|  |
| --- |
| BAN CƠ YẾU CHÍNH PHỦ  **HỌC VIỆN KĨ THUẬT MẬT MÃ**    **THỰC TẬP CƠ SỞ CHUYÊN NGÀNH**  ĐỀ TÀI  **PHƯƠNG PHÁP VÀ KỸ THUẬT KIỂM THỬ LUỒNG ĐIỀU KHIỂN TRONG KIỂM THỬ HỘP TRẮNG**  Giảng viên hướng dẫn : **Ths. Nguyễn Đức Hiếu**  Sinh viên thực hiện : **Phạm Hảo Nam – AT13G**  **Hoàng Thanh Huyền – AT13H**  **Phạm Thái Sơn – AT13H**  **Hà Nội, tháng 11 năm 2019** |

Mục lục

[DANH MỤC CÁC HÌNH VẼ 5](#_Toc21787882)

[DANH MỤC CÁC BẢNG 6](#_Toc21787883)

[LỜI CẢM ƠN 7](#_Toc21787884)

[LỜI MỞ ĐẦU 8](#_Toc21787885)

[CHƯƠNG 1: TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM 1](#_Toc21787886)

[1.1. Tổng quát về kiểm thử 1](#_Toc21787887)

[1.1.1. Khái niệm về kiểm thử phần mềm 1](#_Toc21787888)

[1.1.2. Các mục tiêu chính của kiểm thử phần mềm 1](#_Toc21787889)

[1.1.3. Các lỗi thường gặp trong kiểm thử 2](#_Toc21787890)

[1.2. Các cấp độ kiểm thử phần mềm 3](#_Toc21787891)

[1.2.1. Kiểm thử đơn vị (Unit test) 3](#_Toc21787892)

[1.2.2. Kiểm thử tích hợp (Integration Test) 4](#_Toc21787893)

[1.2.3. Kiểm thử hệ thống (System Test) 4](#_Toc21787894)

[1.2.4. Kiểm thử chấp nhận sản phẩm (Acceptance Test) 5](#_Toc21787895)

[1.3. Các kỹ thuật kiểm thử phần mềm 6](#_Toc21787896)

[1.3.1. Kiểm thử tĩnh – Static testing 6](#_Toc21787897)

[1.3.2. Kiểm thử động – Dynamic testing 7](#_Toc21787898)

[1.4. Kiểm thử hộp đen 7](#_Toc21787899)

[1.4.1. Định nghĩa 7](#_Toc21787900)

[1.4.2. Các kỹ thuật kiểm thử hộp đen 8](#_Toc21787901)

[1.4.3. Ưu, nhược điểm của kiểm thử hộp đen 8](#_Toc21787902)

[1.5. Giới thiệu chung về kiểm thử hộp trắng 9](#_Toc21787903)

[1.5.1. Kiểm thử hộp trắng 9](#_Toc21787904)

[1.5.2. Các kỹ thuật kiểm thử hộp trắng 9](#_Toc21787905)

[1.5.3. Đặc điểm 9](#_Toc21787906)

[1.6. Khi nào thì sử dụng kiểm thử hộp trắng 10](#_Toc21787907)

[1.6.1. Ưu điểm và nhược điểm của kiểm thử hộp trắng 10](#_Toc21787908)

[1.6.2. Sự khác biệt giữa kiểm thử hộp đen và kiểm thử hộp trắng 10](#_Toc21787909)

[1.7. Kết chương 13](#_Toc21787910)

[CHƯƠNG 2: KIỂM THỬ DÒNG ĐIỀU KHIỂN TRONG KIỂM THỬ HỘP TRẮNG 14](#_Toc21787911)

[2.1. Giới thiệu 14](#_Toc21787912)

[2.2. Đường thi hành (Execution path) 14](#_Toc21787913)

[2.3. Đồ thị dòng điều khiển 15](#_Toc21787914)

[2.4. Các độ đo kiểm thử 17](#_Toc21787915)

[2.4.1. Độ đo kiểm thử cấp 1 (C1) 17](#_Toc21787916)

[2.4.2. Độ đo kiểm thử cấp 2 (C2) 18](#_Toc21787917)

[2.4.3. Độ đô kiểm thử cấp 3 (C3) 18](#_Toc21787918)

[2.5. Kiểm thử dựa trên độ đo 20](#_Toc21787919)

[2.5.1. Kiểm thử cho độ đo C1 21](#_Toc21787920)

[2.5.2. Kiểm thử cho độ đo C2 22](#_Toc21787921)

[2.5.3. Kiểm thử cho độ đo C3 23](#_Toc21787922)

[2.6. Kiểm thử đường dẫn cơ bản 25](#_Toc21787923)

[2.6.1. Đồ thị luồng điều khiển 25](#_Toc21787924)

[2.6.2. Độ phức tạp chu trình Cyclomatic Complexity V(G) 26](#_Toc21787925)

[2.6.3. Quy trình xác định các đường cơ bản 27](#_Toc21787926)

[2.6.4. Phát sinh các trường hợp kiểm thử theo đường dẫn cơ sở 28](#_Toc21787927)

[2.7. Quy trình kiểm thử hộp trắng 31](#_Toc21787928)

[2.8. Quy trình xác định Test path 31](#_Toc21787929)

[2.9. Kiểm thử vòng lặp 32](#_Toc21787930)

[2.9.1. Vòng lặp đơn giản 32](#_Toc21787931)

[2.9.2. Vòng lặp lồng nhau 33](#_Toc21787932)

[2.9.3. Vòng lặp nối tiếp 33](#_Toc21787933)

[2.9.4. Vòng lặp phi cấu trúc 34](#_Toc21787934)

[2.10. Kết chương 34](#_Toc21787935)

[CHƯƠNG 3: THỰC NGHIỆM KIỂM THỬ LUỒNG ĐIỀU KHIỂN 35](#_Toc21787936)

[3.1. Đặc tả của bài toán 35](#_Toc21787937)

[3.2. Mã code của chương trình 35](#_Toc21787938)

[3.3. Dựng biểu đồ luồng điều khiển 37](#_Toc21787939)

[3.4. Biểu đồ luồng điều khiển 39](#_Toc21787940)

[3.5. Cách xác định số Testcase 40](#_Toc21787941)

[3.6. Xác định các đường tuyến tính độc lập 40](#_Toc21787942)

[3.7. Bảng Testcase 41](#_Toc21787943)

[3.8. Giới thiệu về công cụ kiểm thử chất lượng code Sonarqube 45](#_Toc21787944)

[3.8.1. Giới thiệu 45](#_Toc21787945)

[3.8.2. Cài đặt trên hệ điều hành Ubuntu 18.04 46](#_Toc21787946)

# DANH MỤC CÁC HÌNH VẼ

[Hình 1.1. Bốn cấp độ cơ bản của kiểm thử phần mềm 3](#_Toc21604196)

[Hình 1.2. Ví dụ về kiểm thử tĩnh 6](#_Toc21604197)

[Hình 2.1. Các thành phần cơ bản của đồ thị chương trình 15](#_Toc21604198)

[Hình 2.2. Các cấu trúc điều khiển phổ biến của chương trình 16](#_Toc21604199)

[Hình 2.3. Ví dụ về đồ thị dòng điều khiển 16](#_Toc21604200)

[Hình 2.4. Quy trình kiểm thử đơn vị chương trình dựa trên độ đo. 20](#_Toc21604201)

[Hình 2.5. Mã nguồn của hàm **foo** và đồ thị dòng điều khiển của nó 21](#_Toc21604202)

[Hình 2.6: Hàm **foo** và đồ thị dòng điều khiển ứng với độ đo C3 24](#_Toc21604203)

[Hình 2.7. Đồ thị luồng điều khiển 25](#_Toc21604204)

[Hình 2.8. Ví dụ sơ đồ chu trình Cyclomatic Complexity 27](#_Toc21604205)

[Hình 2.9 Mã nguồn code hàm **average** 28](#_Toc21604206)

[Hình 2.10. Đồ thị luồng điều khiển hàm **average** minh họa phát sinh các trường hợp kiểm thử theo đường dẫn cơ sở 29](#_Toc21604207)

[Hình 2.11. Các kiểu vòng lặp 32](#_Toc21604208)

# DANH MỤC CÁC BẢNG

[Bảng 1.1. So sánh các tiêu chuẩn giữa kiểm thử hộp đen và kiểm thử hộp trắng 11](#_Toc21604225)

[Bảng 1.2. So sánh ưu và nhược điểm giữa kiểm thử hộp đen và kiểm thử hộp trắng 12](#_Toc21604226)

[Bảng 2.1. Các ca kiểm thử cho độ đo C1 của hàm **foo** 18](#_Toc21604227)

[Bảng 2.2. Các trường hợp cần kiểm thử của độ đo C2 với hàm **foo** 18](#_Toc21604228)

[Bảng 2.3. Các ca cần kiểm thử cho độ đo C2 với hàm **foo** 18](#_Toc21604229)

[Bảng 2.4. Các trường hợp cần kiểm thử của độ đo C3 với hàm **foo** 19](#_Toc21604230)

[Bảng 2.5. Các ca cần kiểm thử của độ đo C3 với hàm **foo** 20](#_Toc21604231)

[Bảng 2.6. Các ca kiểm thử cho độ đo C1 của hàm foo 22](#_Toc21604232)

[Bảng 2.7. Các ca kiểm thử cho độ đo C2 của hàm foo 23](#_Toc21604233)

[Bảng 2.8. Các ca kiểm thử cho độ đo C3 của hàm **foo** 24](#_Toc21604234)

[Bảng 2.9. Các ca kiểm thử cho kiểm thử vòng lặp while của hàm **average** 33](#_Toc21604235)

# LỜI CẢM ƠN

Sau thời gian tìm hiểu, chúng em đã hoàn thành báo cáo đề tài của mình. Lời đầu tiên chúng em xin chân thành gửi tới thầy Ths. Nguyễn Đức Hiếu - Giảng viên khoa công nghệ thông tin Học viện Kỹ thuật Mật Mã, đã tận tình hướng dẫn và chỉ bảo chúng em trong suốt thời gian vừa qua.

Em cũng xin chân thành cảm ơn các thầy cô trong Học viện đã tận tình giảng dạy mang tới cho chúng em những kiến thức bổ ích nhất để phục vụ cho quá trình học tập cũng như công việc sau này của mình.

*Chúng em xin chân thành cảm ơn!*

**NHÓM SINH VIÊN THỰC HIỆN ĐỀ TÀI**

# LỜI MỞ ĐẦU

Kiểm thử phần mềm là một hoạt động giữ vai trò rất quan trọng để đảm bảo chất lượng phần mềm và được tiến hành để cung cấp cho các bên liên quan thông tin về chất lượng của sản phẩm hoặc dịch vụ được kiểm thử. Kiểm thử có thể cung cấp cho các công ty một quan điểm, một cách nhìn độc lập về phần mềm để từ đó cho phép đánh giá và thấu hiểu được những rủi ro trong quá trình triển khai phần mềm và đưa ra những giải pháp xử lý thích hợp với mỗi phần mềm đó.

Với mục đích phát hiện lỗi, kiểm thử phần mềm thường phải trải qua các bước như sau: Tạo dữ liệu thử, thực thi phần mềm trên dữ liệu thử và quan sát kết quả nhận được. Trong các bước này, bước tạo dữ liệu đóng vai trò quan trọng nhất, bởi vì chúng ta không thể tạo ra mọi dữ liệu từ miền vào của chương trình, mà chúng ta chỉ có thể tạo ra các dữ liệu thử có khả năng phát hiện lỗi cao nhất. Để thuận tiện và hiệu quả hơn trong quá trình kiểm thử, một số cộng đồng lớn đã viết ra những công cụ, chương trình kiểm thử mà chỉ cần chạy kiểm tra phần mềm sẽ thông báo cho chúng ta về lỗi, hiệu năng của chương trình phần mềm mà chúng ta không cần nhập dữ liệu thủ công.

Kiểm thử giúp rút ngắn thời gian và giảm chi phó cho sản phẩm phần mềm. Nó giúp cho các chuyên gia kiểm thử tìm ra lỗi trong quá trình tạo ra phần mềm và đảm bảo hơn về chất lượng. Kiểm thử thực hiện chặt chẽ sẽ hạn chế lỗi, tuy nhiên trong phần mềm vẫn còn tiềm ẩn các lỗi và có thể phát sinh bất cứ lúc nào dẫn đến khả năng gây thiệt hại cho nhà sản xuất vì vậy cần thực hiện quá trình kiểm thử liên tục, đều đặn và xuyên suốt trong các giai đoạn phát triển của phần mềm. Đó là phương pháp tốt nhất để đảm bảo cho các yêu cầu của người dùng về thiết kế và ứng dụng phần mềm được đáp ứng đầy đủ.

Kiểm thử hộp trắng (được biết đến là kiểm thử tính rõ ràng, kiểm thử hộp kính, kiểm thử hộp trong suốt và kiểm thử cấu trúc) giúp kiểm thử được cấu trúc nội bộ hoặc hoạt động của một chương trình, như tương phản với chức năng được bộc lộ của người dùng cuối. Một góc nhìn nội bộ của hệ thống trong kiểm thử hộp trắng giống như là các kĩ thuật lập trình được sử dụng để thiết kế ra các tình huống kiểm thử. Các Tester lựa chọn yếu tố đầu vào để thực hiện đường dẫn thông qua các mã và xác định được kết quả đầu ra thích hợp.

Trong kiểm thử hộp trắng có phương pháp kiểm thử về luồng điều khiển của chương trình, kiểm thử dựa trên mã nguồn theo luồng chạy xử lý của chương trình, phương pháp này tập trung kiểm thử tính đúng đắn của các giải thuật sử dụng trong các chương trình hoặc đơn vị phần mềm.

Sau thời gian tìm hiểu và nghiên cứu chúng em nhận ra được luồng điều khiển chạy của một phần mềm rất quan trọng và mang tính quyết định của sản phẩm đó nên chúng em đã chọn hướng nghiên cứu: ***“Tìm hiều về phương pháp và kĩ thuật kiểm thử luồng điều khiển trong kiểm thử hộp trắng”*** cho đề tài của mình.

**Bố cục của đề tài gồm 3 chương:**

*Chương 1:* Tổng quan về kiểm thử phần mềm: Khái niệm về kiểm thử phần mềm, các cấp độ kiểm thử phần mềm, các kỹ thuật và phương pháp kiểm thử chính

*Chương 2:* Kiểm thử dòng điều khiển trong kiểm thử hộp trắng: Cách vẽ đồ thị dòng điều khiển, các độ đo kiểm thử, kiểm thử đường dẫn cơ bản, quy trình kiểm thử hộp trắng, cách xác định các Test path và kiểm thử vòng lặp.

*Chương 3:* Thực nghiệm kiểm thử luồng điều khiển: Thực hiện kiểm thử trên một mã nguồn do chúng em tự code và phát triển

# TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM

## Tổng quát về kiểm thử

### Khái niệm về kiểm thử phần mềm

Kiểm thử phần mềm là quá trình thực thi một hệ thống phần mềm để xác định xem phần mềm có đúng với đặc tả không và 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 bảo đảm rằng lỗi sẽ được sửa một cách kịp thời trước khi đưa vào hoạt động hoặc trong quá trình hoạt động

Kiểm thử có thể cung cấp cho các công ty, doanh nghiệp một quan điểm, một cách nhìn độc lập về phần mềm để từ đó cho phép đánh giá và thấu hiểu được những rủi ro trong quá trình triển khai phần mềm.

Nếu kiểm thử được tiến hành thành công thì nó sẽ làm lộ ra những lỗi trong phần mềm. Việc kiểm thử phần mềm làm việc theo đặc tả nên các yêu cầu hiệu năng dường như là được đáp ứng. Bên cạnh đó, dữ liệu thu thập được khi việc kiểm thử tiến hành đưa ra một chỉ dẫn tốt về độ tin cậy phần mềm và một chỉ dẫn nào đó về phẩm chất phần mềm với tư cách toàn cục. Có một điều mà kiểm thử không làm được: Kiểm thử không thể chứng minh được việc không có khiếm khuyết, nó chỉ có thể chứng minh được khiểm khuyết phần mềm hiện hữu.

### Các mục tiêu chính của kiểm thử phần mềm

Cũng giống như các sản phẩm máy móc và các hệ thống vật lý, mục tiêu của kiểm thử phần mềm là để đảm bảo hệ thống phần mềm có thể làm việc tốt như mong muốn khi chúng được đem ra thị trường tới tay khách hàng và người sử dụng. Cách thường dùng để đưa ra những kiểm chứng về chất lượng cho sản phẩm là đưa sản phẩm vào “chạy thử” hay được kiểm tra trong phòng thí nghiệm trước khi phân phối sản phẩm ra thị trường. Trong ngành công nghệ phần mềm, các sản phẩm phần mềm được kiểm tra, chạy thử được gọi chung là kiểm thử phần mềm

Quy trình kiểm thử phần mềm bao gồm các hoạt động chính như sau:

1. Chuẩn bị và lập kế hoạch kiểm thử: Đặt ra các mục tiêu cụ thể cho việc kiểm thử, chọn chiến lược kiểm thử, chuẩn bị các trường hợp kiểm thử cụ thể và các thủ tục kiểm thử.
2. Thực thi: Tiến hành kiểm thử theo kế hoạch, quan sát và thu thập các kết quả.
3. Phân tích và theo dõi: Sau khi có kết quả, ta sẽ tiến hành phân tích để tìm ra các lỗi. Nếu một lỗi nào đó xuất hiện thì đưa ra giải pháp sửa đổi, sửa đổi và tiếp tục theo dõi đến khi lỗi đó được sửa thành công

Điểm quan trọng trong thực thi kiểm thử đó là tránh để lỗi từ một trường hợp kiểm thử ảnh hưởng đến các trường hợp kiểm thử khác

Mục tiêu chính của quá trình kiểm thử:

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

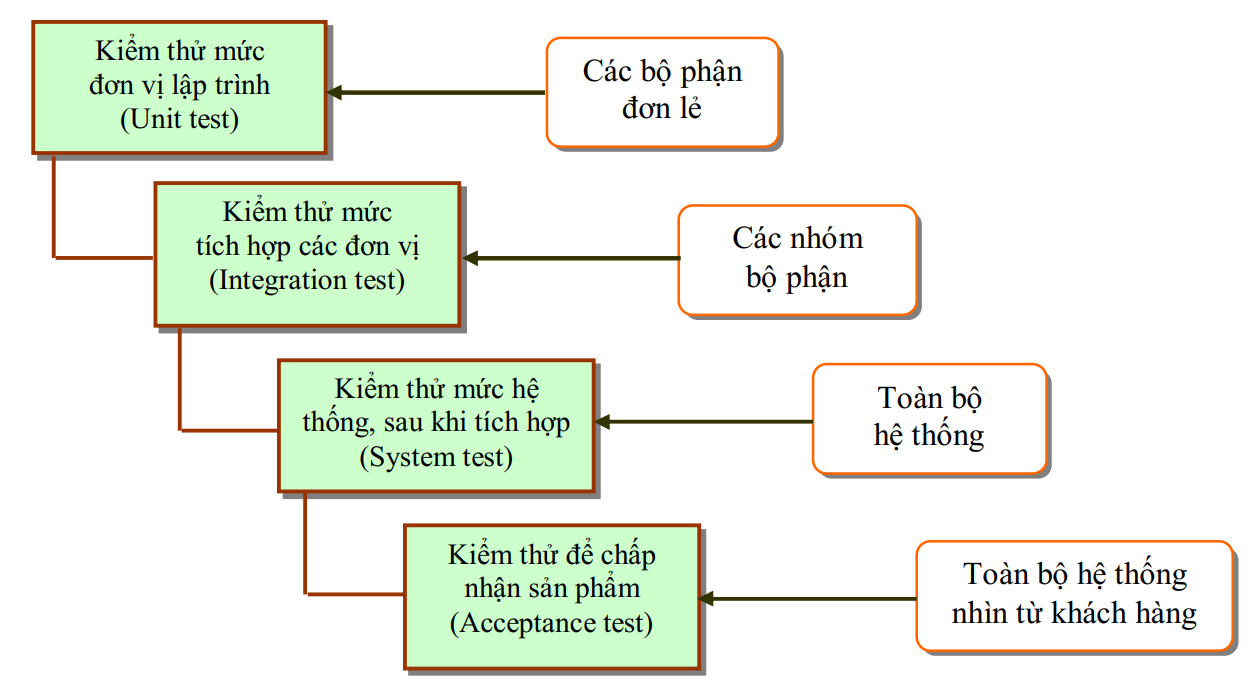
### Các lỗi thường gặp trong kiểm thử

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

* Lỗi cú pháp: Lỗi cú pháp là những từ sai chính tả hay ngữ pháp câu không chính xác và rất rõ ràng trong khi kiểm thử thử giao diện phần mềm. Xin lưu ý rằng chúng tôi KHÔNG đề cập đến cú pháp lỗi trong code. Trình biên dịch sẽ cảnh báo các nhà phát triển về bất kỳ lỗi cú pháp xuất hiện trong code.
* Lỗi xử lý: Bất kỳ lỗi nào xảy ra khi người dùng đang tương tác với các phần mềm cần phải được xử lý một cách rõ ràng và có ý nghĩa. Nếu không, nó được gọi là một lỗi Xử lý lỗi. Nhìn vào bức ảnh này. Thông báo lỗi đưa ra không chỉ ra cái thực sự gây ra lỗi là gì. Đó là do nó thiếu trường bắt buộc, lỗi đang lưu, lỗi tải trang hoặc lỗi hệ thống. Do đó, đây là một lỗi lỗi xử lý Khi có thể, các bước tiếp theo sẽ được liệt kê cho người sử dụng để làm theo. Nếu phần mềm có xác thực những trường bắt buộc cần phải được nhập trước khi lưu thông tin trên một form, các thông báo xác thực phải rõ ràng và chỉ ra những hành động cần thiết cho người dùng.
* Lỗi tính toán: Logic không tốt, công thức tính toán không chính xác, kiểu dữ liệu không phù hợp, lỗi lập trình, các vấn đề gọi chức năng.

## Các cấp độ kiểm thử phần mềm

Cấp độ kiểm thử phần mềm được thể hiện như hình



Hình 1.1. Bốn cấp độ cơ bản của kiểm thử phần mềm

### Kiểm thử đơn vị (Unit test)

Một đơn vị (Unit) là một thành phần phần mềm nhỏ nhất mà ta có thể kiểm thử được, ví dụ: Các hàm (Function), thủ tục (Procedure), lớp (Class), hoặc các phương thức (Method).

Kiểm thử đơn vị thường do lập trình viên thực hiện. Công đoạn này cần được thực hiện càng sớm càng tốt trong giai đoạn viết code và xuyên suốt chu kỳ phát triển phần mềm.

Mục đích của kiểm thử đơn vị là bảo đảm thông tin được xử lý và kết xuất (khỏi Unit) là chính xác, trong mối tương quan với dữ liệu nhập và chức năng xử lý của Unit. Điều này thường đòi hỏi tất cả các nhánh bên trong Unit đều phải được kiểm tra để phát hiện nhánh phát sinh lỗi.

Cũng như các mức kiểm thử khác, kiểm thử đơn vị cũng đòi hỏi phải chuẩn bị trước các ca kiểm thử hay trường hợp kiểm thử (test case) hoặc kịch bản (test script), trong đó chỉ định rõ dữ liệu vào, các bước thực hiện và dữ liệu mong muốn sẽ xuất ra. Các test case và test script được giữ lại để sử dụng sau này.

### Kiểm thử tích hợp (Integration Test)

Kiểm thử tích hợp kết hợp các thành phần của một ứng dụng và kiểm thử như một ứng dụng đã hoàn thành. Trong khi kiểm thử đơn vị kiểm tra các thành phần và Unit riêng lẻ thì kiểm thử tích hợp kết hợp chúng lại với nhau và kiểm tra sự giao tiếp giữa chúng.

Kiểm thử tích hợp có hai mục tiêu chính là:

1. Phát hiện lỗi giao tiếp xảy ra giữa các Unit
2. Tích hợp các Unit đơn lẻ thành các hệ thống con (gọi là subsystem) và cuối cùng là nguyên hệ thống hoàn chỉnh chuẩn bị cho kiểm thử ở mức hệ thống (system test).

Có 4 loại kiểm thử trong kiểm thử tích hợp như sau:

1. Kiểm thử cấu trúc (Structure test): Kiểm thử nhằm bảo đảm các thành phần bên trong của một chương trình chạy đúng, chú trọng đến hoạt động của các thành phần cấu trúc nội tại của chương trình, chẳng hạn các lệnh và nhánh bên trong.
2. Kiểm thử chức năng (Functional test): Kiểm thử chỉ chú trọng đến chức năng của chương trình, không quan tâm đến cấu trúc bên trong, chỉ khảo sát chức năng của chương trình theo yêu cầu kỹ thuật.
3. Kiểm thử hiệu năng (Performance test): Kiểm thử việc vận hành của hệ thống.
4. Kiểm thử khả năng chịu tải (Stress test): Kiểm thử các giới hạn của hệ thống.

### Kiểm thử hệ thống (System Test)

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

Kiểm thử hệ thống kiểm tra cả các hành vi chức năng của phần mềm lẫn các yêu cầu về chất lượng như độ tin cậy, tính tiện lợi khi sử dụng, hiệu năng và bảo mật.

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

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

Sau khi hoàn thành kiểm thử tích hợp, một hệ thống phần mềm đã được hình thành cùng với các thành phần đã được kiểm tra đầy đủ. Tại thời điểm này, lập trình viên hoặc kiểm thử viên (Tester) bắt đầu kiểm thử phần mềm như một hệ thống hoàn chỉnh. Việc lập kế hoạch cho kiểm thử hệ thống nên bắt đầu từ giai đoạn hình thành và phân tích các yêu cầu.

Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan, kiểm thử hệ thống được thực hiện bởi một nhóm kiểm tra viên hoàn toàn độc lập với nhóm phát triển dự án để đảm bảo tính chính xác và khách quan.

Kiểm thử hệ thống thường có các loại kiểm thử sau:

1. Kiểm thử chức năng (Functional test): Bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
2. Kiểm thử khả năng vận hành (Performance test): Bảo đảm tối ưu việc phân bổ tài nguyên hệ thống (ví dụ bộ nhớ) nhằm đạt các chỉ tiêu như thời gian xử lý hay đáp ứng câu truy vấn, ....
3. Kiểm thử khả năng chịu tải (Stress test hay Load test): Bảo đảm hệ thống vận hành đúng dưới áp lực cao (ví dụ nhiều người truy xuất cùng lúc). Stress test tập trung vào các trạng thái tới hạn, các "điểm chết", các tình huống bất thường như đang giao dịch thì ngắt kết nối (xuất hiện nhiều trong test thiết bị như POS, ATM), ....
4. Kiểm thử cấu hình (Configuration test): Đảm bảo hệ thống hoạt động tương thích với các loại phần cứng khác nhau.
5. Kiểm thử khả năng bảo mật (Security test): Bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.
6. Kiểm thử khả năng phục hồi (Recovery test): Bảo đảm hệ thống có khả năng khôi phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến.

### Kiểm thử chấp nhận sản phẩm (Acceptance Test)

Mục đích của kiểm thử chấp nhận là kiểm thử khả năng chấp nhận cuối cùng để chắc chắn rằng sản phẩm là phù hợp và thỏa mãn các yêu cầu của khách hàng và khách hàng chấp nhận sản phẩm.

Trong giai đoạn kiểm thử chấp nhận thì người kiểm tra là khách hàng. Khách hàng sẽ đánh giá phần mềm với mong đợi theo những thao tác sử dụng quen thuộc của họ. Việc kiểm tra ở giai đoạn này có ý nghĩa hết sức quan trọng tránh cho việc hiểu sai yêu cầu cũng như sự mong đợi của khách hàng.

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

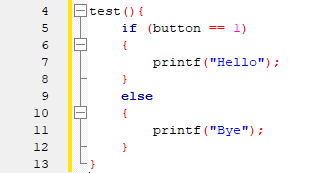
## Các kỹ thuật kiểm thử phần mềm

Có hai phương pháp kiểm thử chính là: Kiểm thử tĩnh và kiểm thử động.

### Kiểm thử tĩnh – Static testing

Kiểm thử tĩnh là loại kiểm thử trong đó chương trình sẽ không được thực hiện hay không thực thi code. Chủ yếu là kiểm tra cú pháp của code hoặc review code (kiểm tra xem code có được viết theo đúng tiêu chuẩn code.

VD: Từ yêu cầu từ phía bên trên thì các coder của chúng ta sẽ phải viết code để làm sao đáp ứng được yêu cầu của người dùng là ấn số 1 thì in ra “Hello” và các trường hợp còn lại thì sẽ in ra “Bye”



Hình 1.2. Ví dụ về kiểm thử tĩnh

Ở đây kiểm thử tĩnh sẽ kiểm tra đoạn code mà các coder viết có đúng và có đáp ứng được yêu cầu được đặt ra hay không, hoặc có bao phủ được một số các trường hợp cơ bản cần phải có hay không. Kiểm thử tĩnh sẽ kiểm tra đến từng dòng code, kiểm tra tính logic cũng như việc thực thi yêu cầu đặt ra.

Đây là loại kiểm thử được thực hiện bởi những người lập trình, làm việc một cách độc lập. Nó có thể được thực hiện bằng tay hay một bộ công cụ.

Lỗi được phát hiện ở giai đoạn phát triển này là ít tốn kém để sửa chữa hơn so với lỗi phát hiện được ở các giai đoạn sau này trong các quy trình phát triển phần mềm.

#### Các kỹ thuật sử dụng trong kiểm thử tĩnh

Inspection: Ở đây mục đích chính là tìm ra các khiếm khuyết. Việc kiểm tra được thực hiện bởi người kiểm duyệt. Đây là loại đánh giá thông thường có một danh sách kiểm tra được chuẩn bị để kiểm tra xem tài lài liệu công việc hoàn thành tới đâu.

Walk-through: Trong loại kỹ thuật này, Leader mở một cuộc họp để giải thích sản phẩm. Những người tham gia có thể đặt ra những câu hỏi nếu chưa hiểu và ghi chú lại, phục vụ cho việc hoàn thành công việc.

Technical reviews: Trong loại kiểm tra này, kiểm tra về kỹ thuật sẽ được kiểm tra 1 vòng. Việc này tiến hành để kiểm tra xem code được thực hiện theo đúng các thông số kỹ thuật và tiêu chuẩn hay không. Nói chung các kế hoạch kiểm tra, chiến lược kiểm thử và các tập lệnh kiểm tra được xem xét ở đây.

Informal reviews: Kỹ thuật kiểm tra tĩnh trong đó tài liệu được xem xét, nhận xét một cách không chính thức và Đưa ra các ý kiến không chính thức.

### Kiểm thử động – Dynamic testing

Cũng như ví dụ trên thì kiểm thử tĩnh tập trung vào code còn kiểm thử động sẽ thực hiện dựng code đó lên rồi thực thi, việc kiểm tra trên môi trường và thiết bị tương tự như môi trường thực tế mà phần mềm hoặc hệ thống phần mềm được sử dụng. Ta sẽ thực hiện kiểm tra hoạt động của hệ thống đúng theo luồng hoạt động và mô tả của yêu cầu bài toán đó là ấn số 1 xem có in ra chữ “Hello” và các nút còn lại in ra “Bye”. Đảm bảo việc hoạt động của chương trình đáp ứng đúng như yêu cầu đưa ra là được. Với việc kiểm thử động chúng ta có thể quan sát được các hành vi chức năng của phần mềm, giám sát hệ thống bộ nhớ, thời gian phản hồi của CPU, hiệu suất của hệ thống.

#### Các kỹ thuật kiểm thử động

Kiểm thử động bao gồm 2 kỹ thuật chính: Kiểm thử hộp trắng và kiểm thử hộp đen

## Kiểm thử hộp đen

### Định nghĩa

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

Kiểm thử hộp đen thực hiện các trường hợp kiểm thử để cố gắng tìm ra các loại lỗi sau:

* Thiếu hoặc sai chức năng
* Lỗi cấu trúc dữ liệu hay dữ liệu bên ngoài
* Lỗi giao diện
* Lỗi bắt đầu hay kết thúc chương trình
* Lỗi thực thi chương trình

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 người kiểm thử xây dựng các nhóm giá trị đầu vào sẽ thực thi đầy đủ tất cả các yêu cầu chức năng của chương trình.

### Các kỹ thuật kiểm thử hộp đen

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

Phương pháp kiểm thử hộp đen bao gồm 2 kỹ thuật chính:

1. Phân vùng tương đương: Chia (partition) đầu vào thành những nhóm tương đương nhau (equivalence). Nếu một giá trị trong nhóm hoạt động đúng thì tất cả các giá trị trong nhóm đó cũng hoạt động đúng và ngược lại. Giảm đáng kể số lượng test case cần phải thiết kế vì với mỗi lớp tương đương ta chỉ cần test trên các phần tử đại diện.
2. Phân tích giá trị biên: Đây là phương pháp test mà chúng ta sẽ test tất cả các giá trị ở vùng biên của dữ liệu vào và dữ liệu ra. Chúng ta sẽ tập trung vào các giá trị biên chứ không test toàn bộ dữ liệu.

Ngoài ra còn có một số kỹ thuật khác có nhiều ưu điểm:

* Kỹ thuật dùng các bảng quyết định (Decision Tables)
* Kỹ thuật kiểm thử các bộ n thần kỳ (Pairwise)
* Kỹ thuật dùng bảng chuyển trạng thái (State Transition)
* Kỹ thật phân tích vùng miền (Domain Analysis)
* Kỹ thuật dựa trên đặc tả Use Case (Use case)
* Kỹ thuật dùng lược đồ quan hệ nhân quả (Cause-Effect Diagram)

### Ưu, nhược điểm của kiểm thử hộp đen

1. Ưu điểm:

* Thích hợp trong việc kiểm tra từng phân đoạn lớn các mã lệnh, chức năng lớn.
* Người thử nghiệm không cần hiểu biết về mã lệnh được viết trong chương trình.
* Tách biệt giữa quan điểm của người sử dụng và người phát triển phần mềm.

1. Nhược điểm:

* Độ bao phủ hạn chế vì chỉ có một phần nhỏ trong số các kịch bản thử nghiệm được thực hiện.
* Kiểm tra không hiệu quả do người thử nghiệm không hiểu biết gì về cấu trúc bên trong phần mềm.
* Tester có hạn chế về hiểu biết về ứng dụng.

## Giới thiệu chung về kiểm thử hộp trắng

### Kiểm thử hộp trắng

Kiểm thử hộp trắng (White box testing) là chiến lược kiểm thử khác, trái ngược hoàn toàn với kiểm thử hộp đen. 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 bảo đảm rằng tất cả các câu lệnh và điều kiện sẽ được thực hiện ít nhất một lần. 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.

Kiểm thử hộp đen và kiểm thử hộp trắng không thể thay thế cho nhau mà chúng cần được sử dụng kết hợp với nhau trong một quy trình kiểm thử thống nhất nhằm đảm bảo chất lượng phần mềm.

Tuy nhiên, để áp dụng các phương pháp kiểm thử hộp trắng, người kiểm thử không chỉ cần hiểu rõ giải thuật mà còn cần có các kỹ năng và kiến thức tốt về ngôn ngữ lập trình được dùng để phát triển phần mềm, nhằm hiểu rõ mã nguồn của chương trình/đơn vị phần mềm cần kiểm thử.

Do vậy, việc áp dụng các phương pháp kiểm thử hộp trắng thường tốn thời gian và công sức nhất là khi chương trình hoặc thành phần phần mềm có kích thước lớn. Vì lý do này, các phương pháp kiểm thử hộp trắng chủ yếu được sử dụng cho kiểm thử đơn vị.

### Các kỹ thuật kiểm thử hộp trắng

Hai phương pháp được sử dụng trong kiểm thử hộp trắng:

1. Kiểm thử dòng điều khiển (control flow testing): Phương pháp kiểm thử dòng điều khiển tập trung kiểm thử tính đúng đắn của các giải thuật sử dụng trong các chương trình/đơn vị phần mềm.
2. Kiểm thử dòng dữ liệu (data flow testing): Phương pháp kiểm thử dòng dữ liệu tập trung kiểm thử tính đúng đắn của việc sử dụng các biến dữ liệu sử dụng trong chương trình/đơn vị phần mềm.

### Đặc điểm

1. Là chiến lược kiểm thử thành dựa vào giải thuật, cấu trúc bên trong chức năng của thành phần phần mềm tương ứng.
2. Người kiểm thử phải có kiến thức nhất định về việc mã hoá, cấu trúc bên trong của chức năng, biết lâp trình phần mềm.
3. Việc kiểm thử được tiến hành dựa vào việc kiểm xem giải thuật, mã lệnh đã làm có đúng không.
4. Mức test này thường yêu cầu các tester phải viết test case đầy đủ các nhánh trong code; khi test, sẽ set điều kiện và data để chạy vào đủ tất cả các nhánh trong giải thuật, đảm bảo thực hiện đầy đủ.

## Khi nào thì sử dụng kiểm thử hộp trắng

Kiểm thử hộp trắng sử dụng các chiến lược cụ thể và sử dụng mã nguồn của chương trình hoặc đơn vị phần mềm cần kiểm thử nhằm kiểm tra xem chương trình hoặc đơn vị phần mềm có thực hiện đúng so với thiết kế và đặc tả hay không. Trong khi các phương pháp kiểm thử hộp đen hay kiểm thử chức năng chỉ cho phép phát hiện các lỗi hay khiếm khuyết có thể quan sát được, kiểm thử hộp trắng cho phép phát hiện các lỗi hay khiếm khuyết tiềm ẩn bên trong chương trình hoặc đơn vị phần mềm. Các lỗi này thường khó phát hiện bởi các phương pháp kiểm thử hộp đen. Khác với các phương pháp kiểm thử hộp đen nơi mà các ca kiểm thử được sinh ra từ đặc tả của hệ thống, các ca kiểm thử trong các phương pháp kiểm thử hộp trắng được sinh ra từ mã nguồn. Kiểm thử hộp đen và kiểm thử hộp trắng không thể thay thế cho nhau mà chúng cần được sử dụng kết hợp với nhau trong một quy trình kiểm thử thống nhất nhằm đảm bảo chất lượng phần mềm.

### Ưu điểm và nhược điểm của kiểm thử hộp trắng

1. Ưu điểm

* Thích hợp trong việc tìm kiếm lỗi và các vẫn đề trong mã lệnh
* Biết được yêu cầu bên trong của phần mềm, kiểm tra sẽ sát hơn
* Cho phép tìm kiếm các lỗi bên trong
* Giúp tối ưu việc mã hóa
* Các lập trình viên có thể tự kiểm tra
* Do yêu cầu kiếm thức cấu trúc bên trong của phần mềm, nên việc kiểm soát lỗi tối đa nhất.

1. Nhược điểm

* Không thể tìm thấy tính năng chưa thực hiện hoặc bỏ sót
* Đòi hỏi hiểu sâu về cấu trúc bên trong của phần mềm được thử nghiệm
* Yêu cầu truy xuất mã lệnh bên trong chương trình

### Sự khác biệt giữa kiểm thử hộp đen và kiểm thử hộp trắng

#### Các tiêu chuẩn

|  |  |  |
| --- | --- | --- |
| **Tiêu chuẩn** | **Kiểm thử hộp đen** | **Kiểm thử hộp trắng** |
| **1. Định nghĩa** | Kiểm tra hộp đen là phương pháp thử nghiệm phần mềm được sử dụng để kiểm tra các phần mềm mà không quan tâm đến cấu trúc bên trong của chương trình. | Kiểm tra hộp trắng là phương pháp kiểm thử phần mềm, sử dụng để kiểm tra phần mềm mà yêu cầu phải biết cấu trúc bên trong của chương trình. |
| **2. Trách nhiệm** | Thử nghiệm được thực hiện bên ngoài, không liên quan đến nhà phát triển phần mềm. | Thông thường, các thử nghiệm được thực hiện bởi nhà phát triển phần mềm. |
| **3. Cấp độ test sử dụng** | Thử nghiệm áp dụng ở cấp độ cao như: kiểm tra hệ thống (System test), kiểm tra chấp nhận (Acceptance test) | Thử nghiệm được áp dụng ở mức độ thấp hơn như thử nghiệm đơn vị (Unit Test), thử nghiệm hội nhập (Integration test) |
| **4. Biết lập trình** | Không yêu cầu hiểu biết về Lập trình | Yêu cầu hiểu biết nhất định về lập trình. |
| **5. Biết việc thực hiện chương trình** | Không yêu cầu hiểu về cấu trúc bên trong chức năng, và không cẩn hiểu làm thế nào để có được chức năng đó | Yêu cầu hiểu cấu trúc bên trong chức năng được thực hiện như nào. |
| **6. Cơ sở tạo Test Cases** | Kiểm tra hộp đen được bắt đầu dựa trên tài liệu yêu cầu kỹ thuật | Kiểm tra hộp trắng được bắt đầu dựa trên các tài liệu thiết kế chi tiết |

Bảng 1.1. So sánh các tiêu chuẩn giữa kiểm thử hộp đen và kiểm thử hộp trắng

#### Ưu điểm và nhược điểm

|  |  |  |
| --- | --- | --- |
| **Nội dung** | **Kiểm thử hộp đen** | **Kiểm thử hộp trắng** |
| **1. Ưu điểm** | - Thích hợp trong việc kiểm tra từng phân đoạn lớn các mã lệnh, chức năng lớn  - Người thử nghiệm không cần hiểu biết về mã lệnh được viết trong chương trình  - Tách biệt giữa quan điểm của người sử dụng và người phát triển phần mềm | - Thích hợp trong việc tìm kiếm lỗi và các vấn đề trong mã lệnh  - Biết được yêu cầu bên trong của phần mềm, kiểm tra sẽ sát hơn  - Cho phép tìm kiếm các lỗi ẩn bên trong  - Các lập trình viên có thể tự kiểm tra  - Giúp tối ưu việc mã hoá  - Do yêu cầu kiến thức cấu trúc bên trong của phần mềm, nên việc kiểm soát lỗi tối đa nhất |
| **2. Nhược điểm** | - Độ bao phủ hạn chế vì chỉ có một phần nhỏ trong số các kịch bản thử nghiệm được thực hiện  - Kiểm tra không hiệu quả do người thử nghiệm không hiểu biết gì về cấu trúc bên trong phần mềm.  - Tester có hạn chế về hiểu biết về ứng dụng | - Không thể tìm thấy tính năng chưa thực hiện hoặc bỏ sót  - Đòi hỏi hiểu sâu về cấu trúc bên trong của phần mềm được thử nghiệm  - Yêu cầu truy xuất mã lệnh bên trong chương trình |

Bảng 1.2. So sánh ưu và nhược điểm giữa kiểm thử hộp đen và kiểm thử hộp trắng

## Kết chương

Trong chương 1 đã giới thiệu tổng quát về kiểm thử phần mềm, các kỹ thuật kiểm thử phần mềm. Tìm hiểu qua hai kỹ thuật kiểm thử chính là kiểm thử hộp đen và kiểm thử hộp trắng và so sánh được những điểm khác nhau cơ bản giữa kĩ thuật kiểm thử hộp đen và hộp trắng. Kiểm thử hộp trắng xem xét chương trình ở mức độ chi tiết và phù hợp khi kiểm tra các mô đun nhỏ. Kiểm thử hộp đen chú trọng vào việc kiểm tra các quan hệ vào ra và những chức năng giao diện bên ngoài, nó thích hợp hơn cho các hệ thống phần mềm lớn hay các thành phần quan trọng của chúng. Nhưng chỉ sử dụng kiểm thử hộp đen là chưa đủ. Bởi vì, kiểm thử chức năng chỉ dựa trên đặc tả của môđun nên không thể kiểm thử được các trường hợp không được khai báo trong đặc tả. Ngoài ra, do không phân tích mã nguồn nên không thể biết được môđun nào của chương trình đã hay chưa được kiểm thử, khi đó phải kiểm thử lại hay bỏ qua những lỗi tiềm ẩn trong gói phần mềm.

Phương pháp kiểm thử hộp trắng và kiểm thử hộp đen là hai phương pháp cơ bản có vai trò rất quan trọng trong quá trình phát triển phần mềm, nếu chúng ta biết kết hợp chúng để bổ sung khiếm khuyết lẫn nhau.

Chương sau chúng ta sẽ tìm hiểu chi tiết về kĩ thuật kiểm thử luồng điều khiển trong kiểm thử hộp trắng.

# KIỂM THỬ DÒNG ĐIỀU KHIỂN TRONG KIỂM THỬ HỘP TRẮNG

## Giới thiệu

Phương pháp kiểm thử dòng điều khiển tập trung kiểm thử tính đúng đắn của các giải thuật sử dụng trong các chương trình hoặc đơn vị phần mềm. Kiểm thử dòng điều khiển sử dụng cấu trúc điều khiển của một chương trình để phát triển các trường hợp thử nghiệm cho chương trình. Các trường hợp thử nghiệm được phát triển đầy đủ bao gồm toàn bộ cấu trúc điều khiển của chương trình. Cấu trúc điều khiển của chương trình có thể được biểu diễn bằng đồ thị dòng khiển (control flow graph) của chương trình.

## Đường thi hành (Execution path)

Là một kịch bản thi hành đơn vị phần mềm tương ứng, cụ thể nó là danh sách có thứ tự các lệnh được thi hành ứng với 1 lần chạy cụ thể của đơn vị phần mềm, bắt đầu từ điểm nhập của đơn vị phần mềm đến điểm kết thúc của đơn vị phần mềm. Mỗi thành phần phần mềm có từ 1 đến n (có thể rất lớn) đường thi hành khác nhau. Mục tiêu của phương pháp kiểm thử luồng điều khiển là đảm bảo mọi đường thi hành của đơn vị phần mềm cần kiểm thử đều chạy đúng. Rất tiếc trong thực tế, công sức và thời gian để đạt mục tiêu trên đây là rất lớn, ngay cả trên những đơn vị phần mềm nhỏ.

Ví dụ đoạn code sau:

for(int i=1;i<=1000;i++){

for (int j=1;j<=1000;j++){

for(itn k=1;k<=1000;k++){

doSomething(i,j,k)

}

}

}

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

Ví dụ đoạn code gồm 32 lệnh if else độc lập sau.

if (c1) s11 else s12;

if (c2) s21 else s22;

if (c3) s31 else s32;

…

if (c32) s321 else s322;

Có = 4 tỉ đường thi hành khác nhau. Mà cho dù có kiểm thử hết được toàn bộ các đường thi hành thì vẫn không thể phát hiện những đường thi hành cần có nhưng chưa được thực hiện:

if(a > 0){

printf(“Đây là số dương”);

}

if(a == 0){

printf(“a bằng 0”)

}

// Ở code này giả sử thiếu trường hợp a<0 nhưng khó có thể phát hiện được

Thiếu xử lý trường hợp a < 0

if(a < 0){

printf(“Đây là số âm”);

}

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

Ví dụ:

int phanSo(int a,int b){

return a/b;

}

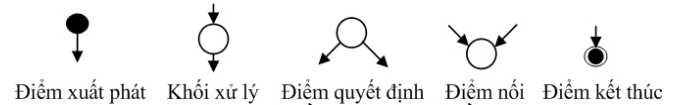
Khi kiểm tra, ta chọn b < 0 hoặc b > 0 thì chạy đúng, nhưng khi dùng thật trong trường hợp b = 0 thì hàm phanSo sẽ bị lỗi chia không.

## Đồ thị dòng điều khiển

Phương pháp kiểm thử dòng điều khiển dựa trên khái niện đồ thị dòng điều khiển (Control flow graph). Đồ thị này được xây dựng từ mã nguồn của chương trình hoặc đơn vị chương trình.

Đồ thị dòng điều khiển là một đồ thị có hướng gồm các đỉnh tương ứng với các câu lệnh hoặc nhóm các câu lệnh và các cạnh là các dòng điều khiển giữa các câu lệnh hoặc nhóm câu lệnh. Nếu i và j là các đỉnh của đồ thị dòng điều khiển thì tồn tại một cạnh từ i đến j nếu lệnh tương ứng với j có thể được thực hiện ngay sau câu lệnh tương ứng với i.

Xây dựng một đồ thị dòng điều khiển từ một chương trình hoặc một đơn vị chương trình khá đơn giản.

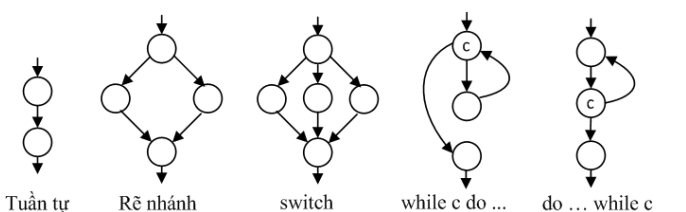


Hình 2.3. Các thành phần cơ bản của đồ thị chương trình

Hình 2.1 mô tả các thành phần cơ bản của đồ thị dòng điều khiển bao gồm:

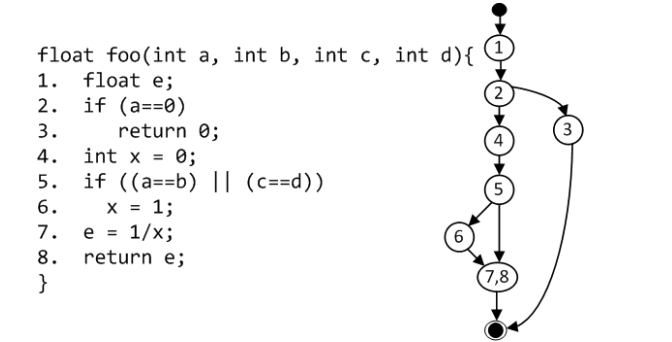
* Điểm bắt đầu của đơn vị chương trình
* Khối xử lý chứa các câu lệnh khai báo hoặc tính toán
* Điểm quyết định ứng với các câu lệnh điều khiển trong các khối lệnh rẽ nhánh hoặc lặp
* Điểm nối ứng với các câu lệnh ngay sau các lệnh rẽ nhánh
* Điểm kết thúc ứng với điểm kết thúc của đơn vị chương trình

Các cấu trúc điều khiển phổ biến cùa chương trình được mô tả trong hình 2.2. Chúng ta sẽ sử dụng các thành phần cơ bản và các cấu trúc phổ biến này để dễ dàng xây dựng đồ thị dòng điều khiển cho mọi đơn vị chương trình viết bằng ngôn ngữ lập trình.



Hình 2.4. Các cấu trúc điều khiển phổ biến của chương trình

Tiếp theo là cách dựng đồ thị dòng điều khiển cho đơn vị chương trình, có mã nguồn bằng ngôn ngữ C như hình 2.3 dưới đây:



Hình 2.5. Ví dụ về đồ thị dòng điều khiển

Chúng ta đánh số các dòng lệnh của đơn vị chương trình và lấy số này làm đỉnh của đồ thị.

* Điểm xuất phát của đơn vị chương trình ứng với câu lệnh khai báo hàm **foo**
* Đỉnh 1 ứng với câu lệnh khai báo biến **e**
* Các đỉnh 2 và 3 ứng với câu lệnh **if**
* Đỉnh 4 ứng với câu lệnh khai báo biến **x**
* Đỉnh 5 và 6 ứng với câu lệnh **if**
* Đỉnh 7 và 8 đại diện cho hai câu lệnh 7 và 8. Trong trường hợp này, chúng ta không tách riêng thành hai đỉnh vì đây là hai câu lệnh tuần tự nên chúng ta ghép chúng thành một đỉnh nhằm tối thiểu số đỉnh của đồ thị dòng điều khiển. Với cách làm này, chúng ta xây dựng được đồ thị dòng điều khiển với số đỉnh nhỏ nhất.

=> Chúng ta sẽ sử dụng đồ thị này để phân tích và sinh ra các ca kiểm thử nên đồ thị càng ít đỉnh thì độ phức tạp của thuật toán phân tích càng nhỏ.

## Các độ đo kiểm thử

Kiểm thử chức năng (kiểm thử hộp đen) có hạn chế là chúng ta không biết có thừa hay thiếu các ca kiểm thử hay không so với chương trình cài đặt và thiếu thừa ở mức độ nào. Độ đo kiểm thử là một công cụ giúp ta đo mức độ bao phủ chương trình của một tập ca kiểm thử cho trước.

Mức độ bao phủ của một bộ kiểm thử (tập các ca kiểm thử) được đo bằng tỷ lệ các thành phần thực sự được kiểm thử so với tổng thể sau khi đã thực hiện các ca kiểm thử.

Thành phần liên quan có thể là:

* Câu lệnh
* Điểm quyết định
* Điều kiện con
* Đường thi hành hay là sự kết hợp của chúng

Độ bao phủ càng lớn thì độ tin cậy của bộ kiểm thử càng cao. Độ đo này giúp chúng ta kiểm soát và quản lý quá trình kiểm thử tốt hơn. Mục tiêu của chúng ta là kiểm thử với số ca kiểm thử tối thiểu nhưng đạt được độ bảo phủ tối đa. Có rất nhiều độ đo kiểm thử đang được sử dụng hiện nay, dưới đây là ba độ đo kiểm thử đang được sử dụng phổ biến nhất trong thực tế hiện nay.

### Độ đo kiểm thử cấp 1 (C1)

Mỗi câu lệnh được thực hiện ít nhất một lần sau khi chạy các ca kiểm thử (test cases). Ví dụ: với hàm **foo** có mã nguồn như trong Hình 2.3, ta chỉ cần hai ca kiểm thử như bảng 2.1 là đạt 100% độ phủ cho độ đo cấp 1 (C1) với EO (expected output) là giá trị đầu ra mong đợi và RO (real output) là giá trị đầu ra thực tế (giá trị này sẽ được điền khi thực hiện ca kiểm thử).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Inputs | EO | RO | Note |
| Testcase 1 (tc1) | 0, 1, 2, 3 | 0 |  |  |
| Testcase 2 (tc2) | 1, 1, 2, 3 | 1 |  |  |

Bảng 2.3. Các ca kiểm thử cho độ đo C1 của hàm **foo**

### Độ đo kiểm thử cấp 2 (C2)

Các điểm quyết định trong đồ thị dòng điều khiển của đơn vị kiểm thử được thực hiện ít nhất 1 lần cả hai nhánh đúng và sai. Ví dụ bảng 2.2 mô tả các trường hợp cần kiểm thử để đạt được 100% độ phủ của độ đo cấp 2 ứng với hàm foo được mô tả trong hình 2.3. Như vậy, với hai ca kiểm thử trong độ đo kiểm thử cấp 1 (tc1 và tc2), ta chỉ kiểm thử được ¾=75% ứng với độ đo kiểm thử cấp 2(C2).

|  |  |  |  |
| --- | --- | --- | --- |
| Điểm quyết định | Điều kiện tương ứng | Đúng | Sai |
| 2 | a==0 | tc1 | tc2 |
| 5 | (a==b) || (c==d) | tc2 | ? |

Bảng 2.4. Các trường hợp cần kiểm thử của độ đo C2 với hàm **foo**

Chúng ta cần một ca kiểm thử nữa ứng với trường hợp sai của điều kiện (a==b) || (c==d) nhằm đạt được 100% độ phủ của độ đo C2. Bảng 2.3 mô tả các ca kiểm thử cho mục đích này.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Inputs | EO | RO | Note |
| tc1 | 0, 1, 2, 3 | 0 |  |  |
| tc2 | 1, 1, 2, 3 | 1 |  |  |
| tc3 | 1, 2, 1, 2 | Lỗi chia cho 0 |  |  |

Bảng 2.5. Các ca cần kiểm thử cho độ đo C2 với hàm **foo**

### Độ đô kiểm thử cấp 3 (C3)

Với các điều kiện phức tạp (chứa nhiều điều kiện con cơ bản), việc chỉ quan tâm đến giá trị đúng sai là không đủ để kiểm tra tính đúng đắn của chương trình ứng với điều kiện phức tạp này.

* VD: Nếu một điều kiện phức tạp gồm hai điều kiện con cơ bản, chúng ta có bốn trường hợp cần kiểm thử chứ không phải hai trường hợp đúng sai như độ đo C2.
* Với các đơn vị chương trình có yêu cầu cao về tính đúng đắn, việc tuân thủ độ đo C3 là hết sức cần thiết. Điều kiện để đảm bảo độ đo này là các điều kiện con phụ thuộc các điều kiện phức tạp tương ứng với các điểm quyết định trong đồ thị dòng điều khiển của đơn vị cần kiểm thử đều được thực hiện ít nhất một lần cả hai nhánh đúng và sai.
* VD: Bảng 2.4 mô tả các trường hợp cần kiểm thử để đạt được 100% độ phủ của độ đo C3 tương ứng với hàm **foo** được mô tả trong hình 2.3.
* Như vậy, với ba ca kiểm thử trong độ đo kiểm thử cấp 2 (tc1, tc2, tc3), ta chỉ kiểm thử được 7/8 = 87,5% ứng với độ đo kiểm thử cấp 3.

|  |  |  |  |
| --- | --- | --- | --- |
| Điểm quyết định | Điều kiện tương ứng | Đúng | Sai |
| 2 | a== 0 | tc1 | tc2 |
| 5 | a==b | tc2 | tc3 |
| 5 | c==d | ? | tc2 |

Bảng 2.6. Các trường hợp cần kiểm thử của độ đo C3 với hàm **foo**

Chúng ta cần một ca kiểm thử nữa ứng với trường hợp sai của điều kiện con cơ bản (c==d) nhằm đạt 100% độ phủ của độ đo C3. Bảng 2.5 mô tả các ca kiểm thử cho mục đích này.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Input | EO | RO | Note |
| tc1 | 0,1,2,3 | 0 |  |  |
| tc2 | 1,1,2,3 | 1 |  |  |
| tc3 | 1,2,1,2 | Lỗi chia 0 |  |  |
| tc4 | 1,2,1,1 | 1 |  |  |

Bảng 2.7. Các ca cần kiểm thử của độ đo C3 với hàm **foo**

## Kiểm thử dựa trên độ đo

Kiểm thử dựa trên độ đo là phương pháp phân tích mã nguồn và thực hiện chương trình hoặc đơn vị chương trình sao cho thoả mãn một độ đo kiểm thử cho trước.

Hình 2.4 mô tả quy trình kiểm thử dựa trên độ đo cho các đơn vị chương trình. Với mỗi đơn vị chương trình, đồ thị dòng điều khiển ứng với các độ đo C1 và C2 là giống nhau trong khi chúng khác với đồ thị dòng điều khiển với độ đo C3. Sở dĩ có sự khác biệt này là bởi vì đồ thị dòng điều khiển ứng với độ đo C3 sẽ bao gồm nhiều đỉnh điều khiển ứng với từng điều kiện con cơ bản. Với mỗi đơn vị chương trình và mỗi độ đo kiểm thử, chúng ta tiến hành xây dựng đồ thị dòng điều khiển tương ứng. Các đường đi của chương trình (xuất phát từ điểm bắt đầu, đi qua các đỉnh của đồ thị và kết thúc ở điểm cuối) được xác định sao cho khi chúng được thực hiện thì độ đo kiểm thử tương ứng được thỏa mãn.

Dựa trên ý tưởng của T.J.McCabe, số đường đi chương trình ứng với đồ thị dòng điều khiển của nó được tính bằng một trong các phương pháp sau:

* Số cạnh – số đỉnh + 2
* Số đỉnh quyết định + 1

Đơn vị chương trình

Độ đo kiểm thử (C1, C2 hoặc C3)

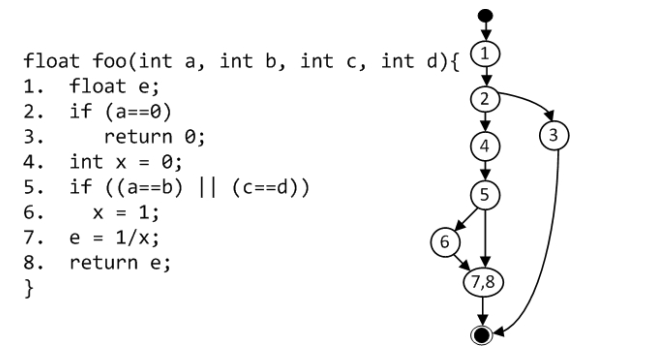
Hình 2.6. Quy trình kiểm thử đơn vị chương trình dựa trên độ đo.

Sau khi có được các đường đi của đơn vị chương trình cần kiểm thử, với mỗi đường đi, chúng ta sẽ sinh một ca kiểm thử tương ứng. Có rất nhiều kỹ thuật để sinh các ca kiểm thử từ các đường đi này.

Để sinh một bộ dữ liệu đầu vào cho ca kiểm thử tương ứng với mỗi đường đi, một kỹ thuật phổ biến là chọn ngẫu nhiên một bộ dữ liệu đầu vào sao cho nó thỏa mãn tất cả các điều kiện trên đường đi này. Nếu chúng ta tìm được một bộ dữ liệu đầu vào như vậy, đường đi tương ứng được gọi là đường đi thực thi được. Ngược lại, đường đi này ứng với các câu lệnh không bao giờ xảy ra hoặc nó là một phần của một đường đi khác liên quan đến vòng lặp trong chương trình. Khi có được bộ giá trị đầu vào, chúng ta sẽ phân tích và xác định giá trị đầu ra mong muốn cho ca kiểm thử này. Cuối cùng, các ca kiểm thử được thực hiện trên đơn vị chương trình nhằm phát hiện các lỗi.

### Kiểm thử cho độ đo C1

Xét lại hàm **foo** có mã nguồn như hình 2.5. Chúng ta xây dựng đồ thị dòng điều khiển ứng với độ phủ C1 cho hàm này như hình 2.5.



Hình 2.7. Mã nguồn của hàm **foo** và đồ thị dòng điều khiển của nó

Để đạt được 100% độ phủ của độ đo C1, ta chỉ cần hai đường đi như sau để đảm bảo được tất cả các câu lệnh của hàm **foo** được kiểm thử ít nhất một lần.

Để kiểm tra việc đảm bảo độ đo C1, chúng ta cần kiểm tra tất cả các lệnh hoặc khối lệnh (1-8) đều được xuất hiện ít nhất một lần trong các đường đi này. Rõ ràng, hai đường đi này thỏa mãn điều kiện trên nên chúng ta đạt được 100% độ phủ C1:

1. 1; 2(F); 4; 5(T); 6; 7,8
2. 1; 2(T); 3

Với đường đi 1; 2(F); 4; 5(T); 6; 7,8 ta sẽ sinh ra một ca kiểm thử để nó được thực thi khi thực hiện ca kiểm thử này. Ý tưởng của việc sinh ca kiểm thử này là tìm một bộ giá trị đầu vào cho a, b, c và d sao cho điều kiện tương ứng với điểm quyết định 2 (a==0) là sai và điều kiện ứng với điểm quyết định 5 ((a==b) || (c==d)) là đúng. Giá trị đầu ra mong đợi (EO) của ca kiểm thử này là 1.

Tương tự, ta sẽ sinh ca kiểm thử ứng với đường đi 1; 2(T); 3 với đầu ra mong đợi là 0. Chúng ta sẽ tìm một bộ đầu vào sao cho điều kiện (a==0) là đúng

Bảng 2.6 là một ví dụ về hai ca kiểm thử được sinh ra bằng ý tưởng trên

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Path** | **Inputs** | **EO** | **RO** | **Note** |
| tc1 | 1; 2(F); 4; 5(T); 6; 7,8 | 2, 2, 3, 5 | 1 |  |  |
| tc2 | 1; 2(T); 3 | 0, 3, 2, 7 | 0 |  |  |

Bảng 2.8. Các ca kiểm thử cho độ đo C1 của hàm foo

### Kiểm thử cho độ đo C2

Như chúng ta đã biết, với mỗi đơn vị chương trình, đồ thị dòng điều khiển ứng với các độ đo C1 và C2 là giống nhau. Vì vậy, đồ thị dòng điều khiển ứng với độ đo C1 của hàm foo được mô tả ở Hình 2.5 cũng là đồ thị dòng điều khiển của hàm này ứng với độ đo C2. Tuy nhiên, để được 100% độ phủ của độ đo C2 chúng ta cần tối thiểu ba đường đi. Tại sao chúng ta biết được điều này? Như đã trình bày ở mục trước, chúng ta có hai cách để tính được con số này. Ví dụ, đồ thị dòng điều khiển của hàm foo có hai điểm quyết định là 2 và 5 nên chúng ta cần 2 + 1 = 3 đường đi để đạt được 100% độ phủ của độ đo C2. Các đường đi cần thiết được liệt kê như sau. Rõ ràng với ba đường đi này, cả hai nhánh đúng và sai của hai điểm quyết định 2 và 5 đều được kiểm tra.

1. 1; 2(F); 4; 5(T); 6; 7,8

2. 1; 2(T); 3

3. 1; 2(F); 4; 5(F); 7,8

Để sinh các ca kiểm thử ứng với các đường đi trên, chúng ta chỉ cần quan tâm đến đường đi (3) vì việc sinh các ca kiểm thử cho các đường đi (1) và (2) đã được trình bày ở mục kiểm thử cho độ đo C1. Với đường đi (3), ta chỉ cần chọn một bộ đầu vào sao cho điều kiện ứng với điểm quyết định 2 (a == 0) là sai và điều kiện ứng với điểm quyết định 5 ((a == b) || (c == d)) cũng là sai. Giá trị đầu ra mong đợi của đường đi này là lỗi chia cho 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test path | Inputs | EO | RO |
| TC1 | 1; 2(F); 4; 5(T); 6; 7,8 | 2,2,3, 5 | 1 |  |
| TC2 | 1; 2(T); 3 | 0,3,2,7 | 0 |  |
| TC3 | 1; 2(F); 4; 5(F); 7,8 | 2,3,4,5 | Lỗi chia không |  |

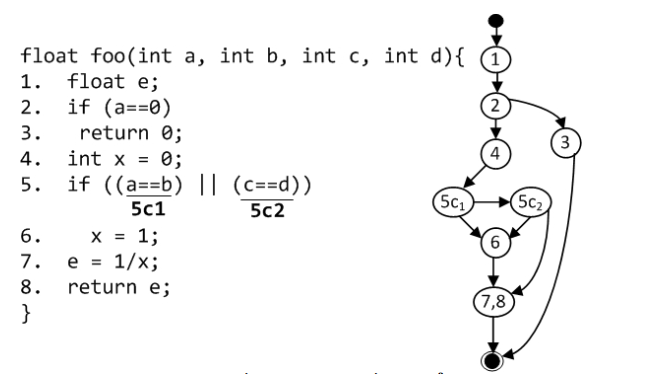
Bảng 2.9. Các ca kiểm thử cho độ đo C2 của hàm foo

Bảng 2.7 là một ví dụ về ba ca kiểm thử được sinh ra bằng ý tưởng trên ứng với các đường đi (1), (2), và (3). Độ đo C1 đảm bảo các câu lệnh được “đi qua” ít nhất một lần khi thực hiện tất cả các ca kiểm thử được sinh ra ứng với độ đo này. Đây là độ đo khá tốt và việc đảm bảo độ đo này trong thực tế cũng khá tốn kém. Tuy nhiên, qua ví dụ trên, chúng ta thấy rằng nếu chỉ sử dụng độ đo C1 với hai ca kiểm thử như trong bảng 2.6, lỗi chia cho không sẽ không được phát hiện. Chỉ khi kiểm tra cả hai nhánh đúng sai của tất cả các điểm quyết định (các lệnh điều khiển) thì lỗi này mới được phát hiện như ca kiểm thử tc3 trong bảng 2.7.

### Kiểm thử cho độ đo C3

Đồ thị dòng điều khiển ứng với độ đo C3 khác với đồ thị dòng điều khiển ứng với các độ đo C1 và C2

VD: Đồ thị dòng điều khiển của hàm **foo** ứng với độ đo C3 được xây dựng như hình 6.6. Với câu lệnh điều kiện 5, vì đây là điều kiện phức tạp nên ta phải tách thành hai điều kiện con cơ bản là (a==b) và (c==d) ứng với 2 điểm quyết định 5c1 và 5c2 trong đồ thị dòng điều khiển. Từ câu lệnh 4, nếu điều kiện con (a==b) đúng, ta không cần kiểm tra điều kiện con còn lại (vì điều kiện phức tạp là hoặc của hai điều kiện con cơ bản) và thực hiện câu lệnh 6. Nếu điều kiện con (a==b) là sai, ta cần tiến hành kiểm tra điều kiện con cơ bản còn lại (c==d). Nếu điều kiện này đúng, ta tiến hành câu lệnh 6. Ngược lại, chúng ta thực hiện các câu lệnh 7 và 8. Trong đồ thị này chúng ta gộp hai lệnh 7 và 8 trong một đỉnh (đỉnh (7,8)) vì đây là hai câu lệnh tuần tự. Mục đích của việc này là nhằm tối thiểu số đỉnh của đồ thị dòng điều khiển. Một đồ thị có số đỉnh cảng nhỏ thì chúng ta càng dễ dàng trong việc sinh các đường đi của chương trình và tránh các sai sót trong quá trình này.



Hình 2.8: Hàm **foo** và đồ thị dòng điều khiển ứng với độ đo C3

Đồ thị dòng điều khiển của hàm **foo** ứng với các độ đo C3 như hình 2.6 có 3 điểm quyết định là 2, 5c1 và 5c2 nên chúng ta cần 3 + 1 = 4 đường đi để được 100% độ phủ của độ đo C3. Các đường đi cần thiết được liệt kê như sau:

1. 1; 2(F); 4; 5c1(T); 6; 7,8
2. 1; 2(F); 4; 5c1(F); 5c2(T); 6; 7,8
3. 1; 2(F); 4; 5c1(F); 5c2(F); 7,8
4. 1; 2(T); 3

Tương tự như các phương pháp kiểm thử độ đo C1 và C2, chúng ta dễ dàng sinh các ca kiểm thử tương ứng cho các đường đi chương trình như đã mô tả trên:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Test Path** | **Inputs** | **EO** | **RO** | **Note** |
| tc1 | 1; 2(F); 4; 5c1(T); 6; 7,8 | 0, 2, 3, 5 | 0 |  |  |
| tc2 | 1; 2(F); 4; 5c1(F); 5c2(T); 6; 7,8 | 2, 2, 2, 7 | 1 |  |  |
| tc3 | 1; 2(F); 4; 5c1(F); 5c2(F); 7,8 | 2,3,4,5 | lỗi chia cho 0 |  |  |
| tc4 | 1; 2(T); 3 | 2,3,4,4 | 1 |  |  |

Bảng 2.10. Các ca kiểm thử cho độ đo C3 của hàm **foo**

Bảng 2.8 là một ví dụ về các ca kiểm thử cho hàm **foo** ứng với độ đo C3. Vì đồ thị dòng điều khiển của hàm **foo** ứng với độ đo C3 có ba (3) đỉnh điều khiển nên chúng ta cần tối thiểu (4) đường đi từ đỉnh đầu đến đỉnh cuối của đồ thị này (bảng 2.8). Ứng với mỗi đường đi, chúng ta cũng sẽ tìm một bộ đầu vào cho các tham số a, b, c và d sao cho các điều kiện ứng với mỗi đường đi thỏa mãn.

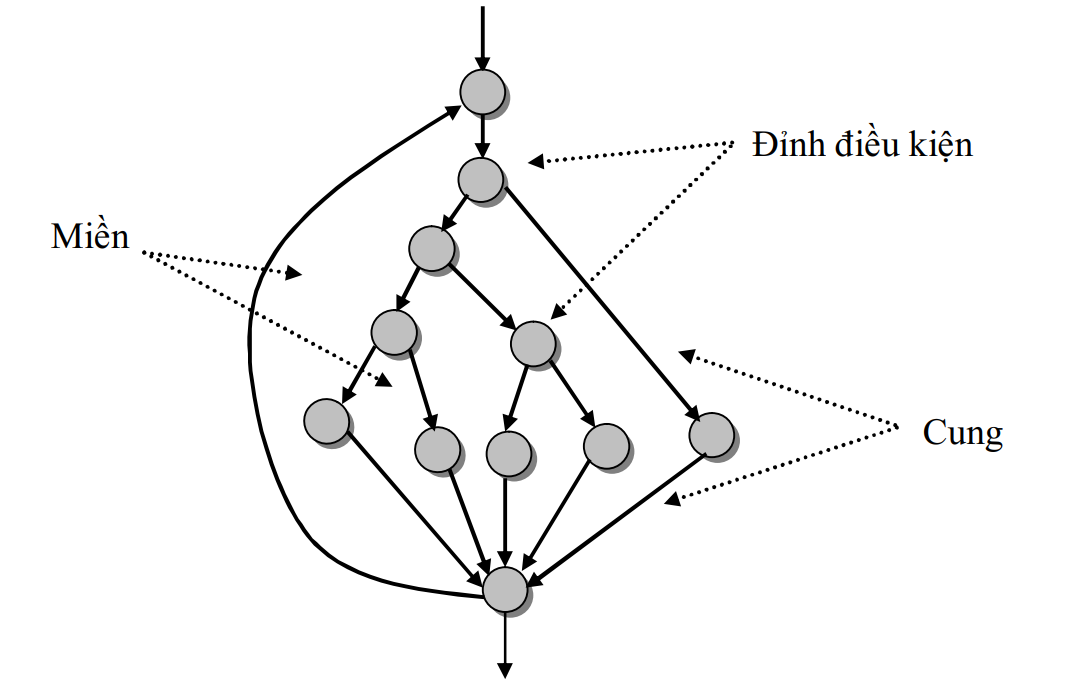
## Kiểm thử đường dẫn cơ bản

* Kiểm thử các đường cơ bản là một trong những phương cách kiểm nghiệm hộp trắng
* Bảo đảm số phép thử là ít nhất đủ để phát hiện các lỗi
* Tất cả các đường cơ bản được thử qua ít nhất một lần
* Thử các điều kiện rẽ nhánh ở cả 2 nhánh true và false
* Thử qua vòng lặp tại biên cũng như bên trong
* Thử qua cấu trúc dữ liệu để đảm bảo tính toàn vẹn của nó
* Kiểm thử đường cơ bản áp dụng cho những module có tính nghiêm ngặt

### Đồ thị luồng điều khiển

Cấu trúc của đồ thị luồng điều khiển bao gồm:

* Mỗi đỉnh (hình tròn) biểu thị một đoạn các câu lệnh thực hiện một cách tuần tự, có thể kết thúc bằng một lệnh rẽ nhánh.
* Mỗi cạnh (cung) biểu diễn dòng điều khiển nối hai nút với nhau.
* Phần được bao bởi các cung và các đỉnh gọi là miền



Hình 2.9. Đồ thị luồng điều khiển

### Độ phức tạp chu trình Cyclomatic Complexity V(G)

Độ phức tạp chu trình là một thước đo phần mềm, cung cấp các 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 chu trình 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.

Việc tính toán đồ phức tạp chu trình sẽ cho chúng ta biết có bao nhiêu đường dẫn cần tìm. Cho đồ thị luồng điều khiển G, khi đó ta có độ phức tạp Cyclomatic Complexity =V(G) của đồ thị dòng điều khiển được tính bởi 1 trong các công thức sau :

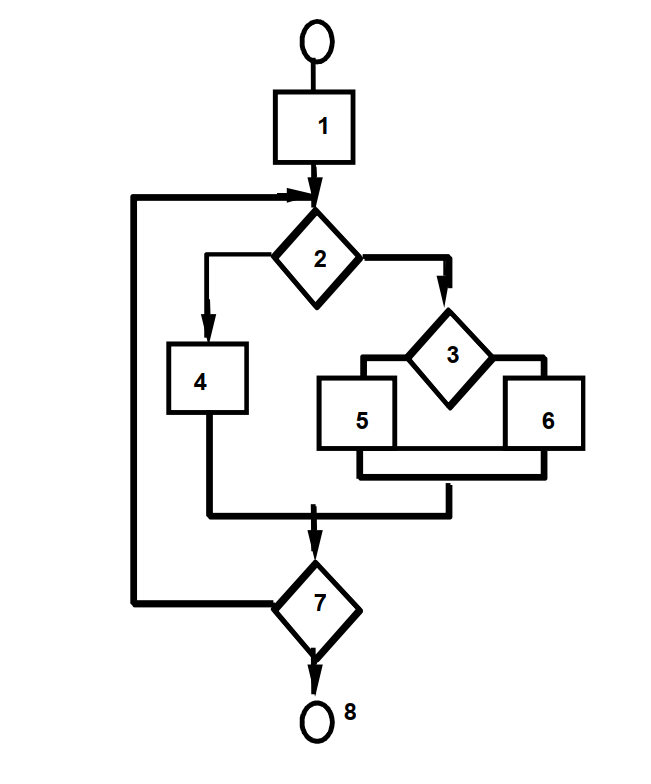
1. V(G) = R, trong đó R là số miền của đồ thị G
2. V(G) = E-N+2, trong đó E là số cung, N là số đỉnh của đồ thị G
3. V(G) =P+1, nếu là đồ thị dòng điều khiển nhị phân (chỉ chứa các nút quyết định lý luận – chỉ có 2 cung xuất True /False) và P là số nút quyết định.

Đối chiều với đồ thị luồng điều khiển trong hình 2.7 độ phức tạp chu trình V(G) được tính như sau :

1. Công thức 1 : V(G) = R = 6
2. Công thức 2 : V(G) = E – N + 2 = 15 – 11 + 2 = 6
3. Công thức 3 : V(G) = P + 1 = 5 + 1 = 6

Độ phức tạp Cyclomatic Complexity chính là số đường thi hành tuyến tính độc lập của thành phần phần mềm cần kiểm thử. Nếu chúng ta lựa chọn được đúng C đường thi hành tuyến tính độc lập của thành phần phần mềm cần kiểm thử và kiểm thử tất cả các đường thi hành này thì sẽ đạt được phủ kiểm thử cấp 3 như đã trình bày trong phần trước.

VD : Ta có sơ đồ như sau



Hình 2.10. Ví dụ sơ đồ chu trình Cyclomatic Complexity

* V(G) = 4
* Ta có các đường cơ bản
* Đường 1 : 1, 2, 3, 6, 7, 8
* Đường 2 : 1, 2, 3, 5, 7, 8
* Đường 3 : 1, 2, 4, 7, 8
* Đường 4 : 1, 2, 4, 7, 2, 4, …7, 8

### Quy trình xác định các đường cơ bản

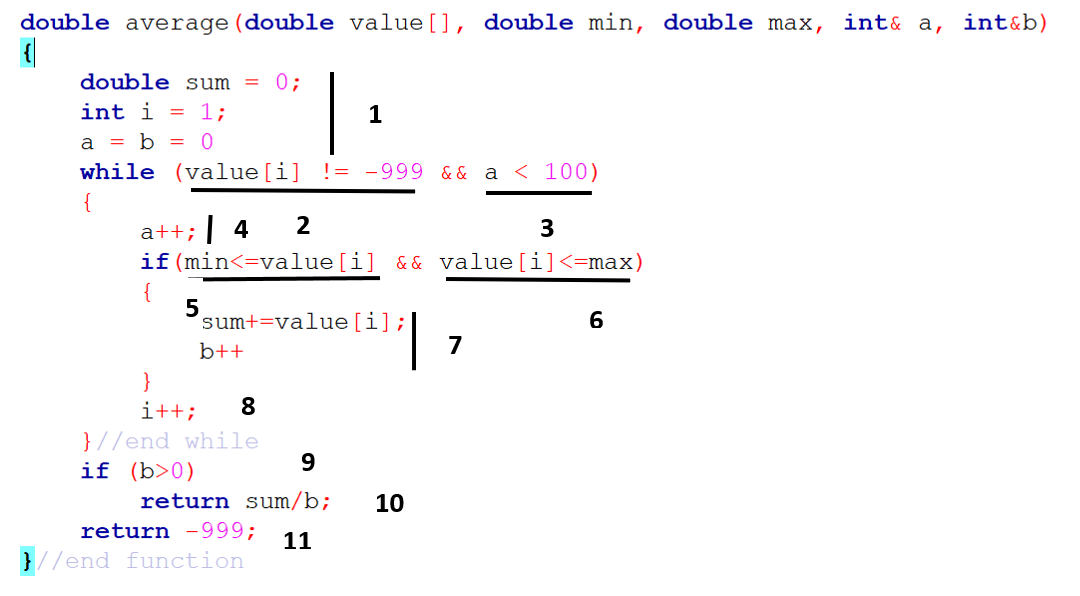
1. Xác định đường cơ bản, đường này nên là đường thi hành phổ biến nhất.
2. Để chọn đường thứ 2, thay đổi cung xuất của nút quyết định đầu tiên và cố gắng giữ lại maximum phần còn lại.
3. Để chọn đường thứ 3, dùng đường cơ bản nhưng thay đổi cung xuất của nút quyết định thứ 2 và cố gắng giữ lại maximum phần còn lại.
4. Tiếp tục thay đổi cung xuất cho từng nút quyết định trên đường cơ bản để xác định đường thứ 4, 5, … cho đến khi không còn nút quyết định nào trong đường cơ bản nữa.
5. Lặp dùng tuần tự các đường tìm được làm đường cơ bản để xác định các đường mới xung quanh nó y như các bước 2,3, 4 cho đến khi không tìm được đường tuyến tính độc lập nào nữa.

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

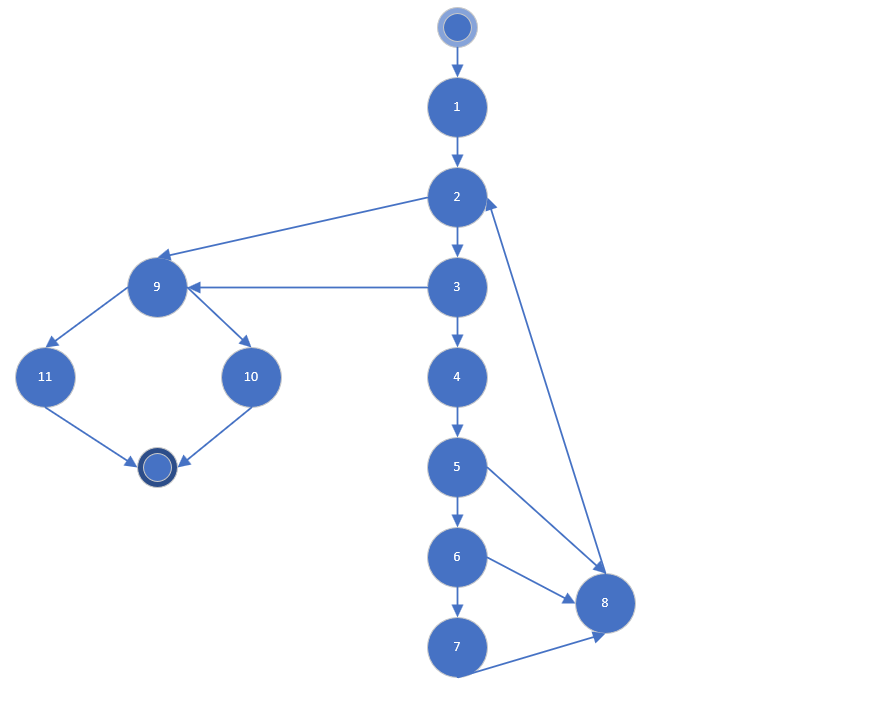
Phương pháp kiểm thử đường dẫn cơ sở có thể áp dụng để kiểm thử thủ tục chi tiết hoặc cho mã nguồn bao gồm các bước sau :

1. Bước 1 : Sử dụng mã nguồn hoặc thiết kế để xây dựng đồ thị luồng điều khiển tương ứng.
2. Bước 2 : Tính toán độ phức tạp chu trình V(G)
3. Bước 3 : Xác định tập cơ sở của các đường dẫn độc lập (một đường dẫn được gọi là độc lập với các đường dẫn khác nếu nó có ít nhất một cạnh không xuất hiện trong các đường dẫn khác).
4. Bước 4 : Chuẩn bị các trường hợp kiểm thử có khả năng thực hiện mỗi đường dẫn trong tập cơ sở.

Chúng ta dùng hàm tính giá trị trung bình cộng của các số, **average** trong C như hình sau 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 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 biên trên (max) và biên dưới (min). Đầu vào này sẽ được kết thúc bằng giá trị -999. Ta có code của hàm **average**



Hình 2.11 Mã nguồn code hàm **average**



Hình 2.12. Đồ thị luồng điều khiển hàm **average** minh họa phát sinh các trường hợp kiểm thử theo đường dẫn cơ sở

**Bước 1** : Vẽ đồ thị luồng điều khiển từ mã nguồn của chương trình như hình ….

**Bước 2** : Tính toán độ phức tạp chu trình V(G)

V(G) = P (số đỉnh điều kiện) + 1 = 5 + 1 = 6 (trong hình 2.10 có 5 đỉnh điều kiện là : 2, 3, 5, 6, 9)

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

* Đường dẫn 1 : 1  2  9  11
* Đường dẫn 2 : 1  2  9  10
* Đường dẫn 3 : 1  2  3  9  11
* Đường dẫn 4 : 1  2  3  4 5 8 2 ….
* Đường dẫn 5 : 1  2  3  4 5 6 8 2…
* Đường dẫn 6 : 1  2  3  4 5 6 7 8 2

**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 giá trị trung bình chính xác

Lưu 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 = {2, 6, 7, …, 120} (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 : Average = (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, 68, 2, -999}, min = 0, max = 100

Đầu ra mong muốn : Average = (7 + 32 + 23 + 68 + 2) / 5

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

* **Trường hợp kiểm thử đường dẫn 6**

Đầu vào : Values = {3, 4, 12, 15, 16, 2, -999}, min = 0, max = 100

Đầu ra mong muốn : Average = (3 + 4 + 12 + 15 + 16 + 2) / 6

Mục đích : Việc tính giá trị trung bình là đúng

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

## Quy trình kiểm thử hộp trắng

Quy trình kiểm thử thành phần phần mềm gồm các bước sau:

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

## Quy trình xác định Test path

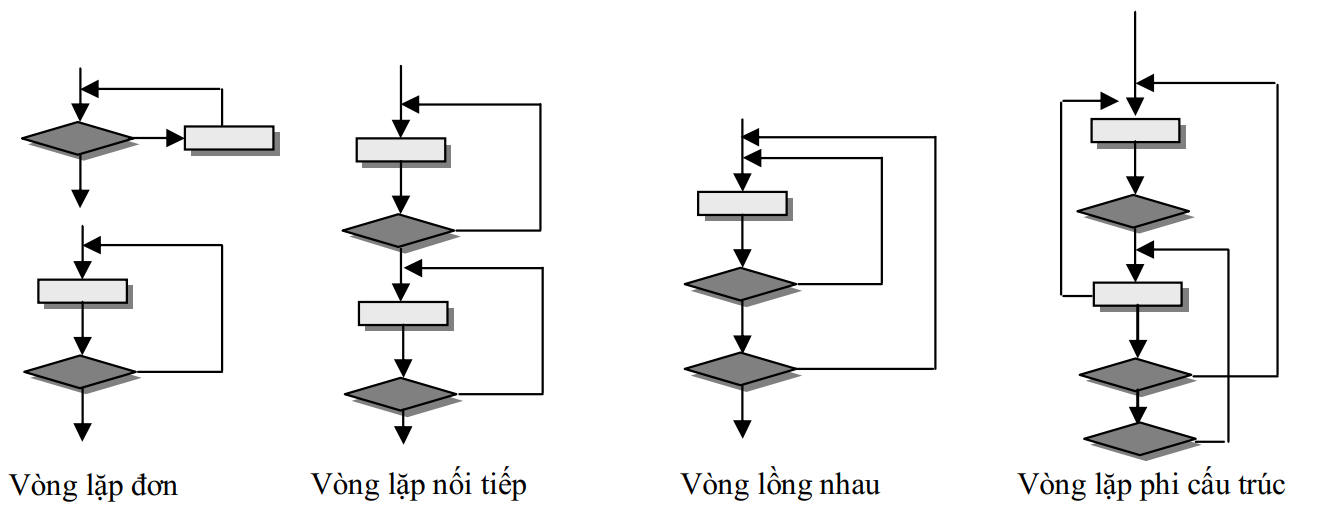
Quy trình xác định C đường tuyến độc lập gồm các bước:

1. Xác đinh đường tuyến tính đầu tiên bằng cách đi dọc theo nhánh bên trái nhất của các nút quyết định. Chọn đường này là pilot.
2. Dựa trên được pilot, thay đổi cung xuất của nút quyết định đầu tiên và cố gắng giữ lại maximum phần còn lại.
3. Dựa trên đường pilot, thay đổi cung xuất của nút quyết định thứ 2 và cố gắng giữ lại phần maximum còn lại
4. Tiếp tục thay đổi cung xuất cho từng nút quyết định trên đường pilot để xác định đường thứ 4,5 … cho đến khi không còn nút quyết định nào trong đường pilot nữa.
5. Lặp chọn tuần tự từng đường tìm được làm pilot để xác định các đường mới xung quanh nó y như các bước 2,3,4 cho đến khi không tìm được đường tuyến tính độc lập nào nữa (khi đủ số C).

## Kiểm thử vòng lặp

Cho dù chúng ta tiến hành kiểm thử các đơn vị chương trình với độ đo C3 (độ đo với yêu cầu cao nhất), phương pháp kiểm thử dòng điều khiển không thể kiểm thử các vòng lặp xuất hiện trong các đơn vị chương trình. Lý do là các đường đi sinh ra từ đồ thị dòng điều khiển không chứa các vòng lặp. Vì lý do này, chúng ta cần sinh thêm các ca kiểm thử cho các vòng lặp nhằm giảm tỷ lệ lỗi của các đơn vị chương trình.

Với mỗi đơn vị chương trình có vòng lặp, chúng ta cần quan tâm đến bốn trường hợp sau:



Hình 2.13. Các kiểu vòng lặp

* Vòng lặp phi cấu trúc: đơn vị chương trình chứa các vòng lặp không theo cấu trúc cố định

### Vòng lặp đơn giản

Vòng lặp đơn giản: đơn vị chương trình chỉ chứa đúng một vòng lặp (thân của vòng lặp không chứa các vòng lặp khác).

Các bước kiểm tra cho vòng lặp đơn giản

1. Vòng lặp thực hiện 0 lần
2. Vòng lặp thực hiện 1 lần
3. Vòng lặp thực hiện 2 lần
4. Vòng lặp thực hiện k lần, 2 < k < n - 1, với n là số lần lặp tối đa của vòng lặp
5. Vòng lặp thực hiện n - 1 lần
6. Vòng lặp thực hiện n lần
7. Vòng lặp thực hiện n + 1 lần

VD: Với vòng lặp **while** trong hàm **average** như hình , vòng lặp này chỉ thực hiện lặp tối đa 100 lần nên chúng ta không thể sinh ca kiểm thử để nó thực hiện n + 1 = 101 lần. Kết quả là chúng ta chỉ cần sinh sáu ca kiểm thử đầu tiên như trong bảng 2.9 nhằm kiểm thử vòng lặp này.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Lần lặp** | **Inputs** | **EO** | **RO** | **Note** |
| tcl0 | 0 | [-999, ...], 1, 2 | -999 |  |  |
| tcl1 | 1 | [1,-999], 1, 2 | 1 |  |  |
| tcl2 | 2 | [1,2,-999], 1, 2 | 1.5 |  |  |
| tclk | 5 | [1,2,3,4,5,-999], 1, 10 | 3 |  |  |
| tcl(n-1) | 99 | [1,2,...,99,-999], 1, 100 | 50 |  |  |
| tcln | 100 | [1,2,...,100], 1, 2 | 50.5 |  |  |
| tcl(n+1) |  |  |  |  |  |

Bảng 2.11. Các ca kiểm thử cho kiểm thử vòng lặp while của hàm **average**

Với các chương trình hoặc đơn vị chương trình có các vòng lặp liền kề, chúng ta tiến hành kiểm thử tuần tự từ trên xuống. Mỗi vòng lặp được kiểm thử bằng bảy ca kiểm thử như vòng lặp đơn giản. Trong trường hợp các vòng lặp lồng nhau, chúng ta tiến hành kiểm thử tuần tự các vòng lặp theo thứ tự từ trong ra ngoài (mỗi vòng lặp cũng dùng 7 ca kiểm thử như đã mô tả ở trên).

### Vòng lặp lồng nhau

Vòng lặp lồng nhau: đơn vị chương trình chỉ chứa các vòng lặp chứa các lệnh lặp khác.

Các bước cần kiểm tra cho vòng lặp dạng lồng nhau

* Khởi đầu với vòng lặp nằm bên trong nhất. Thiết lập các tham số lặp cho các vòng lặp bên ngoài về giá trị nhỏ nhất.
* Kiểm tra với tham số min + 1, 1 giá trị tiêu biểu, max - 1 và max cho vòng lặp bên trong nhất trong khi các tham số lặp của các vòng lặp bên ngoài là nhỏ nhất.
* Tiếp tục tương tự với các vòng lặp liền ngoài tiếp theo cho đến khi tất cả vòng lặp bên ngoài được kiểm tra.

### Vòng lặp nối tiếp

Vòng lặp nối tiếp: đơn vị chương trình chỉ chứa các lệnh lặp kế tiếp nhau.

Nếu các vòng lặp là độc lập với nhau thì kiểm tra như trường các vòng lặp dạng đơn giản, nếu không thì kiểm tra như trường hợp các 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.

## Kết chương

Chương này đã giới thiệu một số kỹ thuật thiết yếu để kiểm thử hộp trắng thành phần phần mềm, đó là kỹ thuật kiểm thử dòng điều khiển. Chúng ta đã giới thiệu các cấp độ kiểm phủ kiểm thử khác nhau, giới thiệu đồ thị dòng điều khiển và đồ thị dòng điều khiển cơ bản của thành phần phần mềm, độ phức tạp Cyclomatic C, quy trình tổng quát để kiểm thử dòng điều khiển. Chương này chúng ta cũng đã giới thiệu một vài ví dụ cụ thể.

# THỰC NGHIỆM KIỂM THỬ LUỒNG ĐIỀU KHIỂN

## Đặc tả của bài toán

Doanh nghiệp có thiết bị phát wifi marketing và cần dựng 1 bản mẫu để hiện thị mỗi khi người dùng đăng nhập vào wifi sẽ dựa vào bản mẫu để hiện thị lên giao diện. Cho người dùng điền thông tin mà doanh nghiệp yêu cầu để có thể truy cập vào wifi. Thực chức năng tạo template thông qua api. Yêu cầu phải validate các thuộc tính gửi lên nếu ko thì trả về thông tin lỗi để người dùng biết.

Các thông tin nhập cần điền :

- Người sở hữu template

- Logo

- File Term

- File Polyci

- Ảnh nền

- Giao diện hiện thị trên desktop

- Giao diện hiển thị trên mobile

## Mã code của chương trình

// Hàm validate User dùng để kiểm tra dữ liệu user của người dùng

private void validateUser(){

1. if(Objects.isNull(owner)){

2. throw new BadRequestException("Owner cannot be null");

}

3. if(Objects.isNull(owner.getEmail())){

4. throw new BadRequestException(" Owner's Email cannot be null");

}

5. if(Objects.isNull(owner.getName())){

6. throw new BadRequestException("Owner's Name cannot be null");

}

}

//Hàm validatePDF dùng để kiểm tra file gửi lên có phải là file PDF hay không

private void validatePDF(){

7. Boolean isTermFilePDF = isMatchPattern(PDF\_PATTERN,termFile);

8. if (!isTermFilePDF){

9. throw new BadRequestException("Term file is not pdf file");

}

10. Boolean isPolicyFilePDF = isMatchPattern(PDF\_PATTERN,policyFile);

11. if (!isPolicyFilePDF){

12. throw new BadRequestException("Policy file is not pdf file");

}

}

// Hàm validateImage là kiểm tra xem có phải file ảnh hay không

private void validateImage(){

13. Boolean isBackgroundImage = isMatchPattern(IMAGE\_PATTERN,background);

14. if(!isBackgroundImage){

15. throw new BadRequestException("Background is not image");

}

16. Boolean islogo = isMatchPattern(IMAGE\_PATTERN,logo);

17. if(!islogo){

18. throw new BadRequestException("Logo is not image");

}

19. Boolean isDesktopPreview = isMatchPattern(IMAGE\_PATTERN,desktopPreview);

20. if(!isDesktopPreview){

21. throw new BadRequestException("Desktop review is not image");

}

22. Boolean isMobilePreview = isMatchPattern(IMAGE\_PATTERN,mobilePreview);

23. if(!isMobilePreview){

24. throw new BadRequestException("Mobile review is not image");

}

}

25.

new ResponseEntity(validateService.insertTemplate(templateDto),HttpStatus.CREATED)

## Dựng biểu đồ luồng điều khiển

// Hàm validate User dùng để kiểm tra dữ liệu user của người dùng

private void validateUser(){

**1**

1. if(Objects.isNull(owner)){

2. throw new BadRequestException("Owner cannot be null");

**2**

}

**3**

3. if(Objects.isNull(owner.getEmail())){

**4**

4. throw new BadRequestException(" Owner's Email cannot be null");

}

**5**

5. if(Objects.isNull(owner.getName())){

**6**

6. throw new BadRequestException("Owner's Name cannot be null");

}

}

//Hàm validatePDF dùng để kiểm tra file gửi lên có phải là file PDF hay không

private void validatePDF(){

**7**

7. Boolean isTermFilePDF = isMatchPattern(PDF\_PATTERN,termFile);

**8**

8. if (!isTermFilePDF){

**9**

9. throw new BadRequestException("Term file is not pdf file");

}

**10**

10. Boolean isPolicyFilePDF = isMatchPattern(PDF\_PATTERN,policyFile);

**11**

11. if (!isPolicyFilePDF){

**12**

12. throw new BadRequestException("Policy file is not pdf file");

}

}

// Hàm validateImage là kiểm tra xem có phải file ảnh hay không

private void validateImage(){

**13**

13. Boolean isBackgroundImage = isMatchPattern(IMAGE\_PATTERN,background);

**14**

14. if(!isBackgroundImage){

15. throw new BadRequestException("Background is not image");

**15**

}

**16**

16. Boolean islogo = isMatchPattern(IMAGE\_PATTERN,logo);

**17**

17. if(!islogo){

18. throw new BadRequestException("Logo is not image");

**18**

}

**19**

19. Boolean isDesktopPreview = isMatchPattern(IMAGE\_PATTERN,desktopPreview);

**20**

20. if(!isDesktopPreview){

**21**

21. throw new BadRequestException("Desktop review is not image");

}

**22**

22. Boolean isMobilePreview = isMatchPattern(IMAGE\_PATTERN,mobilePreview);

**23**

23. if(!isMobilePreview){

**24**

24. throw new BadRequestException("Mobile review is not image");

}

}

**25**

25.

new ResponseEntity(validateService.insertTemplate(templateDto),HttpStatus.CREATED)

## Biểu đồ luồng điều khiển

A close up of a map

Description automatically generated

Hình 3.14. Biểu đồ luồng điều khiển

## Cách xác định số Testcase

Dựa vào độ phức tạp chu trình Cyclomatic Complexity V(G), ta có thể xác định số testcase bằng công thức: P + 1 (P là số đỉnh quyết định)

Theo biểu đồ luồng điều khiển như trong *Hình 3.1* ta xác định được có 9 đỉnh quyết định là 2, 4, 6, 9, 12, 15, 18, 21, 24 => Số testcase = 9 + 1 = 10. Vậy là tổng cộng ta sẽ có 10 Testcase

## Xác định các đường tuyến tính độc lập

**Path 1:**

0  1(T)  2  25

**Path 2:**

0  1(F)  3(T)  4  25

**Path 3:**

0  1(F)  3(F)  5(T)  6  25

**Path 4:**

0  1(F)  3(F)  5(F)  7  8(T)  9  25

**Path 5:**

0  1(F)  3(F)  5(F)  7  8(F)  10  11(T)  12  25

**Path 6:**

0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(T) 15  25

**Path 7:**

0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(T)  18  25

**Path 8:**

0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(F)  19  20 (T)  21  25

**Path 9:**

0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(F)  19  20 (F)  22  23(T)  24  25

**Path 10:**

0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(F)  19  20 (F)  22  23(F)  25.

## Bảng Testcase

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test path | Input | EO (Expected Output) | RO (Real Output) |
| Testcase 1 | 0  1(T)  2  25 | {  "termFile":"null",  "policyFile":"null",  "background":"null",  "logo":"null",  "desktopPreview":"null",  "mobilePreview":"null"  } | {  "message": "Owner cannot be null",  "code": 400  } |  |
| Testcase 2 | 0  1(F)  3(T)  4  25 | {  "owner":{},  "termFile":"null",  "policyFile":"null",  "background":"null",  "logo":"null",  "desktopPreview":"null",  "mobilePreview":"null"  } | {  "message": "Owner's Email cannot be null",  "code": 400  } |  |
| Testcase 3 | 0  1(F)  3(F)  5(T)  6  25 | {  "owner":{  "email":"nam@gmail.com"  },  "termFile":"null",  "policyFile":"null",  "background":"null",  "logo":"null",  "desktopPreview":"null",  "mobilePreview":"null"  } | {  "message": "Owner's Name cannot be null",  "code": 400  } |  |
| Testcase 4 | 0  1(F)  3(F)  5(F)  7  8(T)  9  25 | {  "owner":{  "email":"nam@gmail.com",  "name":"name"  },  "termFile":"http://localhost/term.docx",  "policyFile":"null",  "background":"null",  "logo":"null",  "desktopPreview":"null",  "mobilePreview":"null"  } | {  "message": "Term file is not pdf file",  "code": 400  } |  |
| Testcase 5 | 0  1(F)  3(F)  5(F)  7  8(F)  10  11(T)  12  25 | {  "owner":{  "email":"nam@gmail.com",  "name":"name"  },  "termFile":"http://localhost/term.pdf",  "policyFile":"http://localhost/policy.docx",  "background":"null",  "logo":"null",  "desktopPreview":"null",  "mobilePreview":"null"  } | {  "message": "Policy file is not pdf file",  "code": 400  } |  |
| Testcase 6 | 0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(T) 15  25 | {  "owner":{  "email":"nam@gmail.com",  "name":"name"  },  "termFile":"http://localhost/term.pdf",  "policyFile":"http://localhost/policy.pdf",  "background":"http://localhost/background.pdf",  "logo":"null",  "desktopPreview":"null",  "mobilePreview":"null"  } | {  "message": "Background is not image",  "code": 400  } |  |
| Testcase 7 | 0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(T)  18  25 | {  "owner":{  "email":"nam@gmail.com",  "name":"name"  },  "termFile":"http://localhost/term.pdf",  "policyFile":"http://localhost/policy.pdf",  "background":"http://localhost/background.jpg",  "logo":"http://localhost/logo.pdf",  "desktopPreview":"null",  "mobilePreview":"null"  } | {  "message": "Logo is not image",  "code": 400  } |  |
| Testcase 8 | 0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(F)  19  20 (T)  21  25 | {  "owner":{  "email":"nam@gmail.com",  "name":"name"  },  "termFile":"http://localhost/term.pdf",  "policyFile":"http://localhost/policy.pdf",  "background":"http://localhost/background.jpg",  "logo":"http://localhost/logo.png",  "desktopPreview":"http://localhost/desktop.pdf",  "mobilePreview":"null"  } | {  "message": "Desktop review is not image",  "code": 400  } |  |
| Testcase 9 | 0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(F)  19  20 (F)  22  23(T)  24  25 | {  "owner":{  "email":"nam@gmail.com",  "name":"name"  },  "termFile":"http://localhost/term.pdf",  "policyFile":"http://localhost/policy.pdf",  "background":"http://localhost/background.jpg",  "logo":"http://localhost/logo.png",  "desktopPreview":"http://localhost/desktop.jpg",  "mobilePreview":"http://localhost/mobile.pdf"  } | {  "message": "Mobile review is not image",  "code": 400  } |  |
| Testcase 10 | 0  1(F)  3(F)  5(F)  7  8(F)  10  11(F)  13  14(F) 16  17(F)  19  20 (F)  22  23(T)  24  25 | {  "owner":{  "email":"nam@gmail.com",  "name":"nam"  },  "termFile":"http://localhost/term.pdf",  "policyFile":"http://localhost/policy.pdf",  "background":"http://localhost/background.jpg",  "logo":"http://localhost/logo.png",  "desktopPreview":"http://localhost/desktop.jpg",  "mobilePreview":"http://localhost/mobile.jpg"  } | {  "owner": {  "name": "nam",  "phone": null,  "email": "nam@gmail.com"  },  "termFile": "http://localhost/term.pdf",  "policyFile": "http://localhost/policy.pdf",  "background": "http://localhost/background.jpg",  "logo": "http://localhost/logo.png",  "desktopPreview": "http://localhost/desktop.jpg",  "mobilePreview": "http://localhost/mobile.jpg"  } |  |

Bảng 3.12. Các ca kiểm thử của chương trình

## Giới thiệu về công cụ kiểm thử chất lượng code Sonarqube

### Giới thiệu

Được phát triển bởi 10 năm trước bởi SonarSource, sonarqube là một platform mã nguồn mở giúp chúng ta có thể kiểm tra chất lượng code của dự án, được viết bằng java nhưng nó hổ trợ nhiều ngôn ngữ khác nhau: PHP, Ruby, Java (bao gồm cả Android), C#, JavaScript, TypeScript, C/C++, Kotlin, Go, COBOL, PL/SQL, PL/I, ABAP, VB.NET, VB6, Python, RPG, Flex, Objective-C, Swift, CSS, HTML, và XML và hỗ trợ các database để lưu trữ kết quả: MySql, Postgresql.

Nó được tích hợp vào công cụ phát triển Eclipse, Visual Studio, IntelliJ IDEA thông qua plugin SonarLint và tích hợp với các công cụ khác như LDAP, Active Directory, GitHub….

Code review là một phần không thể thiếu trong quá trình phát triển phần mềm khi làm viêc với nhóm. Tuy nhiên, không phải lúc nào người review của bạn cũng rảnh hoặc kiên nhẫn đễ đọc từng dòng code, từng ký tự trong code của bạn, nhưng Sonarqube thì có thể làm điều này, Sonarqube sẽ quét tất cả code có trong dự án của bạn và đánh giá code dựa theo các coding standard tương ứng của tất cả ngôn ngữ có trong dự án. Ngoài ra Sonarqube có thể làm nhiều hơn là code convention:

* Phát hiện bug
* Phát hiện code smell, duplicate
* Tính toán độ bao phủ của Unit test (Unit-test coverage)
* Tính toán technical debt
* So sánh chất lượng code so với các lần kiểm tra trước
* ….....

### Cài đặt trên hệ điều hành Ubuntu 18.04

**B1: Cài đặt docker-compose**

# apt install docker-compose



**B2: Kiểm tra cài đặt thành công**

# docker-compose -v



**B3: Tạo thư mục chứa file yml**

# mkdir data



**B4: Tạo file yml với tên như sau: “docker-compose.yml”**

# cd data/

# touch docker-compose.yml

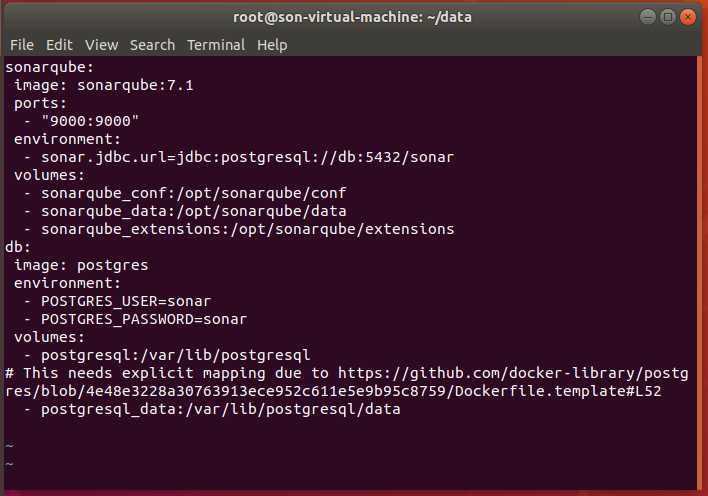


**B5: Chỉnh sửa file docker-compose.yml**

# vi docker-compose.yml



Thêm nội dung file như sau



Sau đó ta lưu lại

**B6: Từ thư mục chứa file docker-compose.yml trên, chạy lệnh sau để để tạo các thành phần mà nó mô tả:**

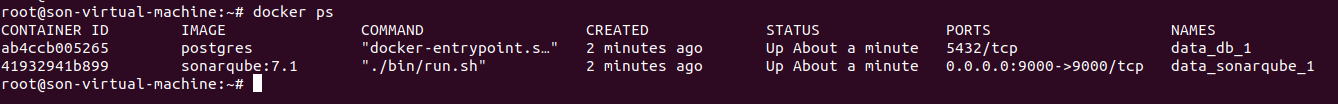
# docker-compose up



**B7: Sau khi tải và chạy xong ta sẽ kiểm tra như sau**

Kiểm tra container đang chạy ở docker

# docker ps



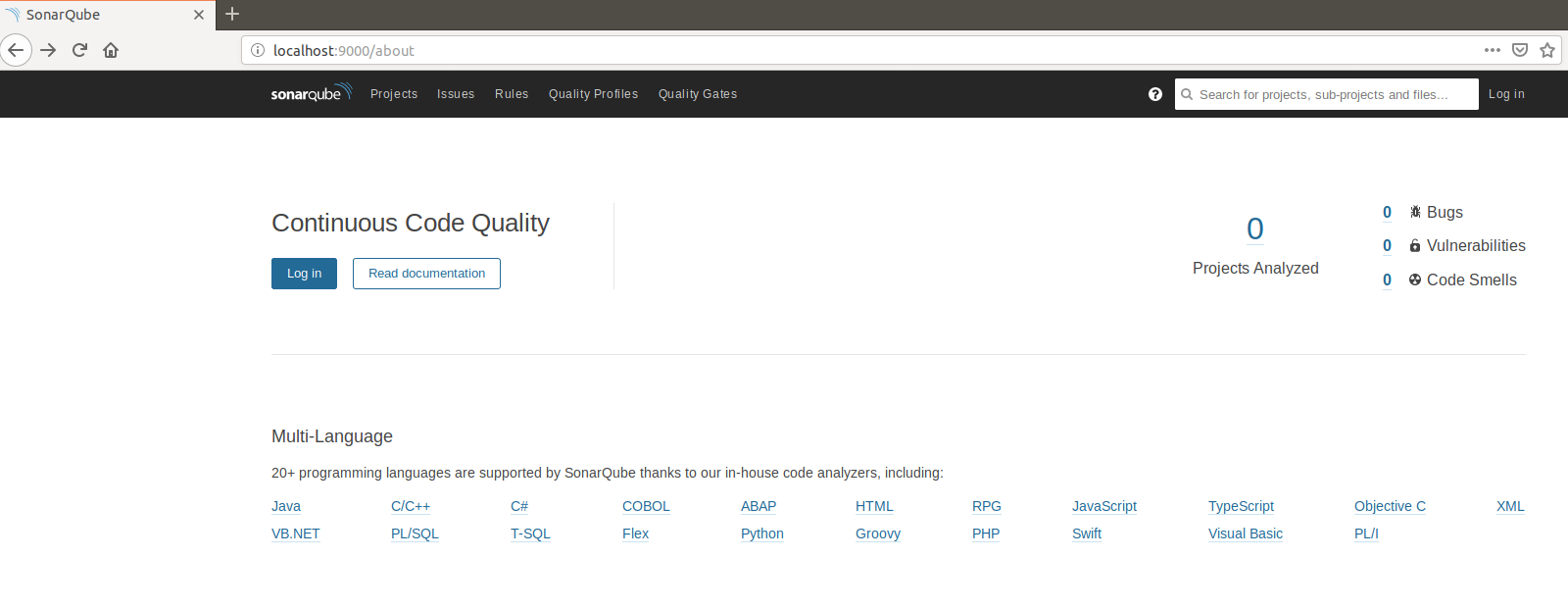
Ta có thể thấy container sonarqube tải về với STATUS: Up About a minute có nghĩa là mới được bật được khoảng 1 phút và đang chạy trên port 9000

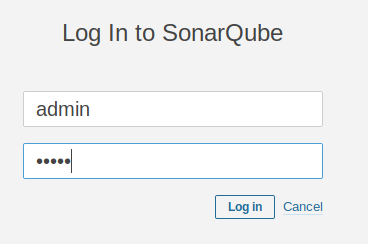
**B8: Truy cập vào Sonarqube**

- Đường dẫn vào Sonarqube

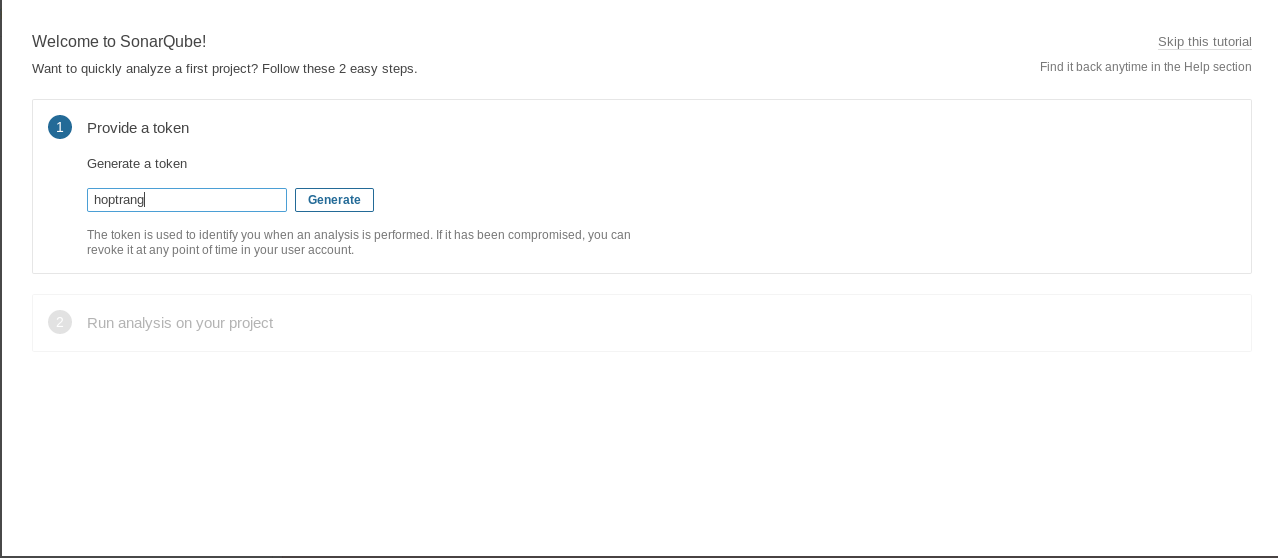
[http://localhost:9000](http://localhost:9000/)

- Đây là giao diện của Sonarqube, chúng ta sẽ đăng nhập bằng tài khoản user và password đều là “admin”

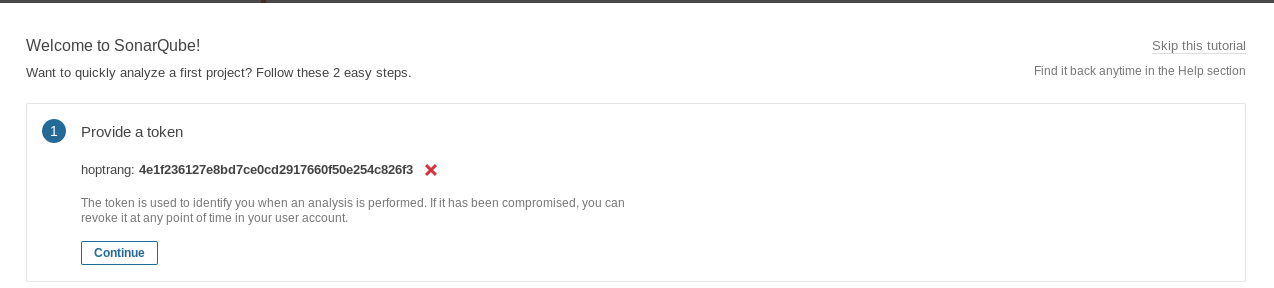




- Khi đăng nhập vào chúng ta sẽ có giao diện như sau:



- Chúng ta cần phải tạo 1 token với tên bất kì do ta chọn để khi kiểm tra lỗi trên mã nguồn code sẽ cần dùng đến token này để gửi về server sonarqube cho mình



Vậy là ta đã tạo xong token và cần lưu giá trị của token “hoptrang” là 4e1f236127e8bd7ce0cd2917660f50e254c826f3

**B9: Cài đặt công cụ sonar-scanner để quét lỗi từ mã nguồn code**

- Tạo thư mục chứa sonar-scanner

# sudo mkdir /opt/sonarscanner



- Chuyển đường dẫn vào thử mục vừa tạo

# cd /opt/sonarscanner



- Tải sonarqube scanner sử dụng wget (Phiên bản sonar-scanner sẽ khác nhau trong từng thời điểm)

# sudo wget <https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-3.2.0.1227-linux.zip>



- Giải nén file vừa tải về

# sudo unzip sonar-scanner-cli-3.2.0.1227-linux.zip



- Xóa file nén

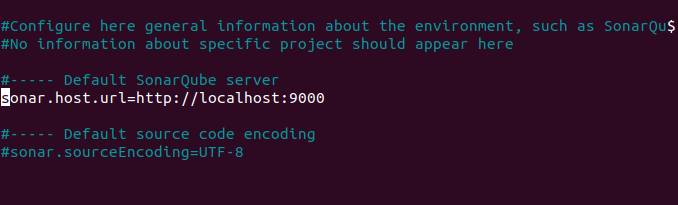
# sudo rm sonar-scanner-cli-3.2.0.1227-linux.zip



- Chỉnh sửa file cấu hình sonar-scanner.propertises

# sudo nano sonar-scanner-3.2.0.1227-linux/conf/sonar-scanner.properties

- Bỏ chú thích # ở “sonar.host.url=https://localhost:9000” để báo cho sonar-scanner biết nơi cần gửi kết quả sau khi phân tích mã nguồn code



- Lưu và đóng file. Bây giờ ta sẽ làm cho sonar-scanner có thể thực thi

# sudo chmod +x sonar-scanner-3.2.0.1227-linux/bin/sonar-scanner

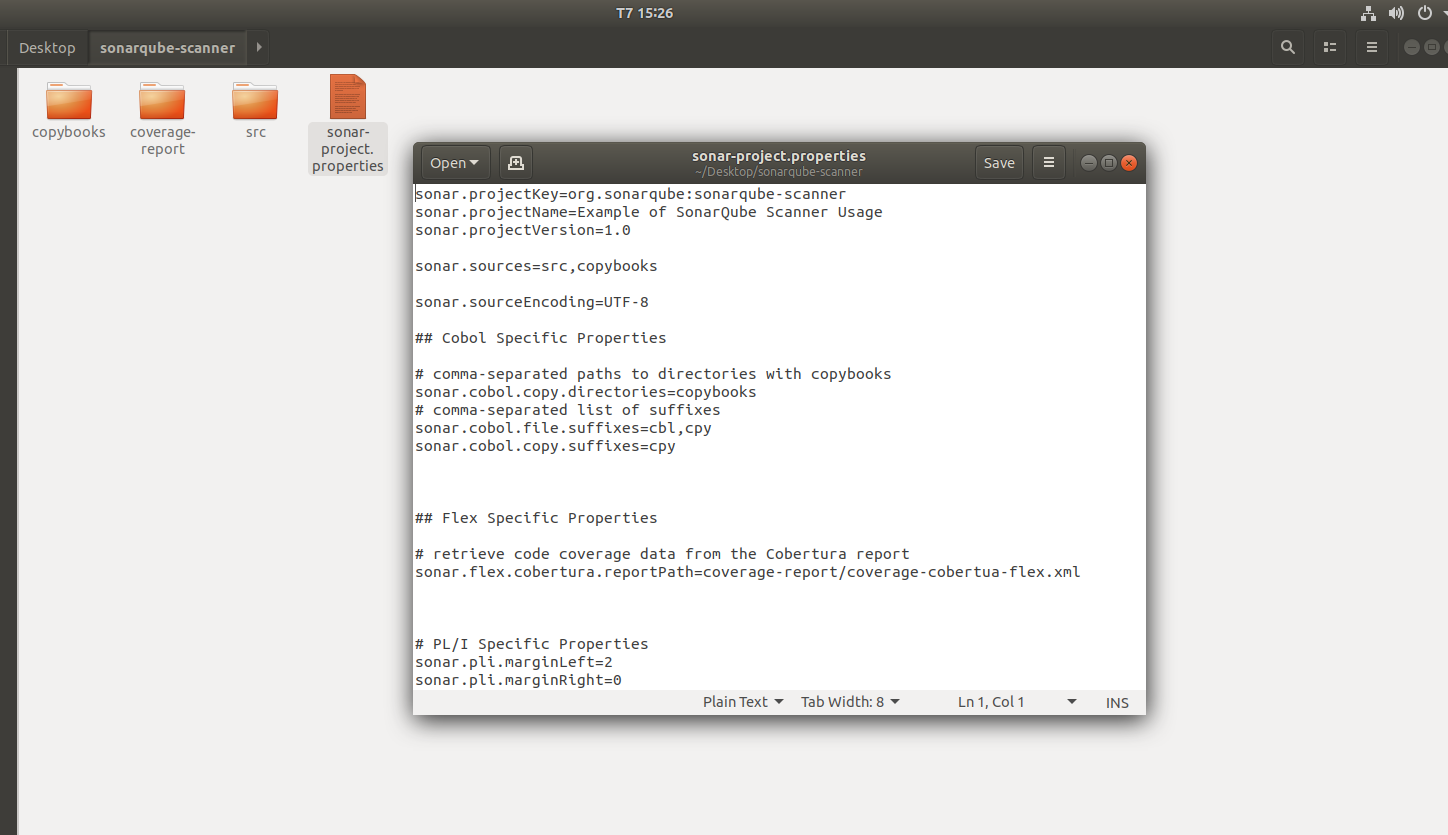


- Sau đó tạo một đường link đến hệ thống hay còn gọi là tạo biến môi trường mà ta có thể gọi sonar-scanner mỗi khi quét mã nguồn code

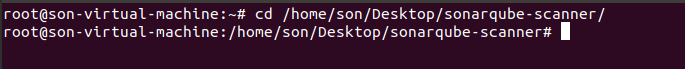
sudo ln -s /opt/sonarscanner/sonar-scanner-3.2.0.1227-linux/bin/sonar-scanner /usr/local/bin/sonar-scanner

**B10: Thử nghiệm quét một project mẫu để phát hiện lỗi**

- Giả sử ta có một project mẫu đã tạo và cấu hình file sonar-project.scanner để có thể quét mã nguồn code của chương trình



- Chỉ đường dẫn đến thư mục chứa project



- Tiến hành quét project

# sonar-scanner -D sonar.login=your\_token\_here

your\_token\_here chính là token mà ta đã tạo ở bước 7 ở trên



- Sau khi quét xong nó sẽ gửi mã lỗi về server Sonarqube

