

**ĐẠI HỌC THÁI NGUYÊN**  
**KHOA CÔNG NGHỆ THÔNG TIN**

---

**LUẬN VĂN THẠC SĨ**

**MỘT SỐ KỸ THUẬT**  
**KIỂM THỬ PHẦN MỀM**

**Chuyên ngành** : **KHOA HỌC MÁY TÍNH**

**Người hướng dẫn khoa học** : **PGS. TSKH. NGUYỄN XUÂN HUY**

**Học viên thực hiện** : **CAO THỊ BÍCH LIÊN**

**Mã số** : **60 48 01**

**Thái Nguyên - Năm 2009**

## LỜI CAM ĐOAN

Tôi xin cam đoan luận văn này là công trình nghiên cứu của riêng tôi. Các số liệu kết quả nêu trong luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nghiên cứu nào khác.

# MỤC LỤC

<b>LỜI CAM ĐOAN .....</b>	<b>i</b>
<b>MỤC LỤC .....</b>	<b>ii</b>
<b>DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT .....</b>	<b>v</b>
<b>DANH MỤC CÁC BẢNG .....</b>	<b>vi</b>
<b>DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ .....</b>	<b>vii</b>
<b>MỞ ĐẦU.....</b>	<b>1</b>
<b>Chương 1 VẤN ĐỀ CHẤT LƯỢNG PHẦN MỀM VÀ KIỂM THỬ</b>	
<b>PHẦN MỀM.....</b>	<b>4</b>
<b>1.1. Sản phẩm phần mềm và vấn đề kiểm thử phần mềm.....</b>	<b>4</b>
1.1.1. Sản phẩm phần mềm là gì? .....	4
1.1.2. Thế nào là lỗi phần mềm?.....	5
1.1.3. Tại sao lỗi phần mềm xuất hiện? .....	6
1.1.4. Chi phí cho việc sửa lỗi .....	7
1.1.5. Kiểm thử phần mềm là gì?.....	8
<b>1.2. Chất lượng phần mềm.....</b>	<b>8</b>
<b>1.3. Quy trình kiểm thử phần mềm .....</b>	<b>9</b>
<b>Chương 2 CÁC KỸ THUẬT KIỂM THỬ PHẦN MỀM .....</b>	<b>12</b>
<b>2.1. Nguyên tắc cơ bản kiểm thử phần mềm .....</b>	<b>12</b>
2.1.1. Mục tiêu kiểm thử .....	12
2.1.2. Luồng thông tin kiểm thử .....	13
2.1.3. Thiết kế trường hợp kiểm thử .....	13
<b>2.2. Kỹ thuật kiểm thử hộp trắng (White-Box Testing).....</b>	<b>14</b>
2.2.1. Kiểm thử đường dẫn cơ sở (Basic Path Testing) .....	16

2.2.2. Kiểm thử cấu trúc điều khiển .....	22
<b>2.3. Kỹ thuật kiểm thử hộp đen (Black-Box Testing) .....</b>	<b>26</b>
2.3.1. Phân hoạch tương đương .....	27
2.3.2. Phân tích giá trị biên (Boundary Value Analysis) .....	30
2.3.3. Kỹ thuật đồ thị nhân-quả (Cause-Effect Graph) .....	31
2.3.4. Kiểm thử so sánh .....	34
<b>2.4. Đoán lỗi .....</b>	<b>34</b>
<b>Chương 3 CHIẾN LƯỢC KIỂM THỬ PHẦN MỀM .....</b>	<b>35</b>
<b>3.1. Nguyên lý thiết kế và kiểm thử phần mềm .....</b>	<b>35</b>
<b>3.2. Phương pháp tiếp cận kiểm thử phần mềm .....</b>	<b>36</b>
3.2.1. Xác minh và thẩm định .....	37
3.2.2. Tổ chức việc kiểm thử .....	37
3.2.3. Chiến lược kiểm thử phần mềm .....	38
3.2.4. Điều kiện hoàn thành kiểm thử .....	39
<b>3.3. Kiểm thử đơn vị .....</b>	<b>42</b>
3.3.1. Các lý do của kiểm thử đơn vị .....	42
3.3.2. Các thủ tục kiểm thử đơn vị .....	45
<b>3.4. Kiểm thử tích hợp .....</b>	<b>45</b>
3.4.1. Kiểm thử tích hợp từ trên xuống (Top-Down Integration) .....	46
3.4.2. Chiến lược kiểm thử từ dưới lên (Bottom-Up Testing) .....	47
3.4.3. Kiểm thử hồi qui .....	48
3.4.4. Các ghi chú trên kiểm thử tích hợp .....	48
<b>3.5. Kiểm thử tính hợp lệ .....</b>	<b>50</b>
3.5.1. Điều kiện kiểm thử tính hợp lệ .....	50

3.5.2. Duyệt lại cấu hình .....	51
3.5.3. Kiểm thử Alpha và Beta .....	51
<b>3.6. Kiểm thử hệ thống .....</b>	<b>52</b>
3.6.1. Kiểm thử khôi phục .....	52
3.6.2. Kiểm thử bảo mật .....	52
3.6.3. Kiểm thử ứng suất .....	53
3.6.4. Kiểm thử khả năng thực hiện .....	53
<b>Chương 4 MỘT SỐ ỨNG DỤNG CỤ THỂ CỦA QUI TRÌNH KIỂM THỬ .....</b>	<b>54</b>
<b>4.1. Mục tiêu .....</b>	<b>54</b>
<b>4.2. Phương pháp luận .....</b>	<b>54</b>
4.2.1. Tổng quan về các phương pháp .....	54
4.2.2. Phạm vi giải quyết .....	54
4.2.3. Phân loại các kiểu kiểm thử .....	55
4.2.4. Tổ chức giao diện kiểm thử .....	56
<b>4.3. Phát sinh các trường hợp kiểm thử .....</b>	<b>57</b>
4.3.1. Chiến lược kiểm thử .....	57
4.3.2. Kiểm thử đơn vị .....	57
4.3.3. Kiểm thử khả năng thực hiện .....	65
<b>KẾT LUẬN .....</b>	<b>66</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>67</b>

## **DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT**

<b>BRO</b>	<b>:</b>	<b>Kiểm thử nhánh và toán tử quan hệ</b>
<b>BVA</b>	<b>:</b>	<b>Phân tích giá trị biên</b>
<b>DU</b>	<b>:</b>	<b>Một chuỗi khai báo - sử dụng</b>
<b>E</b>	<b>:</b>	<b>Là số cạnh của đồ thị lưu trình</b>
<b>N</b>	<b>:</b>	<b>Là số đỉnh của đồ thị lưu trình</b>
<b>P</b>	<b>:</b>	<b>Số đỉnh điều kiện có trong đồ thị lưu trình</b>
<b>R</b>	<b>:</b>	<b>Số vùng của đồ thị lưu trình</b>
<b>V(G)</b>	<b>:</b>	<b>Xác định độ phức tạp Cyclomat</b>
<b>V&amp;V</b>	<b>:</b>	<b>Xác minh và thẩm định</b>

## DANH MỤC CÁC BẢNG

<b>Bảng 1.1:</b>	<b>Tỉ lệ công thức của các giai đoạn phát triển phần mềm</b>	<b>4</b>
.....		
<b>Bảng 2.1:</b>	<b>Bảng liệt kê các lớp tương đương.....</b>	<b>28</b>
<b>Bảng 2.2:</b>	<b>Ví dụ các lớp tương đương</b>	<b>29</b>
.....		
<b>Bảng 2.3:</b>	<b>Các ký hiệu trong đồ thị nhân quả</b>	<b>32</b>
.....		
<b>Bảng 2.4:</b>	<b>Ví dụ bảng quyết định</b>	<b>33</b>
.....		
<b>Bảng 3.1:</b>	<b>So sánh kiểm thử Top-Down và Bottom-Up</b>	<b>49</b>
.....		
<b>Bảng 4.1:</b>	<b>Bảng các trường hợp kiểm thử cho Module Merge</b>	<b>61</b>
.....		
<b>Bảng 4.2:</b>	<b>Các trường hợp kiểm thử cho Module Split</b>	<b>62</b>
.....		

## DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

Hình 1.1:	Sản phẩm phần mềm	5
.....		
Hình 1.2:	Các nguyên nhân gây ra lỗi phần mềm	6
.....		
Hình 1.3:	Chi phí sửa lỗi theo thời gian phát hiện lỗi	7
.....		
Hình 1.4:	Kiểm thử phần mềm trong một số ngữ cảnh	8
.....		
Hình 1.5:	Giai đoạn kiểm thử trong xử lý phần mềm	9
.....		
Hình 1.6:	Quy trình kiểm thử phần mềm	11
.....		
Hình 2.1:	Luồng thông tin kiểm thử	13
.....		
Hình 2.2:	Ví dụ chu trình điều khiển	15
.....		
Hình 2.3:	Ký hiệu đồ thị lưu trình	16



.....	
Hình 2.4: Điều kiện phức	17
.....	
Hình 2.5: Lưu đồ thuật toán và đồ thị lưu trình	17
.....	
Hình 2.6: Độ phức tạp Cyclomat	19
.....	
Hình 2.7: Ví dụ minh họa phát sinh các trường hợp kiểm thử theo đường dẫn cơ sở...	20
Hình 2.8: Các kiểu vòng lặp	25
.....	
Hình 2.9: Ví dụ đồ thị nhân quả	33
.....	
Hình 3.1: Chiến lược kiểm thử	38
.....	
Hình 3.2: Các bước kiểm thử	39
.....	
Hình 3.3: Mật độ lỗi là hàm thời gian thực hiện	41
.....	
Hình 3.4: Quan hệ giữa chi phí kiểm thử và số lỗi chưa được phát hiện	42
.....	
Hình 3.5: (a) Kiểm thử đơn vị (b) Môi trường kiểm thử đơn vị	44
.....	
Hình 3.6: Kiểm thử Top – Down	46
.....	
Hình 3.7: Tích hợp Bottom – Up	47
.....	
Hình 4.1: Giao diện kiểm thử nhúng	56

.....	
Hình 4.2: Minh họa thuật toán sắp xếp MergeSort.....	57
Hình 4.3: Đồ thị lưu trình của chức năng Merge.....	59
Hình 4.4: Giao diện điều khiển kiểm thử thuật toán MergeSort.....	64
Hình 4.5: Kết quả được ghi ra FileLog .....	64
Hình 4.6: Giao diện điều khiển kiểm thử khả năng thực hiện của các thuật toán sắp xếp..	65



# MỞ ĐẦU

## 1. Lý do chọn đề tài

Với sự phát triển như vũ bão của công nghệ thông tin nói chung và công nghệ phần mềm nói riêng, việc phát triển phần mềm ngày càng được hỗ trợ bởi nhiều công cụ tiên tiến, giúp cho việc xây dựng phần mềm đỡ mệt mỏi và hiệu quả hơn. Tuy nhiên, vì độ phức tạp của phần mềm và những giới hạn về thời gian và chi phí, cho dù các hoạt động đảm bảo chất lượng phần mềm nói chung và kiểm thử nói riêng ngày càng chặt chẽ và khoa học, vẫn không đảm bảo được rằng các sản phẩm phần mềm đang được ứng dụng không có lỗi. Lỗi vẫn luôn tiềm ẩn trong mọi sản phẩm phần mềm và cũng có thể gây những thiệt hại khôn lường.

Kiểm thử phần mềm là một quá trình liên tục, xuyên suốt mọi giai đoạn phát triển phần mềm để đảm bảo rằng phần mềm thỏa mãn các yêu cầu thiết kế và các yêu cầu đó đáp ứng các nhu cầu của người dùng. Các kỹ thuật kiểm thử phần mềm đã, đang được nghiên cứu, và việc kiểm thử phần mềm đã trở thành quy trình bắt buộc trong các dự án phát triển phần mềm trên thế giới. Kiểm thử phần mềm là một hoạt động rất tốn kém, mất thời gian, và khó phát hiện được hết lỗi. Vì vậy, việc kiểm thử phần mềm đòi hỏi phải có chiến lược phù hợp, một kế hoạch hợp lý và việc thực hiện được quản lý chặt chẽ.

Ở Việt Nam, trong thời gian qua việc kiểm thử phần mềm bị xem nhẹ, với công cụ lập trình hiện đại, người ta cảm tính cho rằng không kiểm thử cũng không sao, nên chưa có nhiều sự quan tâm, nghiên cứu. Những năm gần đây, một số tổ chức nghiên cứu và phát triển phần mềm đã bắt đầu có những quan tâm hơn đến vấn đề kiểm thử phần mềm. Tuy nhiên, vấn đề kiểm thử phần mềm hầu như vẫn chưa được đầu tư và quan tâm đúng mức. Nước ta đang trong quá trình xây dựng một ngành công nghiệp phần mềm thì không thể xem nhẹ việc kiểm thử phần mềm vì xác suất thất bại sẽ rất cao, hơn nữa, hầu hết các công ty phần mềm có uy tín đều đặt ra yêu cầu nghiêm ngặt là nếu một phần mềm không có tài liệu kiểm thử đi kèm thì sẽ không được chấp nhận.

## **2. Mục tiêu và nhiệm vụ nghiên cứu**

- Luận văn tập trung nghiên cứu, tìm hiểu, đánh giá các nguyên lý, chiến lược và kỹ thuật kiểm thử phần mềm.
- Thiết kế các trường hợp kiểm thử áp dụng cho một vài chương trình cụ thể.

## **3. Đối tượng và phạm vi nghiên cứu**

- Quy trình và bản chất của các kỹ thuật kiểm thử hộp đen và kiểm thử hộp trắng.
- Chiến lược kiểm thử phần mềm.
- Đặc tả thiết kế kiểm thử.

## **4. Phương pháp nghiên cứu**

- Nghiên cứu, tìm hiểu các kỹ thuật, chiến lược kiểm thử phần mềm.
- Sử dụng các phương pháp kiểm thử đã nghiên cứu, thiết kế bộ test cho chương trình cụ thể. Đưa ra tài liệu kế hoạch kiểm thử và đặc tả kiểm thử; xây dựng chương trình thực thi kiểm thử.

## **5. Dự kiến kết quả**

- Thiết kế các trường hợp kiểm thử cho một số chương trình cụ thể.
- Tạo các tài liệu kiểm thử (đặc tả trường hợp kiểm thử và kết quả kiểm thử.)
- Xây dựng chương trình kiểm thử.

## **6. Ý nghĩa khoa học và thực tiễn của Luận văn**

Kết quả nghiên cứu có thể làm tài liệu tham khảo cho các cơ sở đang tiến tới đưa qui trình kiểm thử phần mềm thành một qui trình bắt buộc trong dự án phát triển phần mềm của họ.

## **7. Đặt tên đề tài**

***“Một số kỹ thuật kiểm thử phần mềm.”***

## **8. Bố cục của Luận văn**

Toàn bộ nội dung của Luận văn được chia thành 4 chương như sau:

**Chương 1:** Vấn đề chất lượng phần mềm và kiểm thử phần mềm.

**Chương 2:** Các kỹ thuật kiểm thử phần mềm

**Chương 3:** Chiến lược kiểm thử phần mềm

**Chương 4:** Một số ứng dụng cụ thể (của qui trình kiểm thử)

## CHƯƠNG 1

# VẤN ĐỀ CHẤT LƯỢNG PHẦN MỀM VÀ KIỂM THỬ PHẦN MỀM

### 1.1. Sản phẩm phần mềm và vấn đề kiểm thử phần mềm

#### 1.1.1. Sản phẩm phần mềm là gì?

Phần mềm là một (bộ) chương trình được cài đặt trên máy tính nhằm thực hiện một nhiệm vụ tương đối độc lập nhằm phục vụ cho một ứng dụng cụ thể việc quản lý hoạt động của máy tính hoặc áp dụng máy tính trong các hoạt động kinh tế, quốc phòng, văn hóa, giáo dục, giải trí,...

Việc tạo ra một sản phẩm phần mềm phải trải qua nhiều giai đoạn, người ta gọi là qui trình phát triển phần mềm, bắt đầu từ khi bắt đầu có ý tưởng cho đến khi đưa ra sản phẩm phần mềm thực thi. Khối lượng công việc trong từng giai đoạn của quá trình sản xuất phần mềm cũng thay đổi theo thời gian. Bảng 1.1 minh họa cụ thể hơn về điều này.

**Bảng 1.1 - Tỷ lệ công việc của các giai đoạn phát triển phần mềm**

Giai đoạn	Phân tích yêu cầu	Thiết kế sơ bộ	Thiết kế chi tiết	Lập trình và kiểm thử đơn vị	Tích hợp và kiểm thử tích hợp	Kiểm thử hệ thống
Hai thập kỉ 1960 - 1970	10%			80%	10%	
Thập kỉ 1980	20%		60%		20%	
Thập kỉ 1990	40%	30%		30%		

Theo một tài liệu khác [5], chi phí liên quan từng giai đoạn của vòng đời phần mềm như sau:

#### Các giai đoạn phát triển

Phân tích yêu cầu	3%
Đặc tả	3%
Thiết kế	5%
Lập trình	7%

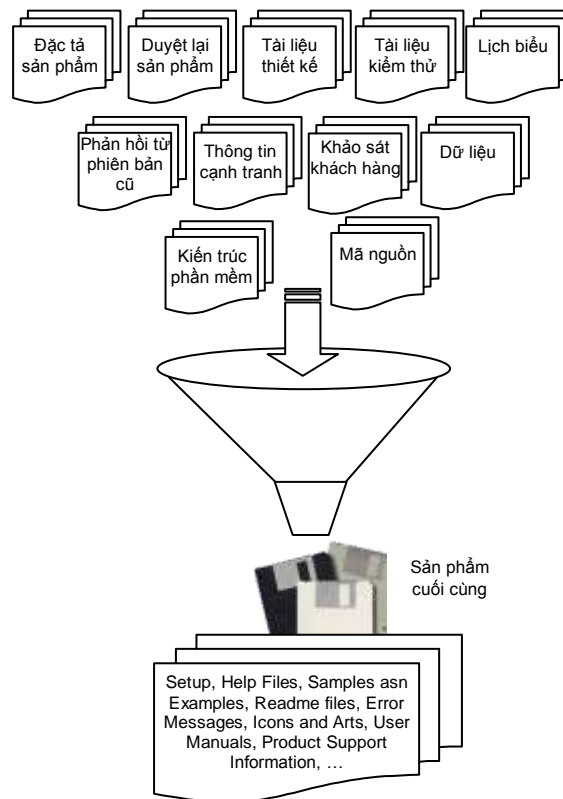
#### Giai đoạn sản phẩm

Vận hành và bảo trì	67%
---------------------	-----

Kiểm thử

15%

Như vậy, một sản phẩm phần mềm không chỉ đơn giản là các đoạn mã chương trình mà còn rất nhiều phần ẩn đằng sau nó (Hình 1.1). Vì vậy, việc mắc lỗi không chỉ xảy ra trong khi lập trình mà còn xảy ra cao hơn trong các công đoạn khác của quy trình phát triển một sản phẩm phần mềm. Việc kiểm thử cũng vì thế phải được tiến hành trong tất cả các phần tạo nên một sản phẩm phần mềm.



**Hình 1.1 – Sản phẩm phần mềm**

### 1.1.2. Thế nào là lỗi phần mềm?

Có rất nhiều định nghĩa khác nhau về lỗi phần mềm, nhưng tựu chung, có thể phát biểu một cách tổng quát: “*Lỗi phần mềm là sự không khớp giữa chương trình và đặc tả của nó.*” [7]

Dựa vào định nghĩa, chúng ta có thể thấy lỗi phần mềm xuất hiện theo ba dạng sau:

- Sai: Sản phẩm được xây dựng khác với đặc tả.

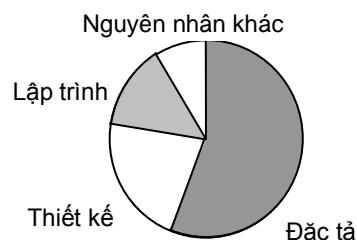


- Thiếu: Một yêu cầu đã được đặc tả nhưng lại không có trong sản phẩm được xây dựng.
- Thừa: Một yêu cầu được đưa vào sản phẩm mà không có trong đặc tả. Cũng có trường hợp yêu cầu này có thể là một thuộc tính sẽ được người dùng chấp nhận nhưng khác với đặc tả nên vẫn coi là có lỗi.

Một hình thức khác nữa cũng được xem là lỗi, đó là phần mềm khó hiểu, khó sử dụng, chậm hoặc dễ gây cảm nhận rằng phần mềm hoạt động không đúng.

### 1.1.3. Tại sao lỗi phần mềm xuất hiện?

Khác với sự cảm nhận thông thường, lỗi xuất hiện nhiều nhất không phải do lập trình. Nhiều nghiên cứu đã được thực hiện trong các dự án từ rất nhỏ đến các dự án rất lớn và kết quả luôn giống nhau. Số lỗi do đặc tả gây ra là nhiều nhất, chiếm khoảng 80%. Có một số nguyên nhân làm cho đặc tả tạo ra nhiều lỗi nhất. Trong nhiều trường hợp, đặc tả không được viết ra. Các nguyên nhân khác có thể do đặc tả không đủ cẩn thận, nó hay thay đổi, hoặc do chưa phối hợp tốt trong toàn nhóm phát triển. Sự thay đổi yêu cầu của khách hàng cũng là nguyên nhân dễ gây ra lỗi phần mềm. Khách hàng thay đổi yêu cầu không cần quan tâm đến những tác động sau khi thay đổi yêu cầu như phải thiết kế lại, lập lại kế hoạch, làm lại những việc đã hoàn thành. Nếu có nhiều sự thay đổi, rất khó nhận biết hết được phần nào của dự án phụ thuộc và phần nào không phụ thuộc vào sự thay đổi. Nếu không giữ được vết thay đổi rất dễ phát sinh ra lỗi.



**Hình 1.2 – Các nguyên nhân gây ra lỗi phần mềm**

Nguồn gây ra lỗi lớn thứ hai là thiết kế. Đó là nền tảng mà lập trình viên dựa vào để nỗ lực thực hiện kế hoạch cho phần mềm.

Lỗi do lập trình gây ra cũng khá dễ hiểu. Ai cũng có thể mắc lỗi khi lập trình. Thời kì đầu, phát triển phần mềm có nghĩa là lập trình, công việc lập trình thì nặng nhọc, do đó lỗi do lập trình gây ra là chủ yếu. Ngày nay, công việc lập trình chỉ là một phần việc của quá trình phát triển phần mềm, cộng với sự hỗ trợ của nhiều công cụ lập trình cao cấp, việc lập trình trở nên nhẹ nhàng hơn, mặc dù độ phức tạp phần mềm lớn hơn rất nhiều. Do đó, lỗi do lập trình gây ra cũng ít hơn. Tuy nhiên, nguyên nhân để lập trình tạo ra lỗi lại nhiều hơn. Đó là do độ phức tạp của phần mềm, do tài liệu nghèo nàn, do sức ép thời gian hoặc chỉ đơn giản là những lỗi “không nói lên được”. Một điều cũng hiển nhiên là nhiều lỗi xuất hiện trên bề mặt lập trình nhưng thực ra lại do lỗi của đặc tả hoặc thiết kế.

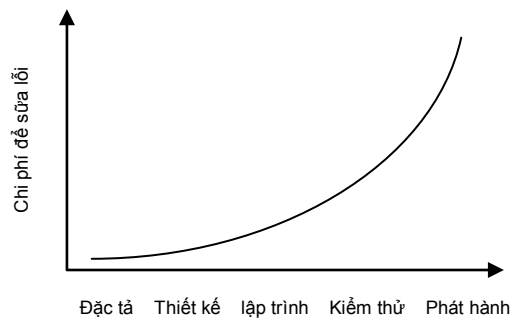
Một nguyên nhân khác tạo ra lỗi là do bản thân các công cụ phát triển phần mềm cũng có lỗi như công cụ trực quan, thư viện lớp, bộ biên dịch,...

#### **1.1.4. Chi phí cho việc sửa lỗi**

Theo tài liệu trích dẫn của Martin và McCabe [7], bảo trì là phần chi phí chính của phần mềm và kiểm thử là hoạt động chi phí đắt thứ hai, ước tính khoảng 40% (15/33) chi phí trong quá trình phát triển ban đầu của sản phẩm phần mềm. Kiểm thử cũng là phần chi phí chính của giai đoạn bảo trì do phải tiến hành kiểm thử lại những thay đổi trong quá trình sửa lỗi và đáp ứng yêu cầu người dùng.

Kiểm thử và sửa lỗi có thể được thực hiện tại bất kỳ giai đoạn nào của vòng đời phần mềm. Tuy nhiên chi phí cho việc tìm và sửa lỗi tăng một cách đáng kể trong quá trình phát triển.

Trong tài liệu Boehm [5], có trích dẫn kết quả nghiên cứu của IBM, GTE và TRW, tổng kết rằng lỗi được phát hiện càng muộn thì chi phí cho việc sửa lỗi càng lớn. Chi phí tăng theo hàm mũ như hình sau.



**Hình 1.3 – Chi phí sửa lỗi theo thời gian phát hiện lỗi**

### 1.1.5. Kiểm thử phần mềm là gì?

Kiểm thử phần mềm thường đồng nghĩa với việc tìm ra lỗi chưa được phát hiện. Tuy nhiên, có nhiều bối cảnh kiểm thử không bộc lộ ra lỗi. Kiểm thử phần mềm là quá trình thực thi một hệ thống phần mềm để xác định xem phần mềm đó có đúng với đặc tả không và thực hiện trong môi trường như mong đợi hay không.

Mục đích của kiểm thử phần mềm là tìm ra lỗi chưa được phát hiện, tìm một cách sớm nhất và đảm bảo rằng lỗi đã được sửa, mà kiểm thử phần mềm không làm công việc chẩn đoán nguyên nhân gây ra lỗi đã được phát hiện và sửa lỗi.

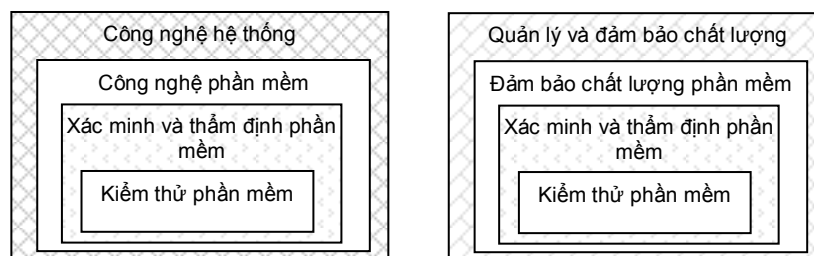
Mục tiêu của kiểm thử phần mềm là thiết kế tài liệu kiểm thử một cách có hệ thống và thực hiện nó sao cho có hiệu quả, nhưng tiết kiệm được thời gian, công sức và chi phí.

## 1.2. Chất lượng phần mềm

Chất lượng phần mềm là một khái niệm đa chiều, không dễ định nghĩa đơn giản theo cách chung cho các sản phẩm là: “*Sản phẩm được phát triển phù hợp với đặc tả của nó.*” [8]. Có một số vấn đề khó trong hệ thống phần mềm, đó là:

- Đặc tả phải định hướng theo những đòi hỏi về chất lượng của khách hàng (như tính hiệu quả, độ tin cậy, tính dễ hiểu, tính bảo mật,...) và những yêu cầu của chính tổ chức phát triển phần mềm vốn không có trong đặc tả (như các yêu cầu về khả năng bảo trì, tính sử dụng lại,...)
- Một số yêu cầu về chất lượng cũng rất khó chỉ ra một cách rõ ràng.

- Những đặc tả phần mềm thường không đầy đủ và hay mâu thuẫn.



(a) Ngữ cảnh quy trình

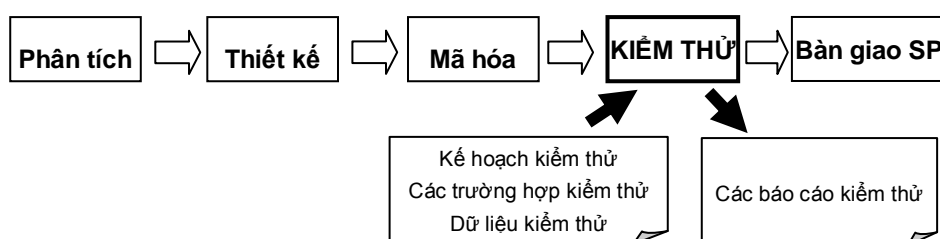
(b) Ngữ cảnh chất lượng

### Hình 1.4 - Kiểm thử phần mềm trong một số ngữ cảnh

Trên quan điểm qui trình, kiểm thử phần mềm là một phần của xác minh và thẩm định phần mềm. Xác minh và thẩm định nằm trong công nghệ phần mềm, công nghệ phần mềm lại là một phần của công nghệ hệ thống (Hình 1.4a). Nhìn từ ngữ cảnh chất lượng (Hình 1.4b), kiểm thử phần mềm cũng là một phần của xác minh và thẩm định phần mềm, nên cũng có thể xem như là một phần của đảm bảo chất lượng phần mềm. Nếu phần mềm là thành phần của hệ thống lớn hơn thì kiểm thử phần mềm cũng được xem như là một phần của quản lý và đảm bảo chất lượng. Và để đạt phần mềm chất lượng cao, thì kiểm thử có thể coi là một thành phần chủ yếu của hoạt động đảm bảo chất lượng phần mềm.

### 1.3. Qui trình kiểm thử phần mềm

Mục đích của kiểm thử là thiết kế một chuỗi các trường hợp kiểm thử mà có khả năng phát hiện lỗi cao. Để cho việc kiểm thử đạt được kết quả tốt cần có sự chuẩn bị về kế hoạch kiểm thử, thiết kế các trường hợp kiểm thử và các dữ liệu kiểm thử cho các trường hợp. Đây chính là đầu vào cho giai đoạn kiểm thử. Và sản phẩm công việc của giai đoạn kiểm thử chính là “báo cáo kiểm thử” mà tài liệu hóa tất cả các trường hợp kiểm thử đã chạy, dữ liệu đầu vào, đầu ra mong đợi, đầu ra thực tế và mục đích của kiểm thử,... (như Hình 1.5)



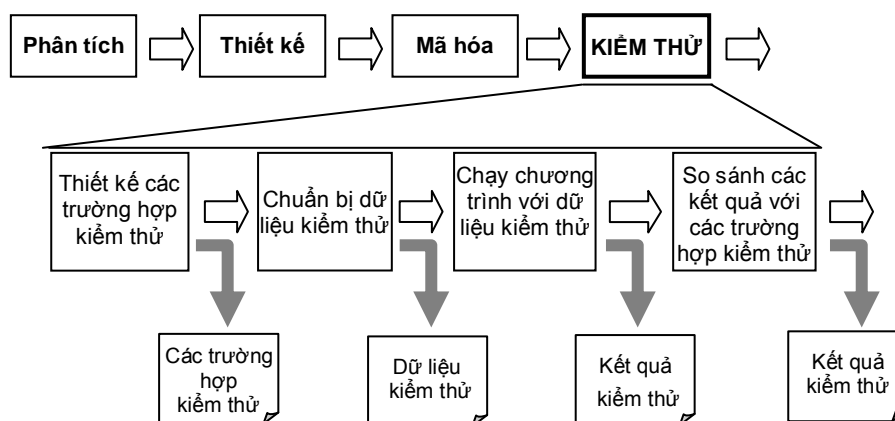
### **Hình 1.5 – Giai đoạn kiểm thử trong xử lý phần mềm**

Quy trình kiểm thử bao gồm một số giai đoạn:

- Lập kế hoạch kiểm thử. Bước đầu tiên là lập kế hoạch cho tất cả các hoạt động sẽ được thực hiện và các phương pháp được sử dụng. Các chuẩn IEEE bao gồm các thông tin về tác giả chuẩn bị kế hoạch, danh sách liệt kê của kế hoạch kiểm thử. Vấn đề quan trọng nhất đối với kế hoạch kiểm thử [6,7]:
  - + Mục đích: Qui định về phạm vi, phương pháp, tài nguyên và lịch biểu của các hoạt động kiểm thử.
  - + Các tài liệu tham khảo.
  - + Các định nghĩa.
  - + Khái quát về xác minh và thẩm định (V&V): tổ chức, tài nguyên, trách nhiệm, các công cụ, kỹ thuật và các phương pháp luận.
  - + Vòng đời của V&V: các nhiệm vụ, các dữ liệu vào và các kết quả ra trên một giai đoạn vòng đời.
  - + Báo cáo xác minh và thẩm định(V&V) phần mềm: mô tả nội dung, định dạng và thời gian cho tất cả các báo cáo V&V.
  - + Các thủ tục quản lý V&V bao gồm các chính sách, thủ tục, các chuẩn, thực nghiệm và các qui ước.
- Giai đoạn bố trí nhân viên kiểm thử. Việc kiểm thử thường phải tiến hành một cách độc lập và các nhóm độc lập có trách nhiệm tiến hành các hoạt động kiểm thử, gọi là các nhóm kiểm thử.
- Thiết kế các trường hợp kiểm thử. Các trường hợp kiểm thử là các đặc tả đầu vào cho kiểm thử và đầu ra mong đợi của hệ thống cùng với các câu lệnh được kiểm thử.
  - + Các phương pháp hộp đen để kiểm thử dựa trên chức năng.

- + Các phương pháp hộp trắng để kiểm thử dựa vào cấu trúc bên trong.
- Xử lý đo lường kiểm thử bằng cách thu thập dữ liệu.
- Đánh giá sản phẩm phần mềm để xác nhận sản phẩm có thể sẵn sàng phát hành được chưa?

Mô hình chung của quy trình kiểm thử phần mềm được thể hiện trong hình 1.6.



**Hình 1.6 – Quy trình kiểm thử phần mềm**

## CHƯƠNG 2

# CÁC KỸ THUẬT KIỂM THỬ PHẦN MỀM

Có thể sử dụng một số kỹ thuật trong quá trình kiểm thử nhằm tăng hiệu quả của hoạt động này. Mc Gregor mô tả các kỹ thuật kiểm thử như những công cụ được thiết kế để đảm bảo rằng tất cả các khía cạnh của sản phẩm đều được khảo sát [1]. Mặt khác, các kỹ thuật kiểm thử là những công cụ để dễ dàng đạt được hiệu quả kiểm thử.

Có thể chia các kỹ thuật kiểm thử phần mềm thành hai loại: các kỹ thuật kiểm thử hộp đen (black-box testing) và kỹ thuật kiểm thử hộp trắng (white-box testing). Kiểm thử sử dụng kỹ thuật hộp đen thường được gọi đơn giản là các kiểm thử black-box. Các kiểm thử black-box tìm các lỗi như thiếu các chức năng, khả năng sử dụng và các yêu cầu phi chức năng.

Các kỹ thuật kiểm thử white-box yêu cầu hiểu biết về cấu trúc chương trình bên trong và các kiểm thử nhận được từ đặc tả thiết kế bên trong hoặc từ mã [2].

### 2.1. Nguyên tắc cơ bản kiểm thử phần mềm

Trong lúc kiểm thử, công nghệ phần mềm phát sinh một chuỗi các trường hợp kiểm thử được sử dụng để “tách từng phần” phần mềm. Kiểm thử là một bước trong quy trình phần mềm mà có thể được xem xét bởi đội ngũ phát triển bằng cách phá vỡ thay vì xây dựng. Các kỹ sư phần mềm chính là những người xây dựng và việc

kiểm thử yêu cầu họ vượt qua các khái niệm cho trước về độ chính xác và giải quyết mâu thuẫn khi các lỗi được xác định.

### 2.1.1. Mục tiêu kiểm thử

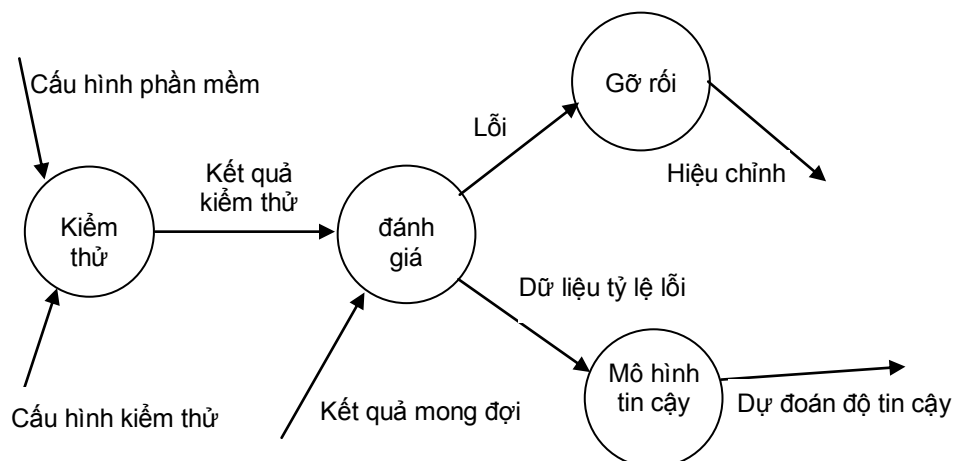
Các nguyên tắc được xem như mục tiêu kiểm thử là:

- Kiểm thử là một quá trình thực thi chương trình với mục đích tìm lỗi.
- Một trường hợp kiểm thử tốt là trường hợp kiểm thử mà có khả năng cao việc tìm thấy các lỗi chưa từng được phát hiện.
- Một kiểm thử thành công là kiểm thử mà phát hiện lỗi chưa từng được phát hiện.

### 2.1.2. Luồng thông tin kiểm thử

Luồng thông tin cho kiểm thử được biểu diễn bởi mô hình trong hình 2.1. Hai kiểu của đầu vào được truyền cho quá trình kiểm thử:

- Cấu hình phần mềm: gồm các đặc tả yêu cầu, đặc tả thiết kế, và mã nguồn.
- Cấu hình kiểm thử: gồm có kế hoạch kiểm thử, các thủ tục, trường hợp kiểm thử, và các công cụ kiểm thử.





## Hình 2.1 - Luồng thông tin kiểm thử

### 2.1.3. Thiết kế trường hợp kiểm thử

Thiết kế kiểm thử phần mềm có thể là một quá trình thu thập, phân tích và thực hiện yêu cầu. Mục tiêu của kiểm thử là phải thiết kế các trường hợp kiểm thử có khả năng cao nhất trong việc phát hiện nhiều lỗi nhất với thời gian và công sức tối thiểu. Như vậy, vấn đề quan trọng nhất trong kiểm thử phần mềm là thiết kế và tạo ra các trường hợp kiểm thử có hiệu quả. Lý do về tầm quan trọng của việc thiết kế các trường hợp kiểm thử xuất phát từ thực tế: Kiểm thử “vét cạn” là điều không thể, và như vậy, kiểm thử một chương trình phải luôn xác định là không thể vét cạn. Vấn đề quan trọng là cố gắng làm giảm sự “không thể vét cạn” nhiều nhất có thể.

Kiểm thử phần mềm còn có các ràng buộc về thời gian, chi phí, ... Chìa khoá của kiểm thử là trả lời của câu hỏi: *“Tập con của tất cả các trường hợp kiểm thử có thể có xác suất phát hiện lỗi cao nhất là gì?”*. Việc nghiên cứu các phương pháp thiết kế trường hợp kiểm thử sẽ cung cấp câu trả lời cho câu hỏi này.

Bất kỳ sản phẩm công nghệ nào có thể được kiểm thử trong hai cách:

- Biết về các chức năng cụ thể mà sản phẩm đã được thiết kế để thực hiện.
- Biết cách hoạt động bên trong của sản phẩm, kiểm thử có thể được thực hiện để đảm bảo rằng “tất cả các thành phần ăn khớp nhau”.

Cách tiếp cận kiểm thử đầu tiên được gọi là kiểm thử hộp đen và cách thứ hai là kiểm thử hộp trắng.

### 2.2. Kỹ thuật kiểm thử hộp trắng (White-Box Testing)

Kiểm thử hộp trắng hay còn gọi là kiểm thử hướng logic, cho phép kiểm tra cấu trúc bên trong của phần mềm với mục đích đảm bảo rằng tất cả các câu lệnh và điều kiện sẽ được thực hiện ít nhất một lần.

Hộp trắng đúng nghĩa phải gọi là hộp trong suốt. Chính vì vậy, kỹ thuật này còn có một số tên khác là kiểm thử hộp thủy tinh (Glass-Box Testing), hay kiểm thử

hộp trong suốt (Clear-Box Testing). Người kiểm thử truy nhập vào mã nguồn chương trình và có thể kiểm tra nó, lấy đó làm cơ sở để hỗ trợ việc kiểm thử.

Tương tự như vấn đề kiểm thử đầu vào trong kỹ thuật kiểm thử hộp đen, đó là vấn đề kiểm thử các đường dẫn lệnh trong kỹ thuật hộp trắng. Nếu phải thực hiện tất cả các đường dẫn của lưu trình điều khiển trong chương trình thông qua việc chạy tất cả các trường hợp kiểm thử thì có thể nói rằng chương trình đã được kiểm thử triệt để. Tuy nhiên điều đó không thể thực hiện được vì số đường dẫn logic khác nhau trong một chương trình là cực kỳ lớn. Ví dụ trong hình 2.2, sơ đồ khối của một chu trình điều khiển. Sơ đồ biểu diễn một vòng lặp từ 1 đến 20 lần. Trong thân của vòng lặp có một tập các lệnh điều kiện rẽ nhánh lồng nhau. Số đường dẫn trong vòng lặp là 5. Như vậy, tổng số đường dẫn có thể là  $(5 + 5^2 + 5^3 + \dots + 5^{20})$  khoảng xấp xỉ  $10^{14}$ .

Ngoài những điều bất khả thi như trên, chương trình cũng có thể còn nhiều lỗi do các nguyên nhân khác. Đây chính là nhược điểm của kỹ thuật kiểm thử hộp trắng.

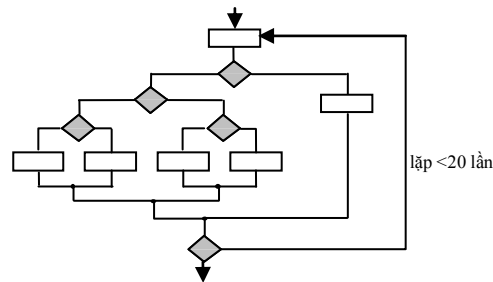
- Việc kiểm thử bằng kỹ thuật hộp trắng không thể đảm bảo rằng chương trình đã tuân theo đặc tả.
- Một chương trình sai do thiếu đường dẫn. Việc kiểm thử hộp trắng không thể biết được sự thiếu sót này.
- Việc kiểm thử bằng kỹ thuật hộp trắng không thể phát hiện được lỗi do dữ liệu.

Như vậy, việc kiểm thử chỉ dùng kỹ thuật hộp trắng là không đủ để phát hiện lỗi. Vì vậy, khi thiết kế các trường hợp kiểm thử thường cần phải kết hợp với cả kỹ thuật kiểm thử hộp đen.

Nguyên tắc của kỹ thuật kiểm thử hộp trắng là:

- Thực hiện mọi đường dẫn độc lập ít nhất một lần.

- Thực hiện mọi điều kiện logic (if-then-else) trên các giá trị true và false của chúng.
- Thực hiện mọi vòng lặp (các vòng lặp for, while-do) tại các biên và trong phạm vi hoạt động của chúng.
- Thực hiện mọi cấu trúc dữ liệu bên trong để đảm bảo tính hợp lệ của chúng.



**Hình 2.2- Ví dụ chu trình điều khiển**

### 2.2.1. Kiểm thử đường dẫn cơ sở (Basic Path Testing)

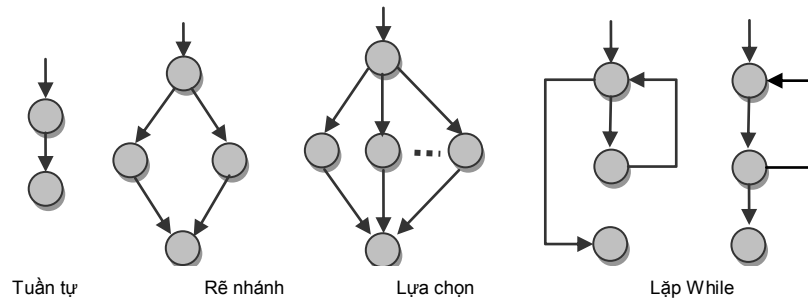
Kiểm thử đường dẫn cơ sở là một kỹ thuật kiểm thử hộp trắng do Tom McCabe đề xuất. Phương pháp đường dẫn cơ sở cho phép người thiết kế trường hợp kiểm thử thực hiện phép đo độ phức tạp logic của thiết kế thủ tục và sử dụng phép đo này như một chỉ dẫn cho việc thiết kế một tập cơ sở các đường dẫn thực hiện. Những trường hợp kiểm thử được suy diễn để thực hiện tập cơ sở. Các trường hợp kiểm thử đó được đảm bảo để thực hiện mỗi lệnh trong chương trình ít nhất một lần trong quá trình kiểm thử.

#### 2.2.1.1. Đồ thị lưu trình (Flow Graph)

Trong thực tế, phương pháp đường dẫn cơ sở có thể được dùng không cần sử dụng đồ thị lưu trình. Tuy nhiên, đồ thị lưu trình là một công cụ hữu ích để hiểu lưu trình điều khiển và minh họa phương pháp tiếp cận. Phần này sẽ trình bày một số ký hiệu đơn giản của lưu trình điều khiển, được gọi là đồ thị lưu trình. Đồ thị lưu trình vẽ lưu trình điều khiển logic sử dụng một số ký hiệu được minh họa như hình 2.3. Mỗi cấu trúc điều khiển có một ký hiệu đồ thị lưu trình tương ứng.

Đồ thị lưu trình gồm có:

- Mỗi hình tròn gọi là *đỉnh*, biểu diễn một hoặc nhiều lệnh thủ tục.
- Con trỏ được gọi là *cung* hoặc *liên kết* biểu diễn lưu trình điều khiển. Một cung cần phải kết thúc tại một đỉnh.
- Mỗi đỉnh có chứa điều kiện gọi là *đỉnh điều kiện*.
- Phần được bao bởi các cung và các đỉnh gọi là *vùng*.



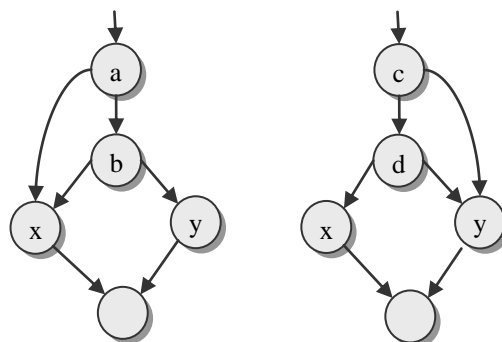
**Hình 2.3- Ký hiệu đồ thị lưu trình**

- ***Biểu diễn các điều kiện phức trong đồ thị lưu trình***

Khi gặp các điều kiện phức xuất hiện trong câu lệnh điều kiện được biểu diễn gồm một hoặc nhiều phép toán logic (AND, OR, NOT), cần phải được chia thành các điều kiện đơn trong thực hiện kiểm thử đường dẫn cơ sở. Mỗi đỉnh chứa điều kiện được gọi là đỉnh điều kiện và được đặc trưng bởi hai hoặc nhiều cạnh bắt nguồn từ nó.

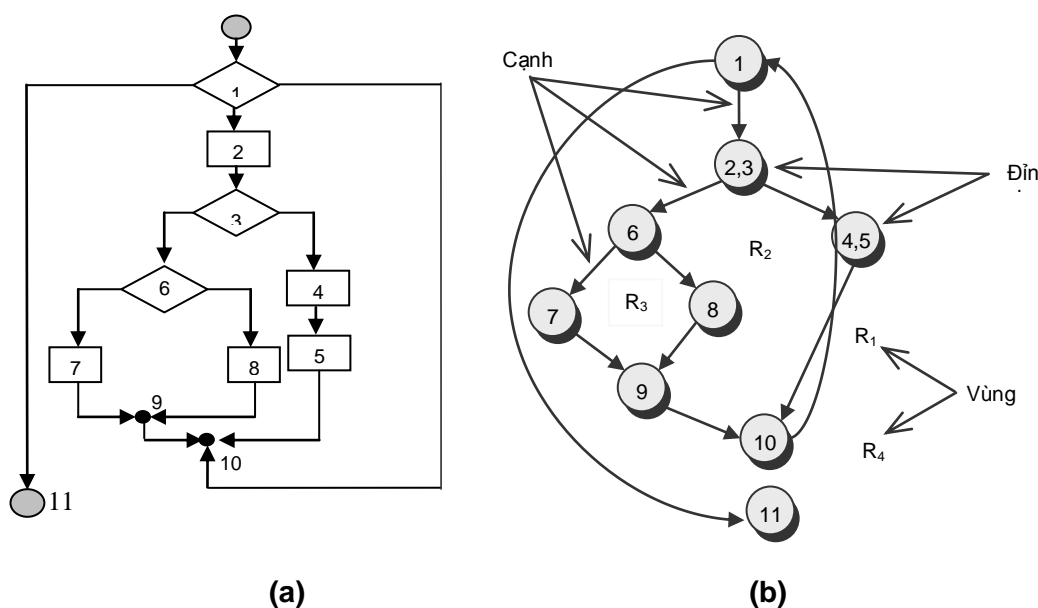
Ví dụ: if (a OR b) then {Procedure x } else {Procedure y }

if (c AND d) then {Procedure x } else {Procedure y }



**Hình 2.4 - Điều kiện phức**

**Ví dụ:** Cho lưu đồ thuật toán như hình 2.5a, đồ thị lưu trình có thể xác định như hình 2.5b..



**Hình 2.5 – Lưu đồ thuật toán (a) và đồ thị lưu trình (b)**

#### 2.2.1.2. Độ phức tạp cyclomat

Độ phức tạp cyclomat là một thước đo phần mềm, cung cấp phép đo định lượng độ phức tạp của chương trình. Khi được sử dụng trong ngữ cảnh của phương pháp đường dẫn cơ sở, giá trị được xác định cho độ phức tạp cyclomat cho biết số lượng đường dẫn độc lập trong một tập cơ sở của chương trình và cung cấp cho chúng ta một giới hạn trên số lượng kiểm thử bắt buộc để đảm bảo rằng tất cả các câu lệnh được thực hiện ít nhất một lần.

Một đường dẫn độc lập là một đường dẫn bất kỳ trong chương trình đưa ra ít nhất một tập lệnh xử lý hoặc điều kiện mới.

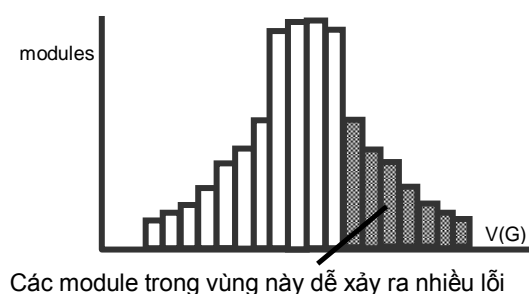
Tập đường dẫn độc lập cho đồ thị lưu trình được minh họa trong hình 2.5b là:

- Đường dẫn 1: 1-11
- Đường dẫn 2: 1-2-3-4-5-10-1-11
- Đường dẫn 3: 1-2-3-6-8-9-10-1-11
- Đường dẫn 4: 1-2-3-6-7-9-10-1-11

Mỗi đường dẫn mới đưa ra một cung mới. Đường dẫn 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 không được xem là một đường dẫn độc lập vì nó chỉ là một tổ hợp các đường dẫn đã được chỉ ra (đường dẫn 2 và 3) và nó sẽ không đi qua một cung mới nào.

Các đường dẫn 1, 2, 3 và 4 tạo thành một *tập cơ sở* trong hình 2.5b. Nếu các trường hợp kiểm thử được thiết kế để thực hiện những đường dẫn này (một tập cơ sở) thì mọi lệnh trong chương trình sẽ đảm bảo được thực hiện ít nhất một lần và mọi điều kiện sẽ được thực hiện cả hai mặt đúng (true) và mặt sai (false). Tuy nhiên, tập cơ sở không phải là duy nhất. Trong thực tế, một số các tập cơ sở khác nhau có thể được suy diễn cho việc thiết kế một thủ tục được đưa ra.

Một số nghiên cứu công nghiệp đã chỉ rằng độ phức tạp Cyclomat càng cao, xác suất hoặc lỗi càng cao.



**Hình 2.6 - Độ phức tạp Cyclomat**

Việc tính toán độ phức tạp cyclomat sẽ cho chúng ta biết có bao nhiêu đường dẫn cần tìm. Cho đồ thị lưu trình  $G$ , độ phức tạp Cyclomat  $V(G)$  được tính theo một trong 3 công thức sau (dựa trên Lý thuyết đồ thị):

1.  $V(G) = R$ , trong đó  $R$  là số vùng của đồ thị lưu trình.
2.  $V(G) = P + 1$ , trong đó  $P$  là số đỉnh điều kiện có trong đồ thị lưu trình  $G$ .
3.  $V(G) = E - N + 2$ , trong đó  $E$  là số cạnh của đồ thị lưu trình,  $N$  là số đỉnh của đồ thị lưu trình  $G$ .

Đối chiếu với đồ thị lưu trình trong hình 2.5b, độ phức tạp Cyclomat được tính như sau:

1. Công thức 1:  $V(G) = R = 4$
2. Công thức 2:  $V(G) = P + 1 = 3 + 1 = 4$
3. Công thức 3:  $V(G) = E - N + 2 = 11 - 9 + 2 = 4$

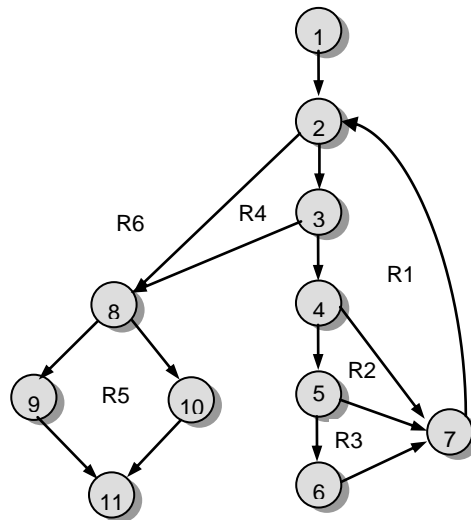
Như vậy, độ phức tạp Cyclomat của đồ thị trong hình 2.4b là 4.

#### 2.2.1.3. Phát sinh các trường hợp kiểm thử theo đường dẫn cơ sở

Phương pháp kiểm thử đường dẫn cơ sở có thể được áp dụng để thiết kế thủ tục chi tiết hoặc cho mã nguồn. Kiểm thử đường dẫn cơ sở có thể được xem như một tập các bước.

- Bước 1:** Sử dụng thiết kế hoặc mã nguồn như là căn cứ để vẽ đồ thị lưu trình tương ứng.
- Bước 2:** Xác định độ phức tạp Cyclomat của đồ thị lưu trình kết quả.
- Bước 3:** Xác định tập cơ sở các đường dẫn độc lập tuyến tính.
- Bước 4:** Chuẩn bị các trường hợp kiểm thử có khả năng thực hiện mỗi đường dẫn trong tập cơ sở.

Chúng ta sẽ dùng hàm tính trung bình cộng của các số, *average* trong C như hình 2.7 để làm ví dụ minh họa cho mỗi bước thiết kế các trường hợp kiểm thử. Hàm *average* là một thuật toán đơn giản có chứa các điều kiện tổ hợp và vòng lặp, trong đó chương trình tính giá trị trung bình của 100 hoặc một vài số trong mảng *values* nằm trong khoảng của biên trên (*min*) và biên dưới (*max*). Đầu vào được kết thúc bằng giá trị -999.



**Hình 2.7 – Ví dụ minh họa phát sinh các trường hợp kiểm thử theo đường dẫn cơ sở**

**Bước 1:** Vẽ đồ thị lưu trình (như hình 2.7)

**Bước 2:** Xác định độ phức tạp cyclomat

$$V(G) = R \text{ (số vùng)} = 6$$

$$V(G) = P \text{ (số đỉnh điều kiện)} + 1 = 5 + 1 = 6$$

$$V(G) = E \text{ (số cạnh)} - N \text{ (số đỉnh)} + 2 = 17 - 13 + 2 = 6$$

**Bước 3:** Tìm tập cơ sở các đường dẫn độc lập.

- + Đường dẫn 1:  $1 \Rightarrow 2 \Rightarrow 8 \Rightarrow 9 \Rightarrow 11$
- + Đường dẫn 2:  $1 \rightarrow 2 \rightarrow 8 \Rightarrow 10 \Rightarrow 11$
- + Đường dẫn 3:  $1 \rightarrow 2 \Rightarrow 3 \Rightarrow 8 \rightarrow 9 \rightarrow 11$
- + Đường dẫn 4:  $1 \rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 7 \Rightarrow 2 \rightarrow \dots$
- + Đường dẫn 5:  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \Rightarrow 5 \Rightarrow 7 \rightarrow 2 \rightarrow \dots$
- + Đường dẫn 6:  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \Rightarrow 6 \Rightarrow 7 \rightarrow 2 \rightarrow \dots$

Trong hình 2.6, các đỉnh 2, 3, 4, 5, 8 là các đỉnh điều kiện.

**Bước 4:** Thiết kế các trường hợp kiểm thử cho mỗi đường dẫn độc lập trong tập cơ sở đã chọn.



- ***Trường hợp kiểm thử đường dẫn 1***

Đầu vào: values = {3, 5, 11, -999}, min = 0, max = 100

Đầu ra mong muốn: average = (3 + 5 + 11)/3

Mục đích: để kiểm thử việc tính trung bình chính xác.

Chú ý: Đường dẫn 1 không thể kiểm thử một mình, mà phải được kiểm thử như là một phần của các kiểm thử đường dẫn 4, 5, và 6.

- ***Trường hợp kiểm thử đường dẫn 2***

Đầu vào: values = {-999}, min = 0, max = 0

Đầu ra mong muốn: average = -999

Mục đích: để tạo ra average = -999

- ***Trường hợp kiểm thử đường dẫn 3***

Đầu vào: values = {3, 5, 30, ..., 76} (101 số), min = 0, max = 100

Đầu ra mong muốn: trung bình của 100 số đầu tiên.

Mục đích: chỉ tính trung bình cho 100 số hợp lệ đầu tiên .

- ***Trường hợp kiểm thử đường dẫn 4***

Đầu vào: values = {67, -2, 12, 23, -999}, min = 0, max = 100

Đầu ra mong muốn: (67 + 12 + 23)/3

Mục đích: Kiểm thử biên dưới (values[i] < min, i < 100).

- ***Trường hợp kiểm thử đường dẫn 5***

Đầu vào: values = {7, 32, 102, 23, 86, 2, -999}, min = 0, max = 100

Đầu ra mong muốn: (7 + 32 + 23 + 86 + 2)/5

Mục đích: Kiểm thử biên trên (values[i] > max, i < 100).

- ***Trường hợp kiểm thử đường dẫn 6***

Đầu vào: values = {7, 32, 99, 23, 86, 2, -999}, min = 0, max = 100

Đầu ra mong muốn:  $(7 + 32 + 99 + 23 + 86 + 2)/6$

Mục đích: Việc tính trung bình là đúng.

Phương pháp đường dẫn cơ sở tập trung trên “giá trị đại diện” của mỗi đường dẫn độc lập. Cần các trường hợp kiểm thử bổ sung (trên các trường hợp kiểm thử đường dẫn cơ sở), nhất là để thực hiện các điều kiện biên.

### 2.2.2. Kiểm thử cấu trúc điều khiển

Phương pháp kiểm thử đường dẫn cơ sở là phương pháp đơn giản và hiệu quả, nhưng nó vẫn chưa đủ. Chúng ta sẽ xem xét các biến thể trên kiểm thử cấu trúc điều khiển mà phủ kiểm thử mở rộng và hoàn thiện chất lượng của kỹ thuật kiểm thử hộp trắng.

#### 2.2.2.1. Kiểm thử điều kiện

Kiểm thử điều kiện là phương pháp thiết kế trường hợp kiểm thử thực thi các điều kiện logic trong module chương trình.

Một số định nghĩa:

- Điều kiện đơn*: là một biến logic hoặc một biểu thức quan hệ, có thể có toán tử NOT (!) đứng trước, ví dụ, NOT ( $a > b$ )
- Biểu thức quan hệ*: là một biểu thức có dạng  $E1 <op> E2$ , trong đó  $E1$ ,  $E2$  là các biểu thức số học và  $<op>$  là toán tử quan hệ có thể là một trong các dạng sau:  $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $=$ ,  $!=$ , ví dụ,  $a > b + 1$ .
- Điều kiện phức*: gồm hai hay nhiều điều kiện đơn, toán tử logic AND (&&) hoặc OR (||) hoặc NOT (!) và các dấu ngoặc đơn ‘(’ và ‘)’, ví dụ,  $(a > b + 1)$  AND  $(a <= max)$ .

Vì vậy, các thành phần trong một điều kiện có thể gồm phép toán logic, biến logic, cặp dấu ngoặc logic (bao một điều kiện đơn hoặc phức), phép toán quan hệ, hoặc biểu thức toán học.

Mục đích của kiểm thử điều kiện là để xác định không chỉ các lỗi điều kiện mà cả các lỗi khác trong chương trình. Có một số phương pháp kiểm thử điều kiện được đề xuất:

- *Kiểm thử nhánh (Branch Testing)*: là phương pháp kiểm thử điều kiện đơn giản nhất.
- *Kiểm thử miền (Domain Testing)*: cần 3 hoặc 4 kiểm thử cho biểu thức quan hệ. Với một biểu thức quan hệ có dạng  $E1 <op> E2$ , cần có 3 kiểm thử được thiết kế cho  $E1 = E2$ ,  $E1 > E2$ ,  $E1 < E2$ .
- *Kiểm thử nhánh và toán tử quan hệ (Branch and Relational Operator – BRO)*:

#### 2.2.2.2. Kiểm thử luồng dữ liệu

Phương pháp *kiểm thử luồng dữ liệu* lựa chọn các đường dẫn kiểm thử của chương trình dựa vào vị trí khai báo và sử dụng các biến trong chương trình. Với kiểm thử luồng dữ liệu mỗi câu lệnh trong chương trình được gán số hiệu lệnh duy nhất và mỗi hàm không thay đổi tham số của nó và biến toàn cục. Cho một lệnh với  $S$  là số hiệu câu lệnh. Ta định nghĩa,

$DEF(S)$  = là tập các biến được khai báo trong  $S$ .

$USE(S)$  = là tập các biến được sử dụng trong  $S$ .

Một chiến lược kiểm thử luồng dữ liệu cơ bản là chiến lược mà mỗi chuỗi DU được phủ ít nhất một lần. Chiến lược này được gọi là chiến lược kiểm thử DU. Kiểm thử DU không đảm bảo phủ hết tất cả các nhánh của một chương trình. Tuy nhiên, một nhánh không đảm bảo được phủ bởi kiểm thử DU chỉ trong rất ít tình huống như cấu trúc *if-then-else* mà trong đó phần *then* không có một khai báo biến nào và có dạng khuyết (không tồn tại phần *else*). Trong tình huống đó, nhánh *else* của lệnh *if* là không cần thiết phải phủ bằng kiểm thử DU.

Chiến lược kiểm thử luồng dữ liệu là rất hữu ích cho việc lựa chọn các đường dẫn kiểm thử của chương trình có chứa các lệnh *if* hoặc vòng lặp lồng nhau.

### 2.2.2.3. Kiểm thử vòng lặp

Vòng lặp là nền tảng cho hầu hết các thuật toán được cài đặt trong phần mềm. Tuy nhiên, chúng ta thường ít quan tâm đến nó khi thực hiện việc kiểm thử phần mềm. Kiểm thử vòng lặp là một kỹ thuật kiểm thử hộp trắng mà tập trung trên tính hợp lệ của các cấu trúc lặp. Việc xây dựng các trường hợp kiểm thử cho mỗi loại cần thực hiện như sau:

#### Vòng lặp đơn

Với vòng lặp đơn trong đó  $N$  là số lần lặp tối đa, các trường hợp kiểm thử sau được sử dụng để kiểm tra mỗi điều kiện sau:

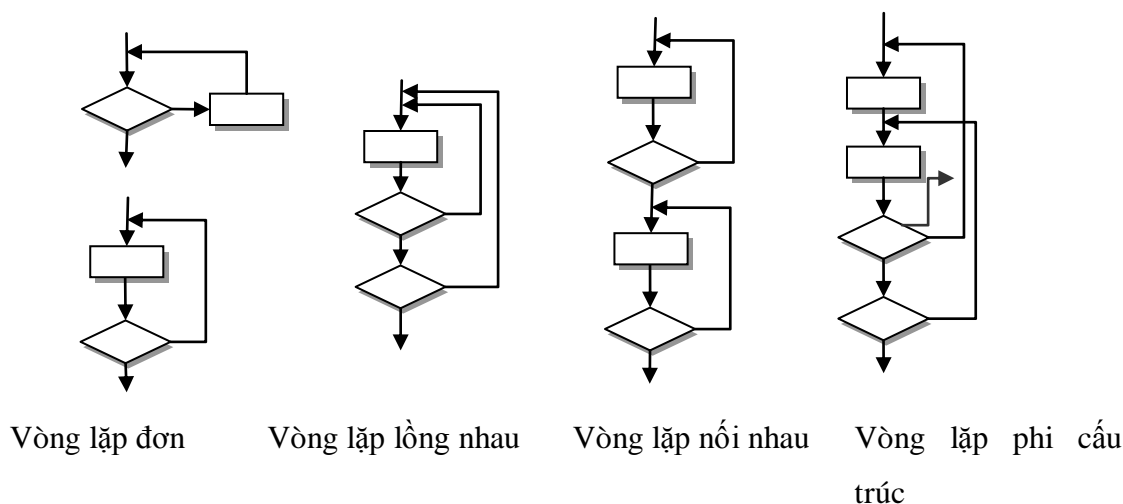
- Bỏ qua vòng lặp
- Chỉ một lần lặp
- Hai lần lặp
- $M$  lần lặp trong đó  $M < N$
- $N-1, N, N+1$  lần lặp.

#### Vòng lặp lồng nhau

Nếu chúng ta mở rộng phương pháp kiểm thử vòng lặp đơn cho vòng lặp lồng nhau thì các kiểm thử có thể sẽ tăng theo mức phát triển vòng lặp. Điều này có thể tạo ra một số không thực tế các trường hợp kiểm thử. Chính vì vậy, một cách tiếp cận đệ quy như sau sẽ giảm bớt số trường hợp kiểm thử.

- Bắt đầu tại vòng lặp trong cùng.
- Xây dựng các kiểm thử vòng lặp đơn cho vòng lặp trong cùng, trong khi đó giữ vòng lặp ngoài cùng tại các giá trị tham số lặp nhỏ nhất của chúng.
- Phát triển ra phía ngoài, xây dựng các kiểm thử cho vòng lặp tiếp theo, nhưng giữ tất cả các vòng lặp bên ngoài với giá trị nhỏ nhất và các vòng lặp lồng nhau khác giá trị “đặc biệt”.

Tiếp tục cho đến khi tất cả các vòng lặp được kiểm thử.



**Hình 2.8 – Các kiểu vòng lặp**

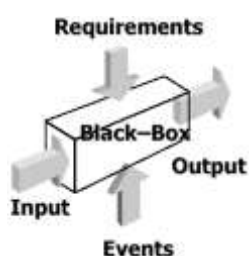
### Vòng lặp nối nhau

Nếu các vòng lặp nối nhau là độc lập thì chúng có thể được xem như hai vòng lặp đơn riêng biệt, sử dụng phương pháp kiểm thử vòng lặp đơn. Nếu vòng lặp thứ hai phụ thuộc vào vòng lặp trước (ví dụ, biến đếm của vòng lặp 1 là giá trị khởi tạo của vòng lặp 2), thì xem chúng như các vòng lặp lồng nhau và sử dụng cách tiếp cận kiểm thử vòng lặp lồng nhau.

### Vòng lặp phi cấu trúc

Nếu gặp các lớp vòng lặp này chúng ta sẽ không kiểm thử mà sẽ thiết kế lại tương ứng với sử dụng việc xây dựng chương trình có cấu trúc.

## 2.3. Kỹ thuật kiểm thử hộp đen (Black-Box Testing)



Kỹ thuật kiểm thử hộp đen còn được gọi là kiểm thử hướng dữ liệu (data-driven) hay là kiểm thử hướng vào/ra (input/output driven).

Trong kỹ thuật này, người kiểm thử xem phần mềm như là một hộp đen. Người kiểm thử hoàn toàn không quan tâm cấu trúc và hành vi bên trong của phần mềm. Người kiểm thử chỉ cần quan tâm đến việc tìm các hiện tượng mà phần mềm không hành xử theo đúng đặc tả của nó. Và vì thế, dữ liệu kiểm thử sẽ xuất phát từ đặc tả.

Như vậy, cách tiếp cận kiểm thử hộp đen tập trung vào các yêu cầu chức năng của phần mềm. Kiểm thử hộp đen cho phép các kỹ sư kiểm thử xây dựng các nhóm giá trị đầu vào mà sẽ thực thi đầy đủ tất cả các yêu cầu chức năng của chương trình. Kiểm thử hộp đen không thay thế kỹ thuật hộp trắng, nhưng nó bổ sung khả năng phát hiện các lớp lỗi khác với các phương pháp hộp trắng.

Kiểm thử hộp đen cố gắng tìm các loại lỗi sau:

- Các chức năng thiếu hoặc không đúng.
- Các lỗi giao diện.
- Các lỗi cấu trúc dữ liệu trong truy cập cơ sở dữ liệu bên ngoài.
- Các lỗi thi hành.
- Các lỗi khởi tạo hoặc kết thúc.
- Và các lỗi khác...

Không giống kiểm thử hộp trắng được thực hiện sớm trong quá trình kiểm thử, kiểm thử hộp đen nhằm đến áp dụng trong các giai đoạn sau của kiểm thử. Vì kiểm thử hộp đen không để ý có chủ đích cấu trúc điều khiển, sự quan tâm tập trung trên miền thông tin. Nếu người ta mong muốn sử dụng phương pháp này để tìm tất cả các lỗi trong chương trình thì điều kiện bắt buộc là phải kiểm thử tất cả các đầu vào, tức là mỗi một điều kiện đầu vào có thể có là một trường hợp kiểm thử. Bởi vì nếu chỉ kiểm thử một số điều kiện đầu vào thì không đảm bảo được chương trình đã hết lỗi. Tuy nhiên, điều này thực tế không thể thực hiện được.

### **2.3.1. Phân hoạch tương đương**

Như đã trình bày, việc kiểm thử tất cả các đầu vào của chương trình là không thể. Vì thế, khi kiểm thử chương trình nên giới hạn một tập con tất cả các trường hợp đầu vào có thể có.

Một tập con như vậy cần có hai tính chất:

- Mỗi trường hợp kiểm thử nên gồm nhiều điều kiện đầu vào khác nhau có thể để giảm thiểu tổng số các trường hợp cần thiết.
- Nên cố gắng phân hoạch các miền đầu vào của một chương trình thành một số xác định các lớp tương đương, sao cho có thể giả định hợp lý rằng việc kiểm thử một giá trị đại diện của mỗi lớp là tương đương với việc kiểm thử một giá trị bất kỳ trong cùng lớp.

Hai vấn đề xem xét ở trên tạo thành một phương pháp của kỹ thuật hộp đen và gọi là phân hoạch tương đương. Vấn đề thứ hai được sử dụng để phát triển một tập các điều kiện cần quan tâm phải được kiểm thử. Vấn đề thứ nhất được sử dụng để phát triển một tập cực tiểu các trường hợp kiểm thử phủ các điều kiện trên.

Thiết kế trường hợp kiểm thử bằng phân hoạch tương đương được xử lý theo hai bước: phân hoạch các miền đầu vào/ra thành các lớp tương đương, và thiết kế các trường hợp kiểm thử đại diện cho mỗi lớp.

#### 2.3.1.1. Xác định các lớp tương đương

“Phân hoạch tương đương” được định nghĩa theo lý thuyết tập hợp.

- Quan hệ  $\rho$  trên hai tập  $A$  và  $B$  là một tập con của tích Descartes  $A \times B$ , nghĩa là  $apb$  trong đó  $a \in A$  và  $b \in B$ .
- Quan hệ có thể được định nghĩa trên chính tập  $A$ , tức là khi  $B = A$ .
- Quan hệ  $\rho$  trên tập  $A$  gọi là phản xạ nếu  $apa$  với  $\forall a \in A$
- Quan hệ  $\rho$  trên tập  $A$  gọi là đối xứng nếu  $apb \Rightarrow bpa$  với  $\forall a, b \in A$
- Quan hệ  $\rho$  trên tập  $A$  gọi là bắc cầu nếu  $apb$  và  $bpc \Rightarrow apc$  với  $\forall a, b, c \in A$
- Một quan hệ có tính phản xạ, đối xứng và bắc cầu gọi là quan hệ tương đương.
- Một quan hệ tương đương phân hoạch tập hợp thành các lớp tương đương rời rạc.

Như vậy, các lớp tương đương được nhận dạng bằng cách lấy mỗi điều kiện đầu vào (thông thường là một câu lệnh hoặc một cụm từ trong đặc tả) và phân hoạch nó thành hai hoặc nhiều nhóm. Các lớp tương đương biểu diễn một tập các trạng thái hợp lệ hoặc không hợp lệ cho điều kiện đầu vào. Điều kiện đầu vào là giá trị số xác định, hoặc miền giá trị, tập giá trị có liên quan, hoặc điều kiện logic. Để làm điều này, chúng ta sử dụng bảng liệt kê các lớp tương đương.

**Bảng 2.1 - Bảng liệt kê các lớp tương đương**

Điều kiện vào/ra	Các lớp tương đương hợp lệ	Các lớp tương đương không hợp lệ

Các lớp tương đương có thể được định nghĩa theo các nguyên tắc sau:

1. Nếu điều kiện đầu vào xác định một *khoảng giá trị*  $[a,b]$ , thì phân hoạch thành một lớp tương đương hợp lệ và một lớp tương đương không hợp lệ. Chẳng hạn, nếu đầu vào  $x$  nằm trong khoảng  $[0,100]$ , lớp hợp lệ là  $0 \leq x \leq 100$ , các lớp không hợp lệ là  $x < 0$  và  $x > 100$ .
2. Nếu điều kiện đầu vào yêu cầu một *giá trị xác định*, phân hoạch thành một lớp tương đương hợp lệ và hai lớp tương đương không hợp lệ. Chẳng hạn, nếu đầu vào  $x=5$ , thì lớp hợp lệ là  $x=5$ , các lớp không hợp lệ là  $x < 5$  và  $x > 5$ .
3. Nếu điều kiện đầu vào xác định một *phần tử của tập hợp*, thì phân hoạch thành một lớp tương đương hợp lệ và một lớp tương đương không hợp lệ.
4. Nếu điều kiện đầu vào là *Boolean*, thì phân hoạch thành một lớp tương đương hợp lệ và một lớp tương đương không hợp lệ tương ứng với hai trạng thái true và false.

Ngoài ra, một nguyên tắc thứ năm được bổ sung là sử dụng khả năng phán đoán, kinh nghiệm và trực giác của người kiểm thử.



### 2.3.1.2. Xác định các trường hợp kiểm thử

Bước thứ hai trong phương pháp phân hoạch tương đương là thiết kế các trường hợp kiểm thử dựa trên sự ước lượng của các lớp tương đương cho miền đầu vào. Tiến trình này được thực hiện như sau:

1. Gán một giá trị duy nhất cho mỗi lớp tương đương.
2. Đến khi tất cả các lớp tương đương hợp lệ được phủ bởi các trường hợp kiểm thử thì viết một trường hợp kiểm thử mới **phủ nhiều nhất có thể** các lớp tương đương hợp lệ chưa được phủ.
3. Đến khi tất cả các lớp tương đương không hợp lệ được phủ bởi các trường hợp kiểm thử thì hãy viết các trường hợp kiểm thử mới sao cho mỗi trường hợp kiểm thử mới **chỉ phủ duy nhất** một lớp tương đương không hợp lệ chưa được phủ.

**Bảng 2.2 – Ví dụ các lớp tương đương**

Điều kiện đầu vào	Các lớp tương đương hợp lệ	Các lớp tương đương không hợp lệ
Số ID của sinh viên	Các ký số	Không phải ký số
Tên sinh viên	Ký tự chữ cái Không rỗng	Không phải chữ cái Rỗng
Giới tính sinh viên	Ký tự chữ cái, “M” hoặc “F”	Không phải chữ cái Không phải “M” hoặc “F”
Điểm của sinh viên	Số Từ 0 đến 100	Không phải số Số nhỏ hơn 0 Số lớn hơn 100

### 2.3.2. Phân tích giá trị biên (BVA - Boundary Value Analysis)

Khi thực hiện việc kiểm thử phần mềm theo dữ liệu, chúng ta kiểm tra xem đầu vào của người dùng, kết quả nhận được và kết quả tạm thời bên trong có được xử lý chính xác hay không.

Các điều kiện biên là tình trạng trực tiếp ở phía trên và dưới của các lớp tương đương đầu vào và lớp tương đương đầu ra. Việc phân tích các giá trị biên khác với phân hoạch tương đương theo hai điểm:

- Từ mỗi lớp tương đương, phân hoạch tương đương sẽ chọn phần tử bất kỳ làm phần tử đại diện, trong khi việc phân tích giá trị biên sử dụng một hoặc một số phần tử. Như vậy, mỗi biên của lớp tương đương chính là đích kiểm thử.
- Không chỉ chú ý tập trung vào những điều kiện đầu vào, các trường hợp kiểm thử cũng được suy ra từ việc xem xét các kết quả ra (tức các lớp tương đương đầu ra).

Rất khó có thể có thể liệt kê hết các hướng dẫn cụ thể cho các trường hợp. Tuy nhiên, cũng có một số nguyên tắc phân tích giá trị biên như sau:

1. Nếu điều kiện đầu vào xác định một khoảng giá trị giữa  $a$  và  $b$ , các trường hợp kiểm thử sẽ được thiết kế với giá trị  $a$  và  $b$ , và các giá trị sát trên và sát dưới  $a$  và  $b$ .
2. Nếu một điều kiện đầu vào xác định một số các giá trị, các trường hợp kiểm thử sẽ được phát triển để thực hiện tại các giá trị cực đại, cực tiểu. Các giá trị sát trên và dưới giá trị cực đại, cực tiểu cũng được kiểm thử.
3. Nguyên tắc 1 và 2 được áp dụng cho các điều kiện đầu ra.
4. Nếu cấu trúc dữ liệu chương trình bên trong được qui định các biên (chẳng hạn, mảng được định nghĩa giới hạn 100 mục), tập trung thiết kế trường hợp kiểm thử để thực thi cấu trúc dữ liệu tại biên của nó.

Ngoài ra, người kiểm thử có thể sử dụng sự xét đoán và sáng tạo của mình để tìm các điều kiện biên.

Tóm lại, chúng ta phải kiểm thử mỗi biên của một lớp tương đương về tất cả các phía. Một chương trình nếu vượt qua những trường hợp kiểm thử đó có thể vượt qua các kiểm thử khác từ lớp đó.

### 2.3.3. Kỹ thuật đồ thị nhân-quả (Cause-Effect Graph)

Trong nhiều trường hợp, việc cố gắng chuyển một chính sách hoặc một thủ tục trong ngôn ngữ tự nhiên vào phần mềm dẫn đến sự thất bại và các vấn đề khó hiểu. Đồ thị nhân - quả là một phương pháp thiết kế trường hợp kiểm thử trên cơ sở đưa ra một sự mô tả súc tích các điều kiện logic và các hành vi kèm theo.

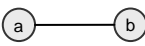
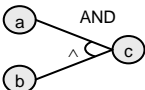
Đồ thị nhân - quả sử dụng mô hình các quan hệ logic giữa nguyên nhân và kết quả cho thành phần phần mềm. Mỗi nguyên nhân được biểu diễn như một điều kiện (đúng hoặc sai) của một đầu vào, hoặc kết hợp các đầu vào. Mỗi kết quả được biểu diễn như là một biểu thức Bool biểu diễn một kết quả tương ứng cho những thành phần vừa thực hiện.

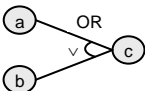
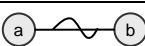
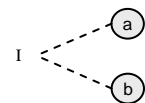
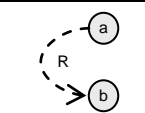
Đồ thị nhân - quả được tạo như sau:

- Tất cả các nguyên nhân (các đầu vào) và các kết quả (các đầu ra) được liệt kê dựa trên đặc tả và được định danh cho mỗi nhân - quả.
- Các quan hệ giữa các nguyên nhân (các đầu vào) và các kết quả (các đầu ra) được biểu diễn trong đồ thị làm rõ ràng các quan hệ logic.
- Từ đồ thị tạo ra bảng quyết định biểu diễn các quan hệ giữa nguyên nhân và kết quả. Dữ liệu kiểm thử được sinh ra dựa trên các qui tắc trong các bảng này.

Các ký hiệu được đơn giản hoá sử dụng trong đồ thị nhân quả, gồm các phần tử mô tả như bảng 2.3.

**Bảng 2.3 - Các ký hiệu trong đồ thị nhân quả**

STT	Ký hiệu	Ý nghĩa	Giải thích
1		Tương đương	Nếu ① đúng thì ② đúng.
2		AND (và)	Nếu ① đúng và ② đúng, thì ③ đúng

3		OR (hoặc)	Nếu <b>a</b> đúng hoặc <b>b</b> đúng, thì <b>c</b> đúng
4		NOT (phủ định)	Nếu <b>a</b> sai, thì <b>b</b> đúng
5		Loại trừ	Nếu <b>a</b> đúng, thì <b>b</b> sai, hoặc nếu <b>a</b> sai thì <b>b</b> đúng.
6		Bao hàm	<b>a</b> bao hàm <b>b</b>
7		Yêu cầu	<b>a</b> yêu cầu <b>b</b>

Các qui tắc trong bảng quyết định được mô tả như sau:

Tên bảng	Qui tắc				Tên bảng: cho biết tên logic
	1	2	..	n	
Điều kiện 1	Y	Y		Y	<b>Qui tắc:</b> đánh số để phân biệt các qui tắc quyết định logic.  <b>Các dòng điều kiện:</b> Mỗi dòng bao gồm các điều kiện để tạo quyết định cho chương trình.  Y: “true” N: “false”  -- : Không có quyết định được tạo ra.
Điều kiện 2	Y	--		Y	
Điều kiện 3	Y	--		N	
...	...	...		...	
Điều kiện n	--	--		Y	
Hành động 1	X	X		X	<b>Các hành động:</b> Mỗi dòng chỉ định có các xử lý được thực hiện hoặc không.  X: Xử lý được thực hiện.  -- : Không có xử lý được thực hiện.
Hành động 2	--	X		X	
Hành động 3	X	--		X	
...	...	...		...	
Hành động n	--	--		X	

**Ví dụ:** Để tính thuế thu nhập, người ta có mô tả sau:

- Người vô gia cư nộp 4% thuế thu nhập
- Người có nhà ở nộp thuế theo bảng sau:

<b>Tổng thu nhập</b>	<b>Thuế</b>
----------------------	-------------

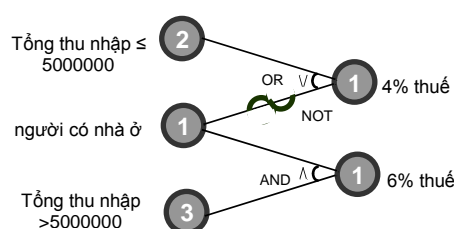
$\leq 5.000.000$  đồng                      4%

$> 5.000.000$  đồng                      6%

– Quan hệ giữa nguyên nhân (đầu vào) và kết quả (đầu ra) như sau:

Nguyên nhân	Kết quả
1. Người có nhà ở	11. Nộp 4 % thuế
2. Tổng thu nhập $\leq 5.000.000$ đồng	12. Nộp 6% thuế
3. Tổng thu nhập $> 5.000.000$ đồng	

– Đồ thị biểu diễn quan hệ logic rõ ràng giữa nguyên nhân-kết quả



**Hình 2.9 - Ví dụ đồ thị nhân-quả**

– Xây dựng bảng quyết định dựa trên đồ thị. Từ đây xây dựng được bốn trường hợp kiểm thử (một trường hợp cho việc nộp thuế 6% và ba trường hợp kiểm thử cần cho việc nộp thuế 4%).

**Bảng 2.4 – Ví dụ bảng quyết định**

Nguyên nhân và kết quả		Trường hợp kiểm thử			
		1	2	3	4
Nguyên nhân	1. Người có nhà ở	Y	Y	N	--
	2. Có tổng thu nhập $\leq 5.000.000$	N	Y	--	Y
	3. Có tổng thu nhập $> 5.000.000$	Y	N	--	--
Kết quả	11. Nộp thuế 4%	--	X	X	X
	12. Nộp thuế 6%	X	--	--	--

Để đảm bảo phủ nhân quả 100%, các trường hợp kiểm thử phải được phát sinh tương ứng với các qui tắc trong bảng quyết định bảng 2.4.

#### 2.3.4. Kiểm thử so sánh

Có một số trường hợp (như điện tử máy bay, điều khiển thiết bị năng lượng hạt nhân) trong đó độ tin cậy của phần mềm là tuyệt đối quan trọng, người ta thường gọi là phần mềm tuyệt đối đúng. Trong các ứng dụng như vậy phần cứng và phần mềm không cần thiết thường được sử dụng để tối thiểu khả năng lỗi. Khi phần mềm không cần thiết được phát triển, các nhóm công nghệ phần mềm riêng biệt phát triển các phiên bản độc lập của ứng dụng sử dụng cùng một đặc tả. Trong các trường hợp như vậy, mỗi phiên bản có thể được kiểm thử với cùng dữ liệu kiểm thử để đảm bảo rằng tất cả cung cấp đầu ra y như nhau. Sau đó tất cả các phiên bản được thực thi song song với so sánh thời gian thực các kết quả để đảm bảo tính chắc chắn. Các phiên bản độc lập là cơ sở của kỹ thuật kiểm thử hộp đen được gọi là *kiểm thử so sánh* hay *kiểm thử back-to-back*.

Kiểm thử so sánh là không rõ ràng. Nếu đặc tả mà tất cả các phiên bản được phát triển trên đó là có lỗi, thì tất cả các phiên bản sẽ có khả năng dẫn đến lỗi. Hơn nữa, nếu mỗi phiên bản độc lập tạo ra giống nhau, nhưng không đúng, các kết quả, kiểm thử điều kiện sẽ thất bại trong việc phát hiện lỗi.

#### 2.4. Đoán lỗi

Không cần một phương pháp đặc biệt nào, một số chuyên gia có thể kiểm tra các điều kiện lỗi bằng cách đoán lỗi dễ xảy ra. Trên cơ sở trực giác và kinh nghiệm, với các chương trình cụ thể, các chuyên gia đoán trước các loại lỗi có thể, rồi viết các trường hợp kiểm thử để phơi ra các lỗi này.

Một ý tưởng khác là chỉ ra các trường hợp kiểm thử liên quan đến giả định rằng lập trình viên đã mắc phải khi đọc đặc tả.

### CHƯƠNG 3

## CHIẾN LƯỢC KIỂM THỬ PHẦN MỀM

Chiến lược kiểm thử phần mềm tích hợp các phương pháp thiết kế trường hợp kiểm thử phần mềm thành một chuỗi các bước được lập kế hoạch rõ ràng để mang lại cấu trúc phần mềm có kết quả. Quan trọng là chiến lược kiểm thử phần mềm cung cấp một phương pháp vạch ra cho người phát triển phần mềm, tổ chức đảm bảo chất lượng, và khách hàng.

### **3.1 Nguyên lý thiết kế và kiểm thử phần mềm**

Trước khi áp dụng các phương pháp để thiết kế các trường hợp kiểm thử hiệu quả, kỹ sư phần mềm cần hiểu các nguyên lý cơ bản hướng dẫn việc kiểm thử phần mềm.

- Tất cả các kiểm thử phải có thể mô tả theo các yêu cầu của khách hàng.*
- Các kiểm thử phải được lập kế hoạch từ lâu trước khi kiểm thử bắt đầu.*
- Kiểm thử cần bắt đầu trong “phạm vi nhỏ” và quá trình hướng đến các kiểm thử trong “phạm vi rộng”.*
- Kiểm thử toàn diện là không thể.*
- Để đạt hiệu quả nhất, kiểm thử cần thực hiện bởi một nhóm độc lập thứ ba. Một lập trình viên nên tránh việc cố gắng kiểm thử chương trình của chính mình; đồng thời một tổ chức lập trình cũng không nên tự kiểm thử phần mềm của chính họ.*
- Các trường hợp kiểm thử phải được viết cho các điều kiện đầu vào không hợp lệ và không mong đợi, cũng như các điều kiện hợp lệ và được mong đợi. Và một phần cần thiết nữa là phải xác định đầu ra hay kết quả mong đợi. Vì vậy, một trường hợp kiểm thử phải gồm hai phần:*
  - + Mô tả chi tiết đầu vào hợp lệ và được mong đợi hoặc không hợp lệ, không được mong đợi.*
  - + Mô tả chi tiết đầu ra đúng cho một tập đầu vào tương ứng.*
- Kiểm tra kỹ kết quả của mỗi trường hợp kiểm thử.*

- Kiểm tra một chương trình xem nó có thực hiện đúng những gì nó phải thực hiện và những gì dự kiến không thực hiện.
- Tránh bỏ qua những trường hợp kiểm thử trừ khi chương trình thực sự là một sản phẩm bỏ đi.
- Không nên đặt kết quả dưới một giả định rằng sẽ không phát hiện một lỗi nào.
- Xác suất tồn tại lỗi càng cao ở những phần có nhiều lỗi được phát hiện.
- Kiểm thử phần mềm là một nhiệm vụ mang tư duy sáng tạo và tính trách nhiệm cao.

### **3.2 Phương pháp tiếp cận kiểm thử phần mềm**

Kiểm thử là một tập các hoạt động có thể được lập kế hoạch trước và được thực hiện một cách có hệ thống. Vì lý do này, một khuôn mẫu để kiểm thử phần mềm - tập các bước xác định các phương pháp thiết kế trường hợp kiểm thử - sẽ được định nghĩa cho quá trình phát triển phần mềm.

Chiến lược kiểm thử phần mềm cung cấp cho người phát triển một khuôn mẫu kiểm thử, và có các đặc điểm sau:

- Kiểm thử bắt đầu tại mức module và các công việc “phát triển” hướng tới việc tích hợp toàn bộ hệ thống.
- Các kỹ thuật kiểm thử khác nhau thích hợp tại những thời điểm khác nhau.
- Kiểm thử được thực hiện bởi người phát triển phần mềm và nhóm kiểm thử độc lập (cho các dự án lớn).
- Kiểm thử và gỡ rối là các hoạt động khác nhau, nhưng gỡ rối phải có trong mọi chiến lược kiểm thử.

#### **3.2.1. Xác minh và thẩm định**

Kiểm thử phần mềm là một phần của đề tài rộng hơn mà thường được đề cập tới như là *sự xác minh và thẩm định* (V&V). Thẩm định và xác minh là từ chung để



chỉ quá trình kiểm tra để đảm bảo phần mềm thoả mãn các yêu cầu của chúng và các yêu cầu đó đáp ứng yêu cầu của khách hàng. Xác minh là một tập các hoạt động đảm bảo rằng phần mềm cài đặt chức năng cụ thể một cách chính xác. Thẩm định là tập hợp hoạt động khác đảm bảo rằng phần mềm đã được xây dựng theo đúng các yêu cầu của khách hàng.

Có thể phát biểu theo cách khác:

Xác minh (Verification): “Sản phẩm có đúng với thiết kế không?”

Thẩm định (Validation): “Sản phẩm có đúng với yêu cầu thực tiễn không?”

Xác minh và thẩm định là một phần của hoạt động đảm bảo chất lượng phần mềm, bao gồm việc duyệt lại kỹ thuật, kiểm định chất lượng và cấu hình, theo dõi hiệu suất, mô phỏng, nghiên cứu tính khả thi, duyệt lại tài liệu, xem lại cơ sở dữ liệu, phân tích thuật toán, kiểm thử phát triển, kiểm thử chất lượng và kiểm thử cài đặt. Kiểm thử đóng vai trò rất quan trọng trong việc xác minh và thẩm định phần mềm và nhiều hoạt động khác trong phát triển phần mềm.

### **3.2.2. Tổ chức việc kiểm thử**

Với mọi dự án phần mềm, có một xung đột cố hữu về quyền lợi xuất hiện khi kiểm thử bắt đầu. Những người xây dựng phần mềm được yêu cầu kiểm thử phần mềm. Điều này tưởng như vô hại: sau tất cả, không có ai hiểu rõ chương trình hơn người phát triển.

Từ quan điểm tâm lý, phân tích và thiết kế phần mềm (cùng với mã hoá) là những công việc *xây dựng*. Người kỹ sư phần mềm tạo ra các chương trình máy tính, các tài liệu của nó và các cấu trúc dữ liệu liên quan.

Thường có một số quan niệm sai có thể dẫn đến kết luận sai từ sự tranh luận trên: (1) người phát triển phần mềm sẽ không thực hiện một kiểm thử nào ; (2) phần mềm sẽ được “tung lên tường” để một người lạ sẽ kiểm thử nó một cách khắt khe; và (3) những người kiểm thử tham gia dự án chỉ khi các bước kiểm thử sắp bắt đầu. Mỗi phát biểu trên là không đúng.

Người phát triển phần mềm luôn có trách nhiệm kiểm thử các đơn vị (module) riêng biệt của chương trình, đảm bảo rằng mỗi đơn vị thực hiện chức năng mà nó đã được thiết kế.

Vai trò của *nhóm kiểm thử độc lập (ITG)* là để loại bỏ các vấn đề cố hữu liên quan đến việc người phát triển tự kiểm thử những gì đã được xây dựng. Kiểm thử độc lập cũng loại bỏ các xung đột khác có thể xảy ra. Cuối cùng, nhân viên nhóm độc lập được trả lương để tìm các lỗi.

### 3.2.3. Chiến lược kiểm thử phần mềm

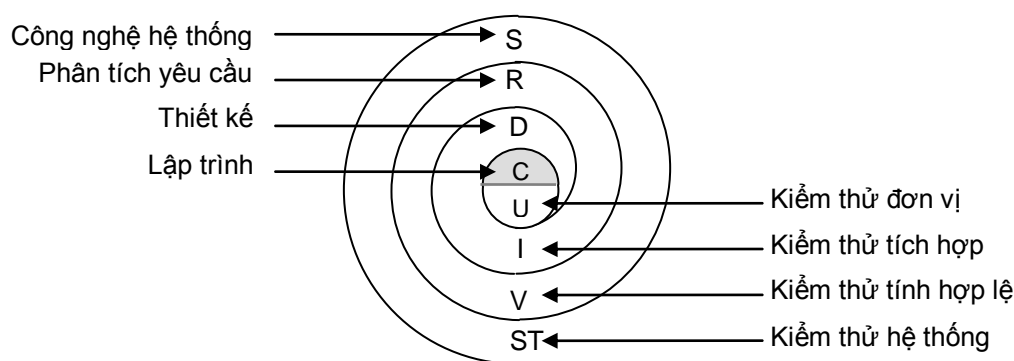
Tiến trình công nghệ phần mềm có thể được xem như một xoắn ốc, hình 3.1.

Việc phát triển phần mềm, đi vào dọc theo đường xoắn ốc, giảm dần các mức trừu tượng trên mỗi vòng, gồm các bước:

*Công nghệ hệ thống* → *Phân tích yêu cầu* → *Thiết kế* → *Mã hoá*.

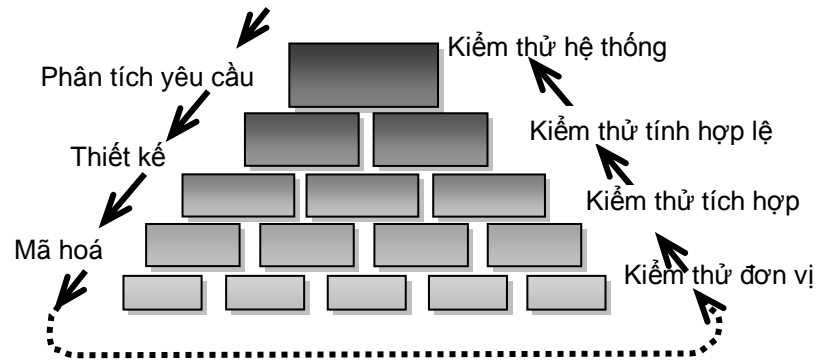
Chiến lược kiểm thử phần mềm cũng có thể di chuyển dọc theo đường xoắn ốc và đi ra theo đường xoắn ốc theo luồng mở rộng phạm vi kiểm thử trên mỗi vòng, tức theo thứ tự ngược lại, tương ứng như sau:

*Kiểm thử hệ thống* ← *Kiểm thử tích hợp lậ* ← *Kiểm thử tích hợp* ← *Kiểm thử đơn vị*.



**Hình 3.1 - Chiến lược kiểm thử**

Theo quan điểm thủ tục, kiểm thử nằm trong ngữ cảnh công nghệ phần mềm trên thực tế là dãy bốn bước được cài đặt tuần tự. Các bước được mô tả như hình 3.2.



**Hình 3.2 – Các bước kiểm thử**

- *Kiểm thử đơn vị*: tập trung trên mỗi module riêng biệt, đảm bảo rằng các chức năng của nó tương ứng với một đơn vị.
- *Kiểm thử tích hợp*: tập trung vào việc thiết kế và xây dựng kiến trúc phần mềm.
- *Kiểm thử tính hợp lệ*: trong đó các yêu cầu đã được thiết lập như một phần của việc phân tích yêu cầu phần mềm được thẩm định, dựa vào phần mềm đã xây dựng. Tiêu chuẩn hợp lệ cần được kiểm thử.
- *Kiểm thử hệ thống*: là một phần của công nghệ hệ thống máy tính, trong đó phần mềm và các thành phần khác của hệ thống được kiểm thử. Kiểm thử hệ thống nhằm xác minh rằng tất cả các thành phần hệ thống khớp nhau một cách hợp lý, và tất cả các chức năng hệ thống và hiệu suất là đạt được.

#### **3.2.4. Điều kiện hoàn thành kiểm thử**

Một câu hỏi khó trong kiểm thử phần mềm, đó là: “*Khi nào chúng ta hoàn thành việc kiểm thử - và làm thế nào để biết chúng ta đã kiểm thử đủ?*”. Không có câu trả lời dứt khoát cho câu hỏi này. Thật ra, “không bao giờ hoàn thành việc kiểm thử, trách nhiệm này thường chuyển từ người phát triển cho các khách hàng”. Mỗi lần, khách hàng (người sử dụng) thực hiện chương trình máy tính, chương trình sẽ được kiểm thử với tập dữ liệu mới. Có rất nhiều tranh cãi về câu trả lời cho câu hỏi trên, tuy nhiên, các kỹ sư phần mềm cần phải có các tiêu chuẩn chặt chẽ để xác định khi nào kiểm thử đạt hiệu quả.

Heiser (1997) [3] đưa ra bốn tiêu chuẩn có thể cho việc kết thúc kiểm thử:

- Khi dự án hết tiền hoặc thời gian.
- Khi đội ngũ kiểm thử không nghĩ được thêm một trường hợp kiểm thử nào.
- Khi kiểm thử được tiếp tục mà không phát hiện được bất kỳ lỗi mới nào.
- Khi đạt đến một mức của độ phù thích hợp.

Chiến lược phổ biến nhất hiện nay là kiểm thử cho đến khi dự án hết tiền hoặc thời gian. Tuy nhiên, chiến lược này sẽ bao gồm một vài rủi ro: nếu việc phát triển đã vượt quá ngân sách thì việc kiểm thử sẽ mất chất lượng.

Một chiến lược khác là sử dụng mô hình thống kê và lý thuyết độ tin cậy phần mềm, các mô hình thất bại phần mềm (được phát hiện trong quá trình kiểm thử) theo hàm thời gian thực hiện có thể được phát triển. Mô hình thất bại được gọi là *mô hình thời gian thực hiện lôga Poisson*, có dạng:

$$f(t) = \left(\frac{1}{p}\right) \ln[l_o p t + 1] \quad (0.1)$$

trong đó:

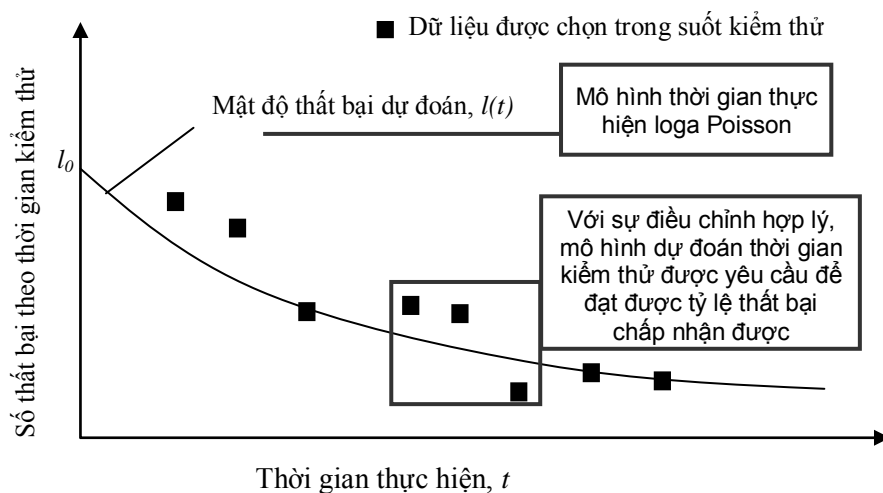
- $f(t)$  là số thất bại lũy tích được dự đoán xuất hiện mỗi lần phần mềm được kiểm thử trong khoảng thời gian thực hiện  $t$  nào đó.
- $l_o$  là *cường độ thất bại* phần mềm ban đầu (số thất bại trên một đơn vị thời gian) khi bắt đầu kiểm thử.
- $p$  là biến đổi số mũ trong cường độ thất bại do các lỗi được phát hiện và các khắc phục được thực hiện.

Cường độ thất bại tức thời,  $l(t)$  có thể được suy ra bằng cách tính đạo hàm  $f(t)$ :

$$l(t) = \frac{l_o}{l_o p t + 1} \quad (0.2)$$

Sử dụng quan hệ được ghi trong phương trình (3.2), người kiểm thử có thể dự đoán việc giảm lỗi trong quá trình kiểm thử. Cường độ lỗi thực tế có thể được vẽ

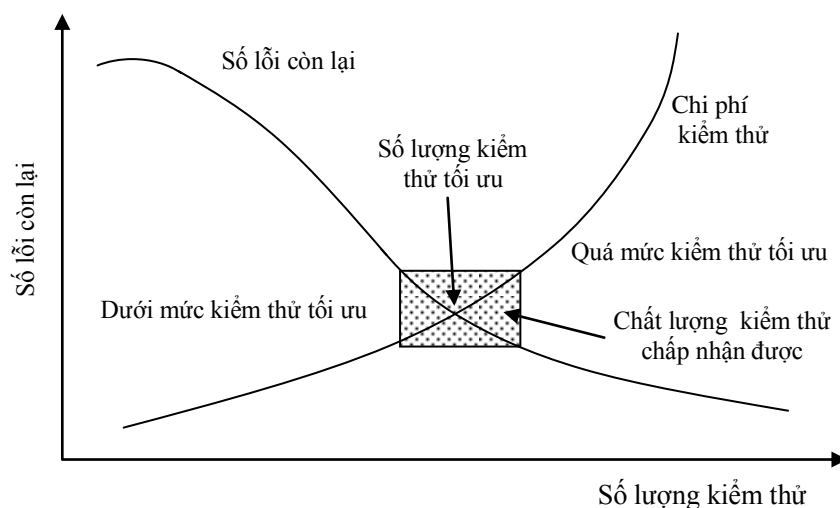
dọc theo đường cong dự đoán (Hình 3.3). Nếu dữ liệu thực tế được tập hợp lại trong quá trình kiểm thử và mô hình thời gian thực hiện loga Poisson là phù hợp với nhau trên một số điểm dữ liệu, mô hình có thể được sử dụng để dự đoán tổng thời gian thực hiện cần để đạt được tỷ lệ thất bại có thể chấp nhận được.



**Hình 3.3 - Mật độ lỗi là hàm thời gian thực hiện**

Bằng các phép tập hợp trong việc kiểm thử và sử dụng các mô hình độ tin cậy phần mềm đang tồn tại, có thể phát triển chỉ dẫn để trả lời cho câu hỏi: “Khi nào chúng ta hoàn thành kiểm thử?”

Hình 3.4 chỉ ra mối quan hệ giữa số lượng kiểm thử được thực hiện và số lỗi được tìm thấy. Nếu chúng ta kiểm thử quá nhiều thì chi phí sẽ tăng một cách khó chấp nhận được, ngược lại nếu kiểm thử ít thì chi phí thấp, nhưng sẽ còn nhiều lỗi. Số lượng kiểm thử tối ưu chỉ ra rằng chúng ta không kiểm thử quá nhiều nhưng cũng không ít quá.



**Hình 3.4- Quan hệ giữa chi phí kiểm thử và số lỗi chưa được phát hiện**

### 3.3. Kiểm thử đơn vị

Kiểm thử đơn vị tập trung vào việc xác minh trên đơn vị nhỏ nhất của thiết kế phần mềm – thành phần phần mềm, module hoặc lớp. Sử dụng các mô tả thiết kế thủ tục để hướng dẫn, các đường dẫn điều khiển quan trọng được kiểm thử để phát hiện lỗi trong phạm vi module. Độ phức tạp liên quan của các kiểm thử và các lỗi đã phát hiện được giới hạn bởi ràng buộc phạm vi thiết lập cho kiểm thử đơn vị. Kiểm thử đơn vị thường hướng hộp trắng, và các bước có thể được thực hiện song song trên nhiều module.

#### 3.3.1. Các lý do của kiểm thử đơn vị

Các kiểm thử xuất hiện như một phần của kiểm thử đơn vị được minh họa trong hình 3.5(a). Các kiểm thử nhằm phát hiện các lỗi trong các phạm vi của module bao gồm:

- Giao diện module,
- Cấu trúc dữ liệu cục bộ,
- Điều kiện biên,
- Đường dẫn độc lập,
- Đường dẫn xử lý lỗi.

• **Giao diện module:** được kiểm thử để đảm bảo thông tin vào, ra hợp lệ của đơn vị chương trình. Thường gồm một số kiểm thử cần thiết :

- Số tham số đầu vào có bằng số đối số không?
- Các thuộc tính của tham số và đối số có phù hợp không?
- Số đối số truyền vào cho module được gọi có phù hợp số tham số?
- Các thuộc tính đối số truyền cho module được gọi có phù hợp với thuộc tính của tham số?
- Khai báo biến toàn cục nhất quán trong các module?
- Các ràng buộc phù hợp với các tham số?
- Các đối số được truyền vào có đúng thứ tự?

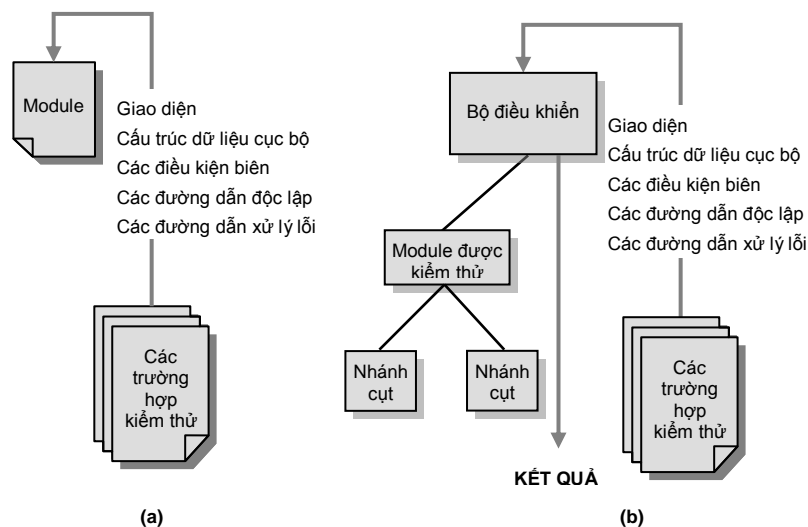
Khi module thực hiện vào ra, cần thực hiện các kiểm thử giao diện bổ sung:

- Các thuộc tính tập tin có đúng không?
- Các lệnh đóng/mở tập tin có đúng không?
- Các đặc tả hình thức phù hợp với lệnh vào/ra.
- Kích thước vùng đệm phù hợp với kích thước bản ghi?
- Các tập tin được mở trước khi sử dụng?
- Xử lý điều kiện kết thúc tập tin?
- Xử lý lỗi vào ra?

• **Cấu trúc dữ liệu cục bộ:** là nguồn lỗi phổ biến, được kiểm tra để đảm bảo rằng dữ liệu được lưu trữ tạm thời đảm bảo tính nguyên vẹn trong tất cả các bước thực hiện của thuật toán. Các trường hợp kiểm thử sẽ được thiết kế để phát hiện các loại lỗi sau:

- Kiểu không thích hợp hoặc mâu thuẫn.
- Giá trị khởi tạo hoặc giá trị mặc định không đúng.

- Tên biến không đúng (sai chính tả hoặc bị cắt bớt).
  - Kiểu dữ liệu không thống nhất.
  - Thiếu hoặc tràn bộ nhớ, các ngoại lệ.
- **Điều kiện biên:** Điều kiện biên được kiểm thử để đảm bảo rằng module hoạt động hợp lệ tại các biên được thiết lập đạt đến giới hạn hoặc xử lý giới hạn.
  - **Đường dẫn độc lập:** Tất cả các đường dẫn độc lập của cấu trúc điều khiển được thực hiện để đảm bảo rằng tất cả các câu lệnh trong module đã được thực hiện ít nhất một lần.
  - **Đường dẫn xử lý lỗi:** Một thiết kế tốt sẽ cho biết các điều kiện lỗi được biết trước và các đường dẫn xử lý lỗi được thiết lập để gửi lại hoặc kết thúc xử lý để dàng khi một lỗi xuất hiện. Một số lỗi tiềm ẩn sẽ được kiểm thử khi việc xử lý lỗi được đánh giá:
    - Mô tả lỗi khó hiểu.
    - Lỗi được chú giải không phù hợp với lỗi gặp phải.
    - Điều kiện lỗi dẫn đến sự can thiệp của hệ thống trước khi xử lý lỗi.
    - Xử lý điều kiện ngoại lệ không chính xác.
    - Mô tả lỗi không cung cấp đủ thông tin để hỗ trợ xác định nguyên nhân lỗi.



**Hình 3.5 – (a) Kiểm thử đơn vị; (b) Môi trường kiểm thử đơn vị**



### 3.3.2. Các thủ tục kiểm thử đơn vị

Kiểm thử đơn vị thường được xem như một phần phụ cho bước mã hoá. Sau khi mã nguồn đã được phát triển, được duyệt lại và được kiểm tra đúng cú pháp, thì bắt đầu thiết kế các trường hợp kiểm thử đơn vị. Việc xem lại thông tin thiết kế sẽ hướng dẫn cho việc thiết lập trường hợp kiểm thử phù hợp nhằm phát hiện các loại lỗi trên. Mỗi trường hợp kiểm thử phải được gắn liền với tập các kết quả mong đợi.

Vì mỗi module không phải là một chương trình độc lập, nên phần mềm *điều khiển* và/hoặc *nhánh cắt* cần được phát triển cho mỗi kiểm thử đơn vị. Môi trường kiểm thử đơn vị được minh hoạ trong hình 3.5(b).

Các nhánh cắt dùng để thay thế các module cấp dưới được gọi bởi các module được kiểm thử.

Kiểm thử đơn vị được đơn giản hoá khi module có sự liên kết cao được thiết kế. Khi chỉ một chức năng được gọi bởi một module, số các trường hợp kiểm thử được giảm xuống và các lỗi có thể dự đoán và phát hiện sớm hơn.

### 3.4. Kiểm thử tích hợp

Kiểm thử tích hợp là một kỹ thuật có hệ thống để xây dựng cấu trúc chương trình trong khi thực hiện các kiểm thử nhằm phát hiện các lỗi liên quan đến giao diện. Mục tiêu là lấy các thành phần đã được kiểm thử và xây dựng cấu trúc chương trình đã được mô tả bởi thiết kế.

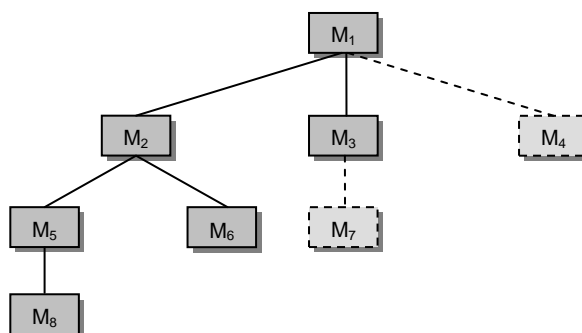
Tích hợp gia tăng là đối lập với cách tiếp cận big-bang. Giống như tô màu trên một tấm bản đồ, chương trình được xây dựng và được kiểm thử từng đoạn nhỏ, trong đó các lỗi sớm được cô lập và được hiệu chỉnh; các giao diện có khả năng được kiểm thử trọn vẹn hơn; và một cách tiếp cận kiểm thử có hệ thống có thể được áp dụng. Trong phần này chúng ta sẽ tìm hiểu một số chiến lược tích hợp gia tăng khác nhau.

### 3.4.1. Kiểm thử tích hợp từ trên xuống (Top-Down Integration)

Tích hợp top-down là cách tiếp cận gia tăng để xây dựng cấu trúc chương trình. Các module được tích hợp bằng cách di chuyển xuống dưới thông qua phân cấp điều khiển, bắt đầu với module điều khiển chính (chương trình chính). Các module mức dưới (và module mức dưới cùng) được kết hợp vào module chương trình chính thành cấu trúc theo chiều rộng hoặc chiều sâu.

Như hình 3.6, tích hợp theo chiều sâu sẽ gom tất cả các phần tử theo đường dẫn điều khiển chính của cấu trúc. Việc chọn đường dẫn điều khiển chính có thể tùy biến và phụ thuộc các đặc trưng của ứng dụng. Quá trình tích hợp được thực hiện theo các bước sau:

- 1) Module điều khiển chính được sử dụng như một bộ điều khiển, và các nhánh cụt được thay thế cho tất cả các module trực tiếp bên dưới module điều khiển chính.
- 2) Phụ thuộc vào cách tiếp cận tích hợp đã chọn (tức là theo chiều rộng hoặc chiều sâu), các nhánh cụt bên dưới được thay thế tại một thời điểm với các module hiện tại.
- 3) Các kiểm thử được thực hiện khi mỗi module được tích hợp.
- 4) Khi hoàn thành mỗi tập kiểm thử, nhánh cụt khác sẽ được thay thế bằng một module thực sự.
- 5) Kiểm thử hồi qui (xem mục 3.4.3) có thể được thực hiện để đảm bảo rằng các lỗi mới không xuất hiện.



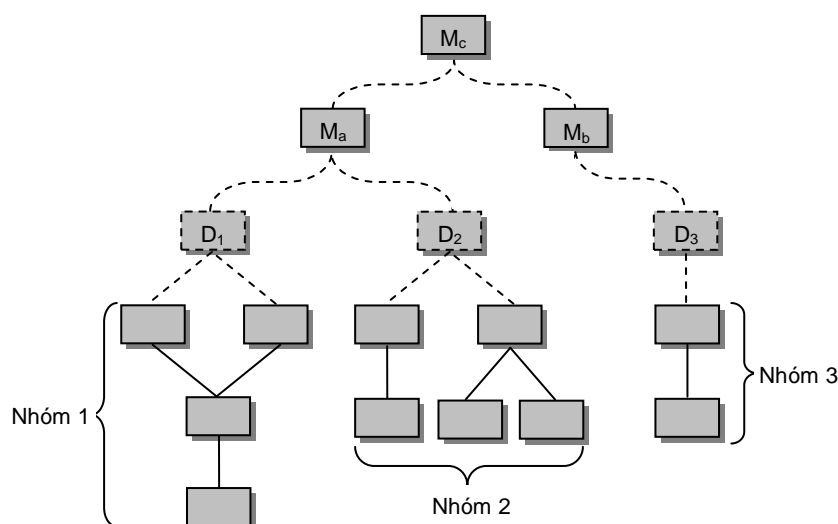
**Hình 3.6 - Kiểm thử Top-Down**

### 3.4.2. Chiến lược kiểm thử từ dưới lên (Bottom-Up Testing)

Kiểm thử tích hợp bottom-up, giống như tên gọi, bắt đầu xây dựng và kiểm thử với các module nguyên tử (tức là, các module ở mức thấp nhất trong cấu trúc chương trình). Chiến lược kiểm thử tích hợp bottom-up có thể được cài đặt theo các bước sau:

- 1) Các module mức thấp được kết hợp thành các *nhóm* (cluster).
- 2) Bộ điều khiển (chương trình điều khiển cho việc kiểm thử) được viết để phối hợp các trường hợp kiểm thử đầu vào và đầu ra.
- 3) Kiểm thử nhóm.
- 4) Các bộ điều khiển được loại bỏ và các nhóm được kết hợp chuyển dịch lên trên trong cấu trúc chương trình.

Tích hợp mẫu sau được minh họa trong hình 3.7. Các module được kết hợp tạo thành các nhóm 1, 2 và 3. Mỗi nhóm được kiểm thử sử dụng bộ điều khiển. Các module trong nhóm 1 và 2 là mức dưới của  $M_a$ . Các bộ điều khiển  $D_1$  và  $D_2$  được loại bỏ, và các nhóm được giao tiếp trực tiếp với  $M_a$ . Tương tự, bộ điều khiển  $D_3$  cho nhóm 3 được loại bỏ trước khi tích hợp với module  $M_b$ . Cả hai  $M_a$  và  $M_b$  cuối cùng sẽ được tích hợp với module  $M_c, \dots$



**Hình 3.7 – Tích hợp Bottom-Up**

### 3.4.3. Kiểm thử hồi qui

Mỗi lần một module mới được thêm vào như một phần của kiểm thử tích hợp, phần mềm sẽ thay đổi. Các đường dẫn luồng dữ liệu mới được thiết lập, vào ra I/O mới có thể xuất hiện, và logic điều khiển mới được yêu cầu. Các thay đổi có thể gây nên các vấn đề với các chức năng đã làm việc hoàn hảo trước đó. Trong ngữ cảnh chiến lược kiểm thử tích hợp, kiểm thử hồi qui là việc thực hiện lại một số tập con các kiểm thử đã được thực hiện trước đó để đảm bảo rằng các thay đổi không sinh ra những hiệu ứng không mong muốn.

*Kiểm thử hồi qui* là hoạt động trợ giúp để đảm bảo rằng các thay đổi (do kiểm thử hoặc do nguyên nhân khác) không đưa ra những hành vi hoặc những lỗi bổ sung không mong đợi.

*Kiểm thử hồi qui* có thể thực hiện thủ công, bằng cách thực hiện lại tập con tất cả các trường hợp kiểm thử hoặc sử dụng các công cụ thu phát tự động. Bộ kiểm thử hồi qui (tập con các kiểm thử sẽ được thực thi) gồm ba lớp các trường hợp kiểm thử khác nhau:

- Một mẫu đại diện của các kiểm thử sẽ thực hiện tất cả các chức năng của phần mềm.
- Các trường hợp kiểm thử bổ sung tập trung vào các chức năng phần mềm có khả năng bị tác động khi có sự thay đổi.
- Các kiểm thử tập trung vào các thành phần phần mềm vừa bị thay đổi.

### 3.4.4. Các ghi chú trên kiểm thử tích hợp

Các chiến lược kiểm thử tích hợp top-down và bottom-up có những ưu và nhược điểm. Ưu điểm của chiến lược này có xu hướng là nhược điểm của chiến lược khác.

**Bảng 3.1 – So sánh kiểm thử top-down và bottom-up**

<b>Kiểm thử top-down</b>	<b>Kiểm thử bottom-up</b>
<b>Ưu điểm</b>	
<ol style="list-style-type: none"> <li>1. Có ưu điểm nếu lỗi tập trung trên đỉnh của chương trình.</li> <li>2. Khi các chức năng vào/ra được bổ sung, thì đưa ra các trường hợp kiểm thử sớm hơn.</li> <li>3. Việc có chương trình khung sẽ sớm tạo ra một tư duy rõ ràng hơn và tạo tâm lý tốt khi kiểm thử.</li> </ol>	<ol style="list-style-type: none"> <li>1. Có ưu điểm nếu những lỗi chính xuất hiện về phía dưới của chương trình.</li> <li>2. Các điều kiện kiểm thử dễ dàng được tạo ra.</li> <li>3. Việc quan sát các kết quả kiểm thử là dễ hơn.</li> </ol>
<b>Nhược điểm</b>	
<ol style="list-style-type: none"> <li>1. Module nhánh cắt bắt buộc phải được tạo.</li> <li>2. Module nhánh cắt thường phức tạp hơn trong lần xuất hiện đầu tiên.</li> <li>3. Trước khi những chức năng vào/ra được thêm, việc đưa ra các trường hợp kiểm thử tại nhánh cắt có thể rất khó khăn.</li> <li>4. Việc tạo ra các điều kiện kiểm thử là không thể hoặc rất khó khăn.</li> <li>5. Việc quan sát đầu ra kiểm thử là khó khăn hơn.</li> <li>6. Làm cho người ta nghĩ nhầm rằng việc thiết kế chương trình và kiểm thử là chồng chéo nhau.</li> <li>7. Gây ra sự trì hoãn việc hoàn thành một module nào đó.</li> </ol>	<ol style="list-style-type: none"> <li>1. Các module điều khiển bắt buộc phải được tạo ra.</li> <li>2. Chương trình như là một thực thể không tồn tại cho đến khi module cuối cùng được tạo ra.</li> </ol>

Khi kiểm thử tích hợp được thực hiện, người kiểm thử cần chỉ ra các module tới hạn. Module tới hạn có một hoặc nhiều đặc trưng như sau:

- Lựa chọn một số yêu cầu phần mềm
- Có mức điều khiển cao (nằm tương đối cao trong cấu trúc chương trình).
- Là phức tạp hoặc dễ xảy ra lỗi.
- Có các yêu cầu thực hiện không rõ ràng.

Các module tới hạn cần được kiểm thử sớm nhất có thể. Hơn nữa, các kiểm thử hồi quy cần tập trung trên chức năng của module tới hạn.

### **3.5. Kiểm thử tính hợp lệ**

Điểm cao nhất của kiểm thử tích hợp, phần mềm được lắp ghép toàn bộ thành một gói; các lỗi giao diện đã được phát hiện và hiệu chỉnh, và dãy kiểm thử phần mềm cuối cùng - *kiểm thử tính hợp lệ* - có thể bắt đầu.

Đặc tả yêu cầu phần mềm tốt có chứa một phần được gọi là “*điều kiện hợp lệ*”, thiết lập cơ sở cho việc thực hiện kiểm thử tính hợp lệ.

#### **3.5.1. Điều kiện kiểm thử tính hợp lệ**

Tính hợp lệ phần mềm đạt được thông qua một dãy các kiểm thử hộp đen nhằm minh chứng sự phù hợp với các yêu cầu. Kế hoạch kiểm thử phác thảo các lớp kiểm thử sẽ được thực hiện, và thủ tục kiểm thử xác định các trường hợp kiểm thử sẽ được sử dụng nhằm cố gắng phát hiện các lỗi trong sự thoả mãn các yêu cầu.

Sau mỗi trường hợp kiểm thử hợp lệ được thực hiện, tồn tại một trong hai điều kiện sau:

- Các đặc trưng chức năng và khả năng thực hiện phù hợp với đặc tả và được chấp nhận.
- Sự sai lệch với đặc tả được phát hiện và danh sách các thiếu sót được tạo ra.

### 3.5.2. Duyệt lại cấu hình

Thành phần quan trọng của quá trình kiểm tra tính hợp lệ là *duyet lại cấu hình*. Mục đích của việc duyệt lại là nhằm đảm bảo rằng tất cả các thành phần của cấu hình phần mềm được phát triển hợp lý, được lập danh mục, và có các chi tiết cần thiết để hỗ trợ giai đoạn bảo trì của vòng đời phần mềm.

### 3.5.3. Kiểm thử Alpha và Beta

Người phát triển phần mềm gần như không thể đoán trước khách hàng sẽ sử dụng chương trình một cách thực sự như thế nào. Các tài liệu hướng dẫn sử dụng có thể bị hiểu sai; các tổ hợp dữ liệu lạ thường có thể thường xuyên được sử dụng; và đầu ra có vẻ rất rõ ràng đối với người kiểm thử có thể là khó hiểu cho người sử dụng.

Khi phần mềm được xây dựng theo hợp đồng đặt hàng của khách hàng, một chuỗi các *kiểm thử chấp nhận* được thực hiện cho phép khách hàng thẩm định tất cả các yêu cầu.

Nếu phần mềm được phát triển như một sản phẩm mang tính phổ dụng để sử dụng cho nhiều khách hàng, thì việc thực hiện các kiểm thử chấp nhận với mỗi khách hàng là không thực tế. Đa số những người xây dựng sản phẩm phần mềm sử dụng *kiểm thử alpha* và *beta* để phát hiện các lỗi mà chỉ người dùng cuối có thể tìm thấy.

- ***Kiểm thử alpha***

Kiểm thử alpha được thực hiện bởi khách hàng trong vị trí của người phát triển. Phần mềm được sử dụng trong môi trường tự nhiên cùng với người phát triển “xem xét trong vai trò” của người dùng và ghi lại các lỗi và các vấn đề sử dụng. Kiểm thử alpha được thực hiện trong môi trường được điều khiển.

- ***Kiểm thử beta***

Kiểm thử beta được thực hiện bởi một hoặc nhiều người dùng cuối của phần mềm. Không giống kiểm thử alpha, người phát triển nói chung không có mặt. Tuy

nhien, kiểm thử beta là một ứng dụng “sống” của phần mềm trong môi trường không được điều khiển bởi người phát triển.

### **3.6. Kiểm thử hệ thống**

Phần mềm chỉ là một thành phần của hệ thống lớn dựa trên máy tính. Cuối cùng, phần mềm kết hợp chặt chẽ với các thành phần khác của hệ thống (như phần cứng, con người, thông tin,...) và một dãy các kiểm thử tích hợp và tính hợp lệ của hệ thống được thực hiện.

Kiểm thử hệ thống thực tế là một tập các kiểm thử khác nhau với mục đích cơ bản là thực hiện đầy đủ hệ thống dựa trên máy tính. Mặc dù mỗi kiểm thử có một mục đích khác nhau, nhưng tất cả các công việc đều nhằm kiểm tra tất cả các thành phần hệ thống đã được tích hợp một cách hợp lý và thực hiện đúng các chức năng đã xác định.

#### **3.6.1. Kiểm thử khôi phục**

Nhiều hệ thống dựa trên máy tính cần khôi phục các sai sót và tiếp tục quá trình xử lý trong một khoảng thời gian xác định trước.

*Kiểm thử khôi phục* là một kiểm thử hệ thống có tác động đến phần mềm bị lỗi theo nhiều cách khác nhau và kiểm tra rằng sự khôi phục được thực hiện hợp lý. Nếu việc khôi phục là tự động (được thực hiện bởi chính hệ thống) thì việc khởi tạo lại, các kỹ thuật điểm kiểm soát, khôi phục dữ liệu và sự bắt đầu lại được ước lượng cho sự chính xác. Nếu việc khôi phục yêu cầu sự can thiệp của con người, thì thời gian trung bình để khôi phục được ước lượng để xác định giới hạn có thể chấp nhận được.

#### **3.6.2. Kiểm thử bảo mật**

Bất kỳ hệ thống dựa trên máy tính có quản lý các thông tin nhạy cảm hoặc dẫn đến các hoạt động có khả năng gây thiệt hại (hoặc lợi ích) không đúng cách cho các cá nhân là mục tiêu cho việc thâm nhập không đúng hoặc bất hợp pháp. *Kiểm thử tính bảo mật* cố gắng kiểm tra rằng các kỹ thuật bảo vệ xây dựng trong hệ thống sẽ bảo vệ nó khỏi việc truy nhập bất hợp pháp.



### 3.6.3. Kiểm thử ứng suất

Trong suốt quá trình kiểm thử phần mềm trước, các kỹ thuật hộp trắng và hộp đen dẫn đến sự ước lượng triệt để các chức năng và khả năng thực hiện chương trình chuẩn tắc. *Kiểm thử ứng suất* được thiết kế để đối chiếu chương trình với các trạng thái không chuẩn tắc.

Kiểm thử ứng suất thực hiện hệ thống với mục đích tìm các giới hạn trong đó hệ thống sẽ thất bại do yêu cầu về tài nguyên với chất lượng, tần suất, số lượng không bình thường, chẳng hạn:

- Các kiểm thử cụ thể được thiết kế cho những trường hợp tỷ lệ ngắt cao hơn bình thường.
- Tốc độ dữ liệu đầu vào có thể được tăng cường độ để xác định các chức năng sẽ đáp ứng đầu vào như thế nào.
- Thực hiện các trường hợp kiểm thử mà yêu cầu bộ nhớ tối đa hoặc các tài nguyên khác.
- Thiết kế các trường hợp kiểm thử có thể gây nên sự thất bại trong hệ điều hành ảo.
- Thực hiện các trường hợp kiểm thử gây nên sự tìm kiếm quá mức cho dữ liệu trên đĩa.

### 3.6.4. Kiểm thử khả năng thực hiện

Với các hệ thống nhúng và thời gian thực, phần mềm cung cấp chức năng được yêu cầu nhưng không tuân theo các yêu cầu về khả năng thực hiện là không được chấp nhận.

Các kiểm thử khả năng thực hiện thường được kết hợp với kiểm thử ứng suất và thường yêu cầu sự trang bị cả phần cứng và phần mềm. Bằng việc cung cấp hệ thống, người kiểm thử có thể phát hiện các trạng thái mà dẫn đến sự suy giảm và thất bại có thể của hệ thống.

## CHƯƠNG 4

### MỘT SỐ ỨNG DỤNG CỤ THỂ CỦA QUI TRÌNH KIỂM THỬ

#### 4.1. Mục tiêu

Phần này đi vào tìm hiểu một số bài toán cụ thể và nghiên cứu xây dựng các bộ dữ liệu kiểm thử cho bài toán cùng các chương trình kiểm thử tự động.

#### 4.2. Phương pháp luận

##### 4.2.1. Tổng quan về các phương pháp

Các chức năng và hành vi của hệ thống phần mềm hoặc một phần xác định của hệ thống không bị thay đổi hoặc ít nhất không bị đi ngược lại khi có sự thay đổi trong mã nguồn. Vì vậy, với mục đích đảm bảo chương trình không bị đi ngược lại, cần thiết phải thực hiện việc kiểm thử hồi qui. Có hai kiểu kiểm thử thường gọi kiểm thử hồi qui.

- Các kiểm thử chức năng* kiểm tra toàn bộ hệ thống có thoả mãn các yêu cầu và các mục đích như sự thực hiện.
- Kiểm thử đơn vị* được tạo bởi người lập trình để kiểm tra mỗi mặt của từng thành phần trong hệ thống như các lớp hoặc các module trong các hành vi được mong đợi.

Việc thực hiện kiểm thử hồi qui có nghĩa là thực hiện nhiều trường hợp kiểm thử khác nhau và thực hiện chúng thường xuyên. Vì thế không thể chấp nhận việc thực hiện thủ công, bởi vì sẽ rất mất thời gian và cũng không tin cậy. Do đó, cần thiết thực hiện một cách tự động.

##### 4.2.2. Phạm vi giải quyết

Có nhiều phương pháp sắp xếp khác nhau đã được nghiên cứu và phát triển. Mỗi phương pháp có ưu và nhược điểm riêng về độ phức tạp tính toán và thời gian thực hiện. Vì vậy, để lấy ví dụ tốt cho việc kiểm thử về khả năng thực hiện, chúng

ta chọn hai nhóm thuật toán sau để thực hiện kiểm thử hộp đen và so sánh kết quả thực hiện.

–Độ phức tạp  $O(n^2)$ : Insertion Sort, Selection Sort, Shell Sort, Bubble Sort.

–Độ phức tạp  $O(n \log n)$ : Heap Sort, Quick Sort, Merge Sort.

Việc kiểm thử đơn vị trên các modul của các thuật toán này sử dụng các phương pháp kiểm thử hộp trắng (phương pháp đường dẫn cơ sở). Và để minh họa cho qui trình kiểm thử tích hợp chúng ta thử lấy module MergeSort để thiết kế bộ kiểm thử.

#### 4.2.3. Phân loại các kiểu kiểm thử

Vấn đề kiểm thử phần mềm, ngoài mục đích phát hiện lỗi còn nhằm để đảm bảo chất lượng phần mềm. Do đó, khi chọn các thuật toán sắp xếp làm ví dụ về qui trình kiểm thử, ngoài lý do đã nêu trên, việc lựa chọn này còn vì nhằm kiểm tra về khả năng thực hiện của phần mềm (mỗi thuật toán có ưu nhược điểm khác nhau về bộ nhớ, thời gian,...).

Phát biểu cho một bài toán sắp xếp như sau:

–Với P là chương trình sắp xếp.

–S là bảng đặc tả cho P như sau:

+ P nhận đầu vào với một số nguyên N ( $N > 0$ ) và một dãy N số nguyên gọi là các phần tử của dãy.  $0 \leq K \leq e - 1$  với e nào đó.

+ K là phần tử bất kỳ của dãy.

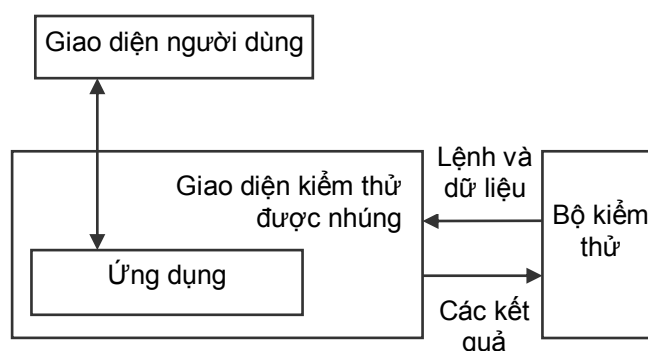
+ Chương trình P sắp xếp dãy theo thứ tự tăng dần và xuất ra dãy đã sắp xếp.

–P được xem là đúng với đặc tả S nếu và chỉ nếu: với mỗi đầu vào hợp lệ, đầu ra của P tương ứng với đặc tả của S.

#### 4.2.4. Tổ chức giao diện kiểm thử

Để có thể thực hiện các trường hợp kiểm thử, cần thiết kế cho chương trình trình một giao diện để kiểm thử. Có nhiều cách thiết kế giao diện khác nhau.

- Các kiểm thử hướng lô: khi giao diện cho chương trình là một dòng lệnh, thì chỉ các bộ kiểm thử tự động có thể được thực hiện, bắt đầu chương trình với các tham số đã cho và xem xét đầu ra, các tập tin có thể được tạo ra.
- Kiểm thử dựa trên luồng: Cách này bao gồm hầu hết các phương pháp kiểm thử.
- Kiểm thử GUI: các giao diện người dùng đồ họa cho việc kiểm thử có một vài vấn đề. Mặc dù có các công cụ cho các giao diện đồ họa kiểm thử bằng cách làm lại các hành động được ghi lại trước đó giống như các sự kiện phím bấm hoặc chuột và các màn hình so sánh, chúng không thể đối phó tốt với các sự cải biên thay đổi rất nhiều của các thành phần giao diện. Vì thế, các trường hợp kiểm thử cần được ghi lại sau mỗi thay đổi của giao diện người dùng.
- Các giao diện và mã kiểm thử nhúng: Trong trường hợp các kiểm thử trên lưu trình và hướng lô đơn giản trên giao diện của ứng dụng là không thể được bởi vì giao diện không cung cấp đủ truy cập hoặc khi GUI cần được kiểm thử, nhưng tập hợp các thành phần được sử dụng không cung cấp giao diện kiểm thử thích hợp, các giao diện kiểm thử nhúng trong ứng dụng là rất hữu ích.



**Hình 4.1– Giao diện kiểm thử nhúng**

### 4.3. Phát sinh các trường hợp kiểm thử

#### 4.3.1. Chiến lược kiểm thử

Việc kiểm thử thuật toán sắp xếp được thực hiện theo nhiều mức khác nhau:

- Mức cao: bao gồm việc kiểm thử chức năng, kiểm thử khả năng thực hiện.
- Mức thấp: bao gồm việc kiểm thử đơn vị và kiểm thử khi tích hợp các thành phần.

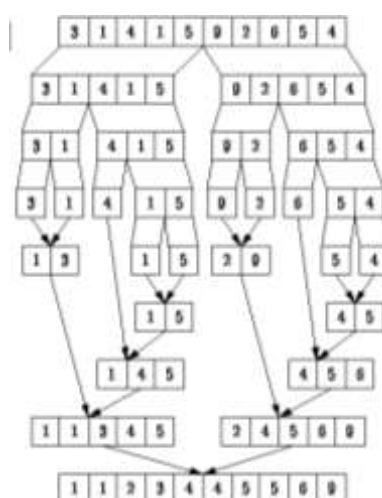
Với việc kiểm thử ở mức cao cho các thuật toán sắp xếp, chúng ta sẽ thiết kế các trường hợp kiểm thử nhằm kiểm tra khả năng tối đa phần tử của dãy và hiệu quả của thuật toán. Vì vậy, ở mức này chúng ta sẽ sử dụng các phương pháp hộp đen để sinh dữ liệu đầu vào và kiểm tra các kết quả đầu ra theo đặc tả của thuật toán.

Ở mức thấp, để thực hiện kiểm thử cho các thuật toán cần thiết kế các trường hợp kiểm thử với mục đích tìm lỗi của mã lệnh. Vì vậy, chúng ta cần thâm nhập vào mã lệnh của thuật toán và áp dụng các phương pháp hộp trắng để phát sinh các trường hợp kiểm thử, và xây dựng bộ dữ liệu kiểm thử tương ứng.

#### 4.3.2. Kiểm thử đơn vị

Trong kiểm thử hộp trắng, giao diện người dùng được bỏ qua. Các đầu vào và đầu ra được kiểm thử trực tiếp tại mức mã và các kết quả được so sánh theo đặc tả.

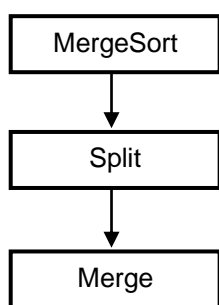
- **Module Merge Sort**



**Hình 4.2 – Minh họa thuật toán sắp xếp MergeSort**

Module MergeSort có cấu trúc như sau:

- Merge:** Module này nối hai mảng đã sắp xếp, các miền kề sát của mảng thành một mảng đơn, mảng đã sắp xếp. Sau đó vùng hai miền được ghi đè bởi mảng đã sắp xếp. Vì vậy chúng ta cần cung cấp không gian tạm thời là tham số cho hàm.
- Split :** Hàm tách nhận vào một miền và chia thành hai nửa, được gọi đệ qui cho mỗi nửa, nếu nó chứa nhiều hơn một phần tử và sau đó nối hai nửa kề sát bằng hàm nối.
- MergeSort:** Module này sẽ là giao diện người dùng cuối cho chức năng sắp xếp. Trong đó bộ nhớ tạm thời được cấp phát và sau đó hàm tách được gọi với các tham số khởi tạo.



Chúng ta sẽ khó để kiểm thử ba chức năng một cách độc lập, nhưng đề xuất gọi phụ thuộc chúng ta có thể áp dụng kiểm thử khi tích hợp các chức năng này bằng cách tích hợp từ trên xuống hoặc tích hợp từ dưới lên. Để dễ phát sinh các trường hợp kiểm thử và quan sát.

#### 4.3.2.1. Xác định các trường hợp kiểm thử có thể và thiết kế bộ kiểm thử

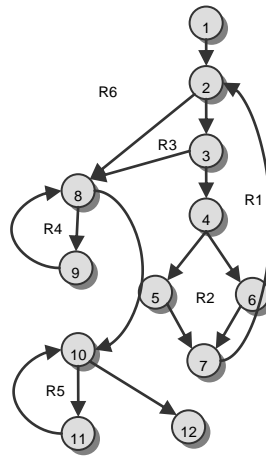
Các trường hợp kiểm thử có thể được sinh ra từ các mô tả chức năng của đơn vị. Có các phương pháp luận kiểm thử với vài cách tiếp cận để phát sinh trường hợp kiểm thử, nhưng đôi khi cần suy đoán để tìm các trường hợp kiểm thử có khả năng phát hiện các lỗi có thể.

Để thiết kế các trường hợp kiểm thử cho các module của mergesort chúng ta có thể áp dụng phương pháp kiểm thử đường dẫn cơ sở.

- **Các trường hợp kiểm thử cho chức năng merge**

Áp dụng phương pháp đường dẫn cơ sở, chúng ta xây dựng đồ thị lưu trình như sau:

- Vẽ đồ thị lưu trình cho hàm merge



**Hình 4.3 - Đồ thị lưu trình của chức năng merge**

–Đối chiếu hình 4.2, xác định độ phức tạp cyclomat  $V(G)$  theo 3 công thức:

$$V(G) = \text{số vùng} = 6$$

$$V(G) = \text{số cạnh} - \text{số đỉnh} + 2 = 16 - 12 + 2 = 6$$

$$V(G) = \text{Số đỉnh điều kiện} + 1 = 6 \text{ (Các đỉnh 2, 3, 4, 8, 10 là các đỉnh điều kiện)}$$

Vậy độ phức tạp cyclomat tính được  $V(G) = 6$ .

–Xác định tập cơ sở các đường dẫn độc lập

- + Đường dẫn 1 :  $1 \Rightarrow 2 \Rightarrow 8 \Rightarrow 10 \Rightarrow 12$
- + Đường dẫn 2:  $1 \rightarrow 2 \rightarrow 8 \rightarrow 10 \Rightarrow 11 \Rightarrow 10 \rightarrow \dots$
- + Đường dẫn 3:  $1 \rightarrow 2 \rightarrow 8 \Rightarrow 9 \Rightarrow 8 \rightarrow \dots$
- + Đường dẫn 4:  $1 \rightarrow 2 \Rightarrow 3 \Rightarrow 8 \rightarrow \dots$
- + Đường dẫn 5:  $1 \rightarrow 2 \rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 7 \Rightarrow 2 \rightarrow \dots$
- + Đường dẫn 6:  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \Rightarrow 6 \Rightarrow 7 \rightarrow 2 \rightarrow \dots$

Các dấu chấm lửng (...) phía sau các đường dẫn có nghĩa là một đường dẫn bất kỳ đi qua phần còn lại của cấu trúc đều có thể chấp nhận được.

–Chuẩn bị các trường hợp kiểm thử để mọi đường dẫn trong tập cơ sở đều được thực hiện. Dữ liệu nên chọn sao cho các điều kiện tại các đỉnh điều kiện là tập thích hợp cho mỗi đường dẫn.

+ Trường hợp 1 (ứng với đường dẫn 1):  $1 \Rightarrow 2 \Rightarrow 8 \Rightarrow 10 \Rightarrow 12$ .

*Dữ liệu cần sắp xếp (Data):* 1 3 2 7 5 6 2

*Các tham số (left, mid, right):* 4 4 3

*Kết quả mong đợi (Data):* 1 3 2 7 5 6 2

+ Trường hợp 2 (ứng với đường dẫn 2):  $1 \rightarrow 2 \rightarrow 8 \rightarrow 10 \Rightarrow 11 \Rightarrow 10 \rightarrow \dots$

*Dữ liệu sắp xếp (Data):* 1 3 2 7 5 6 2

*Các tham số (left, mid, right):* 4 4 6

*Kết quả mong đợi (data):* 1 3 2 7 5 6 2

+ Trường hợp 3 (ứng với đường dẫn 3):  $1 \rightarrow 2 \rightarrow 8 \Rightarrow 9 \Rightarrow 8 \rightarrow \dots$

Chúng ta xét thấy đường dẫn này không thể xảy ra.

+ Trường hợp 4 (ứng với đường dẫn 4):  $1 \rightarrow 2 \Rightarrow 3 \Rightarrow 8 \rightarrow \dots$

*Dữ liệu sắp xếp (Data):* 3 2 7 4 6 5 1

*Các tham số (left, mid, right):* 4 5 4

*Kết quả mong đợi (Data):* 3 2 7 4 6 5 1

+ Trường hợp 5 (ứng với đường dẫn 5):  $1 \rightarrow 2 \rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 7 \Rightarrow 2 \rightarrow \dots$

*Dữ liệu sắp xếp (Data):* 1 6 7 2 3 4 1 8

*Các tham số (left, mid, right):* 0 4 7

*Kết quả mong đợi (Data):* 1 3 4 1 6 7 2 8

+ Trường hợp 6 (ứng với đường dẫn 6)  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \Rightarrow 6 \Rightarrow 7 \rightarrow 2 \rightarrow \dots$

*Dữ liệu sắp xếp (Data):* 3 2 7 4 1 5 8 2 3

*Các tham số (left, mid, right):* 0 4 8

*Kết quả mong đợi (Data):* 1 3 2 5 7 4 8 2 3



**Bảng 4.1 - Bảng các trường hợp kiểm thử cho module Merge**

Số hiệu	Tên kiểm thử	Kiểu kiểm thử	Đặc tả	
			Đầu vào	Kết quả mong đợi
1.1	Merge1	Basic-Path	Data: 1 3 2 7 5 6 2 Các tham số : 4 4 3	1 3 2 7 5 6 2
1.2	Merge2	Basic-Path	Data: 1 3 2 7 5 6 2 Các tham số : 4 4 6	1 3 2 7 5 6 2
1.4	Merge4	Basic-Path	Data: 3 2 7 4 6 5 1 Các tham số : 4 5 4	3 2 7 4 6 5 1
1.5	Merge5	Basic-Path	Data: 1 6 7 2 3 4 1 8 Các tham số : 0 4 7	1 3 4 1 6 7 2 8
1.6	Merge6	Basic-Path	Data: 3 2 7 4 1 5 8 2 3 Các tham số : 0 4 8	1 3 2 5 7 4 8 2 3

- **Các trường hợp kiểm thử cho chức năng `split`**

Với chức năng `split` để chia mảng thành hai nửa, nếu mảng có một hoặc nhiều hơn một phần tử. Cứ như vậy gọi đệ qui cho hai nửa. Sau đó nối hai nửa thành mảng đã sắp xếp. Với chức năng này chúng ta có thể áp dụng phương pháp kiểm thử điều kiện để phát sinh các trường hợp kiểm thử.

Trong chức năng `split` các lệnh sẽ được thực hiện ứng với điều kiện có dạng:

$C = E1 > E2$  trong đó  $E1$  ứng với `left` và  $E2$  ứng với `right`.

Điều kiện ràng buộc cho  $C$  có dạng  $(D_1, D_2)$  với  $D_1$  và  $D_2$  là  $>$ ,  $=$  và  $<$ .

Vậy các trường hợp kiểm thử cho `split` được phát sinh như sau:

– Trường hợp 1: `left > right`

– Trường hợp 2: `left = right`

– Trường hợp 3: `left < right`

Trong đó `left`  $\geq 0$ , `right`  $<$  số phần tử của dãy cần tách.

**Bảng 4.2 – Các trường hợp kiểm thử cho module Split**

Số hiệu	Tên kiểm thử	Kiểu kiểm thử	Đặc tả	
			Đầu vào	Kết quả mong đợi
2.1	Split 1	Condition	Data: 1 5 2 4 3 Các tham số : 3 2	1 5 2 4 3
2.2	Split 2	Condition	Data: 3 1 4 1 5 Các tham số : 1 1	3 1 4 1 5
2.3	Split 3	Condition	Data: 9 2 6 5 4 Các tham số : 0 4	2 4 5 6 9

Data là dãy số cần sắp xếp

Các tham số là vị trí bắt đầu và kết thúc của dãy cần tách.

- **Các trường hợp kiểm thử cho chức năng MergeSort**

Chức năng này gọi thực hiện `split` và được kiểm thử cuối cùng, do đó khả năng xảy ra lỗi ở module này là khó có thể xảy ra. Tuy nhiên, chúng ta cần thiết kế các trường hợp kiểm thử cho module này để thực hiện việc kiểm thử thích hợp khi các module con được tích hợp lại thành module `MergeSort`.

Với lớp dữ liệu đầu vào có thể được tổ chức thành 3 lớp tương đương như sau:

- Dữ liệu đầu vào ngẫu nhiên.
- Dữ liệu đầu vào đã có thứ tự.
- Dữ liệu đầu vào đã xếp theo thứ tự ngược.

Mảng được sắp xếp gồm K phần tử ( $0 \leq K \leq N$ ,  $N = 32000$ ). Do đó miền dữ liệu đầu vào có thể được phân hoạch thành 3 lớp tương đương:

- Lớp tương đương hợp lệ: có K phần tử với  $0 < K < 32000$ .
- Lớp tương đương không hợp lệ: có K phần tử với  $K > 32000$
- Lớp tương đương không hợp lệ: có K phần tử với  $K < 0$ .

Áp dụng phương pháp phân tích giá trị biên, chúng ta sẽ có các trường hợp kiểm thử tại các giá trị biên của các lớp được phân hoạch.

- Trường hợp mảng có 0 phần tử
- Trường hợp mảng có 1 phần tử
- Trường hợp mảng có 32000 phần tử
- Trường hợp mảng có 32001 phần tử.

Từ việc phân tích các trường hợp hợp lệ và không hợp lệ trên, chúng ta có thể tổng kết bảng các trường hợp kiểm thử cho MergeSort như sau.

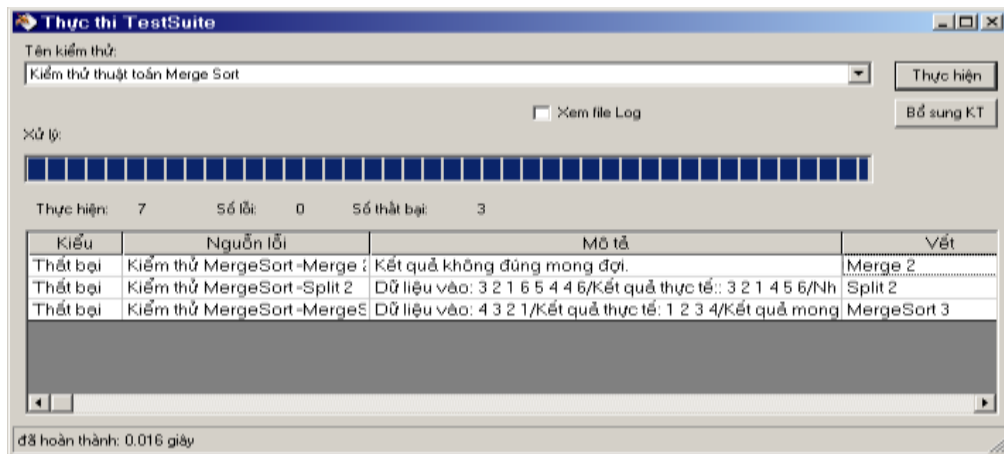
Số hiệu	Tên kiểm thử	Kiểu kiểm thử	Đặc tả	
			Đầu vào	Kết quả mong đợi
3.1	MergeSort	Partition	Data: 1 5 3 4 2 Các tham số : 5	1 2 3 4 5
3.2	MergeSort	Partition	Data: 1 2 3 4 5 Các tham số : 5	1 2 3 4 5
3.3	MergeSort	Partition	Data: 5 4 3 2 1 Các tham số : 5	1 2 3 4 5
3.4	MergeSort	Partition	Data: [] Các tham số: 0	[]
3.5	MergeSort	Partition-VBA	Data: 4 1 7 3 ...2 Các tham số: 32000	1 2 3 4 7 ...
3.6	MergeSort	VBA	Data: 5 Các tham số: 1	5
3.7	MergeSort	VBA	Data: 2 6 2 8 1 .. 7 Các tham số: 32001	1 2 2 6 7 8 ...

#### 4.3.2.2. Bộ điều khiển kiểm thử

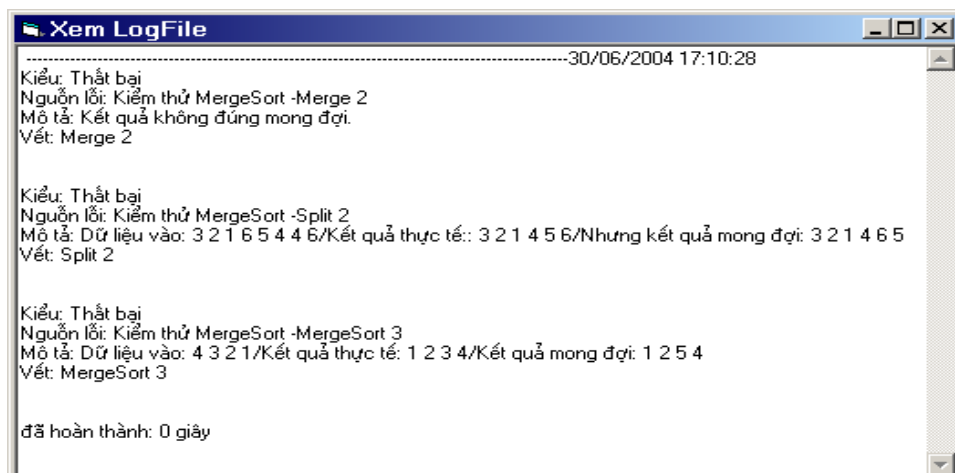
Sau khi xác định giao diện cho các module và các trường hợp kiểm thử, chúng ta có thể bắt đầu mã hoá bộ điều khiển kiểm thử, mà sau đó chúng ta sẽ sử dụng để liên lạc giữa bộ kiểm thử và mã thực. Chương trình sẽ đọc tất cả các tham số vào mảng và gọi thực hiện hàm cụ thể với các tham số. Sau đó, mảng kết quả được xuất ra để kiểm tra với bộ kiểm thử.

#### 4.3.2.3. Kết quả kiểm thử

Việc kiểm thử có thể phát hiện một vài lỗi nào đó trong mã, vì vậy kết quả kiểm thử ứng với các trường hợp kiểm thử cần phải được ghi lại vào tập tin .log của việc thực hiện kiểm thử.



**Hình 4.4 – Giao diện điều khiển kiểm thử thuật toán MergeSort**



**Hình 4.5 - Kết quả được ghi ra fileLog**

### 4.3.3. Kiểm thử khả năng thực hiện

Sau khi các thuật toán sắp xếp đã được cài đặt và được kiểm thử đơn vị để phát hiện lỗi, chúng ta cũng cần thực hiện việc kiểm thử hiệu quả của các thuật toán, để kiểm tra thời gian thực thi của các thuật toán, cũng như một số thông tin khác như số phép so sánh được thực hiện, số lần hoán vị cũng như đối với các thủ tục có gọi đệ qui thì đếm số lần thủ tục được gọi.

The screenshot shows a Windows-style application window titled "Form1". It is divided into several sections:

- Dữ liệu vào (Input Data):** A list box containing the numbers: 4842, 9833, 8710, 4900, 4748, 9327, 5094, 3652, 7284.
- Kết quả sắp xếp (Sorted Results):** A list box containing the sorted numbers: 3, 3, 4, 4, 5, 5, 7, 8, 9.
- Thông kê (Statistics):** A text area showing performance metrics for two different sorting methods.
  - Method 1:** Thuật toán: Insertion Sort, Dữ liệu : Ngẫu nhiên, Kích thước mảng : 10000, Giá trị nhỏ nhất : 1, Giá trị lớn nhất : 9999, Thời gian sắp xếp : 9546 mili giây, Kết quả sắp xếp: Đúng.
  - Method 2:** Thuật toán: Quick Sort, Dữ liệu : Ngẫu nhiên, Kích thước mảng : 10000, Giá trị nhỏ nhất : 3, Giá trị lớn nhất : 9997, Thời gian sắp xếp : 391 mili giây, Kết quả sắp xếp: Đúng.
- Controls:**
  - A dropdown menu for "T. toán sắp xếp:" (Sorting Algorithm) set to "Quick Sort".
  - A checked checkbox "Xem dữ liệu" (View Data).
  - A section for "Dữ liệu sắp xếp" (Sorting Data) with radio buttons for "Từ tập tin:" (From file) and "Tạo ngẫu nhiên:" (Generate random). The "Generate random" option is selected.
  - Fields for "Dữ liệu sắp xếp:" (Sorting Data) set to "Ngẫu nhiên".
  - Fields for "Số p.tử:" (Number of elements) set to 10000, "GTNN:" (Minimum) set to 0, and "GTLN:" (Maximum) set to 10000.
  - Buttons: "Tạo mảng" (Create array), "Sắp xếp" (Sort), and "Kiểm tra KQ" (Check result).

**Hình 4.6 – Giao diện điều khiển kiểm thử khả năng thực hiện của các thuật toán sắp xếp**

## KẾT LUẬN VÀ KIẾN NGHỊ

Kiểm thử phần mềm là một hoạt động quan trọng nhằm đảm bảo chất lượng phần mềm. Vấn đề của đề tài là khá mới mẻ ở Việt Nam.

Việc nghiên cứu lựa chọn các kỹ thuật và chiến lược kiểm thử phần mềm phù hợp giúp cho việc kiểm thử có hiệu quả, giảm chi phí và thời gian. Việc xây dựng tài liệu kiểm thử phần mềm hợp lý sẽ giúp cho việc tổ chức, quản lý và thực hiện kiểm thử có hiệu quả.

Trong khuôn khổ một luận văn thạc sĩ, học viên nghiên cứu các kỹ thuật và chiến lược kiểm thử, từ đó áp dụng để thiết kế kiểm thử cho một vài chương trình cụ thể.

Hiện nay vấn đề kiểm thử phần mềm hầu như vẫn chưa được đầu tư và quan tâm đúng mức. Và Việt Nam đang trong quá trình xây dựng một ngành công nghiệp phần mềm thì không thể xem nhẹ việc kiểm thử phần mềm vì xác suất thất bại sẽ rất cao, hơn nữa, hầu hết các công ty phần mềm có uy tín đều đặt ra yêu cầu nghiêm ngặt là nếu một phần mềm không có tài liệu kiểm thử đi kèm thì sẽ không được chấp nhận.

Kết quả nghiên cứu có thể áp dụng thực tế cho các đề tài và dự án phát triển phần mềm, nó cũng có thể làm tài liệu tham khảo cho các cơ sở đang tiến tới đưa qui trình kiểm thử phần mềm thành một qui trình bắt buộc trong dự án phát triển phần mềm của họ.

# TÀI LIỆU THAM KHẢO

## Tiếng Việt

1. Nguyễn Xuân Huy (1994), *Công nghệ phần mềm*, Đại học Tổng hợp Tp. Hồ Chí Minh.
2. Nguyễn Quốc Toàn (2000), *Bài giảng nhập môn Công trình học phần mềm*, Khoa Công nghệ - Đại học Quốc gia Hà Nội, trang 59- 63.
3. Pressman R, *Introduction to Software Engineering*, Ngô Trung Việt dịch, Nhà xuất bản Giáo dục 1997.

## Tiếng Anh

4. Beizer, B. (1995), *Black- box Testing*, Wiley.
5. Boehm. B. W. (1976), *Software Engineering*, IEEE Transactions on Computers.
6. British Standard (1998), *BS 7925- 1 - Standard for Software Component Vocabulary*, British Computer Society.
7. British Standard (1998), *BS 7925- 2 - Standard for Software Component Testing*, British Computer Society, p. 1- 15.
8. Cem Kaner, Jack Falk, Hung Quoc Nguyen (1999), *Testing Computer Software*, John Wiley & Sons, Inc., p. 27- 141.