

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

**KIỂM THỬ PHẦN MỀM**

**Biên Soạn:**

**ThS Nguyễn Thị Thanh Trúc**

## KIỂM THỬ PHẦN MỀM



\* 1 . 2 0 1 8 . C M P 1 0 3 \*

---

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

# MỤC LỤC

<b>MỤC LỤC .....</b>	<b>I</b>
<b>HƯỚNG DẪN.....</b>	<b>IV</b>
<b>BÀI 1: TỔNG QUAN KIỂM THỬ PHẦN MỀM.....</b>	<b>1</b>
<b>    1.1 CÁC MÔ HÌNH PHÁT TRIỂN PHẦN MỀM.....</b>	<b>1</b>
1.1.1 Water-fall Model.....	1
1.1.2 Prototype Model.....	2
1.1.3 Spiral Model .....	3
1.1.4 SDLC.....	4
1.1.5 Agile Model .....	5
1.1.6 Qui trình Hợp nhất (RUP).....	5
1.1.7 V model.....	7
<b>    1.2 KIỂM THỬ PHẦN MỀM .....</b>	<b>8</b>
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 .....	11
1.2.4 Mục tiêu của kiểm thử.....	12
1.2.5 Tầm quan trọng của kiểm thử .....	12
1.2.6 Các nguyên tắc trong kiểm thử.....	14
1.2.7 Các khái niệm liên quan kiểm thử.....	15
1.2.8 Các đối tượng thực hiện kiểm thử .....	18
1.2.9 Các điểm cần lưu ý khi kiểm thử.....	19
1.2.10 Các hạn chế của kiểm thử.....	20
<b>    1.3 VAI TRÒ CỦA KIỂM THỬ PHẦN MỀM .....</b>	<b>20</b>
<b>    TÓM LƯỢC.....</b>	<b>21</b>
<b>BÀI 2: YÊU CẦU KIỂM THỬ .....</b>	<b>22</b>
<b>    2.1 TÀI LIỆU SẴN PHÂM .....</b>	<b>22</b>
2.1.1 Tài liệu dự án .....	22
2.1.2 Tài liệu đặc tả .....	22
2.1.3 Tài liệu đặc tả thiết kế .....	23
2.1.4 Tài liệu kiểm thử .....	24
<b>    2.2 YÊU CẦU KIỂM THỬ.....</b>	<b>25</b>
2.2.1 Quy trình kiểm thử phần mềm .....	25
2.2.2 Thuộc tính yêu cầu phần mềm .....	30
2.2.3 Cấu trúc của bản kế hoạch kiểm thử .....	30
<b>    2.3 CÁC YÊU TỐ CÂN THIẾT CỦA KIỂM THỬ.....</b>	<b>32</b>
<b>    2.4 CÁCH VIẾT YÊU CẦU KIỂM THỬ .....</b>	<b>33</b>
<b>    TÓM LƯỢC.....</b>	<b>35</b>
<b>BÀI 3: KỸ THUẬT THIẾT KẾ TEST -CASE .....</b>	<b>36</b>
<b>    3.1 CÁC KHAI NIỆM CHINH.....</b>	<b>36</b>

<b>3.2 THIẾT KẾ TEST-CASE: WHITE-BOX.....</b>	<b>38</b>
3.2.1 Tổng quan về kiểm thử hộp trắng .....	38
3.2.2 Ưu điểm/nhược điểm kiểm thử hộp trắng.....	38
3.2.3 Các mức độ áp dụng.....	39
<b>3.3 THIẾT KẾ TEST-CASE: BLACK-BOX .....</b>	<b>39</b>
3.3.1 Tổng quan kiểm thử hộp đen .....	39
3.3.2 Ưu điểm/Nhược điểm của kiểm thử hộp đen.....	40
3.3.3 Qui trình kiểm thử hộp đen.....	41
<b>3.4 CÁC HỆ THÔNG QUAN LY TEST-CASE.....</b>	<b>43</b>
3.4.1 Giới thiệu .....	43
3.4.2 Testcase .....	44
3.4.3 Các thành phần testcase .....	45
<b>TÓM LƯỢC.....</b>	<b>47</b>
<b>BÀI 4: THIẾT KẾ TEST - CASE BLACK-BOX.....</b>	<b>48</b>
<b>4.1 LỚP TƯƠNG ĐƯƠNG VÀ PHÂN TÍCH BIÊN .....</b>	<b>48</b>
4.1.1 Lớp tương đương.....	48
4.1.2 Phân tích giá trị Biên .....	50
<b>4.2 PHÂN TÍCH RANG BUỘC.....</b>	<b>52</b>
<b>4.3 MÔI QUAN HỆ GIỮA HẠM VÀ DỮ LIỆU .....</b>	<b>53</b>
<b>4.4 CHUYÊN TRẠNG THAI .....</b>	<b>55</b>
<b>4.5 TỔ HỢP ĐIỀU KIỆN .....</b>	<b>56</b>
<b>TÓM LƯỢC.....</b>	<b>56</b>
<b>BÀI 5: THIẾT KẾ TEST-CASE: WHITE-BOX .....</b>	<b>57</b>
<b>5.1 KIÊM THƯ ĐƯƠNG CƠ BẢN .....</b>	<b>57</b>
<b>5.2 KIÊM THƯ LUÔNG ĐIỀU KHIÊN VÀ ĐỘ BAO PHỦ .....</b>	<b>59</b>
5.2.1 Một số khái niệm.....	59
5.2.2 Các cấp độ bao phủ (Coverage).....	59
<b>5.3 KIÊM THƯ VONG LẶP .....</b>	<b>60</b>
<b>5.4 KIÊM THƯ LUÔNG DỮ LIỆU .....</b>	<b>62</b>
<b>TÓM LƯỢC.....</b>	<b>66</b>
<b>BÀI 6: BÀI TẬP THIẾT KẾ TESTCASE .....</b>	<b>67</b>
<b>6.1 BÀI TẬP KIÊM THƯ HỘP ĐEN.....</b>	<b>67</b>
6.1.1 Bài tập 1.....	67
6.1.2 Bài tập 2.....	67
6.1.3 Bài tập 3.....	67
6.1.4 Bài tập 4.....	68
6.1.5 Bài tập 5.....	68
<b>6.2 BÀI TẬP KIÊM THƯ HỘP TRẮNG .....</b>	<b>69</b>
6.2.1 Bài tập 1.....	69
6.2.2 Bài tập 2.....	69
6.2.3 Bài tập 3.....	70
6.2.4 Bài tập 4.....	70

6.2.5 Bài tập 5.....	71
<b>TÓM LƯỢC.....</b>	<b>72</b>
<b>BÀI 7: LỖI PHẦN MỀM.....</b>	<b>73</b>
<b>7.1 TỔNG QUAN VỀ LỖI PHẦN MỀM.....</b>	<b>73</b>
7.1.1 Lỗi phần mềm và Bugs.....	73
<b>7.2 NGUYÊN NHÂN GÂY RA LỖI THƯƠNG GẶP.....</b>	<b>74</b>
<b>7.3 CÁC LỖI THƯƠNG GẶP TRONG PHẦN MỀM.....</b>	<b>77</b>
<b>7.4 TÌM LỖI VÀ PHÂN TÍCH LỖI .....</b>	<b>78</b>
7.4.1 Checklist kiểm tra mã nguồn.....	78
7.4.2 Qui tắc xác định lỗi phần mềm .....	85
<b>TÓM LƯỢC.....</b>	<b>86</b>
<b>BÀI 8: CÁC HỆ THỐNG QUẢN LÝ BUG.....</b>	<b>87</b>
<b>8.1 GIỚI THIỆU HỆ THỐNG QUẢN LÝ BUG .....</b>	<b>87</b>
8.1.1 Giới thiệu .....	87
<b>8.2 VÒNG ĐƠI CỦA BUG TRÊN HỆ THỐNG QUẢN LÝ BUG.....</b>	<b>88</b>
8.2.1 Những thông số cần thiết để đo lỗi.....	88
8.2.2 Kiểm soát thực hiện sửa lỗi (Defect tracking) .....	88
8.2.3 Các thông số lỗi .....	89
8.2.4 Môi trường kiểm thử .....	89
<b>8.3 THỰC HANH VỚI HỆ THỐNG QUẢN LY BUGZILLA.....</b>	<b>90</b>
<b>TÓM LƯỢC.....</b>	<b>93</b>
<b>BÀI 9: KIỂM THỬ TỰ ĐỘNG .....</b>	<b>94</b>
<b>9.1 GIỚI THIỆU VỀ AUTOMATION SOFTWARE TESTING .....</b>	<b>94</b>
9.1.1 Quy trình kiểm thử tự động.....	94
9.1.2 Ưu và nhược điểm kiểm thử tự động .....	95
9.1.3 Phân loại công cụ kiểm thử.....	95
<b>9.2 GIỚI THIỆU ACTION-BASED TESTING .....</b>	<b>98</b>
<b>TÓM LƯỢC.....</b>	<b>102</b>
<b>BÀI 10: CÁC CÔNG CỤ HỖ TRỢ KIỂM THỬ TỰ ĐỘNG .....</b>	<b>103</b>
<b>10.1 QUICKTEST PRO .....</b>	<b>103</b>
10.1.1 Giới thiệu .....	103
10.1.2 Cài đặt & thành phần QTP.....	105
<b>10.2 SELENIUM .....</b>	<b>106</b>
10.2.1 Giới thiệu .....	106
10.2.2 Thành phần của Selenium.....	109
<b>TÓM LƯỢC.....</b>	<b>110</b>
<b>PHỤ LỤC – ĐỒ ÁN MÔN HỌC.....</b>	<b>111</b>
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>116</b>

# HƯỚNG DẪN

## MÔ TẢ MÔN HỌC

Bài giảng Kiểm thử phần mềm trang bị cho sinh viên kiến thức về kiểm thử phần mềm và quy trình kiểm thử 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 và các phần mềm hỗ trợ kiểm thử tự động. Qua môn học này cung cấp kiến thức và kỹ năng cho sinh viên để áp dụng trong quy trình kiểm thử phần mềm. Với kiến thức của môn học này sinh viên có thể áp dụng để thực thi việc kiểm thử một phần mềm cụ thể. Sinh viên có khả năng sử dụng thành thạo các công cụ phục vụ cho việc kiểm thử phần mềm.

## NỘI DUNG MÔN HỌC

- BÀI 1: Tổng quan về kiểm thử phần mềm. Giới thiệu tổng quan, các khái niệm cơ bản về kiểm thử phần mềm, các mô hình phát triển phần mềm vai trò kiểm thử phần mềm
- BÀI 2: Yêu cầu kiểm thử. Giới thiệu về tài liệu của sản phẩm, các yêu cầu kiểm thử, các yếu tố cần thiết của kiểm thử phần mềm.
- BÀI 3: 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 4: 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 5: Thiết kế test-case White-box. Giới thiệu bài tập thực hành black-box, bài tập thực hành white-box.
- BÀI 6: Bài tập thiết kế các test-case Giới thiệu tổng quan, quy trình quản lý rủi ro.

- BÀI 7: Lỗi phần mềm. 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.
- BÀI 8: Hệ thống quản lý bug. 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 9: Kiểm thử tự động. Giới thiệu về Automation testing, giới thiệu về action based testing.
- BÀI 10: Các công cụ hỗ trợ kiểm thử tự động. Giới thiệu về Quick Test Pro, Selenium.

## KIẾN THỨC TIỀN ĐỀ

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

## YÊU CẦU MÔN HỌC

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 MÔN HỌC

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 MÔN HỌC

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 KIỂM THỬ PHẦN MỀM

Nội dung gồm các phần sau:

- *Giới thiệu tổng quan, các khái niệm cơ bản về kiểm thử phần mềm*
- *Các mô hình phát triển phần mềm*
- *Vai trò kiểm thử phần mềm*

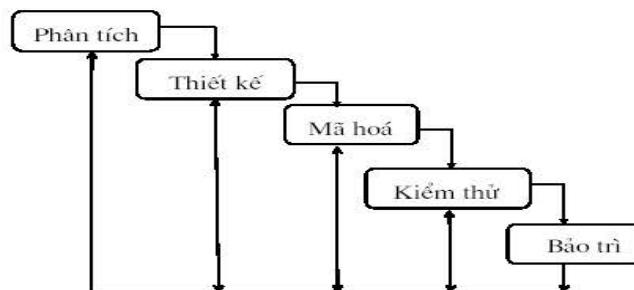
## 1.1 CÁC MÔ HÌNH PHÁT TRIỂN PHẦN MỀM

### 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: được tiến hành ngay sau khi kết thúc việc xác định yêu cầu, mô tả lại thế giới thực thông qua các mô hình (mô hình thế giới thực) trước khi thiết kế.
  - Thiết kế: Được 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.

### 1.1.3 Spiral Model

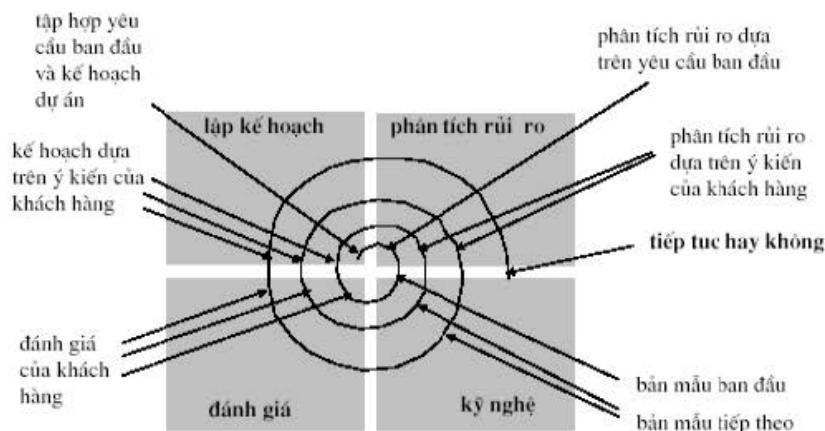
Mô hình là sự kết hợp của mô hình bản mẫu thiết kế và mô hình thác nước được lặp lại nhiều lần. Ở lần lặp kế tiếp hệ thống được tìm hiểu và xây dựng hoàn thiện hơn ở lần lặp trước đó.

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.



**Mô hình làm bản mẫu**

**Hình 1.2: Mô hình bản mẫu**



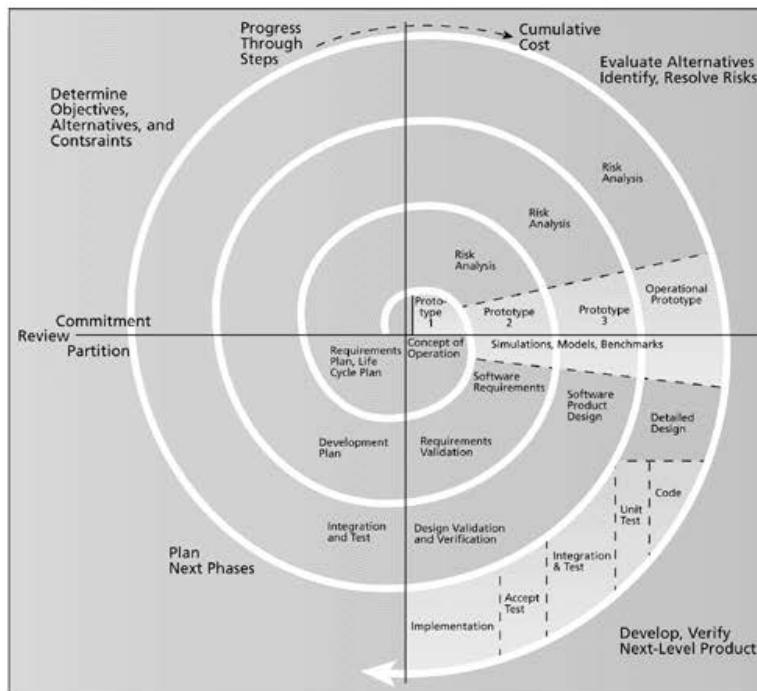
**Mô hình xoắn ốc**

**Hình 1.3: Mô hình xoắn ốc**

thêm công đoạn phân tích mức độ rủi ro để quyết định nên đi tiếp theo hướng này nữa hay không.

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.



**Hình 1.4: Mô hình xoắn ốc**

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, ...

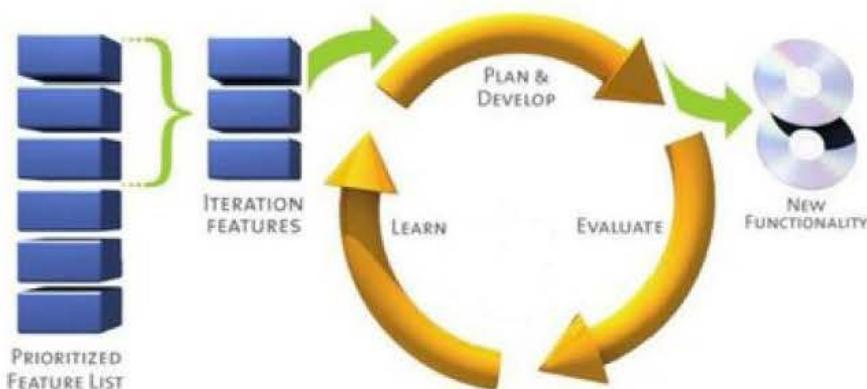
#### 1.1.4 SDLC

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

dụng những khái 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 cộng 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".



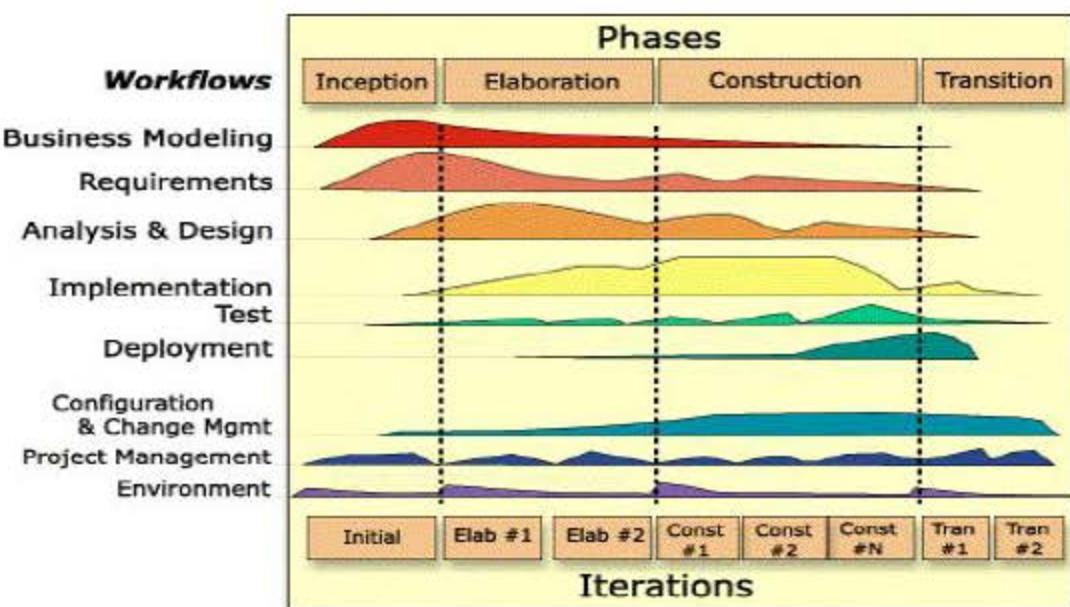
**Hình 1.5: Mô hình Agile**

### 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).

**Hình 1.6: Mô hình RUP**

- **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,...

- **Tinh chế (elaboration):**

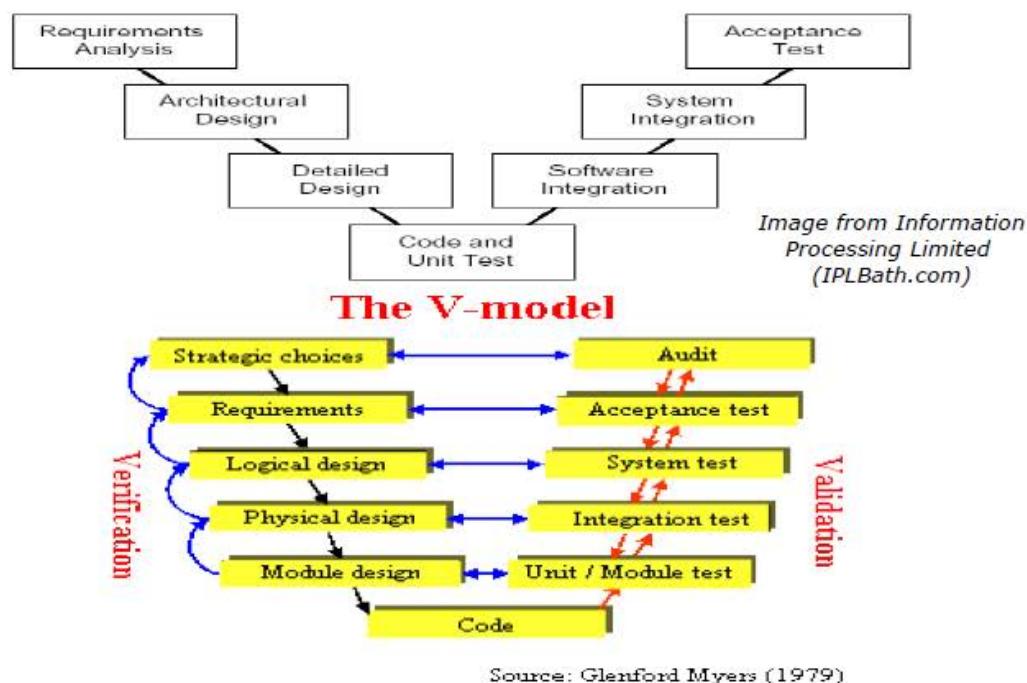
- Tinh chế kiến trúc hệ thống, yêu cầu hệ thống và đảm bảo kế hoạch sự ổn định của kế hoạch
- Đánh giá độ rủi ro, các thành phần sử dụng
- Xây dựng nền kiến trúc nền tảng hệ thống,...

- **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 thỏa thuận,...
- Chuyển giao (transition):**
  - Thực hiện cài đặt hệ thống
  - Thử 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 1.7: 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 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.

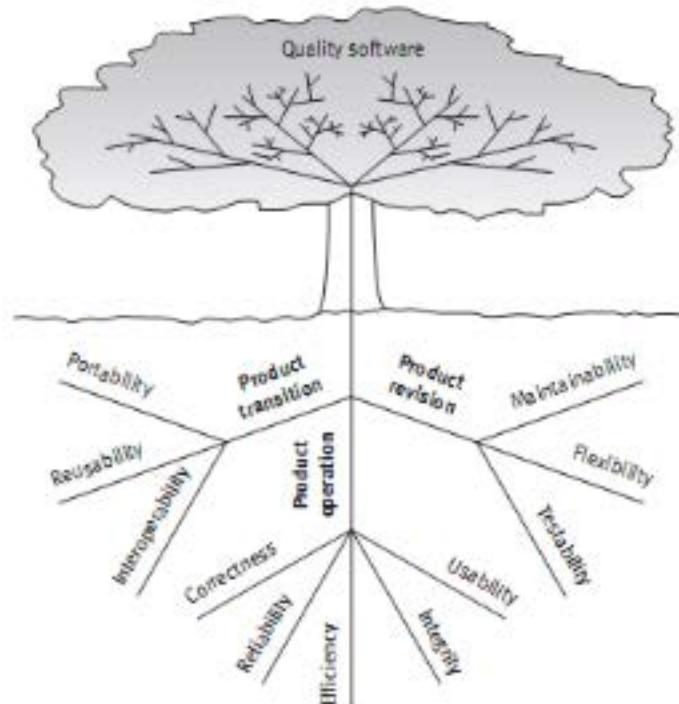
Theo Daniel Galin, đảm bảo chất lượng phần mềm là một tập các hoạt động đã được lập kế hoạch và có hệ thống, cần thiết để cung cấp đầy đủ sự tin cậy vào quy trình phát triển phần mềm hay quy trình bảo trì phần mềm phù hợp với các yêu cầu

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, Marciniak 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 1.8: Cây yếu 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 về 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 modul 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 modul phần mềm trong một dự án mới đang được phát triển mà các modul này ban đầu được thiết kế cho một dự án khác. Các yêu cầu này cũng cho phép các dự án tương lai có thể sử dụng một modul đã có hoặc một nhóm các modul hiện đang được phát triển. Tái sử dụng phần mềm sẽ tiết kiệm tài nguyên phát triển, rút ngắn thời gian phát triển và tạo ra các moduls chất lượng cao hơn. 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

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**

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

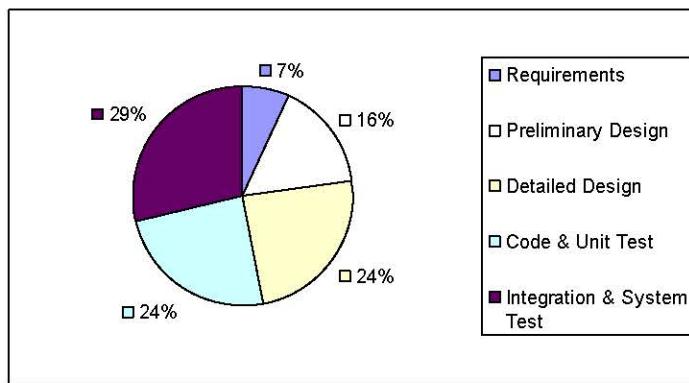
Mức 3: Mục tiêu của kiểm thử là để giảm các rủi ro khi sử dụng phần mềm

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**

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.

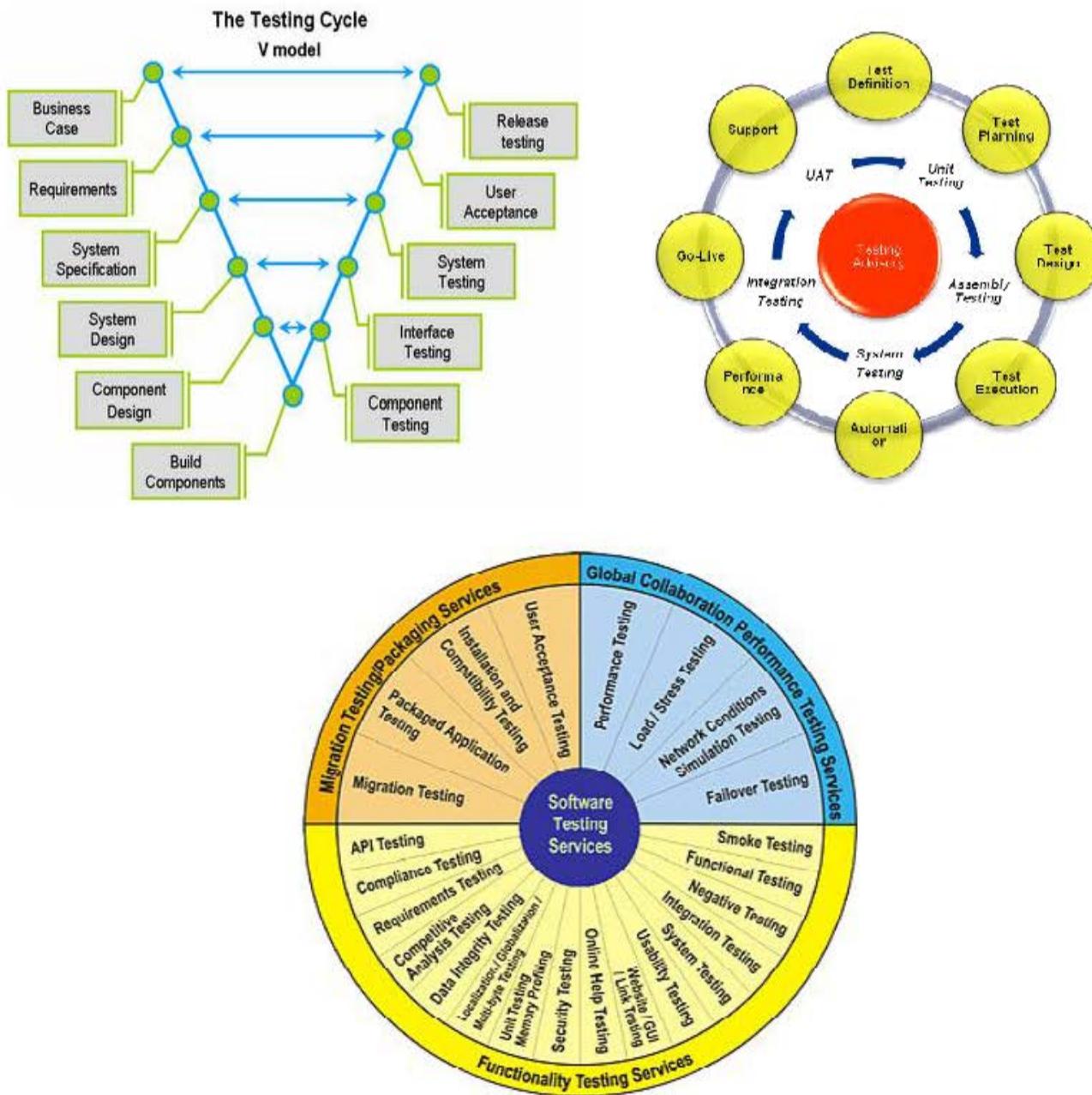


**Hình 1.9: Chi phí phân bổ cho các tác vụ trong phát triển phần mềm**

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 1.10: Các tác vụ trong quản lý chất lượng dự án

### 1.2.6 Các nguyên tắc trong kiểm thử

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 bâng 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**

#### **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.

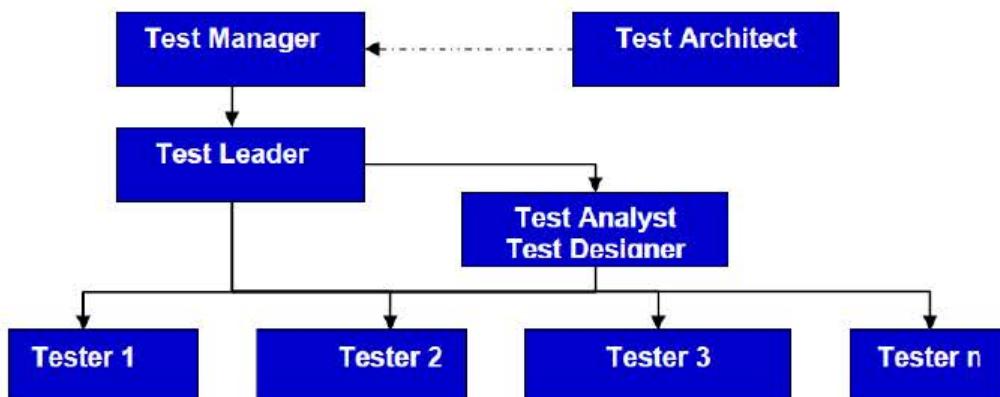
#### - Kiểm thử Thread testing

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**

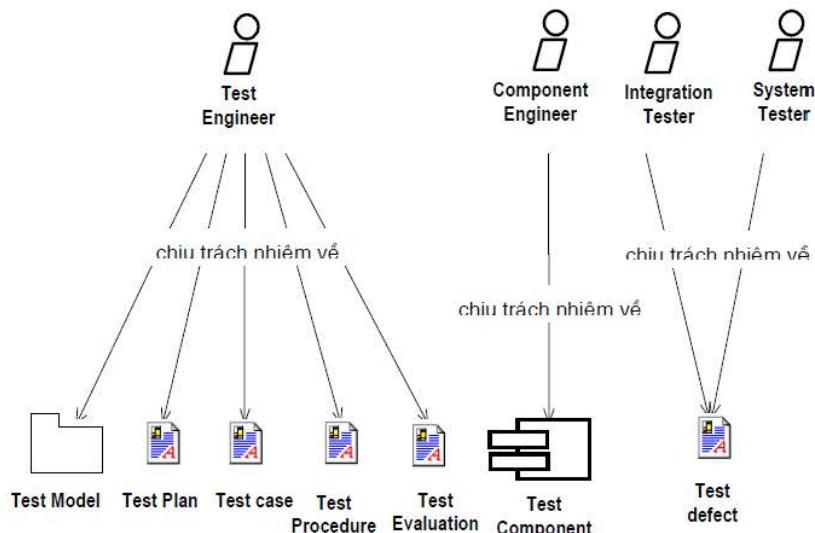
- **Vị trí nhóm trưởng:** tìm và tuyển người để đảm bảo đủ nguồn nhân lực cho đội ngũ QA, tạo các kế hoạch kiểm thử, lựa chọn công cụ nếu cần để thực hiện việc kiểm thử, và trách nhiệm cuối cùng là quản lý hoạt động của cả đội QA.
  - **Vị trí kỹ sư kiểm thử:** thực hiện việc kiểm thử các chức năng, 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**

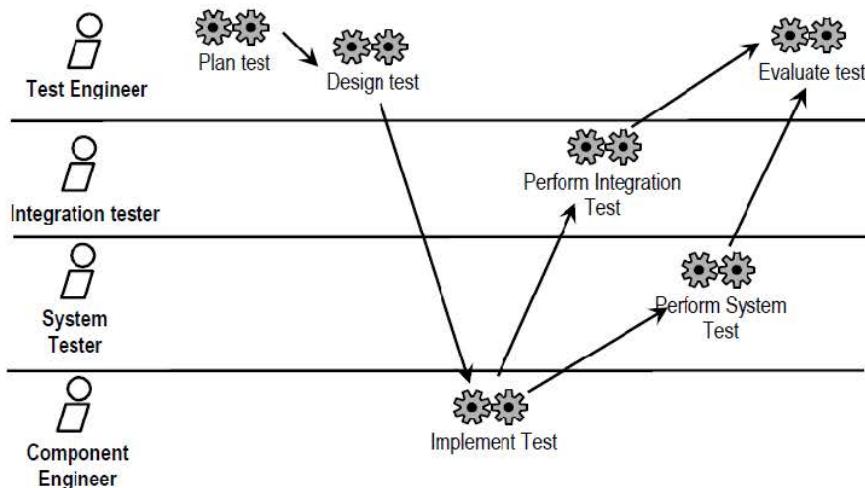


Hình 1.11: Sơ đồ tổ chức kiểm thử

## Worker và các quy trình



Hình 1.12: Worker và các qui trình



Hình 1.13: Worker và công việc kiểm thử

### 1.2.9 Các điểm cần lưu ý khi kiểm thử

- Chất lượng phần mềm không phải do khâu kiểm thử mà do thiết kế quyết định
- Tính dễ kiểm thử phụ thuộc vào cấu trúc chương trình
- Người kiểm thử nên làm việc độc lập với người phát triển phần mềm
- 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.
- 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

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 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 văn đề đúng và hữu dụng.
- Kiểm thử phần mềm là một thành phần trong lĩnh vực rộng hơn, đó là Verification & Validation (V &V).

# TÓM LƯỢC

*Bài học này giới thiệu những kiến thức sau đây:*

- *Mô hình phát triển phần mềm*
- *Yêu cầu kiểm thử phần mềm*
- *Vai trò của kiểm thử phần mềm*

# BÀI 2: YÊU CẦU KIỂM THỬ

Nội dung gồm các phần sau:

- *Tài liệu sản phẩm*
- *Qui trình kiểm thử*
- *Các yếu tố cần thiết kiểm thử phần mềm*

## 2.1 TÀI LIỆU SẢN PHẨM

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.

### 2.1.1 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.

### 2.1.2 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.

### 2.1.3 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.

## 2.1.4 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 lân vét 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.

## **2.2 YÊU CẦU KIỂM THỬ**

---

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 văn đề mong đợi

### **2.2.1 Quy trình kiểm thử 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 2.1: Vòng đời kiểm thử phần mềm**

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 đều 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.

### **REQUIREMENT 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, 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).

## 2.2.2 Thuộc tính yêu cầu phần mềm

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

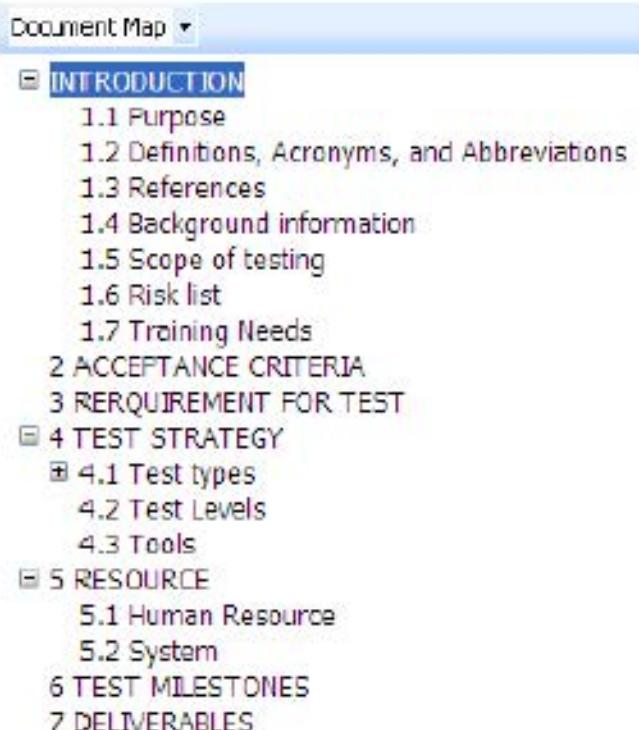
- **Đầy đủ.** Liệu đặc tả còn thiếu cái gì không. Đầy đủ chi tiết chưa. Liệu nó đã bao gồm mọi điều cần thiết để không phụ thuộc vào tài liệu khác.
- **Trúng đích.** Liệu nó đã cung cấp lời giải đúng đắn cho bài toán, liệu nó đã xác định đầy đủ các mục tiêu và không có lỗi.
- **Chính xác, không nhập 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.

## 2.2.3 Cấu trúc của bản kế hoạch kiểm thử

Bản kế hoạch kiểm thử cơ bản bao gồm 7 thành phần:

1. Introduction
2. Acceptance criteria (Điều kiện chấp nhận sản phẩm)

3. Requirements for test
4. Test strategy (Chiến lược kiểm thử)
5. Resources for testing
  - Human and responsibilities
  - System: hardware & software
6. Test milestones (Cột mốc kiểm thử)
7. Deliverables of test: Test Plan, Test Case, Test Reports



## 1. Introduction

Mục đích: Trình bày ngắn gọn về mục đích và tổ chức của tài liệu

- Các định nghĩa, từ viết tắt, thuật ngữ: cung cấp các định nghĩa của các thuật ngữ, từ viết tắt cần thiết để giải thích đúng các kế hoạch kiểm thử
- Tài liệu tham khảo: Liệt kê tất cả những tài liệu dùng để tạo ra bản kế hoạch
- Thông tin cơ bản: Mô tả ngắn gọn về dự án
- Phạm vi kiểm thử:
  - Danh mục các giai đoạn kiểm thử

- Danh sách các loại hình kiểm thử
  - Danh sách các giả định
  - Các khái niệm khuyết theo dự kiến
- Danh mục các rủi ro: các rủi ro có thể ảnh hưởng việc thiết kế, thực thi các kiểm thử
- Nhu cầu đào tạo: Liệt kê các nhu cầu đào tạo của các thành viên trong nhóm để thực thi việc kiểm thử

## 2. Tiêu chí chấp nhận sản phẩm

Danh sách các tiêu chí nhằm xác định mức độ chất lượng kiểm thử đủ để bàn giao cho khách hàng hoặc đủ để sang pha tiếp theo

Các tiêu chí có thể là:

- (1) Tỉ lệ bao phủ của kiểm thử
- (2) Tỉ lệ bao phủ thành công
- (3) Số lượng ca kiểm thử
- (4) Tỉ lệ lỗi tìm được
- (5) Tỉ lệ bỏ qua lỗi

## **2.3 CÁC YẾU TỐ CẦN THIẾT CỦA KIỂM THỬ**

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 vấn 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

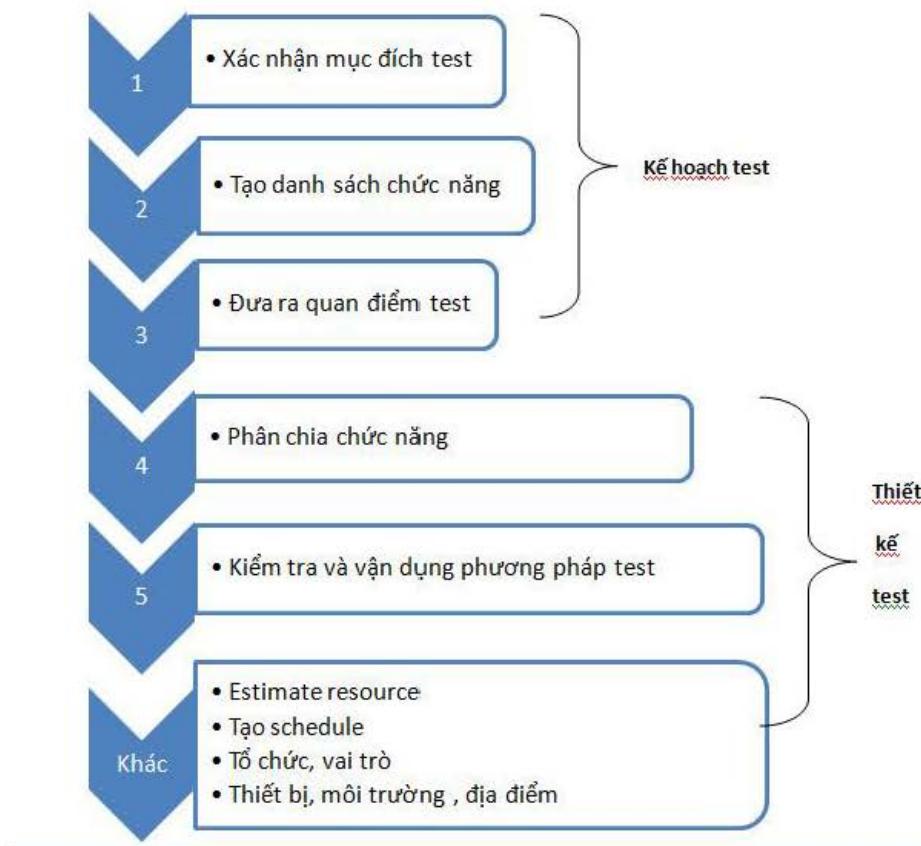
Những kỹ sư kiểm thử giỏi luôn để mắt quan sát những điều bất thường trong quá trình kiểm thử của mình. Những điều bất thường này có khi chỉ là những lỗi lập trình đơn giản, có khi là cả một "ổ" bug, có khi bug mà trước giờ nhiều người vẫn chưa mô phỏng lại được. Hãy ghi nhận lại tất cả những bất thường 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.

## **2.4 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.
- Cần 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.

## TÓM LƯỢC

Bài học này giới thiệu những kiến thức sau đây:

- Yêu cầu phần mềm
- Qui trình kiểm thử
- Thuộc tính yêu cầu phần mềm
- Cấu trúc bảng kế hoạch kiểm thử

# BÀI 3: 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*

## 3.1 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 đầu vào ngẫu nhiên, quá trình kiểm thử một chương trình bằng việc chọn ngẫu nhiên một tập con các giá trị đầu vào có thể. Về mặt khả năng tìm ra nhiều lỗi nhất, tập hợp các ca kiểm thử được chọn ngẫu nhiên có rất ít cơ hội là tập hợp tối ưu 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.

## **3.2 THIẾT KẾ TEST-CASE: WHITE-BOX**

---

### **3.2.1 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.

### **3.2.2 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.

### 3.2.3 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.

## 3.3 THIẾT KẾ TEST-CASE: BLACK-BOX

### 3.3.1 Tổng quan kiểm thử hộp đen

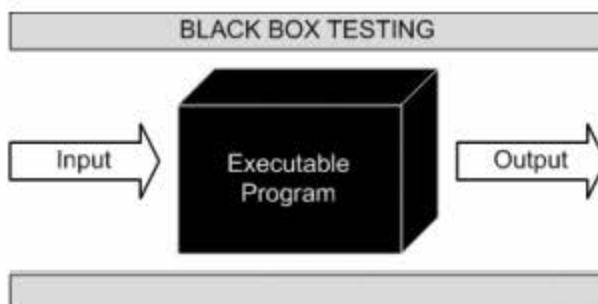
Phương pháp kiểm thử phần mềm được thực hiện khi không biết được cấu tạo bên trong của phần mềm, tester kiểm tra xem hệ thống như chiếc hộp đen.

- 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.



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.

### **3.3.2 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**

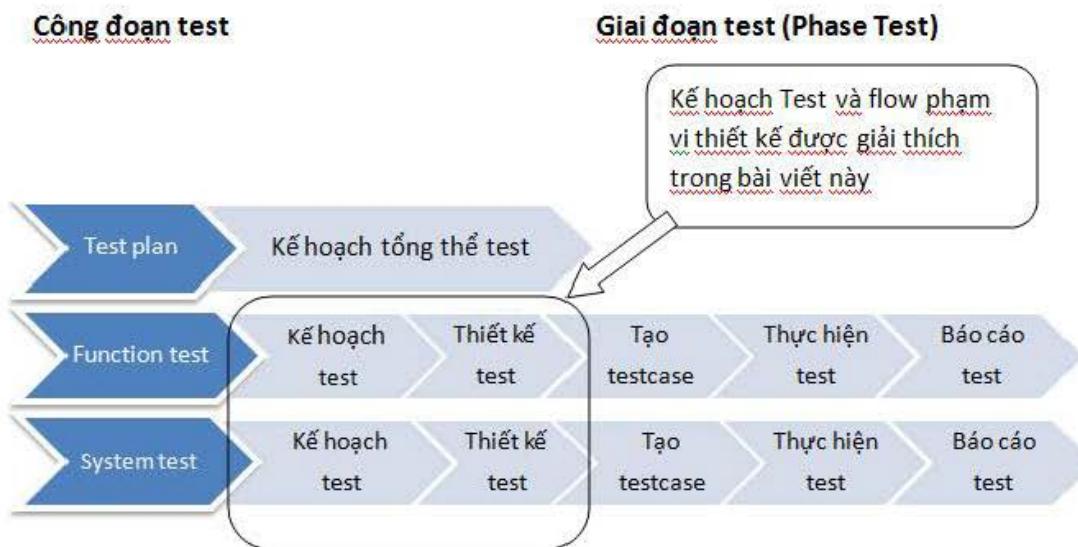
- Dữ liệu đầu vào yêu cầu một khối lượng mẫu (sample) khá lớn
- Nhiều dự án không có thông số rõ ràng thì việc thiết kế test case rất khó và do đó khó viết kịch bản kiểm thử do cần xác định tất cả các yếu tố đầu vào, và thiếu cả thời gian cho việc tập hợp này.
- Chỉ có một số nhỏ các đầu vào có thể được 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.

### **3.3.3 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 3.1: 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 testcase:** 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.

## **3.4 CÁC HỆ THỐNG QUẢN LÝ TEST-CASE**

### **3.4.1 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 mối", từ source code, test

case, document, process, automation test, CI/CD,.. dùng khá tiện dụng, quản lý test case cũng rất tốt.

### 3.4.2 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 đầu, 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 lời 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.

### 3.4.3 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" Thế 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 LƯỢC

Bài học này giới thiệu những kiến thức về.

- *Giới thiệu thiết kế testcase*
- *Cấu trúc thành phần testcase*

# BÀI 4: THIẾT KẾ TEST - CASE BLACK-BOX

Nội dung gồm các phần sau:

- *Kỹ thuật phân lớp tương đương và giá trị biên*
- *Phân tích ràng buộc*
- *Các mối quan hệ hàm và dữ liệu*
- *Chuyển trạng thái*
- *Tổ hợp điều kiện*

## **4.1 LỚP TƯƠNG ĐƯƠNG VÀ PHÂN TÍCH BIÊN**

### **4.1.1 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 cần 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 kế tiếp là cần 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 và 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	Ko Thuê	Thuê bán thời gian	Thuê toàn thời gian	Ko Thuê	Nhập dữ liệu sai
Lớp tương đương	Ko Hợp lệ	Hợp lệ	Hợp lệ	Hợp lệ	Hợp lệ	Ko 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}

## 4.1.2 Phân tích 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.

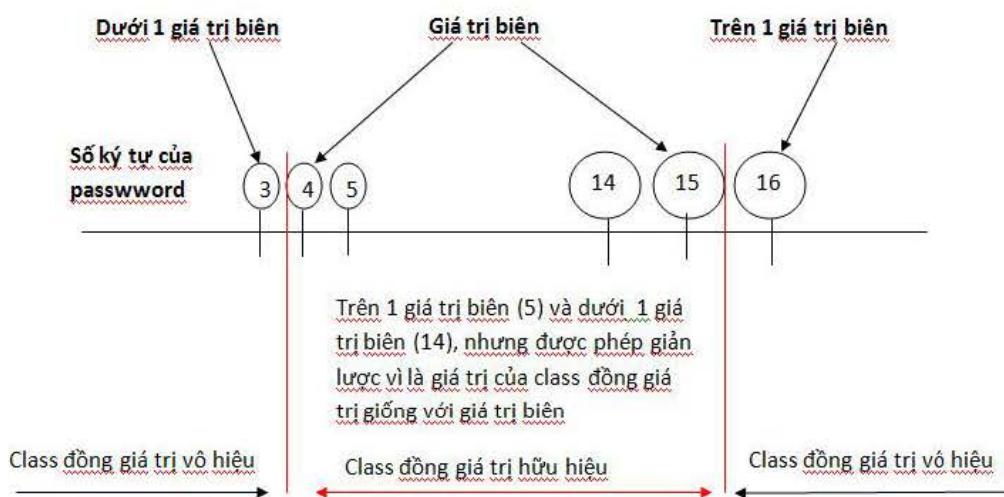
Kỹ sư nhiều kinh nghiệm chắc chắn từng gặp phải các lỗi của hệ thống ngay tại giá trị biên. Đó là lý do tại sao phân tích giá trị biên lại quan trọng khi kiểm thử hệ thống.

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

### Ví dụ

Spec 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 là lệch 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\$.

### **Standard BVA**

- Giả sử biến x có miền giá trị [min,max]
  - Các giá trị được chọn để kiểm tra
    - Min              - Minimal
    - Min+            - Just above Minimal
    - Nom             - Average
    - Max-           - Just below Maximum

- Max
- Maximum

Ví dụ:

Boundary Value Analysis Test Cases				
Case	a	b	c	Expected Output
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a Triangle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	199	100	Isosceles
9	100	200	100	Not a Triangle
10	1	100	100	Isosceles
11	2	100	100	Isosceles
12	199	100	100	Isosceles
13	200	100	100	Not a Triangle

- Ràng buộc:  $1 \leq a, b, c \leq 200$ .
- Áp dụng Standard BVA (số test case  $4*3 + 1 = 13$ )
  - min = 1
  - min+ = 2
  - nom = 100
  - max- = 199
  - max = 200

Ví dụ:

- Bài toán tìm ngày kế tiếp với các ràng buộc:
  - $1 \leq Day \leq 31$ .
  - $1 \leq month \leq 12$ .
  - $1812 \leq Year \leq 2012$
- Áp dụng Standard BVA (số test case  $4*3 + 1 = 13$ )

## 4.2 PHÂN TÍCH RÀNG BUỘC

Phân lớp tương đương và phân tích biên xem mỗi biến (trường) độc lập. Điều này quan trọng nhưng chúng ta cũng phải xem xét mối liên hệ giữa các biến khác. Đồ thị

Cause-Effect là một tiếp cận cho phân tích này, nhưng có thể rất phức tạp. Đây cũng là phương pháp đơn giản hơn mà còn có giá trị.

Các bước chung:

- Nhận diện những biến phụ thuộc và nghiên cứu những ràng buộc tương ứng.
- Nhận diện input và output có thể cho những biến
- Liệt kê những biến theo dạng bảng với input/output và ràng buộc tương ứng

Ví dụ: Mỗi quan hệ dữ liệu của biến Start Date và End Date

	INPUT	OUTPUT		RESTRICTIONS	
Data Field	Entry Source	Display	Print	Constrained by	Constraint
Start Date					End Date
End Date				Start Date	

## 4.3 MỐI QUAN HỆ GIỮA HÀM VÀ DỮ LIỆU

Khái niệm hàm đóng vai trò trung tâm trong phát triển và kiểm thử phần mềm. Nói một cách không hình thức, hàm liên kết các phần tử của các tập hợp. Trong chương trình NextDate, hàm được tính bởi chương trình liên kết một ngày được cho với ngày kế tiếp. Trong bài toán tam giác, hàm của ba số nguyên đầu vào là loại của tam giác tạo bởi các cạnh có độ dài ứng với dữ liệu đầu vào.

Bất kỳ một chương trình có thể được coi là một hàm liên kết dữ liệu đầu ra với dữ liệu đầu vào. Trong phác biểu toán học của hàm, các dữ liệu đầu vào tạo thành miền xác định, và các dữ liệu đầu ra tạo thành miền giá trị.

Để hiểu rõ hơn các công việc trong thiết kế kiểm thử hàm, chúng ta giả sử đang thiết kế kiểm thử cho chương trình  $P$  và giả sử  $Y = P(X)$  trong đó  $X$  là vec-tơ của  $n$  biến đầu vào và  $Y$  là vec-tơ của  $m$  biến đầu ra. Dựa trên hiểu biết của chúng ta về đặc tả của chương trình  $P$ , với một vec-tơ đầu vào  $X = (a_1, \dots, a_n)$  với  $a_i$  là các giá trị cụ thể thì chúng ta tính được một vec-tơ kết quả cụ thể  $Y = (b_1, \dots, b_m)$ . Kết quả  $Y$  này là được gọi là *kết quả mong đợi*. Kết quả thực của chương trình  $P$  với đầu vào  $X$  khi chạy chương trình là  $P(X)$ . Chúng ta cần kiểm tra kết quả mong đợi đúng bằng kết quả thực của chương trình, tức là  $Y$  và  $P(x)$  là một hay không. Nếu đúng thì chương trình

đã chạy đúng với  $X$ . Nếu chương trình chạy đúng với mọi giá trị  $X$  như mô tả trong đặc tả thì chúng ta kết luận chương trình  $P$  đã được cài đặt đúng so với đặc tả.

Trước hết chúng ta hãy xem các công việc chính của thiết kế kiểm thử cho chương trình  $Y = P(X)$ :

- Xác định các biến đầu vào, đầu ra và miền giá trị của từng biến. Tức là chúng ta phải xác định  $X$  và  $Y$  gồm có những biến nào, và miền giá trị của từng biến. Giả sử  $X$  gồm các biến  $x_1, \dots, x_n$  và  $Y$  gồm các biến  $y_1, \dots, y_m$ , chúng ta xác định từng  $\text{Dom}(x_i)$  và  $\text{Dom}(y_j)$  với  $i = 1..n; j = 1..m$  trong đó  $\text{Dom}(x)$  là tập các giá trị  $x$  có thể nhận.
- Chọn một số bộ giá trị của  $X$  làm ca kiểm thử, và tính (tay) kết quả mong đợi tương ứng cho từng bộ giá trị đầu vào. Với một bộ giá trị đầu vào cụ thể, ví dụ  $X = (a_1, \dots, a_n)$ , chúng ta phải tính được  $Y = (b_1, \dots, b_m)$  để có thể so sánh với kết quả của chương trình là  $P(a_1, \dots, a_n)$  từ đó biết chương trình đã đúng như mong đợi chưa. Việc tính thủ công có thể không tăm thường với những chương trình phức tạp. Việc chọn bộ giá trị đầu vào thường dựa trên hiểu biết về miền dữ liệu, và thường lấy các giá trị đặc biệt và giá trị bình thường cho mỗi miền rồi kết hợp với nhau.

Tạo ca kiểm thử dựa trên miền đầu vào của các biến có hai đặc trưng chính. Một là số lượng ca kiểm thử có thể nhiều khi chúng ta phải tạo các tổ hợp của các giá trị đặc biệt hoặc đại diện của các biến. Hai là việc tính kết quả mong đợi thường đơn giản vì đã hiểu chức năng của chương trình hoặc có thể tính “xuôi” kết quả theo đặc tả.

- Ngược lại, chúng ta có thể lấy một số bộ giá trị đầu ra mong đợi  $Y$  trước, rồi tính ngược một bộ giá trị đầu vào  $X$  để xây dựng các ca kiểm thử. Khi lấy bộ giá trị đầu ra mong đợi thường chọn giá trị đặc biệt của miền đầu ra, hoặc chọn các bộ đầu ra dựa trên hiểu biết về chương trình. Như bài toán giải phương trình bậc hai chúng ta có thể lấy  $Y$  có nghiệm kép, có hai nghiệm tách biệt, v.v.

Việc sinh ca kiểm thử dựa trên miền đầu ra có đặc trưng ngược lại so với trên. Thông thường các hàm có số đầu ra ít hơn và chúng ta cũng không thường phải lấy tổ hợp của các giá trị đặc biệt của đầu ra. Việc tính ngược lại giá trị đầu vào sẽ khó hơn vì chúng ta phải lặp ngược lại các bước trong đặc tả.

Do không thể kiểm thử vét cạn được nên chúng ta cần xác định một số ca kiểm thử đại diện cho các lớp hành vi của chương trình. Bài toán thiết kế kiểm thử trở thành bài toán xác định các lớp đại diện chương trình, các lớp này có thể là các lớp của đầu vào, các lớp của đầu ra, hay các lớp hành vi của chương trình dựa trên đặc tả của nó.

## 4.4 CHUYỂN TRẠNG THÁI

Liên quan phân tích mỗi quan hệ trạng thái và sự kiện hay hành động gây ra chuyển từ trạng thái này sang trạng thái khác

### Các bước chung

- Nhận diện tất cả trạng thái được hỗ trợ
- Với mỗi khi kiểm thử định nghĩa:
  - Trạng thái bắt đầu
  - Sự kiện đầu vào gây ra những trạng thái
  - Kết quả đầu ra cho mỗi trạng thái
  - Trạng thái cuối
- Các ca kiểm thử nên bao phủ cả kiểm thử có thể và không thể

Code	VIEW MODE														NAVIGATION COMMAND		
0	Edit View-Record [1st]	Edit View displaying the 1st record													F	First	
1	Edit View-Record [1st+1]	Edit View displaying the 2nd record													P	Previous	
2	Edit View-Record [x]	Edit View displaying record [x]													N	Next	
3	Edit View-Record [x-1]	Edit View displaying record [x-1]													L	Last	
4	Edit View Record [Last]	Edit View displaying the last record													FV	Full View	
5	Edit View-Record [Last-1]	Edit View displaying the next to last record															
6	Full View-Record [1st]	Full View displaying the 1st record													EV	Edit View	
7	Full View-Record [x]	Full View displaying record [x]													LK	Record ID Link	
<b>Test Case No.</b>		<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	
<b>Start View Mode</b>		0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	
<b>Navigation Command (Input)</b>		N	L	FV	F	P	L	FV	F	P	L	FV	F	N	L	FV	
<b>End View Mode</b>		1	4	6	0	0	4	6	0	3	4	6	0	2	4	6	
<b>Test Case No.</b>		<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>	<b>20</b>	<b>21</b>	<b>22</b>	<b>23</b>	<b>24</b>	<b>25</b>						
<b>Start View Mode</b>		4	4	4	5	5	5	5	6	7	7						
<b>Navigation Command (Input)</b>		F	P	FV	F	N	L	FV	EV	EV	LK						
<b>End View Mode</b>		0	5	6	0	4	4	6	0	0	2						

## 4.5 TỔ HỢP ĐIỀU KIỆN

Liên quan phân tích mỗi liên hệ kết hợp các biến như tập browser setting. Mỗi kết hợp thể hiện điều kiện được kiểm thử với cùng kịch bản kiểm thử hoặc thủ tục.

### Các bước chung

- Nhận diện các biến.
- Nhận diện số giá trị duy nhất có thể của mỗi biến.
- Tạo bảng thể hiện kết hợp hoàn chỉnh duy nhất của các điều kiện hình thành các biến và giá trị.

Application Under Test: Yahoo Mail

Example: We have a Test Requirement:

TR-ID	Test Requirements	Notes
TRQ01	User is able to login with valid username and password	



From that Test Requirement, we can design a simple Test Case as below.

TC-01: Login to Yahoo Mail with valid username/password.

Step	Description	Data	Expected Result
1	Open IE and go to <a href="http://mail.yahoo.com">http://mail.yahoo.com</a>		Yahoo page is displayed.
2	Enter a valid yahoo ID into the 'Yahoo ID' text box	Yahoo ID: kimlg1	
3	Enter a valid password into the 'Password' text box	Password: logigear	
4	Click on the 'Sign in' button		Login is successful.

## TÓM LƯỢC

Bài học này cung cấp các kiến thức về:

- Các kỹ thuật lớp tương đương và giá trị biến
- Phân tích ràng buộc
- Tổ hợp điều kiện

# BÀI 5: THIẾT KẾ TEST-CASE: WHITE-BOX

Nội dung gồm các phần sau:

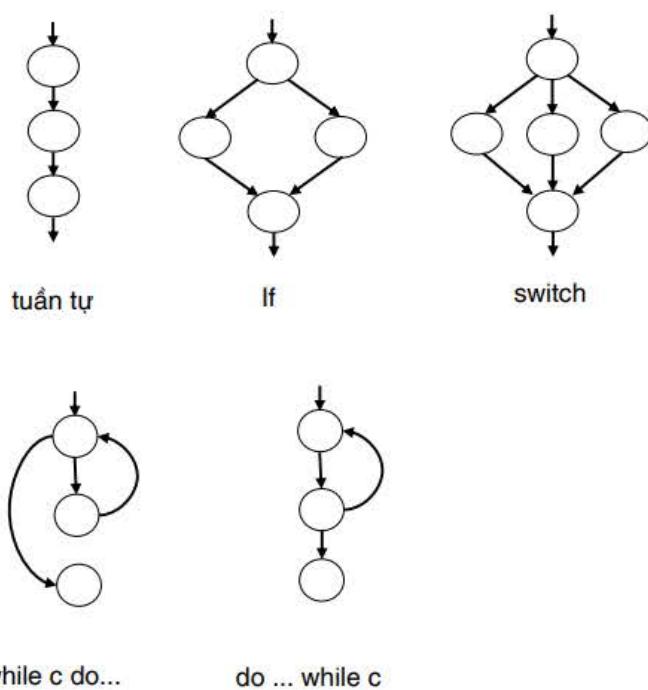
- Kiểm thử đường cơ bản
- Kiểm thử luồng điều khiển và độ bao phủ
- Kiểm thử vòng lặp
- Kiểm thử luồng dữ liệu

## 5.1 KIỂM THỬ ĐƯỜNG CƠ BẢN

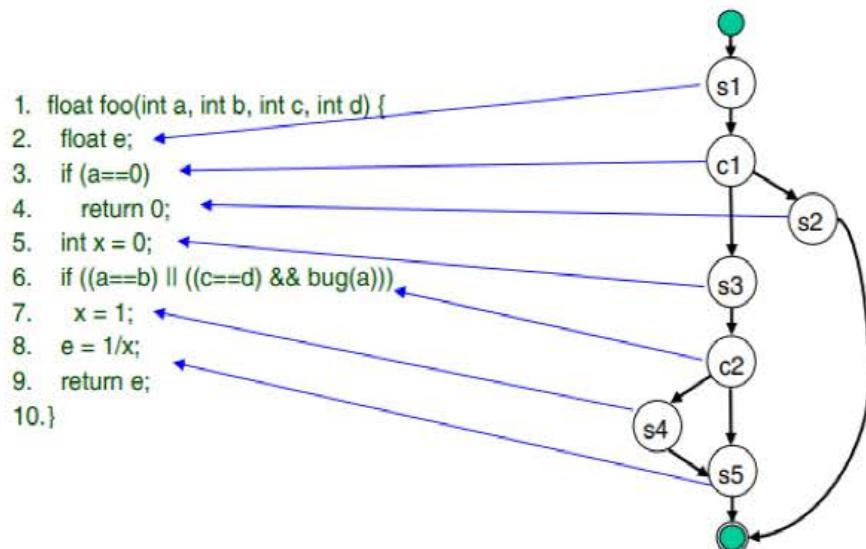
- Là một kỹ thuật dùng trong kiểm thử hộp trắng được Tom McCabe đưa ra đầu tiên. Đồ thị dòng gần giống đồ thị luồng điều khiển của chương trình.
- Là một trong nhiều phương pháp miêu tả thuật giải. Đây là phương pháp trực quan cho chúng ta thấy dễ dàng các thành phần của thuật giải và mối quan hệ trong 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 giữa chúng.
- Các loại nút trong đồ thị dòng điều khiển:

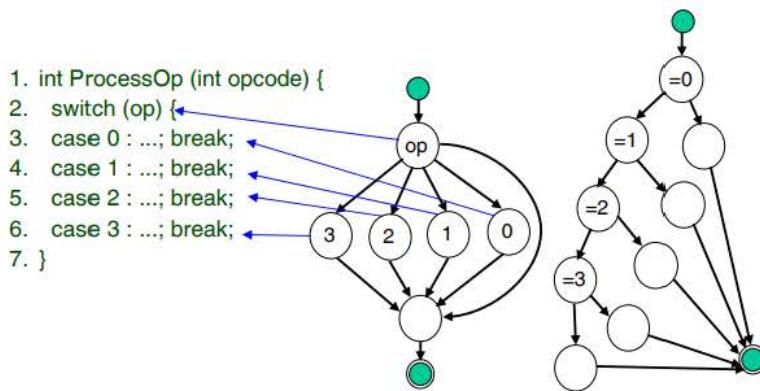


- Các kiểu cấu trúc thành phần đồ thị dòng:



- Thí dụ :





- 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.

## 5.2 KIỂM THỬ LUỒNG ĐIỀU KHIỂN VÀ ĐỘ BAO PHỦ

### 5.2.1 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.

### 5.2.2 Các cấp độ bao phủ (Coverage)

**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. Phù 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.

## 5.3 KIỂM THỬ VÒNG LẶP

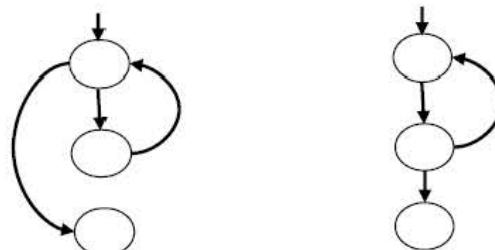
Vòng lặp là các lệnh phổ biến và là cơ bản trong các ngôn ngữ lập trình. Tuy nhiên, việc kiểm thử đối với các vòng lặp là rất phức tạp, việc kiểm thử tất cả các trường hợp của vòng lặp nhiều khi là không thể vì số lượng các trường hợp kiểm thử là rất lớn. Trong phần này sẽ trình bày một số cải tiến để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

1. lệnh lặp đơn giản: thân chỉ chứa các lệnh khác chứ không chứa lệnh lặp khác.
2. lệnh lặp lồng nhau: thân chứa ít nhất lệnh lặp khác...

3. lệnh lặp liền kề: 2 hay nhiều lệnh lặp kế tiếp nhau
  4. lệnh lặp giao nhau: 2 hay nhiều lệnh lặp giao nhau.

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



**while c do...**                    **do ... while c**

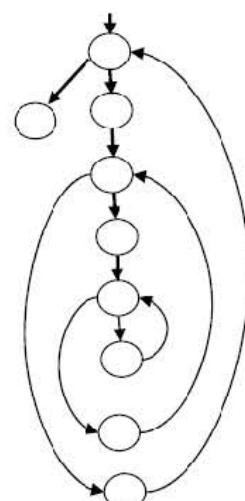
Nên chọn các test case để kiểm thử thân lệnh lắp ở các vị trí sau:

- 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 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.



3. Kiểm thử các vòng lặp liên kết: Kiểm thử tuần tự từng vòng lặp từ trên xuống, mỗi vòng thực hiện kiểm thử.

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

## 5.4 KIỂM THỬ LUỒNG DỮ LIỆU

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

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

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

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

Chỉ có những lệnh nằm trong tầm vực truy xuất biến mới có thể truy xuất/xử lý được biến. Tầm vực truy xuất biến là tập các lệnh được phép truy xuất biến đó. Thường các ngôn ngữ lập trình cho phép định nghĩa tầm vực cho mỗi biến thuộc 1 trong 3 mức chính yếu: toàn cục, cục bộ trong từng module, cục bộ trong từng hàm chức năng.

### Hệ thống ký hiệu trạng thái dữ liệu

<b>Hệ thống ký hiệu</b>	
<b>D</b>	defined, created, initialized
<b>K</b>	killed, terminated, undefined
<b>U</b>	Used c – used in a computation (sử dụng trong biểu thức tính toán) p – used in a predicate (sử dụng trong các biểu thức điều kiện)
<b>~x</b>	Cho biết trước khi tất cả hành động liên quan đến x
<b>x~</b>	Cho biết tất cả hành động không có thông báo liên quan đến x

### Một số ví dụ

- $v = \text{expression}$ 
  - c – use của các biến trong biểu thức
  - definition của v

- read (v1, v2, ..., vn)
  - definitions của v1, ..., vn
- write (v1, v2, ..., vn)
  - c - uses của v1, ..., vn
- method call: P (c1, ..., cn)
  - definition của mỗi tham số
- While B do S
  - p – use của mỗi biến trong biểu thức điều kiện

Ví dụ:

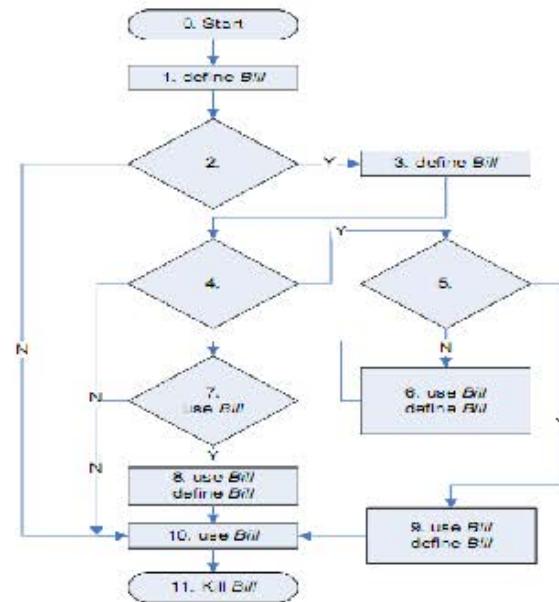
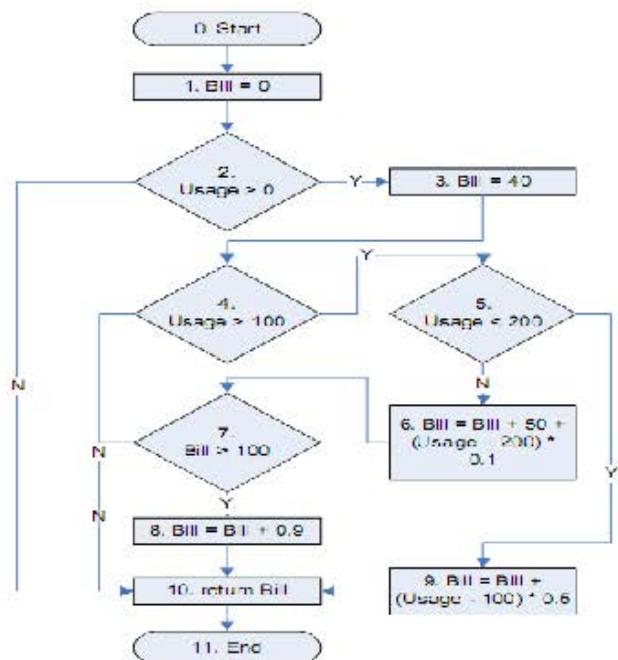
	<i>Def</i>	<i>C-use</i>	<i>P-use</i>
1. read (x, y);	x, y		
2. z = x + 2;	z	x	
3. if (z < y)			
4. w = x + 1;	w	x	
else			
5. y = y + 1;	y	y	
6. print (x, y, w, z);		x, y, w, z	

## Các chiến lược thiết kế test case

Ví dụ:

```
public static double calculateBill (int usage)
{
    double Bill = 0;
    if(usage > 0)
    {
        Bill = 40;
    }
    if(usage > 100)
    {
        if(usage <= 200)
        {
            Bill = Bill + (usage - 100) * 0.5;
        }
        else
        {
            Bill = Bill + 50 + (usage - 200) * 0.1;
            if(Bill >= 100)
            {
                Bill = Bill * 0.9;
            }
        }
    }
    return Bill;
}
```

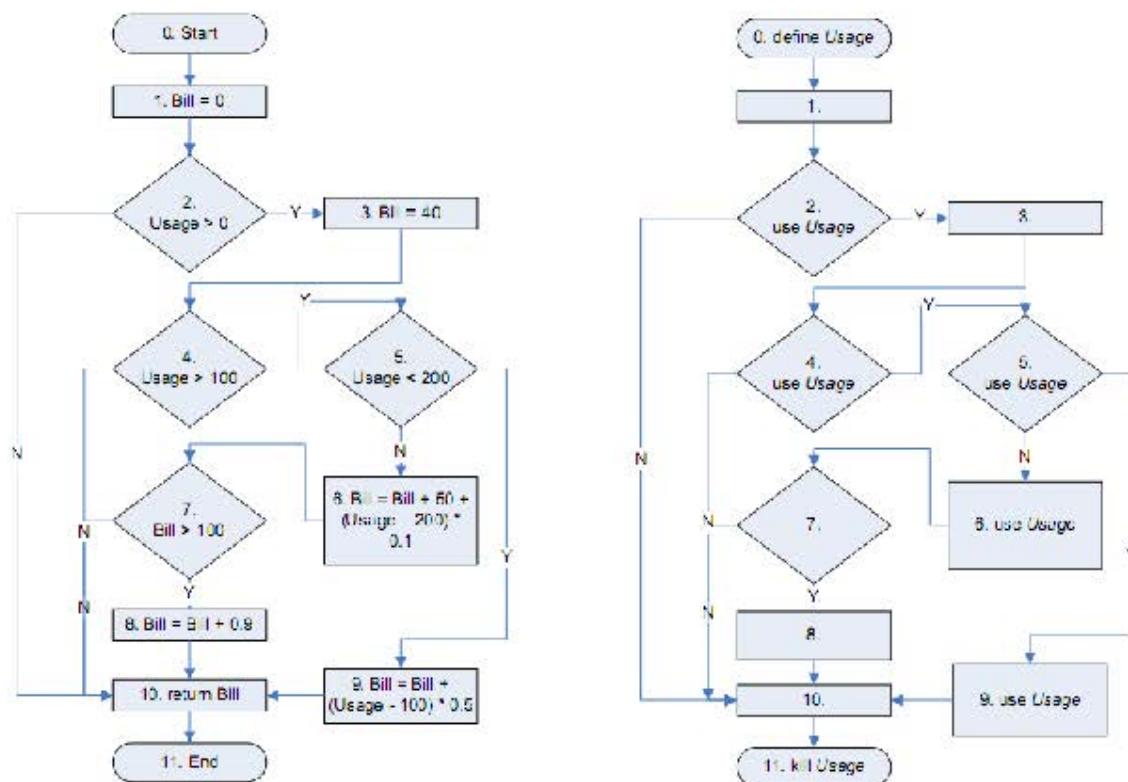
Xét biến "Bill"



Bảng mô tả biến "Bill"

	Anomaly	Explanation
~d	0-1	Allowed. Normal case
dd	0-1-2-3	Potential bug. Double definition.
du	3-4-5-6	Allowed. Normal case.
ud	6	Allowed. Data is used and then redefined.
uk	10-11	Allowed.
dd	1-2-3	Potential bug. Double definition.
uu	7-8	Allowed. Normal case.
k~	11	Allowed. Normal case.

Xét biến "Usage"



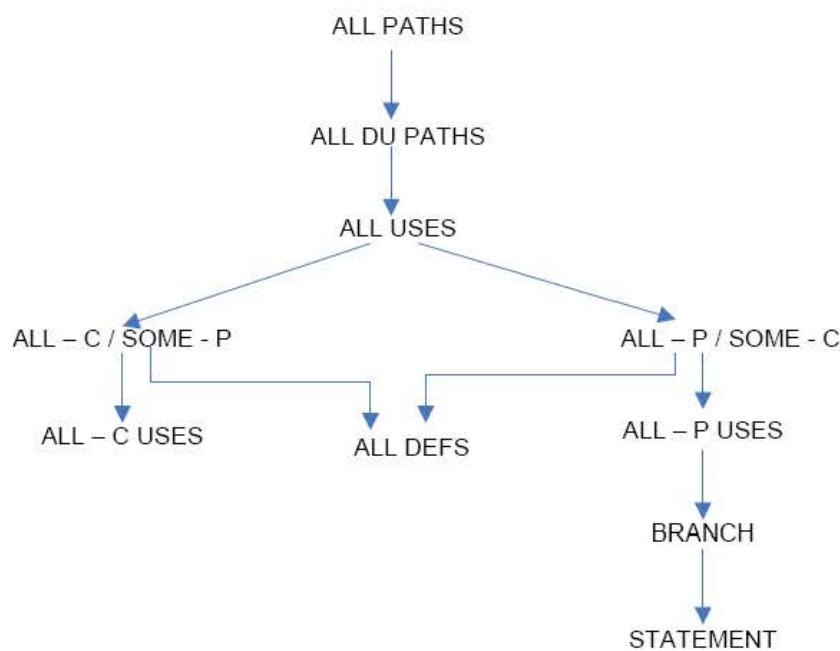
Bảng mô tả biến "Usage"

Anomaly		Explanation
~d	0	Allowed.
du	0-1-2	Allowed. Normal case.
uk	9-10-11	Allowed.
uu	5-6	Allowed. Normal case.
k~	11	Allowed. Normal case.

#### Data-flow testing paths for each variable

Strategy	Bill	Usage	Strategy	Bill	Usage
All uses (AU)	3-4-5-6 6-7-8 8-10 3-4-5-9	0-1-2 0-1-2-3-4 0-1-2-3-4-5 0-1-2-3-4-5-6 0-1-2-3-4-5-9	All p - uses/ same c (APU - C)	1-2-2-4-5-6-7 3-1-2-6-7 6-7 8-10 9-10	0-1-2 0-1-2-3-4 0-1-2-3-4-5
All p - uses (APU)	1-2-3-4-5-6-7 3-4-5-6-7 6-7	0-1-2 0-1-2-3-4 0-1-2-3-4-5	All c - uses/ same p (ACU - P)	1-2-10 3-4-5-6 3-4-5-9 6-7-8 8-10 9-10	0-1-2-3-4-5-6 0-1-2-3-4-5-9
All c - uses (ACU)	1-2-10 3-4-5-6 3-4-5-9 3-4-10 6-7-8 6-7-10 8-10 9-10	0-1-2-3-4-5-6 0-1-2-3-4-5-9	All do (ADP)	(ACU - P) (APU - C)	(ACU - P) (APU - C)
All definition (AD)	1-2-10 3-4-5-6 6-7 8-10 9-10	0-1-2			

### Mối quan hệ giữa các chiến lược data-flow test



## TÓM LƯỢC

*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

# BÀI 6: BÀI TẬP THIẾT KẾ TESTCASE

Nội dung gồm các phần sau:

- Các bài tập kiểm thử hộp đen
- Các bài tập kiểm thử hộp trắng

## 6.1 BÀI TẬP KIỂM THỬ HỘP ĐEN

### 6.1.1 Bài tập 1

Nếu bạn đi xe điện chuyến 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.

### 6.1.2 Bài tập 2

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.

### 6.1.3 Bài tập 3

Viết chương trình dịch, trong đó có câu lệnh FOR, đặc tả câu lệnh FOR như sau: “Lệnh FOR chỉ chấp nhận một tham số duy nhất là biến đếm. Tên biến không được sử

dụng quá hai ký tự khác rỗng. Sau ký hiệu = là cận dưới và cận trên của biến đếm. Các cận trên và cận dưới là các số nguyên dương và được đặt giữa từ khóa TO".

Dùng phương pháp phân hoạch tương đương, thiết kế các ca kiểm thử cho câu lệnh FOR

#### 6.1.4 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

#### 6.1.5 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.

## **6.2 BÀI TẬP KIỂM THỬ HỘP TRẮNG**

---

### **6.2.1 Bài tập 1**

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

```
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;
}
```

### **6.2.2 Bài tập 2**

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;
}
```

### 6.2.3 Bài tập 3

#### 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;
}
```

- Thiết kế ca kiểm thử thỏa mãn phủ cấp 4

### 6.2.4 Bài tập 4

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

```

function goodstring(var count: integer): boolean;
var ch: char;
begin
  goodstring:= false;
  count:=0;
  read(ch);
  if ch='a' then
    begin
      2
      read(ch);
      3
      while (ch='b') or (ch='c') do
        begin
          count:= count+1;
          read(ch);
        end;
      if ch='x' then goodstring= true;
      end;
    end;
  end;

```

### 6.2.5 Bài tập 5

**1. Xác định số test case và liệt kê các testcase với đoạn chương trình sau theo mẫu test case**

Write test cases for below method specification

**XYZ** is integer

THE IS  
START

- Check the value of X input
    - If is 1 or 2
      - Return A
    - Else
      - Check the value of Y input
        - If  $Y \leq 10$ 
          - Return B
        - Else
          - Check the value of Z input
            - If  $Z < 5$ 
              - Return C
            - Else
              - Return D

FND

**Answer (Number of test cases is at least 7)**

No	Input	Expected result	Type of test cases
1	X=1	A	Normal
2	X=2	A	Normal
3	X=5,Y=9	B	Normal
4	X=5,Y=10	B	Boundary
5	X=5,Y=15,Z=4	C	Normal
6	X=5,Y=15,Z=5	D	Boundary
7	X=5,Y=15,Z=6	D	Normal

## TÓM LƯỢC

*Bài học này cung cấp các kiến thức về:*

- *Các dạng bài tập kiểm thử hộp đen*
- *Các dạng bài tập kiểm thử hộp trắng*

# BÀI 7: LỖI PHẦN MỀM

Nội dung gồm các phần sau:

- *Lỗi phần mềm*
- *Nguyên nhân gây ra lỗi*
- *Các lỗi thường gặp*

## 7.1 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.

### 7.1.1 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, lỗ hổng, 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 vấn 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.

## 7.2 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 thông 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 nhật 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 xướng, không được chấp thuận các cải tiến cho phần mềm, mà không có sự chấp thuận của khách hàng, thường xuyên bỏ qua các yêu cầu có vẻ nhỏ 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 cần thiết ở mỗi bước của quá trình. Chúng có tầm quan trọng đặc biệt trong các hệ thống phần mềm phức tạp, nơi các tiến trình được tiến hành một vài bước, mỗi bước trong số đó có thể có nhiều kiểu dữ liệu và cho phép kiểm tra các kết quả trung gian.

#### - **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.

## **7.3 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 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 nhấp "Save and Close", thông tin trong form lưu và đóng form lại. Nếu nhấp vào button "Save and Close" mà không đóng form, thì đó là lỗi dòng điều khiển.

## **7.4 TÌM LỖI VÀ PHÂN TÍCH LỖI**

---

### **7.4.1 Checklist kiểm tra mã nguồn**

1. Các lỗi truy xuất dữ liệu (Data Reference Errors)
2. Các lỗi định nghĩa/khai báo dữ liệu (Data-Declaration Errors)
3. Các lỗi tính toán (Computation Errors)
4. Các lỗi so sánh (Comparison Errors)

5. Các lỗi luồng điều khiển (Control-Flow Errors)
6. Các lỗi giao tiếp (Interface Errors)
7. Các lỗi nhập/xuất (Input/Output Errors)
8. 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, bufin, 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.

## 7.4.2 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 đồi với người sử dụng.

## TÓM LƯỢC

*Bài học này cung cấp các kiến thức về:*

- *Tìm lỗi*
- *Các dạng lỗi thường gặp và nguyên nhân gây ra lỗi*

# BÀI 8: CÁC HỆ THỐNG QUẢN LÝ BUG

## **8.1 GIỚI THIỆU HỆ THỐNG QUẢN LÝ BUG**

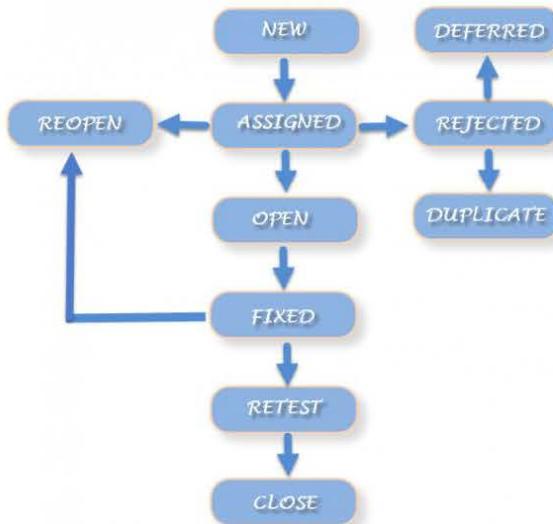
### **8.1.1 Giới thiệu**

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, Apteligent, Promoter.io, Lighthouse.

## 8.2 VÒNG ĐỜI CỦA BUG TRÊN HỆ THỐNG QUẢN LÝ BUG



Hình 8.1 Vòng đời của bug

### 8.2.1 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).

### 8.2.2 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.

### 8.2.3 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.

### 8.2.4 Môi trường kiểm thử

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

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, nền 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.

### **8.3 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: <http://www.bugzilla.org/>

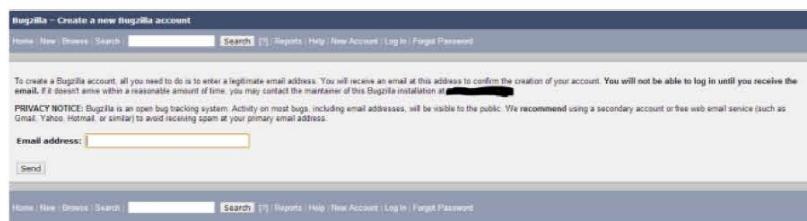
## 1. Tạo mới User và login

Bước 1: Vào trang Bugzilla



**Hình 8.2: Trang chủ Bugzilla**

Bước 2: Click và Tạo Account



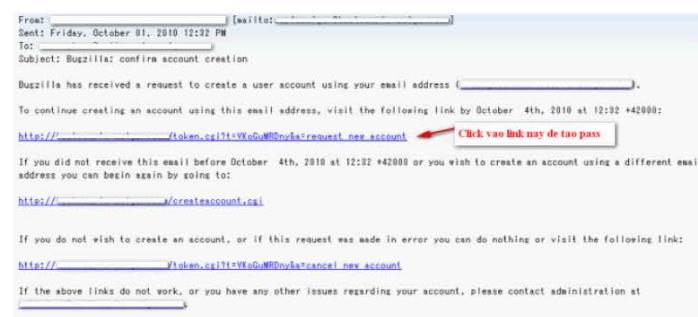
**Hình 8.3: Tạo tài khoản trên Bugzilla**

Nhập địa chỉ mail vào

Click "SEND"

Mail tự gửi đến mail cá nhân

Bước 3: Vào mail cá nhân kích hoạt



**Hình 8.4: Kích hoạt tài khoản**

Bước 4: Nhập Email và Pass tạo tài khoản

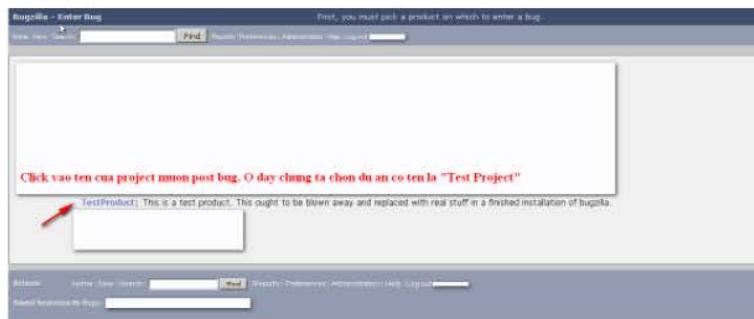


**Hình 8.5: Đăng nhập tài khoản**

Sau khi tạo xong -> chọn SEND

## Đăng nhập Account vào sử dụng Bugzilla

## 2. Tạo Bug



**Hình 8.6: Tạo bug**

**Hình 8.6: Ghi thông tin khi tạo bug**

## 1. Cập nhật thông tin cho Bug

Bugzilla - Bug 278

[Edit](#) [Raw](#) [View](#) [Edit History](#) [Edit Summary](#) [Edit Description](#) [Edit Comments](#) [Edit File Attachment](#) [Edit Component](#) [Edit Status](#) [Edit Resolution](#) [Edit Version](#) [Edit Platform](#) [Edit Assigned To](#) [Edit Dependencies](#) [Edit Blocks](#) [Show Dependency Tree](#)

**Bug 278 - Two video testing and one using bugzilla site**

**Requester:** 2010-10-06 10:07 UTC [View](#) [Edit](#)  
**Modified:** 2010-10-06 10:07 UTC (initial)  
**CC Link:** [\[?\]](#) [Add link To CC List](#)

**Browser:** [IE 2010](#) [View](#) [Edit](#)  
**Revision:** 101704

**Product:** TestProduct [View](#) [Edit](#)  
**Component:** TestComponent [View](#) [Edit](#)  
**Version:** unspecified [View](#) [Edit](#)  
**Platform:** unspecified [View](#) [Edit](#)  
**Assigned To:** [\[?\]](#) [View](#) [Edit](#)  
**Dependencies:** [\[?\]](#) [View](#) [Edit](#)  
**Blocks:** [\[?\]](#) [View](#) [Edit](#)  
**Show Dependency Tree**

**Restrict drop visibility:** Only users in all of the selected groups can view this bug.

**Attachment:** [\[?\]](#) [View](#) [Edit](#) [Delete](#) [Download](#) [Edit Summary](#) [Edit Description](#) [Edit Comment](#)

Obj. Ext.	Current file	Starts Working	Starts Left	ReCorrigible	Comments	Deadline
500.0	0.0	0.0	0.0	0	0.0	2010-10-06 10:00:00
Summary: Two video testing and one using bugzilla site						

**Attachments:** [\[?\]](#) [View](#) [Edit](#) [Delete](#) [Download](#) [Edit Summary](#) [Edit Description](#) [Edit Comment](#)

Two video testing and one using bugzilla site file name(.mp4) [\[?\]](#) [View](#) [Edit](#)  
[\[?\]](#) [View](#) [Edit](#) [Delete](#) [Download](#) [Edit Summary](#) [Edit Description](#) [Edit Comment](#)

**Additional Comments:**

[New](#) [Cancel](#)

**Description/Priority:** [\[?\]](#) [View](#) [Edit](#) [Delete](#) [Download](#) [Edit Summary](#) [Edit Description](#) [Edit Comment](#)

**Description/Priority:** 2010-10-06 10:07:34 (+1 day)  
**Description:** [\[?\]](#) [View](#) [Edit](#) [Delete](#) [Download](#) [Edit Summary](#) [Edit Description](#) [Edit Comment](#)

The two video testing and one using bugzilla site  
1. Two video testing and one using bugzilla site  
2. Two video testing and one using bugzilla site  
3. Two video testing and one using bugzilla site

**Dep Link:** [\[?\]](#) [View](#) [Edit](#) [Delete](#) [Download](#) [Edit Summary](#) [Edit Description](#) [Edit Comment](#)

[New](#) [Cancel](#)

**Hình 8.7: Cập nhật thông tin Bug**

## TÓM LƯỢC

*Bài học này cung cấp các kiến thức về:*

- Giới thiệu lỗi
  - Hệ thống quản lý bug

# BÀI 9: KIỂM THỬ TỰ ĐỘNG

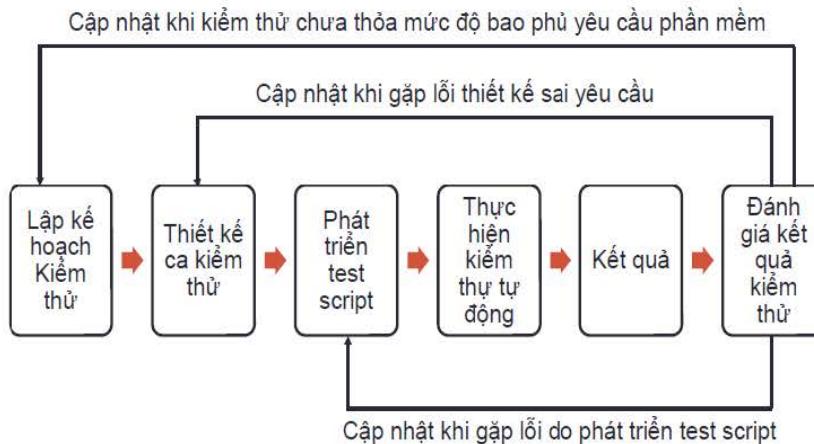
Nội dung gồm các phần sau:

- *Giới thiệu kiểm thử tự động*
- *Kiểm thử dựa trên hành động*

## 9.1 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. Test script: tạo thủ công hoặc tạo tự động dùng công cụ kiểm thử tự động.

### 9.1.1 Quy trình kiểm thử tự động



**Hình 9.1: Quy trình kiểm thử**

1. *Tạo test script:* Giai đoạn này ta dùng test tool để ghi lại các thao tác lên PM cần kiểm tra và tự động sinh ra test script
2. *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
3. *Chạy test script để kiểm thử tự động:* Giám sát hoạt động kiểm tra phần mềm của test script
4. *Đánh giá kết quả:* Kiểm tra kết quả thông báo sau khi thực hiện kiểm thử tự động. Sau đó bổ sung, chỉnh sửa những sai sót

### 9.1.2 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
  - Giả 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

### 9.1.3 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 assertNotSame():** So sánh địa chỉ vùng nhớ của hai tham chiếu hai đối tượng bằng cách sử dụng toán tử ==. Phép thử thất bại trả về nếu cả hai đều tham chiếu đến cùng một đối tượng.
- **Boolean assertNull():** So sánh tham chiếu của một đối tượng với giá trị Null. Phép thử thất bại nếu đối tượng không là Null.
- **Boolean assertSame():** So sánh địa chỉ vùng nhớ của hai tham chiếu đối tượng bằng cách sử dụng toán tử ==. Phép thử thất bại nếu cả hai không tham chiếu đến cùng một đối tượng.
- **Boolean assertTrue():** Đánh giá một biểu thức logic. Phép thử thất bại nếu biểu thức sai.

- **fail():** Phương thức này làm cho test hiện tại thất bại, phương thức này thường được sử dụng khi xử lý các ngoại lệ.

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

*Ví dụ phương thức test hai xâu có trùng nhau không.*

```
@Test
```

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

### **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

## 9.2 GIỚI THIỆU 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 bất 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 miền 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 đến 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.)

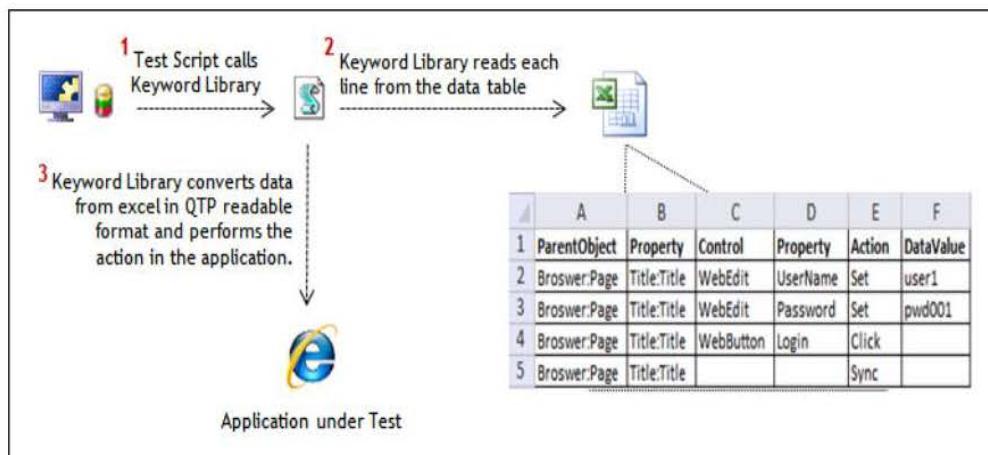
Một khi các yêu cầu kiểm tra được xác định, chúng phục vụ như là một lộ trình để phát triển các trường hợp kiểm thử trong mô-đun và tài liệu cho mục đích của các bài kiểm tra. Mỗi trường hợp kiểm thử được liên kết với một hoặc nhiều yêu cầu kiểm tra, và mỗi yêu cầu kiểm tra phải được giải quyết bởi một 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 9.2” Keyword driven Framework**

## TÓM LƯỢC

*Bài học này cung cấp các kiến thức về:*

- *Giới thiệu kiểm thử tự động*
- *Giới thiệu kiểm thử dựa trên hành động*

# BÀI 10: CÁC CÔNG CỤ HỖ TRỢ KIỂM THỬ TỰ ĐỘNG

Nội dung gồm các phần sau:

- Công cụ kiểm thử tự động QuickTest Pro
- Công cụ kiểm thử tự động Selenium

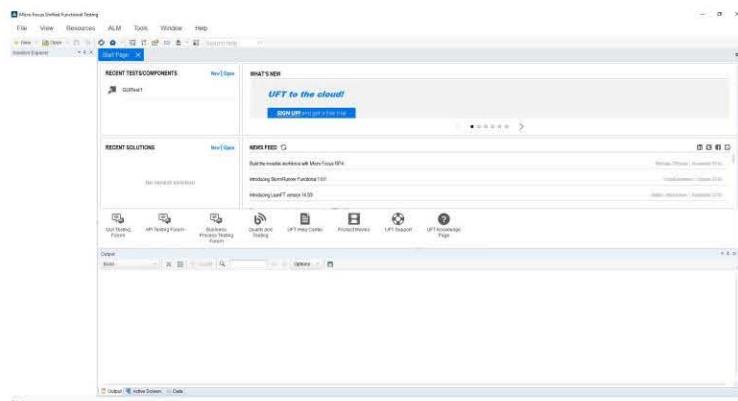
## 10.1 QUICKTEST PRO

### 10.1.1 Giới thiệu

QTP là phần mềm dùng để kiểm tra chức năng (functional test) và cho phép thực hiện kiểm tra hồi qui (regression test) một cách tự động

Đây cũng là công cụ áp dụng phương pháp Keyword – Driven, một kỹ thuật scripting hiện đại, cho phép kỹ thuật viên bổ sung test case bằng cách tạo file mô tả cho nó mà không cần chỉnh sửa hay bổ sung bất cứ script nào cả.

Nó cũng phù hợp trong tình huống chuyển giao công việc mà người mới tiếp nhận chưa có thời gian hoặc không hiểu script vẫn có thể thực hiện kiểm thử phần mềm theo đúng yêu cầu.



Hình 10.1: Giao diện QTP

Loại phần mềm hỗ trợ:

- Ứng dụng Windows chuẩn /Win32.
- Ứng dụng web theo chuẩn HTML, XML chạy trong trình duyệt
- Visual Basic.
- ActiveX.
- QTP hỗ trợ Unicode (UTF-8, UTF-16).

Một số chương trình khác đòi hỏi phải cài đặt thêm các thư viện

<b>.NET</b>	<ul style="list-style-type: none"> <li>• .NET Framework 1.0, 1.1, 2.0 beta</li> <li>• Các đối tượng chuẩn của .NET và các đối tượng khác thừa kế từ các đối tượng chuẩn.</li> </ul>
<b>Java</b>	<ul style="list-style-type: none"> <li>• Sun JDK 1.1 – 1.4.2</li> <li>• IBM JDK 1.2 – 1.4</li> </ul>
<b>Oracle</b>	<ul style="list-style-type: none"> <li>• Oracle Applications 11.5.7, 11.5.8, 11.5.9</li> </ul>
<b>People Soft</b>	<ul style="list-style-type: none"> <li>• PeopleSoft Enterprise 8.0 – 8.8</li> </ul>
<b>SAP</b>	<ul style="list-style-type: none"> <li>• SAP GUI HTML 4.6D, 6.10, 6.20</li> <li>• SAP Workplace 2.11</li> <li>• SAP Enterprise Portal 5.0</li> </ul>
<b>Siebel</b>	<ul style="list-style-type: none"> <li>• Siebel 7.0, 7.5, 7.7</li> </ul>
<b>Terminal Emulators</b>	<ul style="list-style-type: none"> <li>• Attachmate EXTRA! 6.7, 7.1</li> <li>• Attachmate EXTRA! Terminal Viewer 3.1 Java sessions</li> <li>• IBM Personal Communications</li> <li>• ...</li> </ul>

Ưu điểm của QTP

- Hỗ trợ record và playback.
- Dễ sử dụng, bảo trì, tạo test script nhanh.
- Cung cấp dữ liệu kiểm thử rõ ràng và dễ hiểu.
- Kiểm thử phiên bản mới của ứng dụng với rất ít sự thay đổi.
- Hỗ trợ làm việc theo nhóm thông qua chia sẻ thư viện.
- Kiểm thử tự động trên nhiều trình duyệt

Nhược điểm của QTP

- Chỉ chạy trên môi trường window

- Chi phí khá đắt (khoảng 3000\$/1 thiết bị) nên không được sử dụng rộng rãi

## 10.1.2 Cài đặt & thành phần QTP

Các thành phần của QTP

- Action:** Action ghi lại các bước thực hiện kiểm thử và nó có thể được sử dụng lại nhiều lần. Trong một test script có thể có nhiều action

```

1 Browser("YouTube").Page("YouTube").Sync
2 Browser("YouTube").Page("https://www.youtube.com/results")
3 Browser("YouTube").Page("design ui react native").WebEdit("Tim Kiếm").Set "design ui react native"
4 Browser("YouTube").Page("design ui react native").WebEdit("Tim Kiếm").Submit
5 Browser("YouTube").Page("design ui react native").Link("React Native - Building").Click
6 Browser("YouTube").Page("React Native - Building").WebButton("Tạm dừng").Click
7 Browser("YouTube").Page("React Native - Building").Sync
8 Browser("YouTube").CloseAllTabs

```

Hình 10.2: Minh họa Action trong QTP

## 2. Database

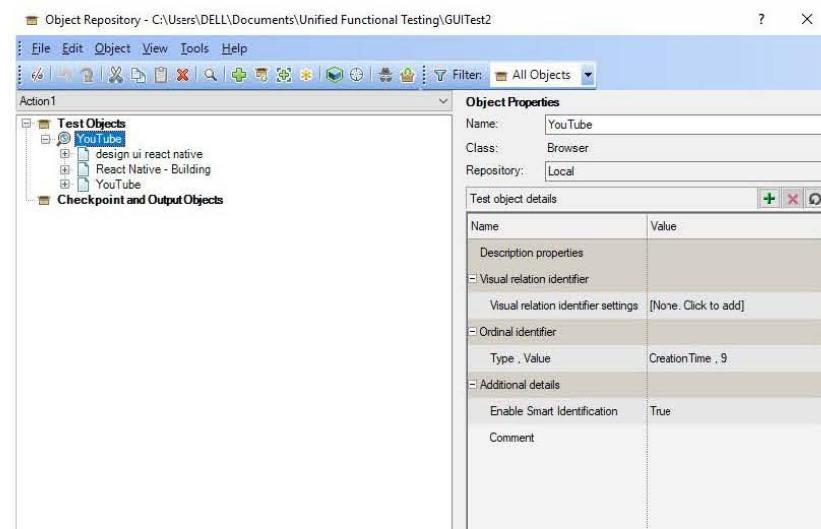
Nơi lưu trữ dữ liệu phục vụ cho kiểm thử. Một test script sẽ có một DataTable được dùng chung cho tất cả các Action

	Departure	Arrival	From Month	To Month	E	F
1	New York	San Francisco	December	December		
2	London	Portland	October	October		
3	Frankfurt	Paris	November	November		
4	Seattle	Sydney	December	December		
5						
6						
7						
8						
9						
10	Global Data Sheet			Local data sheet per action		
11						
12						
13						
14						
15						
16	Global	Flight1				

Hình 10.3: Lưu trữ dữ liệu trong QTP

## 3. Object Repository (OR)

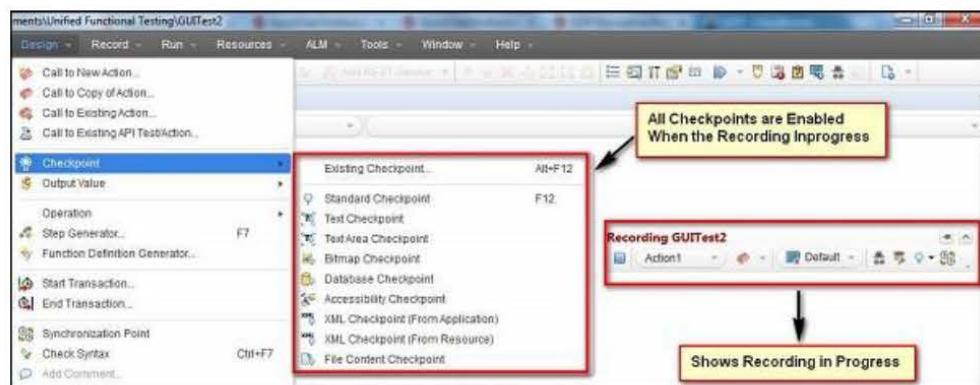
Cấu trúc theo dạng cây mô tả các đối tượng trong phần mềm được kiểm tra



Hình 10.4: Cấu trúc dạng cây OR

#### 4. Checkpoint

Có thể hiểu là nơi kiểm tra trong test script, khi chạy nó sẽ thực hiện so sánh kết quả thực tế với kết quả mong đợi. Sau khi tiến hành so sánh QTP sẽ tự động ghi lại kết quả vào Test Results.

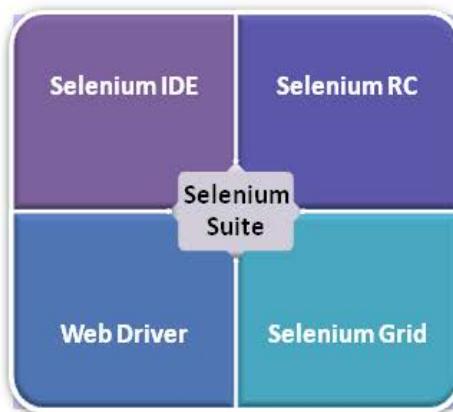


Hình 10.5: Minh họa Checkpoint

## 10.2 SELENIUM

### 10.2.1 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 10.6 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.)

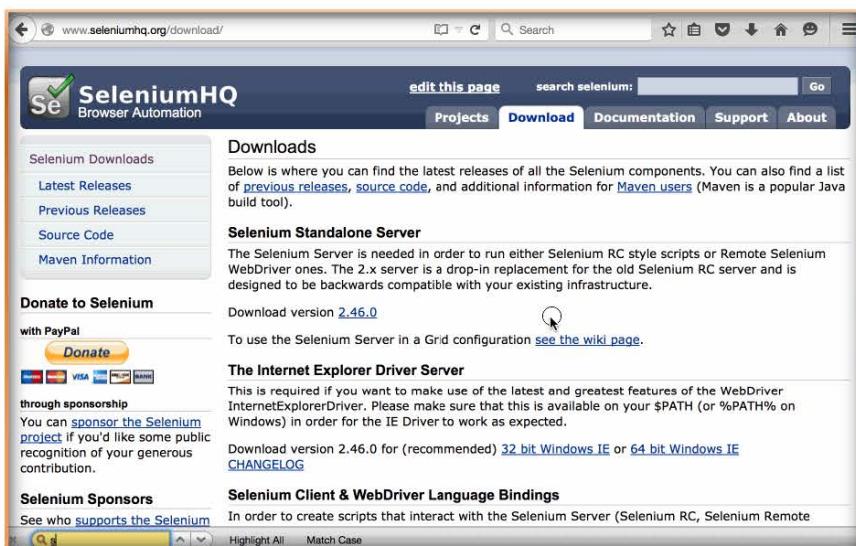
- Selenium hỗ trợ nhiều ngôn ngữ lập trình (Java, C#, Ruby, Python, Pearl etc.)
- Selenium thường xuyên được phát triển và cải tiến

### 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.

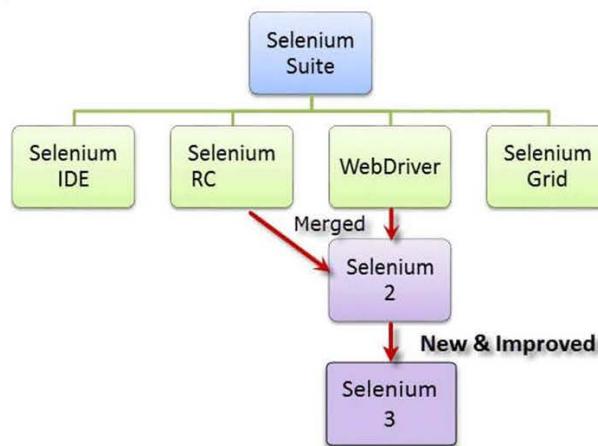
#### Cài đặt Selenium IDE

- 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.



**Hình 10.7: Trang chủ Selenium**

## 10.2.2 Thành phần của Selenium



**Hình 10.8: 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:*

- *Giới thiệu QuickTestPro*
- *Giới thiệu Selenium*

# 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

- 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

## ❖ 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 hỗ trợ kiểm thử tự động Robotium.
- Đề tài 5: Công cụ hỗ trợ kiểm thử tự động AutoIT
- Đề tài 6: Công cụ hỗ trợ kiểm thử Mantis Bug Tracker
- Đề tài 7: Công cụ hỗ trợ kiểm thử Sahi
- Đề tài 8: Công cụ hỗ trợ kiểm thử Soap UI
- Đề tài 9: Công cụ hỗ trợ kiểm thử Behavior Testing
- Chọn một trong các công cụ: HP Quick Test Professional, 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)**

No	Name	Requirements
1	<u>JUnit</u>	OS Independent
2	<u>Findbugs</u>	JRE (or JDK) 1.4.0 or later
3	<u>PMD</u>	JDK 1.3 or higher
4	<u>Checkstyle</u>	OS Independent
5	<u>EclEmma</u>	Eclipse
6	<u>Dbunit</u>	Junit
7	<u>StrutsTestCase for JUnit v1.9.5</u>	OS Independent
8	<u>Emma</u>	Java
9	<u>MockObjects</u>	OS independent
10	<u>JUnitEE</u>	Junit

No	Name	Requirements
1	<u>NUnit</u>	Windows NT/2000
2	<u>NUnitAsp</u>	Windows NT/2000
3	<u>NUnit Addin for Windows</u>	Windows

	<u>Visual Studio.NET</u>	
4	<u>NUnitForms</u>	Windows NT/2000
5	<u>csUnit</u>	csUnit has been tested using the Microsoft .NET framework 1.0 Service Pack 2 runtime on an Intel-compatible platform.
6	<u>NCover</u>	All 32-bit MS Windows (95/98/NT/2000/XP)
7	<u>VSNUnit</u>	All 32-bit MS Windows (95/98/NT/2000/XP)
8	<u>dotUnit</u>	All 32-bit MS Windows (95/98/NT/2000/XP)
9	<u>.NETUnit</u>	OS Independent (Written in an interpreted language)
10	<u>ASPUnit</u>	Microsoft Internet Information Server 5.0 or 5.1

❖ **Công cụ kiểm thử chức năng (Functional Testing)**

No	Name	Desc	Req
1	<u>Software Testing Automation Framework (STAF)</u>		Windows, Linux, Solaris, AS/400, AIX, HP-UX, Irix
2	<u>soapui</u>		Java 1.5
3	<u>Linux Test Project</u>		Linux
4	<u>jWebUnit</u>		OS Independent
5	<u>Abbot Java GUI Test Framework</u>		TBC
6	<u>Software Automation Framework Support</u>		All 32-bit MS Windows (95/98/NT/2000/XP)
7	<u>Jameleon</u>		OS Independent, JDK 1.4 or higher
8	<u>WebInject</u>		Windows, OS Independent, Linux
9	<u>Marathon</u>		Java 1.3 or later
10	<u>Solex</u>		Eclipse 2.1 or above

❖ **Công cụ kiểm hiệu năng (Performance Testing)**

No	Name	Requirements
1	<u>OpenSTA</u>	Windows 2000, NT4 and XP
2	<u>Grinder</u>	OS Independent
3	<u>TPTEST</u>	MacOS/Carbon and Win32
4	<u>Database Opensource Test Suite</u>	Linux, POSIX
5	<u>Sipp</u>	Linux/Unix/Win32-Cygwin
6	<u>WebLOAD</u>	32-bit MS Windows (NT/2000/XP),

		Linux, Windows Server 2003
7	<a href="#">OpenWebLoad</a>	Linux, DOS
8	<a href="#">Hammerhead 2 - Web Testing Tool</a>	Hammerhead has been used with Linux, Solaris and FreeBSD.
9	<a href="#">Dieseltest</a>	Windows
10	<a href="#">DBMonster</a>	OS Independent

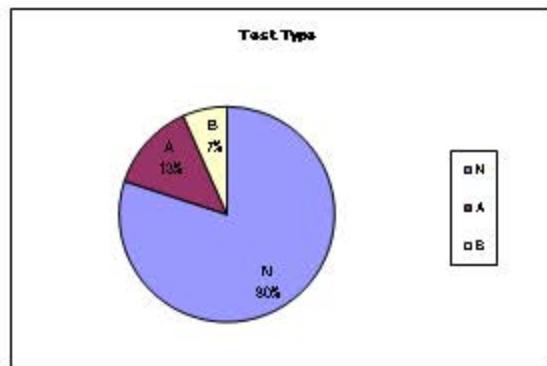
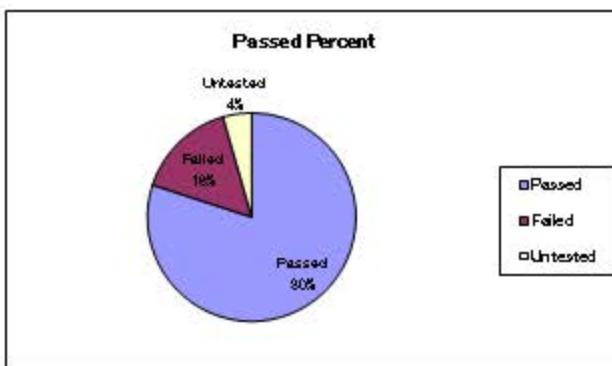
- ❖ Các biểu mẫu (Template) trong kiểm thử Test case, Test report

## UNIT TEST REPORT

<b>Project Name</b>	<Project Name>	<b>Creator</b>						
<b>Project Code</b>	<Project Code>	<b>Reviewer/Approver</b>						
<b>Document Code</b>	<Project Code>. Test Report .vx.vx	<b>Issue Date</b>	<Date when this test report is created>					
<b>Notes</b>	<List modules included in this release> ex: Release 1 includes 2 modules: Module1 and Module2							

No	Function code	Passed	Failed	Untested	N	A	B	Total Test Cases
1	Function1	12	2	1	12	2	1	15
2	Function2	12	3	0	12	2	1	15
3	Function3	12	2	1	12	2	1	15
<b>Sub total</b>		<b>36</b>	<b>7</b>	<b>2</b>	<b>36</b>	<b>6</b>	<b>3</b>	<b>45</b>

Test coverage 95,56 %  
 Test successful coverage 80,00 %  
 Normal case 80,00 %  
 Abnormal case 13,33 %  
 Boundary case 6,67 %



◀ ▶ | Guideline / Cover / FunctionList / **Test Report** / Function1 / Function2 / Function3 / Examples

COMMON DEFECTS & SOLUTION								
#	Title	Time Location	Description	Type	Severity	Cause	Proposed prevention action or Solution	
1	Check attribute file Class Attr. File 01	01/01/2018	Check attribute file may not exactly	Code	Caution	Unperformed developer	Check attribute file -> 01	
2	Same many unnecessary include file	Class Attr. File 02	Check attribute file has been many single method include file given example However, it is simple include file can only use a few lines of code and could not be use than you need to use.	Code	Caution	Development of dependency	Fix example Instead of use the Class Attr. File is more about function (included Class Attr. File) use class Attr. File from Class Attr. File, write using function. ↗ Add file here and return value name file ↗ Add file to file ↗ public void save()	

## TÀI LIỆU THAM KHẢO

1. Paul Ammann, Jeff Offutt (2008): Introduction to Software Testing, Cambridge University Press.
2. Hung Q.Nguyen (2003): Testing Application on the Web: Test planning for mobile and Internet-based system, Wiley publishing
3. LogiGear (2009): Basic Software Testing Skills, LogiGear Corporation..
4. Glenford J. Myers (2004): The art of Software Testing, John Wiley & Son