mô tả các thành phần của phần mềm (mô hình của phần mềm) trước khi tiến hành cài đặt.
- Lập trình (cài đặt): Được tiến hành ngay sau khi kết thúc việc thiết kế, tạo lập phần mềm theo yêu cầu.
- Kiểm thử: Được tiến hành ngay sau khi đã có kết quả (từng phần) của việc lập trình, tăng độ tin cậy của phần mềm.
- Bảo trì: Công việc của giai đoạn bao gồm việc cài đặt và vận hành phần mềm trong thực tế, đảm bảo phần mềm vận hành tốt



Mô hình vòng đời cổ điển (thác nước)

Hình 1.1: Mô hình thác nước

Nhận xét:

Mô hình thác nước có thể dễ dàng phân chia quá trình xây dựng phần mềm thành những giai đoạn hoàn toàn độc lập nhau. Tuy nhiên, các dự án lớn hiếm khi tuân theo dòng chảy tuần tự của mô hình vì thường phải lặp lại các bước để nâng cao chất lượng. Hơn nữa, khách hàng hiếm khi tuyên bố hết yêu cầu trong giai đoạn phân tích.

Mô hình này cũng có hạn chế rất khó thực hiện các thay đổi một khi đã thực hiện xong giai đoạn nào đó. Điều này làm cho việc xây dựng phần mềm rất khó thay đổi các yêu cầu theo ý muốn của khách hàng. Do đó, phương pháp này chỉ thích hợp cho những trường hợp mà chúng ta đã hiểu rất rõ các yêu cầu của khách hàng.

Chú ý: Mô hình thác nước có thể được cải tiến bằng cách cho phép quay lui khi phát hiện lỗi trong giai đoạn phía trước.

1.1.2 Prototype Model

Tương tự như mô hình thác nước bổ sung vào các giai đoạn thực hiện phần mềm mẫu ngay khi xác định yêu cầu nhằm mục tiêu phát hiện nhanh các sai sót về yêu cầu. Các giai đoạn trong mô hình bản mẫu phần mềm có thể tiến hành lặp đi lặp lại chứ không nhất thiết phải theo trình tự nhất định.

Ngay sau giai đoạn xác định yêu cầu, nhà phát triển phần mềm đưa ra bản thiết kế sơ bộ và tiến hành cài đặt bản mẫu đầu tiên và chuyển cho người sử dụng. Bản mẫu này nhằm miêu tả cách thức phần mềm hoạt động cũng như cách người sử dụng tương tác với hệ thống.

Người sử dụng xem xét phản hồi thông tin cần thiết cho nhà phát triển. Nếu người sử dụng đồng ý với bản mẫu thì người phát triển sẽ tiến hành cài đặt. Ngược lại phải quay lại giai đoạn xác định yêu cầu. Công việc này lặp lại liên tục đến khi người sử dụng đồng ý với các bản mẫu do nhà phát triển đưa ra.

Như vậy đây là một hướng tiếp cận tốt khi yêu cầu chưa rõ ràng và khó đánh giá được tính hiệu quả của các thuật toán. Tuy nhiên, nhược điểm mô hình này là tính cấu trúc không cao do đó khách hàng dễ mất tin tưởng.

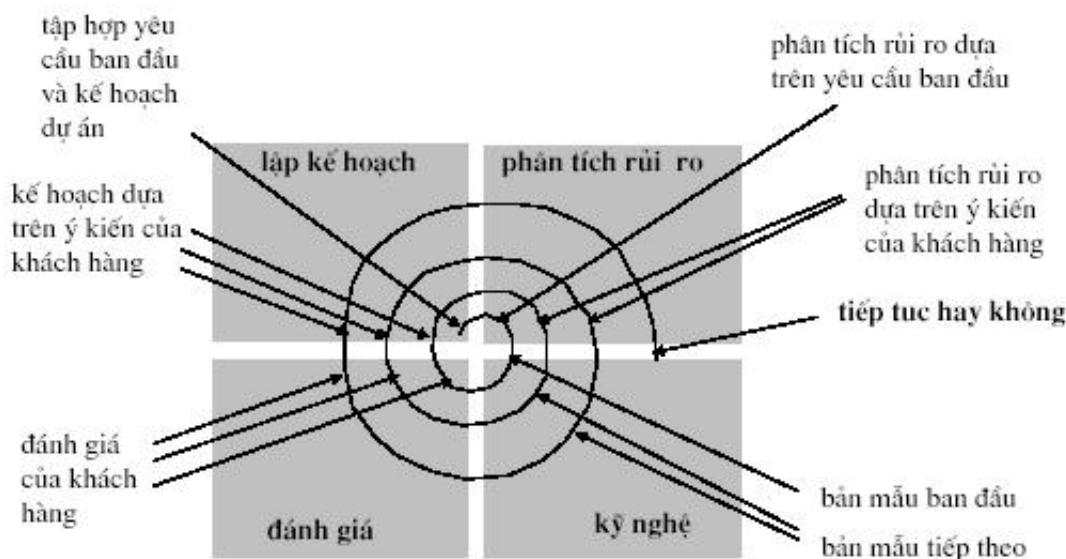


Hình 1.2: Mô hình bản mẫu

1.1.3 Spiral Model

Mô hình là sự kết hợp của mô hình bản mẫu thiết kế và mô hình thác nước được lặp lại nhiều lần. Ở lần lặp kế tiếp hệ thống được tìm hiểu và xây dựng hoàn thiện hơn ở lần lặp trước đó.

Ở mỗi lần lặp các yêu cầu người sử dụng được hiểu rõ ràng hơn và các bản mẫu phần mềm hoàn thiện hơn. Ngoài ra ở cuối mỗi lần lặp thêm công đoạn phân tích mức độ rủi ro để quyết định nên đi tiếp theo hướng này nữa hay không.



Hình 1.3: Mô hình xoắn ốc

Mô hình này phù hợp với các hệ thống phần mềm lớn do có khả năng kiểm soát rủi ro ở từng bước tiến hóa. Tuy nhiên vẫn chưa được sử dụng rộng rãi như mô hình thác nước hoặc bản mẫu đòi hỏi năng lực quản lý, năng lực phân tích rủi ro cao.

Mô hình xoắn ốc, được xem xét bởi Boehm (1988, 1998), đưa ra một phương pháp luận cải thiện nhằm phục vụ cho các dự án cỡ lớn và phức tạp.

Theo mô hình xoắn ốc, sự phát triển phần mềm được hiểu như làm một quy trình lặp; tại mỗi lần lặp, các hoạt động sau được thực hiện:

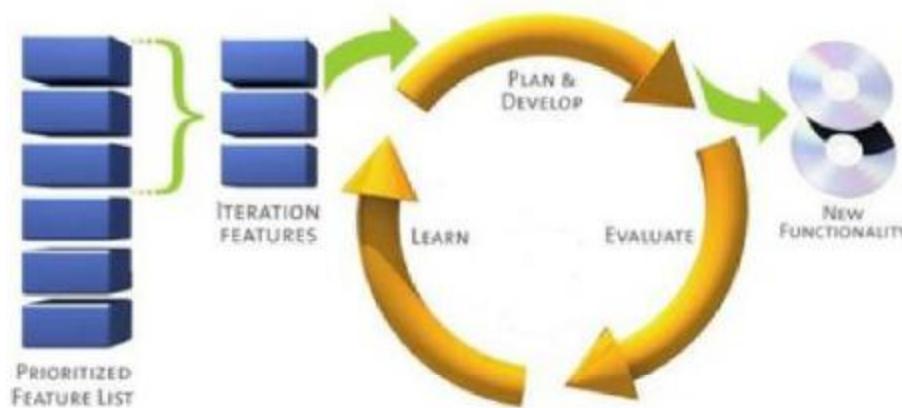
- Lập kế hoạch.
- Phân tích và giải quyết rủi ro.
- Các hoạt động công nghệ theo giai đoạn dự án: thiết kế, coding, test, cài đặt và chuyển giao.
- Sự đánh giá của khách hàng bao gồm các bình luận, các thay đổi và các yêu cầu bổ sung, ...

1.1.4 SDLC

SDLC (Software Development Life Cycle) là một mô hình truyền thống, 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.

1.1.5 Agile Model

Phương thức phát triển phần mềm Agile là một tập hợp các phương thức phát triển lặp và tăng dần trong đó các yêu cầu và giải pháp được phát triển thông qua sự liên kết công tác giữa các nhóm tự quản và liên chức năng. Agile là cách thức làm phần mềm linh hoạt để làm sao đưa sản phẩm đến tay người dùng càng nhanh càng tốt càng sớm càng tốt và được xem như là sự cải tiến so với những mô hình cũ như mô hình "Thác nước (waterfall)" hay "CMMI".



Hình 1.4: Mô hình Agile

1.1.6 Qui trình Hợp nhất (RUP)

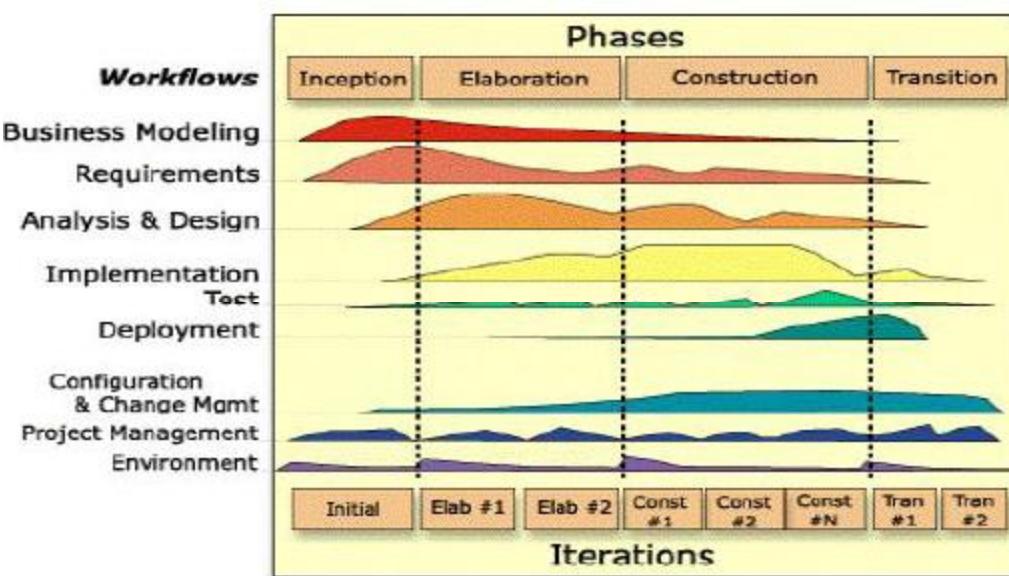
Quy trình phát triển phần mềm hợp nhất RUP (Rational Unified Process).

Quy trình (tiến trình) hợp nhất là sự mở rộng của tiến trình xoắn ốc, nhưng hình thức hơn và chặt chẽ hơn.

Qui trình bao gồm bốn giai đoạn chính và đan xen nhiều dòng hoạt động (activity flow) như là: Mô hình hóa nghiệp vụ, phân tích yêu cầu, phân tích và thiết kế, cài đặt, thử nghiệm triển khai, ... Mỗi giai đoạn được hình thành từ những bước lặp (iteration).

- Khởi tạo (inception):

- Thiết lập phạm vi dự án, các điều kiện ràng buộc phạm vi, các kiến trúc để xuất của hệ thống
- Xác định chi phí và thời gian của dự án
- Xác định độ rủi ro và môi trường hệ thống
- Xác định các thay đổi bổ sung, các tác động thay đổi này, các rủi ro nếu có,...



Hình 1.5: Mô hình RUP

- Tinh chế (elaboration):

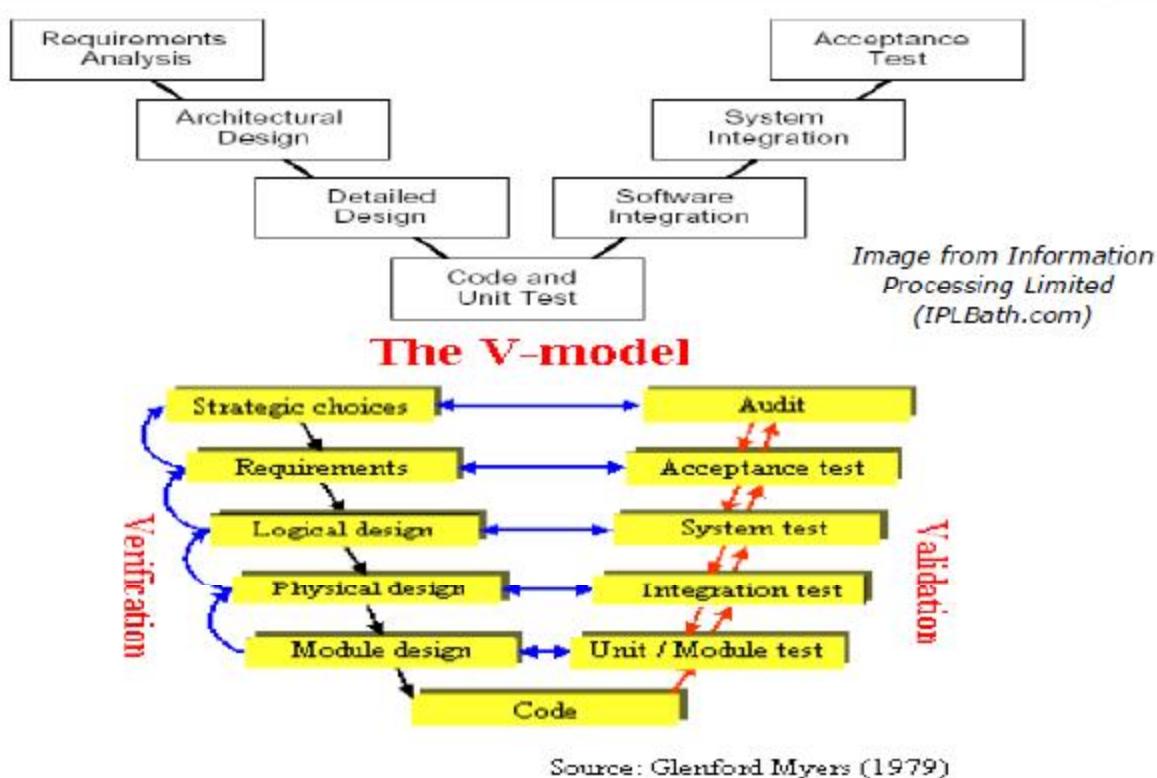
- Tinh chế kiến trúc hệ thống, yêu cầu hệ thống và đảm bảo kế hoạch sự ổn định của kế hoạch
- Đánh giá độ rủi ro, các thành phần sử dụng
- Xây dựng nền kiến trúc nền tảng hệ thống,...

- Xây dựng (construction) :

- Quản lý tài nguyên, kiểm soát và thực hiện tối ưu hóa

- Hoàn thành việc phát triển các thành phần của sản phẩm, thử nghiệm sản phẩm
- Đánh giá sản phẩm cài đặt từ các tiêu chuẩn đã được thoả thuận,...
- Chuyển giao (transition):
 - Thực hiện cài đặt hệ thống
 - Thủ nghiệm sản phẩm đã triển khai
 - Thu thập các phản hồi từ phía người dùng
 - Bảo trì hệ thống

1.1.7 V model



Hình 1.6: Mô hình chữ V

Các tính chất cần ghi nhận trên mô hình chữ V:

- Các hoạt động hiện thực và các hoạt động kiểm thử được tách biệt nhưng độ quan trọng là như nhau.
- Chữ V minh họa các khía cạnh của hoạt động Verification và Validation.

Cần phân biệt giữa các mức độ kiểm thử ở đó mỗi mức kiểm thử là kiểm thử trên mức phát triển phần mềm tương ứng.

1.2 KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

1.2.1 Phần mềm và chất lượng phần mềm

Phần mềm

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

1. Chương trình máy tính giúp vận hành thực thi các yêu cầu ứng dụng.
2. Những thủ tục được yêu cầu để định nghĩa theo thứ tự và lịch biểu 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
3. Các loại tài liệu cần thiết cho người phát triển, người sử dụng và người có nhiệm vụ duy trì.
4. 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.

Chất lượng phần mềm

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

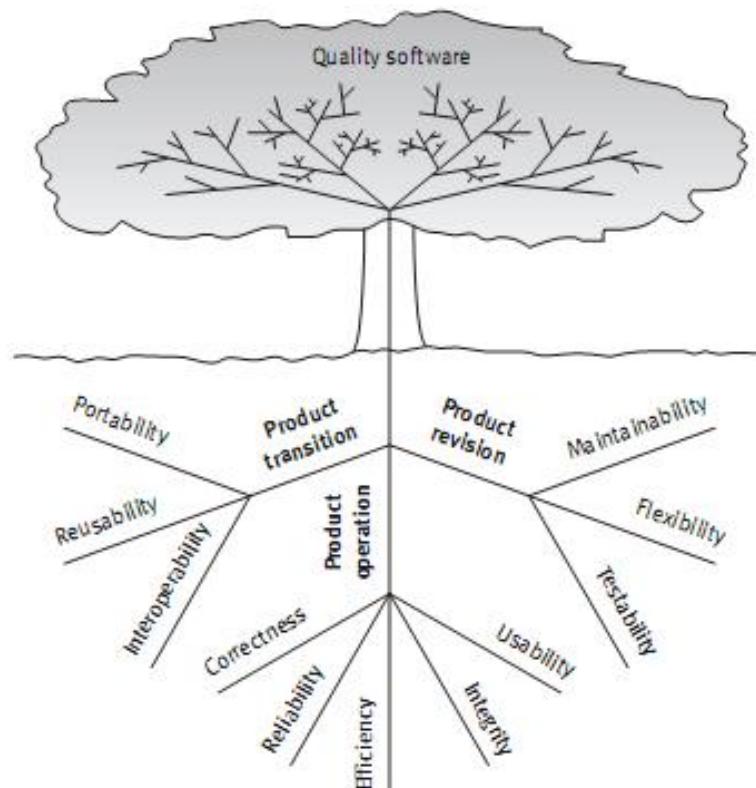
- Mức độ mà hệ thống, thành phần hoặc tiến trình đạt được yêu cầu đã đặc tả
- Mức độ mà hệ thống, thành phần hoặc tiến trình đạt được những nhu cầu hay mong đợi của khách hàng hoặc người sử dụng. Ban đầu đảm bảo chất lượng phần mềm có mục tiêu đạt được các yêu cầu đề ra, tuy nhiên thực tế phát triển phần mềm tồn tại nhiều ràng buộc đòi hỏi người phát triển cần tối ưu hóa công tác quản lý.

Theo Daniel Galin, 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 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.

1.2.2 Các yếu tố ảnh hưởng chất lượng phần mềm

Theo thời gian 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, Marcinia 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 kiểm tra được
- Các yếu tố chuyển giao bao gồm tính khả chuyển, có khả năng sử dụng lại, có khả năng tương tác.



Hình 1.7: Cây yêu tố ảnh hưởng chất lượng phần mềm của McCall

(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:** xác định trong 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.
- **Độ tin cậy:** 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:** giải quyết vấn đề về các tài nguyên phần cứng cần thiết để thực hiện tất cả các chức năng phù hợp của tất cả các yêu cầu khác. 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.
- **Các yêu cầu về tính toàn vẹn giải quyết:** các vấn đề về bảo mật hệ thống phần mềm, để ngăn chặn sự truy cập trái phép, để phân biệt giữa phần lớn nhân viên chỉ được phép xem thông tin với nhóm hạn chế người được phép thêm và thay đổi dữ liệu.
- **Các yêu cầu về khả năng sử dụng:** được đưa ra phạm vi của tài nguyên nhân lực cần thiết để đào tạo nhân viên mới và để vận hành hệ thống phần mềm.

(2) Các yếu tố rà soát sản phẩm: bảo trì được, linh động và kiểm tra được:

- **Khả năng bảo trì được:** 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 module của phần mềm, tài liệu chương trình nội bộ và hướng dẫn sử dụng của lập trình viên.
- **Tính linh động:** khả năng và nỗ lực cần thiết để hỗ trợ các hoạt động bảo trì. Chúng gồm các nguồn lực (manday) cần thiết để thích nghi với gói phần mềm, với các khách hàng trong cùng nghề, với các mức độ hoạt động khác nhau, với loại sản phẩm khác nhau. Các yêu cầu 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:** kiểm tra sự vận hành có tốt hay không của các hệ thống thông tin. Các yêu cầu liên quan tới các tính năng đặc biệt trong chương trình giúp người tester dễ dàng thực hiện công việc của mình hơn.

(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: 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: việc sử dụng các module phần mềm trong một dự án mới đang được phát triển mà các module 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 module đã có hoặc một nhóm các module 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 modules chất lượng cao hơn. 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: 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 có thể xác định cấu trúc đầu ra được chấp nhận như tiêu chuẩn trong ngành công nghiệp cụ thể hoặc lĩnh vực ứng dụng.

1.2.3 Khái niệm kiểm thử và đảm bảo chất lượng phần mềm

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 công 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 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).

Kiểm thử phần mềm là một tập hợp các tiến trình được thiết kế để đảm bảo mã hóa đúng đã được thiết kế, và không thực hiện điều không mong muốn. Đây là một pha quan trọng trong quá trình phát triển hệ thống, giúp cho người xây dựng hệ thống và khách hàng thấy được hệ thống mới đã đáp ứng yêu cầu đặt ra.

1.2.4 Mục tiêu của kiểm thử và đảm bảo chất lượng phần mềm

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).

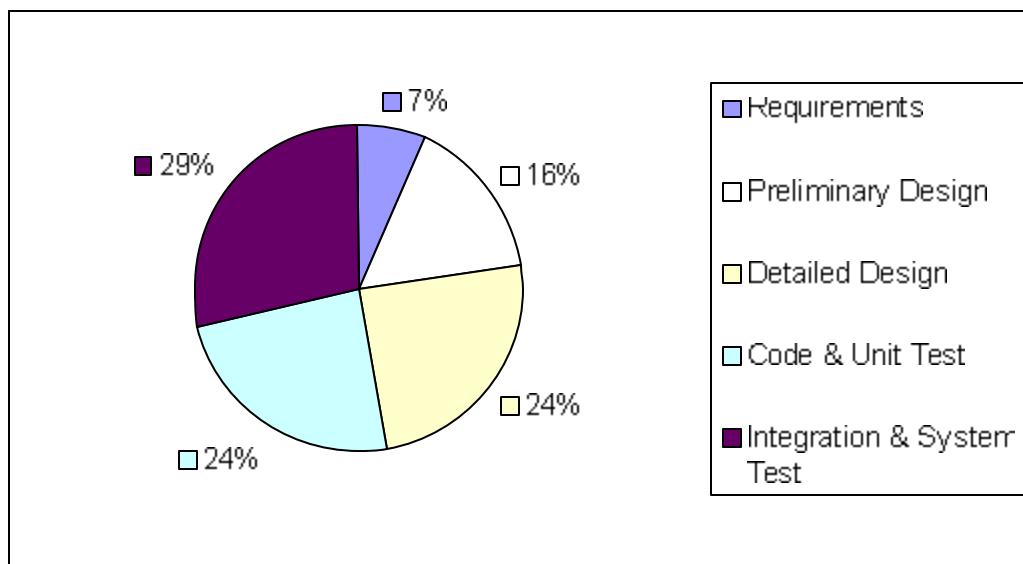
Tuỳ thuộc vào mục tiêu của kiểm thử, ta chia ra thành các mức như sau:

- 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.

1.2.5 Tâm quan trọng của kiểm thử và đảm bảo chất lượng phần mềm

Chi phí cho quá trình kiểm thử và tích hợp hệ thống của một dự án phần mềm chiếm khoảng 29% tổng kinh phí của dự án, nhiều nhất trong số những quá trình còn

lại bao gồm thiết kế ban đầu, thiết kế chi tiết, viết mã chương trình và kiểm thử từng chức năng, xác định yêu cầu.

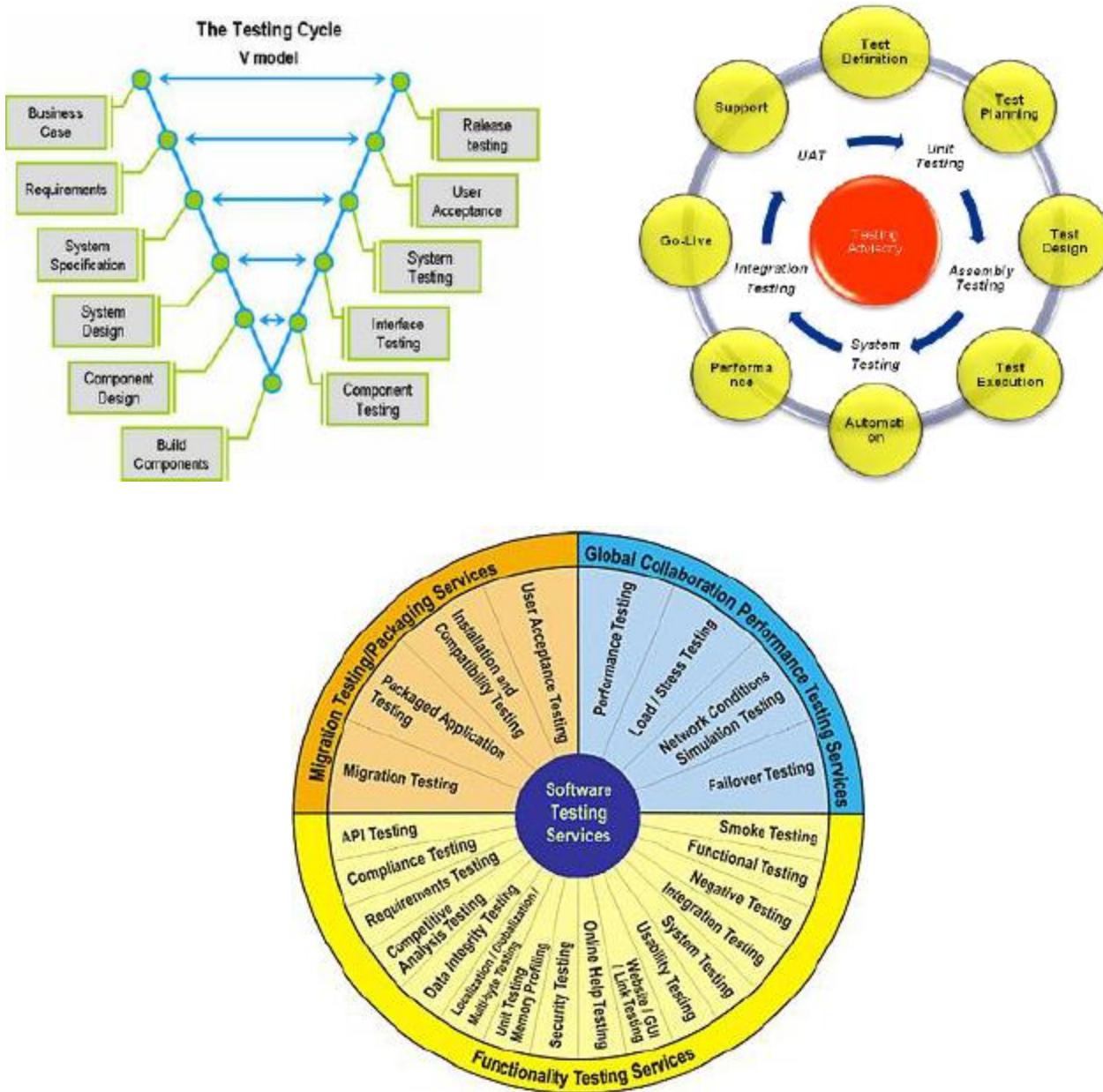


Hình 1.8: Chi phí phân bổ cho các tác vụ trong phát triển phần mềm

Trong quá trình kiểm thử, các tính năng của hệ thống được tích hợp với nhau dần dần, nhóm kỹ sư đảm bảo chất lượng (Quality Assurance) thường xuyên phải làm việc song song với nhóm kỹ sư lập trình (Developer) để chạy và kiểm tra những đoạn mã chương trình được tạo ra liên tục trong dự án. Hai cách tích hợp hệ thống khá phổ biến là *lắp ghép từ trên xuống* (top – down) và *lắp ghép từ dưới lên* (bottom – up). Đó là thời điểm cần nhiều nhân lực và kinh phí nhất trong cả quá trình phát triển dự án, và cũng là thời điểm có nhiều vấn đề nhất, vì lúc này áp lực rất lớn (do ngày giao sản phẩm sắp đến, nhưng chương trình có thể còn nhiều lỗi khi chạy thử). Ngoài ra, vấn đề về việc tạo động cơ thúc đẩy cho đội dự án cũng cần đặt ra, vì sắp kết thúc, hoặc các vấn đề về giải quyết những bất đồng để người sử dụng chấp nhận sản phẩm của đội dự án cũng cần thiết.

Việc đảm bảo chất lượng dự án là quá trình lâu dài hơn, gồm cả quá trình kiểm thử, quá trình kiểm soát các qui trình thực hiện những giai đoạn khác nhau, từ việc tìm hiểu yêu cầu ban đầu của khách hàng, đến việc triển khai và bàn giao và bảo trì sản phẩm. Việc đảm bảo chất lượng dự án là cách tốt nhất để nhìn vào bên trong dự án nhằm kiểm tra, kiểm soát, đảm bảo dự án phát triển tốt.

Đảm bảo chất lượng dự án là một trong những nội dung của chuẩn CMM mức 2, một chuẩn hiện nay được nhiều công ty áp dụng để phát triển qui trình quản lý.



Hình 1.9: Các tác vụ trong quản lý chất lượng dự án

1.2.6 Các nguyên tắc trong kiểm thử và đảm bảo chất lượng phần mềm

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

- Quy tắc 1: quan trọng ca kiểm thử là định nghĩa đầu ra hay kết quả mong muốn.
- Quy tắc 2: Lập trình viên nên tránh tự kiểm tra chương trình của mình.

- Quy tắc 3: Nhóm lập trình không nên kiểm thử chương trình của chính họ.
- Quy tắc 4: Kiểm tra thấu đáo mọi kết quả của mỗi kiểm tra.
- Quy tắc 5: Các ca kiểm thử phải được viết cho các trạng thái đầu vào không hợp lệ và không mong muốn, cũng như cho các đầu vào hợp lệ và mong muốn.
- Quy tắc 6: Khảo sát chương trình để xem liệu chương trình có thực hiện đúng phần cần thực hiện chỉ là một phần, phần còn lại liệu chương trình có thực hiện cái mà nó không cần phải thực hiện hay không.
- Quy tắc 7: Tránh các ca kiểm thử bằng quơ trừ khi chương trình thực sự là chương trình bâng quơ.
- Quy tắc 8: Không dự kiến kết quả kiểm thử theo giả thiết ngầm là không tìm thấy lỗi.
- Quy tắc 9: Xác suất tồn tại lỗi trong đoạn chương trình là tương ứng với số lỗi đã tìm thấy trong đoạn đó.
- Quy tắc 10: Kiểm thử là nhiệm vụ cực kỳ sáng tạo và có tính thử thách trí tuệ.

1.2.7 Các khái niệm liên quan kiểm thử và đàm bảo chất lượng phần mềm

Xác minh (Verification)

- **Xác minh** là quy trình xác định xem sản phẩm của một công đoạn trong quy trình phát triển phần mềm có thỏa mãn các yêu cầu đặt ra trong công đoạn trước hay không.
- Xác minh quan tâm tới việc ngăn chặn lỗi giữa các công đoạn
- Xác minh thường là hoạt động kỹ thuật và nó có sử dụng các kiến thức về các yêu cầu, các đặc tả rời rạc của phần mềm
- Các hoạt động của xác minh: Kiểm thử (Testing) và Rà soát lại (Review)

Thẩm định (Validation)

- Là tiến trình nhằm chỉ ra toàn bộ hệ thống đã phát triển xong phù hợp với tài liệu mô tả yêu cầu. Thẩm định là quá trình kiểm chứng chúng ta xây dựng phần mềm có đúng theo yêu cầu khách hàng.

- Thẩm định chỉ quan tâm đến sản phẩm cuối cùng không còn lỗi.

CÁC LOẠI KIỂM THỬ

❖ Kiểm tra đơn vị (unit test)

Bước này còn được gọi là kiểm thử các mô-đun của chương trình. Đây là loại kiểm tra theo mô hình hộp trắng nhưng trong một số trường hợp thì lại theo mô hình hộp đen. Người thực hiện việc kiểm tra này là các kỹ sư lập trình hay nhóm phát triển hệ thống, thường viết những đoạn mã ngắn bằng chính ngôn ngữ viết mã chương trình để kiểm tra. Những đoạn chương trình ngắn này còn được gọi là “test driver”, được thực hiện trong quá trình viết mã chương trình và kết thúc mỗi chức năng. Đôi khi việc kiểm thử một số chức năng được gộp nhóm lại với nhau được gọi là một bộ kiểm thử (test suites).

❖ Kiểm tra tích hợp (Integration Test)

Đây là bước kiểm thử giao diện tích hợp giữa các chức năng khác nhau trong hệ thống. Đây là bước tiếp theo sau việc kiểm thử từng chức năng. Việc kiểm thử từng chức năng chưa đầy đủ vì có thể riêng rẽ từng chức năng chạy tốt nhưng khi kết hợp lại thì lại không chạy được bởi vì lỗi có thể tiềm nấp trong một mô-đun nhưng lại thể hiện khi chạ mô-đun khác sau khi tích hợp các mô-đun với nhau. Loại kiểm thử này là theo mô hình hộp đen.

❖ Kiểm tra hệ thống (System Test)

Việc này sẽ thực hiện kiểm thử toàn bộ những chức năng của hệ thống thành một chuỗi thực hiện liên hoàn và kiểm thử một số những chức năng mang tính chất hệ thống (khi tích hợp hai hay nhiều chức năng với nhau thì đặc tính này chưa thể hiện được). Đây là một loại kiểm thử theo mô hình hộp đen.

❖ Kiểm tra chấp nhận (Acceptance Test)

Đây là mốc công việc cuối cùng trong giai đoạn kiểm thử, trong một số trường hợp còn được gọi là bản kiểm thử beta. Trong giai đoạn này người khách hàng cuối sẽ kiểm thử và ký vào biên bản chấp nhận sản phẩm nếu họ hài lòng với phần mềm và tất cả các yêu cầu ban đầu về sản phẩm bào giao. Những tiêu chí chấp nhận thực ra đã được thiết lập ngay trong đơn đặt hàng hay hợp đồng với khách hàng. Đó chính là những điều kiện mà phần mềm cần thỏa mãn để khách hàng chấp nhận sản phẩm.

❖ Kiểm thử hồi quy (Regression test)

Đây là việc chạy lại chương trình sau khi thực hiện những thay đổi tới phần mềm hoặc những thay đổi tới môi trường. Quá trình kiểm thử lại này có thể dùng các công cụ tự động thực hiện, rất hữu ích, đỡ tốn công sức của con người.

❖ Kiểm tra tính tương thích (Compatibility Test)

Đây là việc kiểm tra xem hệ thống có tương thích với các môi trường nền khác nhau chẳng hạn như kiểm tra xem chương trình có chạy tốt trong các trình duyệt khác nhau không, có chạy được trong Internet Explorer, Google Chrome có chạy được trong hệ điều hành Window hay Macintosh.

❖ Kiểm thử về khả năng tải và chịu áp lực của hệ thống phần mềm

Quá trình này sẽ đặt hệ thống vào trạng thái ngưỡng chịu tải và chịu áp lực của yêu cầu, thường sẽ thực hiện bằng việc chạy một đoạn chương trình ngắn tự động được thực hiện bởi nhóm cán bộ đảm bảo chất lượng khi việc kiểm thử các chức năng của hệ thống kết thúc.

Các thông số để đo khả năng này là thời gian đáp ứng nhỏ nhất chấp nhận được, số lượng người sử dụng đồng thời nhỏ nhất chấp nhận được và thời gian ngừng hoặc giảm công suất hoạt động của hệ thống nhỏ nhất chấp nhận được.

❖ Kịch bản kiểm thử

Kịch bản kiểm thử (test script) có hai dạng. Thứ nhất là tập các hướng dẫn thực hiện từng bước với mục đích dẫn dắt nhân viên kiểm thử thực hiện thành công việc kiểm tra phần mềm đó. Dạng thứ hai là một đoạn chương trình nhỏ phục vụ cho việc kiểm thử một cách tự động.

❖ Kiểm thử tĩnh (Static Testing)

Hầu hết tất cả các tài liệu quan trọng như bản đề xuất giải pháp của dự án, hợp đồng, lịch thực hiện công việc, yêu cầu của khách hàng đối với hệ thống, mã nguồn chương trình, mô hình dữ liệu, các kế hoạch kiểm thử đều cần duyệt lại. Một phương thức duyệt lại các công việc trong dự án là kiểm tra chéo giữa các thành viên (peer reviews). Đây là một phương pháp kiểm tra chéo, người này kiểm tra kết quả công việc và sản phẩm của hệ thống của một người khác cùng nhóm nhằm xác định những

lỗi và những thay đổi cần sửa. Việc kiểm tra chéo này có tác dụng giảm các lỗi sớm và hiệu quả, được lên kế hoạch bởi giám đốc dự án và được phân công trong các buổi họp và được ghi lại trong các văn bản tài liệu của dự án. Đây là một hoạt động để đảm bảo CMM cấp 3.

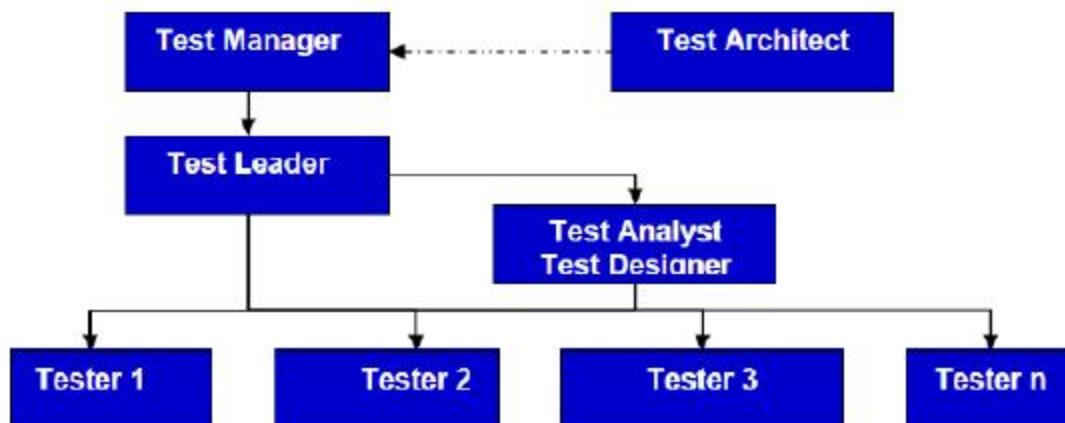
❖ Kiểm thử Thread testing

Sử dụng trong kiểm thử tích hợp để xác định khả năng của các chức năng chính bằng việc thử một chuỗi các units mà thực hiện một chức năng của hệ thống

1.2.8 Các đối tượng thực hiện kiểm thử và đảm bảo chất lượng phần mềm

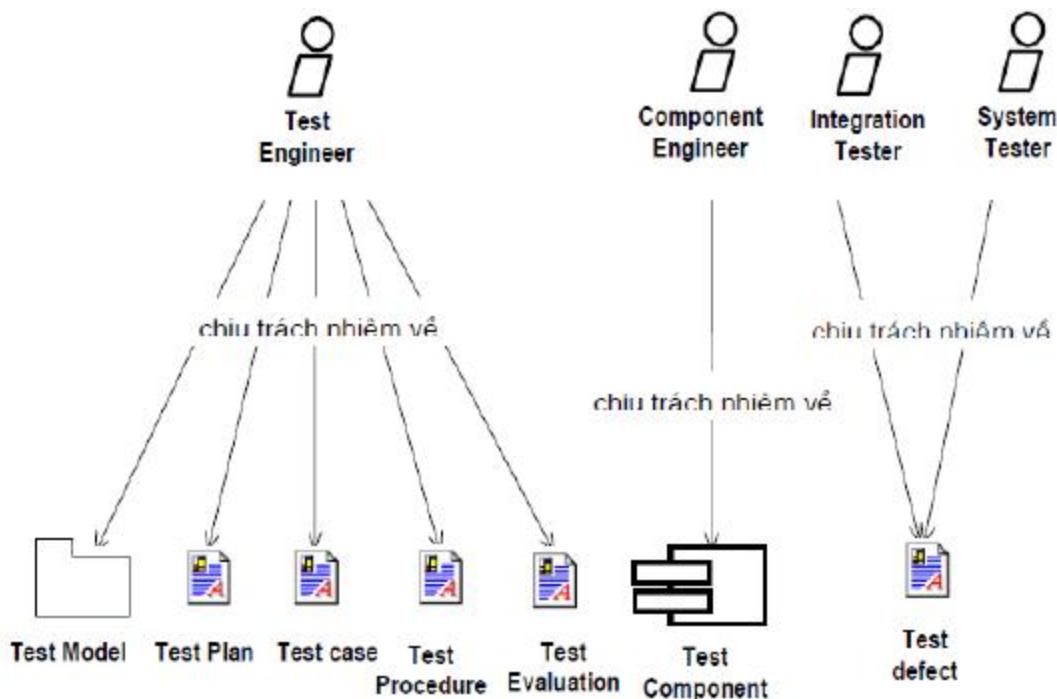
- *Vị trí nhóm trưởng:* tìm và tuyển người để đảm bảo đủ nguồn nhân lực cho đội ngũ QA, tạo các kế hoạch kiểm thử, lựa chọn công cụ nếu cần để thực hiện việc kiểm thử, và trách nhiệm cuối cùng là quản lý hoạt động của cả đội QA.
- *Vị trí kỹ sư kiểm thử:* thực hiện việc kiểm thử các chức năng, tạo ra các đoạn chương trình nhỏ để hướng dẫn kiểm thử hoặc thực hiện việc kiểm thử một cách tự động.
- *Vị trí quản trị hệ thống:* trách nhiệm chính là hỗ trợ cho nhóm QA làm việc nhưng lại không phải là một thành viên chính thức của nhóm.
- *Vị trí biên tập bản sao chép và soạn thảo các tài liệu cho dự án:* trách nhiệm chính cũng là hỗ trợ cho nhóm QA nhưng cũng không phải là một thành viên chính thức của nhóm.

Sơ đồ tổ chức của kiểm thử

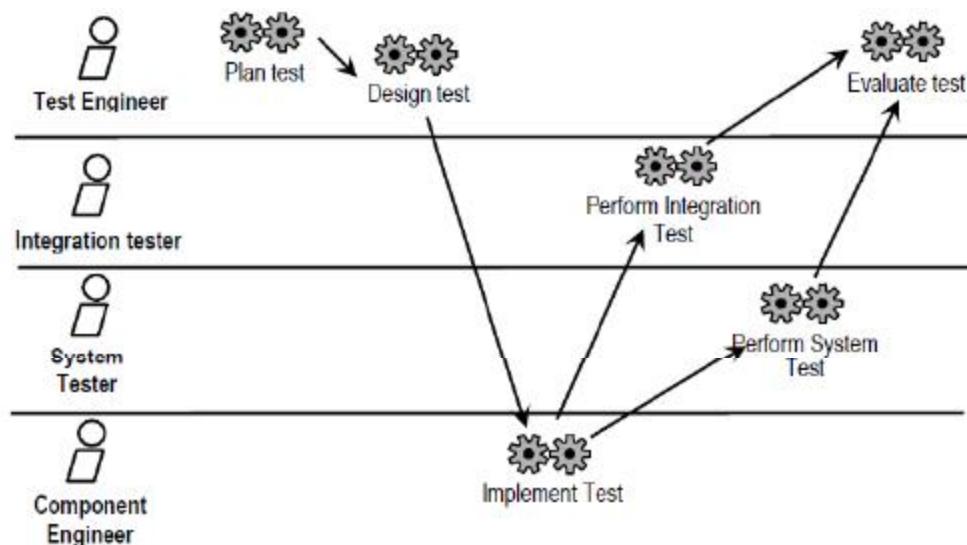


Hình 1.10: sơ đồ tổ chức kiểm thử

Worker và các quy trình



Hình 1.11: Worker và các qui trình



Hình 1.12: Worker và công việc kiểm thử và đảm bảo chất lượng phần mềm

1.2.9 Các điểm cần lưu ý khi kiểm thử và đảm bảo chất lượng phần mềm

- Chất lượng phần mềm không phải do khâu kiểm thử mà do thiết kế quyết định

2. Tính dễ kiểm thử phụ thuộc vào cấu trúc chương trình
3. Người kiểm thử nên làm việc độc lập với người phát triển phần mềm
4. Dữ liệu thử cho kết quả bình thường thì không có ý nghĩa nhiều, cần có những dữ liệu kiểm thử để phát hiện ra lỗi.
5. Khi phát sinh thêm trường hợp kiểm thử thì nên thử lại những trường hợp thử trước đó để tránh ảnh hưởng lan truyền sóng.

1.2.10 Các hạn chế của kiểm thử và đảm bảo chất lượng phần mềm

Ta không thể chắc là các đặc tả phần mềm đều đúng 100%.

- Ta không thể chắc rằng hệ thống hay tool kiểm thử là đúng.
- Không có công cụ kiểm thử nào thích hợp cho mọi phần mềm.
- Kỹ sư kiểm thử không chắc rằng họ hiểu đầy đủ về sản phẩm phần mềm.
- Ta không bao giờ có đủ tài nguyên để thực hiện kiểm thử đầy đủ phần mềm.
- Ta không bao giờ chắc rằng ta đạt đủ 100% hoạt động

1.3 VAI TRÒ CỦA KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

Kiểm thử phần mềm là qui trình chứng minh phần mềm không có lỗi.

- Mục đích của kiểm thử phần mềm là chỉ ra rằng phần mềm thực hiện đúng các chức năng mong muốn.
- Kiểm thử phần mềm là qui trình thiết lập sự tin tưởng về việc phần mềm hay hệ thống thực hiện được điều mà nó hỗ trợ.
- Kiểm thử phần mềm là qui trình thi hành phần mềm với ý định tìm kiếm các lỗi của nó.
- Kiểm thử phần mềm được xem là qui trình cố gắng tìm kiếm các lỗi của phần mềm theo tinh thần "hủy diệt".

Các mục tiêu chính của kiểm thử phần mềm:

Phát hiện càng nhiều lỗi càng tốt trong thời gian kiểm thử xác định trước.

- Chứng minh rằng sản phẩm phần mềm phù hợp với các đặc tả yêu cầu của nó.
- Xác thực chất lượng kiểm thử phần mềm đã dùng chi phí và nỗ lực tối thiểu.
- Tạo các testcase chất lượng cao, thực hiện kiểm thử hiệu quả và tạo ra các báo cáo vẫn đề đúng và hữu dụng.
- Kiểm thử phần mềm là một thành phần trong lĩnh vực rộng hơn, đó là Verification & Validation (V &V).

1.4 TÀI LIỆU VỀ SẢN PHẨM PHẦN MỀM

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.

1.4.1 Tài liệu dự án

Tài liệu liên quan đến sự tồn tại của dự án phát triển phần mềm, kế hoạch phạm vi dự án, lịch biểu dự án, ước lượng thời gian, chi phí dự án, báo cáo tiến độ dự án.

1.4.2 Tài liệu đặc tả

Software Requirement Specification là tài liệu Đặc tả yêu cầu phần mềm, những yêu cầu chính thức về những gì cần phải thực hiện của đội phát triển phần mềm. Tài liệu đặc tả yêu cầu nên bao gồm tất cả các định nghĩa về yêu cầu của người sử dụng và đặc tả yêu cầu của hệ thống. Tài liệu thiết lập thông tin hệ thống phải làm không phải việc mô tả làm việc như thế nào.

Yêu cầu phần mềm giải thích về nhu cầu của khách hàng để phát triển ứng dụng phần mềm. Khi không tạo ra các yêu cầu hoặc hiểu các yêu cầu, thì khó có thể lập kế hoạch kiểm thử phần mềm hoặc chiến lược kiểm thử hoặc các trường hợp kiểm thử hoặc kịch bản kiểm thử hoặc ma trận truy nguyên yêu cầu. Thông thường đội ngũ phát triển và đội kiểm thử hiểu được các yêu cầu từ các tài liệu như: Yêu cầu Hệ thống (SRS), Yêu cầu Chức năng (FRS), các trường hợp thảo luận với Chuyên viên Phân tích Kinh doanh và Chuyên gia Quản lý Thông minh (SME Management Experts)

- SRS: Đây là tài liệu có thông tin về hành vi hoàn chỉnh của hệ thống phần mềm. Nó cung cấp thông tin về phần cứng, phần mềm, thiết bị trung gian, yêu cầu chức năng (tổng quan), và các yêu cầu không phải chức năng (tổng quan)
- FRS: Đây là tài liệu có thông tin về các yêu cầu. Yêu cầu chức năng được giải thích một cách chi tiết. Trong một số dự án, FRS sẽ bao gồm chính SRS.
- Use cases: Các chức năng của phần mềm được giải thích với mục tiêu, các tác nhân, điều kiện tiên quyết, khóa học bình thường, khóa học bổ sung và khóa học đặc biệt

Lưu ý: Đôi khi, hiểu được yêu cầu là một điều khó khăn bởi vì thông tin chưa đầy đủ trong SRS, FRS, Use cases hoặc không có sẵn các tài liệu đó. Để có thêm kiến thức về miền (thể chấp, bảo hiểm, viễn thông, bán lẻ, kinh doanh dụng cụ tài chính) mà bạn đang làm việc, bạn có thể đọc sách và tìm kiếm trên internet.

1.4.3 Tài liệu đặc tả thiết kế

Thiết kế phần mềm đóng vai trò quan trọng trong suốt quá trình thiết kế, những kỹ sư phần mềm đề xuất các mô hình loại kế hoạch chi tiết cho giải pháp để có thể thực hiện được. Chúng ta có thể sử dụng kiểm tra và thẩm định thay thế những giải pháp và những đánh đổi (tradeoffs). Cuối cùng chúng ta sử dụng những mô hình kết quả để lên kế hoạch các hoạt động phát triển tiếp theo: thẩm định và kiểm thử hệ thống, thêm vào đó sử dụng chúng như là đầu vào hay là điểm bắt đầu của xây dựng và kiểm thử phần mềm.

Trong danh sách chuẩn vòng đời phát triển phần mềm như ISO/IEC/IEEE Software Life Cycle Process, thiết kế phần mềm bao gồm 2 hoạt động tương ứng với phân tích yêu cầu phần mềm và xây dựng phần mềm:

Thiết kế kiến trúc: phát triển mức kiến trúc và đưa ra cách tổ chức phần mềm và các thành phần khác nhau trong phần mềm.

Thiết kế chi tiết: chi tiết và đầy đủ thành phần tạo điều kiện xây dựng phần mềm trong pha sau đó

Đầu ra của thiết kế phần mềm sẽ được sử dụng cho xây dựng và kiểm thử nên việc đánh giá một thiết kế có phù hợp rất quan trọng, nếu một thiết kế sai sẽ dẫn đến tất cả các quá trình sau đó cũng sai và cần chỉnh sửa nếu thiết kế được chỉnh sửa.

1.4.4 Tài liệu kiểm thử

Kế hoạch kiểm thử dự án hay kế hoạch đảm bảo chất lượng dự án được mô tả giai đoạn cuối của quá trình tìm hiểu yêu cầu hệ thống. Tài liệu này gồm:

- Mục đích của đảm bảo chất lượng dự án; những tài liệu liên quan có thể tham khảo; việc quản lý chất lượng dự án được tiến hành như thế nào;
- Các chuẩn, các thực tế, các luật được đặt ra cho việc lập trình hoặc kiểm thử, cấu hình, đơn vị đo lường cho chất lượng và việc thực hiện các kiểm thử.
- Việc kiểm tra (reviews) và duyệt lại (audits) các qui trình thực hiện công việc của các giai đoạn, xem lại các công việc cụ thể bao gồm xem lại phần yêu cầu của dự án, kế hoạch kiểm thử, mã nguồn chương trình, và xem lại những kinh nghiệm đúc kết được từ những dự án trước.
- Quản lý những rủi ro của dự án: gắn phần rủi ro của đảm bảo chất lượng dự án với bản kế hoạch quản lý toàn bộ các rủi ro của dự án.
- Báo cáo các vấn đề nảy sinh và các hành động sửa đổi
- Liệt kê và hướng dẫn các công cụ, kỹ thuật và phương pháp luận để đảm bảo chất lượng dự án
- Thu thập và lưu trữ các báo cáo của tất cả các giai đoạn

Chất lượng của phần mềm được kiểm soát dựa trên hai yếu tố sau:

- *Khả năng lần vết để tìm mối liên hệ giữa các sản phẩm khác nhau của dự án*, ví dụ tìm xem sự tương thích giữa yêu cầu của khách hàng với bản thiết kế, với các trường hợp kiểm thử đến mức độ nào, có hoàn toàn giống nhau.

- Thực hiện những kiểm tra xem xét chính thức cuối mỗi giai đoạn của chu trình phát triển hệ thống phần mềm.

1.5 YÊU CẦU KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG

Mô tả những yêu cầu được kiểm thử trong AUT (Application under test)

- Yêu cầu chức năng: những yêu cầu chức năng mà ứng dụng nên làm
- Yêu cầu phi chức năng: những yêu cầu đối thuộc tính mà chức năng nên hoặc trông như thế nào. Có 3 loại yêu cầu phi chức năng:
 - Look n feel
 - Boundary
 - Negative

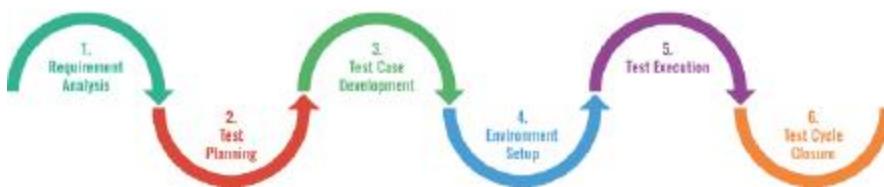
Định dạng có thể khác nhau. Chúng có thể duy nhất mỗi công ty và người viết báo cáo. Tài liệu hy vọng đầy đủ thông tin hữu ích:

- Target Platform
- Những yêu cầu hệ thống
- Mô tả chức năng
- Cách sử dụng
- Vùng vẫn đề mong đợi

1.5.1 Quy trình kiểm thử và đảm bảo chất lượng phần mềm

Quy trình kiểm thử phần mềm xác định các giai đoạn/ pha trong kiểm thử phần mềm. Tuy nhiên, không có STLC tiêu chuẩn cố định nào trên thế giới, nhưng về cơ bản quy trình kiểm thử bao gồm những giai đoạn sau:

Software Testing Life Cycle (STLC)



Hình 1.13: Vòng đời kiểm thử phần đờ

1. Requirement analysis - Phân tích yêu cầu

2. Test planning - Lập kế hoạch kiểm thử

3. Test case development - Thiết kế kịch bản kiểm thử

4. Test environment set up - Thiết lập môi trường kiểm thử

5. Test execution - Thực hiện kiểm thử

6. Test cycle closure - Đóng chu trình kiểm thử

Các giai đoạn kiểm thử được thực hiện một cách tuần tự. Mỗi giai đoạn sẽ có những mục tiêu khác nhau, đầu vào và kết quả đầu ra khác nhau nhưng mục đích cuối cùng vẫn là đảm bảo chất lượng sản phẩm phần mềm tốt nhất. Sau đây, chi tiết thông tin về các hoạt động, ai là người thực hiện, đầu vào, đầu ra của từng giai đoạn trong quy trình kiểm thử phần mềm.

Requirements analysis - Phân tích yêu cầu

Đầu vào: tài liệu đặc tả yêu cầu, tài liệu thiết kế hệ thống, tài liệu khách hàng yêu cầu về các tiêu chí chấp nhận của sản phẩm, bản prototype của khách hàng yêu cầu(nếu có),...

Hoạt động:

Giai đoạn đầu tiên trong quy trình kiểm thử phần mềm.

QA team sẽ đọc hiểu, nghiên cứu và phân tích cụ thể các yêu cầu trong tài liệu đặc tả của dự án hoặc tài liệu khách hàng. Qua đó, QA team sẽ nắm bắt yêu cầu gồm yêu cầu kiểm thử chức năng/phi chức năng.

Ngoài ra, nghiên cứu tài liệu, nếu có câu hỏi phát sinh hay đề xuất giải quyết, QA team sẽ đưa ra câu hỏi với các bên liên quan: BA (Business Analysis), PM (Project Manager), team leader, khách hàng hiểu chính xác hơn về yêu cầu của sản phẩm. Những câu hỏi được lưu trữ vào tập tin Q&A (Question and Answer). Hơn nữa, đối với khách hàng không có sự hiểu biết về lĩnh vực phần mềm mà họ yêu cầu thì càng

không thể trả lời những câu hỏi mang tính chuyên môn cao. Chính chúng ta sẽ là người hỗ trợ và đưa ra giải pháp thích hợp cho khách hàng lựa chọn.

Đầu ra: tài liệu chứa các câu hỏi và câu trả lời liên quan đến nghiệp vụ của hệ thống, tài liệu báo cáo tính khả thi, phân tích rủi ro của việc kiểm thử phần mềm.

Test planning - Lập kế hoạch kiểm thử

Đầu vào: các tài liệu đặc tả đã được cập nhật thông qua các câu hỏi và trả lời được đưa ra trong giai đoạn phân tích yêu cầu, tài liệu báo cáo tính khả thi, phân tích rủi ro của việc kiểm thử phần mềm.

Hoạt động:

Test manager hoặc test leader là người lập kế hoạch kiểm thử cho cả QA team. Lập kế hoạch kiểm thử nhằm xác định một số yếu tố quan trọng sau:

Xác định phạm vi (Scope) dự án: thời gian dự án, những công việc theo thời gian, tạo lịch trình thực hiện phù hợp với toàn bộ dự án.

Xác định phương pháp tiếp cận: ví dụ như: Thời gian cho phép test có phù hợp, yêu cầu chất lượng: cao, thấp hay khắc khe. Công nghệ/kỹ thuật phát triển ứng dụng, lĩnh vực của hệ thống/sản phẩm (domain). Từ đó, test manager có thể đưa ra những phương pháp và kế hoạch phù hợp nhất cho cả quá trình thực hiện dự án cho đúng với các tiêu chí chấp nhận của sản phẩm và kịp tiến độ với các mốc thời gian bàn giao, phát hành.

Xác định các nguồn lực

Con người: người tham gia dự án, ai test, bao nhiêu tester, kinh nghiệm.

Thiết bị: số lượng server, version, máy tính, mobile để thực hiện test,

Lên kế hoạch thiết kế công việc test: Bản kế hoạch kiểm thử gồm các nội dung:

Liệt kê các chức năng cần kiểm thử, cần làm những công việc gì, thời gian, độ ưu tiên, người thực hiện.

Xác định điều kiện bắt đầu: điều kiện tối thiểu bắt đầu từng chức năng.

Xác định điều kiện kết thúc: điều kiện kết thúc việc kiểm thử.

Đầu ra: test plan, test estimation, test schedule.

Test case development - Thiết kế kịch bản kiểm thử

Đầu vào: test plan, test estimation, test schedule, các tài liệu đặc tả

Hoạt động:

Review tài liệu: xem lại tất cả các tài liệu để xác định công việc cần làm, chức năng nào cần test, chức năng nào không cần test lại nữa. Từ đó, vừa có thể tiết kiệm thời gian mà kịch bản kiểm thử đầy đủ và hiệu quả.

Viết test case / check list: Sau đó, tester viết test case dựa vào kế hoạch đã đưa ra và vận dụng các kỹ thuật thiết kế kịch bản kiểm thử. Test case cần bao phủ được tất cả các trường hợp kiểm thử có thể xảy ra cũng như đáp ứng đầy đủ các tiêu chí của sản phẩm. Đồng thời tester cũng cần đánh giá mức độ ưu tiên cho từng test case.

Chuẩn bị dữ liệu kiểm thử: với test case, đội kiểm thử chuẩn bị trước các dữ liệu kiểm thử như test data, test script.

Review test case / check list: đội kiểm thử hoặc test leader review lại test case để bổ sung, hỗ trợ tránh những sai sót trong thiết kế test case và rủi ro.

Đầu ra: test design, test case, check list, test data, test automation script.

Test environment set up - Thiết lập môi trường kiểm thử

Đầu vào: test plan, smoke test case, test data.

Hoạt động:

Môi trường kiểm thử sẽ được quyết định dựa trên những yêu cầu của khách hàng, hay đặc thù của sản phẩm ví dụ như server/ client/ network,...

Tester chuẩn bị một vài test case kiểm tra môi trường cài đặt. Đây là việc thực thi các smoke test case.

Đầu ra: môi trường cài đặt đúng yêu cầu, kết quả của smoke test case.

Test execution - Thực hiện kiểm thử

Đầu vào: test plan, test design, test case, checklist, test data, test script.

Hoạt động:

Thực hiện các test case và mức độ ưu tiên trên môi trường cài đặt.

So sánh với kết quả mong đợi sau báo cáo các bug xảy ra lên tool quản lý lỗi và theo dõi trạng thái của lỗi đến khi được sửa thành công.

Thực hiện re-test để xác nhận các bug được fix và kiểm thử hồi qui khi có sự thay đổi liên quan.

Trong quá trình thực hiện kiểm thử, kiểm thử viên hỗ trợ, đề xuất cho cả đội dự án giải pháp hợp lý và kết hợp công việc hiệu quả.

Đo và phân tích tiến độ: kiểm thử viên cũng cần kiểm soát chặt chẽ tiến độ công việc như so sánh tiến độ thực tế với kế hoạch, nếu chậm cần phải điều chỉnh cho kịp tiến độ dự án, nếu nhanh điều chỉnh vì test lead lên kế hoạch chưa sát với thực tế dự án. Từ đó có thể sửa chữa test plan cần điều chỉnh để phù hợp với tiến độ dự án đưa ra.

Report thường xuyên cho PM và khách hàng về tình hình thực hiện dự án: Cung cấp thông tin trong quá trình kiểm thử đã làm được những chức năng nào, còn chức năng nào, hoàn thành được bao nhiêu phần trăm công việc, báo cáo các trường hợp phát sinh sớm, tránh ảnh hưởng tiến độ công việc của cả ngày.

Đầu ra: test results (kết quả kiểm thử), defect reports(danh sách các lỗi tìm được).

Test cycle closure - Đóng chu trình kiểm thử

Đầu vào: tài liệu phân tích đặc tả yêu cầu, test plan, test results, defect reports, tài liệu Q&A,...

Hoạt động:

Đây là giai đoạn cuối cùng trong quy trình kiểm thử phần mềm.

Ở giai đoạn này, QA team thực hiện tổng kết, báo cáo kết quả về việc thực thi test case, bao nhiêu case pass/fail, bao nhiêu case đã được fix, mức độ nghiêm trọng của lỗi, bao nhiêu lỗi cao/thấp, lỗi còn ở chức năng nào, dev nào nhiều lỗi. Chức năng đã hoàn thành test/chưa hoàn thành test/trễ tiến độ bàn giao.

Đánh giá các tiêu chí hoàn thành như phạm vi kiểm tra, chất lượng, chi phí, thời gian, mục tiêu kinh doanh quan trọng.

Ngoài ra, giai đoạn này cũng thảo luận tất cả những điểm tốt, điểm chưa tốt và rút ra bài học kinh nghiệm cho những dự án sau, giúp cải thiện quy trình kiểm thử.

Đầu ra: Test report, Test results (final).

1.5.2 Thuộc tính yêu cầu phần mềm

Một đặc tả sản phẩm được xây dựng tốt cần có tám thuộc tính sau đây:

- **Đầy đủ.** Liệu đặc tả còn thiếu cái gì không. Đầy đủ chi tiết chưa. Liệu nó đã bao gồm mọi điều cần thiết để không phụ thuộc vào tài liệu khác.
- **Trúng đích.** Liệu nó đã cung cấp lời giải đúng đắn cho bài toán, liệu nó đã xác định đầy đủ các mục tiêu và không có lỗi.
- **Chính xác, không nhập nhằng và rõ ràng.** Mô tả có chính xác không, có rõ ràng và dễ hiểu không, liệu cần có gì là nhập nhằng với ý nghĩa không xác định.
- **Tương thích.** Các đặc trưng và chức năng được mô tả có bị xung đột với nhau.
- **Hợp lệ.** Các khảng định có thực sự cần thiết để mô tả đặc trưng sản phẩm không. Có gì thừa không. Có thể truy ngược về yêu cầu của người dùng không.
- **Khả thi.** Liệu đặc tả có thể được cài đặt trong khuôn khổ nhân lực, công cụ, tài nguyên, thời gian và kinh phí cho phép hay không.
- **Phi mã lệnh.** Trong đặc tả không được dùng các câu lệnh hoặc thuật ngữ cho người lập trình. Ngôn ngữ dùng trong đặc tả phải là phổ biến với người dùng.
- **Khả kiểm thử.** Liệu các đặc trưng có thể kiểm thử được. Liệu đã cung cấp đủ thông tin để có thể kiểm thử và xây dựng các ca kiểm thử.

1.5.3 Các thành phần chính trong bản kế hoạch kiểm thử

- **[1] Mục đích và phạm vi kiểm thử:** Đặc tả mục đích của tài liệu về kế hoạch kiểm thử. Cung cấp văn tắt về phạm vi mà project được hỗ trợ như platform, loại database, hay danh sách văn tắt về các loại project con trong project kiểm thử
- **[2] Cách tiếp cận và các chiến lược được dùng:** Đặc tả về phương pháp luận kiểm thử sẽ được dùng để thực hiện kiểm thử. Ví dụ: Tổng quan về Testing Process Approach cho Project ABC . Đề cập các cấp độ kiểm thử cần thực hiện Các kỹ thuật được dùng cho mỗi kiểu kiểm thử trong project: Kiểm thử tích hợp (Integration Testing). Kiểm thử hệ thống (System Testing). Kiểm thử độ chấp thuận (Acceptant Testing). Kiểm thử chức năng của người dùng (Functionality Testing). Kiểm thử hồi quy (Regrsstion Testing). Kiểm thử việc phục hồi sau lỗi

(Failover and Recovery Testing). Kiểm thử việc kiểm soát an ninh và truy xuất (Security and Access Control Testing). Kiểm thử việc cấu hình và cài đặt (Configuration and Installation Testing). Kiểm thử đặc biệt. Kiểm thử hiệu xuất (Performance Testing)

- **[3] Các tính chất cần được kiểm thử:** Danh sách các tính chất của phần mềm cần được kiểm thử, đây là catalog chưa tắt cả các testcase (bao gồm chỉ số testcase, tiêu đề testcase) cũng như tắt cả các trạng thái cơ bản.
- **[4] Các tính chất không cần được kiểm thử:** Danh sách các vùng phần mềm được loại trừ khỏi kiểm thử, cũng như các testcase đã được định nghĩa nhưng không cần kiểm thử.
- **[5] Rủi ro và các sự cố bất ngờ:** Danh sách tất cả rủi ro có thể xảy ra trong chu kỳ kiểm thử. Phương pháp mà ta cần thực hiện để tối thiểu hóa hay sống chung với rủi ro.
- **[6] Tiêu chí định chỉ và phục hồi kiểm thử:** Tiêu chí định chỉ kiểm thử là các điều kiện mà nếu thỏa mãn thì kiểm thử sẽ dừng lại. Tiêu chí phục hồi là những điều kiện được đòi hỏi để tiếp tục việc kiểm thử đã được ngừng trước đó.
- **[7] Môi trường kiểm thử:** Đặc tả đầy đủ về các môi trường kiểm thử, bao gồm đặc tả phần cứng, phần mềm, mạng, database, hệ điều hành và các thuộc tính môi trường khác ảnh hưởng đến kiểm thử.
- **[8] Lịch kiểm thử:** Lịch kiểm thử ở dạng ước lượng, nên chứa các thông tin: các cột mốc với ngày xác định + Kết quả phân phối của từng cột mốc
- **[9] Tiêu chí dừng kiểm thử và chấp nhận:** Bất kỳ chuẩn chất lượng mong muốn nào mà phần mềm phải thỏa mãn cho việc phân phối đến khách hàng. Có thể bao gồm các thứ sau:Các yêu cầu mà phần mềm phải được kiểm thử dưới các môi trường xác địnhSố lỗi tối thiểu ở các cấp an ninh và ưu tiên khác nhau, số phủ kiểm thử tối thiểuKý kết và đồng thuận của các bên liên quan
- **[10] Nhân sự:** Vai trò và trách nhiệm từng người: Danh sách các vai trò xác định của các thành viên đội kiểm thử trong hoạt động kiểm thử.Các trách nhiệm của từng vai trò.Công tác huấn luyện. Danh sách các huấn luyện cần thiết cho các QC.

- **[11] Các tiện ích phục vụ kiểm thử:** Danh sách tất cả các tiện ích cần dùng trong suốt chu kỳ kiểm thử. Với project kiểm thử tự động, các tiện ích cần được liệt kê với chỉ số version cùng thông tin license.
- **[12] Các kết quả phân phôi:** Danh sách tất cả tài liệu hay artifacts dự định phân phôi nội bộ sau khi mỗi cột mốc kết thúc hay sau khi project kết thúc.

1.6 CÁC YÊU TỐ CẦN THIẾT CỦA KIỂM THỬ VÀ ĐÁM BẢO CHẤT LƯỢNG PHẦN MỀM

Kiểm thử là khâu cuối cùng trước khi chuyển sản phẩm đến khách hàng. Bởi vậy, người kiểm thử có vai trò quan trọng trong sự thành công của dự án và chất lượng sản phẩm. Một số kinh nghiệm để có thể trở thành kỹ sư kiểm thử giỏi:

[1] Cân bằng

Kỹ sư kiểm thử giỏi sẽ luôn biết cách cân bằng sao cho hợp lý trong những tình huống mà họ gặp phải cũng như luôn biết việc cân bằng trong tất cả các công việc mình làm, phải biết cân bằng giữa việc muốn tìm hiểu vẫn đề với yêu cầu phải ra quyết định cũng như nhu cầu hoàn thành công việc.

[2] Thực hành thường xuyên

Trở thành kỹ sư kiểm thử giỏi luôn kiểm thử toàn bộ mọi thứ của sản phẩm chứ không chỉ dừng lại ở tính năng của nó.

[3] Lắc léo

Thay vì chỉ dựa vào checklist hay hướng dẫn có sẵn, kỹ sư kiểm thử luôn hình dung ra nhiều cách khác nhau để tấn công sản phẩm.

[4] Sắp xếp thứ tự ưu tiên

Người kiểm thử luôn biết phải kiểm thử phần nào trước sau và phạm vi kiểm thử, có khả năng tìm bug nhiều nhất và ảnh hưởng khách hàng nhiều được thực thi trước.

Bạn phải nắm được khách hàng là ai. Các bên liên quan đến sản phẩm đều bị ảnh hưởng bởi những con bug bạn tìm được hoặc không tìm được.

[5] Khả năng quan sát

Những kỹ sư kiểm thử giỏi luôn để mắt quan sát những điều bất thường trong quá trình kiểm thử của mình. Những điều bất thường này có khi chỉ là những lỗi lập trình

đơn giản, có khi là cả một "ổ" bug, có khi bug mà trước giờ nhiều người vẫn chưa mô phỏng lại được. Hãy ghi nhận lại tất cả những bất thường quan sát được. Để làm được vậy người kỹ sư kiểm thử phải nắm được sản phẩm của mình từ trong ra ngoài. Bên cạnh đó phải luyện tập thường xuyên kỹ năng quan sát. Để tâm vào chi tiết và phát hiện những điều có vẻ hơi bất bình thường một chút.

[6] Sự chính xác

Khi những kỹ sư kiểm thử giỏi tìm được bug, giảm thiểu số bước cần thiết để mô phỏng bug mức tối thiểu. Họ cũng thường kiểm thử thêm xung quanh bug để hiểu thêm bug. Kiểm thử giỏi khi báo cáo bug chỉ rõ ra chỗ nào là ngầm định, chỗ nào là họ thực tế quan sát được.

1.7 CÁCH VIẾT YÊU CẦU KIỂM THỬ

Công đoạn test có các phase sau: Kế hoạch test, thiết kế test, tạo testcase, thực hiện test, báo cáo test.

Trong đó: "kế hoạch test" và "thiết kế test" là phase quan trọng để phát hiện ra lỗi và xác nhận chất lượng.

(1) Xác nhận mục đích của kiểm thử

- Xem bản kế hoạch tổng thể kiểm thử để xác nhận mục đích của kiểm thử (đặc tính chất lượng, những điểm quan trọng ...).
- Quyết định phạm vi của test, nội dung của test, phương pháp test.

(2) Tạo danh sách chức năng

- Đưa ra toàn bộ chức năng làm đối tượng kiểm thử.
- Cân hiểu trước về phần lớn các hoạt động của chức năng.
- Không phán đoán đối tượng hoặc phi đối tượng của kiểm thử.

(3) Đưa ra quan điểm kiểm thử

- Quan điểm test là "cánh cửa của Test".
- Ví dụ: Xác nhận sự chính xác của kết quả tính toán được hiển thị trên màn hình, xác nhận chức năng check mục nhập, xác nhận thời gian xử lý để biết quan điểm kiểm thử.

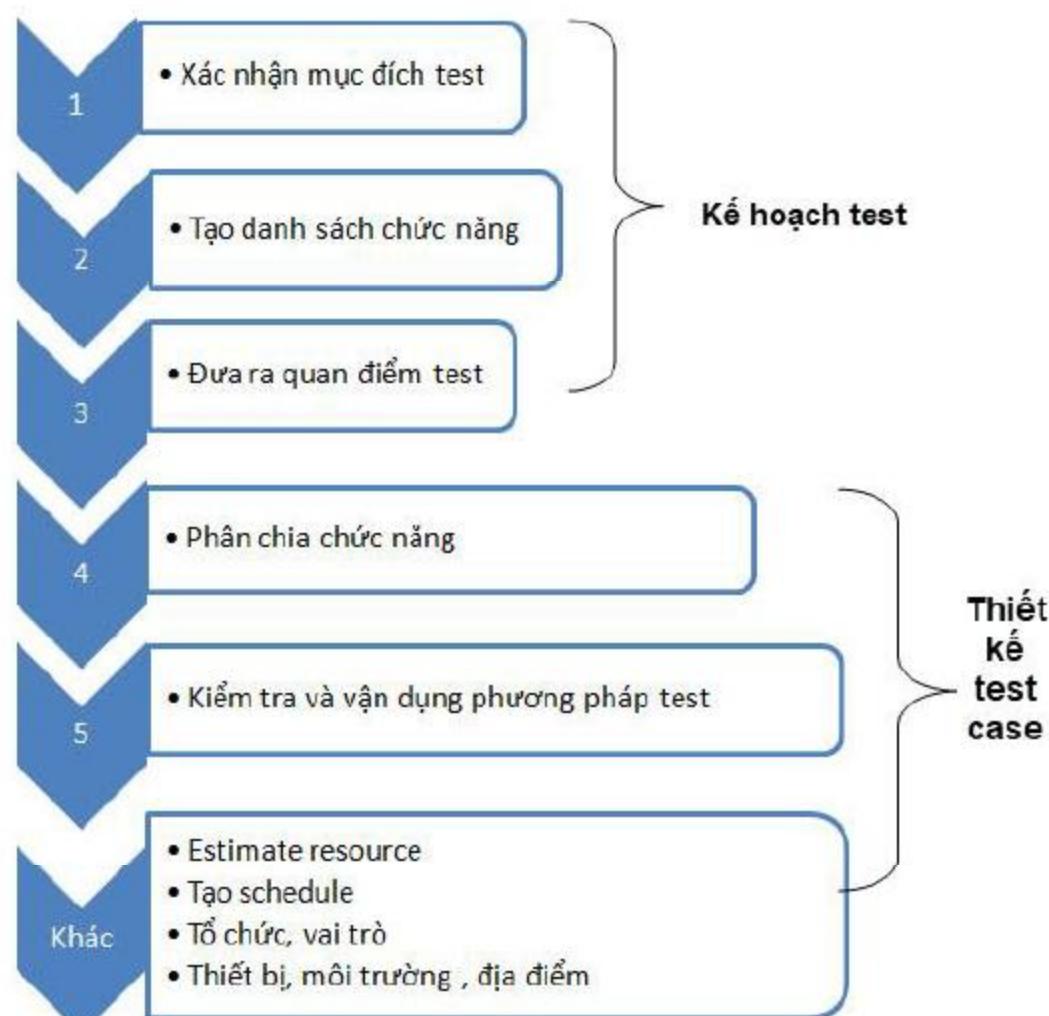
- Quan điểm kiểm thử đã đáp ứng được đúng mục đích kiểm thử.

(4) Phân chia chức năng cho từng quan điểm kiểm thử

- Áp dụng quan điểm kiểm thử cho từng chức năng để tránh bị quên.
- Có thể hình dung việc kiểm thử một cách cụ thể.
- Có thể nắm bắt quy mô kiểm thử và mức độ quan trọng của các quan điểm kiểm thử.

(5) Kiểm tra vận dụng phương pháp và kỹ thuật kiểm thử

- Tiến hành thiết kế kiểm đối với lần lượt từng kết hợp.
- Lựa chọn và quyết định phương pháp kiểm thử có thể phát hiện ra lỗi một cách hiệu quả nhất từ một trong số các phương pháp kiểm thử.



Hình 1.14: Các công đoạn kiểm thử

TÓM TẮT

Bài học này giới thiệu những kiến thức sau đây:

- *Mô hình phát triển phần mềm*
- *Yêu cầu kiểm thử và đảm bảo chất lượng phần mềm*
- *Vai trò của kiểm thử và đảm bảo chất lượng phần mềm*
- *Yêu cầu phần mềm*
- *Qui trình kiểm thử và đảm bảo chất lượng*
- *Thuộc tính yêu cầu phần mềm*
- *Cấu trúc bảng kế hoạch kiểm thử và đảm bảo chất lượng phần mềm*

BÀI 2: KỸ THUẬT THIẾT KẾ TEST CASE

Nội dung gồm các phần sau:

- Thiết kế test case theo White-box
- Thiết kế test case theo Black-Box
- Các hệ thống quản lý testcase

2.1 CÁC KHÁI NIỆM CHÍNH

Thiết kế test-case trong kiểm thử phần mềm là quá trình xây dựng các phương pháp kiểm thử có thể phát hiện lỗi, sai sót, khuyết điểm của phần mềm để xây dựng phần mềm đạt tiêu chuẩn. Thiết kế test-case giữ vai trò quan trọng trong việc nâng cao chất lượng phần mềm vì lý do sau đây:

- Tạo ra các ca kiểm thử tốt nhất có khả năng phát hiện ra lỗi, sai sót của phần mềm một cách nhiều nhất.
- Tạo ra các ca kiểm thử có chi phí rẻ nhất, đồng thời tốn ít thời gian và công sức nhất. Một trong những lý do quan trọng nhất trong kiểm thử chương trình là thiết kế và tạo ra các ca kiểm thử hiệu quả. Với những ràng buộc về thời gian và chi phí đã cho, thì vấn đề then chốt của kiểm thử trở thành: Tập con nào của tất cả ca kiểm thử có thể có khả năng tìm ra nhiều lỗi nhất.

Thông thường, phương pháp kém hiệu quả nhất là kiểm tra tất cả đều vào ngẫu nhiên, quá trình kiểm thử một chương trình bằng việc chọn ngẫu nhiên một tập con các giá trị đầu vào có thể. Về mặt khả năng tìm ra nhiều lỗi nhất, tập hợp các ca kiểm thử được chọn ngẫu nhiên có rất ít cơ hội là tập hợp tối ưu hay gần tối ưu. Sau đây là một số phương pháp để chọn ra một tập dữ liệu kiểm thử một cách thông minh.

Để kiểm thử hộp đen và kiểm thử hộp trắng một cách thấu đáo là không thể. Do đó, một chiến lược kiểm thử hợp lý là chiến lược có thể kết hợp sức mạnh của cả hai phương pháp trên: Phát triển kiểm thử nghiêm ngặt vừa bằng việc sử dụng các phương pháp thiết kế ca kiểm thử hướng hộp đen nào đó và sau đó bổ sung thêm kiểm thử này bằng khảo sát tính logic chương trình, với phương pháp hộp trắng.

Các thành phần dữ liệu Test case

Mỗi test case là một tài liệu tập hợp của các dữ liệu đầu vào và các điều kiện hoạt động được yêu cầu để thực hiện một mục test cùng với kết quả được mong đợi. Người kiểm thử mong đợi sẽ chạy một chương trình cho mỗi mục test theo tài liệu được cung cấp trong trường hợp test, và so sánh các kết quả thực tế với kết quả mong đợi được ghi lại trong các tài liệu. Nếu kết quả thu được hoàn toàn phù hợp với kết quả mong đợi, ko có lỗi hoặc ít nhất đã được xác định. Khi một số hoặc tất cả các kết quả đều không phù hợp với các kết quả mong đợi, một lỗi tiềm tàng được xác định. Phương pháp phân vùng các lớp tương đương được áp dụng để giành được hiệu quả và xác định tính hiệu quả test case, như thiết lập, được sử dụng trong hộp đen.

Hai tài liệu thiết yếu cần thiết cho việc thiết kế các testcase:

1. Đặc tả chức năng module: nêu rõ các thông số đầu vào, đầu ra và các chức năng cụ thể chi tiết của module.
2. Mã nguồn của module. Tính chất các testcase là dựa chủ yếu vào kỹ thuật kiểm thử hộp trắng

Thủ tục thiết kế testcase

Phân tích luận lý của module dựa vào một trong các kỹ thuật kiểm thử hộp trắng. Áp dụng các kỹ thuật kiểm thử hộp đen vào đặc tả của module để bổ sung thêm các testcase khác. Để thực hiện qui trình kiểm thử các module, hãy để ý 2 điểm chính:

1. Làm sao thiết kế được tập các testcase hiệu quả.
2. Cách thức và thứ tự tích hợp các module lại để tạo ra phần mềm chức năng:
 - Viết testcase cho module nào
 - Dùng loại tiện ích nào cho kiểm thử.
 - Coding và kiểm thử các module theo thứ tự nào.

- Chi phí tạo ra các testcase.
- Chi phí debug để tìm và sửa lỗi.

Có 2 phương án để kiểm thử các module:

1. Kiểm thử không tăng tiến hay kiểm thử độc lập (Nonincremental testing): kiểm thử các module chức năng độc lập nhau, sau đó kết hợp chúng lại để tạo ra chương trình.
2. Kiểm thử tăng tiến (Incremental testing): kết hợp module cần kiểm thử vào bộ phận đã kiểm thử (lúc đầu là null) để kiểm thử module cần kiểm thử trong ngữ cảnh.

2.2 THIẾT KẾ TEST-CASE: WHITE-BOX

2.2.1 Tổng quan về kiểm thử hộp trắng

Kiểm thử Hộp Trắng (còn gọi là Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing hoặc Structural Testing) là phương pháp kiểm thử phần mềm mà tester biết về cấu trúc nội bộ/thiết kế. Người kiểm tra chọn đầu vào để thực hiện các đường dẫn thông qua mã và xác định đầu ra thích hợp. Kiến thức lập trình và kiến thức thực hiện là rất cần thiết trong kiểm thử hộp trắng.

Kiểm thử hộp trắng dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu của đơn vị phần mềm cần kiểm thử để xác định đơn vị đó có thực hiện đúng. Do đó người kiểm thử hộp trắng có kỹ năng, kiến thức về ngôn ngữ lập trình, về thuật giải trong phần mềm để hiểu chi tiết đoạn code cần kiểm thử.

2.2.2 Ưu điểm/nhược điểm kiểm thử hộp trắng

- Ưu điểm
 - Test có thể bắt đầu ở giai đoạn sớm hơn, không đợi cho GUI để test
 - Test kỹ càng hơn, có thể bao phủ hầu hết các đường dẫn
 - Thích hợp trong việc tìm kiếm lỗi và các vấn đề trong mã lệnh
 - Cho phép tìm kiếm các lỗi ẩn bên trong

- Các lập trình viên có thể tự kiểm tra
 - Giúp tối ưu việc mã hóa
 - Do yêu cầu kiến thức cấu trúc bên trong của phần mềm, nên việc kiểm soát lỗi tối đa nhất.
- Nhược điểm
- Vì các bài kiểm tra rất phức tạp, đòi hỏi phải có các nguồn lực có tay nghề cao, với kiến thức sâu rộng về lập trình và thực hiện.
 - Maintenance test script có thể là một gánh nặng nếu thể hiện thay đổi quá thường xuyên.
 - Vì phương pháp thử nghiệm này liên quan chặt chẽ với ứng dụng đang được test, nên các công cụ để phục vụ cho mọi loại triển khai / nền tảng có thể không sẵn có.

2.2.3 Các mức độ áp dụng

Phương pháp Kiểm tra Hộp trắng áp dụng cho các mức độ kiểm tra sau:

- Unit Testing (Kiểm thử đơn vị): Để kiểm tra đường dẫn trong một đơn vị.
- Integration Testing (Test tích hợp): Để kiểm tra đường dẫn giữa các đơn vị.
- System Testing (Test hệ thống): Để kiểm tra các đường dẫn giữa các hệ thống con.

Tuy nhiên, nó là chủ yếu áp dụng cho các kiểm thử đơn vị.

2.3 THIẾT KẾ TEST-CASE: BLACK-BOX

2.3.1 Tổng quan kiểm thử hộp đen

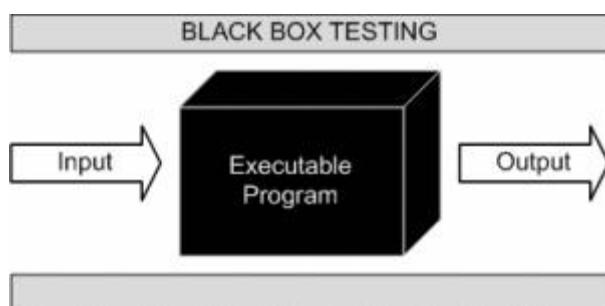
Phương pháp kiểm thử phần mềm được thực hiện khi không biết được cấu tạo bên trong của phần mềm, tester kiểm tra xem hệ thống như chiếc hộp đen.

- Còn gọi là kiểm thử hướng dữ liệu hay là kiểm thử hướng in/out.
- Người kiểm thử nên xây dựng các nhóm giá trị đầu vào mà sẽ thực thi đầy đủ tất cả các yêu cầu chức năng của chương trình.

- Các tester không dùng bất kỳ một kiến thức về cấu trúc lập trình bên trong hệ thống, xem hệ thống là một cấu trúc hoàn chỉnh, không thể can thiệp được.

Black Box Testing chủ yếu thực hiện trong Function test và System test. Phương pháp này cố gắng tìm ra các lỗi trong các loại sau:

- Chức năng không chính xác hoặc thiếu.
- Lỗi giao diện.
- Lỗi trong cấu trúc dữ liệu hoặc truy cập cơ sở dữ liệu bên ngoài.
- Hành vi hoặc hiệu suất lỗi.
- Khởi tạo và chấm dứt các lỗi.



Hình 2.1: Kiểm thử hộp đen

Mọi kỹ thuật nào cũng có ưu điểm và nhược điểm của nó. Các hệ thống thường phải được sử dụng nhiều phương pháp kiểm thử khác nhau để đảm bảo được chất lượng của hệ thống khi đến tay người dùng.

2.3.2 Ưu điểm/Nhược điểm của kiểm thử hộp đen

Ưu điểm

- Các tester được thực hiện từ quan điểm của người dùng và sẽ giúp đỡ trong việc sáng tỏ sự chênh lệch về thông số kỹ thuật.
- Các tester theo phương pháp black box không có "mối ràng buộc" nào với code, và nhận thức của một tester rất đơn giản: một source code có nhiều lỗi. Sử dụng nguyên tắc, "Hỏi và bạn sẽ nhận" các tester black box tìm được nhiều bug ở nơi mà các lập trình viên không tìm thấy.

- Tester có thể không phải IT chuyên nghiệp, không cần phải biết ngôn ngữ lập trình hoặc làm thế nào các phần mềm đã được thực hiện.
- Các tester có thể được thực hiện bởi một cơ quan độc lập từ các developer, cho phép một cái nhìn khách quan và tránh sự phát triển thiên vị.
- Hệ thống thật sự với toàn bộ yêu cầu của nó được kiểm thử chính xác.
- Thiết kế kịch bản kiểm thử khá nhanh, ngay khi mà các yêu cầu chức năng được xác định.

Nhược điểm của kiểm thử hộp đen

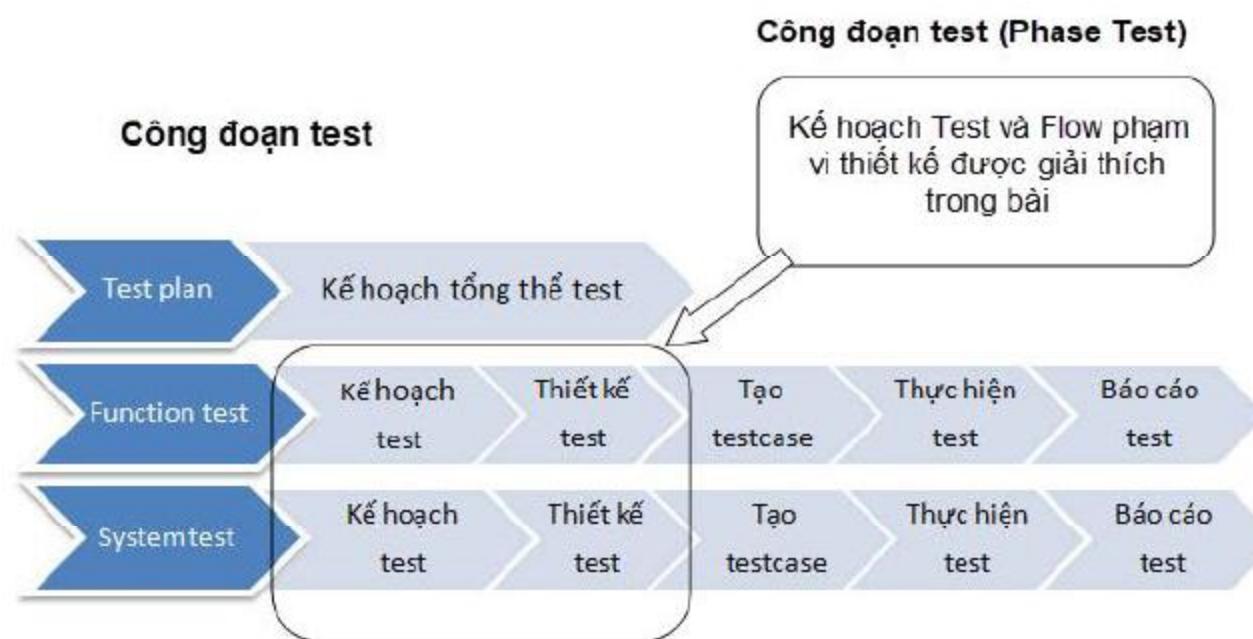
- Dữ liệu đầu vào yêu cầu một khối lượng mẫu (sample) khá lớn
- Nhiều dự án không có thông số rõ ràng thì việc thiết kế test case rất khó và do đó khó viết kịch bản kiểm thử do cần xác định tất cả các yếu tố đầu vào, và thiếu cả thời gian cho việc tập hợp này.
- Chỉ có một số nhỏ các đầu vào có thể được kiểm tra và nhiều đường dẫn chương trình sẽ được để lại chưa được kiểm tra.
- Kiểm thử black box được xem như "là bước đi trong mê cung tối đen mà không mang đèn pin" bởi vì tester không biết phần mềm đang test đã được xây dựng như thế nào. Có nhiều trường hợp khi một tester viết rất nhiều trường hợp test để kiểm tra một số thứ có thể chỉ được test bằng một trường hợp test và/hoặc một vài phần cuối cùng không được test hết.

2.3.3 Qui trình kiểm thử hộp đen

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

- Phân tích đặc tả về các yêu cầu chức năng phần mềm cần thực hiện.
- Dùng kỹ thuật định nghĩa các testcase để định nghĩa testcase. Định nghĩa mỗi testcase là xác định 3 thông tin sau :
- Giá trị dữ liệu nhập để phần mềm xử lý (hoặc hợp lệ hoặc không hợp lệ).
- Trạng thái của phần mềm cần có để thực hiện testcase.
- Giá trị dữ liệu xuất mà phần mềm phải tạo được.

- Kiểm thử các testcase đã định nghĩa.
- So sánh kết quả thu được với kết quả kỳ vọng trong từng testcase, từ đó lập báo cáo về kết quả kiểm thử.



Hình 2.2: Phase test trong công đoạn test

Kế hoạch test: Chỉ ra rõ ràng mục đích và phạm vi của công đoạn test để kiểm tra xem là test bằng approach như thế nào. Điều chỉnh resource thành viên và quyết định cả schedule.

Thiết kế test: Quyết định xem là sẽ sử dụng cái gì cho mục đích và loại test cần được thực hiện trong công đoạn test đó, chức năng đối tượng test, phương pháp test, import và export test. Ngoài ra cũng quyết định cụ thể hơn nguyên liệu cần thiết để thực hiện test hay tiêu chuẩn quyết định thành công/ không thành công.

Tạo test case: Tạo document ghi trạng thái trước khi bắt đầu test và **kết quả mong đợi** (kết quả chạy đối tượng test theo điều kiện và trình tự thao tác khi thực hiện test sẽ như thế nào) và **cột trạng thái** (cột ghi lại kết quả thao tác của đối tượng test).

Thực hiện test: Vừa xem testcase vừa cho chạy phần mềm thực tế để tiến hành test, sau đó đánh dấu kết quả bằng dấu pass hoặc fail vào cột trạng thái testcase. Trường hợp có testcase khác với kết quả mong đợi thì ghi dấu fail vào cột trạng thái, rồi tạo bản báo cáo lỗi. Trong bản báo cáo lỗi: trình bày nội dung mô tả hiện tượng

khác với kết quả mong đợi và hiện tượng đó phát sinh trong trường hợp như thế nào (thao tác, giả nhập, điều kiện,...)

Báo cáo test: Tóm tắt kết quả để báo cáo. Căn cứ vào **các loại dữ liệu** (mục thực hiện, hiệu quả của việc test, công số thực hiện,...) và **dữ liệu lỗi** (số lỗi được tìm ra, số lỗi theo mức độ quan trọng,...) để đánh giá xem có thỏa mãn tiêu chuẩn pass/ fail của test không. Ngoài ra cũng đề xuất thêm risk có thể sinh ra sau khi release và mục cần bổ sung trong dự án cho giai đoạn tiếp theo.

2.4 CÁC HỆ THỐNG QUẢN LÝ TEST-CASE

2.4.1 Giới thiệu

Đầu tiên phải kể đến "Excel - Google sheet". Đây là tool được sử dụng từ rất lâu và ở nhiều công ty, nhiều dự án.

Với Google sheet tốt cho những project vừa và nhỏ, nhóm có 1,2 tester/QA. Với những project lớn, có nhiều tester/QA và phải tạo test run nhiều thì việc quản lý bằng Google sheet khá khó khăn và không rõ ràng.

Trên thị trường hiện tại có rất nhiều công cụ. Nói chung team lớn, project lớn thì chuyển bỏ ra một khoản tiền để cho test case management tool.

Có hai loại tool chính: một loại công cụ để quản lý test case, một loại công cụ nằm trong phần mềm quản lý dự án.

Thứ nhất, công cụ quản lý test case riêng biệt như dùng Git để quản lý code, dùng Jira để quản lý task....):

- VỚI CÔNG CỤ MIỄN PHÍ CÓ THỂ THỬ NHƯ TestLink...tự thiết lập trên server công ty
- VỚI CÔNG CỤ TÍNH PHÍ THÌ CÓ NHIỀU SỰ LỰA CHỌN NHƯ Testrail, Testlodge: hai công cụ này khá tiện cho việc sử dụng, đáp ứng nhu cầu quản lý test case.

Thứ hai, công cụ nằm trong một bộ phần mềm quản lý project. Chúng ta có thể tham khảo Visual Studio Team Service hay Terik. Hai công cụ này khá nổi trên thị trường. Mục đích của bộ tool này là "quy tất cả về một mồi", từ source code, test case, document, process, automation test, CI/CD,.. dùng khá tiện dụng, quản lý test case cũng rất tốt.

2.4.2 Testcase

Sau khi có được tất cả requirement, test plan thì một tester/QA đã biết nhiệm vụ của mình. Và việc tiếp theo phải làm là xây dựng bộ test case cho phần mềm.

Test case là thể hiện được cách hoạt động của chức năng mà có mô tả, điều kiện chi tiết rõ ràng.

Chính vì vậy, một test case tốt là một test case làm sao để bất cứ ai đọc vào họ cũng có thể biết làm sao thực hiện/mô phỏng lại trên ứng dụng và biết chức năng đó đang hoạt động đúng hay sai.

Bộ test case là nơi lưu trữ lại tất cả thông tin ứng dụng, được coi như một tài liệu hiệu quả cho phần mềm.

Bộ test case như một checklist, mỗi khi test chỉ cần dựa trên checklist, không bao giờ bỏ sót trường hợp.

Khi không còn tham gia vào project và giao một tester khác, test case là một tài liệu cực kì quan trọng. Người mới sẽ dựa vào test case để tiếp tục công việc. Vậy nên việc xây dựng một test case hiệu quả là cực kì quan trọng, khi viết ta phải chắc rằng không giả định bất cứ điều gì, test case phải chi tiết, chính xác và được cập nhật liên tục.

Việc có bộ test case sẽ giúp tạo test run dễ dàng và có thể tracking lại quá trình xây dựng phần mềm, tracking bug hiệu quả hơn. Vì khi có một checklist cho tất cả các case mình sẽ test, ta sẽ dễ dàng lọc ra những bug đang có trong phần mềm.

Từ bộ test case ta có thể biết độ bao phủ của test case (test coverage) đã tốt hay chưa. Nếu ta đã thực hiện hết bộ test case mà phần mềm vẫn còn bug có nghĩa là ta biết mình viết test case chưa tốt. Vì vậy *mỗi khi phát sinh bug mà flow đó chưa có trong test case thì hãy bổ sung ngay*. Không ai có thể xây dựng một bộ test case hoàn chỉnh ngay từ đâu, test case sẽ được bổ sung trong suốt quá trình phát triển phần mềm. Nhưng một người có kinh nghiệm thì test coverage thường sẽ cao vì họ biết được phần mềm thường gặp lỗi ở đâu.

Một lợi ích khác cho việc automation, một test case viết tốt thì khi chuyển qua automation, build script sẽ nhanh và tốn ít thời gian hơn. Người viết script chỉ cần follow theo test case, có step đầy đủ và viết, không cần phải suy nghĩ nhiều.

2.4.3 Các thành phần testcase

Test case có thể xem là tương đối "đầy đủ" và đáp ứng mọi mong muốn từ nhóm, giúp test case "rõ ràng" hết mức có thể.

- **Test case ID:** chỉ đơn giản là ID để mình dễ gọi tên. Ví dụ như test case của chức năng "***Sign Up***" thì sẽ là ***SU_001***.
- **Test case summary:** phần tóm tắt, có thể gọi là tên của test case, giúp cho người đọc nhận diện được mục đích của test case. Một cái tên tốt là cái tên mà chỉ cần đọc vào là biết được mình phải làm gì.

Ví dụ: *"Verify that user can sign up with valid data successfully"*

- **Test case description:** mô tả chi tiết test case dùng để làm gì. Vì phần tên thường không thể mô tả đầy đủ được test case, nên chúng ta sẽ mô tả rõ ở đây. Nhưng theo kinh nghiệm thì phần này thường hao hao giống phần "Test case summary", nên nếu thời gian hạn hẹp có thể lược bỏ qua.
- **Pre-conditions:** Không thể cứ mở phần mềm ra là test được liền, mà cần test data, test data phải thoả một số điều kiện. Bên cạnh đó là môi trường test, ta cần phải ở màn hình nào để chuẩn bị để test.

Ví như bạn kiểm tra case sau: "Verify sign in function when there's no internet connection" Thì Pre-condition là: "1. There's no internet connection on device. 2. User is in sign in screen"

- **Test data:** Có những test case sẽ cần test data, hoặc không cần. Với những hệ thống phức tạp thì thường sẽ có, ta phải dùng một tài khoản nhất định thì ta mới có function đó để kiểm thử hoặc với một số trường hợp nhất định như dữ liệu hợp lệ (valid data) chẳng hạn, thì nên đưa test data để dễ dàng test hơn.
- **Reproduce Steps:** mô tả lại từng bước để thực hiện test case. Phần này là phần cực kì quan trọng và chiếm nhiều thời gian khi viết test case.

Một test case tốt khi phần này được viết chi tiết và rõ ràng. Nhớ rằng "Đừng assume" (giả định) bất cứ điều gì khi viết, hãy nghĩ rằng ta là một người không biết gì về phần mềm và đây là lần đầu tiên thao tác, thì test case mới rõ được.

- **Expected result:** tương ứng với một "reproduce steps" ta viết thì sẽ có một expected result. Không nhất thiết là bước nào cũng phải có, nhưng nên là có.

Ví dụ: trong test case "Verify that user can sign up successfully". Ta có steps "Press "Sign In" button" thì tương ứng phải có expected result "Sign in successfully. User go to home screen"

- **Observed/Actual result:** kết quả thực tế. Thường thì phần này ta sẽ thêm vào khi run test case. Nó giúp ta biết được thực tế phần mềm đang có gì, bị lỗi gì hay nó làm "work like expected"
- **Test case status:** hiện trạng của phần mềm, nó là kết quả sau khi "run" một test case. Thường có 4 status: Pass/Fail/Blocked/Skipped
- **Bug ID:** nếu test case bị failed thì ta sẽ tạo một ticket bug và thêm bug ID vào cột này. (Thường team sẽ có một phần mềm để quản lý task và bug)
- **Environment:** môi trường test. Quá trình phát triển phần mềm thường sẽ có nhiều môi trường như Dev, Staging, UAT, Production. Khi ta run test case, thì nhớ ghi rõ là test trên môi trường nào.
- **Notes/Comment:** lưu ý hay phản hồi ta muốn ghi lại cho người khác dễ dàng thực hiện hơn khi đọc

Đó là tất cả những phần mà test case nên có. Có những phần chỉ xuất hiện khi "run" thôi. Nhưng khi thiết kế thì ta nên để sẵn những cột để không bị bỏ qua bất cứ thông tin.

TÓM TẮT

Bài học này giới thiệu những kiến thức về.

- *Giới thiệu thiết kế Test case*
- *Cấu trúc thành phần Test case*

BÀI 3: THIẾT KẾ TEST CASE - BLACK-BOX

Nội dung gồm các phần sau:

- *Kỹ thuật Lớp tương đương*
- *Kỹ thuật Giá trị biên*
- *Kỹ thuật Bảng quyết định*
- *Kỹ thuật Dịch chuyển trạng thái*

3.1 KỸ THUẬT LỚP TƯƠNG ĐƯƠNG

Tinh thần của kỹ thuật này là cố gắng phân các testcase ra thành nhiều nhóm (họ) khác nhau: các testcase trong mỗi họ sẽ kích hoạt phần mềm thực hiện cùng một hành vi. Mỗi nhóm testcase thỏa mãn tiêu chuẩn trên được gọi là lớp tương đương, ta chỉ căn xác định testcase đại diện cho nhóm và dùng testcase này để kiểm thử phần mềm. Như vậy ta đã giảm rất nhiều testcase cần định nghĩa và kiểm thử, nhưng chất lượng kiểm thử không bị giảm sút bao nhiêu so với vét cạn. Điều này là dựa vào kỵ vọng rất hợp lý sau:

- Nếu một testcase trong lớp tương đương nào gây lỗi phần mềm thì các testcase trong lớp này cũng sẽ gây lỗi như vậy.
- Nếu một testcase trong lớp tương đương nào không gây lỗi phần mềm thì các testcase trong lớp này cũng sẽ không gây lỗi. Vấn đề tiếp là căn định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả. Điều này phụ thuộc vào tinh thần kiểm thử:
- Nếu ta dùng tinh thần kiểm thử theo hợp đồng (Testing-by-Contract) thì không cần định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả vì không cần thiết.

- Còn nếu ta dùng tinh thần kiểm thử phòng vệ (Defensive Testing), nghĩa là kiểm thử hoàn hảo, thì phải định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả để xem phần mềm phản ứng như thế nào với những testcase này.

Ví dụ: Phần mềm “Quản lý hồ sơ nhân lực” với đặc tả chức năng: mỗi lần nhận 1 hồ sơ xin việc, phần mềm sẽ ra quyết định ban đầu dựa vào tuổi của ứng viên theo bảng:

Tuổi ứng viên Kết quả sơ bộ

- | | |
|---------|-------------------------|
| - 0-15 | Không thuê |
| - 16-17 | Thuê dạng bán thời gian |
| - 18-54 | Thuê toàn thời gian |
| - 55-99 | Không thuê |

Bằng phương pháp phân hoạch tương đương và phân tích giá trị biên, hãy thiết các trường hợp kiểm thử cho phần mềm trên.

Tuổi	<0	0-15	16-17	18-54	55-99	>99
Kết quả	Nhập dữ liệu sai	Không Thuê	Thuê bán thời gian	Thuê toàn thời gian	Không Thuê	Nhập dữ liệu sai
Lớp tương đương	Không Hợp lệ	Hợp lệ	Hợp lệ	Hợp lệ	Hợp lệ	Không Hợp lệ
Đánh dấu	I1	V1	V2	V3	V4	I2

Các lớp tương đương:

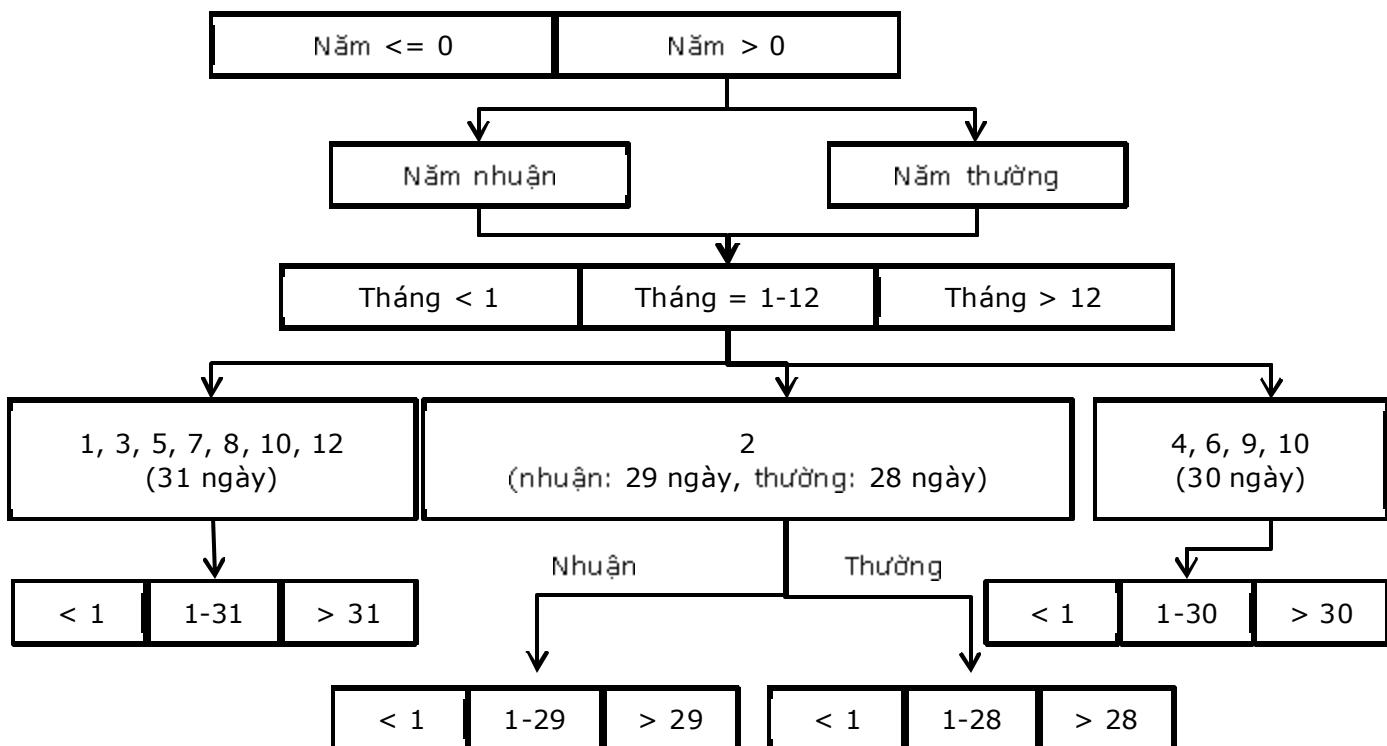
- Lớp I1: {-2,-1,0} Lớp I2: {99,100,101}
- Lớp V1: {-1,0,1} {14,15,16}
- Lớp V2: { 15,16,17} {16,17,18}
- Lớp V3: {17,18,19}, {53,54,55}
- Lớp V4: {54,55,56}, {98,99,100}

Ví dụ: Chương trình kiểm tra một ngày tháng nhập vào có hợp lệ hay không? Cần nhập vào năm, tháng, ngày của ngày muốn kiểm tra.

Ví dụ 12/12/2017 là ngày hợp lệ, 32/12/2017 là ngày không hợp lệ.

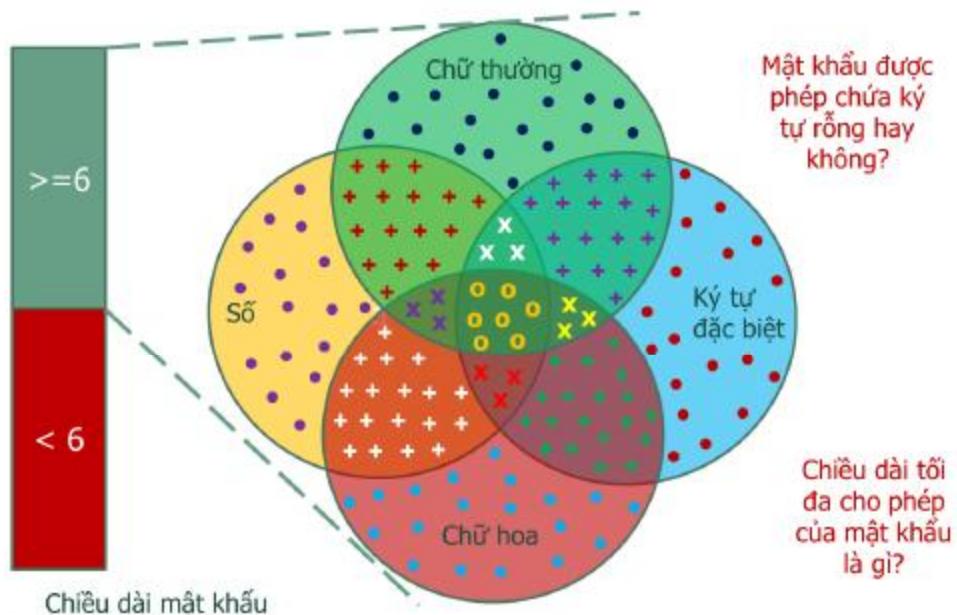
Sử dụng phương pháp phân vùng tương đương thiết kế các test case để kiểm thử chương trình trên.

Hướng dẫn:



Ví dụ: Chức năng đăng ký tài khoản của một ứng dụng học tập yêu cầu mật khẩu phải có chứa chữ hoa, chữ thường, số, ký tự đặc biệt và chiều dài tối thiểu là 6.

Sử dụng kỹ thuật phân vùng tương đương viết các test case kiểm tra ô nhập liệu mật khẩu.



3.2 KỸ THUẬT GIÁ TRỊ BIÊN

Phân tích giá trị biên (Boundary Value Analysis - BVA): kỹ thuật kiểm thử phần mềm có liên quan đến việc xác định biên (ranh giới) của điều kiện mô tả cho các giá trị đầu vào và chọn giá trị ở biên và bên cạnh giá trị biên làm dữ liệu kiểm thử. Phương pháp phân tích giá trị biên sẽ đưa ra các giá trị đặc biệt, bao gồm loại dữ liệu, giá trị lỗi, bên trong, bên ngoài biên giá trị, lớn nhất và nhỏ nhất.

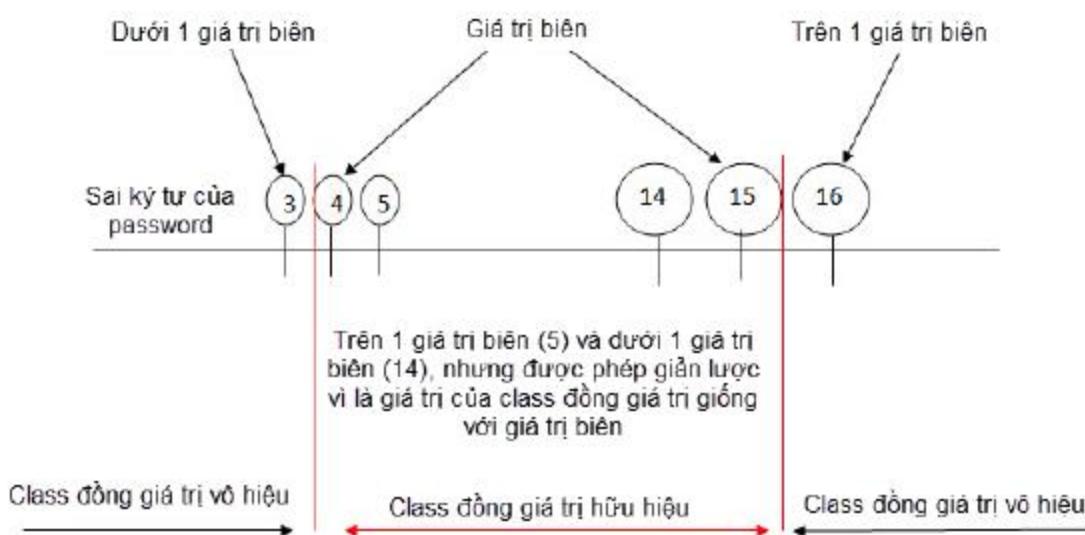
Kỹ sư nhiều kinh nghiệm chắc chắn từng gặp phải các lỗi của hệ thống ngay tại giá trị biên. Đó là lý do tại sao phân tích giá trị biên lại quan trọng khi kiểm thử hệ thống.

- Test giá trị biên được thực hiện theo trình tự dưới đây:

1. Tìm ra đường biên
2. Quyết định giá trị biên
3. Quyết định giá trị để test

- Giá trị biên.
- Dưới 1 giá trị biên. (Nếu là class đồng giá trị)
- Trên 1 giá trị biên. (Nếu là class đồng giá trị)

Ví dụ: Yêu cầu nhập $4 \leq \text{password} \leq 15$. Giá trị được test sẽ như sau



Kinh nghiệm trong cuộc sống đời thường cũng như trong lập trình các giải thuật lặp cho chúng ta biết rằng lỗi thường nằm ở biên (đầu hay cuối) của khoảng liên tục nào đó (lớp tương đương). Do đó ta sẽ tập trung tạo testcase ứng với những giá trị ở biên.

Ý tưởng của kỹ thuật kiểm thử dựa trên các trị biên là chỉ định nghĩa các testcase ứng với các giá trị ngay trên biên hay lân cận biên của từng lớp tương đương. Do đó kỹ thuật này chỉ thích hợp với các lớp tương đương xác định bởi các giá trị liên tục (số nguyên, số thực), chứ không thích hợp với lớp tương đương được xác định bởi các giá trị liệt kê mà không có mối quan hệ lẫn nhau.

Qui trình cụ thể để thực hiện kiểm thử dựa trên các giá trị biên:

- Nhận dạng lớp tương đương dựa trên đặc tả yêu cầu chức năng của phần mềm.
- Nhận dạng 2 biên của mỗi lớp tương đương.
- Tạo các testcase cho mỗi biên của mỗi lớp tương đương :
 - 1 testcase cho giá trị biên.
 - 1 testcase ngay dưới biên.
 - 1 testcase ngay trên biên.
- Ý nghĩa ngay trên và ngay dưới biên phụ thuộc vào đơn vị đo lường cụ thể:
- Nếu là số nguyên, ngay trên và ngay dưới biên 1 đơn vị.
- Nếu đơn vị tính là “\$ và cent” thì ngay dưới của biên 5\$ là 4.99\$, trên là 5.01\$.
- Nếu đơn vị là \$ thì ngay dưới của 5\$ là 4\$, ngay trên 5\$ là 6\$.

Ví dụ: Xếp loại cuối năm học sinh ở trường THPT dựa trên điểm trung bình (ĐTB) như sau:

- $0 \leq \text{ĐTB} < 5$: yếu, kém
- $5 \leq \text{ĐTB} < 7$: trung bình
- $7 \leq \text{ĐTB} < 8$: khá
- $8 \leq \text{ĐTB} < 9$: giỏi
- $9 \leq \text{ĐTB} \leq 10$: xuất sắc

❖ Biết điểm **trung bình làm tròn 1 chữ số thập phân**.

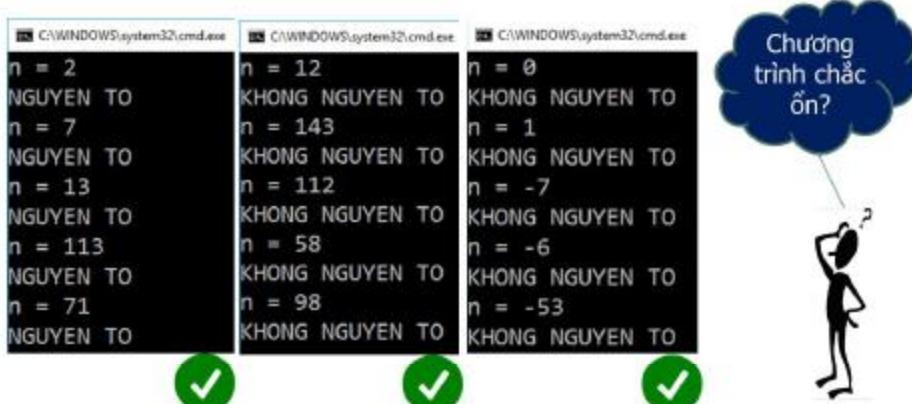


Để áp dụng phân tích giá trị biên vào kiểm thử chương trình lấy giá trị lớn nhất và nhỏ nhất trong các đoạn

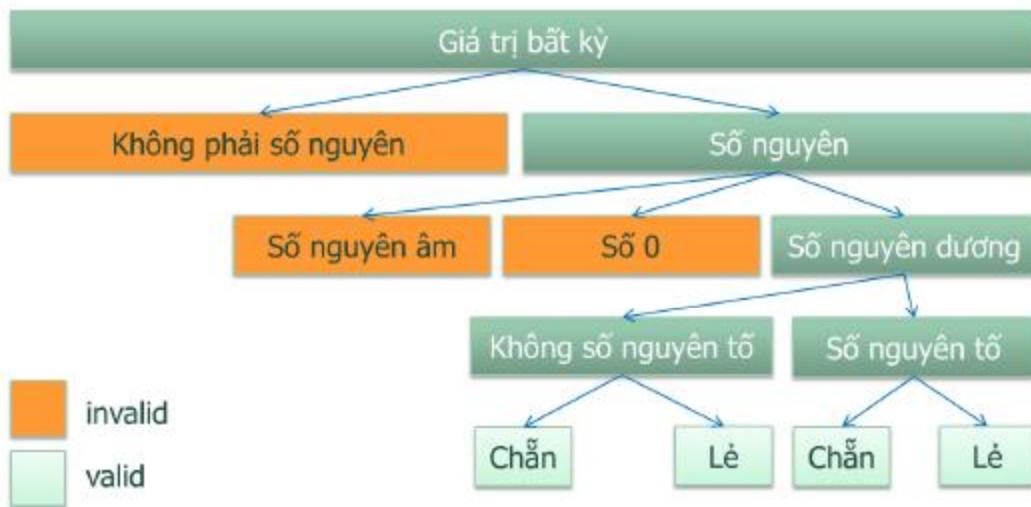
- Đoạn “yếu, kém”: 0 và 4.9
- Đoạn “trung bình”: 5 và 6.9
- Đoạn “khá”: 7 và 7.9
- Đoạn “giỏi”: 8 và 8.9
- Đoạn “xuất sắc”: 9 và 10
- Lấy giá trị đầu tiên trong đoạn > 10 là 10.1
- Lấy giá trị cuối cùng trong đoạn < 0 là -0.1

Ví dụ: Một developer đã phát triển xong chức năng **kiểm tra số nguyên dương n có phải là số nguyên tố không** và giao cho một tester tiến hành kiểm thử chức năng này với màn hình console cho phép nhập số nguyên dương n và xuất kết quả thông báo số đó có phải nguyên tố hay không?

- *Ghi chú: số nguyên tố là số tự nhiên chỉ có hai ước số dương phân biệt là 1 và chính nó.*
- Một **tester ít kinh nghiệm** kiểm thử bằng cách chạy một số giá trị nguyên n để kiểm tra chương trình trên.



- Một **tester có kinh nghiệm** tiến hành phân vùng tương đương để thiết kế test case kiểm thử như sau:



- Áp dụng phân tích giá trị biên cho phân vùng số nguyên dương, chọn các giá trị biên sau đại diện cho các phân vùng để kiểm thử:
 - Phân vùng số nguyên tố chẵn: **2**
 - Phân vùng số nguyên tố lẻ: **3**
 - Phân vùng số lẻ không phải số nguyên tố: **1**
 - Phân vùng số chẵn không là số nguyên tố: **4**
- Developer mở chương trình kiểm tra phát hiện thiếu dấu "="

```

C:\WINDOWS\system32
n = 2
NGUYEN TO
n = 3
NGUYEN TO
n = 1
KHONG NGUYEN TO
n = 4
NGUYEN TO

```



```

bool ktNguyenTo(int n) {
    if (n < 2)
        return false;

    for (int i = 2; i < sqrt(n); i++)
        if (n % i == 0)
            return false;

    return true;
}

```

Chương trình minh họa bằng C++

```

bool ktNguyenTo(int n) {
    if (n < 2)
        return false;

    for (int i = 2; i <= sqrt(n); i++)
        if (n % i == 0)
            return false;

    return true;
}

```

Chương trình minh họa bằng C++

Ưu điểm của phương pháp Phân tích giá trị biên

- Đơn giản.
- Hiệu quả cho các hàm có biến độc lập.
- Có thể tự động sinh test case khi xác định được giá trị biên của các biến.

Khuyết điểm của phương pháp Phân tích giá trị biên

- Không quan tâm đặc trưng của hàm, ngữ nghĩa các biến, cũng như quan hệ giữa các biến.
- Khó áp dụng cho trường hợp các biến có quan hệ ràng buộc nhau.

3.3 KỸ THUẬT BẢNG QUYẾT ĐỊNH

Bảng quyết định là kỹ thuật kiểm thử hộp đen dùng xác định **những kịch bản** (scenario) kiểm thử cho những trường hợp có logic nghiệp vụ phức tạp.

Bảng quyết định giúp tester xác định hiệu quả sự **kết hợp các đầu vào khác nhau** với các tình trạng phần mềm thực thi đúng quy tắc nghiệp vụ.

Bảng quyết định nên được sử dụng trong những chương trình có **nhiều lệnh rẽ nhánh** và **các biến đầu vào có mối quan hệ với nhau**.

Cấu trúc bảng quyết định:



Xây dựng bảng quyết định:

- Liệt kê tất cả **các điều kiện/đầu vào**.
- Tính **số lượng kết hợp** các giá trị của các điều kiện/ đầu vào và đặt các kết hợp đó vào trong phần giá trị các điều kiện.

- **Xác định các test case** tương ứng cho các điều kiện được thỏa mãn.
- **Các hành động** chính là kết quả mong đợi của test case.
- Trong giá trị các điều kiện có một giá trị đặc biệt là “-” thể hiện cho các điều kiện kết hợp chỉ định không thể xảy ra.
- Chú ý: thứ tự các điều kiện và thứ tự thực hiện hành động là không quan trọng.

Ví dụ: Xây dựng bảng quyết định cho chức năng login với hai thông tin đầu vào là username và password trên một ứng dụng web.

- Xác định các điều kiện/đầu vào: username và password.
- Mỗi đầu vào nhận một trong 3 giá trị: rỗng (blank - B), hợp lệ (valid - V) và không hợp lệ (invalid - I).
- Số kết hợp giá trị các điều kiện có thể xảy ra là 9

Các điều kiện	Username	B	B	B	I	I	I	V	V	V
	Password	B	I	V	B	I	V	B	I	V
Các hành động	Thông điệp lỗi	M1	M1	M1	M3	M3	M3	M2	M4	
	Chuyển đến trang	L	L	L	L	L	L	L	L	H

M1: Vui lòng nhập username

M2: Vui lòng nhập password

M3: Username không hợp lệ

M4: Password không hợp lệ

L: Trang login

H: Trang home

- Rút gọn các kết hợp các giá trị đầu vào.
 - Những trường hợp (cột quy tắc) có **cùng giá trị hành động**, nhưng chỉ **khác giá trị của một điều kiện** duy nhất.
 - Chuyển giá trị của điều kiện khác nhau đó thành “-” và gom các cột lại thành một.
 - Lặp lại hai bước trên cho đến khi không còn các test case nào như thế.

The diagram illustrates the reduction of a test case matrix. At the top is a large matrix with 'Các điều kiện' (Conditions) and 'Các hành động' (Actions) as rows, and various input values as columns. A red arrow points down to a smaller matrix, which is a simplified version of the first one, showing only the essential combinations of conditions and actions.

Các điều kiện	Username	B	B	B	I	I	I	V	V	V
	Password	B	I	V	B	I	V	B	I	V
Các hành động	Thông điệp lỗi	M1	M1	M1	M3	M3	M3	M2	M4	
	Chuyển đến trang	L	L	L	L	L	L	L	L	H
	Username	B	I	V	V	V				
	Password	-	-	B	I	V				
Các hành động	Thông điệp lỗi	M1	M3	M2	M4					
	Chuyển đến trang	L	L	L	L	H				

- Các quy tắc chuyển từ bảng quyết định thành các test case:
 - Nếu giá trị các điều kiện nhập là các giá trị **luận lý** (true/false) thì **mỗi cột** quy tắc được chuyển thành một test case.
 - Nếu giá trị các điều kiện nhập có **nhiều giá trị** thì **mỗi cột** quy tắc được chuyển thành nhiều test case sử dụng kỹ thuật phân vùng tương đương hoặc phân tích giá trị biên.

Ví dụ: Một thư viện ABC có chức năng cho phép độc giả mượn sách. Theo đó, độc giả mượn sách không được quá 500 quyển sách trong năm (không phân biệt tên đầu sách), nhưng không được phép mượn quá 5 quyển trong một lần mượn, và phải trả các cuốn sách đã mượn mới được phép mượn tiếp nữa.

Sử dụng bảng quyết định và phân tích giá trị biên thiết kế test case kiểm thử độc giả có được phép mượn sách không và được mượn tối đa bao nhiêu quyển trong lần mượn mới?

Hướng dẫn:

- K = không còn nợ sách đã mượn.
- M là số sách đã mượn trong năm đến thời điểm hiện tại ($0 \leq M \leq 500$). X là số sách định mượn ($1 \leq X \leq 5$). Độc giả sẽ được mượn số sách này nếu $0 \leq M + X \leq 500$ và K đúng.

- Bảng quyết định

		C ₁	C ₂	C ₃	C ₄
Điều kiện	K	✗	✓	✓	✓
	1 ≤ X ≤ 5	-	✗	✓	✓
	0 ≤ M + X ≤ 500	-	-	✗	✓
Hành động	Mượn sách?	✗	✗	✗	✓
	Số sách tối đa được mượn.	0	0	0	500–M < 5 ? 500–M:5

Cột	Đầu vào			Mong muốn	
	K	X	M	Mượn sách?	Số sách tối đa được mượn.
C ₁	✗	1	499	✗	0
C ₂	✓	6	494	✗	0
	✓	6	495	✗	0
C ₃	✓	1	500	✗	0
	✓	5	496	✗	0
C ₄	✓	1	499	✓	1
	✓	1	498	✓	2
	✓	5	495	✓	5
	✓	5	494	✓	5
	✓	2	498	✓	2
	✓	2	497	✓	3
	✓	4	496	✓	4
	✓	4	495	✓	5

3.4 KỸ THUẬT DỊCH CHUYỂN TRẠNG THÁI

- Các khía cạnh của hệ thống được mô tả thông qua **lược đồ trạng thái**.
- Hệ thống sẽ có nhiều trạng thái khác nhau, sự dịch chuyển từ một trạng thái này sang trạng thái khác được quyết định bởi một sự kiện nào đó.
- Một mô hình dịch chuyển trạng thái có 4 phần cơ bản:
 - Các trạng thái (**states**) phần mềm có thể xảy ra.
 - Sự dịch chuyển (**transitions**) từ trạng thái này sang trạng thái khác.
 - Các sự kiện (**events**) dẫn đến sự dịch chuyển trạng thái.

- Các hành động (**actions**) là kết quả của việc dịch chuyển trạng thái.

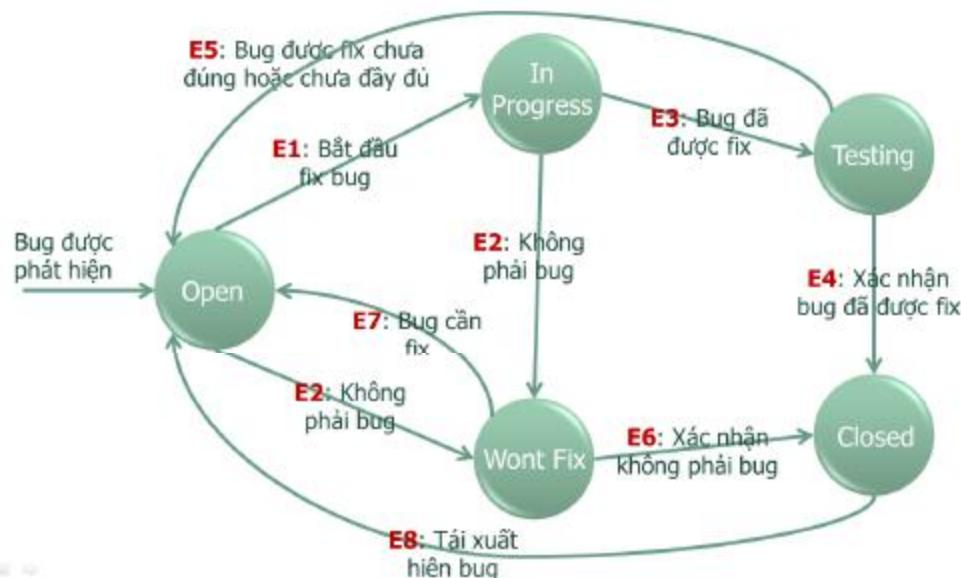
Ví dụ: Một quy trình sửa lỗi (fix bug) của một hệ thống bug tracking ở một công ty phần mềm như sau:

- Tester phát hiện bug và tạo báo cáo bug bắt đầu với trạng thái "Open".
- Developer xem xét nếu thấy nó không phải bug thì chuyển bug sang trạng thái "Wont Fix" và giải thích cho tester.
- Nếu tester cũng đồng ý đó không phải bug thì chuyển bug sang trạng thái "Closed", ngược lại chuyển về trạng thái "Open".
- Nếu developer xem qua thấy bug được tạo đúng là lỗi phần mềm và tiến hành fix bug thì chuyển bug sang trạng thái "In Progress".
- Sau khi fix bug xong, developer chuyển nó sang trạng thái "Testing" để tester tiến hành xác nhận (verify) thật sự bug đã được fix.
- Trong quá trình đang fix bug, developer nhận ra nó không phải lỗi phần mềm thì developer chuyển về trạng thái "Wont Fix".
- Nếu tester kiểm tra qua bug đã được fix và thấy ổn thì chuyển bug sang trạng thái "Closed", ngược lại chuyển nó về trạng thái "Open" và yêu cầu developer fix lại.
- Sau khi bug đã đóng, nhưng quá trình test sau đó lại thấy nó tái xuất hiện thì tester có thể chuyển nó về trạng thái "Open" và yêu cầu developer tiếp tục fix.

Hướng dẫn:

- Các trạng thái: O = Open, IP = In Progress, WF = Wont Fix, T = Testing, C = Closed.
- Các dịch chuyển:
 - Open → In Progress, Wont Fix
 - In Progress → Wont Fix, Testing
 - Wont Fix → Open, Close
 - Testing → Open, Closed
 - Closed → Open

- Các sự kiện:
 - E1: bắt đầu fix bug
 - E2: không phải bug
 - E3: bug đã được fix
 - E4: xác nhận bug đã được fix
 - E5: bug được fix chưa đúng hoặc chưa đầy đủ
 - E6: xác nhận không phải bug
 - E7: bug cần fix
 - E8: tái xuất hiện bug
- Các hành động kết quả: khi có bất kỳ sự thay đổi trạng thái nào của bug, hệ thống sẽ gửi email thông báo đến tất cả các thành viên có liên quan đến bug đó về trạng thái hiện tại của bug.

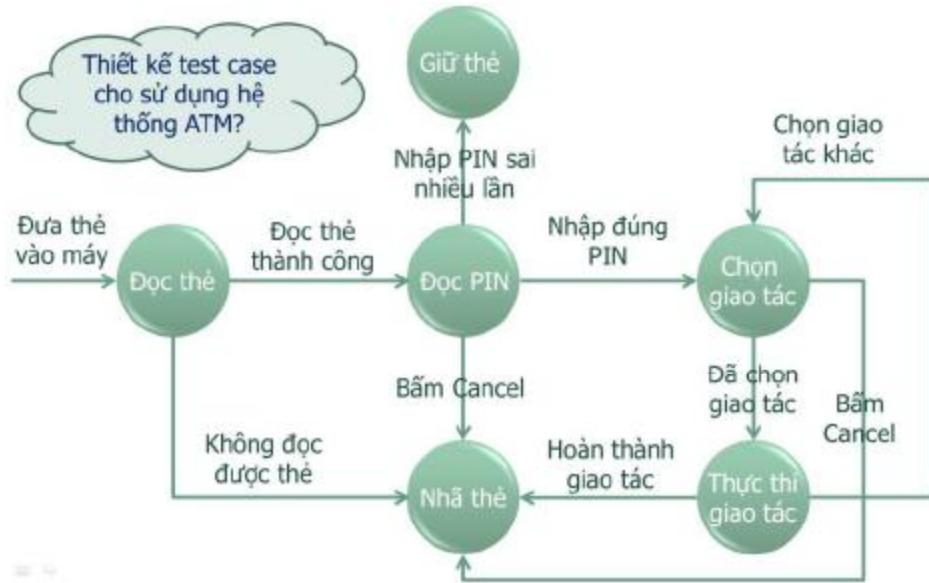


- Bảng mô tả trạng thái

	E1	E2	E3	E4	E5	E6	E7	E8
O	IP	WF						
IP		WF	T					
T				C	O			
C							O	
WF						C	O	

- Dựa vào bảng trạng thái có thể thiết kế 9 test case hợp lệ: O → IP, O → WF, IP → WF, IP → T, T → C, T → O; C → O, WF → C, WF → O và một vài trường hợp không hợp lệ.

Ví dụ: Sử dụng hệ thống ATM



3.5 BÀI TẬP THIẾT KẾ TEST CASE – BLACK BOX

Bài tập 1

Một hệ thống ngân hàng trực tuyến của ngân hàng ABC quy định không được chuyển khoản quá 10 triệu (tr) trong ngày, tối thiểu mỗi lần chuyển khoản là 1tr.

Sử dụng phương pháp phân tích giá trị biên thiết kế test case để kiểm tra số tiền chuyển khoản của khách hàng A có được phép chuyển trong ngày hiện tại không?

Bài tập 2

Nếu bạn đi xe điện chuyển trước 9:30 sáng hoặc từ sau 4:00 chiều đến 7:30 tối (giờ cao điểm), thì bạn phải mua vé thường. Vé tiết kiệm (giá thấp hơn vé thường) có hiệu lực cho các chuyến xe từ 9:30 sáng đến 4:00 chiều và sau 7:30 tối. Tàu hoạt động từ 4:00 sáng tới 23:00 đêm

Thiết kế các ca kiểm thử để kiểm tra yêu cầu trên dựa vào phương pháp phân vùng tương đương và phân tích giá trị biên.

Bài tập 3

Phần mềm “xét đơn cầm cố nhà” với đặc tả như sau: mỗi lần nhận 1 đơn xin cầm cố, phần mềm sẽ ra quyết định chấp thuận nếu 4 điều kiện sau thỏa mãn:

- Thu nhập hàng tháng của người nộp đơn nằm trong khoảng từ 1000\$ với 83333\$
- Số nhà xin cầm cố từ 1-5

Dùng phương pháp phân hoạch tương đương và phân tích giá trị biên để thiết kế các trường hợp kiểm thử cho phần mềm trên.

Bài tập 4

Bài toán tìm nghiệm thực cho phương trình bậc 2:

- Biết a,b,c là các số thực € [-10, 100]
- Đầu ra có thể gặp sau khi nhập bộ 3 số a,b,c và bấm nút Calculate là:
 1. Không phải là phương trình bậc 2.
 2. Phương trình vô nghiệm
 3. Phương trình có một nghiệm (đưa ra giá trị của nghiệm)
 4. Phương trình có 2 nghiệm (đưa ra giá trị của 2 nghiệm)
 5. Nhập sai dữ liệu
- Thiết kế các ca kiểm thử dùng phương pháp phân hoạch tương đương

Bài tập 5

Mô tả chức năng:

1. Khởi tạo màn hình:
 - Item 2 được check mặc định ở "Male"
 - Item 3 và 5 null
 - Item 4 ở trạng thái enable (có thể click được)
2. Mô tả xử lý chính:
 - Khi click vào item 4 thì xử lý như sau:

- Nếu là Male
 - Độ tuổi từ 18 đến 35 thì nhận được 100€
 - Độ tuổi từ 36 đến 50 thì nhận được 120€
 - Độ tuổi từ 51 đến 145 thì nhận được 140€
 - Độ tuổi khác thì hiển thị thông báo lỗi: "Xin vui lòng nhập độ tuổi chính xác"
- Nếu là Female
 - Độ tuổi từ 18 đến 35 thì nhận được 80€
 - Độ tuổi từ 36 đến 50 thì nhận được 110€
 - Độ tuổi từ 51 đến 145 thì nhận được 140€
 - Độ tuổi khác thì hiển thị thông báo lỗi: "Xin vui lòng nhập độ tuổi chính xác"

Dùng phương pháp phân hoạch tương đương và phân tích giá trị biên để xây dựng các ca kiểm thử cho chương trình trên.

Bài tập 6:

Cho chương trình xác định một tam giác có phải là tam giác cân không, biết người dùng nhập vào chiều dài 3 cạnh a, b, c của tam giác.

Sử dụng bảng quyết định thiết kế test case để kiểm thử chương trình trên.

Bài tập 7

Một chương trình khuyến mãi tri ân khách hàng của hãng A cho những khách hàng mua dòng điện thoại cao cấp của hãng diễn ra từ ngày 20/11/2017 đến hết ngày 31/12/2017 – dòng điện thoại được gọi là cao cấp nếu giá ≥ 20 tr. Theo đó, nếu khách hàng mua điện thoại cao cấp của hãng A trong khoảng thời gian đó sẽ được tặng 1 loa bluetooth và miếng dán màn hình. Ngoài ra đối với những khách hàng đã từng dùng dòng điện thoại cao cấp của hãng A, tính từ thời điểm đã mua cho đến thời điểm mua mới, nếu khoảng thời gian này.

- Không quá 1 năm thì khách hàng sẽ được giảm thêm 2 triệu trên giá sản phẩm.
- Từ trên 1 năm đến nhỏ hơn hoặc bằng 2 năm thì khách hàng được giảm thêm 1 triệu trên giá sản phẩm.

- Sử dụng các phương pháp bảng quyết định và phân tích giá trị biên thiết kế các test case kiểm thử các khuyến mãi mà người dùng nhận được khi mua điện thoại cao cấp hàng A? Có cần đặt câu hỏi gì thêm để làm rõ hơn yêu cầu?

Bài tập 8

Một hệ thống quản lý học tập có chức năng cho phép đăng bài viết, một bài viết khi đăng mới chỉ được phép cập nhật hoặc xóa trong vòng 15 phút kể từ lúc submit đăng bài, sau khoảng thời gian này bài viết không được phép chỉnh sửa hay xóa nữa và bài viết sẽ tự động được xuất bản trên hệ thống để người khác có thể đọc.

Ngoài ra, khi vừa soạn xong bài viết hoặc trong vòng 15 phút từ lúc submit bài viết, tác giả bài viết có quyền bấm nút “Publish” để xuất bản bài viết, và tất nhiên không được xóa hoặc cập nhật bài viết sau khi đã xuất bản. Sau khi một bài viết được xuất bản, tác giả bài viết muốn xóa hoặc cập nhật bài viết cần phải liên hệ với admin thực hiện. Chú ý sau khi admin chỉnh sửa bài viết đã xuất bản, bài viết đó vẫn ở trạng thái xuất bản để người khác đọc.

Vẽ lược đồ dịch chuyển trạng thái tin nhắn và viết các test case cho chúng.

Bài tập 9

Một hệ thống quản lý cho phép gửi và nhận tin nhắn trong hệ thống, khi người dùng nhận tin mới có trạng thái là tin chưa đọc, nếu người nhận mở ra đọc thì tin đó thành trạng thái đã đọc. Sau khi đọc tin, người dùng cũng có thể chuyển nó thành tin chưa đọc để ghi nhớ.

Ngoài ra, người dùng cũng có thể xóa tin tức, ban đầu tin xóa tạm nằm trong thùng rác, trong 24h kể từ lúc xóa người dùng có thể phục hồi lại trạng thái trước khi xóa, sau khoảng thời gian này tin sẽ bị xóa vĩnh viễn.

Vẽ lược đồ dịch chuyển trạng thái tin nhắn và viết các test case cho chúng.

TÓM TẮT

Bài học này cung cấp các kiến thức về:

- *Các kỹ thuật lấp漏洞 và giá trị biên*
- *Phân tích ràng buộc*
- *Tổ hợp điều kiện*
- *Các dạng bài tập kiểm thử hộp đen*

BÀI 4: THIẾT KẾ TEST CASE – WHITE - BOX

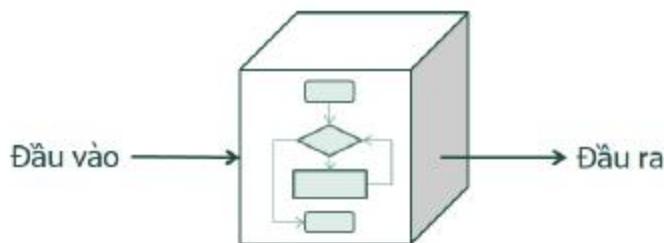
Nội dung gồm các phần sau:

- Tổng quan về kiểm thử hộp trắng
- Kiểm thử đường cơ bản
- Kiểm thử luồng điều khiển độ bao phủ
- Khiểm thử vòng lặp
- Kiểm thử luồng dữ liệu

4.1 TỔNG QUAN KIỂM THỬ HỘP TRẮNG

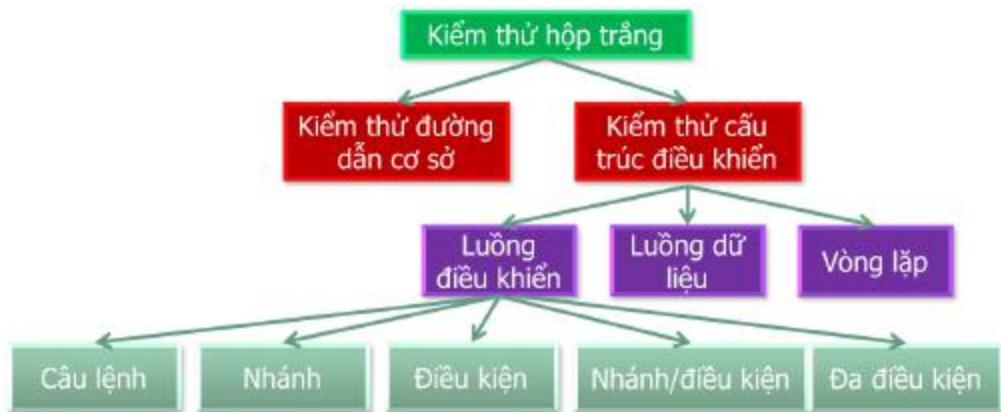
Kỹ thuật này còn gọi là structural testing, hoặc glass testing, hoặc open-box testing.

Tester cần biết cách thức (HOW) thực thi bên trong của phần mềm.



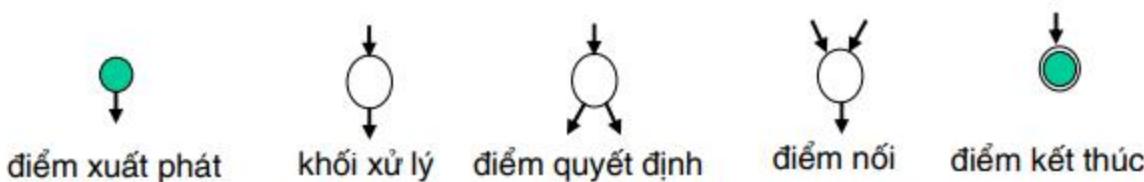
- **Ưu điểm**
 - Dễ dàng xác định loại dữ liệu để kiểm tra, nên việc kiểm tra hiệu quả hơn.
 - Nhờ biết mã nguồn, nên tester có thể phủ tối đa khi viết kịch bản kiểm thử.
- **Khuyết điểm**
 - Chi phí tăng vì tester cần đọc, hiểu mã nguồn.

- Khó xét hết các ngõ ngách trong mã nguồn nên có thể sót đường dẫn không được test.
- Tiếp cận kiểm thử hộp trắng
 - Kiểm thử đường dẫn cơ sở (basis path testing)
 - Kiểm thử cấu trúc điều khiển (control structural testing)[

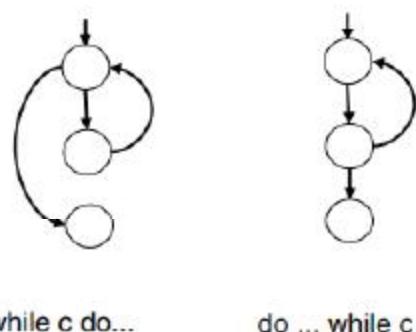
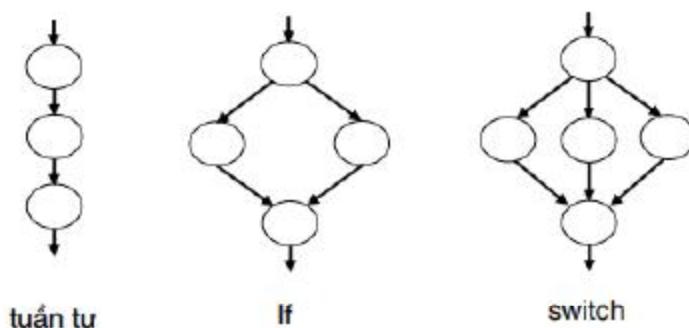


4.2 KIỂM THỬ ĐƯỜNG CƠ BẢN

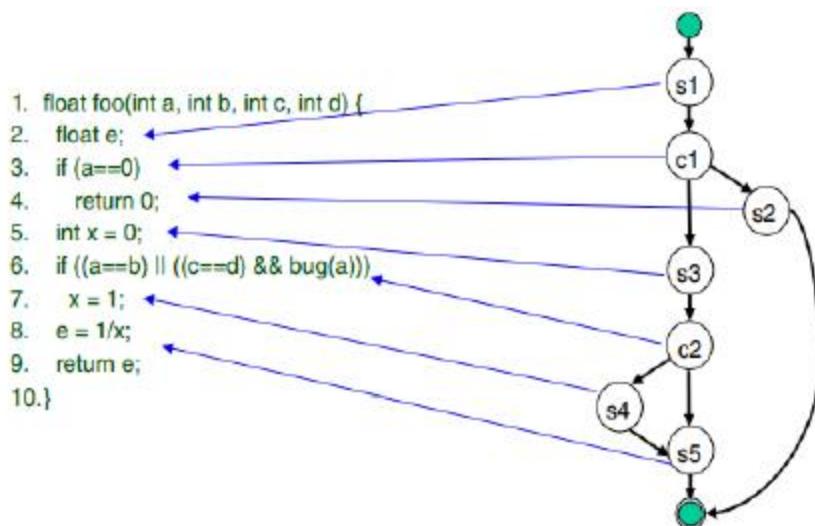
- Là một kỹ thuật dùng trong kiểm thử hộp trắng được Tom McCabe đưa ra đầu tiên. Đồ thị dòng gần giống đồ thị luồng điều khiển của chương trình.
- Là một trong nhiều phương pháp miêu tả thuật giải. Đây là phương pháp trực quan cho chúng ta thấy dễ dàng các thành phần của thuật giải và mối quan hệ trong việc thực hiện các thành phần này.
- Kỹ thuật đường cơ bản - đồ thị dòng có thể giúp những người thiết kế ca kiểm thử nhận được một độ phức tạp của logic thủ tục.
- Gồm 2 loại thành phần : các nút và các cung nối kết giữa chúng.
- Các loại nút trong đồ thị dòng điều khiển:



- Các kiểu cấu trúc thành phần đồ thị dòng:



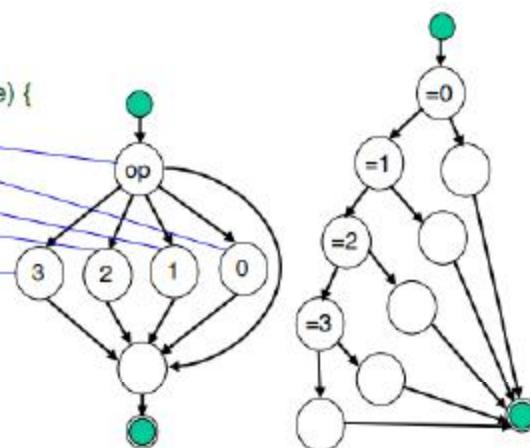
- Ví dụ :



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

```

1. int ProcessOp (int opcode) {
2.   switch (op) {
3.     case 0 : ...; break;
4.     case 1 : ...; break;
5.     case 2 : ...; break;
6.     case 3 : ...; break;
7.   }
  
```



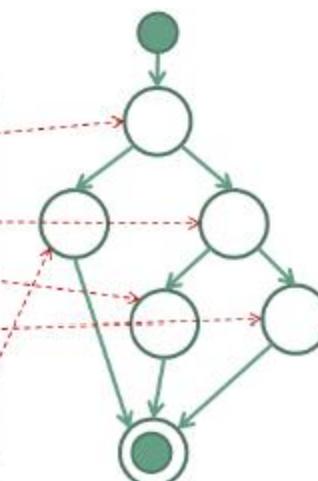
- Độ phức tạp Cyclomatic $C = V(G)$ của đồ thị dòng điều khiển được tính bởi 1 trong các công thức sau: $V(G) = E - N + 2$, trong đó E là số cung, N là số nút của đồ thị. $V(G) = P + 1$, nếu là đồ thị dòng điều khiển nhị phân (chỉ chứa các nút quyết định luận lý - chỉ có 2 cung xuất True/False) và P số nút quyết định. Độ phức tạp Cyclomatic C chính là số đường thi hành tuyến tính độc lập của phần mềm cần kiểm thử.

Ví dụ: Viết các test case kiểm thử chương trình giải và biện luận phương trình $ax + b = 0$, với đó a, b là các số thực. Chương trình như sau:

```

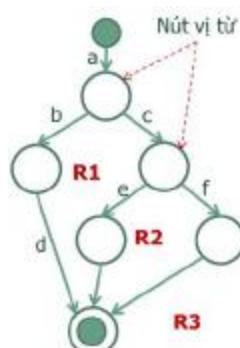
void giaoPTBacNhat(double a, double b)
{
    if (a == 0)
        if (b == 0)
            printf("PT vo so nghiem\n");
        else
            printf("PT vo nghiem\n");
    else
        printf("PT co nghiem:%f", -b / a);
}
  
```

Chương trình minh họa bằng C++



Xác định độ phức tạp Cyclomatic

- Số lượng các vùng của G : 3
- $V(G) = E - N + 2 = 8 - 7 + 2 = 3$
- $V(G) = P + 1 = 2 + 1 = 3$



Xác định tập đường dẫn cơ sở

- a, b, d
- a, c, e,g
- a, c, f,h

Đường dẫn	Đầu vào		Đầu ra mong muốn
	a	b	
a, b, d	5	-10	Nghiem x = 2
a, c, e	0	0	PT vo so nghiem
a, c, f	0	5	PT vo nghiem

4.3 KIỂM THỬ ĐỘ BAO PHỦ

4.3.1 Một số khái niệm

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

Mỗi phần mềm có từ 1 đến n (có thể rất lớn) đường thi hành khác nhau.

Độ bao phủ (**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.

Kiểm thử bao phủ dùng kiểm tra mức độ phủ (coverage) của các test case. Các loại kiểm thử bao phủ:

- Phủ câu lệnh (statement coverage)
- Phủ nhánh (branch coverage)
- Phủ đường (path coverage)
- Phủ điều kiện (condition coverage)
- Phủ nhánh và điều kiện (condition and branch coverage)
- Phủ đa điều kiện (multi-conditions coverage)

Các cấp độ bao phủ:

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

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

Phủ cấp 0: kiểm thử những gì có thể kiểm thử được, phần còn lại để người dùng phát hiện và báo lại sau. Đây là mức độ kiểm thử không thực sự có trách nhiệm.

Phủ cấp 1: kiểm thử sao cho mỗi lệnh được thực thi ít nhất 1 lần. Bao phủ câu lệnh (**statement coverage**):

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

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

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

4.3.2 Phủ câu lệnh

Phủ câu lệnh (statement coverage): **mỗi câu lệnh** được thực thi ít nhất một lần.

Ví dụ: Hàm xét học bổng như sau

```

bool xetHocBong(double d1, double d2, double d3, double diemRL)
{
    double diemTB = 0.0;
    if (d1 >= 5 && d2 >= 5 && d3 >= 5)
        diemTB = (d1 + d2 + d3) / 3;

    bool kq = false;
    if ((diemTB >= 8.5) || (diemTB >= 7.0 && diemRL >= 70))
        kq = true;

    return kq;
}

```

Chương trình minh họa bằng C++

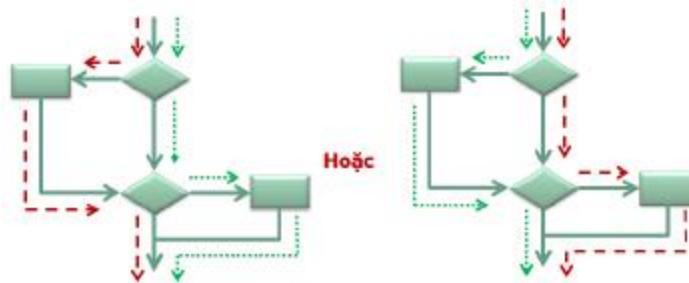
Để kiểm thử phủ câu lệnh trong hàm trên chỉ cần test case sau:

Đầu vào				Đầu ra mong muốn
d1	d2	d3	diemRL	
7	7	7	70	Hàm trả về kết quả true

4.3.3 Phù nhánh

Phù nhánh (branch coverage): **mỗi nhánh** phải được thực hiện ít nhất một lần.

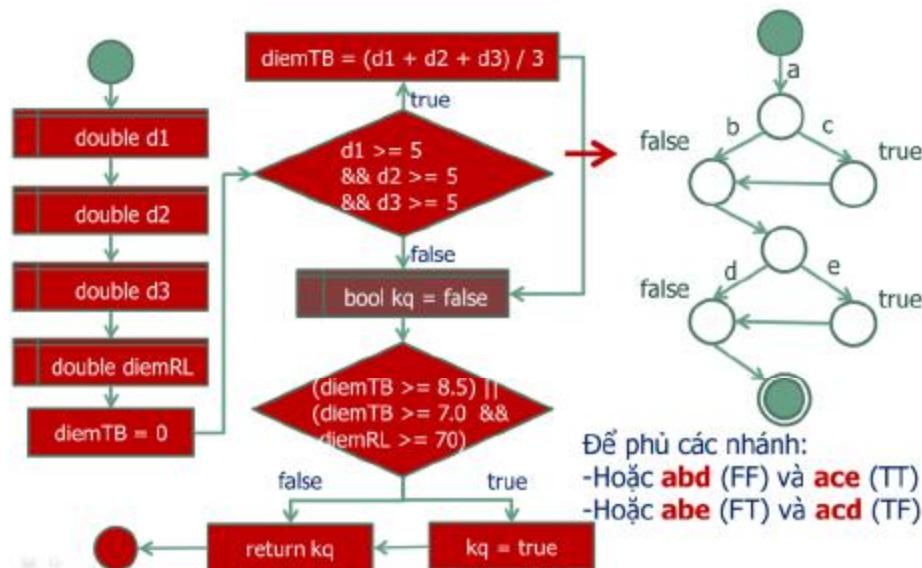
Phù nhánh **đảm bảo** phủ câu lệnh.



Nhận xét: trong đồ thị luồng

Phù nhánh có nghĩa là **các cạnh** được **đi qua ít nhất** một lần.

- Để phù nhánh phải thiết kế dữ liệu kiểm thử sao cho **mỗi nút vị từ** (predicate) xảy ra tất cả các kết quả (true/false) có thể của nó, nên phù nhánh còn gọi là **phù quyết định** (decision coverage).

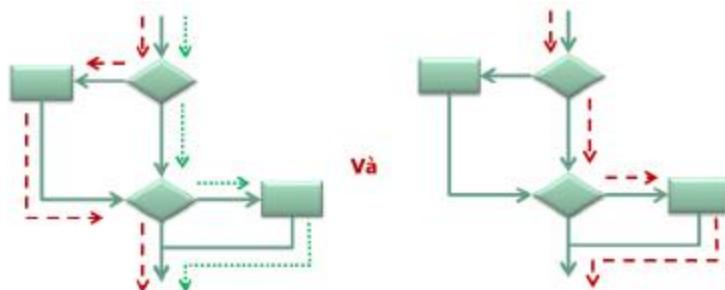


- Chọn abd và ace kiểm thử phủ nhánh, thiết kế các test case như sau:

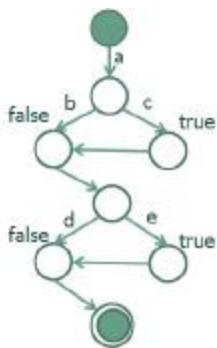
Nhánh	Đầu vào				Đầu ra mong muốn
	$d1$	$d2$	$d3$	$diemRL$	
abd	4	5	5	70	Hàm trả về kết quả false
ace	7	7	7	70	Hàm trả về kết quả true

4.3.4 Phủ đường dẫn

- Phủ đường dẫn (path coverage): mỗi đường dẫn qua ít nhất một lần.
- Đường là một tập các nhánh, nên **phủ đường chắc chắn phủ nhánh**, nhưng ngược lại chưa chắc đúng.



- Ví dụ: Quay lại ví dụ xét học bối, phủ đường dẫn trong đồ thị luồng bằng 4 test case sau đi qua các đường abd, ace, acd, abe.
- Chú ý: không có test case nào đi qua đường abe (đường bắt khả thi). Do đó, để phủ đường chỉ cần 3 test case sau:



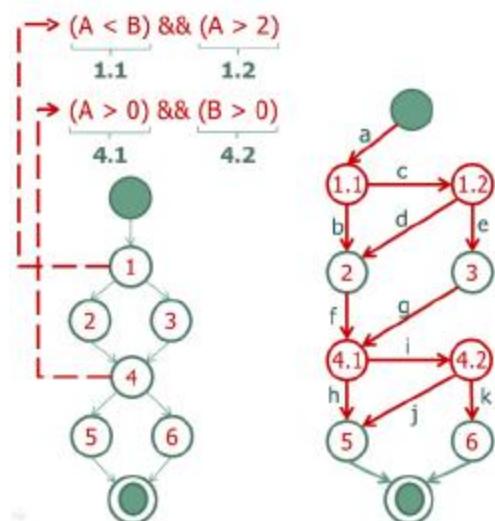
Đầu vào				Đầu ra mong muốn
d1	d2	d3	diemRL	
4	5	5	70	Hàm trả về kết quả false
7	7	7	70	Hàm trả về kết quả true
7	7	7	60	Hàm trả về kết quả false

4.3.5 Phủ Điều kiện

- Thông thường vi từ quyết định nhánh được thực thi là **tổ hợp nhiều điều kiện**.
- Ví dụ:
 - if ($d1 \geq 5 \ \&\& d2 \geq 5 \ \&\& d3 \geq 5$) ...
 - if ((diemTB ≥ 8.5) || (diemTB $\geq 7.0 \ \&\& diemRL \geq 70$)) ...
 - if ((nam % 400 == 0) || (nam % 4 == 0 $\&\&$ nam % 100 != 0)) ...
- Phủ điều kiện (condition coverage): **mỗi điều kiện trong các vị** từ được thực hiện ít nhất một lần cho cả trường hợp **true và false (không bắt buộc** các kết hợp giữa chúng).
- Phủ điều kiện **chứa chắc** đảm bảo phủ các nhánh.

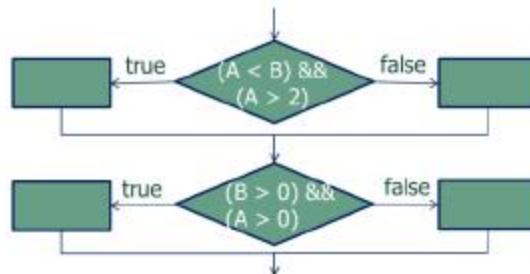
4.3.6 Phủ Nhánh và Điều kiện

- Phủ nhánh và điều kiện: **mỗi điều kiện** trong các vị và **các nhánh** rẽ từ các vị từ đó cũng được thực thi ít nhất một lần.
- Ví dụ: phủ nhánh và điều kiện trong sơ đồ luồng sau với các test case
 - A = 3, B = 4
 - A = -3, B = 4
 - A = -3, B = -4

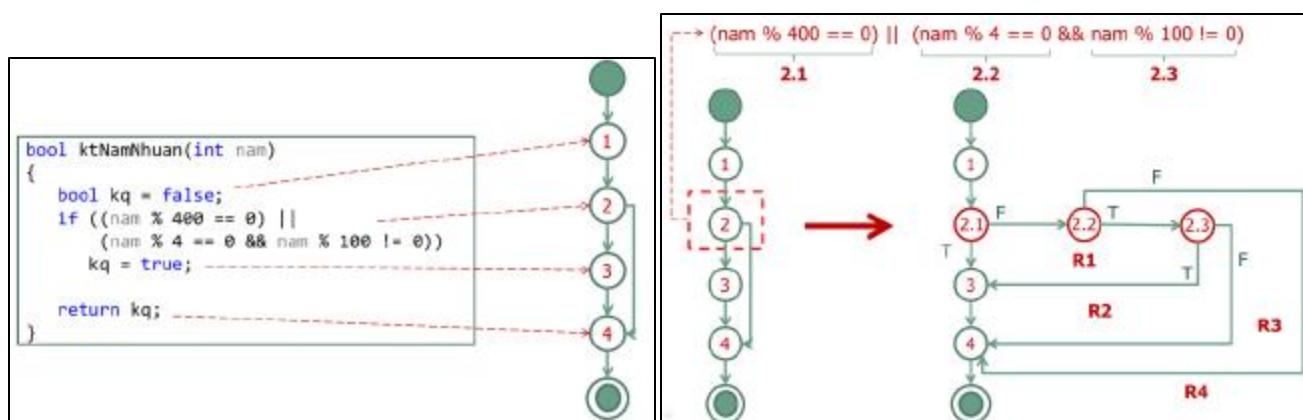


Ta chọn các **đường dẫn phủ nhánh** của đồ thị này:

- a, c, e, g, i, k
- a, c, d, f, i, j
- a, b, f, h



Ví dụ: thiết kế các test case phủ nhánh và điều kiện của hàm kiểm tra năm nhuận.



- Độ phức tạp Cyclomatic:
 - Số lượng các vùng của G: 4
 - $V(G) = E - N + 2 = 10 - 8 + 2 = 4$
 - $V(G) = P + 1 = 3 + 1 = 4$
- Các đường dẫn sau đảm bảo phủ nhánh và điều kiện:
 - a, c, e, f
 - a, c, e, g
 - a, c, d
 - a, b
- Để phủ nhánh và điều kiện cần tối thiểu các test case minh họa:

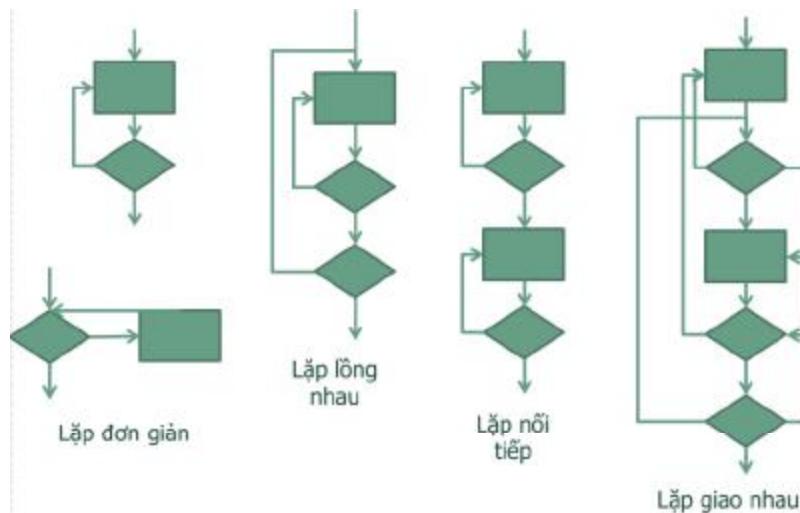
Đường dẫn	Đầu vào	Đầu ra mong muốn
	Năm	
a, c, e, f	2016	Năm nhuận
a, c, e, g	1900	Không phải năm nhuận
a, c, d	2017	Không phải năm nhuận
a, b	1600	Năm nhuận

4.4 KIỂM THỬ VÒNG LẶP

Vòng lặp là các lệnh phổ biến và là cơ bản trong các ngôn ngữ lập trình. Tuy nhiên, việc kiểm thử đối với các vòng lặp là rất phức tạp, việc kiểm thử tất cả các trường hợp của vòng lặp nhiều khi là không thể vì số lượng các trường hợp kiểm thử là rất lớn. Trong phần này sẽ trình bày một số cải tiến để kiểm thử cho các vòng lặp.

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

- Lệnh lặp đơn giản: thân chỉ chứa các lệnh khác chứ không chứa lệnh lặp khác.
- Lệnh lặp lồng nhau: thân chứa ít nhất lệnh lặp khác...
- Lệnh lặp liền kề: 2 hay nhiều lệnh lặp kế tiếp nhau
- Lệnh lặp giao nhau: 2 hay nhiều lệnh lặp giao nhau.



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

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

- chạy 0 bước.
- chạy 1 bước.
- chạy 2 bước.
- chạy k bước, k là giá trị nào đó thỏa $2 < k < n-1$.
- chạy n-1 bước
- chạy n bước
- chạy n+1 bước.

2. Kiểm thử các vòng lặp liền kề: Kiểm thử tuần tự từng vòng lặp từ trên xuống, mỗi vòng thực hiện kiểm thử.
3. Kiểm thử vòng lặp lồng nhau:

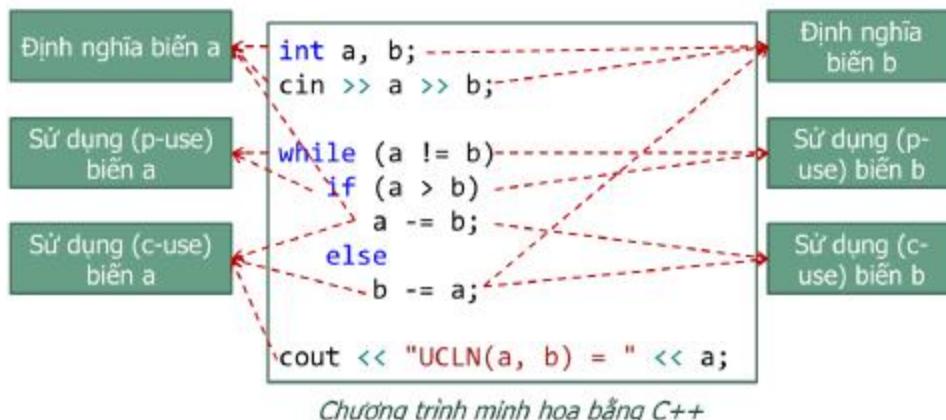
Kiểm thử tuần tự từng vòng lặp từ trong ra ngoài theo đề nghị sau đây:

- Kiểm thử vòng lặp trong cùng: cho các vòng ngoài chạy với giá trị min, kiểm thử vòng lặp trong cùng.
 - Kiểm thử từng vòng lặp còn lại: cho các vòng ngoài nó chạy với giá trị min, còn các vòng bên trong nó chạy với giá trị điển hình.
4. Riêng các vòng lặp giao nhau thì thường do việc viết code chưa tốt tạo ra vậy nên cấu trúc lại đoạn code sao cho không chứa dạng giao nhau này.

4.5 KIỂM THỬ LUÔNG DỮ LIỆU

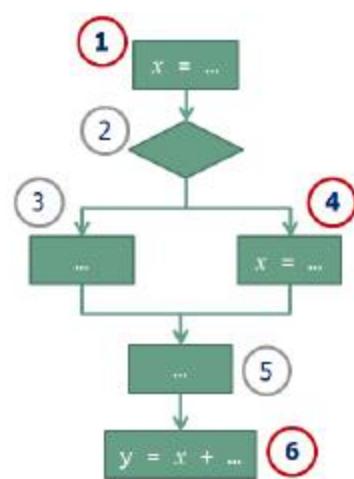
- Phương pháp kiểm thử luồng dữ liệu sẽ kiểm thử **vòng đời của biến** trong từng luồng thực thi của chương trình.
- Vòng đời của một biến được thể hiện thông qua ba hành động:
 - Định nghĩa biến (**Define**).
 - Sử dụng biến (**Use**).
 - Xóa biến (**Delete**).

- Kiểm thử luồng dữ liệu lựa chọn các đường dẫn để kiểm thử dựa trên **vị trí** định nghĩa (**define**) và sử dụng (**use**) các biến trong chương trình.
- Đặt
 - **DEF(S)** = {X| câu lệnh S chứa định nghĩa biến X}
 - X = ... (nhập, gán, gọi thủ tục)
 - **USE(S)** = {X| câu lệnh S sử dụng biến X}
 - ... = X ... (xuất, gán, điều kiện)
 - Nếu câu lệnh S là biểu thức vị từ thì ký hiệu là **p-use**
 - Nếu câu lệnh S là biểu thức tính toán thì ký hiệu là **c-use**



- Kiểm thử luồng dữ liệu giúp phát hiện các vấn đề sau:
 - Một biến được khai báo, nhưng không sử dụng.
 - Một biến sử dụng nhưng không khai báo.
 - Một biến được định nghĩa nhiều lần trước khi được sử dụng.
 - Xóa biến trước khi sử dụng.
- Cho biến $x \in \text{DEF}(S) \cap \text{USE}(S')$, trong đó S và S' là các câu lệnh.
- **Đường dẫn DU** (definition-use path) của biến x là đường nối từ S đến S' trên đồ thị luồng sao cho **không tồn tại** một định nghĩa nào khác của x trên đường này.

- Cặp **DU** (definition-use pairs) của biến là cặp S và S', sao cho tồn tại ít nhất một đường DU nối S và S'.
- DEF(1) = {x, ...}
- DEF(4) = {x, ...}
- USE(6) = {x, ...}
- (1, 2, 3, 5, 6) và (4, 5, 6) là các đường dẫn DU của biến x.
- (1, 2, 4, 5, 6) không là đường của biến x vì nó được định nghĩa lại ở câu lệnh 4.
- Các cặp DU là (1, 6) và (4, 6)



Ví dụ: cho biết đâu là cặp DU của biến **kq**

```

bool ktNguyenTo(int n)
{
    bool kq = false; // (1)
    if (n >= 2)
    {
        kq = true; // (2)
        for (int i=2; i<=sqrt(n) && kq == true; i++) // (3)
            if (n % i == 0)
                kq = false; // (4)
    }
    return kq; // (5)
}

```

Ví dụ: Cho biết đâu là cặp DU của biến **heSo**

```

float tinhLuong(int loaiNhanVien, int soGioLam)
{
    float heSo = 1.0f; // (1)
    if (loaiNhanVien == 1)
    {
        heSo = 1.5f; // (2)
        if (soGioLam > 40)
            heSo = heSo + 0.2f; // (3)
    } else if (loaiNhanVien == 2)
        heSo = 1.2f; // (4)

    return 1200000 * soGioLam * heSo; // (5)
}

```

- Chiến lược kiểm thử luồng dữ liệu
 - Mỗi **cặp DU** nên được thực thi ít nhất một lần.
 - Mỗi **đường DU** đơn giản (không chứa vòng lặp) nên được thực hiện ít nhất một lần.

4.6 BÀI TẬP THIẾT KẾ TEST CASE – WHITE BOX

Bài tập 1

- Số test case tối thiểu để phủ câu lệnh?
- Số test case tối đa phủ đường dẫn cơ sở?
- Số test case tối thiểu phủ điều kiện?
- Thiết kế test case phủ đường dẫn cơ sở?
- Thiết kế các test case phủ nhánh của hàm?

```
int reverse(int n)
{
    bool isNegative = false;
    if (n < 0)
    {
        isNegative = true;
        n = -n;
    }

    int result = 0;
    while (n > 0)
    {
        result = result * 10 + n % 10;
        n /= 10;
    }
    if (isNegative == true)
        result = - result;
    return result;
}
```

Bài tập 2

Xét đoạn code sau, yêu cầu: thiết kế các ca kiểm thử đạt bao phủ mức 4 (phủ nhánh và điều kiện).

```
using namespace std;
# include<iostream>

main() {
    int n;
    cout<< "Nhập n" << endl;
    cin >>n;
    for (n; n>0; n--) {
        cout<< n << ",";
    }
    cout<< "Kết thúc";
    return 0;
}
```

Bài tập 3

- Viết các test case phủ nhánh và điều kiện hàm:

```

bool xetHocBong(double diemMH[], int soMH, int diemRL)
{
    if (soMH > 0)
    {
        int i;
        double tongDiem = 0;
        for (i = 0; i < soMH; i++)
            tongDiem = tongDiem + diemMH[i];

        double diemTB = tongDiem / soMH;
        if (diemTB >= 8 || (diemTB >= 7 & diemRL >= 80))
            return true;
    }
    return false;
}

```

Bài tập 4

Xét đoạn code sau, yêu cầu thiết kế các ca kiểm thử đạt bao phủ mức 2 và 4.

```

using namespace std;
#include <iostream>
main() {
    int a,b,c,d,x,y;
    cout<<"Nhập a, b, c, d, x, y"<<endl;
    cin>>a>>b>>c>>d>>x>>y;
    if (a>0&&b==1){x=x+1;}
    if (c==3 || d<0) {y=0;}
    cout<<"x = "<<x<<endl;
    cout<<"y = "<<y<<endl;
}

```

Bài tập 5

Thiết kế các ca kiểm thử thỏa mãn tiêu chuẩn phủ cấp 4

```

double average(int[] values, int min, int max)
{
    int sum=0, count=0, item=0;
    double average= 0;
    while(values[item] !=-999 && item <100) {
        if (values[item]>= min && values[ item] <= max) {

```

```
        sum += values [item];  
        count ++;  
    }  
    item++;  
}  
if (count>0) average= (double) sum/count;  
else average =-999;  
return average;  
}
```

TÓM TẮT

Bài học này cung cấp các kiến thức về:

- *Kiểm thử luồng điều khiển và độ bao phủ*
- *Kiểm thử vòng lặp*
- *Mối quan hệ hàm và dữ liệu*
- *Kiểm thử luồng dữ liệu*
- *Các dạng bài tập kiểm thử hộp trắng*

BÀI 5: LỖI PHẦN MỀM VÀ HỆ THỐNG QUẢN LÝ LỖI

Nội dung gồm các phần sau:

- *Lỗi phần mềm*
- *Nguyên nhân gây ra lỗi*
- *Các lỗi thường gặp*
- *Giới thiệu về hệ thống quản lý bug*
- *Vòng đời của bug trên hệ thống quản lý bug*
- *Hệ thống quản lý bug Bugzilla*

5.1 TỔNG QUAN VỀ LỖI PHẦN MỀM

Lỗi phần mềm thuộc nhiều loại. Một lỗi là một lỗi không có vấn đề gì. Nhưng đôi khi, điều quan trọng là phải hiểu được bản chất, ý nghĩa của nó và nguyên nhân để xử lý nó tốt hơn. Điều này giúp cho việc đổi ứng nhanh hơn và quan trọng nhất, đổi ứng thích hợp. Chúng ta sẽ thảo luận về các loại lỗi phần mềm phổ biến và làm thế nào để xác định chúng trong quá trình kiểm thử với một số ví dụ và bài tập đơn giản. Chúng ta hãy bắt đầu bằng việc xác định lỗi phần mềm và lỗi.

Lỗi phần mềm và Bugs

Theo định nghĩa tại Wikipedia “Một lỗi là một sai lệch dựa vào độ chính xác hoặc tính đúng đắn” và “ Một lỗi phần mềm là một lỗi, lỗi hỏng, thất bại, hoặc có lỗi trong một chương trình máy tính hoặc hệ thống đó là nguyên nhân nó tạo ra kết quả không chính xác hoặc không mong muốn, hoặc vận hành theo cách không được định hướng trước”.

Vì vậy, sau đây có thể đưa ra kết luận:

- Lỗi là sự khác nhau của kết quả thực tế và kết quả mong đợi
- Lỗi là một loại của lỗi phần mềm
- Lỗi có thể được giới thiệu như là kết quả của việc không hoàn thành hoặc sai yêu cầu hoặc do vẫn đề nhập dữ liệu của con người

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 kỳ failure nào là một errors.

5.2 NGUYÊN NHÂN GÂY RA LỖI THƯỜNG GẶP

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:

- 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.

- 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.

- 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-khởi xưởng, 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 đối với nhà phát triển. Như vậy những thay đổi "nhỏ" có thể, cuối cùng, gây ra lỗi phần mềm.

- **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 - 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

- **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...

- **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.

- Thiếu sót trong quá trình thử nghiệm

Thiếu sót trong 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:

- 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 chỉ dẫn không phù hợp trong những lý do cho lỗi này.
- Không hoàn chỉnh sửa các lỗi được phát hiện do sơ suất hay thời gian áp lực.

- Các lỗi thủ tục

Các thủ tục trực tiếp cho người sử dụng đối với các hoạt động là cẩn 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.

- 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.

5.3 CÁC LỖI THƯỜNG GẶP TRONG PHẦN MỀM

1. Lỗi chức năng

Phần mềm có một lỗi chức năng nếu một cái gì đó mà ta mong muốn phần mềm làm là rắc rối, khó hiểu, không khả thi.

2. Lỗi giao tiếp

Những lỗi xảy ra trong giao tiếp giữa phần mềm và người dùng cuối. Bất cứ điều gì mà người dùng cuối cần biết để sử dụng các phần mềm nên được làm sẵn có trên màn hình. Ví dụ về lỗi giao tiếp: không cung cấp bảng hướng dẫn trợ giúp/menu.

3. Lỗi thiếu lệnh

Điều này xảy ra khi một lệnh dự kiến là thiếu.

4. Lỗi cú pháp

Lỗi cú pháp là những từ sai chính tả hay ngữ pháp câu không chính xác và rất rõ ràng trong khi kiểm thử giao diện phần mềm. Trình biên dịch sẽ cảnh báo các nhà phát triển về bất kỳ lỗi cú pháp xuất hiện trong code.

5. Lỗi lỗi xử lý

Bất kỳ lỗi nào xảy ra khi người dùng đang tương tác với các phần mềm cần phải được xử lý một cách rõ ràng và có ý nghĩa. Nếu không, nó được gọi là một lỗi Xử lý lỗi. Thông báo lỗi đưa ra không chỉ ra thực sự gây ra lỗi là gì. Đó là do nó thiếu trường bắt buộc, lỗi đang lưu, lỗi tải trang hoặc lỗi hệ thống. Do đó, đây là một lỗi lỗi xử lý

6. Lỗi tính toán

Những lỗi này xảy ra do các lý do sau đây: logic không tốt, công thức tính toán không chính xác, kiểu dữ liệu không phù hợp, lỗi lập trình.

7. Lỗi dòng điều khiển

Việc kiểm soát flow của phần mềm mô tả những gì sẽ làm gì tiếp theo và dựa trên điều kiện. Ví dụ, người dùng điền mẫu đơn và sử dụng: Save, Save và Close, và Cancel. Nếu người dùng nhập "Save and Close", thông tin trong form lưu và đóng form lại. Nếu nhập vào button "Save and Close" mà không đóng form, thì đó là lỗi dòng điều khiển.

5.4 TÌM LỖI VÀ PHÂN TÍCH LỖI

5.4.1. Checklist kiểm tra mã nguồn

- Các lỗi truy xuất dữ liệu (Data Reference Errors)

- Các lỗi định nghĩa/khai báo dữ liệu (Data-Declaration Errors)
- Các lỗi tính toán (Computation Errors)
- Các lỗi so sánh (Comparison Errors)
- Các lỗi luồng điều khiển (Control-Flow Errors)
- Các lỗi giao tiếp (Interface Errors)
- Các lỗi nhập/xuất (Input/Output Errors)
- Các lỗi khác (Other Checks)

Các lỗi truy xuất dữ liệu (Data Reference Errors)

1. Dùng biến chưa có giá trị xác định.

```
int i, count;
for (i = 0; i < count; i++) {...}
```

2. Dùng chỉ số của biến array nằm ngoài phạm vi.

```
int list[10];
if (list[10] == 0) {...}
```

3. Dùng chỉ số không thuộc kiểu nguyên của biến array.

```
int list[10];
double idx=3.1416;
if (list[idx] == 0) {...}
```

4. Tham khảo đến dữ liệu không tồn tại (dangling references).

```
int *pi;
if (*pi == 10) {...} //pi đang tham khảo đến địa chỉ không hợp lệ - Null
int *pi = new int;
...
delete (pi);
if (*pi = 10) {...} //pi đang tham khảo đến địa chỉ
//mà không còn dùng để chứa số nguyên
```

5. Truy xuất dữ liệu thông qua alias có đảm bảo thuộc tính dữ liệu đúng.

```
int pi[10];
pi[1] = 25;
char* pc = pi;
if (pc[1] == 25) {...} //pc[1] khác với pi[1];
```

6. Thuộc tính của field dữ liệu trong record có đúng với nội dung gốc.

```
struct {int i; double d;} T_Rec;
T_Rec rec;
read(fdin,&rec, sizeof(T_Rec));
if (rec.i == 10) {...} //lỗi nếu field d nằm trước i
//trong record gốc trên file
```

7. Cấu trúc kiểu record có tương thích giữa client/server.

```
Private Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128 ' Maintenance string
End Type

Private Declare Function GetVersionEx Lib "kernel32" _
    Alias "GetVersionExA" (lpVersionInformation As
    OSVERSIONINFO) As Long
```

8. Dùng chỉ số bị lệch.

```
int i, pi[10];
for (i = 1; i <= 10; i++) pi [i] = i;
```

9. Class có hiện thực đủ các tác vụ trong interface mà nó hiện thực.
10. Class có hiện thực đủ các tác vụ "pure virtual" của class cha mà nó thừa kế.

Các lỗi khai báo dữ liệu

1. Tất cả các biến đều được định nghĩa hay khai báo tường minh chưa.

```
int i;  
  
extern double d;  
  
d = i*10;
```

2. Định nghĩa hay khai báo đầy đủ các thuộc tính của biến dữ liệu chưa.

```
static int i = 10;
```

3. Biến array hay biến chuỗi được khởi động đúng chưa.

```
int pi[10] = {1, 5, 7, 9} ;
```

4. Kiểu và độ dài từng biến đã được định nghĩa đúng theo yêu cầu chưa.

```
short IPAddress;  
  
byte Port;
```

5. Giá trị khởi động có tương thích với kiểu biến.

```
short IPAddress = inet_addr("203.7.85.98");  
  
byte Port = 65535;
```

6. Có dùng các biến ý nghĩa khác nhau nhưng tên rất giống nhau không.

```
int count, counts;
```

Các lỗi tính toán (Computation Errors)

1. Thực hiện phép toán trên toán hạng không phải là số.

```
CString s1, s2;  
  
int ketqua = s1/s2;
```

2. Thực hiện phép toán trên các toán hạng có kiểu không tương thích.

```
byte b;
```

```
int i;
double d;
b = i * d;
```

3. Thực hiện phép toán trên các toán hạng có độ dài khác nhau.

```
byte b;
int i;
b = i * 500;
```

4. Gán dữ liệu vào biến có độ dài nhỏ hơn.

```
byte b;
int i;
b = i * 500;
```

5. Kết quả trung gian bị tràn.

```
byte i, j, k;
i = 100; j = 4;
k = i * j / 5;
```

6. Phép chia có mẫu bằng 0.

```
byte i, k;
i = 100 / k;
```

7. Mất độ chính xác khi mã hóa/giải mã số thập phân/số nhị phân.

8. Giá trị biến nằm ngoài phạm vi ngữ nghĩa.

```
int tuoi = 3450;
tuoi = -80;
```

9. Thứ tự thực hiện các phép toán trong biểu thức mà người lập trình mong muốn có tương thích với thứ tự mà máy thực hiện. Người lập trình hiểu đúng về thứ tự ưu tiên các phép toán chưa.

```
double x1 = (-b-sqrt(delta)) / 2*a;
```

10. Kết quả phép chia nguyên có chính xác theo yêu cầu không.

```
int i = 3;
if (i/2*2) == i) {...}
```

Các lỗi so sánh (Comparison Errors)

1. So sánh 2 dữ liệu có kiểu không tương thích.

```
int ival;
char sval[20];
if (ival == sval) {...}
```

2. So sánh 2 dữ liệu có kiểu không cùng độ dài.

```
int ival;
char cval;
if (ival == cval) {...}
```

3. Toán tử so sánh đúng ngữ nghĩa mong muốn. Dễ nhầm giữa = và !=, <= và >=, and và or...

4. Có nhầm lẫn giữa biểu thức Bool và biểu thức so sánh.

```
if (2 < i < 10) {...}
if (2 < i && i < 10) {...}
```

5. Có hiểu rõ thứ tự ưu tiên các phép toán.

```
if(a==2 && b==2 || c==3) {...}
```

6. Cách thức tính biểu thức Bool của chương trình dịch như thế nào.

```
if(y==0 || (x/y > z))
```

Các lỗi luồng điều khiển (Control-Flow Errors)

1. Thiếu thực hiện 1 số nhánh trong lệnh quyết định theo điều kiện số học.

```
switch (i) {
    case 1: ... //cần hay không cần lệnh break;
```

case 2: ...

case 3: ...

}

2. Mỗi vòng lặp thực hiện ít nhất 1 lần hay sẽ kết thúc.

`for (i=x ; i<=z; i++) {...} //nếu x > z ngay từ đầu thì sao.`

`for (i = 1; i <= 10; i--) {...} //có dừng được không.`

3. Biên của vòng lặp có bị lệch.

`for (i = 0; i <= 10; i++) {...} //hay i < 10`

4. Có đủ và đúng cặp token begin/end, {}

Các lỗi giao tiếp (Interface Errors)

1. Số lượng tham số cụ thể được truyền có = số tham số hình thức của hàm gọi
2. thứ tự các tham số có đúng không.
3. thuộc tính của từng tham số thực có tương thích với thuộc tính của tham số hình thức tương ứng của hàm được gọi.

`char* str = "Nguyen Van A";`

`MessageBox (hWnd, str,"Error", MB_OK); //sẽ bị lỗi khi dịch ở chế độ Unicode`

4. Đơn vị đo lường của tham số thực giống với tham số hình thức.

`double d = cos (90);`

5. Tham số read-only có bị thay đổi nội dung bởi hàm không.

6. Định nghĩa biến toàn cục có tương thích giữa các module chức năng không.

Các lỗi nhập/xuất (Input/Output Errors)

1. Lệnh mở/tạo file có đúng chế độ và định dạng truy xuất file.

```
if ((fdout = open ("tmp0", O_WRONLY| O_CREAT|
O_BINARY, S_IREAD| S_IWRITE)) < 0)
pr_error_exit("Khong the mo file tmp0 de ghi");
```

```
if ((fdtmp = open ("tmp2", O_RDWR | O_CREAT |  
O_BINARY, S_IREAD | S_IWRITE)) < 0)
```

2. Kích thước của buffer có đủ chứa dữ liệu đọc vào không.

```
char buffin[100];  
  
sl = read(fd, buffin, MAXBIN); //MAXBIN <= 100
```

3. Có mở file trước khi truy xuất không.

4. Có đóng file lại sau khi dùng không. Có xử lý điều kiện hết file .

5. Có xử lý lỗi khi truy xuất file không.

6. Chuỗi xuất có bị lỗi từ vựng và cú pháp không.

Các lỗi khác (Other Checks)

1. Có biến nào không được tham khảo trong danh sách tham khảo chéo (cross-reference).
2. Cái gì được kỳ vọng trong danh sách thuộc tính.
3. Có các cảnh báo hay thông báo thông tin.
4. Có kiểm tra tính xác thực của dữ liệu nhập chưa.
5. Có thiếu hàm chức năng.

5.4.2. Qui tắc xác định lỗi phần mềm

- Quy tắc 1: Phần mềm không thực hiện một số thứ giống như mô tả trong bản đặc tả phần mềm
- Quy tắc 2: Phần mềm thực hiện một số việc mà bản đặc tả yêu cầu nó không được thực hiện
- Quy tắc 3: Phần mềm thực hiện một số chức năng mà bản đặc tả không đề cập
- Quy tắc 4: Phần mềm không thực hiện một số việc mà bản đặc tả không đề cập tới, nhưng là những việc nên làm
- Quy tắc 5: Trong con mắt của người kiểm thử, phần mềm là khó hiểu, khó sử dụng, chậm đỗi với người sử dụng.

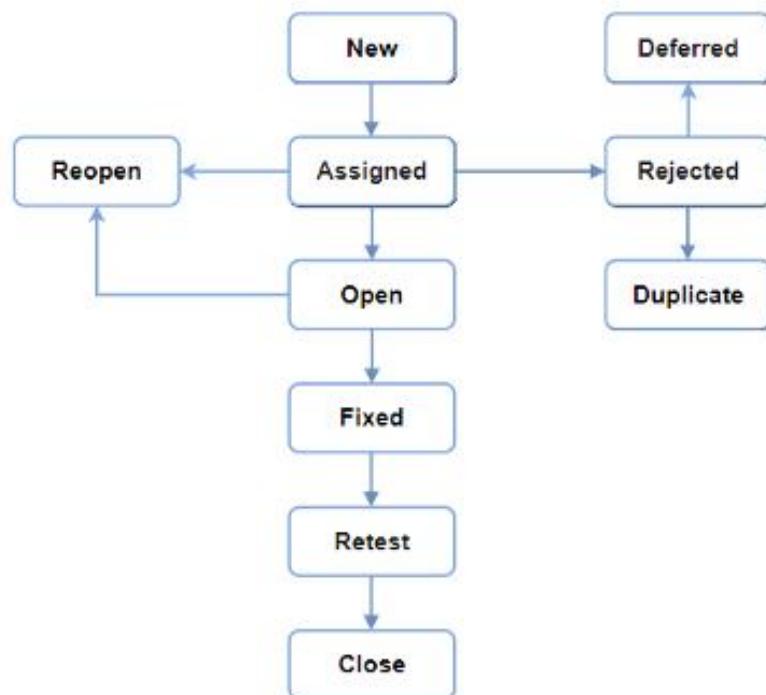
5.5 GIỚI THIỆU VỀ HỆ THỐNG QUẢN LÝ BUG

Lỗi phần mềm là lỗi, thiếu sót, thất bại, hoặc lỗi trong chương trình máy tính hoặc hệ thống sản xuất 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 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 hệ thống theo dõi lỗi là 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 ...

10 công cụ quản lý bug hiệu quả: Bugherd, Doorbell, Usersnap, Taperecorder, Jira, Rollbar, Bugclipper, Aptelligent, Promoter.io, Lighthouse.

5.6 VÒNG ĐỜI CỦA BUG TRÊN HỆ THỐNG QUẢN LÝ BUG



Hình 5.1: Vòng đời của bug

5.6.1 Những thông số cần thiết để đo lỗi

Đây là vấn đề quan trọng đối với người giám đốc dự án. Để đánh giá lỗi trong quá trình kiểm thử, các lỗi quan trọng được phân loại theo mức độ nghiêm trọng: Rất nghiêm trọng/Nghiêm trọng/Khá nghiêm trọng/Ít nghiêm trọng, và chỉ rõ những người chịu trách nhiệm cho việc đó. Thêm nữa, các lỗi của dự án cũng sẽ được đánh dấu là đã được giải quyết xong (closed) hay vẫn chưa xong (opened).

5.6.2 Kiểm soát thực hiện sửa lỗi (Defect tracking)

- Được thực hiện nhờ sự hỗ trợ của các công cụ như Bugzilla, TestTrackPro, Rational ClearCase. Một số công cụ tốt thường được cung cấp miễn phí hoặc là với chi phí rất nhỏ.
- Đảm bảo tất cả thành viên liên quan đều có quyền truy nhập vào hệ thống kiểm thử.
- Cần tổ chức những buổi họp thường xuyên để xem lại những lỗi đã sửa: thường là hàng tuần trong quá trình kiểm thử thông thường và hàng ngày đối với thời điểm gấp rút.
- Cho phép tất cả nhân viên (kỹ sư đảm bảo chất lượng, kỹ sư lập trình, nhà phân tích, nhà quản lý, và đôi khi là người sử dụng và giám đốc dự án) nhập các lỗi phát hiện được của dự án vào hệ thống kiểm soát theo dõi lỗi của dự án hoặc của chung công ty.

Cấu trúc của hệ thống kiểm soát lỗi thường bao gồm những trường sau đây:

- *Trạng thái:* mở, đóng, đang trì hoãn
- Ngày nhập lỗi vào hệ thống, ngày cập nhật thông tin và ngày đóng lỗi
- Mô tả vấn đề của lỗi được phát hiện
- Số hiệu phiên bản phần mềm mà lỗi xuất hiện
- Người phát hiện ra lỗi
- Thứ tự ưu tiên được giải quyết của lỗi: thấp, trung bình, cao, rất cao
- *Những nhận xét, chú thích* được thực hiện bởi cán bộ đảm bảo chất lượng, kỹ sư lập trình và những thành viên khác liên quan.

5.6.3 Các thông số lỗi

Ta cần quan tâm về những thông số liên quan đến việc kiểm soát lỗi:

- *Tỉ lệ mở* (Open rate): liên quan tới số lỗi xuất hiện mới trong khoảng thời gian nhất định.
- *Tỉ lệ đóng* (Close rate): liên quan tới số lỗi được sửa xong (đóng) trong cùng khoảng thời gian trên.
- *Tỉ lệ thay đổi* (Change rate): số lần cùng một vấn đề được cập nhật
- *Số lần sửa lỗi sai* (Fix Failed Counts): số lỗi mà đã được thực hiện việc sửa nhưng chưa được sửa đúng. Đây cũng là một đơn vị để đo khả năng giao động của dự án (vibration).

Tỉ lệ lỗi trung bình do Microsoft nghiên cứu qua thống kê là 10–20 lỗi/KLOC được phát hiện trong quá trình kiểm thử và 0.5 lỗi/1 KLOC sau khi bàn giao sản phẩm cho khách hàng.

5.6.4 Môi trường kiểm thử

Ta thường chia làm hai môi trường chính để kiểm thử: môi trường phần cứng và phần mềm.

Việc kiểm thử môi trường phần cứng liên quan tới các nhóm kỹ sư lập trình, nhóm kỹ sư đảm bảo chất lượng dự án, nên xây dựng dự án và các sản phẩm.

Môi trường kiểm thử điển hình cho kỹ sư kỹ sư lập trình là cấu hình phần cứng mà tại đó người lập trình phát triển hệ thống và đồng thời thực hiện quá trình kiểm thử các chức năng trên đó.

Môi trường kiểm thử cho những kỹ sư chất lượng hoặc kỹ sư kiểm thử là cấu hình phần cứng cho việc thực hiện kiểm thử tích hợp, kiểm thử hệ thống và kiểm thử lại sau khi sửa chữa.

Môi trường kiểm thử phần cứng còn cần được xác định cho quá trình kiểm tra khả năng chịu tải và quá trình triển khai cuối cùng của hệ thống phần mềm.

5.7 THỰC HÀNH VỚI HỆ THỐNG QUẢN LÝ BUGZILLA

Bugzilla là phần mềm máy chủ cho phép quản lý các lỗi phát sinh trong quá trình

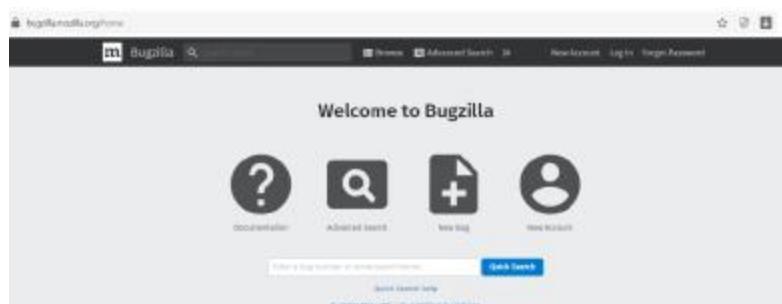
phát triển dự án phần mềm được phát triển bởi tổ chức Mozilla. Các tính năng:

- 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 khai báo lỗi và bàn giao sửa lỗi cho người khác.
- Cho phép quản lý quá trình hoạt động, 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 thành viên tham gia dự án qua chức năng gửi thư.

Link trang chủ: <https://bugzilla.mozilla.org/home>

1. Tạo mới tài khoản

Bước 1: Vào trang Bugzilla



Hình 5.2: Trang chủ Bugzilla

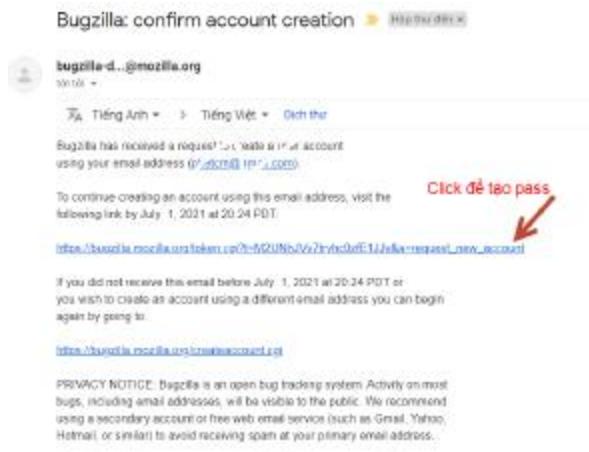
Bước 2: Click "New Account" để tạo Account

The screenshot shows the 'Create an account' page. At the top, there's a navigation bar with links for 'Advanced Search', 'New Account', 'Log in', and 'Forgot Password'. Below the navigation, there's a section with 5 numbered steps. Step 1: 'Please read our Bug Writing Guidelines'. Step 2: 'Bugzilla is a public place. Your comments and other activities on bugs will generally be publicly visible, and your email address will be accessible through public APIs and will be visible to all logged-in users of Bugzilla. Some people use an alternative email address for this reason. See Mozilla's Websites, Communications & Cookies Privacy Notice for more information on our privacy policies.' Step 3: 'When using Bugzilla to submit patches, comments, code, and any other content, you agree to our policies on open source licensing and content submission. See Mozilla's Websites and Communications Terms of Use for more information.' Step 4: 'You understand that your conduct on this site is subject to both Bugzilla Etiquette, and the Mozilla Community Participation Guidelines. By creating an account, you agree to abide by them.' Step 5: 'Please give us an email address you want to use. Once we confirm that it works, you'll be asked to set a password and then you can start filing bugs and helping fix them.' At the bottom left, there's a 'Create an account' button with a red arrow pointing to it. Below the button, there's an input field for 'Email Address' containing 'phucanh.tiger4vn@t11' and a checkbox for accepting terms and conditions. A red arrow also points to this checkbox.

Hình 5.3: Tạo tài khoản trên Bugzilla

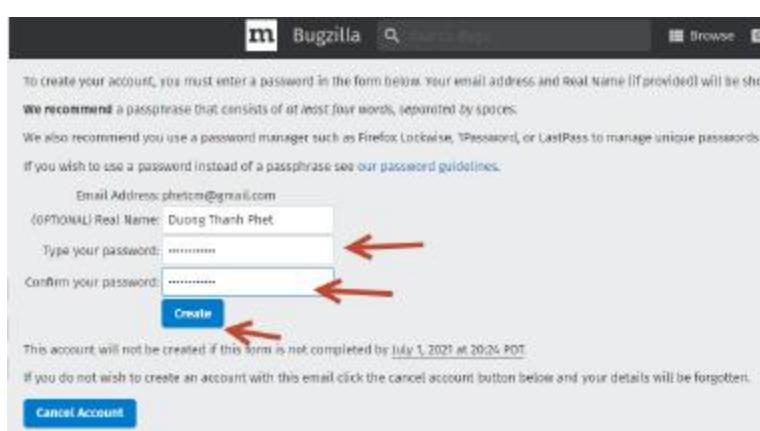
Nhập địa chỉ mail vào, Click “Create Account”. Mail tự gửi đến mail cá nhân

Bước 3: Vào mail cá nhân kích hoạt



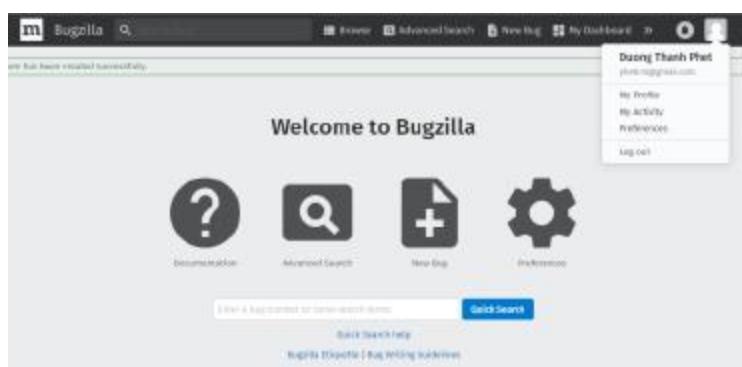
Hình 5.4: Kích hoạt tài khoản

Bước 4: Nhập Email và Pass tạo tài khoản chọn “Create”



Hình 5.5: Tạo mật khẩu

2. Đăng nhập Account



Hình 5.6: Đăng nhập tài khoản

3. Tạo báo cáo lỗi trong Bugzilla

Bước 1: Để tạo một lỗi mới trong Bugzilla, hãy truy cập trang chủ của Bugzilla và nhấp vào tab New từ menu chính



Hình 5.7: Tạo mới báo cáo lỗi

Bước 2: Trong cửa sổ tiếp theo

- Nhập sản phẩm
- Nhập thành phần
- Đưa ra mô tả thành phần
- Chọn phiên bản
- Chọn mức độ nghiêm trọng
- Chọn phần cứng
- Chọn hệ điều hành
- Nhập Tóm tắt
- Nhập mô tả
- Đính kèm tập tin
- Submit

LƯU Ý: Các trường trên sẽ thay đổi tùy theo tùy chỉnh Bugzilla

The screenshot shows a web-based bug reporting form. At the top, there's a navigation bar with links like Home, New, Browse, Search, and a search bar. Below that is a notice about password resets. The main form area has several fields with red circles containing numbers indicating they are required:

- Product:** Sam's Widget (Required)
- Component:** Widget Gears (Required)
- Version:** unspecified (Required)
- Severity:** normal (Required)
- Hardware:** PC (Required)
- OS:** Windows NT (Required)
- Summary:** (Required, highlighted in yellow)
- Description:** (Required)
- Attachment:** Add an attachment
- Submit Bug:**

A tooltip for the 'Component' field says: "(* = Required Field)". A note at the bottom right says: "We've made a guess at your operating system and platform. Check them and make any corrections if necessary."

Hình 5.8: Khai báo các thông tin báo cáo lỗi

LƯU Ý: Các trường bắt buộc được đánh dấu *.

Trong trường hợp này: Tổng kết, Mô tả là bắt buộc. Nếu không điền sẽ nhận được thông báo như bên dưới.

The screenshot shows a bug reporting interface with a validation error message for the 'Summary' field:

You must enter a Summary for this bug.

The 'Summary' field contains: "Gears for sams widget twisted".

The 'Description' field contains: "the widget gears are twisted at the end and not showing correct signal".

A callout box points to the 'Description' field with the text: "Put your description overhere".

Possible Duplicates:	Bug ID	Summary	Status	Action
	2776	when using the Widget Gears, the mV signal unexpectedly goes to 0	CONFIRMED	Add Me to the CC List
	2722	Widget Gears causes wrong mV signal to appear	CONFIRMED	Add Me to the CC List
	12431	Widget Gears cannot start	IN_PROGRESS	Add Me to the CC List
	12480	The Gear of sams widgets failed its validation	CONFIRMED	Add Me to the CC List
	15407	Sams Widget came pipe	CONFIRMED	Add Me to the CC List
	21019	Gears are bound up	CONFIRMED	Add Me to the CC List
	21041	Widget gears are stuck	CONFIRMED	Add Me to the CC List

Hình 5.9: Thông báo lỗi

Bước 3: Bug được tạo ID # 26320 được gán cho Bug. Ta cũng có thể thêm thông tin bổ sung vào lỗi được chỉ định như URL, từ khóa, bảng trắng, thẻ, v.v. Thông tin bổ sung này rất hữu ích để cung cấp thêm chi tiết về Lỗi đã tạo.

- Nhập văn bản
- URL
- Bảng trắng
- Từ khóa

- Thẻ
- Phụ thuộc
- Blocks
- Tài liệu đính kèm

The screenshot shows a bug report for "Bug 26320 - Gears for some widget twisted". A callout box highlights the "Bug ID number is assigned to newly created bug" field. Other visible fields include Product: Sam's Widget, Component: Widget Gears, Version: unspecified, Platform: PC, Windows NT, Importance: P2, and Assigned To: James (edit) (last). A dependency section lists "Depends on" and "Blocks". A calendar dropdown shows "Always/Appears Also Always Appears That Value, Always". The "Deadline" field is highlighted with a red border.

Hình 5.10: Khai báo các thông tin báo cáo lỗi bổ sung

Bước 4: Vẫn ở cửa sổ này, nếu cuộn xuống có thể chọn deadline và trạng thái của lỗi. Deadline trong Bugzilla thường đưa ra giới hạn để giải quyết lỗi trong khung thời gian nhất định.

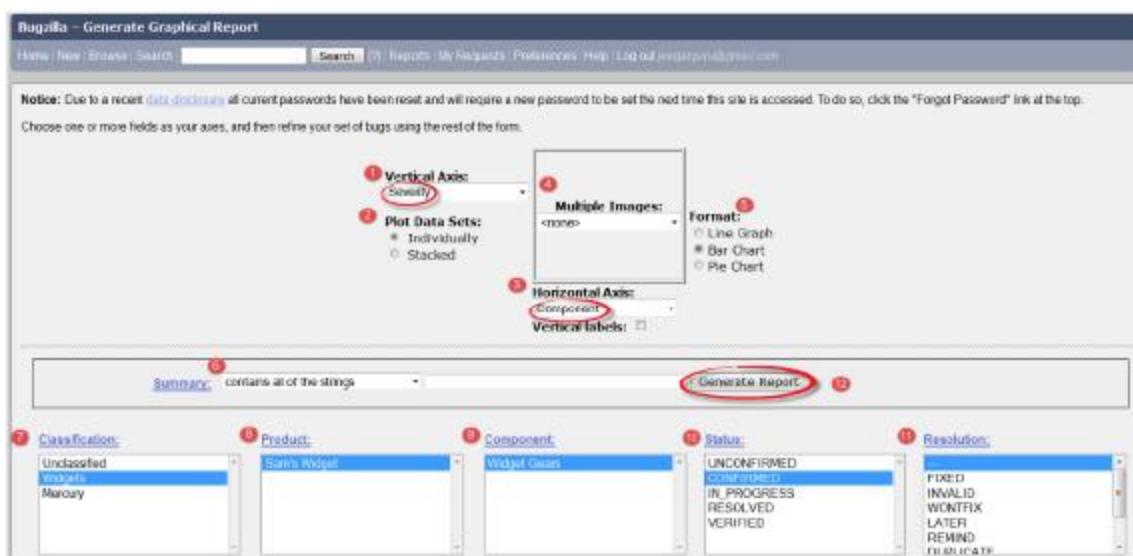
The screenshot shows the same bug report interface. A callout box highlights the "Status" dropdown menu, which includes options like CONFIRMED, IN_PROGRESS, and RESOLVED. Another callout box highlights the "Deadline" field, which is currently empty. A calendar for January 2015 is shown, with the 7th circled in red. A large red oval surrounds the "Save Changes" button at the bottom right.

Hình 5.11: Khai báo các thông tin báo cáo lỗi bổ sung thời gian

4. Tạo biểu đồ báo cáo

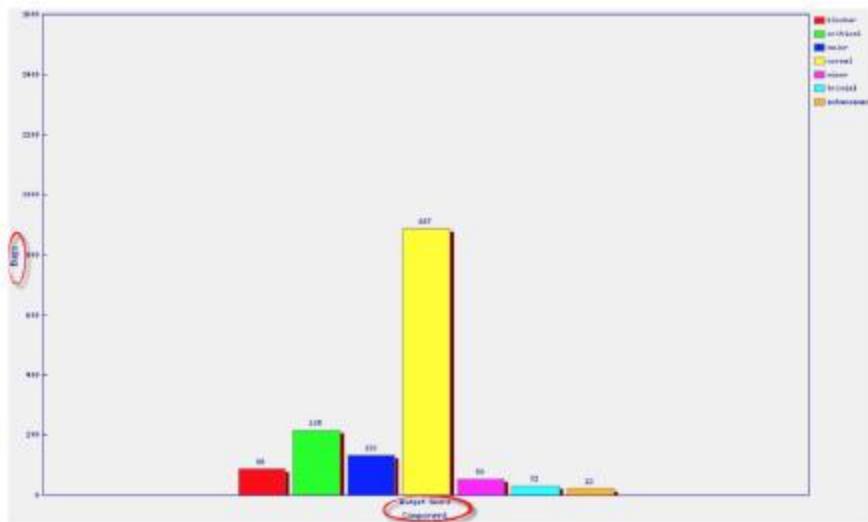
Báo cáo biểu đồ là một cách để xem trạng thái hiện tại của lỗi. Ta có thể chạy các báo cáo thông qua bảng HTML hoặc biểu đồ cột / tròn. Ý tưởng đằng sau biểu đồ báo cáo trong Bugzilla là xác định một tập hợp các lỗi bằng giao diện tìm kiếm tiêu chuẩn và sau đó chọn một số khía cạnh của tập hợp đó để vẽ trên trực ngang và trực dọc. Ta cũng có thể nhận được báo cáo 3 chiều bằng cách chọn tùy chọn "Multiple Pages".

Các báo cáo rất hữu ích theo nhiều cách, ví dụ, nếu muốn biết thành phần nào có số lượng lỗi lớn nhất được báo cáo. Để hiển thị điều đó trong biểu đồ, ta có thể chọn mức độ nghiêm trọng trên trục X và thành phần trên trục Y, sau đó nhấp vào để tạo báo cáo. Nó sẽ tạo ra một báo cáo với thông tin quan trọng.



Hình 5.12: Báo cáo biểu đồ

Biểu đồ bên dưới là biểu đồ cột thể hiện mức độ nghiêm trọng của bug trong thành phần "Widget Gears". Ở biểu đồ bên dưới, các lỗi nghiêm trọng nhất (màu đỏ) có số lượng là 88 trong khi các lỗi có mức độ nghiêm trọng bình thường (màu vàng) là 667 lỗi.

**Hình 5.13: Biểu đồ báo cáo lỗi**

Tương tự như vậy, chúng ta sẽ tạo biểu đồ đường, thể hiện quan hệ giữa % hoàn thành và deadline

Bước 1: Xem báo cáo.

Nhấp vào Reports từ Menu chính

Nhấp vào Graphical reports

Current State

- [Search](#) - list sets of bugs.
- [Tabular reports](#) - tables of bug counts in 1, 2 or 3 dimensions, as HTML or CSV.
- ② • [Graphical reports](#) - line graphs, bar and pie charts.
- [Duplicates](#) - list of most frequently reported bugs.

Change Over Time

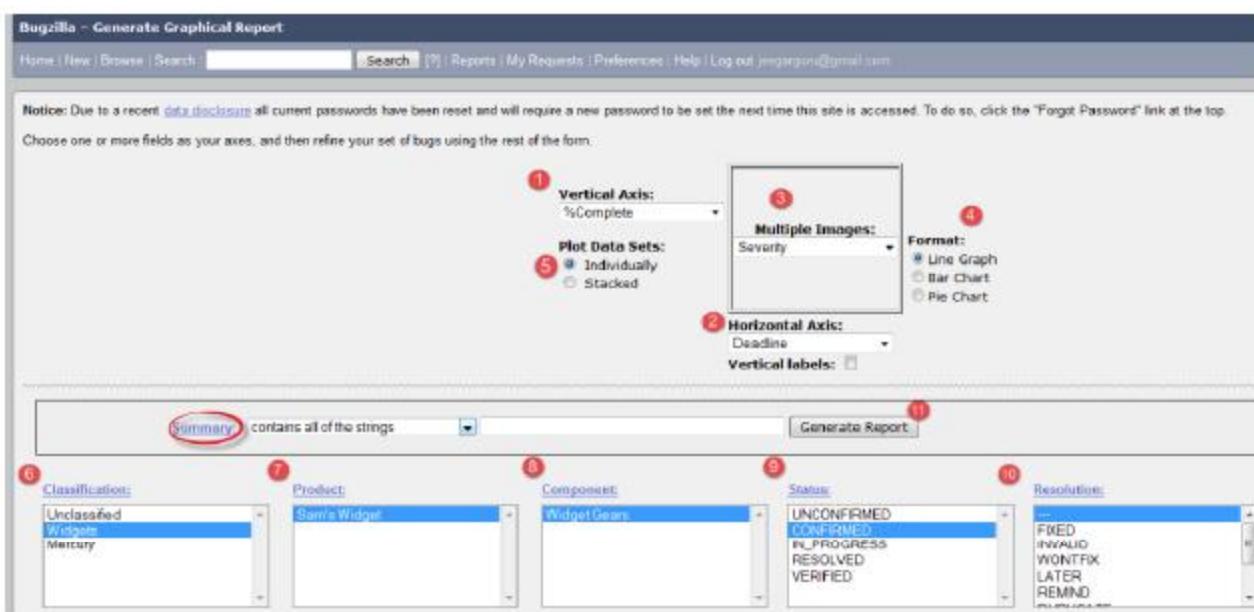
- [Old Charts](#) - plot the status and/or resolution of bugs against time, for each product in your database.
- [New Charts](#) - plot any arbitrary search against time. Far more powerful.

Hình 5.14: Xem báo cáo.

Bước 2: Hãy tạo một biểu đồ % Hoàn thành Vs Deadline

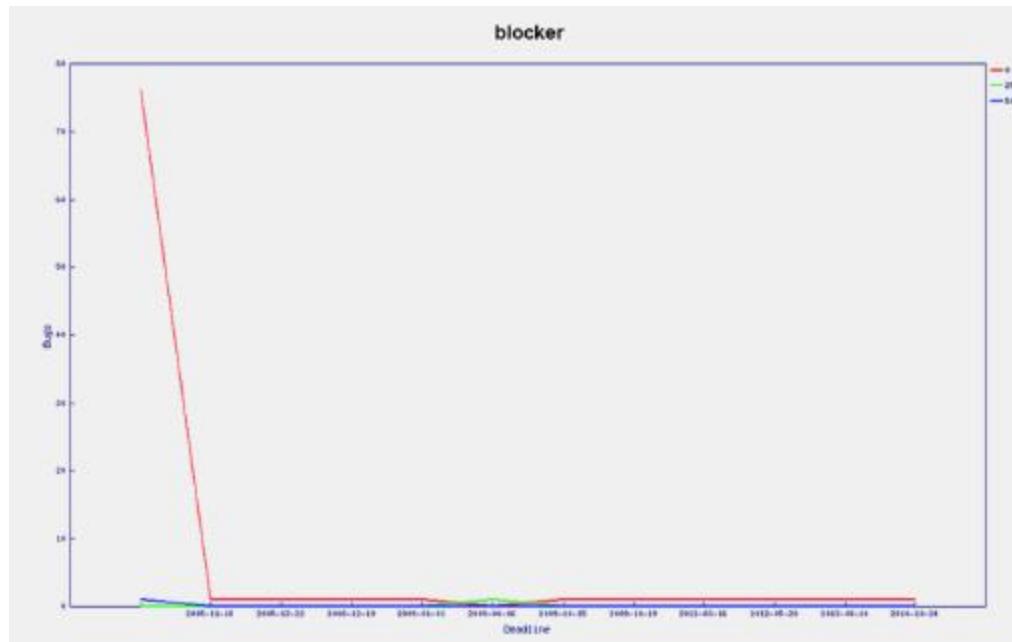
Ở đây trên trục tung chọn % hoàn thành và trên trục hoành chọn deadline . Điều này sẽ đưa ra biểu đồ về số lượng công việc được thực hiện theo tỷ lệ phần trăm so với thời hạn đã đặt. Bây giờ, đặt tùy chọn khác nhau để trình bày báo cáo bằng biểu đồ

- Trục tung
- Trục hoành
- Nhiều hình ảnh
- Loại biểu đồ: biểu đồ đường, biểu đồ cột hoặc biểu đồ hình tròn
- Lô dữ liệu
- Phân loại lỗi của bạn
- Phân loại sản phẩm của bạn
- Phân loại thành phần của bạn
- Phân loại trạng thái lỗi
- Chọn độ phân giải
- Bấm vào để tạo một báo cáo



Hình 5.15: Tạo một biểu đồ %

Hình ảnh của biểu đồ sẽ xuất hiện như thế này



Hình 5.16: Biểu đồ được tạo

5. Chức năng Browse

Bước 1: Để xác định vị trí lỗi của bạn, chúng tôi sử dụng chức năng Browse, nhấp vào nút Browse từ menu chính.



Hình 5.17: Chức năng Browse

Bước 2: Ngay sau khi nhấp vào nút browse, một cửa sổ sẽ mở ra thông báo "**Chọn danh mục sản phẩm cần duyệt**" như hình, chúng ta duyệt lỗi theo danh mục.

Sau khi nhấp vào nút browse

Chọn sản phẩm "Sam's Widget" vì như vậy đã tạo ra một bug bên trong nó



Hình 5.18: Chọn sản phẩm "Sam's Widget"

Bước 3: Một cửa sổ khác hiện ra, click vào "**widget gears**" . Thành phần Bugzilla là các phần phụ của sản phẩm. Chẳng hạn, trong đó sản phẩm là WIDGET SAM có thành phần là WIDGET GEARS .



Hình 5.19: Thành phần là WIDGET GEARS

Bước 4: Khi bạn bấm vào từng thành phần, nó sẽ mở một cửa sổ khác. Tất cả các lỗi được tạo ra theo loại cụ thể sẽ được liệt kê ở đây. Từ danh sách lỗi đó, chọn ID # bug để xem thêm chi tiết về lỗi.

ID	Product	Component	Assignee	Status	Resolution	Summary	Changed
<i>This result was limited to 500 bugs. See all search results for this query.</i>							
1256	Sam's WI	Widget G	justdave@syndicomm.com	CONF	---	summary	2014-10-23
4719	Sam's WI	Widget G	justdave@syndicomm.com	CONF	---	standard	2012-06-13
4742	Sam's	Click on Bug ID	stdeville@syndicomm.com	CONF	---	Just a test	2009-01-26
5509	Sam's	number to see the details	stdave@gavinsharp.com	CONF	---	Test Bug	2014-05-04
2506	Sam's	details	ndfill@gavinsharp.com	CONF	---	test	2009-11-04
9302	Sam's WI	Widget G	mexxi4@gmail.com	CONF	---	Want true	2010-09-21
3010	Sam's WI	Widget G	mickesnow@yahoo.com.mx	CONF	---	bugfix	2012-03-15
24741	Sam's WI	Widget G	neha.malk02@gmail.com	CONF	---	cancel button not working	2014-10-10

Hình 5.20: Xem chi tiết lỗi

Sẽ mở một cửa sổ khác, nơi thông tin về lỗi, có thể được nhìn thấy chi tiết hơn. Trong cùng một cửa sổ, cũng có thể thay đổi người được assignee, liên hệ QA hoặc danh sách CC.

Bug 1256 - summary [edit]

Status: CONFIRMED (edit)

Product: Sam's Widget

Component: Widget Gears

Version: unspecified

Hardware: HP

Importance: P1 - normal

Target Milestone: ---

Assigned To: Dave Miller (take) (take)

QA Contact: (edit) (take)

Reported: 2003-05-27 13:28 PDT by Jason McCallum

Modified: 2014-10-23 08:09 PDT (history)

CC List: Add me to CC list
6 users (edit)

Save Changes

Hình 5.21: Xem chi tiết lỗi nhiều hơn

Ngoài ra trong Bugzilla còn có một số chức năng hữu ích khác.

Nguồn: <https://www.guru99.com/bugzilla-tutorial-for-beginners.html#3>

TÓM TẮT

Bài học này cung cấp các kiến thức về:

- *Tìm lỗi*
- *Các dạng lỗi thường gặp và nguyên nhân gây ra lỗi*
- *Giới thiệu lỗi*
- *Hệ thống quản lý bug*

BÀI 6: KIỂM THỬ TỰ ĐỘNG VÀ CÔNG CỤ

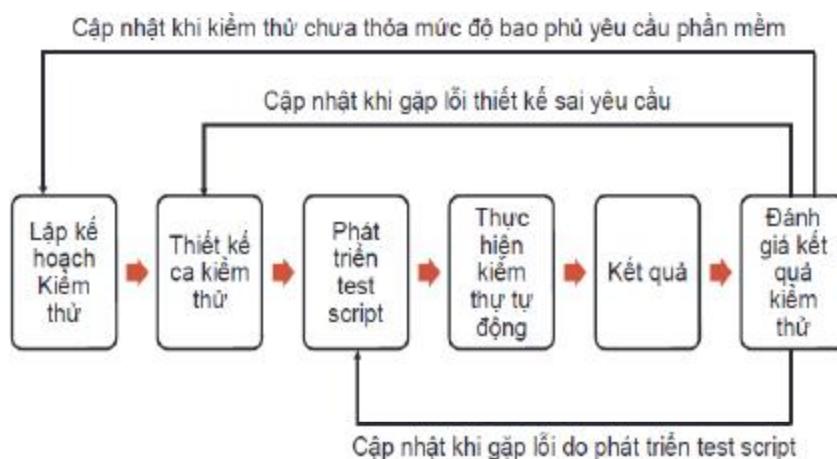
Nội dung gồm các phần sau:

- Giới thiệu kiểm thử tự động (*Automation Software testing*)
- Giới thiệu kiểm thử dựa tên hành động (*Action-based testing*)
- Công cụ kiểm thử tự động *Katolon Studio*
- Công cụ kiểm thử tự động *Selenium*

6.1 GIỚI THIỆU VỀ AUTOMATION SOFTWARE TESTING

- Kiểm thử tự động: áp dụng công cụ giúp thực hiện kiểm thử phần mềm.
- Nên sử dụng công cụ tự động khi:
 - Không đủ tài nguyên
 - Kiểm thử hồi quy
 - Kiểm tra khả năng vận hành của phần mềm trong môi trường đặc biệt.
- Test script: nhóm mã lệnh đặc tả kịch bản để tự động hóa trình tự kiểm thử.

6.1.1 Quy trình kiểm thử tự động



Hình 6.1: Quy trình kiểm thử

- Tạo test script:* Giai đoạn này ta dùng test tool để ghi lại các thao tác lén PM cần kiểm tra và tự động sinh ra test script
- Chỉnh sửa test script:* chỉnh sửa lại test script thực hiện kiểm tra theo đúng yêu cầu đặt ra, cụ thể là làm theo test case cần thực hiện
- Chạy test script để kiểm thử tự động:* Giám sát hoạt động kiểm tra phần mềm của test script
- 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

6.1.2 Ưu và nhược điểm kiểm thử tự động

- **Ưu điểm:**
 - Kiểm thử phần mềm không cần can thiệp của tester
 - Giảm chi phí thực hiện kiểm tra số lượng lớn các test case hoặc test case lặp lại nhiều lần
 - Giải lập tình huống khó có thể thực hiện bằng tay
- **Nhược điểm:**
 - Mất chi phí tạo các script để thực hiện kiểm thử tự động
 - Tốn chi phí dành cho bảo trì các script

- Đòi hỏi tester phải có kỹ năng tạo và thay đổi script cho phù hợp testcase
- Không áp dụng tìm được các lỗi mới cho phần mềm

6.1.3 Phân loại công cụ kiểm thử

Mỗi công cụ kiểm thử chỉ có thể hỗ trợ khía cạnh nào đó của thử nghiệm. Các công cụ có thể được phân loại dựa trên một số tiêu chí như mục đích, thương mại/miễn phí/ mã nguồn mở/phần mềm chia sẻ, công nghệ, hoạt động của mỗi công cụ. Phân loại các công cụ kiểm thử theo mục đích của mỗi loại công cụ kiểm thử. Theo đó, ta có thể phân loại các công cụ kiểm thử tự động thành: Unit Testing Tools, Regression Testing Tools, Functional Testing Tools, Performance Testing Tools.

Công cụ kiểm thử đơn vị - Unit Testing Tools

Kiểm thử đơn vị - *Unit Testing* là cách tiếp cận kiểm tra các đơn vị cá nhân của mã nguồn và kiểm tra nếu phù hợp với mục đích. Kiểm thử đơn vị cho phép lập trình viên sửa đổi và duy trì mã nguồn và đảm bảo 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.

Các công cụ Unit Testing: JUnit và NUnit, và xem thêm phục lục.

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 assertEquals():** 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);
}
```

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.

Công cụ 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 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.

Các công cụ kiểm thử hiệu năng (Performance Testing): xem phụ lục

Các công cụ kiểm thử chức năng (Functional Testing): xem phụ lục

6.2 GIỚI THIỆU VỀ ACTION-BASED TESTING

Kiểm thử dựa trên hành động (Action-Based Testing - ABT) cung cấp một khung làm việc mạnh mẽ để tổ chức thiết kế kiểm thử, tự động hóa và thực hiện xung quanh các từ khóa. Trong các từ khóa ABT được gọi là "hành động" để làm cho khái niệm này hoàn toàn rõ ràng. Hành động là các nhiệm vụ được thực hiện trong một thử

nghiệm. Thay vì tự động hóa toàn bộ kiểm thử dưới dạng một tập lệnh dài, kỹ sư tự động hóa có thể tập trung vào việc tự động hóa các hành động như các khối xây dựng riêng lẻ có thể được kết hợp theo bất kỳ thứ tự nào để thiết kế kiểm thử. Sau đó, các kỹ sư kiểm tra phi kỹ thuật và các nhà phân tích kinh doanh có thể xác định các kiểm thử của họ dưới dạng một loạt các từ khóa tự động này, tập trung vào kiểm thử thay vì ngôn ngữ kịch bản.

Thiết kế kiểm thử truyền thống bắt đầu với một câu chuyện bằng văn bản phải được giải thích bởi mỗi người kiểm tra hoặc kỹ sư kiểm thử tự động. Thiết kế kiểm thử ABT diễn ra trong một bảng tính, với các hành động được liệt kê trong một chuỗi rõ ràng, được tổ chức tốt. Các hành động, dữ liệu kiểm thử và bất kỳ thông tin giao diện GUI cần thiết nào được lưu trữ trong các bảng tính riêng biệt, nơi chúng có thể được tham chiếu bởi mô-đun kiểm thử chính. Các kiểm thử sau đó được thực hiện ngay trong bảng tính, sử dụng các công cụ tạo tập lệnh của bên thứ ba hoặc tự động hóa được xây dựng trong TestArchitect.

Để nhận ra toàn bộ sức mạnh của kiểm thử dựa trên hành động, điều quan trọng là sử dụng các hành động cấp cao bắt cứ khi nào có thể trong thiết kế kiểm thử. Hành động cấp cao có thể hiểu được bởi những người quen thuộc với logic nghiệp vụ của kiểm thử. Ví dụ: khi người dùng nhập số, hệ thống sẽ tính toán thế chấp hoặc kết nối với điện thoại. Hành động cấp cao tốt có thể không cụ thể cho hệ thống đang được kiểm tra. "Nhập lệnh" là một bước cao cấp tốt có thể được sử dụng một cách tổng quát để chỉ các bước cụ thể ở mức độ thấp diễn ra trong nhiều kiểm thử của nhiều ứng dụng khác nhau.

Tự động hóa sau đó được hoàn thành thông qua kịch bản (lập trình) của các hành động cấp thấp. TestArchitect cung cấp một tập hợp đầy đủ các hành động cấp thấp cần thiết thông qua tính năng tự động cài sẵn. Trong trường hợp đó, việc tạo hành động cấp cao theo yêu cầu thiết kế kiểm thử sẽ chỉ liên quan đến việc kéo và thả một số hành động cấp thấp để tạo hành động cấp cao. Các hành động cấp thấp phía sau "nhập lệnh" sẽ là các bước cụ thể cần thiết để hoàn thành hành động đó thông qua các giao diện khác nhau như html, GUI của Windows, vv Ví dụ về hành động cấp thấp sẽ là "nút ấn".

Bất cứ khi nào kịch bản của một kỹ sư tự động hóa được yêu cầu, việc chia nhỏ công việc này thành các hành động cấp thấp có thể tái sử dụng giúp tiết kiệm thời gian và tiền bạc bằng cách thực hiện các thay đổi trong tương lai không cần thiết ngay cả khi phần mềm đang được kiểm thử trải qua các phiên bản chính. Việc thay đổi hành động thường là tất cả những gì cần thiết. Nếu nhiều kịch bản là cần thiết, nó chỉ liên quan đến việc viết lại các hành động cá nhân thay vì sửa đổi toàn bộ các kịch bản tự động hóa và kết quả tích lũy của một thư viện rộng lớn của tự động hóa cũ.

Kiểm thử dựa trên hành động cho phép các nhóm kiểm thử tạo ra một khuôn khổ tự động hóa kiểm thử hiệu quả hơn, khắc phục hạn chế của phương pháp khác: Tham gia đầy đủ của Nhóm kiểm thử Tự động hóa kiểm tra hầu hết các nhóm kiểm thử bao gồm chủ yếu là những người có kiến thức mạnh mẽ về ứng dụng được kiểm thử hoặc miễn doanh nghiệp, nhưng có chuyên môn về lập trình nhẹ. Các thành viên trong nhóm hoàn thành vai trò của kỹ sư tự động hóa kiểm thử thường là những người có nền tảng phát triển phần mềm hoặc khoa học máy tính nhưng thiếu chuyên môn về kiểm tra các nguyên tắc cơ bản, phần mềm được kiểm tra hoặc miễn doanh nghiệp. Kiểm thử dựa trên hành động cho phép cả hai loại thành viên nhóm đóng góp vào nỗ lực tự động hóa thử nghiệm bằng cách cho phép mỗi người tận dụng các kỹ năng đặc đáo của họ để tạo các thử nghiệm tự động hiệu quả. Người kiểm thử xác định các thử nghiệm dưới dạng một loạt các hành động cấp cao có thể tái sử dụng. Sau đó, nhiệm vụ của kỹ sư tự động hóa là xác định cách tự động hóa các hành động cấp thấp cần thiết và kết hợp chúng để tạo ra các hành động cấp cao cần thiết, cả hai đều có thể được sử dụng lại trong nhiều kiểm thử trong tương lai. Cách tiếp cận này cho phép người thử nghiệm tập trung vào việc tạo ra các bài kiểm tra tốt, trong khi các kỹ sư tự động hóa tập trung vào thách thức kỹ thuật của việc thực hiện các hành động.

Giảm thiểu đáng kể khả năng tự động hóa kiểm thử

Nhiều tổ chức xây dựng một bộ kiểm thử tự động sử dụng đáng kể các phương pháp tự động hóa cũ và bắt đầu thấy một số lợi ích, chỉ đến khi gặp khó khăn với nỗ lực bảo trì rất lớn khi ứng dụng thay đổi. Gánh nặng bảo trì là do thực tế các kiểm tra tự động phụ thuộc rất nhiều vào giao diện người dùng của ứng dụng được kiểm thử; khi giao diện người dùng thay đổi, vì vậy phải tự động hóa kiểm thử. Nó thường là trường hợp các quy trình nghiệp vụ cốt lõi được xử lý bởi một ứng dụng sẽ không thay

đổi, mà là giao diện người dùng được sử dụng để ban hành các thay đổi quy trình nghiệp vụ đó.

Kiểm thử dựa trên hành động làm giảm đáng kể gánh nặng bảo trì bằng cách cho phép người dùng xác định các kiểm thử của họ ở cấp quy trình nghiệp vụ. Thay vì xác định các kiểm thử dưới dạng một loạt tương tác với giao diện người dùng, các nhà thiết kế thử nghiệm có thể xác định các thử nghiệm dưới dạng một loạt các hành động kinh doanh. Ví dụ: kiểm tra ứng dụng ngân hàng có thể chứa các hành động 'mở tài khoản mới', 'ký quỹ' và 'rút tiền'. Ngay cả khi giao diện người dùng cơ bản thay đổi, các quy trình kinh doanh này vẫn sẽ vẫn như cũ, do đó, trình thiết kế kiểm thử không cần cập nhật kiểm thử. Nó sẽ là công việc của kỹ sư tự động hóa để cập nhật các hành động bị ảnh hưởng bởi các thay đổi giao diện người dùng và bản cập nhật này thường chỉ cần được thực hiện ở một nơi duy nhất.

Cải thiện chất lượng của các kiểm thử tự động

Trong kiểm thử dựa trên hành động, các nhà thiết kế kiểm thử thực hiện theo cách tiếp cận từ trên xuống để đảm bảo rằng có một mục đích được nêu rõ ràng cho mọi thử nghiệm. Bước đầu tiên là xác định xem nỗ lực tự động hóa kiểm thử tổng thể sẽ được chia thành các mô-đun kiểm thử riêng lẻ như thế nào. Một số cách kiểm tra nhóm phổ biến bao gồm:

- Các khu chức năng khác nhau của ứng dụng.
- Các loại kiểm thử khác nhau (tích cực, tiêu cực, dựa trên yêu cầu, kết thúc đền cuối, dựa trên kịch bản, v.v.).
- Các thuộc tính chất lượng khác nhau đang được kiểm thử (quy trình nghiệp vụ, tính nhất quán UI, hiệu suất, v.v.)

Một khi các yêu cầu kiểm tra được xác định, chúng phục vụ như là một lộ trình để phát triển các trường hợp kiểm thử trong mô-đun và tài liệu cho mục đích của các bài kiểm tra. Mỗi trường hợp kiểm thử được liên kết với một hoặc nhiều yêu cầu kiểm tra, và mỗi yêu cầu kiểm tra phải được giải quyết bởi một hoặc nhiều trường hợp kiểm thử. Bằng cách nêu rõ các yêu cầu kiểm tra, có thể dễ dàng xác định mục đích của kiểm thử và để xác định xem kiểm thử có đáp ứng đủ các yêu cầu kiểm tra đó không. Các nhà phát triển kiểm thử chính xác và súc tích trong quá trình tạo kiểm thử, tạo ra

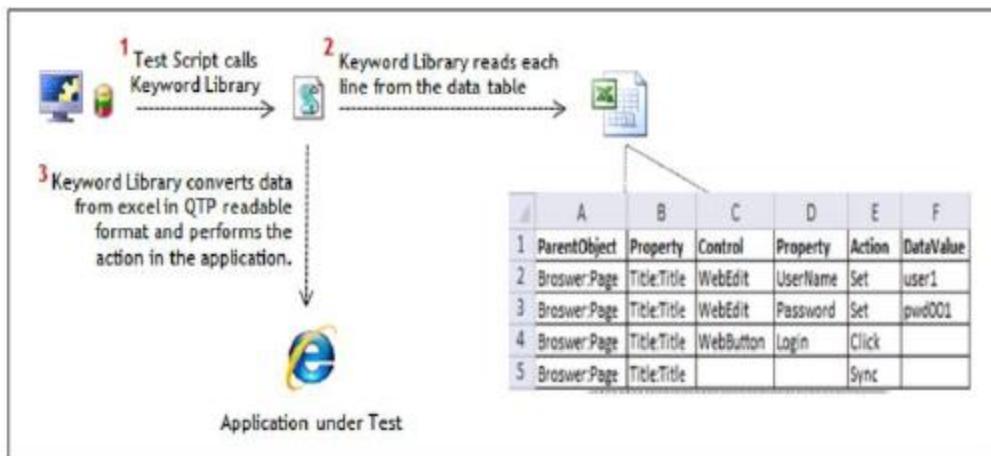
đủ các bài kiểm tra để đáp ứng các yêu cầu đã nêu mà không đưa ra dự phòng không mong muốn.

Sau khi xác định rõ ràng các yêu cầu kiểm tra, các nhà thiết kế kiểm thử có thể bắt đầu triển khai các trường hợp kiểm thử bằng cách sử dụng các hành động được xác định trước hoặc bằng cách xác định các hành động mới. Các nhà thiết kế kiểm thử có thể xác định các kiểm thử của họ là các quy trình nghiệp vụ cấp cao, cho phép các bài kiểm tra dễ đọc hơn các bài kiểm tra được xác định bằng cách sử dụng các tương tác giao diện cấp thấp.

Kiểm thử dựa trên hành động cung cấp một khuôn khổ tích hợp toàn bộ tổ chức kiểm thử hỗ trợ tự động kiểm tra hiệu quả. Các nhà phân tích kinh doanh, kiểm thử các loại, kỹ sư tự động hóa, kiểm tra khách hàng tiềm năng và người quản lý QA đều làm việc trong khuôn khổ để hoàn thành kế hoạch kiểm thử, thiết kế kiểm thử, tự động kiểm tra và thực hiện kiểm tra.

Kiểm thử dựa trên hành động cung cấp một khuôn khổ đã được chứng minh để tổ chức kiểm tra và kiểm tra thư viện tự động hóa với cấu trúc rõ ràng, ngăn chặn sự gián đoạn có thể gây ra bởi chênh lệch múi giờ và múi giờ.

TestArchitect, một công cụ dựa trên ABT, đưa nó lên cấp độ tiếp theo bằng cách cho phép chia sẻ từ xa các kho lưu trữ cơ sở dữ liệu của các mô-đun kiểm thử, các hành động và các thành phần khác, và cung cấp điều khiển rõ ràng và báo cáo cho người quản lý truy cập, thay đổi và kết quả.



Hình 6.2: Keyword driven Framework

6.3 KIỂM THỬ TỰ ĐỘNG VỚI KATALON STUDIO

6.3.1 Giới thiệu

Katalon Studio là một bộ công cụ toàn diện cho kiểm thử tự động hóa ứng dụng trên web và điện thoại di động. Công cụ này bao gồm một gói đầy đủ các tính năng mạnh mẽ giúp vượt qua những thách thức phổ biến trong tự động hóa thử nghiệm giao diện web, ví dụ như pop-up, iFrame và wait-time. Giải pháp thân thiện và linh hoạt này giúp tester thực hiện công tác kiểm tra tốt hơn, làm việc nhanh hơn và khởi chạy phần mềm chất lượng cao nhờ vào sự thông minh mà nó cung cấp cho toàn bộ quá trình tự động hóa kiểm thử.

6.3.2 Các tính năng chính của Katalon Studio

- a. Simple deployment:** Một gói triển khai duy nhất, gắn kết chứa mọi thứ bạn cần để triển khai một công cụ kiểm tra tự động mạnh mẽ.
- b. Quick & easy set-up:** Không chỉ cung cấp sự cài đặt đơn giản, Katalon Studio cũng giúp bạn dễ dàng thiết lập môi trường. Tester có thể chạy test script đầu tiên của họ khá nhanh bằng cách sử dụng mẫu được xây dựng trước và các test scripts, chẳng hạn như object repositories và keyword libraries.
- c. Faster & Better results:** Tích hợp sẵn mẫu với hướng dẫn rõ ràng giúp tester nhanh chóng xây dựng và chạy các test scripts tự động hóa. Họ có thể thực hiện từng bước với tốc độ và hiệu quả, từ thiết lập dự án, tạo ra thử nghiệm, thực hiện, tạo báo cáo và bảo trì.
- d. Flexible modes:** Một tester mới có thể sử dụng recording và keywords để xây dựng các bài kiểm tra tự động hóa, trong khi các chuyên gia kiểm tra có một IDE hoàn chỉnh để xây dựng các kịch bản nâng cao.
- e. Ease of use:** Nó không thể được dễ dàng hơn, ngay cả hướng dẫn sử dụng với kinh nghiệm lập trình tối thiểu cũng có thể khai thác lợi ích của nó một cách dễ dàng.
- f. Cross-browser application:** Katalon Studio hỗ trợ nhiều nền tảng: Windows 32 và 64 (7, 8 và 10) và OS X 10.5+.

6.3.3 Cài đặt Katalon Studio

Tạo một tài khoản Katalon Account tại địa chỉ: <https://www.katalon.com/sign-up>

Create a free account

Use your business email to receive exclusive features.

Full name

Business email

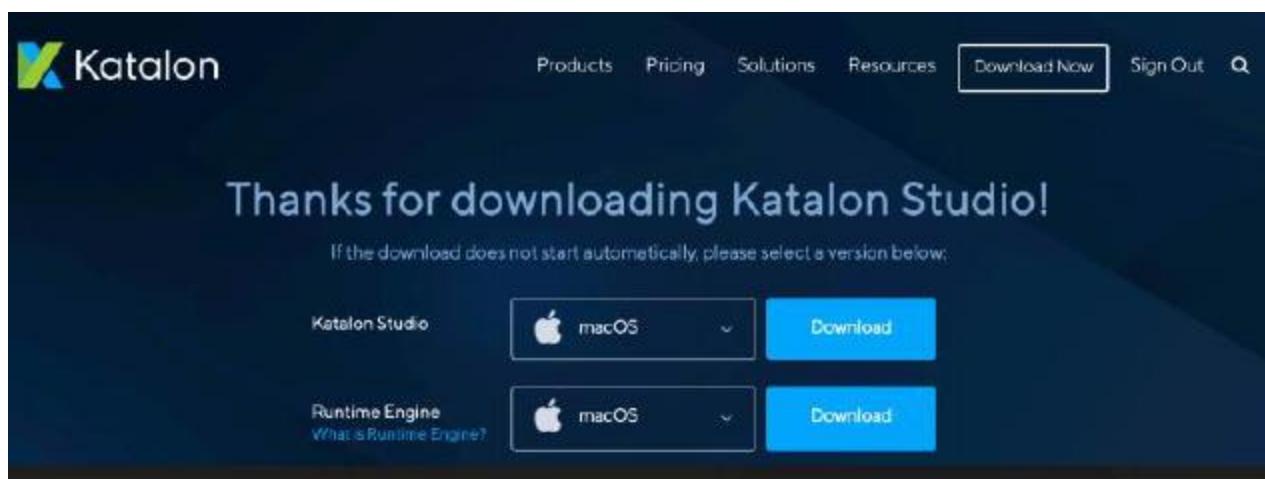
Password

By clicking **Get Started!**, you agree to [Katalon Terms](#) and [Privacy Policy](#).

[Get Started!](#)

Hình 6.3: Giao diện tạo tài khoản

Sau khi tạo xong tài khoản và đăng nhập, click vào **Download Now** để tải về máy. Tại đây, sẽ có hai phiên bản khác nhau được phân biệt như sau:



Hình 6.4: Download phiên bản Katalon Studio

- **Katalon Studio (KS)**: Được sử dụng như một IDE để viết test case, scripting

- **Katalon Runtime Engine (KRE)**: Hỗ trợ chạy test trong chế độ Command Line Interface (CLI)

Lưu ý: Nếu bạn dùng tài khoản email đăng ký là business email (không phải email miễn phí như Gmail, Yahoo....) thì sẽ được 30 ngày trial enterprise verison, hết 30 ngày bạn sẽ tự động convert sang dạng miễn phí.

Sau download về máy tiến hành giải nén và mở ứng dụng trên máy tính của bạn. Để khởi động Katalon Studio, nhấp đúp vào katalon.exe

	config	7/21/2015 11:26 AM	File folder
	configuration	7/21/2015 11:25 AM	File folder
	features	7/21/2015 11:23 AM	File folder
	p2	7/21/2015 11:23 AM	File folder
	plugins	7/21/2015 11:25 AM	File folder
	artifacts	7/20/2015 3:30 PM	XML Document 51 KB
	eclipsec	7/20/2015 3:29 PM	Application 18 KB
	katalon	7/20/2015 3:29 PM	Application 306 KB
	katalon	7/20/2015 3:30 PM	Configuration sett... 1 KB

Hình 6.5: Thư mục chương trình Katalon Studio

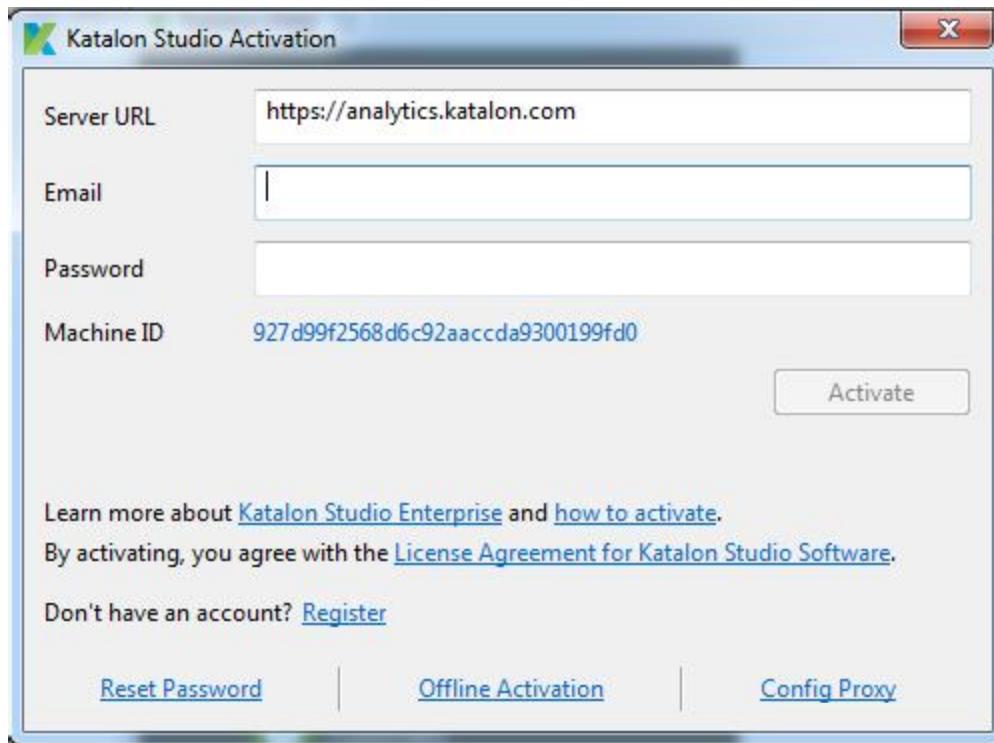
Màn hình ứng dụng bắt đầu như sau:



Hình 6.6: Màn hình bắt đầu cài đặt Katalon Studio

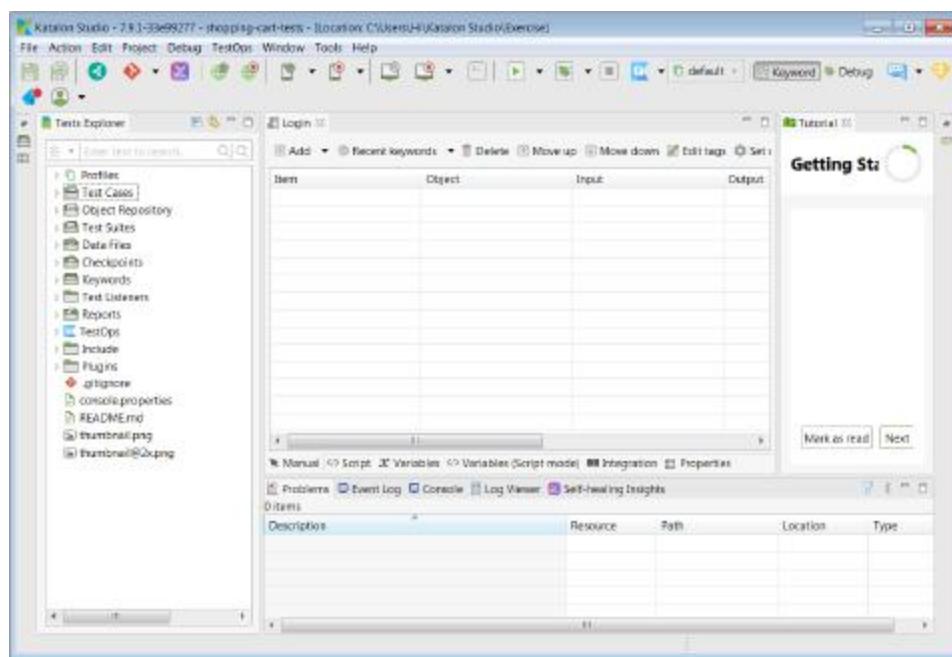
Ở lần hiển thị đầu tiên, cửa sổ kích hoạt Studio Katalon: nhằm mục đích kiểm tra tài khoản mà bạn đang sử dụng là phiên bản bạn sử dụng là phiên bản enterprise,

free hay đang trial. Nhập email và mật khẩu đã đăng ký cho tài khoản Katalon của bạn, sau đó nhấp vào nút **Activate**



Hình 6.7: Màn hình Active chương trình

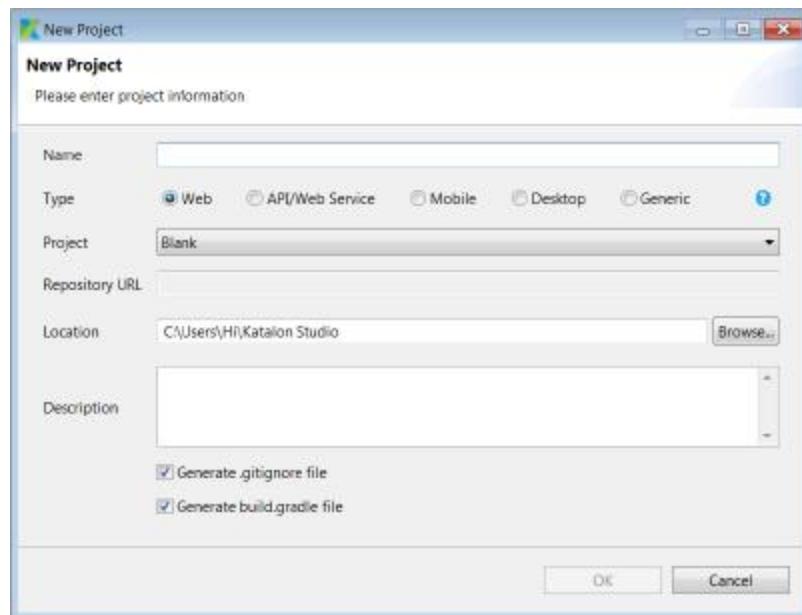
Sau khi đăng nhập vào được giao diện chính:



Hình 6.8: Cửa sổ làm việc của Katalon Studio

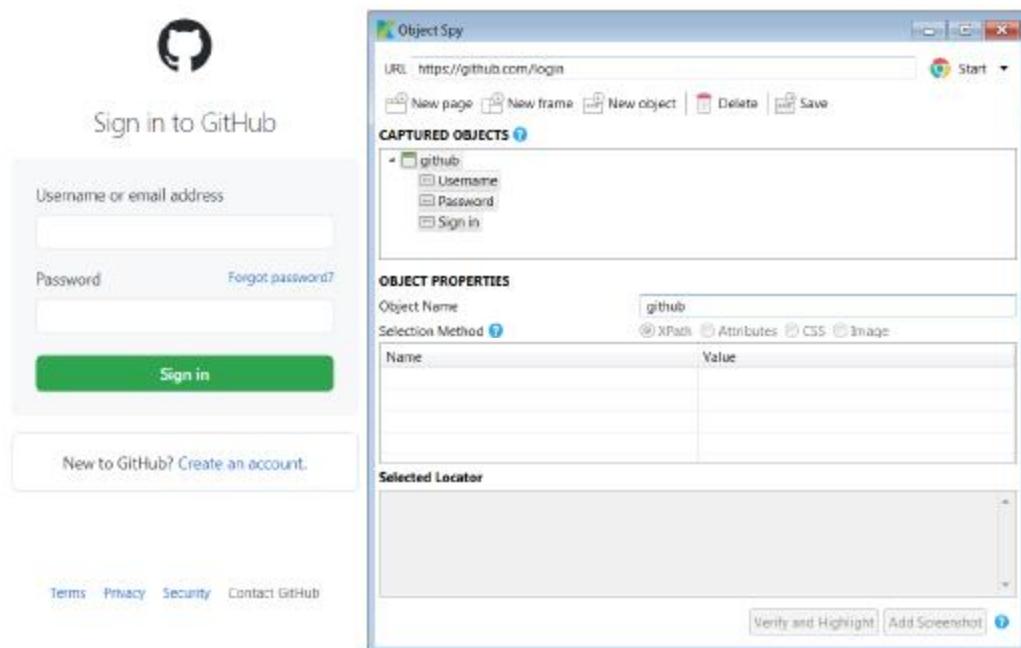
6.3.4 Viết kịch bản test với Katalon Studio

Tạo một Project mới trong Katalon Studio: **File > New > Project**



Hình 6.9: Cửa sổ tạo Project mới trong Katalon Studio

Chọn biểu tượng **Spy web** (giao diện chính) để chụp đối tượng hay còn gọi là lấy ID của đối tượng:

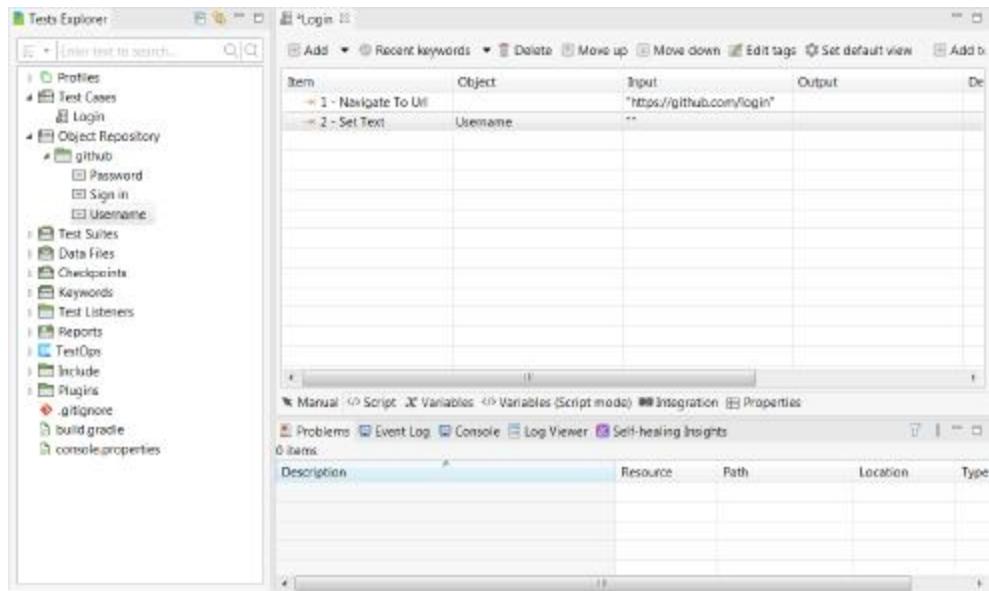


Hình 6.10: Cửa sổ Spy Web dùng để lấy ID của Object

Katalon Studio hỗ trợ người dùng 2 chế độ để thiết lập kịch bản test: **Manual view** và **Script view**

Manual view

Ở chế độ manual view này, chỉ cần click vào các đối tượng trong mục Object Repository để thực hiện kéo thả các ID vào mục Object

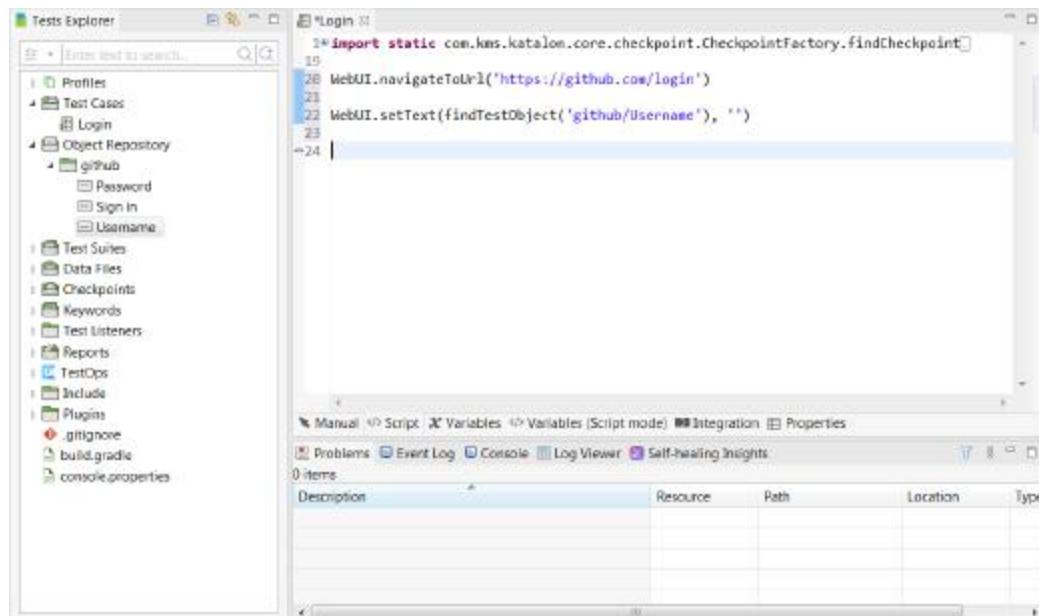


Hình 6.11: Cửa sổ làm việc ở chế độ Manual view

Các điều kiện Input, Output là tham số truyền vào theo yêu cầu của Object.

Script view

Ở chế độ này, Katalon Studio cho phép nhập các câu lệnh để thực hiện chạy kịch bản test.



Hình 6.12: Cửa sổ làm việc ở chế độ Script view

6.4 KIỂM THỬ TỰ ĐỘNG VỚI SELENIUM

6.4.1 Giới thiệu

- Selenium là bộ kiểm thử tự động miễn phí (mã nguồn mở) tự động dành cho các ứng dụng web trên các trình duyệt và nền tảng khác nhau.
- Selenium tập trung vào việc tự động hóa các ứng dụng dựa trên web.
- Selenium là một gói các tool cho cùng một chức năng và được biết đến như là một Suite (bộ). Mỗi tool sẽ được thiết kế để phục vụ cho các yêu cầu và môi trường riêng.



Hình 6.13: thành phần của Selenium

- *Selenium IDE* Được tạo ra bởi Shinaya Kasatani của Nhật. Selenium IDE là phần mở rộng của Firefox có thể tự động hóa trình duyệt thông qua tính năng ghi lại và phát lại.
- *Selenium Remote Control(Selenium 1)* Được tạo ra bởi Paul Hammant – 1 kỹ sư của ThoughtWork. Paul Hammant đã quyết định tạo một máy chủ sẽ hoạt động như một proxy HTTP để "đánh lừa" trình duyệt để tin rằng Selenium Core và ứng dụng web được thử nghiệm đến từ cùng một tên miền.
- *Web driver* được tạo ra bởi Simon Stewart vào năm 2006 khi các trình duyệt và các ứng dụng web đang trở nên mạnh hơn và hạn chế hơn với các chương trình JavaScript như Selenium Core.
- *Selenium Grid* được phát triển bởi Patrick Lightbody để thực hiện các kịch bản giống hoặc khác nhau trên nhiều nền tảng và trình duyệt, đồng thời để đạt được

thực hiện kiểm thử phân tán, kiểm thử ở nhiều môi trường khác nhau và tiết kiệm đáng kể thời gian thực hiện.

Vì sao sử dụng Selenium

- Selenium là tool free và có open source
- Selenium có cộng đồng sử dụng đông đảo
- Selenium có khả năng tương thích trên nhiều Browser (Firefox, chrome, Internet Explorer, Safari etc.)
- Selenium có khả năng tương thích tốt platform (Windows, Mac OS, Linux etc.)
- Selenium hỗ trợ nhiều ngôn ngữ lập trình (Java, C#, Ruby, Python, Pearl etc.)
- Selenium thường xuyên được phát triển và cải tiến

Cài đặt

- Cách đơn giản nhất để sử dụng ngay Selenium là dùng trình duyệt web Firefox.
- Ứng dụng xem code cơ bản, bạn có thể sử dụng text editor có sẵn trên hệ điều hành. Mình khuyên dùng "Notepad++" trên Windows hoặc "Sublime Text" trên Mac OS hoặc Linux.
- Extension Selenium IDE trên Firefox.

Cài đặt Selenium IDE

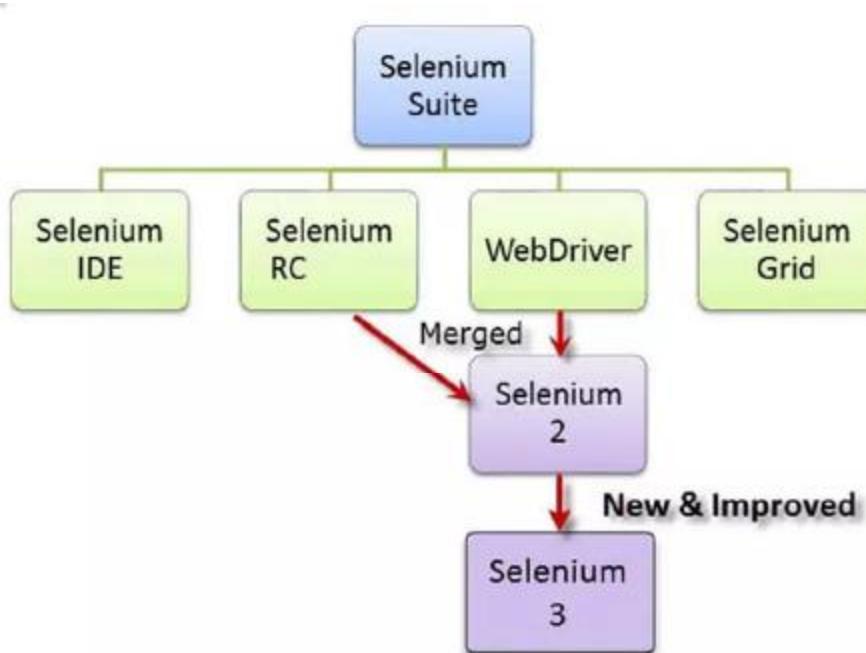
- Truy cập địa chỉ: <http://www.seleniumhq.org/download> bằng trình duyệt Firefox
- Tìm đến mục Selenium IDE và click vào version mới nhất để tải.
- Hoặc có thể vào phần tiện ích của trình duyệt và add-on selenium vào.

The screenshot shows the Selenium Downloads page. At the top, there's a navigation bar with links for About, Downloads, Projects, Documentation, Support, and Blog, along with a search bar. Below the navigation is a large blue header box with the word "Downloads". Underneath, there are three main sections:

- Selenium Server (Grid)**: Shows the "Latest stable version 3.141.59". It includes a note about using it in a Grid configuration and links to documentation and a beta version.
- The Internet Explorer Driver Server**: Notes that it's required for using the latest features of the WebDriver InternetExplorerDriver. It specifies compatibility with 32-bit Windows IE and 64-bit Windows IE, and provides a changelog link.
- WebDriver**: Shows the "Downloaded version 3.150.1 for 32-bit Windows IE (recommended) 64-bit Windows IE CHANGELOG".

Hình 6.14: Trang chủ Selenium

6.4.2 Thành phần của Selenium



Hình 6.15: Các thành phần của Selenium

Selenium là bộ phần mềm, mỗi bộ đáp ứng nhu cầu kiểm thử khác nhau Selenium gồm có 4 phần:

- Selenium IDE
- Selenium Remote Control (Selenium 1)
- Web Driver
- Selenium Grid

Selenium IDE: Là thành phần của Selenium. Nó hoạt động như trình duyệt Firefox add-on. Tác dụng chính của Selenium IDE là ghi lại quá trình tạo ra một mẫu test case và có khả năng phát lại quá trình tạo ra test case đó. Hơn nữa, nó hỗ trợ lưu test case dưới nhiều định dạng file khác nhau như html, java, php.

WebDriver là một khuôn khổ tự động hóa web cho phép bạn thực hiện các kiểm thử của mình trên các trình duyệt khác nhau. Nó nằm trong bộ kiểm thử tự động Selenium.

Selenium RC:

- Selenium RC là framework kiểm thử hàng đầu của toàn bộ dự án Selenium trong một thời gian dài.
- Đây là công cụ kiểm tra web tự động đầu tiên cho phép người dùng sử dụng ngôn ngữ lập trình mà họ thích.
- Tính đến phiên bản 2.25.0, RC có thể hỗ trợ các ngôn ngữ lập trình sau: Java, C#, PHP, Python, Perl, Ruby

Selenium Grid là một công cụ được sử dụng cùng với Selenium RC để chạy thử nghiệm song song trên các máy khác nhau và các trình duyệt khác nhau cùng một lúc. Thực hiện song song có nghĩa là chạy nhiều test case cùng một lúc. Tính năng, đặc điểm:

- Cho phép chạy đồng thời các test case trong nhiều trình duyệt và môi trường.
- Tiết kiệm nhiều thời gian.
- Sử dụng khái niệm hub-and-nodes. Hub hoạt động như một nguồn chính của lệnh Selenium cho mỗi kết nối với nó.

TÓM LƯỢC

Bài học này cung cấp các kiến thức về:

- Kiểm thử tự động (*Automation Software testing*)
- Kiểm thử dựa tên hành động (*Action-based testing*)
- Giới thiệu *Katolon Studio*
- Giới thiệu *Selenium*

BÀI 7: KIỂM THỬ ĐƠN VỊ (UNIT TEST)

Nội dung gồm các phần sau:

- Tổng quan về kiểm thử đơn vị
- Giới thiệu JUnit, NUnit
- Một số phương pháp trong JUnit, NUnit
- Thực hiện kiểm thử bằng JUnit, NUnit
- Tạo phương thức JUnit, NUnit

7.1 TỔNG QUAN VỀ KIỂM THỬ ĐƠN VỊ

7.1.1 Kiểm thử đơn vị là gì?

Kiểm thử đơn vị là một trong những loại kiểm thử phần mềm bao gồm giai đoạn kiểm thử ban đầu, trong đó các thành phần nhỏ nhất hoặc các mô-đun của phần mềm được kiểm tra riêng lẻ. Với phương pháp kiểm thử này, cả người kiểm thử và nhà phát triển đều có thể cô lập từng mô-đun, xác định và sửa chữa các lỗi hệ thống ở giai đoạn rất sớm của vòng đời phát triển phần mềm (SDLC). Về cơ bản, kiểm thử đơn vị xác minh các khía cạnh hành vi khác nhau của hệ thống được kiểm tra và có thể được phân loại rộng rãi thành kiểm thử đơn vị dựa trên trạng thái và dựa trên tương tác.

Một thử nghiệm đơn vị điển hình bao gồm ba giai đoạn bao gồm giai đoạn khởi tạo đầu tiên, nơi nó khởi tạo một phần nhỏ của ứng dụng mà nó muốn thử nghiệm. Giai đoạn thứ hai là giai đoạn bổ sung trong đó nó thêm một sự kích thích vào hệ thống đang được thử nghiệm và cuối cùng, giai đoạn thứ ba là giai đoạn kết quả nơi nó quan sát hành vi của ứng dụng mang lại. Rõ ràng, nếu hành vi quan sát được phù hợp với mong đợi, thì thử nghiệm đơn vị vượt qua, còn nếu không, nó không thành công. Điều này cho thấy có sự cố ở đâu đó trong hệ thống đang được kiểm tra. Ba giai đoạn kiểm tra này được đặt tên là Sắp xếp, Hành động và Khẳng định hoặc thường được gọi là AAA (Arrange, Act and Assert).

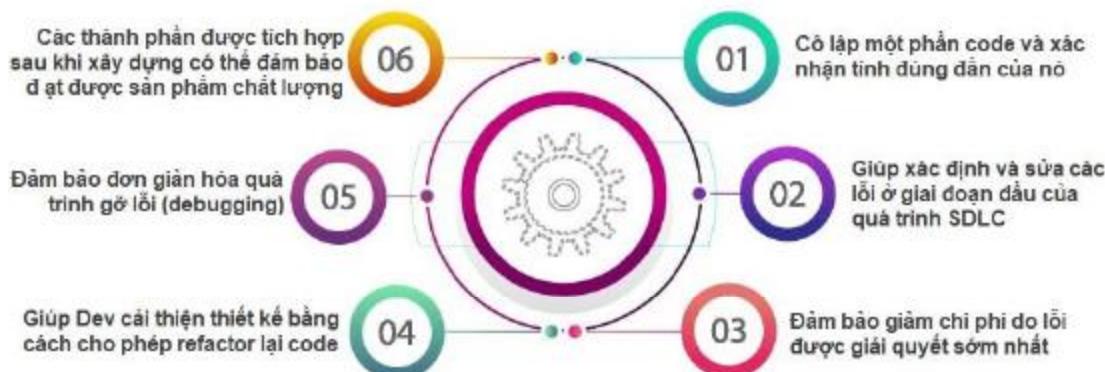
7.1.2 Tại sao Kiểm thử đơn vị lại quan trọng?

Kiểm thử đơn vị là kỹ thuật kiểm thử phần mềm trong đó một nhóm các thành phần hoặc mô-đun chương trình phần mềm được kiểm tra riêng lẻ. Kỹ thuật này giúp ích một cách hiệu quả trong việc xác nhận tính chính xác của một phần mã bằng cách xem xét các đoạn mã, đối tượng giả (mock objects), drivers và các unit test frameworks. Vì nó được thực hành ở giai đoạn thử nghiệm ban đầu, kỹ thuật thử nghiệm này đảm bảo xác định và sửa các lỗi ở giai đoạn đầu của SDLC để tránh tình trạng chúng trở nên đắt đỏ hơn cho doanh nghiệp để sửa khi được xác định ở các giai đoạn sau.

Một số Dev có thể cố gắng tiết kiệm thời gian bằng cách thực hiện rất ít kiểm thử đơn vị hoặc bỏ qua kiểm thử đơn vị, điều đó rõ ràng dẫn đến chi phí sửa lỗi cao hơn trong quá trình kiểm tra hệ thống, kiểm tra tích hợp và thậm chí là kiểm tra beta khi ứng dụng được hoàn thành.

Hơn nữa, ngoài những điều này, kiểm thử đơn vị giúp các nhóm phát triển hiểu code base, xác nhận tính đúng đắn của mã đã phát triển, sử dụng lại mã và thực hiện các thay đổi trong mã nhanh hơn. Với thực tiễn kiểm thử đơn vị thích hợp được áp dụng, các Dev và tester có thể giúp tiết kiệm thời gian vì các lỗi có thể được xác định sớm trong quá trình vì đây là giai đoạn đầu của kiểm thử. Và, việc bỏ qua hoặc hạn chế thực hiện kiểm thử đơn vị có thể làm tăng bất lợi các khuyết điểm và việc sửa chữa chúng trở nên phức tạp ở giai đoạn sau. Do đó, điều cần thiết là phải thực hành kiểm thử đơn vị ở giai đoạn đầu của quy trình kiểm thử phần mềm trước khi lập kế hoạch kiểm thử tích hợp.

7.1.3 Lợi ích của Kiểm thử Đơn vị



Hình 7.1: Lợi ích của Kiểm thử đơn vị

- Cố lập một phần code và xác nhận tính đúng đắn của nó
- Giúp xác định và sửa các lỗi ở giai đoạn đầu của quá trình SDLC
- Đảm bảo giảm chi phí do lỗi được giải quyết sớm nhất
- Giúp Dev cải thiện thiết kế bằng cách cho phép refactor lại code
- Đảm bảo đơn giản hóa quá trình gỡ lỗi (debugging)
- Với thực hiện kiểm tra đơn vị thích hợp, các thành phần được tích hợp sau khi xây dựng có thể đảm bảo đạt được sản phẩm chất lượng

7.1.4 Các loại Kiểm thử Đơn vị



Hình 7.2: Các loại Kiểm thử đơn vị

Về cơ bản, người kiểm thử đơn vị sử dụng ba loại kiểm thử đơn vị trong khi kiểm tra từng mô-đun một cách riêng biệt.

7.1.5 Ai thực hiện Kiểm thử đơn vị?

Kiểm thử đơn vị là giai đoạn kiểm thử phần mềm đầu tiên trong SDLC và nó thường được thực hiện trong quá trình phát triển ứng dụng. Các trường hợp kiểm thử đơn vị này được viết và thực thi bởi các Devs. Tuy nhiên, trong một số trường hợp đặc biệt mà các Devs không thực hiện quá trình này, kỹ thuật kiểm tra hộp trắng này được thực hiện bởi các kỹ sư QA.

7.1.6 Làm thế nào để làm Unit Testing?

Kiểm thử đơn vị có thể được thực hiện theo hai phương pháp, tức là kiểm tra thủ công và kiểm tra tự động.

Để thực hành kiểm thử đơn vị với phương pháp kiểm thử thủ công, điều cần thiết là người kiểm thử đơn vị phải có tài liệu hướng dẫn từng bước. Tuy nhiên, nếu xét đến

những efforts cần thiết cho kiểm thử thủ công, kiểm thử đơn vị tự động thường được hầu hết các doanh nghiệp ưa thích và lựa chọn hơn.

7.1.7 Quy trình kiểm thử đơn vị

Quy trình kiểm thử đơn vị được thực hiện trong 4 giai đoạn:

1. Creating test cases
2. Reviewing test cases
3. Baselinining test cases
4. Executing test cases

7.1.8 Quá trình kiểm tra đơn vị bao gồm:

- Devs viết code trong ứng dụng để kiểm tra chức năng
- Đoạn code sau đó được các Devs cô lập để xác nhận sự phụ thuộc giữa code và các đơn vị khác. Cách cô lập code này giúp xác định và loại bỏ các phần phụ thuộc.
- Devs sử dụng đáng kể các Unit test frameworks hoặc các tools kiểm thử đơn vị để phát triển các trường hợp kiểm thử tự động.
- Trong khi thực hiện các trường hợp kiểm thử, các frameworks kiểm thử đơn vị giúp gắn cờ và báo cáo các trường hợp kiểm thử không thành công. Ngoài ra, dựa trên các lỗi trong các trường hợp thử nghiệm, các frameworks kiểm thử đơn vị giúp dừng thử nghiệm liên quan.

7.1.9 Các thực hành tốt nhất về Unit Testing



Hình 7.3: Các thực hành tốt nhất về Unit Testing

Đảm bảo các Unit tests độc lập với nhau:

Trong khi thực hiện kiểm thử đơn vị, hãy đảm bảo rằng tất cả kiểm thử đơn vị là độc lập. Nếu có bất kỳ phụ thuộc nào, thì các bài kiểm tra đơn vị có thể bị ảnh hưởng khi có bất kỳ thay đổi hoặc cải tiến nào. Ngoài ra, nó có thể dẫn đến sự phức tạp cho các trường hợp thử nghiệm để chạy và debug. Do đó, hãy luôn đảm bảo rằng các trường hợp kiểm thử đơn vị là độc lập.

Luôn chỉ thực hiện một unit test vào một thời điểm:

Khi kiểm thử một unit của code, mặc dù nó liên quan đến nhiều trường hợp khác nhau, người kiểm thử đơn vị phải kiểm tra từng trường hợp sử dụng trong các trường hợp thử nghiệm khác nhau. Điều này sẽ đơn giản hóa một cách hiệu quả các nhóm thực hiện thay đổi mã hoặc tái cấu trúc (refactor).

Sử dụng AAA để dễ đọc:

AAA viết tắt của Arrange, Act, and Assert (Sắp xếp, Hành động và Khẳng định). Mô hình này giúp tách những gì đang được kiểm tra khỏi các bước "Arrange" và "Assert"; do đó làm giảm sự đan xen giữa các xác nhận với sự trợ giúp của "Act". Do đó, các trường hợp thử nghiệm dễ đọc hơn.

Sửa các lỗi trước khi chuyển sang Kiểm tra tích hợp:

Kiểm thử đơn vị là giai đoạn kiểm thử đầu tiên và nó được thực hiện trước khi chuyển sang giai đoạn kiểm thử tích hợp. Do đó, trước khi chuyển sang cấp thử nghiệm tiếp theo, hãy đảm bảo sửa tất cả các lỗi đã xác định trong giai đoạn thử nghiệm đơn vị.

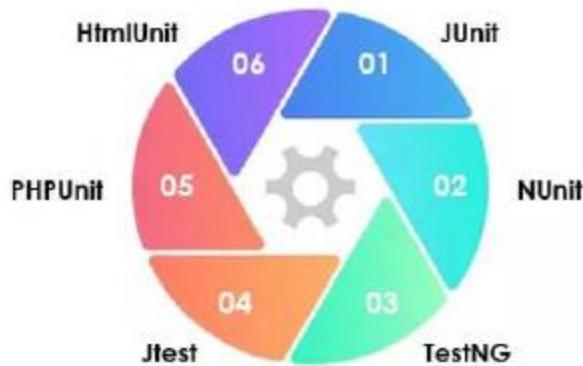
Đảm bảo đặt tên biến thích hợp:

Trong kiểm thử đơn vị, một trong những thực hành quan trọng và cần quan tâm nhất đó là đặt tên phù hợp cho các biến. Do đó, tránh sử dụng các magic strings, và cũng tuân theo các quy ước đặt tên rõ ràng và nhất quán.

Luôn tách biệt code ở môi trường test và production

Trong khi thực hiện kiểm thử đơn vị, hãy đảm bảo rằng unit test code không được triển khai cùng với mã nguồn trong bản build của bạn.

7.1.10 Một số framework Unit Test



Hình 7.4: Các framework của Unit Test

7.2 KIỂM THỬ JUNIT

7.2.1 JUnit là gì?

JUnit là một framework mã nguồn mở, miễn phí, đơn giản dùng để unit test cho ngôn ngữ lập trình Java. Trong Java, chúng ta thường sẽ sử dụng method để làm unit test. Chúng ta có thể sử dụng JUnit để viết code test cho cả unit testing và integration testing.

Trong Java, để thực hiện viết code cho Unit Test chúng ta có thể sử dụng một trong hai Framework: JUnit và TestNG.

7.2.2 Các tính năng của JUnit

- JUnit là một framework mã nguồn mở, được sử dụng để viết và chạy kiểm thử.
- Cung cấp các annotation để định nghĩa các phương thức kiểm thử.
- Cung cấp các Assertion để kiểm tra kết quả mong đợi.
- Cung cấp các test runner để thực thi các test script.
- Test case JUnit có thể được chạy tự động.
- Test case JUnit có thể được tổ chức thành các test suite.
- JUnit cho thấy kết quả test một cách trực quan: pass (không có lỗi) là màu xanh và fail (có lỗi) là màu đỏ.

7.2.3 Một số khái niệm trong JUnit

- Unit Test case: là 1 chuỗi code để đảm bảo rằng đoạn code được kiểm thử làm việc như mong đợi. Mỗi function sẽ có nhiều test case, ứng với mỗi trường hợp function chạy.
- Setup: Đây là hàm được chạy trước khi chạy các test case, thường dùng để chuẩn bị dữ liệu để chạy test.
- Teardown: Đây là hàm được chạy sau khi các test case chạy xong, thường dùng để xóa dữ liệu, giải phóng bộ nhớ.
- Assert: Mỗi test case sẽ có một hoặc nhiều câu lệnh Assert, để kiểm tra tính đúng đắn của hàm.
- Mock: là một đối tượng ảo, mô phỏng các tính chất và hành vi giống hệt như đối tượng thực được truyền vào bên trong khối mã đang vận hành nhằm kiểm tra tính đúng đắn của các hoạt động bên trong. Giả sử chương trình của chúng ta được chia làm 2 module: A và B. Module A đã code xong, B thì chưa. Để test module A, ta dùng mock để làm giả module B, không cần phải đợi tới khi module B code xong mới test được.
- Test Suite : Test suite là một tập các test case và nó cũng có thể bao gồm nhiều test suite khác, test suite chính là tổ hợp các test.

7.2.4 Cài đặt JUnit

Ngày nay, JUnit được tích hợp sẵn trong hầu hết các Java IDE (Eclipse, NetBeans và IntelliJ). Nếu không có sẵn, các bạn có thể tạo một project Maven và thêm thư viện JUnit vào file **pom.xml** như sau:

```

1 <!-- https://mvnrepository.com/artifact/junit/junit -->
2 <dependency>
3   <groupId>junit</groupId>
4   <artifactId>junit</artifactId>
5   <version>4.12</version>
6   <scope>test</scope>
7 </dependency>
```

7.2.5 Ví dụ sử dụng JUnit trên Eclipse

Giả sử chúng ta có một class util có 2 phương thức devide() và add().

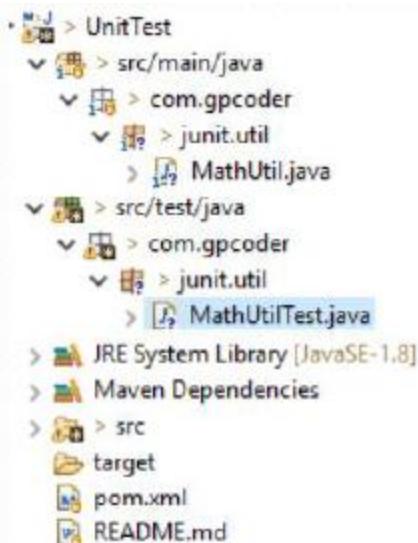
- Phương thức divide() : thực hiện chia phần nguyên của 2 số. Phương thức này nhận 2 đối số: số bị chia (dividend) và số chia (divisor). Nếu số chia là 0 thì chương trình sẽ throw một ngoại lệ, ngược lại chương trình sẽ trả về kết quả sau khi thực hiện chia nguyên.
- Phương thức add() : sẽ thực hiện tính tổng của 2 số nguyên.

Chương trình của chúng ta như sau:

```

1 package com.gpcoder.junit.util;
2
3 public class MathUtil {
4
5     private MathUtil() {
6         throw new UnsupportedOperationException("Cannot call constructor directly!");
7     }
8
9     public static int divide(int dividend, int divisor) {
10        if (divisor == 0) {
11            throw new IllegalArgumentException("Cannot divide by zero (0).");
12        }
13        return dividend / divisor;
14    }
15
16    public static int add(int number1, int number2) {
17        return number1 - number2;
18    }
19 }
```

Bây giờ chúng ta sẽ sử dụng JUnit để kiểm tra phương thức trên với các đầu vào khác nhau.



- Đầu tiên chúng ta sẽ tạo một class mới với suffix là xxxTest. Đây là một naming convention cho Unit Test. Class này nên được đặt trong thư mục test, cùng package name với tên của class cần viết unit test để dễ dàng quản lý.
- Tiếp theo, chúng ta sẽ tạo test case để test các trường hợp có thể có của một phương thức. Mỗi test case nên tạo một phương thức để kiểm tra:
 - Phương thức devide() có thể có một số trường hợp test sau: trường hợp kết quả phép chia ra một số nguyên, trường hợp kết quả phép chia ra số lẻ, trường hợp số chia là số 0.
 - Phương thức add() : chỉ đơn giản kiểm tra kết quả cộng 2 số.

```
1 package com.gpcoder.junit.util;
2
3 import org.junit.Assert;
4 import org.junit.Test;
5
6 public class MathUtilTest {
7
8     @Test
9     public void divide_SixDividedByTwo_ReturnThree() {
10         final int expected = 3;
11
12         final int actual = MathUtil.divide(6, 2);
13
14         Assert.assertEquals(expected, actual);
15     }
16
17     @Test
18     public void divide_OneDividedByTwo_ReturnZero() {
19         final int expected = 0;
20
21         final int actual = MathUtil.divide(1, 2);
22
23         Assert.assertEquals(expected, actual);
24     }
25
26     @Test(expected = IllegalArgumentException.class)
27     public void divide_OneDividedByZero_ThrowsIllegalArgumentException() {
28         MathUtil.divide(1, 0);
29     }
30
31     @Test
32     public void add_SixAddedByTwo_ReturnEight() {
33         final int expected = 8;
34
35         final int actual = MathUtil.add(6, 2);
36
37         Assert.assertEquals(expected, actual);
38     }
39 }
```

Để chạy kiểm tra các test case trên, chúng ta sẽ chọn chuột phải trên class tương ứng cần test, sau đó chọn **Run As -> Unit Test**. Tương tự, chúng ta cũng có thể thực thi test cho một phương thức hoặc cả project.

Chúng ta có kết quả như sau:

```

1 package com.gpcoder.junit.util;
2
3 import org.junit.Assert;
4 import org.junit.Test;
5
6 public class MathUtilTest {
7
8     @Test
9     public void divide_SixDividedByTwo_ReturnThree() {
10         final int expected = 3;
11
12         final int actual = MathUtil.divide(6, 2);
13
14         Assert.assertEquals(expected, actual);
15     }
16
17     @Test
18     public void divide_OneDividedByTwo_ReturnZero() {
19
20     }
21 }

```

JUnit 22
Finished after 0.046 seconds

Runs: 4/4 Errors: 0 Failures: 1

com.gpcoder.junit.util.MathUtilTest [Runner: JUnit 4] (0.055 s) Failure Trace

- divide_OneDividedByZero_ThrowsIllegalArgumentException (0.000 s)
- divide_OneDividedByTwo_ReturnZero (0.000 s)
- divide_SixDividedByTwo_ReturnThree (0.000 s)
- add_SixAddedByTwo_ReturnEight (0.002 s)

java.lang.AssertionError: expected:<8> but was:<4>
at com.gpcoder.junit.util.MathUtilTest.add_SixAddedByTwo_ReturnEight(MathUtilTest.java:37)

Trên kết quả test, chúng ta có thể thấy được tổng thời gian thực hiện tất cả các test case (0.055 seconds), thời gian thực thi mỗi test case, kết quả các test case tương ứng.

Như trong ví dụ trên, chúng ta có 3 phương thức pass (có màu xanh) cho phương thức divide(). Điều này có nghĩa là code logic của phương thức devide() đã đúng như mong đợi.

Phương thức add_SixAddedByTwo_ReturnEight() có màu đỏ, điều này có nghĩa là logic của phương thức add() đã có gì đó không đúng như mong đợi. Khi chúng ta click chuột vào test case bị lỗi, IDE sẽ hiển thị một vài thông tin chi tiết tại sao kết quả lại fail. Như hình bên trái, chúng ta có thể thấy là phương thức add() đang mong muốn là 8 nhưng kết quả là 4. Từ đó, chúng ta có thể kiểm tra lại code của chương trình để tìm nguyên nhân xảy ra kết quả không mong đợi.

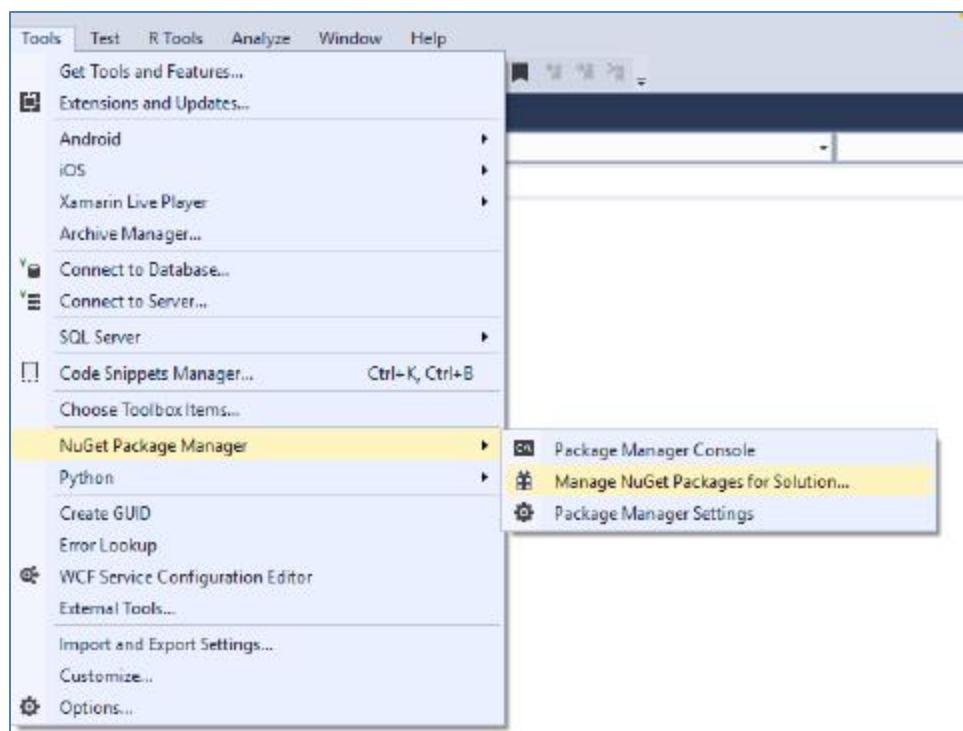
7.3 KIỂM THỬ NUNIT

7.3.1 NUnit là gì?

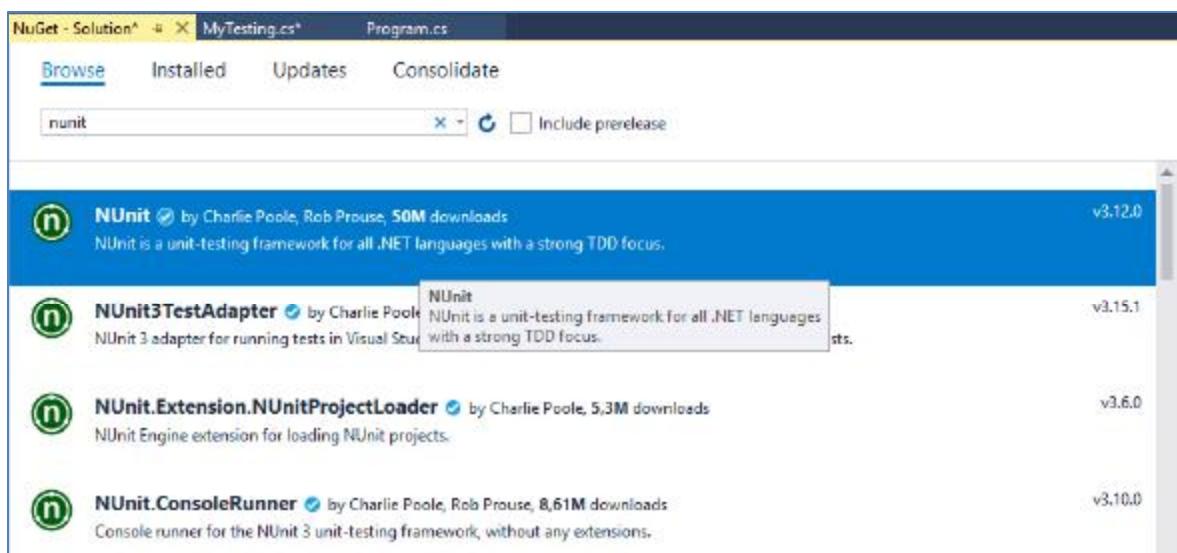
Unit Testing có thể được thực hiện dễ dàng nhờ các framework. Framework phổ biến hỗ trợ cho các nhà phát triển .NET là NUnit vì nó dễ dùng, dễ nhớ, có nhiều tính năng nổi trội và dễ dàng cài đặt đến Visual Studio qua NuGet.

7.3.2 Cài đặt NUnit

Để cài **NUnit**, trong Visual Studio chọn Tools > NuGet Package Manager > Manage NuGet Packages for Solution



Trong cửa sổ NuGet – Solution chọn tab Browse và gõ cụm từ 'nunit' trong ô tìm kiếm và chọn NUnit đầu tiên:



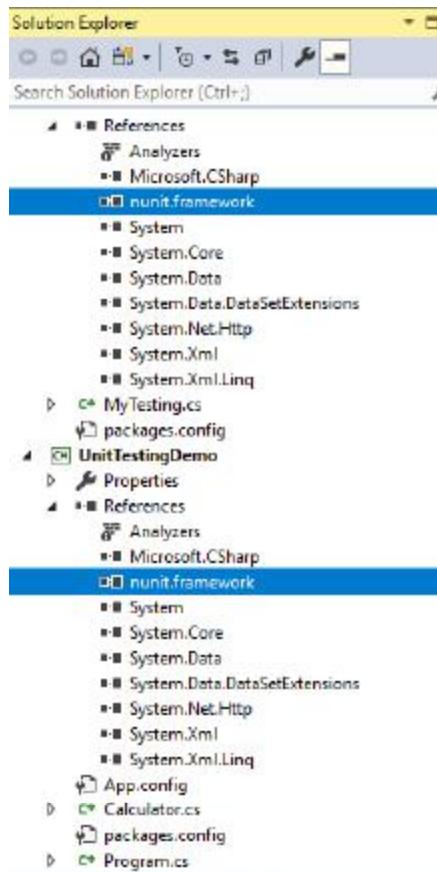
Bên khung cửa sổ NUnit bên phải chọn Project (bao gồm hai dự án) và nhấn Install:



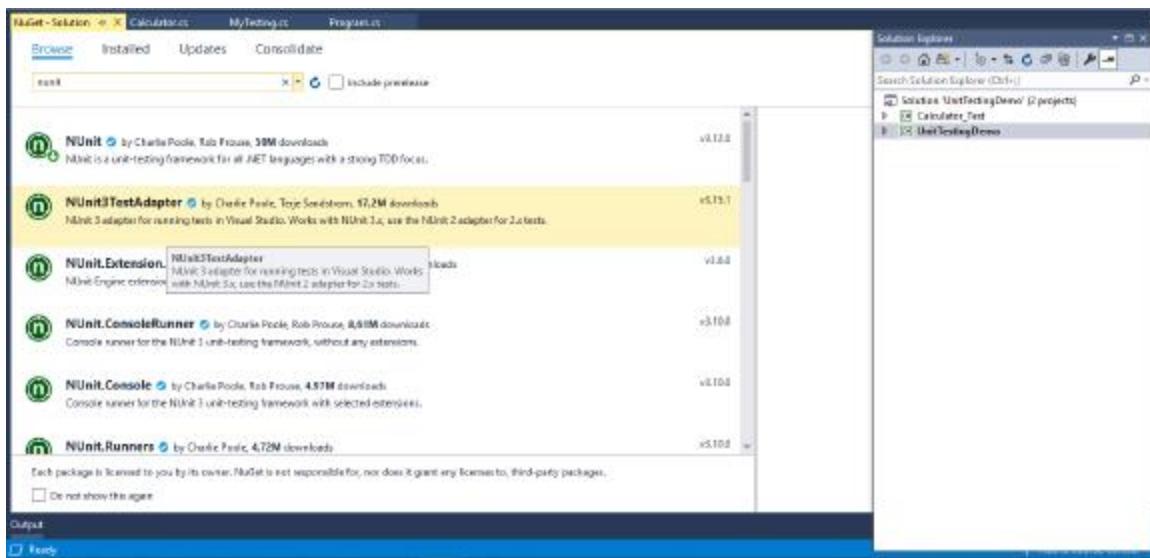
Nếu hộp thoại Preview Changes xuất hiện thì nhấn OK. Nếu cài đặt thành công thì kết quả từ hộp thoại Output sẽ như sau:

```
Output
Show output from: Package Manager
Retrieving package 'NUnit 3.12.0' from 'nuget.org'.
Adding package 'NUnit.3.12.0' to folder 'D:\CONG VIEN 2019-2020\Learn 2018-2019\Unit Testing\UnitTestingDemo\packages'
Added package 'NUnit.3.12.0' to Folder 'D:\CONG VIEN 2019-2020\Learn 2018-2019\Unit Testing\UnitTestingDemo\packages'
Added package 'NUnit.3.12.0' to 'packages.config'
Successfully installed 'NUnit 3.12.0' to Calculator_Test
Executing nuget actions took 3,67 sec
Found package 'NUnit 3.12.0' in 'D:\CONG VIEN 2019-2020\Learn 2018-2019\Unit Testing\UnitTestingDemo\packages'.
Package 'NUnit.3.12.0' already exists in folder 'D:\CONG VIEN 2019-2020\Learn 2018-2019\Unit Testing\UnitTestingDemo\packages'
Added package 'NUnit.3.12.0' to 'packages.config'
Successfully installed 'NUnit 3.12.0' to UnitTestingDemo
Executing nuget actions took 472,99 ms
Time Elapsed: 00:00:04.5181912
===== Finished =====
```

Đóng cửa sổ NuGet – Solution và Output. Trở lại cửa sổ Solution Explorer tìm đến mục Reference của hai sự án sẽ xuất hiện nunit.framework:



Ngoài ra, chúng ta cũng cài đặt **NUnit3TestAdapter**:



Mục đích cài đặt thư viện này là để thực thi các đơn vị kiểm thử NUnit trong Visual Studio.

7.3.3 Tạo dự án kiểm thử

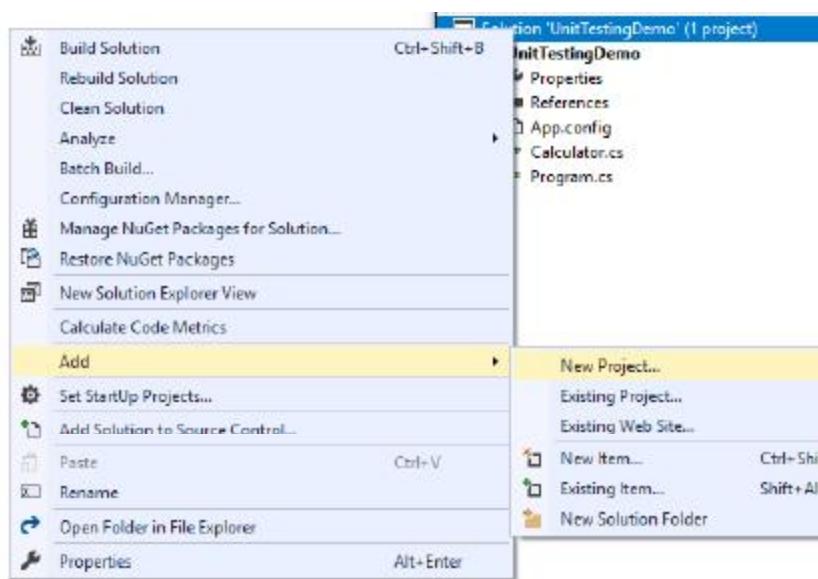
Dự án sử dụng framework **NUnit** trong Visual Studio 2017 Community. Để đơn giản, chúng ta sẽ tạo một dự án Console App (.NET Framework) tên **UnitTestingDemo** và thêm một lớp tên **Calculator** đến dự án này. Thay đổi nội dung tập tin **Calculator.cs** như sau:

```

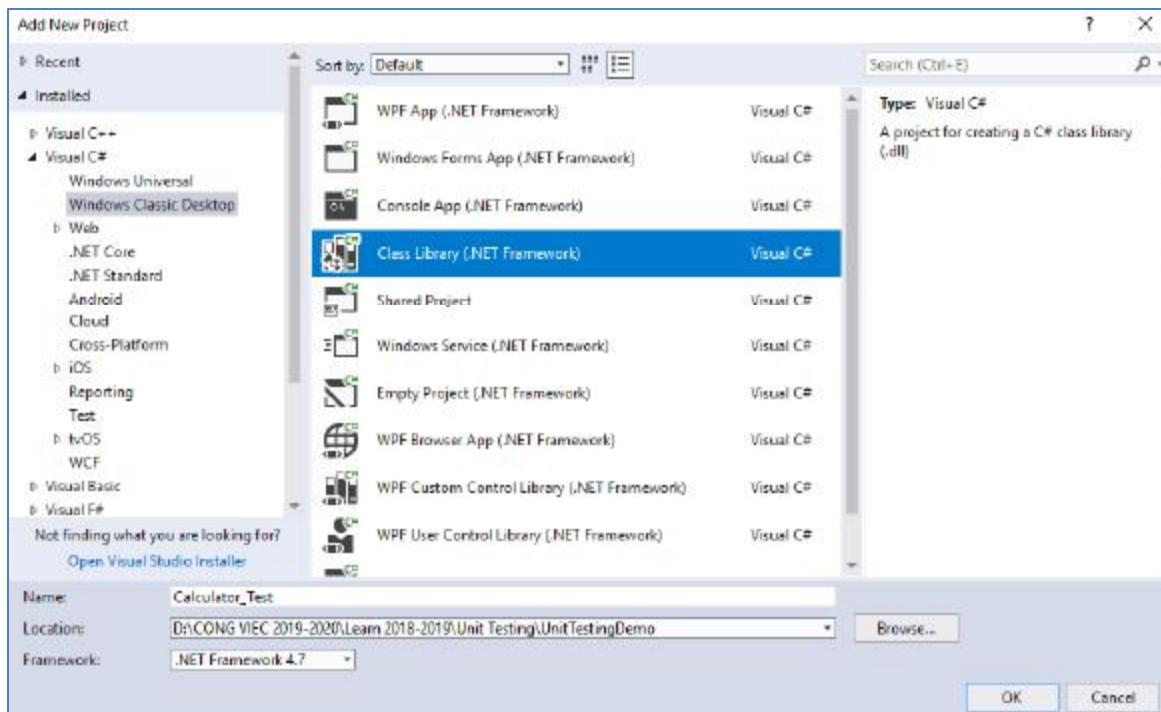
1  using System;
2
3  using System.Collections.Generic;
4
5  using System.Linq;
6
7  using System.Text;
8
9  using System.Threading.Tasks;
10
11
12  namespace UnitTestingDemo
13  {
14
15      public class Calculator
16      {
17
18          public int Add(int a, int b)
19          {
20              return a + b;
21
22          }
23
24
25          public int Sub(int a, int b)
26          {
27
28              return a - b;
29
30          }
31
32
33      }
34
35
36  }

```

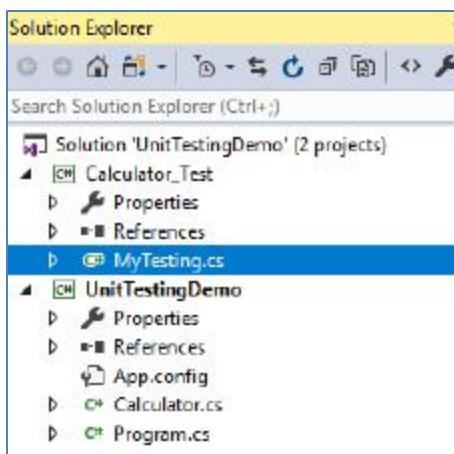
Kế tiếp, thêm một lớp thư viện đến Solution bằng cách nhấp chuột phải vào Solution trong cửa sổ Solution Explorer chọn Add > New Project



Trong cửa sổ New Project chọn Class Library (.NET Framework), đặt tên lớp là **Calculator_Test**:

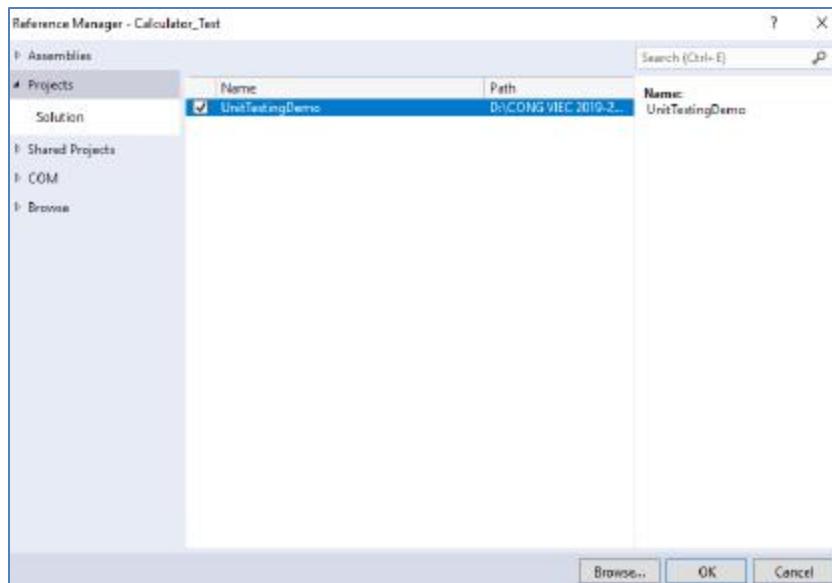


Nhấn OK. Đổi tên tập tin Class1.cs trong dự án **Calculator_Test** vừa tạo thành **MyTesting.cs**



Ngoài ra, chúng ta sẽ thêm một tham chiếu từ dự án **Calculator_Test** đến dự án **UnitTestingDemo** với mục đích là tham chiếu các lớp hay phương thức từ lớp **UnitTestingDemo**.

Nhấn chuột phải vào **Calculator_Test**, chọn Add > Reference. Trong cửa sổ Reference Manager chọn Project và dự án **UnitTestingDemo**:



7.3.4 Các thuộc tính **TestFixture**, **Test** và lớp **Assert**

NUnit chứa hai thuộc tính **TestFixture** và **Test** dùng để đánh dấu lớp và phương thức sẽ là các đơn vị kiểm thử (unit tests). Trước khi sử dụng hai thuộc tính này, chúng ta cần khai báo thư viện NUnit:

```
1 | using NUnit.Framework;
```

Nếu muốn lớp **MyTesting** sẽ là lớp kiểm thử, thêm thuộc tính **TestFixture** và khai báo thư viện NUnit đến tập tin **MyTesting.cs** như sau:

```

1 | using System;
2 |
3 | using System.Collections.Generic;
4 |
5 | using System.Linq;
6 |
7 | using System.Text;
8 |
9 | using System.Threading.Tasks;
10 |
11 | using NUnit.Framework;
12 |
13 | namespace Calculator_Test
14 |
15 | {
16 |
17 |     [TestFixture]
18 |
19 |     public class MyTesting
20 |
21 |     {
22 |
23 |     }
24 |
25 | }
```

Lưu ý rằng, thuộc tính **TestFixture** được đặt trong dấu ngoặc vuông.

Sau khi chọn lớp kiểm thử, chúng ta sẽ xây dựng hai phương thức Add_Test và Sub_Test dùng để kiểm thử các phương thức Add và Sub từ lớp **Calculator**. Lớp **MyTesting** được thay đổi như sau:

```
1 [TestFixture]
2
3 public class MyTesting
4 {
5
6     [Test]
7     public void Add_Test() {
8
9         }
10    }
11
12    [Test]
13    public void Sub_Test() {
14
15        }
16    }
17
18    }
```

Thuộc tính Test được đặt trong dấu ngoặc vuông để đánh dấu các phương thức là đơn vị kiểm thử. Trước khi viết vài đoạn mã cho hai phương thức Add_Test và Sub_Test, chúng ta sẽ tìm hiểu lớp **Assert** từ namespace **NUnit.Framework**.

Lớp **Assert** như là chiếc cầu nối giữa mã chương trình cần kiểm thử và NUnit. Một đơn vị kiểm thử sẽ dùng lớp này với mục đích khai báo một giả định nào đó là tồn tại và nếu các đối số được chuyển vào lớp **Assert** được thực hiện với kết quả khác với giả định thì đơn vị kiểm thử này thất bại.

Thay đổi nội dung hai phương thức Add_Test và Sub_Test như sau:

```

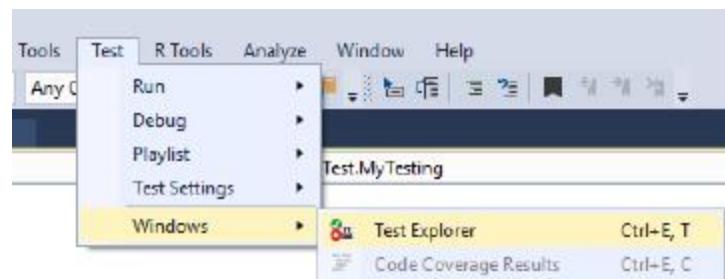
1 [TestFixture]
2
3 public class MyTesting
4 {
5
6     [Test]
7
8     public void Add_Test()
9     {
10         Calculator cal = new Calculator();
11
12         int add_Result = cal.Add(3, 5);
13
14         Assert.That(add_Result, Is.EqualTo(8));
15
16     }
17
18     [Test]
19
20     public void Sub_Test()
21     {
22         Calculator cal = new Calculator();
23
24         int sub_Result = cal.Sub(3, 5);
25
26         Assert.That(sub_Result, Is.EqualTo(8));
27
28     }
29
30 }
31

```

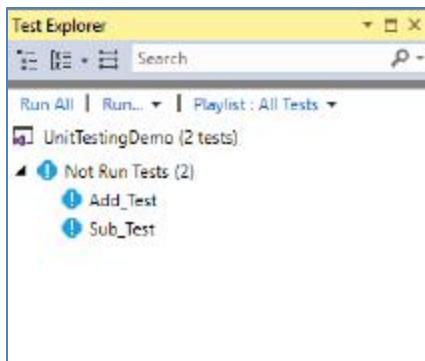
Chúng ta sử dụng phương thức **That** của lớp **Assert** để xác nhận một kết quả là đúng với giả định hay không. Trong hai phương thức Add_Test và Sub_Test, chúng ta gọi các phương thức Add và Sub từ lớp Calculator với các đối số là 3 và 5. Kết quả từ các phương thức này sẽ được so sánh với kết quả giả định là 8.

7.3.5 Thực hiện kiểm thử với công cụ Test Explorer

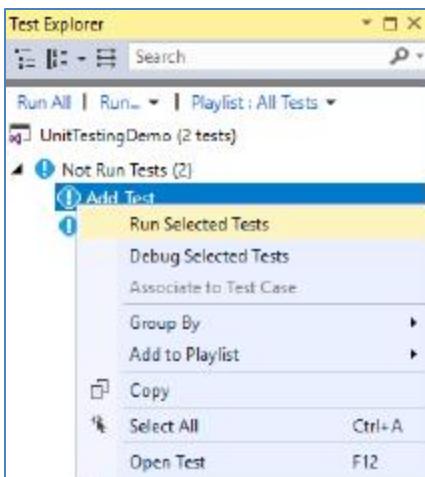
Chúng ta thực hiện kiểm thử bằng cách dùng công cụ **Test Explorer** từ mục Test trong Visual Studio 2017:



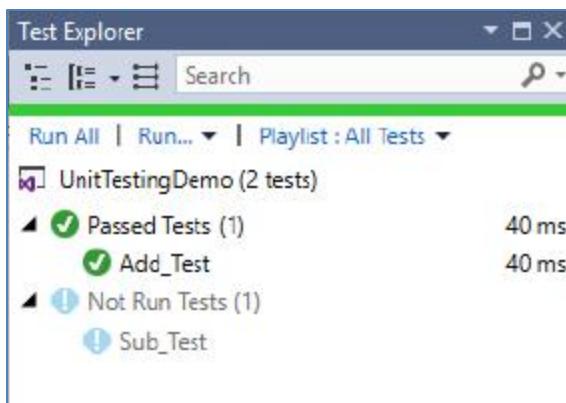
Cửa sổ Test Explorer như sau:



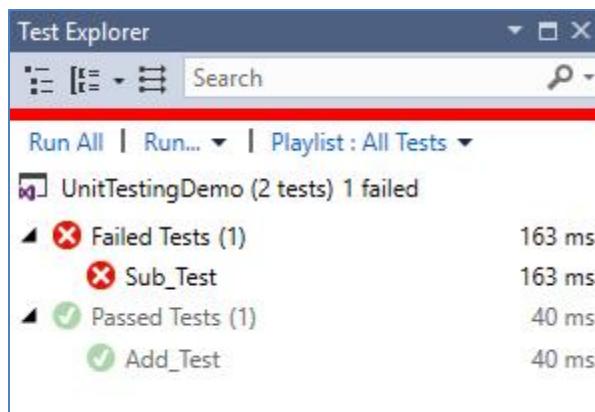
Chúng ta có thể chọn Run All để chạy cả hai phương thức kiểm thử hay chúng ta có thể chọn một trong hai phương thức và nhấn chuột phải chọn Run Selected Tests, ví dụ chọn Add_Test:



Kết quả:



Như vậy đơn vị kiểm thử Add_Test đã thành công vì tổng 3 và 5 là 8 khớp với kết quả giả định. Tương tự, chúng ta thực thi Sub_Test:



Sub_Test đã thất bại vì kết quả hiệu 3 và 5 là -2 khác với kết quả giả định là 8.

TÓM TẮT

Bài học này cung cấp các kiến thức về:

- Giới thiệu Kiểm thử đơn vị (*Unit test*)
- Kiểm thử *JUnit*
- Kiểm thử *NUnit*

BÀI 8: QUẢN LÝ CHẤT LƯỢNG PHẦN MỀM

Nội dung gồm các phần sau:

- Tìm hiểu về quá trình quản lý chất lượng và các hoạt động quá trình trung tâm của sự đảm bảo của chất lượng, lập kế hoạch chất lượng và kiểm soát chất lượng.
- Tìm hiểu sự quan trọng của các chuẩn mực trong quá trình quản lý chất lượng.
- Hiểu độ đo phần mềm là gì và sự khác biệt giữa độ đo tiên nghiệm và độ đo điều khiển.
- Tìm hiểu cách đo có thể hữu ích trong việc đánh giá các thuộc tính chất lượng sản phẩm.
- Có hiểu biết về các giới hạn hiện tại về độ đo phần mềm.

8.1 CHẤT LƯỢNG QUÁ TRÌNH VÀ CHẤT LƯỢNG SẢN PHẨM

Chất lượng phần mềm là một khái niệm phức tạp, nó không thể so sánh một cách trực tiếp với chất lượng trong sản xuất. Trong sản xuất, khái niệm của chất lượng được đưa ra là: sản phẩm phát triển phải phù hợp với đặc tả của nó (Crosby, 1979). Nhìn chung, định nghĩa này được áp dụng cho tất cả các sản phẩm, tuy nhiên đối với hệ thống phần mềm, nảy sinh một số vấn đề với định nghĩa này:

1. Đặc tả phải được định hướng tới các đặc trưng của sản phẩm mà khách hàng mong muốn. Tuy nhiên, tổ chức phát triển có thể cũng có các yêu cầu (như các yêu cầu về tính bảo trì) mà không được kể đến trong các đặc tả.
2. Chúng ta không biết làm cách nào định rõ các đặc trưng chất lượng (ví dụ như tính bảo trì được) một cách rõ ràng.

3. Việc viết được đầy đủ các đặc tả phần mềm là một công việc rất khó khăn. Vì vậy, mặc dù sản phẩm phần mềm có thể phù hợp với các đặc tả của nó, nhưng người sử dụng có thể không coi đó là sản phẩm chất lượng cao bởi vì nó không phù hợp với những mong đợi của họ.

Các nhà quản lý chất lượng tốt có mục tiêu là phát triển một “văn hoá chất lượng” nơi mà trách nhiệm của mọi cam kết cho sự phát triển sản phẩm để đạt tới mức độ cao của chất lượng sản phẩm. Họ khuyến khích các nhóm chịu trách nhiệm cho chất lượng công việc của mình và để phát triển các cách tiếp để cải tiến chất lượng. Khi mà các chuẩn và các thủ tục là phần cơ bản của quản lý chất lượng, kinh nghiệm của người quản lý chất lượng cho thấy có những mong đợi không thể nhìn thấy được về chất lượng phần mềm (tính bắt mắt, tính dễ đọc, ...) mà không thể được thể hiện một cách rõ ràng theo các chuẩn.

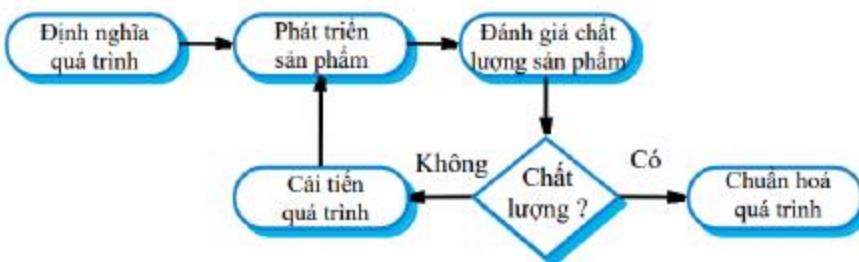
Việc quản lý chất lượng đã được chính thức hoá là rất quan trọng đối với các nhóm có nhiệm vụ phát triển các hệ thống lớn và phức tạp. Tài liệu về chất lượng là một bản ghi bao gồm những việc đã làm bởi mỗi nhóm nhỏ trong một dự án. Nó trợ giúp con người kiểm tra những nhiệm vụ quan trọng không được phép quên, hay một phần của nhóm tạo ra các giả định về những gì mà các nhóm khác đã làm. Nó cho phép các nhóm chịu trách nhiệm cho sự phát triển của hệ thống, thể hiện những gì mà nhóm phát triển đã thực hiện.

Đối với những hệ thống nhỏ, quản lý chất lượng vẫn rất quan trọng, nhưng với một cách tiếp cận đơn giản hơn được áp dụng. Không cần thiết nhiều công việc giấy tờ bởi vì một nhóm phát triển nhỏ có thể trao đổi trực tiếp. Vấn đề then chốt chất lượng cho sự phát triển các hệ thống nhỏ là việc thiết lập một “văn hoá chất lượng” và bảo đảm rằng tất cả các thành viên nhóm có ý thức tiếp cận một cách tích cực với chất lượng phần mềm.

Quản lý chất lượng phần mềm cho các hệ thống lớn có thể được chia vào 3 hoạt động chính.

1. Sự đảm bảo chất lượng: thiết lập một bộ khung có tổ chức các thủ tục và các chuẩn để hướng đến sản phẩm chất lượng cao.

2. Lập kế hoạch chất lượng: Việc chọn lựa các thủ tục và các chuẩn thích hợp từ khung này, được sửa chữa cho các dự án phần mềm riêng biệt.
3. Kiểm soát chất lượng: Định nghĩa và đưa ra các quá trình để đảm bảo rằng đội phát triển phần mềm phải tuân theo các thủ tục và các chuẩn chất lượng dự án.



Hình 8.1: Chất lượng dựa trên quá trình

Một đội độc lập chịu trách nhiệm đối với việc quản lý chất lượng và sẽ báo cáo tới người quản lý dự án ở cấp cao hơn. Đội quản lý chất lượng không được liên kết với bất cứ nhóm phát triển riêng biệt nào nhưng nhận trách nhiệm cho việc quản lý chất lượng. Lý do cho việc này là người quản lý dự án phải duy trì ngân sách dự án và lập lịch dự án. Nếu vẫn đề xuất hiện, họ có thể bị lôi cuốn vào việc thoả hiệp chất lượng sản phẩm, vì vậy họ phải có kế hoạch. Một đội quản lý chất lượng độc lập bảo đảm rằng mục tiêu tổ chức của chất lượng không bị thoả hiệp bởi ngân sách ngắn hạn và chi phí đã lên kế hoạch.

8.2 ĐẢM BẢO CHẤT LƯỢNG VÀ CÁC CHUẨN CHẤT LƯỢNG

Đảm bảo chất lượng là quá trình của việc định rõ làm cách nào để chất lượng sản phẩm có thể đạt được và làm thế nào để cho tổ chức phát triển biết phần mềm có yêu cầu chất lượng ở cấp độ nào. Đảm bảo chất lượng tiến trình có liên quan đầu tiên đến việc định ra hoặc chọn lựa các chuẩn sẽ được áp dụng cho quá trình phát triển phần mềm hay sản phẩm phần mềm. Như là một phần của quá trình đảm bảo chất lượng, bạn có thể chọn lựa hoặc tạo ra các công cụ và các phương pháp để phục vụ cho các chuẩn này.

Có 2 loại chuẩn có thể được áp dụng như là một phần của quá trình đảm bảo chất lượng là:

1. Các chuẩn sản phẩm: Những chuẩn này áp dụng cho sản phẩm phần mềm phát triển. Chúng bao gồm các định nghĩa của đặc tả, như là cấu trúc của tài liệu yêu cầu; các chuẩn tài liệu, như các tiêu đề giải thích chuẩn cho định nghĩa lớp đối tượng; và các chuẩn mã để định rõ làm cách nào ngôn ngữ lập trình có thể được sử dụng.
2. Các chuẩn quá trình: Những chuẩn này định ra quá trình nên được tuân theo trong quá trình phát triển phần mềm. Chúng có thể bao gồm các việc xác định các đặc tả. Quá trình thiết kế và kiểm định quá trình và một bản mô tả các tài liệu nên được ghi lại trong giai đoạn của những quá trình này.

Có một sự liên hệ rất gần giữa các chuẩn sản phẩm và chuẩn quá trình. Các chuẩn sản phẩm áp dụng cho đầu ra của quá trình phần mềm và trong nhiều trường hợp, các chuẩn quá trình bao gồm các hoạt động quá trình riêng biệt mà đảm bảo rằng các chuẩn sản phẩm được tuân theo.

Các chuẩn phần mềm là rất quan trọng vì những lý do sau:

1. Các chuẩn phần mềm dựa trên hiểu biết về những thực tiễn thích hợp nhất cho công ty. Kinh nghiệm này thường chỉ đạt được sau rất nhiều lần thử nghiệm và lỗi. Bổ xung nó vào các chuẩn giúp cho công ty tránh sự lặp lại sai lầm trong quá khứ. Các chuẩn chứa đựng các kinh nghiệm từng trải này rất có giá trị cho tổ chức.
2. Các chuẩn phần mềm cung cấp một cái khung cho việc thực thi quá trình đảm bảo chất lượng. Đưa ra các chuẩn tổng kết thực tiễn, đảm bảo chất lượng bao gồm việc bảo đảm rằng các chuẩn được tuân theo một cách chặt chẽ.
3. Các chuẩn phần mềm trợ giúp tính liên tục khi mà một người tiếp tục công việc của người khác đã bỏ dở. Các chuẩn đảm bảo rằng tất cả kỹ sư trong tổ chức chấp nhận cùng thói quen. Do vậy công sức nghiên cứu khi bắt đầu công việc mới sẽ giảm xuống.

Sự phát triển của các chuẩn dự án kỹ thuật phần mềm là quá trình rất khó khăn và tốn thời gian. Các tổ chức quốc gia, quốc tế như US DoD, ANSI, BSI, NATO và IEEE chủ động tạo ra các chuẩn. Những chuẩn này là chuẩn chung mà có thể được áp dụng

ở phạm vi của các dự án. Các tổ chức như NATO và các các tổ chức bảo vệ có thể yêu cầu các chuẩn của họ được tuân theo trong các hợp đồng phần mềm.

Các chuẩn quốc gia và quốc tế đã phát triển bao gồm cả công nghệ kỹ thuật phần mềm, ngôn ngữ lập trình như Java, và C++, các ký hiệu như là biểu tượng bản đồ, các thủ tục cho các yêu cầu nhận và viết phần mềm, các thủ tục đảm bảo chất lượng, kiểm tra phần mềm và quá trình thông qua (IEEE, 2003).

Các nhóm đảm bảo chất lượng mà đang phát triển các chuẩn cho công ty thường dựa trên chuẩn quốc gia và quốc tế. Sử dụng những chuẩn này như là điểm bắt đầu, nhóm đảm bảo chất lượng phải thảo ra một tài liệu tóm tắt chuẩn. Tài liệu này phải định ra những tiêu chuẩn được yêu cầu bởi tổ chức của họ.

Bảng 8.1: Các chuẩn quá trình và chuẩn sản phẩm

Các chuẩn sản phẩm	Các chuẩn quá trình
Mẫu rà soát thiết kế	Sắp đặt rà soát thiết kế
Cấu trúc tư liệu yêu cầu	Sự đệ trình tư liệu đến CM(???)
Phương pháp định dạng tiêu đề	Quá trình phát hành phiên bản
Kiểu lập trình Java	Quá trình thông qua kế hoạch dự án
Định dạng kế hoạch dự án	Quá trình kiểm soát thay đổi
Mẫu yêu cầu thay đổi	Quá trình ghi nhận kiểm tra.

Các kỹ sư phần mềm đôi khi coi các chuẩn là phức tạp và không thích hợp đối với hoạt động công nghệ của việc phát triển phần mềm. Mặc dù các kỹ sư phần mềm thường đồng ý về các yêu cầu chung cho các tiêu chuẩn, các kỹ sư thường tìm nhiều lý do tại sao các chuẩn không thực sự thích hợp với dự án riêng của họ. Để tránh những vấn đề này, những người quản lý chất lượng thiết lập những tiêu chuẩn cẩn thiết là những tài nguyên tương xứng, và nên tuân theo các bước sau:

1. Bao gồm các kỹ thuật phần mềm trong việc chọn lựa các chuẩn sản phẩm. Họ nên hiểu tại sao các tiêu chuẩn được thiết kế và cam kết tuân theo chuẩn này. Tài liệu chuẩn không chỉ là đơn giản là nói rõ chuẩn được tuân theo mà nó phải bao gồm lý do cẩn bản tại sao các tiêu chuẩn riêng biệt được chọn.
2. Kiểm tra và thay đổi các tiêu chuẩn một cách đều nhau phản ánh các công nghệ thay đổi. Một khi các tiêu chuẩn được phát triển, chúng có xu hướng được lưu trữ

trong tài liệu tóm tắt các tiêu chuẩn của công ty, và việc quản lý thường khó có thể thay đổi chúng. Một tài liệu tóm tắt tiêu chuẩn là rất cần thiết nhưng nó nên mở ra việc phản ánh các tình huống thay đổi và công nghệ thay đổi.

3. Cung cấp các công nghệ phần mềm để phục vụ các tiêu chuẩn bất kể khi nào có thể. Các tiêu chuẩn văn phòng là nguyên nhân của nhiều tranh phiền bởi vì công việc quá dài dòng để thực hiện chúng. Nếu công cụ phục vụ là có hiệu lực, bạn không cần cõ gắng thêm để tuân theo các chuẩn phát triển phần mềm.

Các chuẩn quá trình có thể gây ra nhiều khó khăn nếu một quá trình không có tính thực tế được áp đặt cho nhóm phát triển. Các kiểu khác nhau của phần mềm cần các quá trình phát triển khác nhau. Không nhất thiết phải quy định cách làm việc nếu nó không thích hợp cho một dự án hay đội dự án. Mỗi người quản lý dự án phải có quyền thay đổi các chuẩn quá trình theo những trường hợp riêng. Tuy nhiên, các chuẩn mà liên quan đến chất lượng sản phẩm và quá trình gửi- phát phải chỉ được thay đổi sau khi có sự cân nhắc cẩn thận.

Người quản lý dự án và người quản lý chất lượng có thể tránh nhiều vấn đề về các chuẩn không hợp lý bằng cách lập kế hoạch chất lượng chu đáo sớm trong dự án. Họ phải quyết định những chuẩn đưa vào trong tài liệu, mà những chuẩn không thay đổi được ghi vào tài liệu, còn những chuẩn nào có thể được chỉnh sửa và những chuẩn nào có thể được bỏ qua. Những chuẩn mới có thể phải được tạo ra để đáp ứng những yêu cầu riêng biệt của từng dự án. Ví dụ, tiêu chuẩn cho các đặc tả hình thức có thể được yêu cầu nếu những đặc tả này không được sử dụng trong các dự án trước. Khi mà đội có thêm kinh nghiệm với chúng, bạn nên lập kế hoạch chỉnh sửa và đưa ra những chuẩn mới.

8.3 LẬP KẾ HOẠCH VÀ KIỂM SOÁT CHẤT LƯỢNG

8.3.1 Lập kế hoạch chất lượng

Lập kế hoạch chất lượng là quá trình của sự phát triển một kế hoạch chất lượng cho một dự án. Kế hoạch chất lượng phải thiết lập các chất lượng phần mềm được yêu cầu và mô tả làm bằng cách nào những chất lượng này có thể được quyết định. Bởi vậy nó định ra phần mềm chất lượng cao thực sự có ý nghĩa như thế nào. Nếu không

có sự định trước này các kỹ sư có thể tạo ra các giả định khác nhau và đôi khi là xung đột với nhau về các thuộc tính sản phẩm sẽ được tối ưu hóa.

Kế hoạch chất lượng sẽ chọn những chuẩn tổ chức mà nó thích hợp với một sản phẩm riêng biệt và quá trình phát triển. Những chuẩn mới có thể phải được định nghĩa nếu dự án sử dụng các phương pháp và công cụ mới. Humphrey (Humphrey, 1989) trong cuốn sách kinh điển về quản lý phần mềm, gợi ý rằng một cấu trúc phân cấp cho kế hoạch chất lượng. Điều này bao gồm:

1. Sự giới thiệu sản phẩm Một mô tả về sản phẩm, mô tả định hướng thị trường dự định và các mong đợi chất lượng cho sản phẩm.
2. Các kế hoạch sản phẩm Kì hạn phát hành và các trách nhiệm sản phẩm cùng với các dự án cho việc phân phối và dịch vụ sản phẩm.
3. Các mô tả quá trình Các quá trình phát triển và dịch vụ sẽ được sử dụng cho quản lý và phát triển sản phẩm.
4. Các mục tiêu chất lượng Các mục tiêu và kế hoạch chất lượng cho sản phẩm bao gồm việc xác định và điều chỉnh các thuộc tính chất lượng quan trọng của sản phẩm.
5. Rủi ro và quản lý rủi ro Các rủi ro chính mà có thể ảnh hưởng đến chất lượng và các hoạt động sản phẩm.

Các kế hoạch chất lượng thực sự khác biệt trong chi tiết phụ thuộc vào kích thước và kiểu của hệ thống mà đang được phát triển. Tuy nhiên, khi viết các kế hoạch chất lượng, bạn nên cẩn gắng giữ cho chúng ngắn nhất có thể. Nếu như tài liệu quá dài, mọi người sẽ không thể đọc nó, điều này sẽ phá huỷ mục đích của việc tạo ra kế hoạch chất lượng.

Có một phạm vi rộng của các thuộc tính chất lượng phần mềm tiềm năng mà bạn nên xem xét trong quá trình lập kế hoạch chất lượng. Trong kế hoạch chất lượng, bạn phải định ra những thuộc tính chất lượng quan trọng nhất cho phần mềm đang được phát triển. Điều này nghĩa là trong từng dự án, bạn phải xác định yếu tố nào là quan trọng và yếu tố nào buộc phải bỏ qua. Nếu bạn đã phát biểu điều này trong kế hoạch chất lượng, các kỹ sư phát triển có thể hợp tác để đạt được điều này. Kế hoạch phải bao gồm việc định rõ quá trình đánh giá chất lượng. Điều này nên là một phương

pháp chuẩn của việc đánh giá một số chất lượng, như khả năng bảo trì hay tính bền vững được hiện diện trong sản phẩm.

Bảng 8.2: Danh sách các tiêu chí để đánh giá chất lượng tham khảo

Tính an toàn	Tính có thể hiểu được	Tính di động
Tính bảo mật	Tính có thể kiểm tra	Tính tiện dụng
Tính tin cậy	Tính thích khi	Tính tái sử dụng
Tính mềm dẻo	Tính mô đun	Tính hiệu quả
Tính bền vững	Tính phức tạp	Tính dễ học.

8.3.2 Kiểm soát chất lượng

Kiểm soát chất lượng bao gồm việc kiểm tra quá trình phát triển phần mềm để đảm bảo rằng các thủ tục và các chuẩn đảm bảo chất lượng được tuân theo. Có hai cách tiếp cận bổ xung cho nhau mà có thể được sử dụng để kiểm tra chất lượng của mức độ thực hiện của dự án.

- Việc rà soát lại chất lượng nơi mà phần mềm, tài liệu của nó và các quá trình đã sử dụng để tạo ra mà phần mềm được rà soát bởi một nhóm người. Việc rà soát chịu trách nhiệm việc kiểm tra các chuẩn dự án được tuân theo và phần mềm và các tài liệu làm cho phù hợp với những chuẩn này. Sự lệch khỏi các chuẩn này phải được chú ý và người quản lý dự án phải được cảnh báo về chúng.
- Đánh giá phần mềm tự động là nơi phần mềm và các tài liệu được sẽ được tạo ra xử lý bởi một số chương trình và được so sánh với các chuẩn áp dụng cho dự án phát triển riêng biệt. Đánh giá tự động này có thể bao gồm việc đo một số thuộc tính phần mềm và so sánh những độ đo với một số mức độ mong muốn.

Sau đó là bước rà soát chất lượng, được coi là một phương thức được sử dụng rộng rãi nhất trong việc rà soát chất lượng của một quá trình hay sản phẩm. Việc rà soát bao gồm một nhóm người kiểm tra một phần hay tất cả một quá trình phần mềm, hệ thống hay các tài liệu liên quan với các vấn đề tiềm tàng phát hiện. Các kết luận của việc rà soát được ghi lại và thông qua một cách chính thức tới tác giả hay bất kỳ người nào chịu trách nhiệm việc sửa lại những vấn đề được phát hiện.

Kiểu rà soát	Mục đích chủ yếu
Kiểm tra thiết kế hay chương trình	Phát hiện các lỗi chi tiết trong các yêu cầu, thiết kế hay mã. Danh sách kiểm tra các lỗi có thể sẽ dẫn đến việc rà soát.
Rà soát tiến độ	Cung cấp thông tin cho việc quản lý tiến độ của dự án. Đây cũng vừa là quá trình và vừa là rà soát sản phẩm nó có liên quan đến chi phí, kế hoạch, lập lịch.
Rà soát chất lượng	Tiến hành các phân tích công nghệ của các thành phần sản phẩm hay tư liệu để tìm ra chỗ không tương xứng giữa đặc tả và thiết kế thành phần, mã hay tư liệu và đảm bảo rằng các chuẩn chất lượng đã được đưa ra được tuân theo.

Vấn đề của đội rà soát là để tìm ra các lỗi và các mâu thuẫn và chuyển giao chúng cho người thiết kế hay tác giả của tài liệu. Các việc rà soát được dựa trên tài liệu nhưng nó không giới hạn tới các đặc tả, các thiết kế hay mã. Các tài liệu như các mô hình quá trình, kế hoạch kiểm tra, các thủ tục quản lý cấu hình, các chuẩn quá trình và tài liệu chỉ dẫn người dùng có thể tất cả được rà soát lại.

Đội rà soát nên có hạt nhân là ba hay bốn người mà được chọn như là người rà soát chủ yếu. Một thành viên nên là người thiết kế lâu năm người mà có thể chịu trách nhiệm cho việc ra quyết định công nghệ quan trọng. Những người xét duyệt quan trọng có thể mời các thành viên dự án khác. Họ có thể không phải xét duyệt toàn bộ tài liệu. Hơn nữa, họ tập trung vào một số phần mà ảnh hưởng đến công việc của họ. Đội rà soát có thể chuyển tài liệu đã được rà soát và yêu cầu cho các lời chú giải từ hình ảnh rộng của các thành viên dự án.

Các tài liệu được rà soát phải được phân phối tốt trước khi xét duyệt để cho phép những người rà soát có thể đọc và hiểu chúng. Mặc dù sự trễ này có thể phá vỡ quá trình phát triển, việc rà soát là không hiệu quả nếu đội xét duyệt không hiểu một cách đúng đắn các tài liệu trước khi việc rà soát diễn ra.

8.4 MÔ HÌNH CMM/CMMI

8.4.1 CMM và CMMI là gì?

CMM và CMMI là chuẩn quản lý quy trình chất lượng của các sản phẩm phần mềm được áp dụng cho từng loại hình công ty khác nhau. Hay nói cách khác đây là các phương pháp phát triển hay sản xuất ra các sản phẩm phần mềm.

Tháng 8/ 2006, SEI (Software Engineering Institute – Viện Công Nghệ Phần Mềm Mỹ) - tổ chức phát triển mô hình CMM/CMMI đã chính thức thông báo về phiên bản mới CMMI 1.2. Như vậy là sau gần 6 năm ban hành và sử dụng thay thế cho CMM (từ tháng 12/2001), CMMI phiên bản 1.1 (CMMI 1.1) đã được chính thức thông báo với lộ trình thời gian chuyển tiếp lên phiên bản mới CMMI 1.2.

Mặc dù số lượng các công ty phần mềm tại Việt Nam đạt được CMM/CMMI đến nay vẫn chưa nhiều, nhưng với sự khởi sắc trong lĩnh vực gia công và sản xuất phần mềm vài năm trở lại đây, sự cạnh tranh cũng như yêu cầu ngày càng cao của khách hàng, đã thúc đẩy các công ty xây dựng hệ thống quản lý chất lượng theo các mô hình quốc tế.

CMM và CMMI là một bộ khung (framework) những chuẩn đề ra cho một tiến trình sản xuất phần mềm hiệu quả, mà nếu như các tổ chức áp dụng nó sẽ mang lại sự khả dụng về mặt chi phí, thời gian biểu, chức năng và chất lượng sản phẩm phần mềm.

Mô hình CMM và mô tả các nguyên tắc, các thực tiễn nằm bên trong tính “thành thực” quá trình phần mềm và chủ ý giúp đỡ các công ty phần mềm hoàn thiện khả năng thuần thực quá trình sản xuất phần mềm, đi từ tự phát, hỗn độn tới các quá trình phần mềm thành thực, có kỷ luật.

Bằng việc thực hiện CMM các công ty thu được những lợi ích xác thực, giảm được rủi ro trong phát triển phần mềm và tăng được tính khả báo - do đó trở thành đối tác hay một nhà cung ứng hấp dẫn hơn đối với các khách hàng trên toàn thế giới. Tuy nhiên, CMM không phải không đòi hỏi chi phí. Những nguồn lực đáng kể của công ty phải được dành cho việc hướng tới các vùng tiến trình then chốt, cần thiết để lên từng bậc thang của chứng nhận CMM. CMM đưa ra một loạt các mức độ để biểu thị mức độ thành thực đã đạt được. Mức 1 ứng với mức độ thành thực thấp nhất và mức 5 ứng với mức độ thành thực cao nhất. Gần đây, SEI đã xúc tiến CMMI, một mô hình kế thừa CMM và CMMI hiện nay các công ty cũng đang bắt đầu triển khai việc sử dụng mô hình này.

8.4.2 Cấu trúc của CMM

a. Các level của CMM

CMM bao gồm 5 levels và 18 KPAs (Key Process Area) 5 levels của CMM như sau:

- 1: Initial, 2: Repeatable, 3: Defined, 4: Managed, 5: Optimising

Nói cách khác mỗi một level đều tuân theo một chuẩn ở mức độ cao hơn. Muốn đạt được chuẩn cao hơn thì các chuẩn của các level trước phải thỏa mãn. Mỗi level đều có đặc điểm chú ý quan trọng của nó cần các doanh nghiệp phải đáp ứng được

Level 1 thì không có KPAs nào cả

Level 2 : có 6 KPAs

Level 3: có 7 KPAs

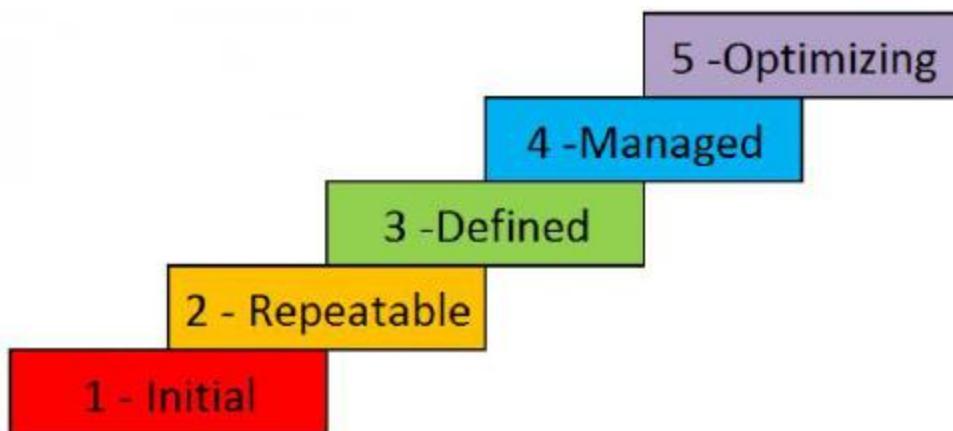
Level 4: có 2 KPAs

Level 5: có 3 KPAs

18 KPAs của CMM được đều có 5 thuộc tính (chức năng) chung trong đó có các qui định về key practice là những hướng dẫn về các thủ tục (procedure), qui tắc (polities), và hoạt động (activities) của từng KPA.

Để thực hiện KPA này ta cần phải thực hiện theo những qui tắc sau để bảo đảm đạt được KPA đó:

- (1) Commitment to Perform (Tam dịch là cam kết thực hiện)
- (2) Ability to Perform (Khả năng thực hiện)
- (3) Activities Performed (Các hoạt động lâu dài)
- (4) Measurement and Analysis (Khuân khổ và phân tích)
- (5) Verifying and Implementation



Hình 8.2: Các level (cấp độ) của CMM

b. Các level của CMM**Level 1**

- Level 1 là bước khởi đầu của CMM, mọi doanh nghiệp, công ty phần mềm, cá nhân, nhóm, cá nhân đều có thể đạt được. Ở level này CMM chưa yêu cầu bất kỳ tính năng nào. Ví dụ: không yêu cầu quy trình, không yêu cầu con người, miễn là cá nhân, nhóm, doanh nghiệp... đều làm về phần mềm đều có thể đạt tới CMM này.
- Đặc điểm của level 1:
 - Hành chính: Các hoạt động của lực lượng lao động được quan tâm hàng đầu nhưng được thực hiện một cách vã hắp tấp
 - Không thống nhất: Đào tạo quản lý nhân lực nhỏ lẻ chủ yếu dựa vào kinh nghiệm cá nhân
 - Quy trách nhiệm: Người quản lý mong bộ phận nhân sự điều hành và kiểm soát các hoạt động của lực lượng lao động
 - Quan liêu: Các hoạt động của lực lượng lao động được đáp ứng ngay mà không cần phân tích ảnh hưởng
 - Doanh số thường xuyên thay đổi: Nhân viên không trung thành với tổ chức

Level 2

- Có 6 KPAs bao gồm như sau:
 - Requirement Management (Lấy yêu cầu khách hàng, quản lý các yêu cầu đó)
 - Software Project Planning (Lập các kế hoạch cho dự án)
 - Software Project Tracking (Theo dõi kiểm tra tiến độ dự án)
 - Software SubContract Management (Quản trị hợp đồng phụ phần mềm)
 - Software Quality Assurance (Đảm bảo chất lượng sản phẩm)
 - Software Configuration Management (Quản trị cấu hình sản phẩm=> đúng yêu cầu của khách hàng không)
- Khi ta áp dụng Level 2, KPA 2 (Software Project Planning), ta sẽ có những common feature (đặc điểm đặc trưng) như sau:

- Mục tiêu(Goal): các hoạt động và những đề xuất của một dự án phần mềm phải được lên kế hoạch và viết tài liệu đầy đủ
 - Đề xuất/ Xem xét (Commitment): dự án phải tuân thủ theo các qui tắc của tổ chức khi hoạch định
 - Khả năng (Ability): 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
 - Đo lường (Measument): 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
 - Kiểm chứng(Verification): Các hoạt động khi lập kế hoạch dự án phải được sự xem xét của cấp senior manager
- Nói đến software quality,ắt hẳn các bạn QA không thể không biết đến tam giác "Quality Triangle" (Cost,Functionality,Schedule) => đúng chức năng , đúng chi phí, và đúng thời hạn. Chúng tôi đặt ra các tiêu chuẩn của qui trình mà từ đó chúng tôi sản xuất được phần mềm tốt nhất, chúng tôi thực hiện và chúng tôi có những người kiểm chứng và theo sát các hoạt động của nhóm thực hiện dự án sao cho đúng qui trình. Đó là QA Engineer, lâu nay vai trò của người QA trong các công ty phần mềm bị đồng hoá với vai trò của một tester hay còn gọi là QC - Quality Control.
- Để đạt được Level 2 thì người quản lý phải thiết lập được các nguyên tắc cơ bản và quản lý các hoạt động diễn ra. Họ có trách nhiệm quản lý đội ngũ của mình
- Các KPAs (Key Process Areas) của nó chú trọng tới các thành phần sau :
- Chế độ đãi ngộ
 - Đào tạo
 - Quản lý thành tích
 - Phân công lao động
 - Thông tin giao tiếp
 - Môi trường làm việc
- Sẽ có người hỏi để từ level 1 tiến tới level 2 cần có những gì:

- Trả lời:

Trước tiên nó phải thỏa mãn các điều kiện ở level1, hiển nhiên rồi. Tiếp theo là phải chú trọng tới các phần sau:

1. *Môi trường làm việc*: đảm bảo điều kiện làm việc, tạo hứng thú trong công việc, không bị ảnh hưởng, mất tập trung bởi các nhân tố khác.
2. *Thông tin*: Xây dựng cơ chế truyền tin thông suốt từ trên xuống dưới và ngược lại nhằm giúp 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
3. *Xây dựng đội ngũ nhân viên*: Ngay từ khâu tuyển dụng, lựa chọn kỹ càng và định hướng, thể chế hóa quy trình tuyển dụng
4. *Quản lý thành tích*: 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.
5. *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 đội, nhóm, kỹ năng quản lý... nhằm tạo cơ hội cho người lao động phát huy khả năng, cơ hội học hỏi và phát triển bản thân.
6. *Chế độ đãi ngộ*: Hoạch định chiến lược đãi ngộ, thu thập ý kiến lực lượng lao động và công bố công khai. Chế độ đãi 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, cống 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

- Các vùng tiến trình chủ chốt ở mức 3 nhằm vào cả hai vấn đề về dự án và tổ chức, vì một tổ chức (công ty) tạo nên cấu trúc hạ tầng thể chế các quá trình quản lý và sản xuất phần mềm hiệu quả qua tất cả các dự án. Chúng gồm có Tập trung Tiến trình Tổ chức (Organization Process Focus), Phân định Tiến trình Tổ chức (Organization Process Definition), Chương trình Đào tạo (Training Program), Quản

trị Phần mềm Tích hợp (Integrated Software Management), Sản xuất Sản phẩm Phần mềm (Software Product Engineering), Phối hợp nhóm (Intergroup Coordination), và Xét duyệt ngang hàng (Peer Reviews).

- Để đạt được level 3 thì người quản lý phải biến đổi cải tiến các hoạt động đang diễn ra, cải tiến môi trường làm việc.
- Lực lượng lao động sở hữu những kiến thức, kỹ năng cốt lõi 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 lên Level 3: Các KPA cần thực hiện
 - 1. *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.
 - 2. *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.
 - 3. *Phát triển sự nghiệp:* Tạo điều kiện cho mỗi cá nhân phát triển nghề nghiệp và có cơ hội thăng tiến trong nghề nghiệp, nó bao gồm: thảo luận về lựa chọn nghề nghiệp với mỗi cá nhân, xác định các cơ hội, theo dõi sự tiến bộ trong mục tiêu công việc, giao quyền và khuyến khích thực hiện những mục tiêu trong công việc.
 - 4. *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 chế độ phúc lợi, đãi ngộ dựa trên hiệu quả...

giúp bảo đảm rằng mọi hoạt động của tổ chức đều xuất phát từ mục đích phục vụ cho phát triển nguồn nhân lực

5. *Văn hóa cá thể*: Tạo lập được cơ chế 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, giúp nhau cùng tiến bộ. Trao quyền thúc đẩy nhân viên tham gia ý kiến, ra quyết định

Level 4

- 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 Quản lý quá trình định lượng (Quantitative Process Management) và Quản lý chất lượng phần mềm (Software Quality Management)
- 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
 - Cỗ 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.
- Level 4 này sẽ chú trọng vào những người đứng đầu của một công ty, họ có khả năng quản lý các công việc như thế nào.

Level 5

- Các vùng tiến trình chủ yếu ở mức 5 bao trùm các vấn đề mà cả tổ chức và dự án phải nhắm tới để thực hiện hoàn thiện quá trình sản xuất phần mềm liên tục, đo đếm được. Đó là Phòng ngừa lỗi (Defect Prevention), Quản trị thay đổi công nghệ (Technology Change Management), và Quản trị thay đổi quá trình (Process Change Management).

- Để đạt được Level 5 thì doanh nghiệp đó phải liên tục cải tiến hoạt động tổ chức, tìm kiếm các phương pháp đổi mới để nâng cao năng lực làm việc của lực lượng lao động trong tổ chức, hỗ trợ các nhân phát triển sở trường chuyên môn.
- Chú trọng vào việc quản lý, phát triển năng lực của nhân viên.
- Huấn luyện nhân viên trở thành các chuyên gia.

8.4.3 So sánh giữa CMM và CMMi

Nếu đặc tả một cách trực quan thì ta thấy CMM và CMMI chỉ khác nhau có một chữ "I" (Integration). Chữ "I" đó tạo ra sự khác biệt đáng kể giữa CMM và CMMI.

Trước tiên hãy xem qua nguồn gốc của hai định nghĩa này. Nếu nói rằng CMM ra đời trước CMMI thì cũng đúng nhưng mà nói CMMI có trước khi CMM ra đời cũng chẳng sai. Thật ra khi CMM được chính thức công bố vào cuối năm 1990 thì CMMI đã được manh mún được nhắc đến từ nhiều năm trước đó (chính xác hơn là từ 1979-Crosby's maturity grid (Quality is Free)) thông qua cấu trúc Continuous & Staged. Có thể nói CMMI là một phiên bản cải thiện tất yếu của CMM.

Trong khi CMM được hoàn thiện và phát triển bởi viện SEI của Mỹ, thì CMMI là sản phẩm của sự cộng tác giữa viện này và chính phủ Mỹ. Từ khi CMM được công nhận và áp dụng trên thế giới thì tầm quan trọng của nó đã vượt qua giới hạn của một viện khoa học. Với tốc độ phát triển không ngừng và đòi hỏi sự cải thiện liên tục trong ngành công nghệ thông tin, việc chính phủ Mỹ cùng với viện SEI kết hợp để hoàn thiện CMM và cho ra đời phiên CMMI là một hệ quả tất yếu.

Mô hình CMM trước đây gồm có 5 mức: khởi đầu, lặp lại được, được định nghĩa, được quản lý và tối ưu. Một điểm đặc biệt là mỗi doanh nghiệp có thể áp dụng mô hình CMM ở bất kỳ mức nào mà không cần tuân theo bất kỳ một qui định nào, không cần phải đạt mức thấp trước rồi mới có thể đạt mức cao (có thể đi thẳng lên mức cao, hoặc cũng có thể tự hạ xuống mức thấp hơn). Về nguyên tắc, SEI không chính thức đứng ra công nhận CMM mà thông qua các tổ chức tư vấn, các đánh giá trưởng được SEI ủy quyền và thừa nhận.

Từ cuối 2005, SEI không tổ chức huấn luyện SW-CMM và chỉ thừa nhận các đánh giá theo mô hình CMMi mới từ tháng 12/2005. CMMi được tích hợp từ nhiều mô hình

khác nhau, phù hợp cho cả những doanh nghiệp phần cứng và tích hợp hệ thống, chứ không chỉ đơn thuần áp dụng cho doanh nghiệp sản xuất phần mềm như CMM trước đây. Có 4 mô hình áp dụng CMMI là CMMI-SW (dành cho công nghệ phần mềm), CMMI-SE/SW (dành cho công nghệ hệ thống và phần mềm), CMMI- SE/SW/IPPD (dành cho công nghệ hệ thống + công nghệ phần mềm với việc phát triển sản phẩm và quy trình tích hợp), CMMI-SE/SW/IPPD/SS (dành cho công nghệ hệ thống + công nghệ phần mềm với việc phát triển sản phẩm và quy trình tích hợp có sử dụng thầu phụ). Có 2 cách diễn đạt và sử dụng CMMI: Staged (phù hợp cho tổ chức có trên 100 người) và Continuos (phù hợp cho tổ chức dưới 40 người). CMMI cũng bao gồm 5 mức như CMM: khởi đầu, lặp lại được, được định nghĩa, được quản lý và tối ưu.

CMMI đưa ra cụ thể các mô hình khác nhau cho từng mục đích sử dụng có đặc riêng bao gồm:

- CMMI-SW mô hình chỉ dành riêng cho phần mềm.
- CMMI-SE/SW mô hình tích hợp dành cho các hệ thống và kỹ sư phần mềm.
- CMMI-SE/SW/IPPD mô hình dành cho các hệ thống, kỹ sư phần mềm và việc tích hợp sản phẩm cùng quá trình phát triển nó.

Như vậy các đặc điểm khác nhau quan trọng nhất giữa CMM và CMMI là:

- CMMI đưa ra cụ thể hơn 2 khái niệm đối ứng staged VS continuous
- Chu kỳ phát triển của CMMI được phát triển sớm hơn.
- Nhiều khả năng tích hợp (Integration) hơn
- Sự đo lường, định lượng được định nghĩa là một Process area (vùng tiến trình). Chứ không chỉ là một đặc trưng cơ bản.
- Bao hàm nhiều Process Areas hơn.
- Đạt được thành quả dễ dàng hơn với mỗi Process area.

8.4.4 Lợi ích của CMM đem lại cho doanh nghiệp

- a. Viễn cảnh mà CMM mang lại

Ý nghĩa của việc áp dụng những nguyên tắc:

- Quản lý chất lượng tổng thể
 - Quản lý nguồn nhân lực
 - Phát triển tổ chức
 - Tính cộng đồng
 - Phạm vi ảnh hưởng rộng: từ các ngành công nghiệp đến chính phủ
 - Hoàn toàn có thể xem xét và mở rộng tầm ảnh hưởng với bên ngoài
 - Chương trình làm việc nhằm cải tiến, nâng cao hoạt động của đội ngũ lao động
 - Đánh giá nội bộ
 - Các hoạt động của đội ngũ lao động được cải tiến
 - Các chương trình nhằm nâng cao năng lực, hiệu quả công việc luôn được tổ chức
- b. Mục tiêu chiến lược
- Cải tiến năng lực của các tổ chức phần mềm bằng cách nâng cao kiến thức và kỹ năng của lực lượng lao động
 - Đảm bảo rằng năng lực phát triển phần mềm là thuộc tính của tổ chức không phải của một vài cá thể
 - Hướng các động lực của cá nhân với mục tiêu tổ chức
 - Duy trì tài sản con người, duy trì nguồn nhân lực chủ chốt trong tổ chức
- c. Lợi ích CMM mang lại cho Doanh nghiệp gói gọn trong 4 từ: Attract, Develop, Motivate và Organize
- d. Lợi ích CMM mang lại cho người lao động:
- Môi trường làm việc, văn hóa làm việc tốt hơn
 - Vạch rõ vai trò và trách nhiệm của từng vị trí công việc
 - Đánh giá đúng năng lực, công nhận thành tích
 - Chiến lược, chính sách đãi ngộ luôn được quan tâm
 - Có cơ hội thăng tiến
 - Liên tục phát triển các kỹ năng cốt yếu.

TÓM LƯỢC

Bài học này cung cấp các kiến thức về:

- Quản lý chất lượng và các hoạt động đảm bảo của chất lượng
- Lập kế hoạch chất lượng và kiểm soát chất lượng.
- Các chuẩn mực trong quá trình quản lý chất lượng.
- Mô hình CMM / CMMi

BÀI 9: QUẢN LÝ CẤU HÌNH

Nội dung gồm các phần sau:

- Quản lý cấu hình
- Lập kế hoạch quản lý cấu hình
- Quản lý phiên bản và phát hành
- Quản lý sự thay đổi
- Xây dựng hệ thống
- Các công cụ CASE cho quản lý cấu hình

9.1 KẾ HOẠCH QUẢN LÝ CẤU HÌNH

Quản lý cấu hình (Configuration Management - CM) là một quy trình để quản lý, tổ chức và kiểm soát một cách có hệ thống những thay đổi trong tài liệu, mã và các thực thể khác trong vòng đời phát triển phần mềm (Software Development Life Cycle – SDLC). Mục tiêu chính là tăng hiệu quả với những sai lầm tối thiểu. CM là một phần của lĩnh vực quản lý cấu hình liên ngành và nó có thể xác định chính xác ai đã thực hiện bản sửa đổi nào.

Kế hoạch quản lý cấu hình:

1. Xác định các loại tài liệu được quản lý và sơ đồ đặt tên tài liệu.
2. Xác định người chịu trách nhiệm về các thủ tục CM và việc tạo ra các đường cơ sở.
3. Xác định các chính sách để kiểm soát thay đổi và quản lý phiên bản.
4. Mô tả các công cụ nên được sử dụng để hỗ trợ quá trình và bất kỳ hạn chế nào trong việc sử dụng chúng.
5. Xác định cơ sở dữ liệu quản lý cấu hình được sử dụng để ghi lại thông tin cấu hình.

Bất kỳ phần mềm quản lý thay đổi nào cũng phải có 3 tính năng chính sau:

- Quản lý đồng thời: Khi cùng một tập tin được nhiều người chỉnh sửa cùng một lúc.
- Kiểm soát phiên bản: Lưu mọi thay đổi được thực hiện vào tệp, có thể quay lại phiên bản trước trong trường hợp có vấn đề.

9.2 QUẢN LÝ VIỆC THAY ĐỔI

Quản lý thay đổi là một thủ tục đảm bảo chất lượng và tính nhất quán của phần mềm khi các thay đổi được thực hiện trên cấu hình phần mềm. Quản lý sự thay đổi liên quan tới việc theo dõi các thay đổi và đảm bảo rằng chúng được thực hiện theo cách hiệu quả nhất về chi phí.

Ai là người đưa ra các yêu cầu thay đổi đối với hệ thống?

- Các yêu cầu thay đổi đối với hệ thống phần mềm có thể bắt nguồn từ: Người dùng, Nhà phát triển, Áp lực thị trường.

Biểu mẫu yêu cầu thay đổi:

- Sự định nghĩa của một biểu mẫu yêu cầu thay đổi là một phần của quy trình lập kế hoạch CM.
- Biểu mẫu này lưu sự thay đổi được đề nghị, người yêu cầu thay đổi, lý do tại sao sự thay đổi này được đề nghị và tính cấp bách của sự thay đổi.
- Nó còn lưu ước lượng về sự thay đổi, phân tích ảnh hưởng, chi phí thay đổi và các đề nghị.

Các công cụ theo dõi sự thay đổi:

- Một vấn đề chính trong quản lý sự thay đổi là theo dõi trạng thái của sự thay đổi.
- Các công cụ theo dõi sự thay đổi theo dõi trạng thái của từng yêu cầu thay đổi và đảm bảo rằng các yêu cầu thay đổi được gửi tới đúng người, đúng thời điểm.
- Được tích hợp với các hệ thống e-mail để cho phép sự phân phát các yêu cầu thay đổi điện tử.

Ban kiểm soát sự thay đổi:

- Các thay đổi nên được xem lại bởi một nhóm bên ngoài những người quyết định xem chúng có mang lại lợi nhuận hay không theo quan điểm chiến lược và tổ chức hơn là theo quan điểm kỹ thuật.
- Ban kiểm soát sự thay đổi nên là một nhóm độc lập của dự án.
- Ban kiểm soát sự thay đổi có thể gồm một đại diện cấp cao từ phía khách hàng và nhân viên đấu thầu.

Lịch sử tiến hóa:

- Là hồ sơ về các thay đổi được áp dụng cho các thành phần mã lệnh.
- Cấu hình sẽ được lưu (những nét chính) sự thay đổi được tạo ra, mỗi quan hệ đối với sự thay đổi, ai tạo ra sự thay đổi và khi nào nó được thực hiện.
- Cấu hình có thể được xem như một chú thích trong mã lệnh. Nếu một mẫu phần mở đầu chuẩn được sử dụng cho lịch sử tiến hóa, các công cụ có thể xử lý nó một cách tự động.

9.3 QUẢN LÝ PHIÊN BẢN VÀ BẢN PHÁT HÀNH

9.3.1 Quản lý phiên bản

Các bước tiến hành quản lý phiên bản:

- Phát triển một sơ đồ định danh cho các phiên bản của hệ thống
- Lập kế hoạch khi một phiên bản của hệ thống mới được tạo ra
- Đảm bảo rằng các công cụ và thủ tục quản lý phiên bản được áp dụng một cách đúng đắn.
- Lập kế hoạch phân phối các phát hành của hệ thống mới.

Phiên bản / Biến thể / Phát hành:

- Phiên bản (Version): Một thể hiện của hệ thống mà nó khác biệt chức năng với các thể hiện khác của hệ thống theo cách nào đó.
- Biến thể (Variant): Một thể hiện của hệ thống mà nó giống về chức năng nhưng khác về phi chức năng với các thể hiện khác của hệ thống.

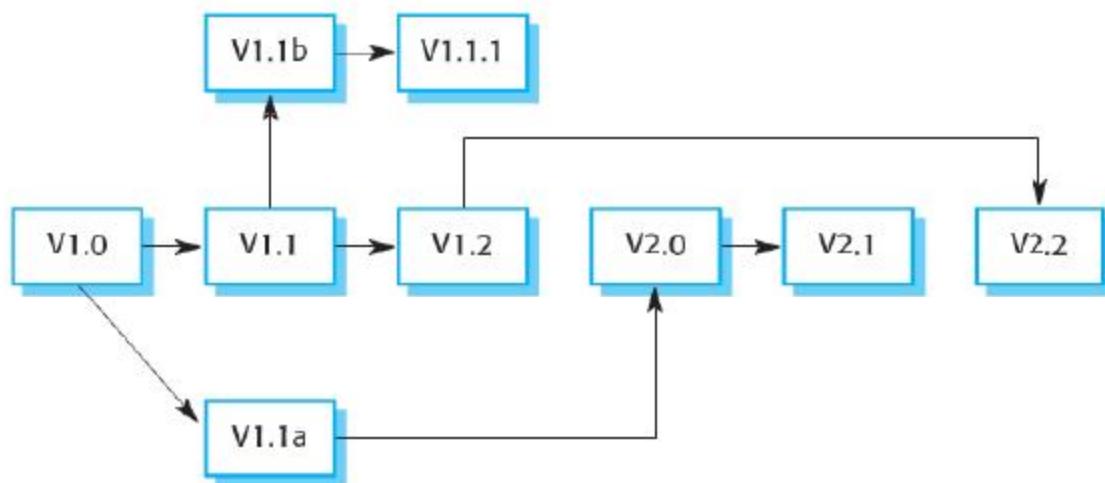
- Phát hành (Release): Một thể hiện của hệ thống mà nó được phân phối cho người dùng bên ngoài nhóm phát triển.

Xác minh phiên bản:

- Các thủ tục xác minh phiên bản nên định nghĩa một cách rõ ràng việc nhận dạng các phiên bản thành phần.
- Ba kỹ thuật cơ bản để xác minh thành phần: Đánh số phiên bản, Xác minh dựa trên thuộc tính, Xác minh hướng tới sự thay đổi.

Đánh số phiên bản:

- Sơ đồ đánh số đơn giản nhất sử dụng sự tiến hóa tuyến tính: V1, V1.1, V1.2, V2.1, v.v
- Cấu trúc tiến hóa thực tế là một cây hay một mạng hơn là một sự liên tục.
- Sơ đồ đặt tên phân cấp đưa đến ít lỗi hơn trong việc xác minh phiên bản.



Hình 9.1: Cấu trúc tiến hóa của phiên bản

Xác minh dựa trên thuộc tính:

- Các thuộc tính có thể được sử dụng để nhận dạng phiên bản. Các thuộc tính có thể là ngày, người tạo ra, ngôn ngữ lập trình, khách hàng, trạng thái, v.v. Tập các thuộc tính phải được chọn để tất cả các phiên bản có thể được định danh duy nhất. Trong thực tiễn, một phiên bản còn cần một tên kết hợp để tham khảo dễ dàng.
- Một thuận lợi quan trọng của xác minh dựa trên thuộc tính là nó có thể hỗ trợ các truy vấn. Truy vấn chọn ra một phiên bản phụ thuộc vào các giá trị thuộc tính.

Nhận dạng hướng tới sự thay đổi:

- Tích hợp các phiên bản và các thay đổi được thực hiện để tạo ra các phiên bản đó
- Được sử dụng cho hệ thống hơn là cho thành phần.
- Mỗi một thay đổi hệ thống được đề nghị có một tập thay đổi kết hợp mà nó mô tả các thay đổi được thực hiện cho các thành phần của hệ thống.

9.3.2 Quản lý phát hành

Phát hành hệ thống là một phiên bản của hệ thống mà nó được phân phối tới khách hàng. Nhà cung cấp sản phẩm phần mềm thường chỉ đưa ra các phát hành mới cho các nền mới hay thêm các chức năng mới rất cần thiết. Các hệ thống hiện nay thường được phát hành trên đĩa quang hoặc các tập tin cài đặt có thể tải xuống từ trang web.

Các vấn đề phát hành:

- Khách hàng có thể không muốn bản phát hành mới của hệ thống.
- Quản lý phát hành không nên giả sử rằng tất cả các phát hành trước đó được chấp nhận. Tất cả những tập tin cần cho một phát hành nên được tái tạo khi một phát hành mới được cài đặt.

Đưa ra quyết định phát hành:

- Việc chuẩn bị và phân phối một bản phát hành hệ thống là một quy trình tốn kém.
- Các yếu tố như chất lượng kỹ thuật và tổ chức tác động đến việc quyết định khi nào đưa ra một phát hành của hệ thống mới.

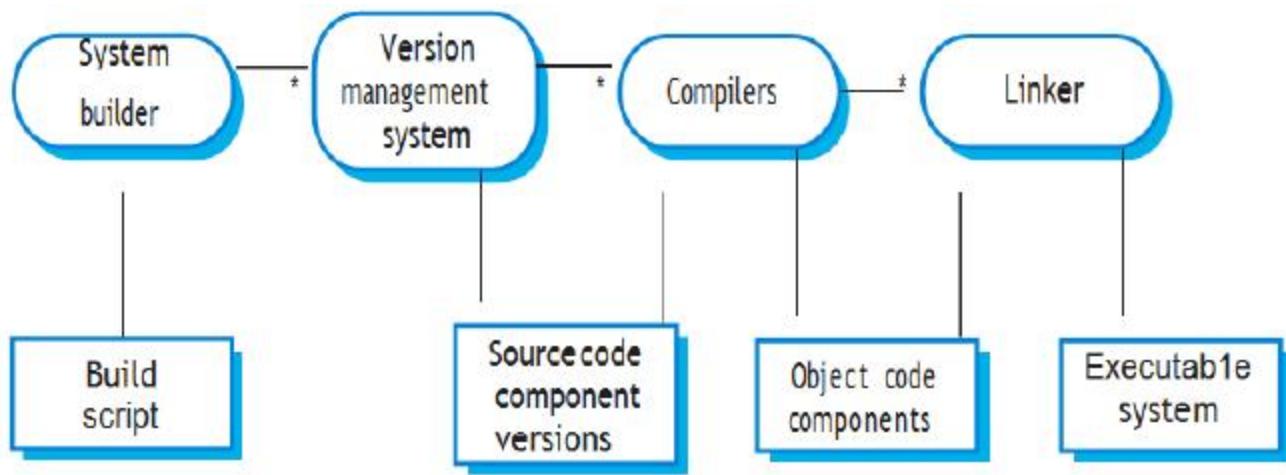
Những thành phần đi kèm khi phát hành một hệ thống:

- Phát hành hệ thống không chỉ là một tập các chương trình có thể thực thi được mà có thể bao gồm: các tập tin cấu hình định nghĩa cách thức phát hành được cấu hình cho một sự cài đặt cụ thể, các tập tin dữ liệu cần cho sự vận hành hệ thống, một chương trình cài đặt hay một script tiện ích để cài đặt hệ thống lên phần cứng, các tư liệu ở dạng giấy hay dạng điện tử, đóng gói và quảng cáo liên quan.

- Tư liệu hóa sự phát hành: Ghi lại các phiên bản cụ thể của các bộ phận mã nguồn được sử dụng để tạo ra mã thực thi; Lưu bản sao của mã nguồn, mã thực thi, tất cả dữ liệu và các tập tin cấu hình; Ghi lại phiên bản của hệ điều hành, thư viện, bộ biên dịch và những công cụ được sử dụng để xây dựng phần mềm.

9.4 XÂY DỰNG HỆ THỐNG

Xây dựng hệ thống là quy trình biên dịch và liên kết các bộ phận phần mềm vào một chương trình mà nó thực hiện trên một cấu hình đích cụ thể. Các hệ thống khác nhau được xây dựng từ các kết hợp khác nhau về các bộ phận phần mềm. Quy trình này hiện nay luôn được hỗ trợ bởi các công cụ tự động.



Hình 9.2: Quy trình xây dựng hệ thống

9.5 CÁC CÔNG CỤ CASE CHO QUẢN TRỊ CẤU HÌNH

9.5.1 GitHub

Git/GitHub là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System). Hiểu nôm na rằng Git là 1 hệ thống giúp cho việc quản lý tài liệu, source code... của 1 nhóm các developer cùng làm chung dự án. Git sẽ ghi nhớ lại toàn bộ lịch sử thay đổi của source code trong dự án. Bạn sửa file nào, thêm dòng code nào, xóa dòng code nào, bỏ thừa dấu ở đâu... tất cả các hành động đều được Git ghi lại. Qua đó giúp dự án có thể điều tra nguyên nhân gây lỗi hệ thống, tổng hợp code trở nên dễ dàng hơn.

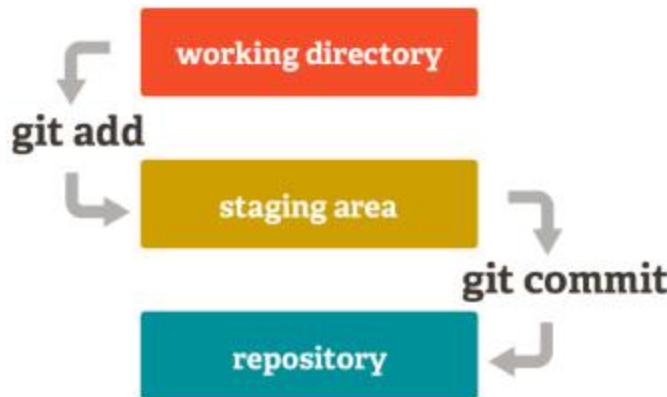
Trước khi đi vào sử dụng git ta cần hiểu một số khái niệm liên quan đến Git như sau:

Repository: Repository hiểu đơn giản nó chính là cái kho lưu trữ tất cả những thông tin cần thiết để quản lý các sửa đổi và lịch sử của toàn bộ project. Repository của Git được phân thành 2 loại là remote repository và local repository.

- Local Repository: là repository nằm trên chính máy tính của chúng ta, repository này sẽ đồng bộ hóa với remote repository bằng các lệnh của git.
- Remote Repository: là repository được cài đặt trên server chuyên dụng. Ví dụ: GitHub, GitLab, Bitbucket,...

=> **GitHub** chính là 1 Remote Repository lưu trữ tất cả những thông tin cần thiết để quản lý các sửa đổi và lịch sử của toàn bộ project.

Working tree và Index (hoặc staging area): Là những thư mục được đặt trong sự quản lý của Git, nơi mọi người thực hiện công việc trên đó, được gọi là **working tree**. Giữa repository và working tree tồn tại một nơi gọi là index hay staging area . staging area là nơi để chuẩn bị cho việc commit vào repository.



Hình 9.3: Working tree

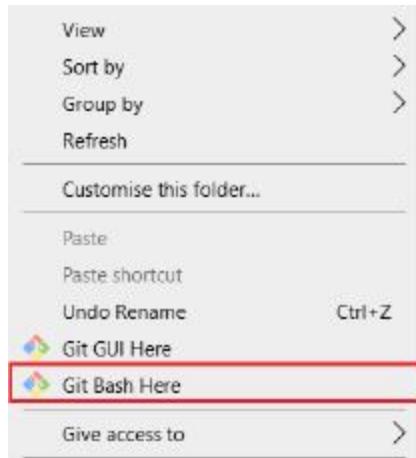
Download GitHub theo địa chỉ: <https://github.com/> và cài đặt theo hướng dẫn.

Một số lệnh cơ bản thao tác với local repository:

Lệnh: **git init**

- Tác dụng : Khởi tạo 1 git repository 1 project mới hoặc đã có.

- Cách dùng: Tạo 1 folder mới => vào trong folder đó => click chuột phải chọn Git Bash Here như hình dưới:



- Cửa sổ console git bash hiện lên => gõ lệnh **git init**

```
MINGW64/d/STUDY/TestGit
$ git init
Initialized empty Git repository in D:/STUDY/TestGit/.git/
$ |
```

- Sau khi tạo thành công thì trong folder sẽ xuất hiện folder .git => folder này sẽ chứa tất cả những thông tin cần thiết để quản lý các sửa đổi và lịch sử của toàn bộ project.

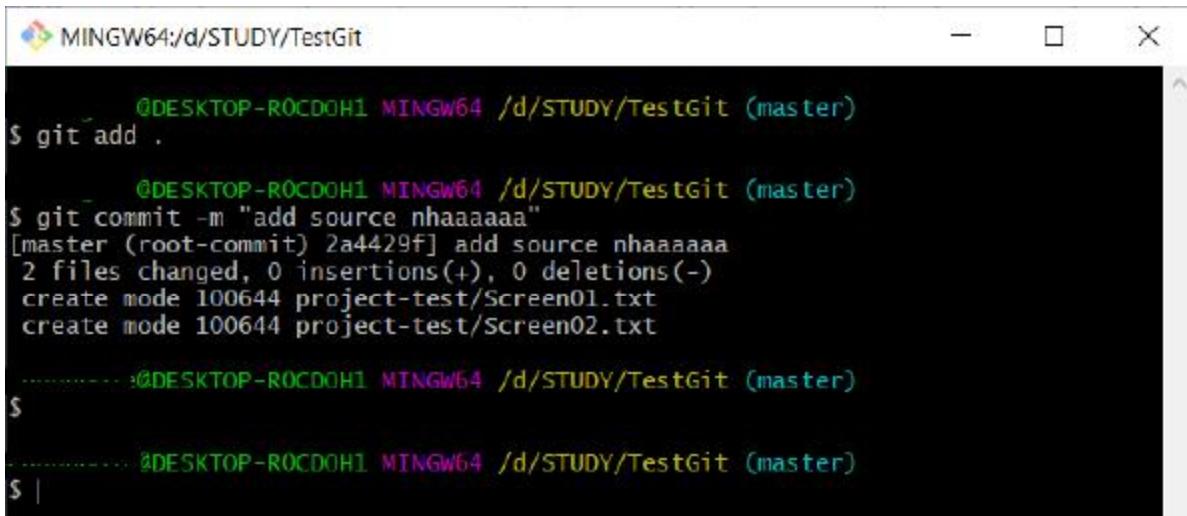
Lệnh: **git add**

- Tác dụng : Thêm thay đổi vào stage/index trong thư mục làm việc.
- Cách dùng: Tại thư mục làm việc => **git add .**
- Khi add thành công

```
MINGW64/d/STUDY/TestGit
$ git add .
$ |
```

Lệnh: **git commit**

- Tác dụng: commit là một action để Git lưu lại các sự thay đổi trong thư mục làm việc vào repository
- Cách dùng: **git commit -m "add source nhaaaaaa"**
- Khi commit thành công



```

MINGW64:/d/STUDY/TestGit
$ git add .

@DESKTOP-R0CDOH1 MINGW64 /d/STUDY/TestGit (master)
$ git commit -m "add source nhaaaaaa"
[master (root-commit) 2a4429f] add source nhaaaaaa
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 project-test/Screen01.txt
 create mode 100644 project-test/Screen02.txt

@DESKTOP-R0CDOH1 MINGW64 /d/STUDY/TestGit (master)
$ |
```

9.5.2 Team Foundation Server

Team Foundation Server (TFS) là một chương trình server được sử dụng để quản lý mã nguồn của các lập trình viên trong các dự án chung.

Với các tính năng nổi bật như:

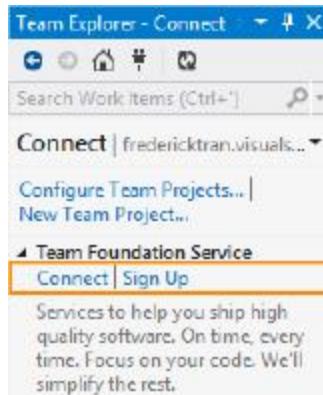
- Lưu trữ mã nguồn online.
- Tự động tổng hợp và đưa ra so sánh về mã nguồn từ các phiên bản được upload.
- Lưu trữ các phiên bản của mã nguồn và cho phép tải lại khi cần.
- Quản lý thay đổi trong project.

TFS được tích hợp sẵn trong Visual Studio Ultimate, có thể download bản cài tại <http://www.visualstudio.com/en-us/downloads>

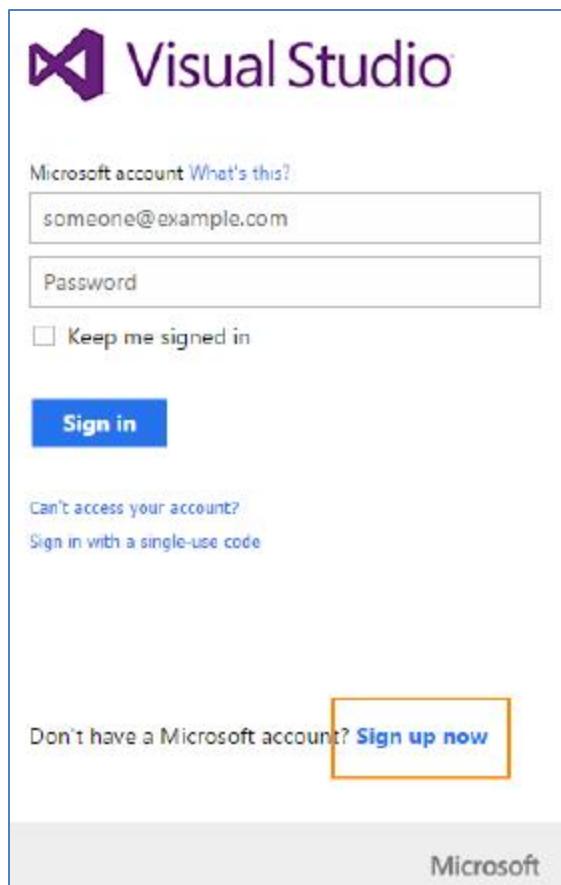
Bước 1: Đăng ký tài khoản trên Visual Studio Online

- Khởi động Visual Studio.
- Tìm trên thanh Menu, vào TEAM → Connect to Team Foundation Server... hoặc VIEW → Team Explorer.

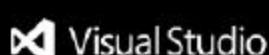
- Lúc này cửa sổ Team Explorer sẽ hiện lên, nếu chưa có tài khoản Microsoft, click vào Sign Up.



- Đăng nhập vào Visual Studio Online của Microsoft, bấm vào Sign up now để đăng ký tài khoản mới.



- Điền các thông tin cần thiết. ZIP Code hay Postal Code là mã bưu chính nơi đang sống, cụ thể có thể tra tại trang <http://postcode.vnpost.vn>
- Sau khi login vào trang Visual Studio Online bằng tài khoản Microsoft.



Create a Visual Studio Online Account

Account URL * [?](#)

[Create Account](#)

By clicking **Create Account**, you agree to the [Terms of Service](#) and [Privacy Statement](#).

- Nhập địa chỉ server vào khung và bấm Create Account.
- Một trang web mới hiện ra với đường dẫn `https://<tenserver>.visualstudio.com` báo hiệu đã thành công.

Welcome. Your account, <https://tranminhtuan.visualstudio.com/>, has been created and is ready to go. The next step is to create your first team project where you'll host your code and backlog. [Learn more](#)

Project name:

Description:

Version control: [Team Foundation Version Control](#) [Learn more](#) [Git](#) [Learn more](#)

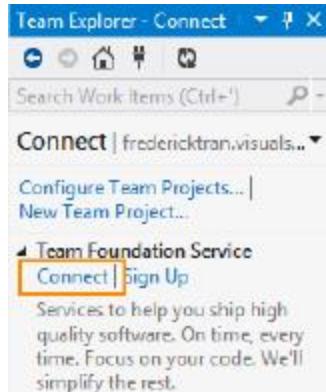
Process template: [Microsoft Visual Studio Scrum 2013.3](#) [Learn more](#)

[Create project](#)

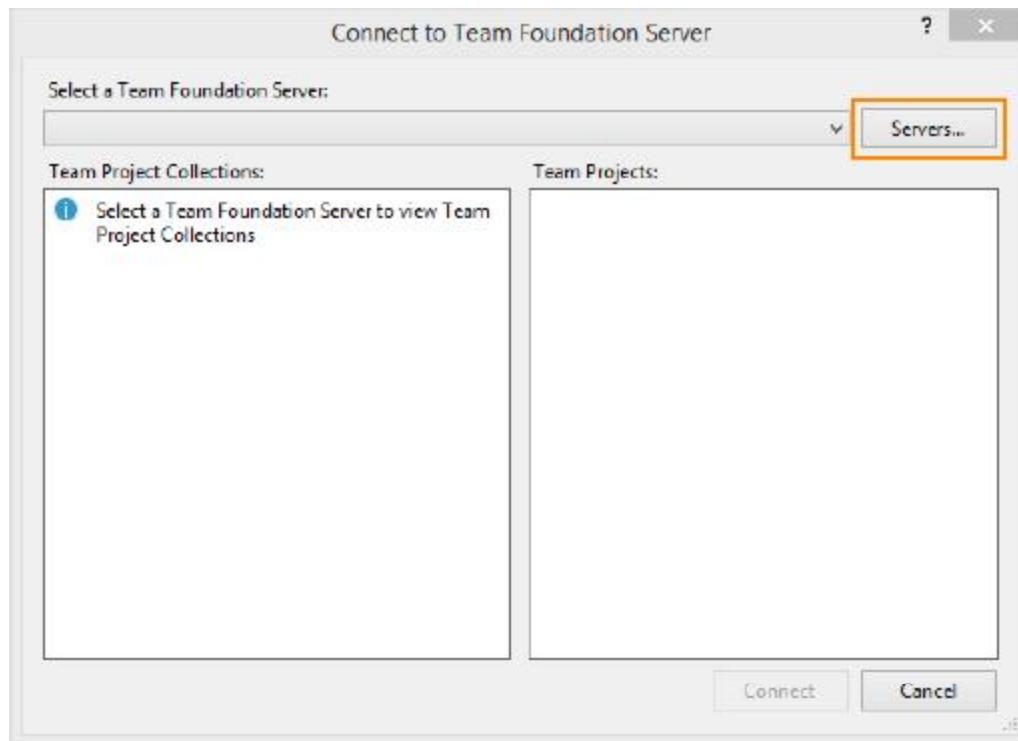
- Các lập trình viên trong nhóm sẽ cần đường dẫn này để tham gia thao tác trên project tại server.
- Đăng nhập vào trang này cho phép xem các thống kê về các project trong server và sử dụng một số công cụ về quản lý project.

Bước 2: Kết nối đến server

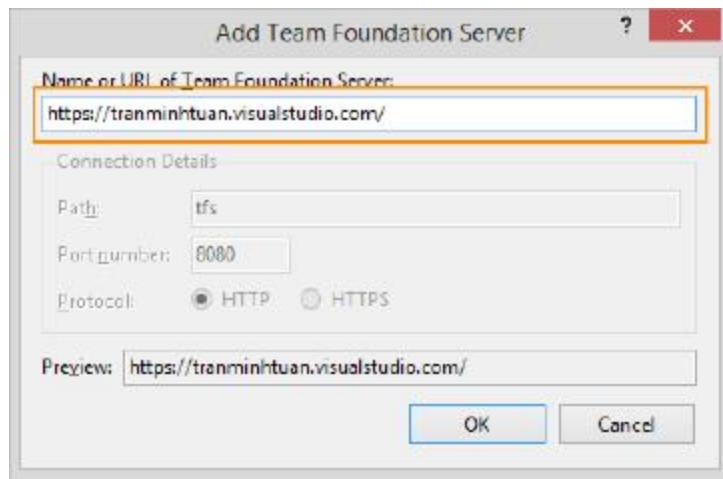
- Quay lại với Visual Studio, tại cửa sổ Team Explorer hãy click vào Connect.



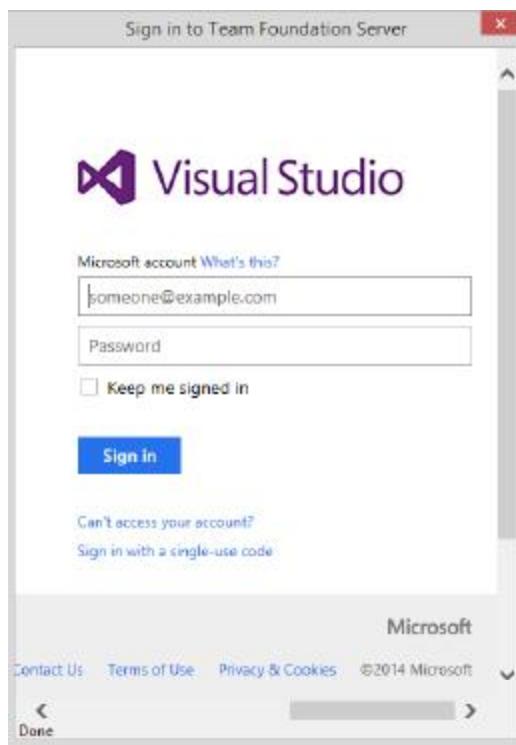
- Cửa sổ Connect to Team Foundation Project liệt kê ra các Team Project và Project có thể thao tác.
- Do chưa từng kết nối đến server nào nên danh sách hoàn toàn trống. Click vào Servers...



- Tại cửa sổ mới bật lên, click vào Add...
- Một cửa sổ khác lại bật lên, tại khung Name or URL of Team Foundation Server: điền vào địa chỉ server, bấm OK.



- Quay lại cửa sổ trước đó, bấm Close để kết thúc chọn server. VS sẽ lại bật lên một cửa sổ mới và yêu cầu đăng nhập tài khoản Microsoft.

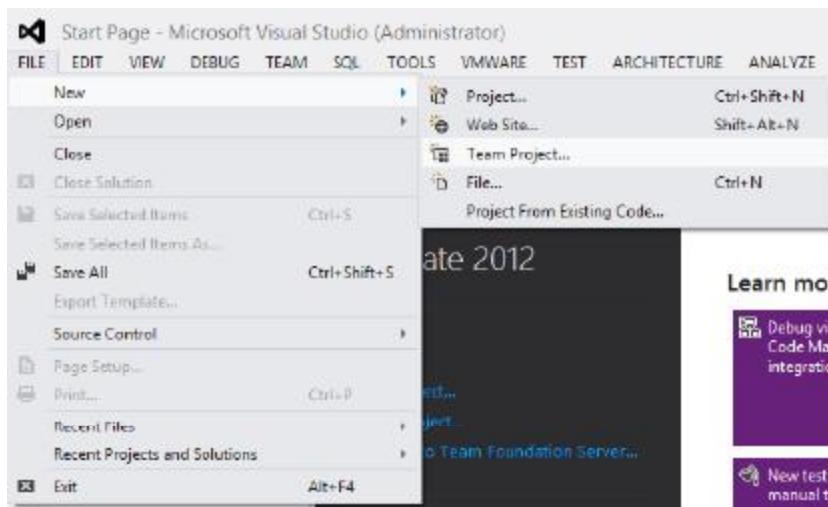


- Bước xác nhận có được truy cập vào server hay không.
- Điền tài khoản và mật khẩu vào khung đăng nhập và nhấn Sign in.

Bước 3: Tạo và quản lý Team project

a. Tạo Team Project

- Để tạo một project mới, phải tạo một Team Project. Vào File → New → Team Project hoặc bấm vào New Team Project... trên giao diện của Team Explorer.



- Giao diện tạo Team Project như sau sẽ hiện ra, cấu hình cho project theo hướng dẫn của trang.

Create your first project

Welcome. Your account, <https://tranminhtuan.visualstudio.com/>, has been created and is ready to go. The next step is to create your first team project where you'll host your code and backlog. [Learn more](#)

Project name: Stdio

Description: Testing description.

Version control: [Team Foundation Version Control](#) [Git](#)

Process template: [Microsoft Visual Studio Scrum 2013.3](#)

[Create project](#)

- Sử dụng Team Foundation Version Control (TFVC). Tạo Team Project thành công.

Your new project, Stdio, is ready!

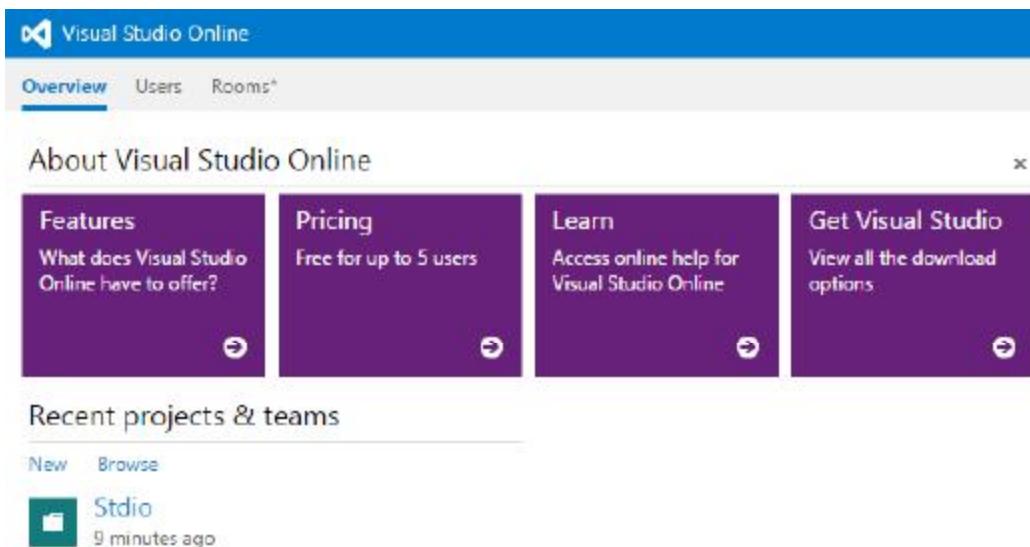
Are you ready to write some code? Get started by connecting your project directly to Visual Studio, or learn how to connect with [XCode](#), or [Eclipse](#). We think you and your team are going to absolutely love this.

[Open with Visual Studio to connect](#)

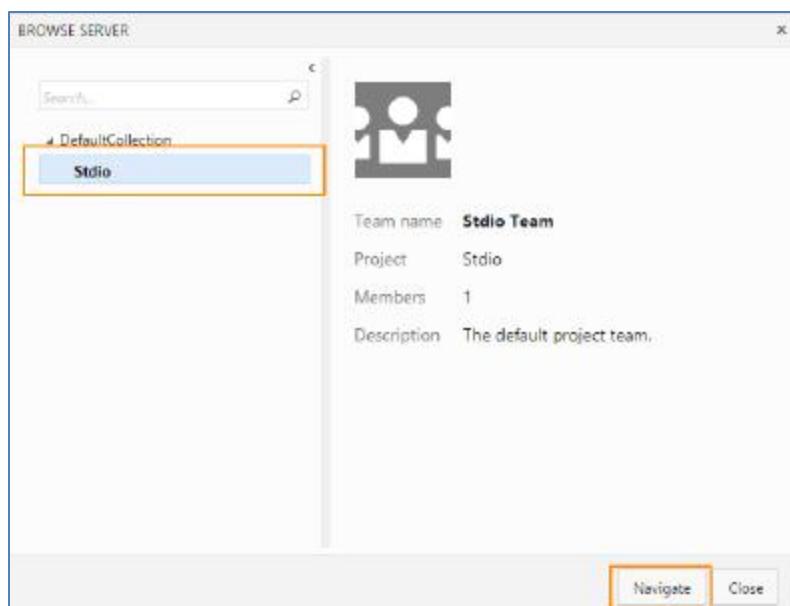
Requires Visual Studio 2013

b. Quản lý Team Project

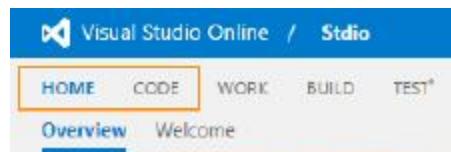
- Để đến trang quản lý của team project vừa tạo:
- Quay lại trang chủ server tại <https://<tenserver>.visualstudio.com>
- Click vào Browse.



- Trong giao diện mới bật lên, chọn project vừa tạo và bấm Navigate.

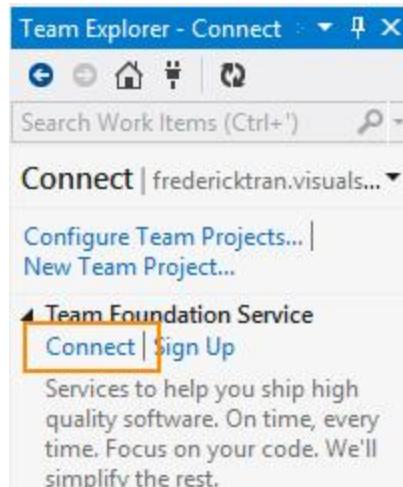


- Trình duyệt web sẽ chuyển hướng đến trang quản lý của project.
- Ở đây sẽ hiện ra các thông kê và thông số tổng quan của project (menu HOME), hay danh sách các file mã nguồn có trong project (menu CODE).

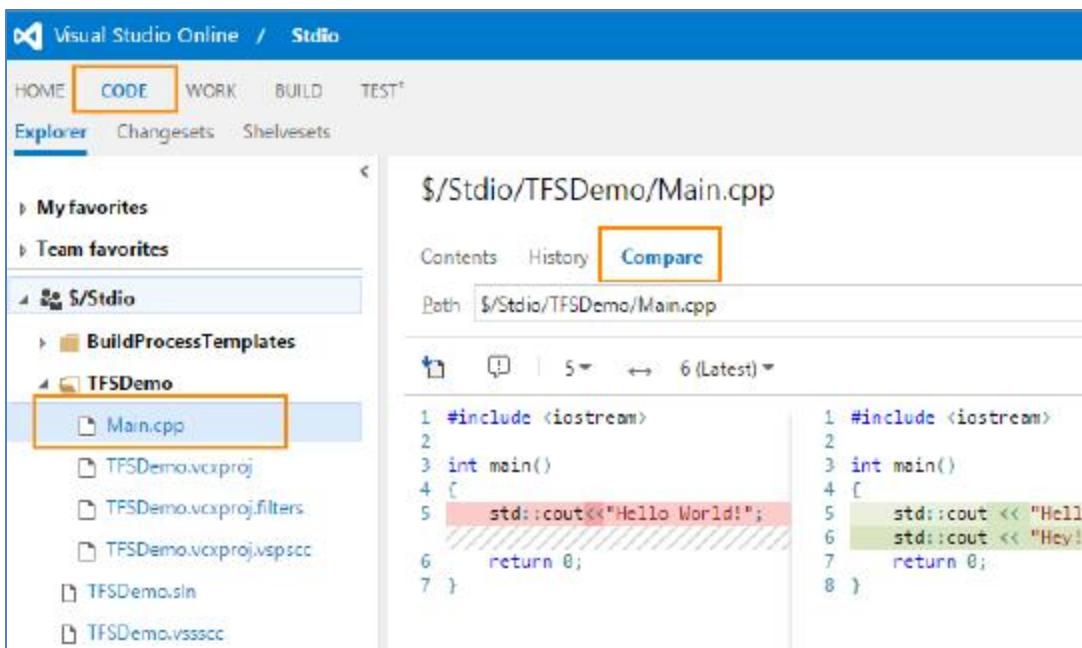


Bước 4: Tạo project

- Quay lại với cửa sổ Connect to Team Foundation Project (Team Explorer → Connect)



- Click vào DefaultCollection và chọn Team Project vừa tạo.
- Nếu không thấy DefaultCollection, thử Sign Out và đăng nhập lại bằng nút ở góc dưới bên trái cửa sổ.
- Sau khi chọn được team project, bấm Connect để Visual Studio
- Để thấy được sự thay đổi của mã nguồn, TFS cung cấp công cụ so sánh cho phép thấy sự thay đổi của mỗi bản update.
- Ví dụ vừa update project Studio lần 2 sau khi có một số thay đổi, so sánh chúng như sau:
 1. Quay lại trang quản lý Team Project, click vào menu CODE.
 2. Click vào file muốn kiểm tra thay đổi (Main.cpp) và chọn Compare.
 3. 2 phiên bản của file sẽ hiện lên với những đánh dấu rất chi tiết về những gì đã được thay đổi ở phiên bản 2.



Ngoài ra, TFS phát triển giúp dự án có thể kiểm soát nhóm, check in, check out file, quản lý các version, quản lý hoạt động nhóm, nhận biết các thay đổi trong source code.

TÓM LƯỢC

Bài học này cung cấp các kiến thức về:

- Quản lý cấu hình
- Lập kế hoạch quản lý cấu hình
- Quản lý phiên bản và phát hành
- Quản lý sự thay đổi
- Xây dựng hệ thống
- Các công cụ CASE cho quản lý cấu hình

PHỤ LỤC – ĐỒ ÁN MÔN HỌC

❖ YÊU CẦU THỰC HIỆN ĐỒ ÁN MÔN HỌC

- Mỗi nhóm 3 – 4 sinh viên chọn và thực thi một dự án, không được chọn trùng.
- Thực hiện theo yêu cầu mô tả trong mỗi nhóm bài tập.
- Các sinh viên dùng:
 - Phần mềm MS Excel để thực hiện việc viết test case
- Các quyển báo cáo khi nộp cần:
 - Ghi rõ tên, mã số Project
 - Ghi rõ tên các thành viên trong nhóm, công việc cụ thể của mỗi thành viên trong nhóm.

❖ NỘI DUNG YÊU CẦU ĐỒ ÁN

- Viết đặc tả yêu cầu phần mềm (SRS- Software Requirement Specification)
- Viết Test Plan (Kế hoạch kiểm thử)
- Viết Test Case
- Viết Test Report, Defect Tracking & Solutions

❖ THÔNG TIN ĐỒ ÁN

Phần mềm demo: QUẢN LÝ BÁN HÀNG (WEBSITE / APPLICATION)

- Áp dụng Black Box Testing kiểm thử trên form Login
- Áp dụng White Box Testing kiểm thử chức năng nhập xuất sản phẩm.
- Download phần mềm có sẵn hoặc phần mềm nhóm thực hiện, thực hiện yêu cầu kiểm thử.

❖ Danh sách seminar công cụ hỗ trợ kiểm thử

- Đề tài 1: Hệ thống quản lý bug Bugzilla
- Đề tài 2: Kiểm thử trên thiết bị di động (mobile testing)

- Đề tài 3: Công cụ kiểm thử tự động Selenium
- Đề tài 4: Công cụ kiểm thử tự động Katalon Studio
- Đề tài 5: Công cụ hỗ trợ kiểm thử tự động Robotium.
- Đề tài 6: Công cụ hỗ trợ kiểm thử tự động AutoIT
- Đề tài 7: Công cụ hỗ trợ kiểm thử Mantis Bug Tracker
- Đề tài 8: Công cụ hỗ trợ kiểm thử Sahi
- Đề tài 9: Công cụ hỗ trợ kiểm thử Soap UI
- Đề tài 10: Công cụ hỗ trợ kiểm thử Behavior Testing

Chọn một trong các công cụ: Katalon Studio, Selenium, IBM Rational Functional Tester, SilkTest, TestComplete, Testing Anywhere, WinRunner, LoadRunner, Visual Studio Test Professional, WATIR, Defect tracking tool, Test Effort tracking tool, Test schedule, Test automation tools: *Rational Robot (Functional & Performance test)*, *OpenSTA (Open source)*, *Witir (Open source)*

- ❖ **Công cụ kiểm thử đơn vị (Unit Testing):** JUnit, Findbugs, PMD, Checkstyle, EclEmma, Dbunit, StrungTestCase for JUnit, Emma, MockObjects, JunitEE, NUnit, NUnitAsp, NUnit addin for Visual Studio.Net, NUnitForms, csUnit, NCover, VSNUnit, dotUnit, .NETUnit, ASPUnit.
- ❖ **Công cụ kiểm thử chức năng (Functional Testing):** Software Testing Automation Framework (STAF), soapui, Linux Test Project, jWebUnit, Abbot Java GUI Test Framework, Software Automation Framework Support, Jameleon, WebInject, Marathon, Solex.
- ❖ **Công cụ kiểm hiệu năng (Performance Testing):** OpenSTA, Grinder, TPTEST, Database Opensource Test Suite, Sipp, WebLOAD, OpenWebLoad, Hammerhead 2 - Web Testing Tool, Dieseltest, DBMonster
- ❖ **Các biểu mẫu (Template) trong kiểm thử:**

A. Master Test Plan Outline

<Product/Component Name>

Project Test Plan

Last update: dd-mmm-yyyy
Submitted By: <Author Name>

Version: <x.x>

Revision History

DATE	REV	AUTHOR	DESCRIPTION

Test Plan Identifier

1. Test plan short name
2. Version/Revision Identifier
3. Author and contact information

References

<List of documents that support this test plan>

Introduction

1. Objectives
2. Resource
3. Scope
4. Testing Strategies
5. Other items

Product Risk Issues

Issues critical to product success including reasons for criticality

- Critical areas (possibly including)
- Essential functions
- Reliability
- Usability
- Safety
- Security
- Privacy
- Failure impact risks (covering)
- Enterprise
- Operations
- Fault likelihood risks
- General Risk assumptions

Features to be tested

<List of the features to be tested and levels of risk>

- Categories of test objectives
 - Functions
 - External interfaces
 - Constraints
 - States
 - Data conditions
 - Scenarios
 - Combinations of attributes
- Combinations of objectives
- Excluded features

Features not to be tested

<List of the features NOT to be tested and reasons>

Approach

1. Levels of testing including environments and responsible parties
2. Major evaluation activities, techniques, and tools
3. Reuse of test-ware descriptions and/or implementations
4. Data sources and extract & validation techniques
5. Result recording & checking methods
6. Test metrics
7. Tracing requirements
8. Relationship of testing to development activities
9. Communication and coordination policies and procedures
10. Missing test guidelines and Test Configuration Management procedures
11. Special requirement
12. Constrain

Termination Criteria and Resumption Requirements

1. Resumption requirements
2. Regression Testing Policy
3. Completion criteria
4. Suspension criteria
5. Pass/fail criteria

Test Deliverables

1. Test Descriptions.
2. Test-ware implementation
3. Test Reports

DELIVERABLE	ASSIGNED TO	COMPLETION DATE
Develop Test cases		
Develop Automated test suites		
Requirements Validation Matrix		

Environmental Needs

1. Hardware
2. Software
3. Security
4. Tools
5. Publications

Staffing and Training needs

1. Skill requirements
 - Training on the product
 - Training for any test tools to be used
2. Staffing pattern
3. Training needs and options

Responsibilities

RESPONSIBLE PERSON	RESPONSIBILITIES
Project Manager	
Developer Lead	
Test Lead	
Developers	
Testers	

Schedule

1. Availability and deadline constraints
2. Key activities and major milestones
3. Task estimates
4. References to project schedule

Planning Risks and Contingencies

1. Product risk / plan element correlation
2. High-risk planning assumptions
3. Strategy options

Approvals

Name (Print)

Signature

Date

1.

2.

3.

Glossary

Key terms, Definitions and Acronyms

Appendix

Test Result

B. Test Requirement List

A	B	C	D	E
1	Test Requirement List			
2				
3				
4	TR-ID	Test Requirements	TR Type	Traceability
5	Login			Notes
6	TR-001	User can create a new article.	Functional	
7	1			
8	2			
9	3			
10	Send mail			
11	4			
12	5			
13	6			
14	7			
15				
16				
17				
18				
19				
20				
21				
22				

C. Test Case List

	A	B	C	D	E	F	G	H	I
1	Test Case List								
2									
3									
4	TR ID	TC ID	TC Description	Pre-Condition	Steps	Expected Result	Test Result	Build	Notes
5	<Section> - For example. Article Manager								
6									
7									
8									
9									
10									

D. Daily Report Template

Daily Status Report

Project:

Reported by:

Report Date:

1. Issues/ Questions

-
-

2. Assignment

#	Task	Time Spent	% Completed
1	Creating Test Cases for Login function	1.5 hr	50%
2	Executing Test Cases for Send Mail function	3 hrs	20%
3	Exploratory Testing for Register function	3.5 hrs	60%

3. Results

3.1. Test Case Creation and Execution

#	Task	# of Created TCs	# of Executed TCs	% Completed	Note
1	Creating Test Cases for Login function	10 / 20	-	50%	
2	Executing Test Cases for Send Mail function	-	20/100	20%	
3					

3.2. Bug Reports

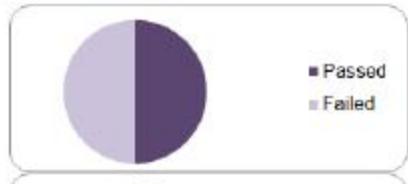
#	Task	# of New Bugs Found	# of Verified Bugs	% Completed	Note
1	New Bug Reports	5	-	-	
2	Bugs Verifying	-	2/4	50%	
3					

Best Regards,

<Reporter Name>

E. Daily Report - Day

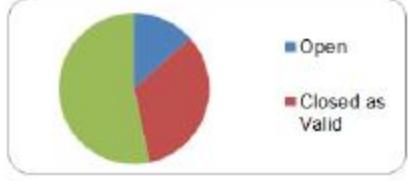
A	B	C	D	E	F	G	H	I																	
1	Daily Report - Day 1																								
2	Date	25-May-20																							
3	Build	0.9.8																							
4																									
5	Team 1		<table border="1"><thead><tr><th></th><th>Name</th></tr></thead><tbody><tr><td>- Test Lead</td><td></td></tr><tr><td>- Tester 1</td><td></td></tr><tr><td>- Tester 2</td><td></td></tr><tr><td>- Tester 3</td><td></td></tr><tr><td>- Tester 4</td><td></td></tr><tr><td>- Tester 5</td><td></td></tr></tbody></table>									Name	- Test Lead		- Tester 1		- Tester 2		- Tester 3		- Tester 4		- Tester 5		
	Name																								
- Test Lead																									
- Tester 1																									
- Tester 2																									
- Tester 3																									
- Tester 4																									
- Tester 5																									
6																									
7																									
8																									
9																									
10																									
11																									
12																									
13																									
14	Smoke Test		<table border="1"><thead><tr><th></th><th>#</th><th>%</th></tr></thead><tbody><tr><td>Passed</td><td>1</td><td>50%</td></tr><tr><td>Failed</td><td>1</td><td>50%</td></tr><tr><td>Total:</td><td>2</td><td>100%</td></tr></tbody></table>									#	%	Passed	1	50%	Failed	1	50%	Total:	2	100%			
	#	%																							
Passed	1	50%																							
Failed	1	50%																							
Total:	2	100%																							
15																									
16																									
17																									
18																									
19																									
20	TC List		<table border="1"><thead><tr><th></th><th>#</th><th>%</th></tr></thead><tbody><tr><td>Passed</td><td>1</td><td>50%</td></tr><tr><td>Failed</td><td>1</td><td>50%</td></tr><tr><td>Total:</td><td>2</td><td>100%</td></tr></tbody></table>									#	%	Passed	1	50%	Failed	1	50%	Total:	2	100%			
	#	%																							
Passed	1	50%																							
Failed	1	50%																							
Total:	2	100%																							
21																									
22																									
23																									
24																									
25																									
26	Bug Status		<table border="1"><thead><tr><th></th><th>#</th><th>%</th></tr></thead><tbody><tr><td>Open</td><td>2</td><td>13%</td></tr><tr><td>Closed as Valid</td><td>5</td><td>33%</td></tr><tr><td>Closed as Invalid</td><td>8</td><td>53%</td></tr><tr><td>Total:</td><td>15</td><td>100%</td></tr></tbody></table>									#	%	Open	2	13%	Closed as Valid	5	33%	Closed as Invalid	8	53%	Total:	15	100%
	#	%																							
Open	2	13%																							
Closed as Valid	5	33%																							
Closed as Invalid	8	53%																							
Total:	15	100%																							
27	<i>(Type in)</i>																								
28																									
29																									
30																									
31																									



■ Passed
■ Failed



■ Passed
■ Failed



■ Open
■ Closed as Valid

TÀI LIỆU THAM KHẢO

- [1]ThS Nguyễn Thị Thanh Trúc (2018). Kiểm thử phần mềm. HUTECH
- [2]Paul Hamill (2004) Unit Test Frameworks
- [3]Thạc Bình Cường, Nguyễn Đức Mận (2010) Kiểm thử và đảm bảo chất lượng phần mềm.
- [4]Paul Ammann, Jeff Offutt (2008): Introduction to Software Testing, Cambridge University Press.
- [5]Hung Q.Nguyen (2003): Testing Application on the Web: Test planning for mobile and Internet-based system, Wiley publishing
- [6]LogiGear (2009): Basic Software Testing Skills, LogiGear Corporation..
- [7]Glenford J. Myers (2004): The art of Software Testing, John Wiley & Son