

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN



ThS. Nguyễn Lan Oanh

ThS. Hoàng Thị Cành

ThS. Đào Thị Thu

ThS. Nguyễn Thị Tính

BÀI GIẢNG

**KIỂM THỬ VÀ ĐẢM BẢO
CHẤT LƯỢNG PHẦN MỀM**

Tài liệu lưu hành nội bộ

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

ThS. Nguyễn Lan Oanh

ThS. Hoàng Thị Cành

ThS. Đào Thị Thu

ThS. Nguyễn Thị Tính

BÀI GIẢNG
KIỂM THỬ VÀ ĐẢM BẢO
CHẤT LƯỢNG PHẦN MỀM

Thái Nguyên, tháng 12 năm 2023

MỤC LỤC

MỤC LỤC	3
MỘT SỐ THUẬT NGỮ	7
MỞ ĐẦU.....	8
CHƯƠNG 1: TỔNG QUAN VỀ KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM.....	11
Bài 1: Tổng quan về kiểm thử và đảm bảo chất lượng phần mềm (Số tiết: 03 tiết)	11
1.1. Các định nghĩa cơ bản về kiểm thử phần mềm.....	11
1.2. Các kiểm thử phần mềm	15
1.3. Mô tả bài toán kiểm thử qua biểu đồ Venn	16
1.4. Phân loại lỗi	18
1.5. Các mức kiểm thử	21
1.6 Các nguyên tắc kiểm thử phần mềm.....	22
1.7 Đảm bảo chất lượng phần mềm.....	24
1.7.1 Những mục tiêu đảm bảo chất lượng phần mềm.....	25
1.7.2 Phân loại yêu cầu phần mềm ứng với các yếu tố chất lượng phần mềm ..	25
CHƯƠNG 2: QUY TRÌNH KIỂM THỬ PHẦN MỀM.....	27
Bài 2: Quy trình kiểm thử phần mềm (Số tiết: 03 tiết).....	27
2.1 Quy trình kiểm thử phần mềm tổng quát.....	27
2.2 Phân tích quy trình kiểm thử phần mềm.....	28
2.2.1. Requirement analysis - Phân tích yêu cầu	28
2.2.2. Test planning - Lập kế hoạch kiểm thử.....	29
2.2.3. Test case development - Thiết kế kịch bản kiểm thử.....	30
2.2.4. Test environment set up - Thiết lập môi trường kiểm thử	31
2.2.5. Test execution - Thực hiện kiểm thử	31
2.2.6. Test cycle closure – Kết thúc chu kỳ kiểm thử	32
2.3. Tổng quát	33
CHƯƠNG 3: KIỂM THỬ HÀM.....	36
Bài 3: Các phương pháp kiểm thử hàm (Số tiết: 03 tiết).....	36
3.1 Tổng quan	36
3.1.1 Phương pháp hệ thống	38
3.1.2 Lựa chọn phương pháp phù hợp	42
3.2 Kiểm thử giá trị biên.....	44

3.2.1 Kiểm thử giá trị biên mạnh	47
3.2.2 Kiểm thử giá trị biên tổ hợp	48
3.2.3 Kiểm thử các giá trị đặc biệt.....	48
3.3 Kiểm thử lớp tương đương.....	49
3.3.1 Kiểm thử lớp tương đương yếu	50
3.3.2 Kiểm thử lớp tương đương mạnh	51
3.3.3 Kiểm thử lớp tương đương đơn giản	52
Bài 4: Các phương pháp kiểm thử hàm - tiếp (Số tiết: 03 tiết).....	54
3.4 Kiểm thử bằng bảng quyết định	54
3.5 Kiểm thử tổ hợp	56
3.5.1 Kiểm thử đôi một.....	57
3.5.2 Ma trận trực giao	58
CHƯƠNG 4: KỸ THUẬT KIỂM THỬ HỘP TRẮNG	60
Bài 5: Kiểm thử dòng điều khiển (Số tiết: 03 tiết).....	60
4.1 Tổng quan về kiểm thử hộp trắng	60
4.2 Kiểm thử dựa trên dòng điều khiển	60
4.2.1 Đồ thị dòng điều khiển	60
4.2.2 Các độ đo kiểm thử.....	62
Bài 6: Kiểm thử dòng điều khiển dựa trên độ đo (Số tiết: 03 tiết)	64
4.2.3 Kiểm thử dựa trên độ đo.....	64
4.2.4 Tổng kết.....	71
Bài 7: Kiểm thử dòng dữ liệu (Số tiết: 03 tiết).....	72
4.3 Kiểm thử dòng dữ liệu	72
4.3.1 Các vấn đề phổ biến về dòng dữ liệu	72
4.3.2 Tổng quan về kiểm thử dòng dữ liệu	76
4.3.3 Đồ thị dòng dữ liệu	77
Bài 8: Các độ đo cho kiểm thử dòng dữ liệu (Số tiết: 03 tiết)	80
4.3.4 Các khái niệm về dòng dữ liệu	80
4.3.5 Các độ đo cho kiểm thử dòng dữ liệu	82
4.3.6 Sinh các ca kiểm thử.....	86
CHƯƠNG 5: QUẢN LÝ LỖI PHẦN MỀM VÀ BÁO CÁO KIỂM THỬ.....	89
Bài 9: Quản lý lỗi phần mềm và báo cáo kiểm thử (Số tiết: 03 tiết).....	89
5.1 Các thành phần của lỗi	89
5.1.1 Giới thiệu.....	89

5.1.2 Vòng đời của lỗi.....	90
5.1.3 Phương pháp viết báo cáo lỗi tốt.....	91
5.2 Tổng quan về TestReport	92
5.2.1 Khái niệm thế nào là Test Report?.....	92
5.2.2 Tầm quan trọng của bản Test Report?	93
5.2.3 Nội dung cần có trong 1 bản Test Report hoàn chỉnh.....	94
5.2.4 Những khó khăn trong quá trình phân tích Test Report.....	95
5.2.5 Tiêu chí cơ bản để lựa chọn Report Tool	96
5.3 Cấu trúc của TestReport	97
5.3.1 Thông tin dự án	98
5.3.3 Tóm tắt kiểm thử.....	99
5.3.4 Kết luận về thiếu sót.....	100
CHƯƠNG 6: CÁC THÀNH PHẦN VÀ CÁC CHUẨN ĐẢM BẢO CHẤT	
LƯỢNG PHẦN MỀM.....	102
Bài 10: Đảm bảo chất lượng phần mềm (Số tiết: 03 tiết)	102
6.1. Độ đo chất lượng phần mềm	102
6.1.1 Mục tiêu đo lường phần mềm	103
6.1.2 Phân loại độ đo chất lượng phần mềm.....	103
6.2. Giá thành của chất lượng phần mềm	104
6.3. SQA trong các tiêu chuẩn ISO	108
6.3.1 ISO 9001 và ISO 9000-3.....	108
6.3.2 Các mô hình tăng trưởng khả năng – phương pháp đánh giá CMM và CMMI	110
6.4. SQA trong các hệ tiêu chuẩn IEEE	111
6.4.1 IEEE/EIA Std 12207- các tiến trình vòng đời phần mềm	111
6.4.2 IEEE Std 1012 – xác minh và thẩm định	113
6.4.3 IEEE Std 1028 – rà soát	114
CÁC CÂU HỎI THƯỜNG GẶP	118
BÀI TẬP THỰC HÀNH	119
Bài thực hành số 1 (Số tiết: 05 tiết).....	119
Bài thực hành số 2 (số tiết: 05 tiết)	121
Bài thực hành số 3 (số tiết: 05 tiết)	124
Bài thực hành số 4 (số tiết: 05 tiết)	127
Bài thực hành số 5 (số tiết: 05 tiết)	130
Bài thực hành số 6 (số tiết: 05 tiết)	134

CÁC TỪ VIẾT TẮT

TT	Từ viết tắt	Ý nghĩa của từ
1	SQA	Software Quality Assurance – Đảm bảo chất lượng phần mềm
2	STLC	Software Testing Life Cycle – Vòng đời kiểm thử phần mềm
3	TC	TestCase – Ca kiểm thử
4	ISO	International Organization for Standardization
5	IEEE	Institute of Electrical and Electronics Engineers

MỘT SỐ THUẬT NGỮ

TT	Thuật ngữ	Diễn giải ý nghĩa
1	Lỗi (Error)	Lỗi là những vấn đề mà con người mắc phải trong quá trình phát triển các sản phẩm phần mềm.
2	Sai (Fault)	Sai là kết quả của lỗi, hay nói khác đi, lỗi sẽ dẫn đến sai
3	Thất bại (Failure)	Thất bại xuất hiện khi một lỗi được thực thi
4	Sự cố (Incident)	Sự cố là triệu chứng liên kết với một thất bại và thể hiện cho người dùng hoặc người kiểm thử về sự xuất hiện của thất bại này.
5	Ca kiểm thử	Ca kiểm thử gồm một tập các dữ liệu đầu vào và một xâu các giá trị đầu ra mong đợi đối với phần mềm
6	Kiểm thử hàm	Kiểm thử hàm là các hoạt động kiểm tra chương trình dựa trên tài liệu mô tả chức năng, yêu cầu phần mềm
7	kiểm thử dòng điều khiển	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
8	kiểm thử dòng dữ liệu	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..
9	Test Report	Test Report là bản báo cáo quan trọng nó bao gồm toàn bộ dữ liệu liên quan tới mục tiêu kiểm thử, hoạt động kiểm thử và kết quả của toàn bộ quá trình kiểm thử đó.

DANH MỤC HÌNH ẢNH

Hình 1.1: Một vòng đời của việc kiểm thử.	14
Hình 1.2: Thông tin về một ca kiểm thử tiêu biểu	16
Hình 1.3: Các hành vi được cài đặt và được đặc tả.	17
Hình 1.4: Các hành vi được cài đặt, được đặc tả và được kiểm thử	18
Hình 1.5: Phân loại sai bằng độ nghiêm trọng	19
Hình 1.6: Các mức trừu tượng và mức kiểm thử trong mô hình thác nước	21
Hình 2.1 Quy trình kiểm thử phần mềm	27
Hình 3.1: Các bước chính của phương pháp hệ thống cho kiểm thử hàm	39
Hình 3.2: Miền xác định của hàm hai biến	44
Hình 3.3: Các ca kiểm thử phân tích giá trị biên cho một hàm hai biến	45
Hình 3.4: Các ca kiểm thử mạnh cho hàm hai biến	47
Hình 3.5: Các ca kiểm thử biên tổ hợp của hàm hai biến	48
Hình 4.1: Các thành phần cơ bản của đồ thị chương trình	61
Hình 4.2: Các cấu trúc điều khiển phổ biến của chương trình	61
Hình 4.3: Mã nguồn của hàm foo và đồ thị dòng điều khiển của nó	62
Hình 4.4: Quy trình kiểm thử đơn vị chương trình dựa trên độ đo	65
Hình 4.5: Hàm foo và đồ thị dòng điều khiển của nó	66
Hình 4.6: Hàm foo và đồ thị dòng điều khiển ứng với độ đo C3	67
Hình 4.7: Hàm average và đồ thị dòng điều khiển ứng với độ đo C3.	69
Hình 4.8: Tuần tự các câu lệnh có vấn đề thuộc loại 1	73
Hình 4.9: Tuần tự các câu lệnh có vấn đề thuộc loại 2	74
Hình 4.10: Sơ đồ chuyển trạng thái của một biến	75
Hình 4.11: Ví dụ về định nghĩa và sử dụng các biến	78
Hình 4.12: Mã nguồn của hàm ReturnAverage bằng ngôn ngữ C	79
Hình 4.13: Đồ thị dòng dữ liệu của hàm ReturnAverage trong Hình 4.12	79
Hình 4.14: Mối quan hệ giữa các độ đo cho kiểm thử dòng dữ liệu	86
Hình 4.15: Mối quan hệ giữa các độ đo dòng dữ liệu thực thi được	88
Hình 5.1 Vòng đời của lỗi phần mềm	90
Hình 5.2 Lựa chọn công cụ báo cáo	97
Hình 5.3 Cấu trúc TestReport	98
Hình 5.4 Thông tin dự án	98
Hình 5.5 Ví dụ về TestReport	99
Hình 5.6 Ví dụ minh họa bản Test Report cho các lỗi	100
Hình 5.7 Biểu đồ biểu diễn dữ liệu lỗi	101
Hình 6.1 Mô hình tính chi phí truyền th	106
Hình 6.2 Mô hình tính chi phí mở rộng	107
Hình 6.3 Biểu đồ xương cá- Những tiến trình vòng đời phần mềm theo chuẩn IEEE/EIA 12207	112

DANH MỤC BẢNG

Bảng 1.1: Các sai lầm về đầu vào/đầu ra	19
Bảng 1.2: Các sai lầm về logic	20
Bảng 1.3: Các sai lầm về tính toán	20
Bảng 1.4: Các sai lầm về giao diện	20
Bảng 1.5: Các sai lầm về dữ liệu	20
Bảng 3.1: Các ca kiểm thử lớp tương đương cho Triangle	51
Bảng 3.2: Các ca kiểm thử lớp tương đương mạnh cho Triangle	52
Bảng 3.3: Ví dụ bảng quyết định	54
Bảng 3.4: Bảng quyết định để khắc phục sự cố máy in	55
Bảng 3.5: Các ca kiểm thử đôi một cho hàm g	57
Bảng 3.6: Mảng trực giao $L_4(2^3)$	57
Bảng 4.1: Các ca kiểm thử cho độ đo C1 của hàm foo	62
Bảng 4.2: Các ca kiểm thử cho độ đo C2 của hàm foo	63
Bảng 4.3: Các trường hợp cần kiểm thử của độ đo C3 với hàm foo	63
Bảng 4.4: Các ca kiểm thử cho độ đo C3 của hàm foo	64
Bảng 4.5: Các ca kiểm thử cho độ đo C1 của hàm foo	65
Bảng 4.6: Các ca kiểm thử cho độ đo C2 của hàm foo	66
Bảng 4.7: Các ca kiểm thử cho độ đo C3 của hàm foo	68
Bảng 4.8: Các ca kiểm thử cho độ đo C_3 của hàm average	69
Bảng 4.9: Các ca kiểm thử cho vòng lặp while của hàm average	70

MỞ ĐẦU

Bài giảng **Kiểm thử và đảm bảo chất lượng phần mềm** được tập thể giảng viên thuộc bộ môn Công nghệ phần mềm biên soạn nhằm phục vụ cho việc giảng dạy của giảng viên và học tập của sinh viên Trường Đại học Công nghệ thông tin và Truyền thông - Đại học Thái Nguyên. Tập bài giảng này được biên soạn theo nội dung đề cương chi tiết học phần **Kiểm thử và đảm bảo chất lượng phần mềm** ở trình độ đại học.

Nội dung tài liệu cung cấp cho sinh viên và giảng viên những chất liệu cơ bản bao phủ những nét chính về những phát triển lý thuyết cơ sở của việc kiểm thử phần mềm và các thực hành kiểm thử chung trong ngành công nghiệp phần mềm. Vì các khái niệm về chất lượng phần mềm là quá rộng, tài liệu chỉ định giới thiệu những nét chung nhất và cái nhìn tổng thể về kiểm thử và đảm bảo chất lượng phần mềm mà thôi. Thực ra thì phần mềm có rất nhiều loại khác nhau, với nhiều miền ứng dụng khác nhau. Ở mỗi loại và mỗi miền ứng dụng riêng biệt lại có các đặc thù riêng và cần được hỗ trợ bởi các kỹ thuật kiểm thử riêng cho chúng. Tài liệu không có tham vọng đi vào các chi tiết như vậy mà chỉ giới thiệu lý thuyết và thực hành kiểm thử chung và cơ bản nhất nhằm trang bị cho sinh viên những kỹ năng cơ bản để có thể hiểu và tự phát triển các kỹ thuật kiểm thử thích hợp cho các hệ thống phức tạp và chuyên dụng hơn trong thực tiễn sau này. Nội dung tài liệu gồm 6 chương:

Chương 1. Tổng quan về kiểm thử và đảm bảo chất lượng phần mềm.

Chương 2. Quy trình kiểm thử phần mềm.

Chương 3. Kiểm thử hàm.

Chương 4. Kỹ thuật kiểm thử hộp trắng.

Chương 5. Quản lý lỗi phần mềm và báo cáo kiểm thử.

Chương 6. Các thành phần và các chuẩn đảm bảo chất lượng phần mềm.

Mặc dù tập thể tác giả đã dành nhiều thời gian và công sức để biên soạn, song khó tránh khỏi thiếu sót. Vậy, chúng tôi kính mong quý thầy cô và các bạn sinh viên đóng góp ý kiến để cuốn bài giảng được hoàn thiện hơn. Xin trân trọng cảm ơn.

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

Nội dung chính của chương

Kiểm thử nhằm đánh giá chất lượng hoặc tính chấp nhận được của sản phẩm. Kiểm thử cũng nhằm phát hiện lỗi hoặc bất cứ vấn đề gì về sản phẩm. Chúng ta cần kiểm thử vì biết rằng con người luôn có thể mắc sai lầm. Điều này đặc biệt đúng trong lĩnh vực phát triển phần mềm và các hệ thống điều khiển bởi phần mềm. Chương này nhằm phác họa một bức tranh tổng thể về kiểm thử phần mềm. Các chương còn lại sẽ nằm trong khuôn khổ của bức tranh này và ở mức chi tiết hơn.

Mục tiêu cần đạt được của chương

- Hiểu các khái niệm, thuật ngữ về kiểm thử và đảm bảo chất lượng phần mềm
- Phân biệt các mức kiểm thử
- Hiểu cá nguyên tắc kiểm thử phần mềm

Bài 1: Tổng quan về kiểm thử và đảm bảo chất lượng phần mềm (Số tiết: 03 tiết)

1.1. Các định nghĩa cơ bản về kiểm thử phần mềm

Lỗi (Error): Lỗi là những vấn đề mà con người mắc phải trong quá trình phát triển các sản phẩm phần mềm. Trong thực tế, con người luôn có thể phạm lỗi. Khi lập trình viên phạm lỗi trong lập trình, ta gọi các lỗi đó là bug (con bọ). Lỗi có thể phát tán. Chẳng hạn, một lỗi về xác định yêu cầu có thể dẫn đến sai lầm về thiết kế và càng sai khi lập trình theo thiết kế này. Lỗi là nguyên nhân dẫn đến sai.

Sai (Fault): Sai là kết quả của lỗi, hay nói khác đi, lỗi sẽ dẫn đến sai. Cũng có thể nói sai là một biểu diễn của lỗi dưới dạng một biểu thức, chẳng hạn chương trình, văn bản, sơ đồ dòng dữ liệu, biểu đồ lớp,... Sai lầm có thể khó bị phát hiện. Khi nhà thiết kế mắc lỗi bỏ sót trong quá trình thiết kế, sai kết quả từ lỗi đó là thiếu mất cái gì đó mà lẽ ra cần phải có. Sai về nhiệm vụ xuất hiện khi vào sai thông tin, còn sai về bỏ quên xuất hiện khi không vào đủ thông tin. Loại sai thứ hai khó phát hiện và khó sửa hơn loại sai thứ nhất.

Thất bại (Failure): Thất bại xuất hiện khi một lỗi được thực thi. Có hai điều cần lưu ý ở đây. Một là thất bại chỉ xuất hiện dưới dạng có thể chạy được mà thông thường là mã nguồn. Hai là các thất bại chỉ liên kết với các lỗi về nhiệm vụ. Còn các thất bại tương ứng với các lỗi về bỏ quên thì xử lý thế nào? Những cái lỗi không

bao giờ được tiến hành, hoặc không được tiến hành trong khoảng thời gian dài cần được xử lý thế nào? Virus Michaelangelo là một ví dụ về lỗi loại này. Nó chỉ được tiến hành vào ngày sinh của Michaelangelo, tức ngày 6/3 mà thôi. Việc khảo sát có thể ngăn chặn nhiều thất bại bằng cách tìm ra các lỗi thuộc cả hai loại.

Sự cố (Incident): Khi thất bại xuất hiện, nó có thể hiển thị hoặc không, tức là rõ ràng hoặc không rõ ràng đối với người dùng hoặc người kiểm thử. Sự cố là triệu chứng liên kết với một thất bại và thể hiện cho người dùng hoặc người kiểm thử về sự xuất hiện của thất bại này.

Yêu cầu của khách hàng và đặc tả của phần mềm: Phần mềm được viết để thực hiện các nhu cầu của khách hàng. Các nhu cầu của khách hàng được thu thập, phân tích và khảo cứu và là cơ sở để quyết định chính xác các đặc trưng cần thiết mà sản phẩm phần mềm cần phải có. Dựa trên yêu cầu của khách hàng và các yêu cầu bắt buộc khác, đặc tả được xây dựng để mô tả chính xác các yêu cầu mà sản phẩm phần mềm cần đáp ứng, và có giao diện thế nào. Tài liệu đặc tả là cơ sở để đội ngũ phát triển phần mềm xây dựng sản phẩm phần mềm. Khi nói đến thất bại trên đây là nói đến việc sản phẩm phần mềm không hoạt động đúng như đặc tả. Lỗi một khi được tiến hành có thể dẫn đến thất bại. Do đó, lỗi về bỏ quên được coi là tương ứng với các lỗi khi xây dựng đặc tả.

Kiểm chứng và thẩm định: Kiểm chứng (verification) và thẩm định (validation) hay được dùng lẫn lộn, nhưng thực ra chúng có ý nghĩa khác nhau. Kiểm chứng là quá trình để đảm bảo rằng một sản phẩm phần mềm thỏa mãn đặc tả của nó. Còn thẩm định là quá trình để đảm bảo rằng sản phẩm đáp ứng được yêu cầu của người dùng (khách hàng). Trong thực tế, chúng ta cần thực hiện kiểm chứng trước khi thực hiện việc thẩm định sản phẩm phần mềm. Vì vậy, chúng ta có thuật ngữ V&V (Verification & Validation). Lý do của việc này là chúng ta cần đảm bảo sản phẩm đúng với đặc tả trước. Nếu thực hiện việc thẩm định trước, một khi phát hiện ra lỗi, chúng ta không thể xác định được lỗi này do đặc tả sai hay do lập trình sai so với đặc tả.

Chất lượng và độ tin cậy của phần mềm: Theo từ điển, chất lượng của một sản phẩm được thể hiện bằng các đặc trưng phù hợp với đặc tả của nó. Theo cách hiểu này, chất lượng của một sản phẩm phần mềm là sự đáp ứng các yêu cầu về chức năng

(tức là các hàm cần được tính toán), sự hoàn thiện và các chuẩn đã được đặc tả, cùng các đặc trưng mong chờ từ mọi sản phẩm phần mềm chuyên nghiệp. Chất lượng phần mềm đặc trưng cho “độ tốt, độ tuyệt hảo” của phần mềm, và gồm có các yếu tố về chất lượng như: tính đúng đắn (hành vi đúng như đặc tả), tính hiệu quả (tiết kiệm thời gian và tiền bạc), độ tin cậy, tính khả kiểm thử (kiểm thử được và dễ), dễ học, dễ sử dụng, dễ bảo trì ... Như vậy, độ tin cậy chỉ là một yếu tố để đánh giá chất lượng phần mềm. Người kiểm thử hay nhầm lẫn độ tin cậy với chất lượng. Khi kiểm thử đạt tới mức phần mềm chạy ổn định, có thể phụ thuộc vào nó, người kiểm thử thường cho rằng phần mềm đã đạt chất lượng cao. Các yếu tố về mặt chất lượng mà liên quan trực tiếp đến việc phát triển phần mềm được gọi là các tiêu chuẩn chất lượng như tính có cấu trúc, tính đơn thể, tính khả kiểm thử, ...

Độ tin cậy của phần mềm là xác suất để phần mềm chạy không có thất bại trong một khoảng thời gian nhất định. Nó được xem là một yếu tố quan trọng của chất lượng phần mềm. Ngoài ra, thời gian trung bình cho việc khắc phục một sự cố cũng là một thông số quan trọng trong việc đánh giá độ tin cậy của sản phẩm phần mềm

Kiểm thử: Rõ ràng việc kiểm thử liên quan đến các khái niệm trên: lỗi, sai, thất bại và sự cố. Có hai mục đích chính của một phép thử: tìm thất bại hoặc chứng tỏ việc tiến hành của phần mềm là đúng đắn.

Vai trò của kiểm thử phần mềm: Kiểm thử phần mềm đóng vai trò quan trọng trong việc đánh giá và thu được chất lượng cao của sản phẩm phần mềm trong quá trình phát triển. Thông qua chu trình “*kiểm thử - tìm lỗi - sửa lỗi*”, ta hy vọng chất lượng của sản phẩm phần mềm sẽ được cải tiến. Mặt khác, thông qua việc tiến hành kiểm thử mức hệ thống trước khi cho lưu hành sản phẩm, ta biết được sản phẩm của ta tốt ở mức nào. Vì thế, nhiều tác giả đã mô tả việc kiểm thử phần mềm là một quy trình kiểm chứng để đánh giá và tăng cường chất lượng của sản phẩm phần mềm. Quy trình này gồm hai công việc chính là phân tích tĩnh và phân tích động.

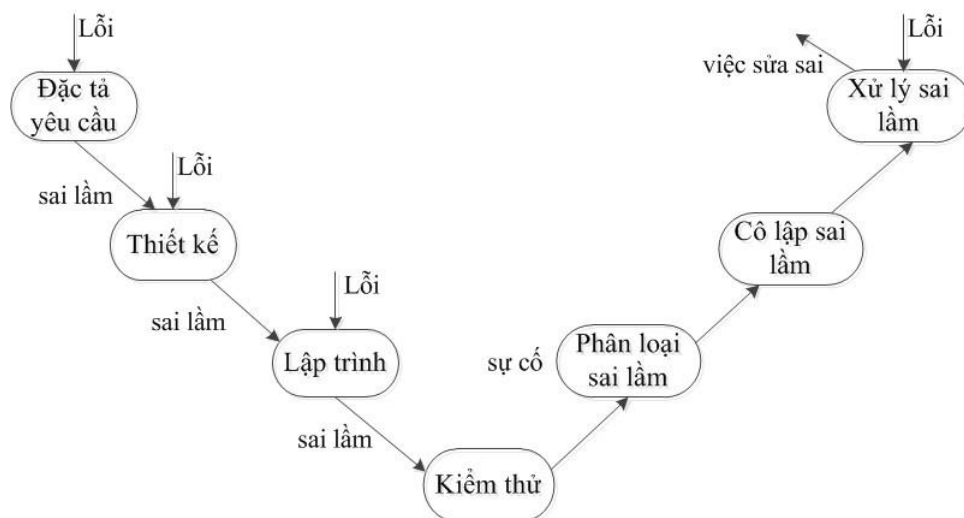
Phân tích tĩnh: Việc phân tích tĩnh được tiến hành dựa trên việc khảo sát các tài liệu được xây dựng trong quá trình phát triển sản phẩm như tài liệu đặc tả nhu cầu người dùng, mô hình phần mềm, hồ sơ thiết kế và mã nguồn phần mềm. Các phương pháp phân tích tĩnh truyền thống bao gồm việc khảo sát đặc tả và mã nguồn cùng các tài liệu thiết kế. Các kỹ thuật khảo sát này sẽ được giới thiệu trong chương 4.

Người ta cũng có thể dùng các kỹ thuật phân tích hình thức như kiểm chứng mô hình (model checking) và chứng minh định lý (theorem proving) để chứng minh tính đúng đắn của thiết kế và mã nguồn. Các kỹ thuật này tương đối phức tạp và nằm ngoài khuôn khổ của cuốn giáo trình này. Công việc này không động đến việc thực thi chương trình mà chỉ duyệt, lý giải về tất cả các hành vi có thể của chương trình khi được thực thi. Tối ưu hóa các chương trình dịch là các ví dụ về phân tích tĩnh.

Phân tích động: Phân tích động liên quan đến việc thực thi chương trình để phát hiện những thất bại có thể có của chương trình, hoặc quan sát các tính chất nào đó về hành vi và hiệu quả (performance). Vì gần như không thể thực thi chương trình trên tất cả các dữ liệu đầu vào có thể, ta chỉ có thể chọn một tập con các dữ liệu đầu vào để thực thi, gọi là các “ca kiểm thử”. Chọn như thế nào để được các bộ dữ liệu đầu vào hiệu quả (tức là các bộ dữ liệu có xác suất phát hiện thất bại (nếu có) cao hơn là công việc cần suy nghĩ và là nội dung chính của các giáo trình này

Bằng việc phân tích tĩnh và động, người kiểm thử muốn phát hiện nhiều lỗi nhất có thể được để chúng có thể được sửa ở giai đoạn sớm nhất trong quá trình phát triển phần mềm. Phân tích tĩnh và động là hai kỹ thuật bổ sung cho nhau và cần được làm lặp đi lặp lại nhiều trong quá trình kiểm thử.

Ca kiểm thử: Mỗi ca kiểm thử có một tên và được liên kết với một hành vi của chương trình. Ca kiểm thử gồm một tập các dữ liệu đầu vào và một xâu các giá trị đầu ra mong đợi đối với phần mềm



Hình 1.1: Một vòng đời của việc kiểm thử.

Hình 1.1 mô tả vòng đời của việc kiểm thử ứng với mô hình thác nước. Lưu ý rằng trong giai đoạn phát triển phần mềm, lỗi có thể được đưa vào tại các giai đoạn đặc tả yêu cầu, thiết kế và lập trình. Các lỗi này có thể tạo ra những sai lan truyền sang các phần còn lại của quá trình phát triển. Một nhà kiểm thử lỗi lạc đã tóm tắt vòng đời này như sau: Ba giai đoạn đầu là “đưa lỗi vào”, giai đoạn kiểm thử là để tìm lỗi, và ba giai đoạn cuối là “khử lỗi đi” [Pos90]. Bước sửa sai là cơ hội mới cho việc đưa vào lỗi (và các sai mới). Vì vậy, việc sửa sai này có thể làm cho phần mềm từ đúng trở thành sai. Trong trường hợp này, việc sửa sai là không đầy đủ. Kiểm thử hồi quy (sẽ được giới thiệu trong chương 11) là giải pháp tốt để giải quyết vấn đề này.

Các thuật ngữ trên đây cho thấy các ca kiểm thử chiếm vị trí trung tâm trong việc kiểm thử dựa trên phân tích động. Quá trình kiểm thử dựa trên phân tích động được chia thành các bước sau: lập kế hoạch kiểm thử, phát triển ca kiểm thử, chạy các ca kiểm thử và đánh giá kết quả kiểm thử. Tiêu điểm của cuốn giáo trình này là việc xác định tập hữu ích các ca kiểm thử, tức là các ca kiểm thử giúp ta cải tiến tốt hơn chất lượng của sản phẩm

1.2. Ca kiểm thử phần mềm

Cốt lõi của kiểm thử phần mềm dựa trên phân tích động là việc xác định tập các ca kiểm thử sao cho chúng có khả năng phát hiện nhiều nhất các lỗi (có thể có) của hệ thống cần kiểm thử. Vậy cái gì cần đưa vào các ca kiểm thử? Rõ ràng thông tin đầu tiên là đầu vào. Đầu vào có hai kiểu: tiền điều kiện (pre-condition) - tức là điều kiện cần thỏa mãn trước khi tiến hành ca kiểm thử - và dữ liệu đầu vào thực sự được xác định bởi phương pháp kiểm thử. Thông tin tiếp theo cần đưa vào là đầu ra mong đợi. Cũng có hai loại đầu ra: hậu điều kiện (post-condition) và dữ liệu đầu ra thực sự. Phần đầu ra của ca kiểm thử thường hay bị bỏ quên vì nó là phần khó xác định. Giả sử ta cần kiểm thử phần mềm tìm đường đi tối ưu cho máy bay khi cho trước các ràng buộc về hành lang bay và dữ liệu về thời tiết trong ngày của chuyến bay. Đường đi tối ưu của máy bay thực sự là gì? Có nhiều câu trả lời cho câu hỏi này. Câu trả lời lý thuyết là giả thiết về sự tồn tại của một cây đũa thần (oracle) biết được tất cả các câu trả lời. Câu trả lời thực tế, được gọi là kiểm thử tham chiếu, là hệ thống được kiểm thử dưới sự giám sát của các chuyên gia về lĩnh vực ứng dụng của phần mềm, những người có thể phán xét xem liệu các dữ liệu đầu ra đối với việc

tiến hành trên các dữ liệu đầu vào của ca kiểm thử có chấp nhận được hay không.

Hoạt động kiểm thử dẫn đến việc thiết lập các tiên điều kiện cần thiết, việc cung cấp các ca kiểm thử, quan sát dữ liệu đầu ra và so sánh chúng với các đầu ra mong đợi để xác định phát hiện các lỗi/khiếm khuyết (có thể có) của sản phẩm phần mềm.

Hình 1.2 mô tả các thông tin cơ bản trong một ca kiểm thử được phát triển đầy đủ, chủ yếu là để trợ giúp việc quản lý. Các ca kiểm thử cần phải định danh bằng tên/chỉ số và lý do tồn tại (chẳng hạn đặc tả nhu cầu tương ứng là một lý do). Cũng nên bổ sung thêm lịch sử tiến hành của một ca kiểm thử bao gồm cả việc chúng được thực hiện bởi ai và khi nào, kết quả của mỗi lần thực hiện ra sao, qua hay thất bại và được thực hiện trên phiên bản nào của phần mềm. Với các ca kiểm thử cho các hoạt động kiểm thử giao diện người dùng, ngoài thông tin về đầu vào, chúng ta cần bổ sung thêm các thông tin về trình tự nhập các đầu vào cho giao diện. Tóm lại, ta cần nhận thức rằng ca kiểm thử ít nhất cũng quan trọng như mã nguồn. Các ca kiểm thử cần được phát triển, kiểm tra, sử dụng, quản lý và lưu trữ một cách khoa học.

Chỉ số của ca kiểm thử
Mục đích
Tiên điều kiện
Đầu vào
Đầu ra mong đợi
Hậu điều kiện
Lịch sử thực hiện ca kiểm thử:
<i>Ngày Kết quả thực tế Phiên bản Kiểm thử viên</i>

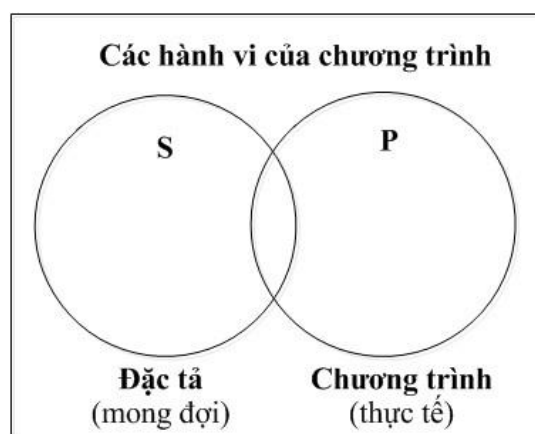
Hình 1.2: Thông tin về một ca kiểm thử tiêu biểu

1.3. Mô tả bài toán kiểm thử qua biểu đồ Venn

Kiểm thử chủ yếu liên quan tới hành vi của chương trình nơi mà hành vi phản ánh quan điểm về cấu trúc phổ dụng đối với các nhà phát triển hệ thống hoặc phần mềm. Sự khác biệt là quan điểm cấu trúc tập trung vào “là cái gì”, còn quan điểm hành vi lại tập trung vào “làm gì”. Một trong những nguyên nhân gây khó cho người kiểm thử là các tài liệu cơ sở thường được viết bởi và viết cho người phát triển và vì thế chúng thiên về thông tin cấu trúc và coi nhẹ thông tin về hành vi của chương

trình cần kiểm thử. Trong mục này, chúng ta phát triển một biểu đồ Venn đơn giản nhằm làm sáng tỏ vài điều về kiểm thử.

Xét một vũ trụ của hành vi chương trình cần kiểm thử, lưu ý rằng chúng ta đang quan tâm đến bản chất của việc kiểm thử là xác định tập các ca kiểm thử hữu ích. Cho trước một chương trình cùng đặc tả của nó. Gọi S là tập các hành vi được đặc tả và P là tập các hành vi của chương trình. Hình 1.3 mô tả mối quan hệ giữa vũ trụ các hành vi được lập trình và hành vi được đặc tả. Trong tất cả các hành vi có thể của chương trình, những hành vi được đặc tả nằm trong vòng tròn với nhãn S , còn những hành vi được lập trình là ở trong vòng tròn với nhãn P . Từ biểu đồ này, ta thấy rõ các bài toán mà người kiểm thử cần đối mặt là gì. Nếu có hành vi được đặc tả nhưng không được lập trình thì theo thuật ngữ trước đây, đây là những sai về bỏ quên. Tương tự, nếu có những hành vi được lập trình nhưng không được đặc tả, thì điều đó tương ứng với những sai về nhiệm vụ, và tương ứng với những lỗi xuất hiện sau khi đặc tả đã hoàn thành. Tương giao giữa S và P là phần đúng đắn, gồm có các hành vi vừa được đặc tả vừa được cài đặt. Chú ý rằng tính đúng đắn chỉ có nghĩa đối với đặc tả và cài đặt và là khái niệm mang tính tương đối.

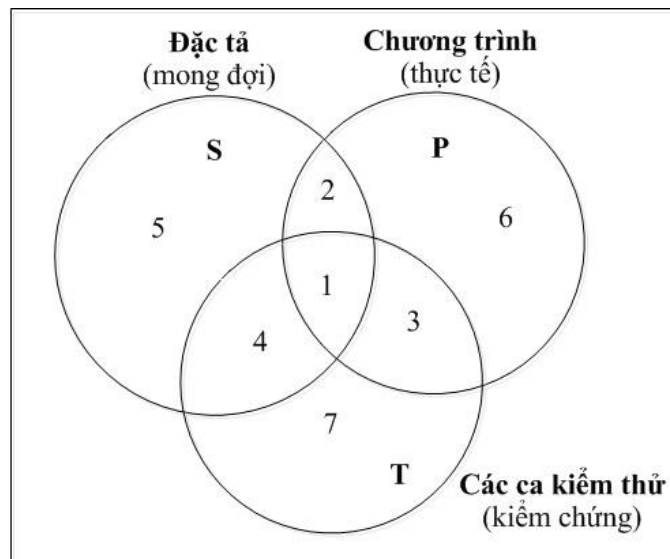


Hình 1.3: Các hành vi được cài đặt và được đặc tả.

Vòng tròn mới (vòng tròn T) trong hình 1.4 là cho các ca kiểm thử. Lưu ý rằng tập các hành vi của chương trình nằm trọn trong vũ trụ chuyên đề mà ta đang xét. Vì một ca kiểm thử cũng xác định một hành vi (xin các nhà toán học sẽ bỏ qua cho việc dùng thuật ngữ quá tải này). Xét mối quan hệ giữa S , P và T . Có thể có các hành vi được đặc tả mà không được kiểm thử (các miền 2 và 5), các hành vi được

đặc tả và được kiểm thử (các miền 1 và 4), và các ca kiểm thử tương ứng với các hành vi không được đặc tả (các miền 3 và 7).

Tương tự, có thể có các hành vi được lập trình mà không được kiểm thử (các miền 2 và 6), các hành vi được lập trình và được kiểm thử (các miền 1 và 3), và các ca kiểm thử tương ứng với các hành vi không được lập trình (các miền 4 và 7). Việc xem xét tất cả các miền này là hết sức quan trọng. Nếu có các hành vi được đặc tả mà không có các ca kiểm thử tương ứng, việc kiểm thử là chưa đầy đủ. Nếu có các ca kiểm thử tương ứng với các hành vi không được đặc tả, có thể có hai khả năng: hoặc đặc tả còn thiếu hoặc ca kiểm thử không đảm bảo. Theo kinh nghiệm, một người kiểm thử giỏi sẽ thường cho các ca kiểm thử thuộc loại đầu, và đây chính là lý do người kiểm thử cần tham gia vào giai đoạn khảo sát đặc tả và thiết kế.



Hình 1.4: Các hành vi được cài đặt, được đặc tả và được kiểm thử.

Ta có thể thấy việc kiểm thử như là công việc của một nghệ nhân: người kiểm thử có thể làm gì để làm cho miền tương giao (phần giao) của các tập (miền 1) là lớn nhất có thể? Làm thế nào để xác định các ca kiểm thử trong tập T? Câu trả lời là các ca kiểm thử cần được xác định bởi một phương pháp kiểm thử phù hợp.

1.4. Phân loại lỗi

Các định nghĩa của ta về lỗi và sai xoay quanh sự phân biệt giữa quy trình và sản phẩm: quy trình nói ta làm điều gì đó thế nào, còn sản phẩm là kết quả cuối cùng của quy trình. Kiểm thử phần mềm và đảm bảo chất lượng phần mềm (Software Quality Assurance - SQA) gặp nhau ở điểm là SQA cố gắng cải tiến chất lượng sản

phẩm bằng việc cải tiến quy trình. Theo nghĩa này thì kiểm thử là định hướng sản phẩm. SQA quan tâm nhiều hơn đến việc giảm thiểu lỗi trong quá trình phát triển, còn kiểm thử quan tâm chủ yếu đến phát hiện sai trong sản phẩm. Cả hai nguyên lý này đều sử dụng định nghĩa về các loại sai. Các sai được phân loại theo vài cách: giai đoạn phát triển khi cái sai tương ứng xuất hiện, các hậu quả của các thất bại tương ứng, độ khó cho việc giải quyết, độ rủi ro của việc không giải quyết được, v.v. Một cách phân loại được ưa thích là dựa trên việc xuất hiện bất thường: chỉ một lần, thỉnh thoảng, xuất hiện lại hoặc lặp đi lặp lại nhiều lần. Hình 1.5 minh họa một cách phân loại sai dựa trên độ nghiêm trọng của hậu quả.

1 Nhẹ	Lỗi chính tả
2 Vừa	Hiểu lầm hoặc thừa thông tin
3 Khó chịu	Tên bị thiếu, cắt chữ hoặc hóa đơn có giá trị 0.0 đồng
4 Bực mình	Vài giao dịch không được xử lý
5 Nghiêm trọng	Mất giao dịch
6 Rất nghiêm trọng	Xử lý giao dịch sai
7 Cực kỳ nghiêm trọng	Lỗi rất nghiêm trọng xảy ra thường xuyên
8 Quá quắt	Hủy hoại cơ sở dữ liệu
9 Thảm họa	Hệ thống bị tắt
10 Dị họa	Thảm họa lây lan

Hình 1.5: Phân loại sai bằng độ nghiêm trọng.

Để xử lý các loại sai, hãy tham khảo [IEE93] về việc phân loại chuẩn cho các bất thường của phần mềm. Chuẩn IEEE định nghĩa quy trình giải quyết bất thường một cách chi tiết gồm bốn giai đoạn: nhận biết, khảo sát, hành động và bố trí lại. Một số bất thường phổ biến được mô tả trong các bảng từ Bảng 1.1 đến Bảng 1.5. Hầu hết các bất thường này đều đã được đề cập trong chuẩn IEEE. Ngoài ra, chúng tôi cũng bổ sung thêm một số bất thường khác.

Bảng 1.1: Các sai lầm về đầu vào/đầu ra

Loại	Các trường hợp
dữ liệu đầu vào	dữ liệu đầu vào đúng nhưng không được chấp nhận
	dữ liệu đầu vào sai nhưng được chấp nhận
	mô tả sai hoặc thiếu
	tham số sai hoặc thiếu
dữ liệu đầu ra	khuôn dạng sai
	kết quả sai
	kết quả đúng tại thời gian sai (quá sớm hoặc quá muộn)

	kết quả không đầy đủ hoặc thiếu
	kết quả giả tạo
	văn phạm/chính tả
	các trường hợp khác

Bảng 1.2: Các sai lầm về logic

trường hợp
lặp thừa trường hợp
điều kiện cực đoan bị bỏ qua
thể hiện sai
thiếu điều kiện
điều kiện ngoại lai
kiểm thử sai biến
việc lặp của chu trình không đúng
phép toán sai (chẳng hạn dùng $<$ cho \leq)

Bảng 1.3: Các sai lầm về tính toán

thuật toán sai
thiếu tính toán
toán hạng sai
sai về dấu ngoặc
chưa đủ độ chính xác (làm tròn hoặc cắt đuôi)
hàm đi kèm sai

Bảng 1.4: Các sai lầm về giao diện

xử lý gián đoạn sai (trong các hệ thống nhúng)
thời gian vào ra (time out)
gọi sai thủ tục
gọi đến một thủ tục không tồn tại
tham số không sánh cặp được (mismatch) (chẳng hạn về kiểu và số)
kiểu không tương thích
bao hàm thừa

Bảng 1.5: Các sai lầm về dữ liệu

khởi tạo sai
lưu trữ và truy cập sai
giá trị chỉ số và cờ báo sai
gói và mở sai
sử dụng sai biến
tham chiếu sai dữ liệu
đơn vị hoặc thang chia sai
chiều của dữ liệu sai
chỉ số sai
sai về kiểu
sai về phạm vi
dữ liệu cảm biến vượt ra ngoài miền cho phép

lỗi thừa, thiếu một số với biên
dữ liệu không tương thích

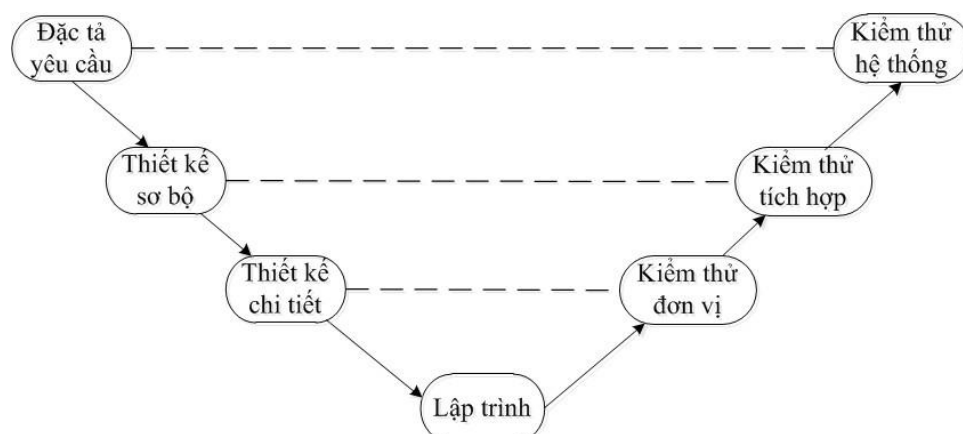
1.5. Các mức kiểm thử

Một trong các khái niệm then chốt của việc kiểm thử là các mức của việc kiểm thử. Các mức của việc kiểm thử phản ánh mức độ trừu tượng được thấy trong mô hình thác nước của vòng đời của việc phát triển phần mềm.

Dù có một số nhược điểm, mô hình này vẫn rất hữu ích cho việc kiểm thử, là phương tiện để xác định các mức kiểm thử khác nhau và làm sáng tỏ mục đích của mỗi mức. Một dạng của mô hình thác nước được trình bày trong hình 1.6. Dạng này nhấn mạnh sự tương ứng của việc kiểm thử với các mức thiết kế. Lưu ý rằng theo các thuật ngữ của việc kiểm thử hàm, ba mức của định nghĩa (đặc tả, thiết kế sơ bộ và thiết kế chi tiết) tương ứng trực tiếp với ba mức của việc kiểm thử là kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử hệ thống. Các mức của kiểm thử cũng làm nảy sinh vấn đề về thứ tự kiểm thử: dưới lên, trên xuống hoặc các khả năng khác

Các mức kiểm thử có thể được mô tả sơ bộ như sau:

- Kiểm thử đơn vị: Kiểm thử đơn vị là việc kiểm thử các đơn vị chương trình một cách cô lập. Thế nào là một đơn vị chương trình? Câu trả lời phụ thuộc vào ngữ cảnh công việc. Một đơn vị chương trình là một đoạn mã nguồn như hàm hoặc phương thức của một lớp, có thể được gọi từ ngoài, và cũng có thể gọi đến các đơn vị chương trình khác. Đơn vị cũng còn được coi là một đơn thể để kết hợp. Đơn vị chương trình cần được kiểm thử riêng biệt để phát hiện lỗi trong nội tại và khắc phục trước khi được tích hợp với các đơn vị khác.



Hình 1.6: Các mức trừu tượng và mức kiểm thử trong mô hình thác nước.

- Kiểm thử tích hợp: Mức kế tiếp với kiểm thử đơn vị là kiểm thử tích hợp. Sau khi các đơn vị chương trình để cấu thành hệ thống đã được kiểm thử, chúng cần được kết nối với nhau để tạo thành hệ thống đầy đủ và có thể làm việc. Công việc này không hề đơn giản và có thể có những lỗi về giao diện giữa các đơn vị, và cần phải kiểm thử để phát hiện những lỗi này. Công đoạn này gồm hai giai đoạn: giai đoạn kiểm thử tích hợp và giai đoạn kiểm thử hệ thống. Kiểm thử tích hợp nhằm đảm bảo hệ thống làm việc ổn định trong môi trường thí nghiệm để sẵn sàng cho việc đưa vào môi trường thực sự bằng cách đặt các đơn vị với nhau theo phương pháp tăng dần.
- Kiểm thử hệ thống: Kiểm thử mức này được áp dụng khi đã có một hệ thống đầy đủ sau khi tất cả các thành phần đã được tích hợp. Mục đích của kiểm thử hệ thống là để đảm bảo rằng việc cài đặt tuân thủ đầy đủ các yêu cầu được đặc tả của người dùng. Công việc này tốn nhiều công sức, vì có nhiều khía cạnh về yêu cầu người dùng cần được kiểm thử.
- Kiểm thử chấp nhận: Khi nhóm kiểm thử hệ thống đã thỏa mãn với một sản phẩm, sản phẩm đó đã sẵn sàng để đưa vào sử dụng. Khi đó hệ thống cần trải qua giai đoạn kiểm thử chấp nhận. Kiểm thử chấp nhận được thực thi bởi chính các khách hàng nhằm đảm bảo rằng sản phẩm phần mềm làm việc đúng như họ mong đợi. Có hai loại kiểm thử chấp nhận: kiểm thử chấp nhận người dùng, được tiến hành bởi người dùng, và kiểm thử chấp nhận doanh nghiệp, được tiến hành bởi nhà sản xuất ra sản phẩm phần mềm.

1.6 Các nguyên tắc kiểm thử phần mềm

Kiểm thử phần mềm là quá trình vận hành một chương trình nhằm tìm ra lỗi của nó. Để phần mềm hoạt động trơn tru, nó cần phải sạch lỗi. Và nếu kiểm thử phần mềm được thực hiện thành công, việc này sẽ khiến các lỗi không còn xuất hiện. Tuy nhiên, để tiết kiệm thời gian và công sức truy tìm các lỗi, có **7 nguyên tắc kiểm thử** quan trọng mà chúng ta cần tuân theo.

Nguyên tắc 1. Kiểm thử phần mềm chứng minh sự hiện diện của lỗi

Bằng việc kiểm thử, chúng ta có thể làm giảm lượng bugs khi áp dụng nhiều phương pháp kiểm thử lên phần mềm. Tuy nhiên khi được đưa lên môi trường thật, người dùng cuối hoàn toàn có thể thấy nhiều lỗi khác không tìm thấy trong quá trình kiểm thử. Kiểm

thử chứng minh được sản phẩm có lỗi nhưng không thể chứng minh rằng sản phẩm không còn lỗi. Điều này có nghĩa là, sẽ luôn có lỗi không được phát hiện trong phần mềm, ngay cả khi không tìm thấy lỗi, cũng không đồng nghĩa rằng phần mềm đúng hoàn toàn.

Nguyên tắc 2. Kiểm thử toàn bộ là không khả thi

Đúng vậy, rất khó để kiểm tra toàn bộ các module cũng như các tính năng, kết hợp với đầu vào và đầu ra trong suốt quá trình kiểm tra. Các sản phẩm phần mềm hiện nay cực kỳ đa dạng và phức tạp, được phát triển trên nhiều nền tảng, thêm vào đó, ngày càng có nhiều công nghệ mới, khả năng kết nối dữ liệu lớn... khiến việc kiểm thử toàn bộ gần như là không thể. Thay vì cố gắng kiểm thử toàn bộ, hãy xác định mức độ quan trọng và độ ưu tiên của các module để kiểm thử những phần cần thiết hoặc gặp nhiều nguy cơ hơn.

Nguyên tắc 3. Kiểm thử càng sớm càng tốt

Nguyên tắc kiểm thử sớm có nghĩa là việc kiểm thử cần được thực hiện càng sớm càng tốt trong vòng đời phát triển phần mềm. Vậy ở bước nào thì được coi là sớm? Nguyên tắc này cho thấy ta cần phát hiện bug ngay từ các bước đầu tiên như nghiên cứu yêu cầu (requirement) hay design. Phát hiện lỗi càng muộn, chi phí bỏ ra để xử lý càng cao. Vì vậy, việc thay đổi yêu cầu không đúng từ sớm sẽ khiến tốn ít chi phí để thay đổi tính năng hơn.

Nguyên tắc 4. Lỗi thường được phân bố tập trung

Nguyên lý về phân bố lỗi chỉ ra rằng, chỉ một số ít module chứa phần lớn số lỗi phát hiện được. Những module này thường là những thành phần, chức năng chính của hệ thống. Điều này cũng đúng theo nguyên lý Pareto: 80 – 20: 80% số lỗi tìm thấy ở chỉ 20% module. Bằng kinh nghiệm, các QA/ Tester có thể xác định được những module có tính rủi ro và nhiều lỗi như vậy, giúp tập trung tìm kiếm lỗi nhanh và hiệu quả hơn. Tuy nhiên, cách tiếp cận này cũng ẩn chứa vấn đề: nếu thực hiện kiểm thử tương tự lặp đi lặp lại, dễ thấy rằng những test case cũ sẽ khó tìm thêm được bug mới.

Nguyên tắc 5. Nghịch lý thuốc trừ sâu

Trong trồng trọt, nếu người nông dân sử dụng lặp lại một liều trừ sâu, các loại sâu bệnh sẽ dần dần thích nghi và trở nên “nhờn” với loại thuốc trừ sâu đó. Tương tự với kiểm

thử phần mềm, khi lặp đi lặp lại một test case, thì xác suất tìm được lỗi là rất thấp. Nguyên nhân là hệ thống hoàn thiện hơn, lỗi tìm thấy đã được sửa theo test case cũ. Để xử lý hiệu ứng “thuốc trừ sâu” này, test case cần được thường xuyên xem lại và chỉnh sửa, thêm nhiều test case mới để tìm lỗi mới (regression test).

Thêm vào đó, QA/ Tester cũng không nên phụ thuộc quá nhiều vào các kỹ thuật test sẵn có. Bạn cần liên tục cải tiến các phương pháp có sẵn để làm việc kiểm thử hiệu quả hơn.

Nguyên tắc 6. Kiểm thử phụ thuộc vào ngữ cảnh

Kiểm thử phụ thuộc vào ngữ cảnh, nói một cách đơn giản, là việc kiểm thử một trang thương mại điện tử sẽ phải khác cách test một ứng dụng đọc tin tức. Tất cả các phần mềm đều được phát triển theo cách khác nhau. Và việc áp dụng chung một “cách giải” là sai lầm. Bạn cần sử dụng cách tiếp cận khác nhau, phương thức, kỹ thuật test khác nhau, loại test phụ thuộc vào loại phần mềm/ ứng dụng/ website.

Nguyên tắc 7. Quan niệm sai lầm về việc “hết lỗi”

Một phần mềm sạch bug 99% vẫn có thể không sử dụng được. Đây là trường hợp phần mềm được kiểm thử bằng một requirement sai. Kiểm thử không chỉ để tìm ra lỗi, mà còn để kiểm tra xem phần mềm có đáp ứng được đúng nhu cầu hay không. Chính vì vậy, việc Không còn lỗi hay Hết lỗi là quan niệm sai lầm.

Quan điểm: “Nguyên tắc kiểm thử chỉ là để tham khảo, không có tính ứng dụng thực tế”?

Đây là quan điểm cực kỳ sai lầm. Các nguyên tắc kiểm thử giúp tạo ra một chiến lược kiểm thử rõ ràng và tạo ra những trường hợp kiểm thử sát sao, để bắt được bug. Những tester dày dặn kinh nghiệm áp dụng những nguyên tắc kiểm thử nhằm hướng đến độ họ không nghĩ rằng họ đang áp dụng chúng. Vì vậy, việc nguyên tắc kiểm thử không có tính ứng dụng thực tế là sai lầm.

1.7 Đảm bảo chất lượng phần mềm

Theo Daniel Galin, khái niệm đảm bảo chất lượng phần mềm được xác định như sau :

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

cầu chức năng kỹ thuật cũng như với các yêu cầu quản lý mà giữ cho lịch biểu và hoạt động trong phạm vi ngân sách.

1.7.1 Những mục tiêu đảm bảo chất lượng phần mềm

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

- Phát triển phần mềm (hướng tiến trình)

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

- Bảo trì phần mềm (hướng sản phẩm)

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

1.7.2 Phân loại yêu cầu phần mềm ứng với các yếu tố chất lượng phần mềm

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

những mô hình này chỉ bổ sung hay sửa đổi một vài yếu tố chất lượng. Theo McCall, các yếu tố chất lượng phần mềm được chia làm ba loại:

- Các yếu tố hoạt động của sản phẩm bao gồm tính chính xác, tin cậy, hiệu quả, tính toàn vẹn, sử dụng được

- Các yếu tố rà soát bao gồm tính bảo trì, linh hoạt, có thể test được

Các yếu tố chuyển giao bao gồm tính khả chuyển, có khả năng sử dụng lại, có khả năng giao tác.

Bài tập cuối chương

Bài 1.1. Các định nghĩa và thuật ngữ cơ bản về kiểm thử phần mềm?

Bài 1.2. Mô tả mỗi miền trong bảy miền trong hình 1.4?

Bài 1.3. Hãy vẽ biểu đồ Venn phản ánh khẳng định: ta đã không làm cái mà lẽ ra ta cần phải làm, và làm cái mà lẽ ra ta không được làm?

Bài 1.4. Phân loại lỗi?

Bài 1.5. Phân biệt các mức kiểm thử?

Bài 1.6. Trình bày các nguyên tắc kiểm thử?

Bài 1.7. Một trong các câu chuyện cũ về lĩnh vực phần mềm mô tả một nhân viên cấu hình viết một chương trình quản lý lương. Chương trình có chức năng kiểm tra số chứng minh thư của cán bộ và nhân viên trước khi đưa ra bản tính lương. Nếu có lúc người nhân viên này bị đuổi việc, chương trình sẽ tạo ra một mã độc gây hại cho cơ quan. Hãy bàn về tình trạng này theo các thuật ngữ trên đây về lỗi, sai, dạng thất bại và quyết định dạng kiểm thử nào là thích hợp.

Bài 1.8. Hãy so sánh hai cách tiếp cận kiểm thử hàm và kiểm thử cấu trúc?

Bài 1.9. Mục tiêu đảm bảo chất lượng phần mềm?

Bài 1.10. Phân loại yêu cầu phần mềm ứng với các yếu tố chất lượng phần mềm?

CHƯƠNG 2: QUY TRÌNH KIỂM THỬ PHẦN MỀM

Nội dung chính của chương

Nội dung chính của chương bao gồm:

- Quy trình kiểm thử phần mềm tổng quát
- Lập kế hoạch kiểm thử
- Chuẩn bị môi trường kiểm thử
- Xây dựng kế hoạch kiểm thử
- Ghi nhận và xử lý lỗi

Mục tiêu cần đạt được của chương

- Nắm được quy trình kiểm thử phần mềm
- Biết lập kế hoạch kiểm thử
- Biết cách ghi nhận và xử lý lỗi
- Áp dụng vào bài toán cụ thể

Bài 2: Quy trình kiểm thử phần mềm (Số tiết: 03 tiết)

2.1 Quy trình kiểm thử phần mềm tổng quát

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

Software Testing Life Cycle (STLC)



Hình 2.1 Quy trình kiểm thử phần mềm

1. **Requirement analysis** - Phân tích yêu cầu
2. **Test planning** - Lập kế hoạch kiểm thử
3. **Test case development** - Thiết kế kịch bản kiểm thử

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

5. Test execution - Thực hiện kiểm thử

6. Test cycle closure – Kết thúc chu kỳ kiểm thử

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

2.2 Phân tích quy trình kiểm thử phần mềm

2.2.1. Requirement analysis - Phân tích yêu cầu

Đầu vào

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

Hoạt động

- Phân tích yêu cầu là giai đoạn đầu tiên trong quy trình kiểm thử phần mềm.
- QA team sẽ thực hiện đọc hiểu, nghiên cứu và phân tích cụ thể các yêu cầu trong tài liệu đặc tả của dự án hoặc tài liệu khách hàng. Qua hoạt động này, QA team sẽ nắm bắt được các yêu cầu mà dự án đưa ra bao gồm yêu cầu kiểm thử chức năng/ phi chức năng nào.
- Ngoài ra, trong quá trình phân tích, nghiên cứu tài liệu, nếu có câu hỏi phát sinh hay đề xuất giải quyết, QA team sẽ đưa ra câu hỏi với các bên liên quan như BA(Business Analysis), PM(Project Manager), team leader, khách hàng để hiểu chính xác hơn về yêu cầu của sản phẩm. Những câu hỏi này sẽ được lưu trữ vào file Q&A(Question and Answer). Các câu hỏi nên được đưa ra dưới dạng Yes/No question hoặc các lựa chọn để tiết kiệm thời gian trả lời cũng như hỗ trợ đưa ra những gợi ý hay để xây dựng sản phẩm ngay từ đầu. Như vậy, đương nhiên là chúng ta không nên nêu ra những câu hỏi dạng là gì, như thế nào, tại sao,... Những câu hỏi như thế thường mất thời gian để giải thích và cũng khó có thể giải thích một cách chi tiết nhất có thể. Hơn nữa, đối với khách hàng không có sự hiểu biết về lĩnh vực phần mềm mà họ yêu cầu thì càng không thể trả lời

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

Đầu ra

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

2.2.2. Test planning - Lập kế hoạch kiểm thử

Đầu vào

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

Hoạt động

Dựa vào các tài liệu được cung cấp và cập nhật mới nhất, thông thường, test manager hoặc test leader sẽ là người lập kế hoạch kiểm thử cho cả QA team. Lập kế hoạch kiểm thử nhằm xác định một số yếu tố quan trọng sau:

- **Xác định phạm vi(Scope) dự án:** Dự án thực hiện trong thời gian bao lâu? Bao gồm những công việc gì cho từng khoảng thời gian xác định? Từ đó đưa ra lịch trình thực hiện cho từng công việc nhỏ sao cho phù hợp với toàn bộ đội dự án.
- **Xác định phương pháp tiếp cận:** Nói về cách tiếp cận để kiểm thử cho một đối tượng nào đó, thì phải dựa vào nhiều thứ, ví dụ như: Thời gian cho phép test có phù hợp với con số ước lượng, nhiều hay ít, yêu cầu chất lượng từ phía khách hàng thế nào? Cao, thấp hay khắc khe hay sao cũng được? Công nghệ / kỹ thuật sử dụng để phát triển ứng dụng này là gì? Lĩnh vực của hệ thống/sản phẩm đang được test (domain) là gì?...Từ đó, test manager có thể đưa ra những phương pháp và kế hoạch phù hợp nhất cho cả quá trình thực hiện dự án sao cho đúng với các tiêu chí chấp nhận của sản phẩm và kịp tiến độ với các mốc thời gian bàn giao, phát hành.
- **Xác định các nguồn lực**

Con người: Bao nhiêu người tham gia dự án, ai sẽ test phần nào, bao nhiêu tester tham gia? Tester và nhóm phát triển có kinh nghiệm về lĩnh vực này không?

Thiết bị: số lượng server, version, máy tính, mobile để thực hiện test là bao nhiêu.

- **Lên kế hoạch thiết kế công việc test:** Bản kế hoạch kiểm thử sẽ bao gồm các nội dung:
Liệt kê các chức năng cần kiểm thử.

Để thực hiện test chức năng này thì cần làm những công việc gì, trong thời gian bao lâu, cái nào thực hiện trước, cái nào thực hiện sau, ai là người thực hiện.

Xác định điều kiện bắt đầu: xác định những điều kiện tối thiểu để bắt đầu hoạt động kiểm thử cho từng chức năng.

Xác định điều kiện kết thúc : khi có những điều kiện nào thì sẽ kết thúc việc kiểm thử.

Đầu ra

Đầu ra của giai đoạn lập kế hoạch bao gồm các tài liệu như test plan, test estimation, test schedule.

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

Đầu vào

Đầu vào của giai đoạn thiết kế kịch bản kiểm thử là test plan, test estimation, test schedule, các tài liệu đặc tả đã được cập nhật.

Hoạt động

- **Review tài liệu:** Đầu tiên, các kiểm thử viên cần review lại tất cả các tài liệu để xác định công việc cần làm, các công việc có khác gì so với dự án trước khách hàng đưa cho, chức năng nào cần test, chức năng nào không cần test lại nữa. Từ đó, vừa có thể tiết kiệm thời gian mà vẫn đưa ra được một kịch bản kiểm thử đầy đủ và hiệu quả.
- **Viết test case/ check list:** Sau đó, tester bắt tay vào việc viết test case chi tiết dựa vào kế hoạch đã đưa ra và vận dụng các kỹ thuật thiết kế kịch bản kiểm thử. Test case cần bao phủ được tất cả các trường hợp kiểm thử có thể xảy ra cũng như đáp ứng đầy đủ các tiêu chí của sản phẩm. Đồng thời tester cũng cần đánh giá mức độ ưu tiên cho từng test case.
- **Chuẩn bị dữ liệu kiểm thử:** Cùng với việc tạo ra các test case chi tiết, đội kiểm thử cũng cần chuẩn bị trước các dữ liệu kiểm thử cho các trường hợp cần thiết như test data, test script.

- **Review test case/ check list:** Sau khi hoàn thành, các thành viên trong đội kiểm thử hoặc test leader cũng cần review lại test case đã tạo để có thể bổ sung, hỗ trợ lẫn nhau nhằm tránh những sai sót trong thiết kế test case và rủi ro về sau.

Đầu ra

Sau khi hoàn thành thiết kế kịch bản kiểm thử, đội kiểm thử sẽ có các tài liệu bao gồm: test design, test case, check list, test data, test automation script.

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

Đầu vào

Đầu vào của giai đoạn cài đặt môi trường kiểm thử là test plan, smoke test case, test data.

Hoạt động

- Việc cài đặt môi trường kiểm thử là giai đoạn cũng rất quan trọng trong vòng đời phát triển phần mềm. Môi trường kiểm thử sẽ được quyết định dựa trên những yêu cầu của khách hàng, hay đặc thù của sản phẩm ví dụ như server/ client/ network,...
- Tester cần chuẩn bị một vài test case để kiểm tra xem môi trường cài đặt đã sẵn sàng cho việc kiểm thử hay chưa. Đây chính là việc thực thi các smoke test case.

Đầu ra

Đầu ra của giai đoạn này là môi trường đã được cài đặt đúng theo yêu cầu, sẵn sàng cho việc kiểm thử và kết quả của smoke test case.

2.2.5. Test execution - Thực hiện kiểm thử

Đầu vào

Tài liệu đầu vào của giai đoạn này là test plan, test design, test case, check list, test data, test automation script.

Hoạt động

- Thực hiện các test case như thiết kế và mức độ ưu tiên đã đưa ra trên môi trường đã được cài đặt.
- So sánh với kết quả mong đợi sau báo cáo các bug xảy ra lên tool quản lý lỗi và theo dõi trạng thái của lỗi đến khi được sửa thành công.

- Thực hiện re-test để verify các bug đã được fix và regression test khi có sự thay đổi liên quan.
- Trong quá trình thực hiện kiểm thử, kiểm thử viên cũng có thể hỗ trợ, đề xuất cho cả đội dự án để có giải pháp hợp lý và kết hợp công việc hiệu quả.
- Đo và phân tích tiến độ: kiểm thử viên cũng cần kiểm soát chặt chẽ tiến độ công việc của mình bằng cách so sánh tiến độ thực tế với kế hoạch, nếu chậm cần phải điều chỉnh sao cho kịp tiến độ dự án, nếu nhanh cũng cần điều chỉnh vì có thể test lead lên kế hoạch chưa sát với thực tế dự án. Từ đó có thể sửa chữa test plan cần điều chỉnh để phù hợp với tiến độ dự án đưa ra.
- Report thường xuyên cho PM và khách hàng về tình hình thực hiện dự án: Cung cấp thông tin trong quá trình kiểm thử đã làm được những chức năng nào, còn chức năng nào, hoàn thành được bao nhiêu phần trăm công việc, báo cáo các trường hợp phát sinh sớm, tránh ảnh hưởng tiến độ công việc của cả ngày.

Đầu ra

Đầu ra của giai đoạn này là test results(kết quả kiểm thử), defect reports(danh sách các lỗi tìm được).

2.2.6. Test cycle closure – Kết thúc chu kỳ kiểm thử

Đầu vào

Đầu vào của giai đoạn đóng chu trình kiểm thử là bao gồm tất cả những tài liệu liên quan đã được tổng hợp, ghi chép và hoàn thiện đầy đủ trong suốt quy trình kiểm thử của dự án: tài liệu phân tích đặc tả yêu cầu, test plan, test results, defect reports, tài liệu Q&A,...

Hoạt động

- Đây là giai đoạn cuối cùng trong quy trình kiểm thử phần mềm.
- Ở giai đoạn này, QA team thực hiện tổng kết, báo cáo kết quả về việc thực thi test case, bao nhiêu case pass/ fail, bao nhiêu case đã được fix, mức độ nghiêm trọng của lỗi, bao nhiêu lỗi cao/ thấp, lỗi còn nhiều ở chức năng nào, dev nào nhiều lỗi. Chức năng nào đã hoàn thành test/ chưa hoàn thành test/ trễ tiến độ bàn giao.
- Đánh giá các tiêu chí hoàn thành như phạm vi kiểm tra, chất lượng, chi phí, thời gian, mục tiêu kinh doanh quan trọng.

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

Đầu ra

Đầu ra của giai đoạn này bao gồm các tài liệu: Test report, Test results(final)

2.3. Tổng quát

Như vậy, chúng ta đã tìm hiểu xong từng giai đoạn của quy trình kiểm thử phần mềm. Tóm lại, chúng ta cần ghi nhớ những thông tin chính sau đây:

Giai đoạn	Đầu vào	Hoạt động	Đầu ra	Người thực hiện
Requirement analysis	<ul style="list-style-type: none"> * Tài liệu đặc tả yêu cầu * Tài liệu khách hàng * Tài liệu thiết kế hệ thống 	Nghiên cứu, phân tích yêu cầu dự án	<ul style="list-style-type: none"> * Q&A document * Tài liệu báo cáo tính khả thi, phân tích rủi ro của việc kiểm thử phần mềm 	QA team
Test planning	<ul style="list-style-type: none"> * Tài liệu đặc tả yêu cầu(đã được cập nhật) * Tài liệu báo cáo tính khả thi, phân tích rủi ro của việc kiểm thử phần mềm 	<ul style="list-style-type: none"> * Xác định phạm vi dự án * Xác định phương pháp tiếp cận * Xác định nguồn lực * Lên kế hoạch thiết kế công việc test 	<ul style="list-style-type: none"> * Test Plan * Test Estimation * Test Schedule 	Test manager/ Test leader
Test case development	<ul style="list-style-type: none"> * Tài liệu đặc tả yêu cầu(đã được cập nhật) * Test plan * Test estimation * Test schedule 	<ul style="list-style-type: none"> * Review tài liệu * Viết test case/ check list * Chuẩn bị dữ liệu kiểm thử * Review test case/ check list 	<ul style="list-style-type: none"> * Test design * Test case/check list * Test data * Test automation script 	Tester

Giai đoạn	Đầu vào	Hoạt động	Đầu ra	Người thực hiện
Test environment set up	<ul style="list-style-type: none"> * Test plan * Smoke test case * Test data 	<ul style="list-style-type: none"> * Thiết lập môi trường kiểm thử (server/ client/ network,...) * Kiểm tra môi trường kiểm thử bằng các smoke test case 	<ul style="list-style-type: none"> * Môi trường kiểm thử * Kết quả của smoke test case 	Tester
Test execution	<ul style="list-style-type: none"> Test plan * Test design, test case, check list * Test data * Test automation script 	<ul style="list-style-type: none"> * Thực hiện kiểm thử phần mềm * So sánh với kết quả mong đợi và báo cáo các bug xảy ra lên tool quản lý lỗi * Thực hiện re-test để verify các bug đã được fix và regression test khi có sự thay đổi liên quan * Đo và phân tích tiến độ * Điều chỉnh, sửa chữa tài liệu tiến độ dự án theo tình hình thực tế * Report thường xuyên cho Project Manager và khách hàng về tình hình thực hiện dự án 	<ul style="list-style-type: none"> * Test results * Defect reports 	Tester
Test cycle closure	<ul style="list-style-type: none"> * Tài liệu đặc tả yêu cầu * Test plan * Test results * Defect reports * Tài liệu Q&A 	<ul style="list-style-type: none"> * Tổng kết, báo cáo kết quả về việc thực thi test case * Đánh giá các tiêu chí hoàn thành như phạm vi kiểm tra, chất lượng, chi phí, thời gian, mục tiêu kinh doanh quan trọng * Thảo luận và rút ra bài học kinh nghiệm 	<ul style="list-style-type: none"> * Test report * Test results (final) 	QA team

Bài tập cuối chương

Bài 2.1. Nêu quy trình kiểm thử phần mềm?

Bài 2.2. Phân tích các giai đoạn trong quy trình kiểm thử phần mềm?

Bài 2.3 Các bước lập kế hoạch kiểm thử?

Bài 2.4. Chuẩn bị môi trường kiểm thử gồm những nội dung gì?

Bài 2.5. Cách ghi nhận và xử lý lỗi?

Bài 2.6. Lập kế hoạch kiểm thử cho phần mềm Quản lý bán hàng?

Bài 2.7. Lập kế hoạch kiểm thử cho website dangkytinchi.ictu.edu.vn?

Bài 2.8. Lập kế hoạch kiểm thử cho phần mềm Quản lý nhân sự?

Bài 2.9. Lập kế hoạch kiểm thử cho phần mềm Quản lý thư viện?

Bài 2.10. Lập kế hoạch kiểm thử cho website thương mại điện tử?

CHƯƠNG 3: KIỂM THỬ HÀM

Nội dung chính của chương

Nội dung chính của chương bao gồm:

- Tổng quan về kiểm thử hàm
- Kiểm thử giá trị biên
- Kiểm thử lớp tương đương
- Kiểm thử bằng bảng quyết định
- Kiểm thử tổ hợp

Mục tiêu cần đạt được của chương

Biết sử dụng các kỹ thuật kiểm thử hàm để thiết kế testcase

Vận dụng vào kiểm thử hàm các phần mềm cụ thể.

Bài 3: Các phương pháp kiểm thử hàm (Số tiết: 03 tiết)

3.1 Tổng quan

Kiểm thử hàm (functional testing) là các hoạt động kiểm tra chương trình dựa trên tài liệu mô tả chức năng, yêu cầu phần mềm, hay còn gọi là *đặc tả chức năng* (functional specification). Đặc tả chức năng là tài liệu mô tả yêu cầu, hành vi của chương trình mong muốn. Tài liệu này là cơ sở để chúng ta tiến hành xây dựng các ca kiểm thử, thuật ngữ chuyên môn gọi là *thiết kế kiểm thử* (test design) và người thực hiện việc này là người thiết kế kiểm thử. Thiết kế kiểm thử là hoạt động chính trong kiểm thử hàm.

Cần nhấn mạnh rằng kiểm thử hàm là việc thiết kế các ca kiểm thử hàm của chương trình chỉ dựa trên đặc tả của chương trình mà không dựa trên việc phân tích mã nguồn của chương trình. Kiểm thử hàm còn được gọi với tên khác chính xác hơn là kiểm thử dựa trên đặc tả, hay kiểm thử hộp đen, hay kiểm thử chức năng. Còn việc thiết kế kiểm thử có dựa vào mã nguồn của chương trình được gọi là kiểm thử cấu trúc, hay kiểm thử hộp trắng.

Kiểm thử hàm là một kỹ thuật cơ bản giúp phát hiện nhiều lỗi mà kiểm thử hộp trắng không phát hiện được. Ví dụ dễ thấy nhất là các lỗi do cài đặt (implementation) không đầy đủ so với đặc tả. Nếu phần mềm thiếu hẳn một chức năng thì chúng ta dễ nhận thấy, nhưng nếu phần mềm thiếu một trường hợp của một chức năng thì việc này không dễ phát hiện nếu không kiểm tra kỹ chương trình so với đặc tả.

Tài liệu đặc tả dùng để thiết kế các ca kiểm thử có thể đơn giản là mô tả về chức năng của chương trình viết bằng ngôn ngữ của người sử dụng, nhưng thường là cả các tài liệu phân tích, thiết kế ở các mức độ chi tiết hơn với các hình vẽ, bảng biểu, sơ đồ làm rõ hành vi của chương trình vì chúng mô tả sát hơn các tính chất hàm của chương trình mà chúng ta sẽ dựa vào đó để thiết kế kiểm thử. Chúng ta gọi chung các tài liệu này là đặc tả chức năng.

Để nhận thấy do chỉ dựa trên đặc tả chức năng, kiểm thử hàm có ưu điểm là có thể thực hiện được sớm, trước khi cài đặt chương trình. Thời điểm tốt để chúng ta bắt đầu thiết kế kiểm thử là khi tài liệu đặc tả đã ổn định, nhưng cũng có nhiều qui trình phần mềm hiện đại đang đẩy việc này lên sớm hơn, ngay trong giai đoạn xác định yêu cầu đã lồng ghép xác định các ca kiểm thử. Các ca kiểm thử này thường được mô tả dưới dạng các ví dụ, vừa có tác dụng giải thích, làm rõ tài liệu đặc tả, vừa có tác dụng sử dụng luôn làm các kịch bản kiểm thử, các ca kiểm thử.

Thông thường kết quả của việc thiết kế kiểm thử hàm sẽ đưa ra *đặc tả ca kiểm thử*. Tương tự như khái niệm đặc tả chương trình là mô tả chương trình mong muốn, đặc tả kiểm thử mô tả các ca kiểm thử cần được tạo ra. Từ một đặc tả ca kiểm thử chúng ta có thể tạo ra nhiều ca kiểm thử cụ thể để tiến hành kiểm tra với chương trình. Ví dụ với một hàm giải phương trình bậc hai với đầu vào là ba hệ số là a, b, c , đặc tả ca kiểm thử cho nghiệm kép là với mọi a, b, c thỏa mãn $b^2 = 4ac$. Từ đặc tả này ta có thể tạo ra các ca kiểm thử cụ thể, ví dụ $a = 1, b = 2, c = 1$, và kết quả mong đợi là nghiệm kép -1 .

Bản chất của các đặc tả ca kiểm thử là mỗi đặc tả ca kiểm thử sẽ xác định một lớp hành vi của chương trình. Do đó việc thiết kế kiểm thử hàm là việc phân tách đặc tả các hành vi có thể của chương trình thành một số lớp mà mỗi lớp này sẽ có hành vi nhất quán hay tương tự. Với ví dụ hàm giải phương trình bậc hai trên chúng ta có thể chia đặc tả thành hành vi phương trình cho nghiệm kép, hành vi phương trình cho kết quả vô nghiệm, v.v.

Việc phân tích đặc tả sẽ giúp chúng ta xác định và phân tách được các lớp này. Bên cạnh đó việc phân tích cũng thường giúp chúng ta xác định những điểm không rõ ràng, không đầy đủ của đặc tả. Đây là quá trình phân tích tổng hợp nhiều nguồn thông tin, từ đặc tả, từ kiến thức, kinh nghiệm của người thiết kế kiểm thử, do đó nó

mang tính thủ công và cũng dễ mắc lỗi. Nhiều chuyên gia về thiết kế kiểm thử hàm nhiều khi vẫn bỏ sót một số ca kiểm thử quan trọng. Để giảm thiểu các sai sót này, chúng ta cần có phương pháp hệ thống giúp chia quá trình thiết kế này thành các bước cơ bản để thực hiện. Từ đó một số bước sẽ được tự động hóa để giảm công sức, giảm sai sót, và tăng được chất lượng công việc. Thực tế việc tự động hoàn toàn chưa thực hiện được và một số bước cơ bản vẫn phải thực hiện thủ công. Do đó kỹ năng và kinh nghiệm của người thiết kế kiểm thử đóng vai trò quyết định đến chất lượng của bộ kiểm thử.

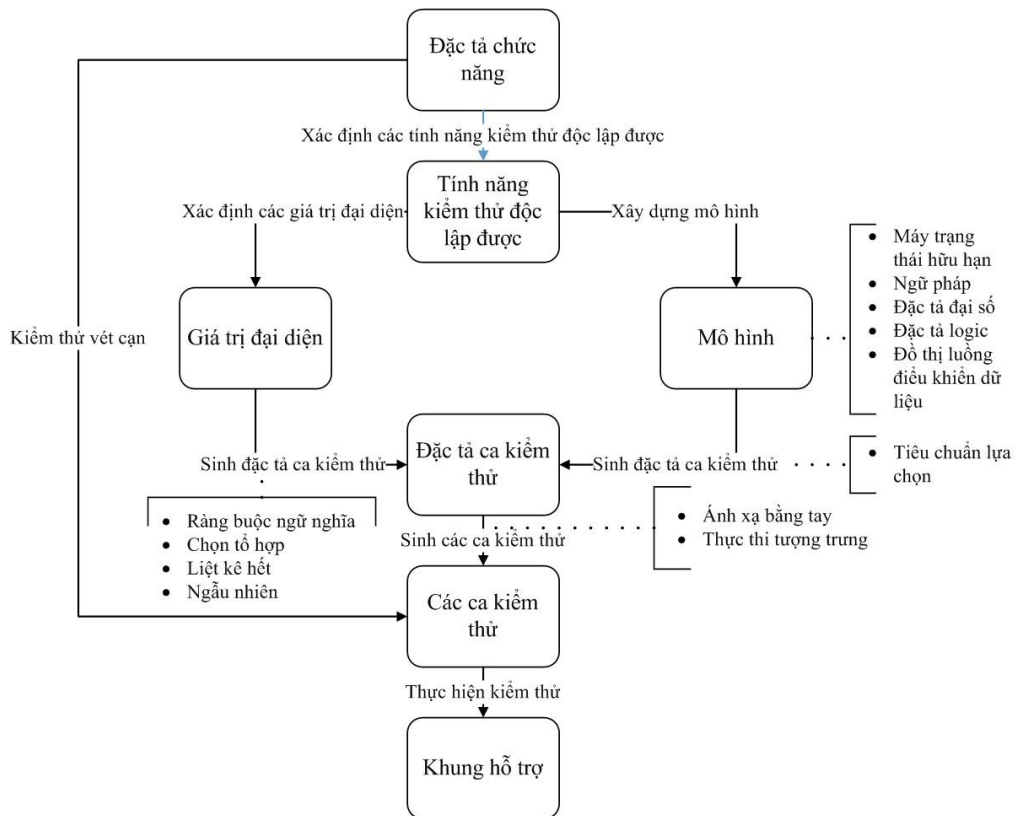
Để hiểu rõ hơn các công việc trong thiết kế kiểm thử hàm, chúng ta giả sử đang thiết kế kiểm thử cho chương trình P và giả sử $Y = P(X)$ trong đó X là vec-tơ của n biến đầu vào và Y là vec-tơ của m biến đầu ra. Dựa trên hiểu biết của chúng ta về đặc tả của chương trình P , với một vec-tơ đầu vào $X =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

3.1.1 Phương pháp hệ thống

Việc xác định các ca kiểm thử từ đặc tả chức năng là quá trình phân tích để xác định và chia không gian đầu vào/đầu ra/hành vi của chương trình thành các miền con. Phương pháp hệ thống sẽ chia quá trình này thành các bước cơ bản, từ đó làm từng bước được dễ dàng hơn hoặc tự động hóa một số bước nếu có thể. Phương pháp hệ thống giúp chúng ta xác định bộ ca kiểm thử đã đủ chưa đồng thời tránh tạo ra nhiều ca kiểm thử trùng lặp, không cần thiết.

Hình 3.1 mô tả các bước chính của một phương pháp hệ thống. Một số bước trong qui trình này có thể phức tạp, một số bước khá đơn giản và một số bước có thể bỏ qua tùy theo miền ứng dụng và đặc tả chương trình cũng như kinh nghiệm của người thiết kế. Dựa trên qui trình tổng quát này chúng ta có thể đưa ra các qui trình cụ thể bằng việc cụ thể hóa các bước trong qui trình. Chúng ta sẽ xem xét chi tiết

hơn các bước trong phương pháp này.



Hình 3.1: Các bước chính của phương pháp hệ thống cho kiểm thử hàm

Xác định các chức năng kiểm thử độc lập được: Đặc tả phần mềm trên thực tế là lớn và phức tạp ngay cả với những hệ thống nhỏ. Chúng thường được phân rã thành các đơn vị nhỏ hơn như các hệ thống con hay cụ thể hơn là các ca sử dụng (use case) hoặc các kịch bản sử dụng (user stories). Chúng ta gọi chung là các chức năng. Ví dụ trang web tìm kiếm của Google có thể có tính năng tìm kiếm, hiển thị kết quả, hiển thị quảng cáo, hay đăng ký người sử dụng mới, v.v.. Tuy nhiên ranh giới của một chức năng nhiều khi không rõ ràng. Ví dụ chức năng tìm kiếm của Google Search có chức năng tính toán một biểu thức số học nếu văn bản tìm kiếm là một biểu thức số học.

Khi thiết kế kiểm thử chúng ta tách các chức năng càng chi tiết, cụ thể càng tốt vì nó vừa giúp đơn giản việc thiết kế kiểm thử và vừa cho phép chúng ta xác định lỗi trong chương trình dễ dàng hơn. Mỗi ca kiểm thử chỉ nên kiểm tra một kịch bản cụ thể, chi tiết của một chức năng của hệ thống.

Tiếp đó với mỗi chức năng chúng ta sẽ xác định tất cả các đầu vào có ảnh hưởng đến việc thực hiện chức năng đó và các đầu ra có thể bị tác động bởi chức năng đó.

Các đầu vào và đầu ra thường được mô tả tường minh trong tài liệu đặc tả. Nhưng cũng có nhiều trường hợp các đầu vào và đầu ra này nằm ẩn trong tài liệu đặc tả mà người thiết kế phải phát hiện ra. Ví dụ như các đặc tả không hình thức mô tả chức năng đăng ký tài khoản mới thường chỉ nêu các trường dữ liệu cần khai báo nhưng không nói gì đến chi tiết kiểu dữ liệu sẽ lưu thông tin hồ sơ người sử dụng. Hay một ví dụ khác khá rắc rối với tên tiếng Việt khi chúng ta chỉ có hai trường họ và tên để nhập vào. Một là thông thường chúng ta không nói ra cụ thể chiều dài tối đa, tối thiểu, hay qui định bảng mã ký tự của các trường này. Hai là chúng ta không nói rõ phần tên đệm sẽ nằm ở phần họ hay phần tên, nên khi người sử dụng nhập dữ liệu sẽ xảy ra tình trạng mỗi người nhập một kiểu. Tên là Nguyễn Văn Hùng thì có người nhập phần họ là Nguyễn, phần tên là Văn Hùng; có người lại nhập họ là Nguyễn Văn, tên là Hùng; lại có người chỉ nhập họ là Nguyễn, tên là Hùng, vì nghĩ phần Văn là tên đệm không hỏi thì không nhập.

Xác định các lớp giá trị đại diện: Các giá trị đại diện có thể được xác định trực tiếp từ đặc tả phi hình thức, như ngôn ngữ tự nhiên. Chúng cũng có thể được xác định gián tiếp dựa trên một mô hình được xây dựng để phục vụ cho việc này. Trong cả hai trường hợp này mục tiêu đều là xác định các giá trị của mỗi đầu vào một cách độc lập, bằng việc lấy các giá trị đã liệt kê tường minh trong đặc tả, hoặc thông qua một mô hình phù hợp. Chúng ta chưa xét đến tổ hợp của các giá trị này. Lý do là chúng ta đang tách bài toán xác định các lớp giá trị của mỗi đầu vào và bài toán tổ hợp chúng để tạo thành các ca kiểm thử. Tức là chúng ta đang tách một bước phức tạp thành các bước đơn giản hơn.

Với các đặc tả không hình thức người thiết kế kiểm thử phải dựa vào việc liệt kê tường minh các lớp giá trị đại diện. Trong trường hợp này người thiết kế kiểm thử cần cẩn thận xét tất cả các trường hợp có thể và khai thác tối đa những thông tin có sẵn trong đặc tả. Tương tự, với kết quả mong đợi đầu ra, chúng ta cũng có thể xác định các lớp giá trị của miền đầu ra, cũng như các giá trị biên và các giá trị đặc biệt để có lỗi.

Ví dụ khi xem xét các thao tác trên một danh sách không rỗng chúng ta cần xét cả trường hợp danh sách rỗng (giá trị lỗi) và danh sách có một phần tử (giá trị biên chiều dài danh sách) là các trường hợp đặc biệt. Ở bước này chúng ta chỉ quan tâm đến tính chất của giá trị, chứ chưa cần cụ thể giá trị là gì. Tức là chúng ta quan tâm

đặc tả ca kiểm thử chứ chưa phải ca kiểm thử cụ thể. Với ví dụ danh sách một phần tử, chúng ta chưa cần quan tâm phần tử của danh sách này cụ thể là gì, chỉ biết danh sách chỉ có một phần tử.

Để liệt kê các giá trị không tường minh chúng ta cần xây dựng mô hình hoặc một phần của mô hình của đặc tả. Ví dụ chúng ta có thể dùng văn phạm để mô tả ngôn ngữ và sinh các từ của ngôn ngữ này để liệt kê các giá trị cần xác định. Mô hình này có thể có sẵn trong đặc tả nhưng thông thường người thiết kế kiểm thử phải xây dựng chúng.

Trong hai cách trên, việc liệt kê trực tiếp có vẻ đơn giản hơn và ít tốn kém hơn việc xây dựng mô hình để sinh ra các giá trị. Tuy nhiên về lâu về dài các mô hình sẽ có giá trị hơn, khi khối lượng giá trị lớn chúng ta có thể tự động hóa việc liệt kê, hay chúng ta cũng có thể điều chỉnh số lượng ca kiểm thử nhiều hay ít để cân bằng với các ràng buộc kiểm thử khác, như chi phí, thời gian. Các mô hình cũng có thể được sử dụng lại, sửa đổi dễ dàng hơn trong quá trình phát triển hệ thống. Quyết định cuối cùng nên dùng cách nào vẫn tùy thuộc vào lĩnh vực ứng dụng, kỹ năng của người thiết kế kiểm thử, và các công cụ thích hợp đang có.

Sinh các đặc tả ca kiểm thử: Đặc tả kiểm thử được tạo ra bằng việc tổ hợp các giá trị của các đầu vào của chương trình đang được kiểm thử. Bước trên chúng ta đã xác định các giá trị đại diện thì ở bước này chúng ta sinh các tổ hợp của chúng bằng tích Đề-các của các giá trị đại diện đã chọn. Nếu ở bước trước chúng ta tạo ra mô hình hình thức thì ở bước này các đặc tả ca kiểm thử sẽ là các hành vi cụ thể hoặc tổ hợp của các tham số của mô hình và một đặc tả ca kiểm thử có thể được thỏa mãn bởi nhiều đầu vào cụ thể. Trong cả hai cách này việc tạo ra tất cả các tổ hợp thường tạo ra số lượng quá lớn các ca kiểm thử.

Ví dụ một hàm có năm đầu vào, mỗi đầu vào có sáu giá trị đại diện thì tích Đề-các sẽ tạo ra $6^5 = 7,776$ đặc tả ca kiểm thử. Số lượng này là quá lớn với một chức năng, ngay cả khi được tự động hóa và chạy bằng máy tính hoàn toàn. Thông thường, rất nhiều tổ hợp là không hợp lệ hoặc không khả thi. Chúng ta sẽ học các phương pháp để giảm số tổ hợp này nhưng vẫn đảm bảo chất lượng của bộ kiểm thử.

Để giảm số ca kiểm thử, chúng ta cần loại bỏ các tổ hợp giá trị không hợp lệ bằng việc đưa thêm các ràng buộc về cách tổ hợp. Ví dụ hàm *NextDate* chúng ta cần đưa

ràng buộc để tránh tạo ra các giá trị không hợp lệ như ngày 31/11/2013. Hoặc chúng ta cũng không xét hết tất cả các tích Đề-các mà chỉ cần mỗi cặp giá trị của hai biến bất kỳ đều xuất hiện là đủ. Quá trình này có thể không phải một bước làm là xong mà có thể cần điều chỉnh và thử một số lần để có kết quả là bộ ca kiểm thử phù hợp, vừa hạn chế được các ca kiểm thử vô lý không cần thiết, vừa tiết kiệm được chi phí thực hiện kiểm thử.

Sinh ca kiểm thử và thực hiện kiểm thử: Quá trình kiểm thử hoàn tất khi chúng ta cụ thể hóa các đặc tả kiểm thử thành các ca kiểm thử cụ thể, rồi tính kết quả mong đợi của từng ca kiểm thử, và tiến hành kiểm tra với phần mềm. Việc thực hiện kiểm thử cần được tự động hóa tối đa khi có thể, vì một hệ thống phần mềm cần được kiểm thử rất nhiều lần. Ngay cả khi sản phẩm hoàn tất, trong tương lai chúng ta vẫn sẽ cần thực hiện kiểm thử nhiều lần nữa, khi phần mềm tiến hóa, thay đổi theo những yêu cầu mới, hay các lỗi được phát hiện và phần mềm cần phải sửa.

3.1.2 Lựa chọn phương pháp phù hợp

Trong các chương sau chúng ta sẽ thấy một số kỹ thuật kiểm thử hàm thích hợp cho các loại đặc tả chương trình khác nhau. Với một đặc tả có thể có một số kỹ thuật phù hợp để xây dựng các ca kiểm thử và một số kỹ thuật sẽ rất khó áp dụng hoặc kết quả sẽ không thỏa mãn. Một số kỹ thuật có thể cho kết quả như nhau với một số đặc tả. Một số kỹ thuật lại bổ sung cho nhau nên cần kết hợp chúng. Ví dụ như khi mỗi kỹ thuật lại áp dụng tốt cho những khía cạnh khác nhau của cùng một đặc tả, hoặc ở các giai đoạn khác nhau của việc sinh ca kiểm thử.

Việc lựa chọn kỹ thuật để tạo ra các ca kiểm thử phụ thuộc vào các yếu tố: tính tự nhiên của đặc tả, dạng của đặc tả, kinh nghiệm của người thiết kế kiểm thử, cấu trúc của tổ chức, công cụ sẵn có, ngân sách và các ràng buộc về chất lượng, chi phí thiết kế và cài đặt các hệ thống hỗ trợ.

Tính tự nhiên và dạng của đặc tả: Các kỹ thuật kiểm thử khai thác các đặc điểm khác nhau của đặc tả. Ví dụ, khi biến đầu vào có một số ràng buộc thì có thể kỹ thuật kiểm thử lớp tương đương sẽ phù hợp. Nếu có sự chuyển dịch trong các trạng thái của hệ thống thì phương pháp kiểm thử dựa trên máy trạng thái có thể nên áp dụng. Hay khi các đầu vào thay đổi với kích thước không xác định được thì phương pháp dùng ngữ pháp có thể phù hợp.

Kinh nghiệm của người thiết kế kiểm thử: Kinh nghiệm của người kiểm thử nhiều khi sẽ ảnh hưởng đến kỹ thuật kiểm thử nào được sử dụng. Ví dụ một người thiết kế kiểm thử giỏi về bảng quyết định sẽ thích kỹ thuật đó khi một kỹ thuật khác cũng có thể áp dụng được.

Công cụ: Một số kỹ thuật đòi hỏi sử dụng các công cụ mà đôi khi phải mua nên chi phí cũng là một yếu tố ảnh hưởng đến việc lựa chọn kỹ thuật kiểm thử. Ví dụ đặc tả sử dụng UML và có công cụ thương mại sinh ca kiểm thử dựa trên một số biểu đồ UML thì công cụ đóng vai trò quan trọng ảnh hưởng đến phương pháp thực hiện kiểm thử, nếu ngân sách cho kiểm thử cho phép. Một số đơn vị làm phần mềm chuyên nghiệp đôi khi phải xây dựng các công cụ này để phục vụ cho đơn vị, hoặc đôi khi cho từng dự án cụ thể.

Ngân sách và ràng buộc chất lượng: Ngân sách và ràng buộc chất lượng có thể dẫn đến sự lựa chọn khác nhau về phương pháp kiểm thử. Ví dụ nếu yêu cầu là nhanh, kiểm thử tự động, nhưng không cần độ tin cậy cao thì kiểm thử ngẫu nhiên là phù hợp. Ngược lại, yêu cầu độ tin cậy phải rất cao thì chúng ta phải sử dụng các phương pháp tinh vi để kiểm thử một cách kỹ lưỡng nhất có thể. Khi chọn một phương pháp, điều quan trọng là phải đánh giá tất cả các yếu tố và chi phí liên quan.

Chi phí nền tảng kiểm thử: Mỗi đặc tả ca kiểm thử sẽ phải được cụ thể hóa thành một ca kiểm thử và được chạy đi chạy lại nhiều lần trong suốt vòng đời của sản phẩm. Mỗi lần chạy một ca kiểm thử bao gồm việc phải đưa đầu vào vào chương trình, thực hiện chương trình, và so sánh kết quả đầu ra thực tế với đầu ra mong đợi. Nếu chúng ta tự động hóa được tất cả các công việc này thì phương pháp tổ hợp sinh ra tất cả các tổ hợp có thể áp dụng. Tuy nhiên nếu một số bước phải thực hiện bằng tay, hoặc chúng ta phải viết mã cho từng đặc tả kiểm thử này thì có lẽ chúng ta phải chọn phương pháp khác để khả thi về mặt thời gian và công sức.

Tóm lại kiểm thử hàm cũng giống nhiều hoạt động kỹ thuật khác đòi hỏi phải phân tích kỹ lưỡng các ưu nhược điểm để chọn phương pháp cân bằng phù hợp với các khó khăn và các vấn đề không lường hết được mà phải là người thiết kế có kinh nghiệm mới có cảm giác tốt được. Kiểm thử hàm không phải là việc tập để chọn một phương pháp tối ưu mà là một loạt các hoạt động để tìm ra sự tổ hợp của một số mô hình và kỹ thuật để tạo ra một tập các ca kiểm thử thỏa mãn chi phí và ràng

buộc chất lượng đặt ra. Sự cân bằng này cũng có thể mở rộng ra ngoài thiết kế kiểm thử sang thiết kế phần mềm để thuận lợi cho việc kiểm thử. Thiết kế phần mềm thích hợp sẽ không chỉ giúp quá trình phát triển được tốt hơn mà còn giúp việc kiểm thử cũng thuận lợi để giúp tiết kiệm chi phí tổng thể của dự án phần mềm.

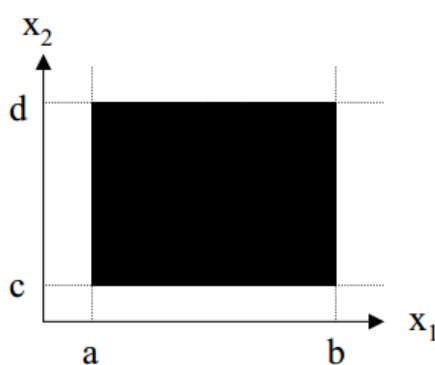
3.2 Kiểm thử giá trị biên

Kiểm thử giá trị biên (boundary testing) là một trong những kỹ thuật được áp dụng phổ biến nhất trong kiểm thử hàm. Chúng ta sẽ coi một chương trình là một hàm toán học với đầu vào của chương trình tương ứng với các tham số của hàm và đầu ra của chương trình là giá trị trả về của hàm. Vì hàm toán học là ánh xạ từ miền xác định của hàm đến miền giá trị của hàm. Chúng ta sẽ tập trung vào các giá trị biên và cạnh biên của hai miền đầu vào và đầu ra này của hàm để xây dựng các ca kiểm thử. Khi chúng ta dùng biên đầu ra tức là chúng ta cho các kết quả mong đợi nằm ở trên biên và cạnh biên đầu ra.

Giả sử $y = f(x_1, x_2)$ với $x_1, x_2, y \in \mathbb{N}$ là một hàm toán học của một chương trình. Khi đó thông thường x_1 và x_2 có miền xác định thể hiện bằng các biên. Ví dụ:

$$a \leq x_1 \leq b \text{ và } c \leq x_2 \leq d$$

trong đó a, b, c, d là các hằng số nào đó. Hình 3.2 thể hiện miền xác định của hai biến này.

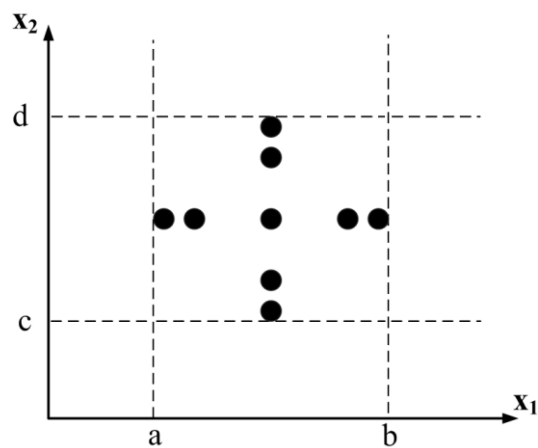


Hình 3.2: Miền xác định của hàm hai biến.

Ý tưởng kiểm thử giá trị biên xuất phát từ quan sát nhiều lỗi xảy với các giá trị biên này, khi chúng ta không kiểm tra khoảng giá trị hợp lệ của dữ liệu được nhập vào, hay do lỗi lập trình hoặc đặc tả làm các biểu thức điều kiện không chính xác. Ví dụ đúng ra phải là dấu \leq nhưng người lập trình hoặc đặc tả lại chỉ viết $<$ hoặc ngược lại. Kiểm thử với các giá trị biên giúp chúng ta phát hiện các lỗi này. Trong

nhiều trường hợp các biên này là ẩn, không được viết rõ ra trong yêu cầu nên lập trình viên dễ mắc lỗi không kiểm tra các giá trị đầu vào, hoặc không kiểm tra kết quả có hợp lệ không trước khi trả về.

Để tăng khả năng phát hiện lỗi, kiểm thử giá trị biên thường lấy năm ca kiểm thử cho mỗi biến là các giá trị: cực đại, cực tiểu, các giá trị cạnh chúng trong miền xác định (gọi là cận biên hoặc cạnh biên), và một giá trị ở giữa miền xác định đại diện cho giá trị thông thường. Chúng ta đánh chỉ số cho chúng lần lượt là *max*, *min*, *min*, *max+* và *nom*. Hình 3.3 minh họa các giá trị này.



Hình 3.3: Các ca kiểm thử phân tích giá trị biên cho một hàm hai biến.

Thông thường lỗi chương trình xảy ra ngay khi có một sai sót trong phần mềm, chứ không cần kết hợp của nhiều sai sót mới gây ra lỗi. Khi đó các ca kiểm thử theo phương pháp kiểm thử giá trị biên được xây dựng bằng cách lấy một bộ giá trị *nom* của các biến, rồi lần lượt thay mỗi giá trị đó của từng biến bằng giá trị biên và cận biên để tạo ra ca kiểm thử mới. Ví dụ với hàm *f* trên ta có bộ kiểm thử sau, thể hiện trên Hình 3.3.

$$\begin{aligned} & \{(x_{1nom}, x_{2nom}), \\ & (x_{1min}, x_{2nom}), (x_{1min+}, x_{2nom}), \\ & (x_{1max-}, x_{2nom}), (x_{1max}, x_{2nom}), \\ & (x_{1nom}, x_{2min}), (x_{1nom}, x_{2min+}), \\ & (x_{1nom}, x_{2max-}), (x_{1nom}, x_{2max})\} \end{aligned}$$

Tổng quát hóa kiểm thử giá trị biên cho hàm *n* biến số và mỗi biến có các giá trị biên và cận biên khác nhau ta có thể dễ thấy sẽ có $1 + 4n$ ca kiểm thử vì xuất phát từ một ca kiểm thử gồm các giá trị trung bình của các biến, ta thay nó bằng bốn giá trị biên và cận biên: *min*, *min+*, *max*, và *max*. Với *n* biến sẽ tạo thêm $4n$ bộ kiểm thử. Do

đó tổng số ca kiểm thử là $1 + 4n$. Tuy nhiên tùy theo miền xác định của biến mà số lượng này thực tế có thể ít hơn. Ví dụ biến nguyên thuộc khoảng $[1, 2]$ thì không có cận biên và không có giá trị ở giữa.

Với cách tạo bộ kiểm thử giá trị biên này, mỗi giá trị biên và cận biên xuất hiện một lần với các giá trị trung bình của các biến còn lại, chứ không phải tổ hợp các bộ giá trị của các biên. Tổ hợp của các biên được gọi là kiểm thử giá trị biên mạnh sẽ được thảo luận trong phần tiếp theo.

Trong thực tế, miền xác định của các biến không chỉ là các miền số, nên chúng ta cần có các vận dụng thích hợp để xác định các giá trị biên tinh tế hơn. Ví dụ trong chương trình NextDate biến tháng month có thể là số nguyên trong khoảng từ 1 đến 12. Nhưng không phải ngôn ngữ lập trình nào cũng cho phép khai báo miền xác định của biến trong khoảng này. Chúng ta cũng có thể dùng kiểu liệt kê (enumeration) để khai báo các tháng bằng tên Januray, ..., December và chúng ta vẫn có giá trị biên và cận biên. Tuy nhiên như trong ngôn ngữ lập trình Java chúng ta phải dùng biến kiểu nguyên int và không khai báo miền giá trị cho chúng được, thì chúng ta sẽ phải dùng các cận của kiểu làm biên. Trong Java, biên của số nguyên là Interger.MIN_VALUE và Interger.MAX_VALUE. Trong C, chúng ta cũng có các khoảng giá trị này.

Với biến kiểu Boolean, các giá trị của miền chỉ có TRUE và FALSE, chúng ta không có cận biên và giá trị ở giữa. Với kiểu xâu ký tự, mảng, hay danh sách, tập hợp (collection), chúng ta có thể dựa vào chiều dài/kích thước để làm các giá trị biên. Với các kiểu cấu trúc dữ liệu khác chúng ta có thể xây dựng các giá trị biên và cận biên dựa theo các thành phần của cấu trúc.

Ưu nhược điểm: Khi các biến của hàm là độc lập, không có quan hệ ràng buộc lẫn nhau, thì kiểm thử giá trị biên tỏ ra hiệu quả. Nhưng khi chúng có quan hệ phụ thuộc nào đó thì phương pháp này dễ tạo ra các ca kiểm thử không hợp lý, vì các giá trị cực đại, cực tiểu có thể không kết hợp với nhau để tạo thành ca kiểm thử hợp lệ.

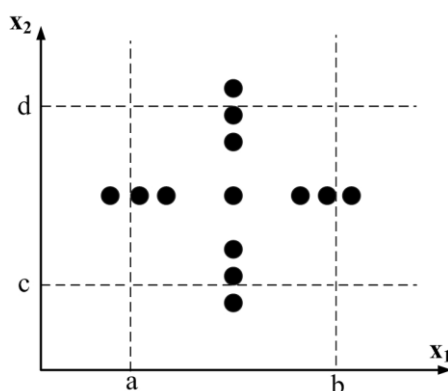
Ví dụ khác với hàm NextDate thì biến ngày, tháng và năm có ràng buộc. Một số tháng có 31 ngày, một số lại có 30 ngày. Năm nhuận lại có ràng buộc với số ngày trong tháng hai. Nếu không xét sự phụ thuộc này kiểm thử giá trị biên không tạo ra được ca kiểm thử ngày 28 tháng 2 năm 2012, vì 28 không là biên và cận biên của khoảng ngày từ 1 đến 31.

Kiểm thử giá trị biên cũng không quan tâm đến tính chất, đặc trưng của hàm, đồng thời cũng không xét đến ngữ nghĩa của biến hay quan hệ giữa các biến. Nó chỉ máy móc lấy các giá trị biên, cận biên và trung bình để tổ hợp tạo ra các ca kiểm thử.

Ưu điểm của phương pháp này là đơn giản và có thể tự động hóa việc sinh các ca kiểm thử khi chúng ta đã xác định được các biên của các biến. Với nhiều kiểu dữ liệu cơ bản có sẵn chúng ta cũng có thể xây dựng sẵn các biên này để sử dụng.

3.2.1 Kiểm thử giá trị biên mạnh

Kiểm thử giá trị biên mạnh là mở rộng của kiểm thử giá trị biên bằng việc bổ sung các giá trị cận biên bên ngoài miền xác định. Ngoài 5 giá trị biên đã nêu ở phần trước chúng ta lấy thêm các giá trị cận biên ở ngoài miền xác định là $max+$ và min như trong Hình 3.4. Các ca kiểm thử này sẽ giúp ta kiểm tra chương trình với dữ liệu không hợp lệ, nằm ngoài khoảng mong đợi. Ví dụ khi nhập ngày 32/1/2013 chương trình cần có thông báo lỗi thích hợp.

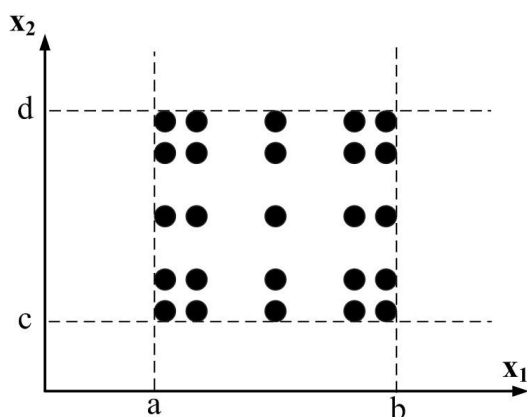


Hình 3.4: Các ca kiểm thử mạnh cho hàm hai biến.

Các ca kiểm thử mạnh này giúp chúng ta kiểm tra xem chương trình có xử lý các ngoại lệ hay kiểm tra biến đầu vào hay không. Với các ngôn ngữ lập trình không kiểm tra kiểu khi biên dịch hay dữ liệu được nhập vào bên ngoài, do người sử dụng đưa vào hoặc lấy từ hệ thống khác, việc này là rất cần thiết. Rất nhiều lập trình viên mới vào nghề thường xuyên không kiểm tra dữ liệu đầu vào nên chương trình chỉ chạy với dữ liệu nhập đúng như suy nghĩ của người lập trình. Hậu quả của việc này không chỉ đơn thuần là chương trình bị sai, mà thường gây vi phạm bộ nhớ, dẫn đến tắt chương trình (crash), và là một trong những lỗ hổng an ninh dễ bị khai thác.

3.2.2 Kiểm thử giá trị biên tổ hợp

Ở trên chúng ta chỉ tạo các ca kiểm thử với giá trị biên và cận biên cho từng biến. Nếu chúng ta mở rộng với hai hoặc với tất cả các biến đều được đẩy đến giá trị biên và cận biên thì chúng ta sẽ tạo ra được các ca kiểm thử giá trị biên tổ hợp. Tức là từ bộ giá trị 5 phần tử $min, min+, nom, max$ và max của mỗi biến ta lấy tích Đề-Các (Cartesian) của chúng để tạo ra các ca kiểm thử. Với hai biến Hình 3.5 minh họa các ca kiểm thử này.



Hình 3.5: Các ca kiểm thử biên tổ hợp của hàm hai biến.

Có thể thấy cách tổ hợp các biên và cận biên này sẽ kiểm tra kỹ hơn kiểm thử giá trị biên thông thường. Tuy nhiên số ca kiểm thử theo cách này tăng lên đáng kể, lên đến 5^n ca kiểm thử so với $4n + 1$ ca theo kiểm thử biên thông thường.

Tương tự như kiểm thử giá trị biên mạnh, ta có thể mở rộng kiểm thử biên tổ hợp với bộ 7 giá trị của kiểm thử giá trị biên mạnh. Chúng ta sẽ kiểm tra được kỹ hơn nhưng cũng mất nhiều công sức hơn, đến 7^n ca kiểm thử.

3.2.3 Kiểm thử các giá trị đặc biệt

Kiểm thử các giá trị đặc biệt cũng là một phương pháp phổ biến. Đây cũng là phương pháp trực quan nhất và không theo một khuôn dạng cụ thể nào. Dựa trên hiểu biết về bài toán và miền ứng dụng kết hợp với kinh nghiệm cá nhân, người kiểm thử đưa ra các giá trị kiểm thử. Do đó không có hướng dẫn cụ thể nào cho phương pháp này. Mức độ hiệu quả của phương pháp này phụ thuộc nhiều vào khả năng của người kiểm thử. Trên thực tế các đơn vị phát triển phần mềm vẫn áp dụng phương pháp này, vì nhiều khi nó giúp phát hiện lỗi nhanh, không tốn nhiều công sức.

Ví dụ dựa trên hiểu biết về số ngày của các tháng trong một năm chúng ta sẽ kiểm thử các ngày 28 tháng 2 và 29 tháng 2 ở cả năm nhuận và năm thường. Số lượng ca kiểm thử sẽ ít hơn nhiều so với kiểm thử giá trị biên và cũng hiệu quả trong việc kiểm tra lỗi. Đây cũng là tính chất “thủ công” của nghề kiểm thử phần mềm – “khéo tay” và có kinh nghiệm sẽ làm tốt hơn.

❖ Kinh nghiệm áp dụng

Kiểm thử giá trị biên là phương pháp “thô” nhất trong các phương pháp kiểm thử hàm. Phương pháp này nên áp dụng cho các hàm có các biến đầu vào độc lập, không phụ thuộc vào nhau. Khi chúng ta có giả định một khiếm khuyết trong chương trình sẽ gây ra lỗi ngay thì chúng ta sử dụng kiểm thử biên thông thường. Trái lại khi lỗi thường xuất hiện khi có hơn một khiếm khuyết trong chương trình thì chúng ta cần sử dụng kiểm thử giá trị biên mạnh. Các phương pháp này đều đơn giản, dễ dàng tự động hóa việc sinh và chạy các ca kiểm thử này.

Phần này chúng ta chủ yếu tập trung vào biên của biến đầu vào. Chúng ta có thể mở rộng cho biến đầu ra, tức là kết quả của hàm, sao cho các kết quả này có thể nằm ở biên, cận biên. Ngoài ra trong chương trình máy tính còn có các biến cục bộ, hay các biến toàn cục khác. Chúng ta có thể vận dụng mở rộng cho các biến này để có thể phát hiện thêm các lỗi tinh vi khác trong chương trình.

3.3 Kiểm thử lớp tương đương

Ý tưởng của kiểm thử lớp tương đương là chia miền đầu vào thành các phần sao cho các giá trị trong mỗi phần có tác động tương tự với chương trình. Khi đã chia được thành các miền con như vậy, ta chỉ cần lấy một giá trị bất kỳ trong mỗi miền con để xây dựng bộ kiểm thử. Phương pháp kiểm thử lớp tương đương này cho ta cảm giác đã kiểm thử đầy đủ hệ thống đồng thời cũng không có các ca kiểm thử trùng lặp. Chúng ta cũng có thể áp dụng lớp tương đương cho miền đầu ra.

Kiểm thử lớp tương đương là phương pháp chia miền dữ liệu kiểm thử thành các miền con sao cho dữ liệu trong mỗi miền con có cùng tính chất đối với chương trình, có nghĩa là các ca kiểm thử của một miền con sẽ cùng gây lỗi cho chương trình, hay cùng cho kết quả đúng, hay cùng cho kết quả sai tương tự nhau. Sau khi chia miền dữ liệu của chương trình thành và miền con tương đương, chúng ta chỉ cần chọn một phần tử đại diện của mỗi miền con này làm bộ dữ liệu kiểm thử. Các miền con này

chính là các lớp tương đương.

Ví dụ trong bài toán Triangle ta lấy một lớp tương đương là tất cả các tam giác đều. Khi đó ta chọn một ca kiểm thử là (5, 5, 5) để thực hiện thì sau đó ta có thể kết luận tương tự về kết quả kiểm thử cho các ca kiểm thử khác cùng lớp như (6, 6, 6) hay (100, 100, 100). Bản chất ở đây là chúng ta cho rằng chương trình sẽ thực hiện cùng các lệnh giống nhau với các ca kiểm thử cùng một lớp tương đương, nên hành vi của chương trình là tương tự nhau – cùng có lỗi hay ra kết quả sai hay cùng ra các kết quả đúng.

Bây giờ vấn đề chính là làm sao xác định được các lớp tương đương của miền đầu vào. Việc này có thể thực hiện bằng cách phân tích mã nguồn của chương trình. Nếu không phân tích được mã nguồn, chúng ta có thể dựa vào hiểu biết về chức năng của chương trình mà suy ra cách cài đặt của chúng. Từ đó chúng ta có thể suy ra các lớp tương đương theo kinh nghiệm và trực giác của mình.

Kiểm thử lớp tương đương được chia thành một số loại. Chúng ta sẽ sử dụng chương trình minh họa sau để giải thích các loại kiểm thử lớp tương đương dễ dàng hơn.

Giả sử chương trình của chúng ta là một hàm của ba biến a , b , c và không gian đầu vào là ba tập A , B , C . Giả sử tiếp là chúng ta đã xác định được các miền tương đương cho mỗi không gian đầu vào:

$$A = A_1 \cup A_2 \cup A_3$$

$$B = B_1 \cup B_2 \cup B_3 \cup B_4$$

$$C = C_1 \cup C_2$$

Chúng ta ký hiệu phần tử của các miền tương đương trên bằng chữ thường với cùng chỉ số trong các phần sau. Ví dụ $a_1 \in A_1$, $b_3 \in B_3$ và $c_2 \in C_2$.

3.3.1 Kiểm thử lớp tương đương yếu

Kiểm thử lớp tương đương yếu chỉ yêu cầu mỗi không gian tương đương có ít nhất một phần tử xuất hiện trong một ca kiểm thử nào đó. Với chương trình minh họa trên thì phải có một giá trị thuộc A_1 trong một ca kiểm thử nào đó. Tương tự với các tập con khác. Ta có bộ kiểm thử thỏa mãn kiểm thử lớp tương đương yếu như

trong Bảng 3.1.

Bảng 3.1: Các ca kiểm thử lớp tương đương cho Triangle

TT	a	b	c
1	a_1	b_1	c_1
2	a_2	b_2	c_2
3	a_3	b_3	c_3
4	a_1	b_4	c_2

Bộ kiểm thử trên sử dụng một giá trị từ mỗi lớp tương đương. Để kiểm tra chúng ta chỉ cần để ý đến các cột, ví dụ cột a có xuất hiện cả a_1, a_2, a_3 của ba miền con của A , như vậy là đủ.

Chúng ta có thể lập qui tắc để tự động xây dựng bộ kiểm thử lớp tương đương yếu này dễ dàng và cũng có thể dễ nhận thấy là số ca kiểm thử tối thiểu chính là số lớp tương đương lớn nhất của các miền đầu vào, trong ví dụ này là B .

3.3.2 Kiểm thử lớp tương đương mạnh

Kiểm thử lớp tương đương mạnh sẽ kết hợp các tổ hợp có thể của các miền tương đương. Với ví dụ trên ta sẽ có $3*4*2 = 24$ tổ hợp tương ứng với 24 ca kiểm thử như bảng 3.2.

Việc xác định các ca kiểm thử ở bảng trên tương tự như việc xây dựng bảng giá trị chân lý cho logic mệnh đề nên cũng dễ dàng tự động hóa được. Tích Đề-các này cho ta một bộ kiểm thử đầy đủ, có tất cả các khả năng kết hợp của các miền tương đương của các miền đầu vào.

Ở hai phần trên chúng ta chỉ chú trọng các miền tương đương của biến đầu vào. Chúng ta có thể áp dụng các phương pháp này cho miền đầu ra. Việc này thường khó hơn vì chúng ta phải tính ngược bộ giá trị kiểm thử ở miền đầu vào để chúng tạo ra giá trị ở miền đầu ra mong muốn. Với chương trình có nhiều biến đầu ra, chúng ta cũng có hai dạng mạnh và yếu tương tự với đầu vào.

Bảng 3.2: Các ca kiểm thử lớp tương đương mạnh cho Triangle

TT	a	b	c
1	a_1	b_1	c_1
2	a_1	b_1	c_2
3	a_1	b_2	c_1
4	a_1	b_2	c_2
5	a_1	b_3	c_1
6	a_1	b_3	c_2
7	a_1	b_4	c_1
8	a_1	b_4	c_2
9	a_2	b_1	c_1
10	a_2	b_1	c_2
11	a_2	b_2	c_1
12	a_2	b_2	c_2
13	a_2	b_3	c_1
14	a_2	b_3	c_2
15	a_2	b_4	c_1
16	a_2	b_4	c_2
17	a_3	b_1	c_1
18	a_3	b_1	c_2
19	a_3	b_2	c_1
20	a_3	b_2	c_2
21	a_3	b_3	c_1
22	a_3	b_3	c_2
23	a_3	b_4	c_1
24	a_3	b_4	c_2

3.3.3 Kiểm thử lớp tương đương đơn giản

Kiểm thử lớp tương đương đơn giản chỉ phân chia một lớp gồm các giá trị hợp lệ và các miền giá trị không hợp lệ. Ví dụ nếu A là khoảng 1 đến 200 các số nguyên thì miền hợp lệ là các giá trị từ 1 đến 200, miền không hợp lệ gồm hai miền. Miền thứ nhất là các giá trị từ 0 trở xuống. Miền còn lại là các giá trị từ 201 trở lên. (Để đơn giản ta không xét miền các giá trị không phải là số nguyên.) Sau khi đã có các lớp tương đương theo cách đơn giản này chiến lược kiểm thử có thể áp dụng tương tự kiểm thử tương đương mạnh và yếu.

Có hai vấn đề với kiểm thử lớp tương đương đơn giản. Thứ nhất là thông thường khi mô tả bài toán thì tài liệu không nói rõ các trường hợp khi đầu vào không hợp lệ thì kết quả là gì. Nếu có nói thì cũng thường chỉ đề cập đến một số trường hợp đã biết mà không xét đến mọi khả năng có thể. Ví dụ chúng ta viết chương trình giải

phương trình bậc hai với ba hệ số nguyên a, b, c được nhập vào từ bàn phím. Rất nhiều lập trình viên sẽ chuyển ngay xâu ký tự người sử dụng nhập vào thành các số và gán cho các hệ số này. Nhưng nếu người sử dụng nhập một xâu chữ thì chương trình không kiểm tra để có thông báo phù hợp cho người sử dụng. Khi thiết kế kiểm thử chúng ta phải bổ sung một loạt các trường hợp mà hệ thống phải đưa ra kết quả bổ sung hoặc các thông báo lỗi phù hợp cho các trường hợp này. Thứ hai là một số ngôn ngữ lập trình có hệ thống kiểm tra kiểu mạnh như Java, C# thì không thể đưa giá trị sai kiểu vào được nên chúng ta không cần các ca kiểm thử cho giá trị sai kiểu, nhưng vẫn cần kiểm tra dữ liệu do người sử dụng nhập vào, và chỉ kiểm tra dữ liệu không hợp lệ cùng kiểu. Tuy nhiên trong các ngôn ngữ lập trình script như Javascript, PHP, Python, không có kiểm tra kiểu mạnh nên các lỗi loại này khá phổ biến do đó ta cần xét kỹ việc chia miền đầu vào thành các lớp tương đương.

❖ Kinh nghiệm áp dụng

Từ các ví dụ trên chúng ta có một số quan sát và định hướng sử dụng kiểm thử lớp tương đương.

- Kiểm thử lớp tương đương đơn giản kém kiểm thử lớp tương đương yếu kém kiểm thử lớp tương đương mạnh.

- Sử dụng kiểm thử tương đương đơn giản khi ngôn ngữ không có kiểm tra kiểu mạnh.

- Nếu cần kiểm tra ngoại lệ chúng ta nên mở rộng kiểm thử lớp tương đương với các lớp giá trị ngoài miền xác định.

- Kiểm thử lớp tương đương phù hợp với dữ liệu đầu vào có miền giá trị là khoảng và hữu hạn.

- Kiểm thử lớp tương đương kết hợp với kiểm thử giá trị biên sẽ tốt hơn.

- Nên dùng kiểm thử lớp tương đương với các chương trình phức tạp.

- Nên dùng kiểm thử lớp tương đương mạnh khi các biến là độc lập. Khi các biến phụ thuộc nhau thì dễ tạo ra các ca kiểm thử vô lý.

Câu hỏi cuối bài:

Câu 1. Nêu ý tưởng phương pháp kiểm thử giá trị biên?

Câu 2. Nêu nguyên tắc chọn dữ liệu thử cho kiểm thử giá trị biên?

Câu 3. Phân loại kiểm thử giá trị biên?

Câu 4. Nguyên tắc phân hoạch lớp tương đương?

Câu 5. Phân loại kiểm thử lớp tương đương

Bài 4: Các phương pháp kiểm thử hàm - tiếp (Số tiết: 03 tiết)

3.4 Kiểm thử bằng bảng quyết định

Kỹ thuật kiểm thử lớp tương đương và kiểm thử giá trị biên thích hợp cho các hàm có các biến đầu vào không có quan hệ ràng buộc với nhau. Kỹ thuật kiểm thử dựa trên bảng quyết định chúng ta xem xét trong chương này sẽ phù hợp cho các hàm có các hành vi khác nhau dựa trên tính chất của bộ giá trị của đầu vào.

Kiểm thử dựa trên bảng quyết định là phương pháp chính xác nhất trong các kỹ thuật kiểm thử chức năng. Bảng quyết định là phương pháp hiệu quả để mô tả các sự kiện, hành vi sẽ xảy ra khi một số điều kiện thỏa mãn.

Cấu trúc Bảng quyết định:

Cấu trúc bảng quyết định chia thành bốn phần chính như trong Bảng 3.3:

- Các biểu thức điều kiện C_1, C_2, C_3 .
- Giá trị điều kiện T, F, –.
- Các hành động A_1, A_2, A_3, A_4 .
- Giá trị hành động, có (xảy ra) hay không, X là có.

Bảng 3.3: Ví dụ bảng quyết định

Điều kiện	Các trường hợp (qui tắc)			
	TH ₁	TH ₂	TH ₃	TH ₄
đk1 (C_1)	T	T	F	F
đk2 (C_2)	T	F	T	F
Hành động của hệ thống				
A_1	T (Có)	F (không)	T	F
A_2	T	T	F	F

Khi lập bảng quyết định ta thường tìm các điều kiện có thể xảy ra, để xét các tổ

hợp của chúng mà từ đó chúng ta sẽ xác định được các ca kiểm thử tương ứng cho các điều kiện được thỏa mãn. Các hành động xảy ra chính là kết quả mong đợi của ca kiểm thử đó.

Bảng quyết định với các giá trị điều kiện chỉ là T, F, và – được gọi là *bảng quyết định logic*. Chúng ta có thể mở rộng các giá trị này bằng các tập giá trị khác, ví dụ 1, 2, 3, 4, khi đó chúng ta có *bảng quyết định tổng quát*.

Kỹ thuật thực hiện: Để xác định các ca kiểm thử dựa trên bảng quyết định, ta dịch các điều kiện thành các đầu vào và các hành động thành các đầu ra. Đôi khi các điều kiện sẽ xác định các lớp tương đương của đầu vào và các hành động tương ứng với các mô-đun xử lý chức năng đang kiểm thử.

Do đó mỗi cột tương ứng với một ca kiểm thử. Vì tất cả các cột bao phủ toàn bộ các tổ hợp đầu vào nên chúng ta có một bộ kiểm thử đầy đủ.

Trên thực tế không phải tổ hợp đầu vào nào cũng là hợp lệ, do đó khi sử dụng bảng quyết định người ta thường bổ sung thêm một giá trị đặc biệt ‘–’ để đánh dấu các điều kiện không thể cùng xảy ra này. Các giá trị – (không quan tâm) có thể hiểu là luôn sai, không hợp lệ. Nếu các điều kiện chỉ là T và F ta có 2^n cột qui tắc.

Bảng 3.4 là một ví dụ đơn giản về một bảng quyết định để khắc phục sự cố máy in. Khi máy in có sự cố, chúng ta sẽ xem xét tình trạng dựa trên các điều kiện trong bảng là đúng hay sai, từ đó xác định được cột duy nhất có các điều kiện thỏa mãn, và thực hiện các hành động khắc phục sự cố tương ứng.

Bảng 3.4: Bảng quyết định để khắc phục sự cố máy in

Điều kiện	Máy in không in	Y	Y	Y	Y	N	N	N	N
	Đèn đỏ nhấp nháy	Y	Y	N	N	Y	Y	N	N
	Không nhận ra máy in	Y	N	Y	N	Y	N	Y	N
Hành động	Kiểm tra dây nguồn			X					
	Kiểm tra cáp máy in	X		X					
	Kiểm tra phần mềm in	X		X		X		X	
	Kiểm tra mực in	X	X			X	X		
	Kiểm tra kẹt giấy		X		X				

Chú ý là ở đây thứ tự các điều kiện và thứ tự thực hiện hành động không quan trọng, nên chúng ta có thể đổi vị trí các hàng. Với các hành động cũng vậy, tuy nhiên tùy trường hợp chúng ta có thể làm mịn hơn bằng việc đánh số thứ tự hành động

xảy ra thay cho dấu X để chỉ ra hành động nào cần làm trước. Với bảng quyết định tổng quát, các giá trị của điều kiện không chỉ nhận giá trị đúng (T) hoặc sai (F), khi đó ta cần tăng số cột để bao hết các tổ hợp có thể của các điều kiện.

Kinh nghiệm áp dụng:

So với các kỹ thuật kiểm thử khác, kiểm thử bằng bảng quyết định tốt hơn với một số bài toán (như NextDate) nhưng cũng kém hơn với một số bài toán (như Commission). Những bài toán phù hợp với bảng quyết định khi chương trình có nhiều lệnh rẽ nhánh (như Triangle) và các biến đầu vào có quan hệ với nhau (như NextDate).

1. Nên dùng kỹ thuật bảng quyết định cho các ứng dụng có một trong các tính chất sau:

- Chương trình có nhiều lệnh rẽ nhánh – nhiều khối If-Then-Else
- Các biến đầu vào có quan hệ với nhau
- Có các tính toán giữa các biến đầu vào
- Có quan hệ nhân quả giữa đầu vào và đầu ra
- Có độ phức tạp cao Cyclomatic cao

2. Bảng quyết định không dễ áp dụng cho các bài toán lớn (với n điều kiện có 2^n quy tắc). Nên dùng dạng mở rộng và sử dụng đại số để đơn giản hóa bảng.

Có thể cần một số lần thử và rút kinh nghiệm để dần dần lập được bảng tối ưu.

3.5 Kiểm thử tổ hợp

Thông thường lỗi phần mềm do một điều kiện nào đó làm phát sinh, không liên quan gì đến các điều kiện khác. Ví dụ khi nhập ngày sinh của một học sinh lớp bốn là 19/8/2013 vào danh sách lớp năm học 2013-2014 thì thông báo lỗi cần xuất hiện vì học sinh này chưa được một tuổi. Tuy nhiên cũng có những lỗi phần mềm chỉ xuất hiện do kết hợp của một số điều kiện. Để kiểm tra kỹ một phần mềm chúng ta phải kiểm tra tất cả các tổ hợp của các điều kiện có thể. Nhưng thông thường số tổ hợp của tất cả các điều kiện này sẽ bùng nổ rất nhanh và sẽ rất tốn công để kiểm tra tất cả chúng. Ví dụ giả sử hàm $y = f(x_1, x_2)$ với x_1 và x_2 chỉ nhận giá trị nguyên từ 1 đến 5. Chúng ta có $5 * 5 = 25$ tổ hợp phải kiểm thử là (1, 1), (1, 2), ..., (5, 5).

Tổng quát hóa với hàm $Y = P(X)$ trong đó $X = (x_1, \dots, x_n)$ và để đơn giản ta giả sử với mỗi biến x_i , $1 \leq i \leq n$ chỉ có k_i giá trị đáng chú ý. Nếu tính tất cả các tổ hợp của các giá trị đáng chú ý này chúng ta có $k_1 \dots k_n$ ca kiểm thử.

Bằng cách không lấy hết tất cả các tổ hợp có thể mà chúng ta chỉ yêu cầu mọi tổ hợp của n biến bất kỳ đều xuất hiện trong một ca kiểm thử nào đó thì chúng ta có thể giảm đáng kể số ca kiểm thử so với tất cả các tổ hợp có thể. Kết quả nghiên cứu thực tế cho thấy tổ hợp đôi một đã có khả năng phát hiện khoảng 80% lỗi. Kiểm thử tổ hợp bốn đến sáu biến có thể phát hiện được 100% lỗi [KWG04]. Chúng ta sẽ xem xét một dạng phổ biến nhất với $n = 2$ gọi là kiểm thử đôi một.

3.5.1 Kiểm thử đôi một

Kiểm thử đôi một (pairwise testing) là một trường hợp đặc biệt của kiểm thử tất cả các tổ hợp. Kiểm thử đôi một chỉ yêu cầu mỗi cặp giá trị của (x_i, x_j) xuất hiện trong một ca kiểm thử nào đó. Một ca kiểm thử thường có nhiều cặp giá trị này với các i, j khác nhau nên dễ thấy số lượng ca kiểm thử sẽ giảm đáng kể so với tổ hợp tất cả các ca kiểm thử.

Với kiểm thử đôi một, mỗi cặp giá trị của hai biến bất kỳ cần xuất hiện trong ít nhất một ca kiểm thử. Bảng 3.5 là bộ kiểm thử đôi một, tuy chỉ có bốn ca kiểm thử, nhưng chúng chứa tất cả các cặp giá trị có thể của các biến.

Bảng 3.5: Các ca kiểm thử đôi một cho hàm g

TT	x_1	x_2	x_3
TC1	<i>True</i>	0	a
TC2	<i>True</i>	5	b
TC3	<i>False</i>	0	b
TC4	<i>False</i>	5	a

Bảng 3.6: Mảng trực giao $L_4(2^3)$

TT	1	2	3
1	1	1	1
2	1	2	2

3	2	1	2
4	2	2	1

3.5.2 Ma trận trực giao

Chúng ta hãy xem xét mảng hai chiều trong Bảng 3.6. Bảng này có hai tính chất thú vị. Tất cả các cặp (1,1), (1,2), (2,1), and (2,2) đều xuất hiện trong hai cột bất kỳ. Tuy nhiên không phải tất cả các tổ hợp của 1 và 2 đều có trong bảng. Ví dụ (2, 2, 2) là tổ hợp hợp lệ nhưng nó không nằm trong bảng. Chỉ bốn trong tám tổ hợp nằm trong bảng. Trong bảng chỉ có 4 tổ hợp trên tổng số 8 tổ hợp có thể. Đây là một ví dụ về ma trận trực giao $L_4(2^3)$. Chỉ số 4 là số hàng của mảng, 3 là số cột của mảng, và 2 là số giá trị lớn nhất một phần tử của mảng có thể nhận.

Xem lại ví dụ của hàm g trên chúng ta thấy bảng này là một thể hiện của bảng trực giao tổng quát $L_4(2^3)$ trong Bảng 3.6. Các giá trị 1, 2 trong Bảng 3.6 đượ thay bằng các giá trị tương ứng của biến của hàm g . Ví dụ 1 tương ứng với True, 2 tương ứng với False. Chúng ta có thể kiểm tra là mỗi cặp giá trị bất kỳ của hai biến bất kỳ đều xuất hiện trong ít nhất một ca kiểm thử ở trong Bảng 3.5.

❖ Kinh nghiệm áp dụng

Đến đây chúng ta đã hiểu về khái niệm ma trận trực giao và ứng dụng của nó trong việc sinh các ca kiểm thử cặp đôi. Sau đây là một qui trình thực hiện việc ứng dụng ma trận trực giao để sinh các ca kiểm thử theo phương pháp kiểm thử cặp đôi.

Bước 1. Xác định tất cả các biến đầu vào, vì chúng sẽ tương ứng với các cột của ma trận trực giao.

Bước 2. Xác định số lượng giá trị của mỗi biến có thể nhận để xác định được số lượng lớn nhất vì các giá trị của ma trận sẽ nằm trong khoảng này.

Bước 3. Tìm ma trận trực giao thích hợp với số lượng hàng nhỏ nhất với hai số cột và giá trị của các phần tử ma trận đã xác định ở Bước 1 và Bước 2.

Bước 4. Ánh xạ các biến với các cột của ma trận và giá trị của biến với các phần tử của ma trận.

Bước 5. Kiểm tra còn phần tử nào trong ma trận chưa được ánh xạ không. Chọn

các giá trị hợp lệ tùy ý cho những phần tử này.

Bước 6. Chuyển các hàng của ma trận thành các ca kiểm thử

Bài tập cuối chương

Bài 3.1. Trình bày nguyên tắc chọn dữ liệu thử?

Bài 3.2. Trình bày nguyên tắc phân hoạch lớp tương đương?

Bài 3.3 Trình bày các thành phần của bảng quyết định?

Bài 3.4. Cấu trúc của bảng quyết định?

Bài 3.5. Các bước thực hiện kiểm thử giá trị biên?

Bài 3.6. Các bước thực hiện kiểm thử lớp tương đương?

Bài 3.7. Các bước thực hiện kiểm thử bằng bảng quyết định?

Bài 3.8. Các bước thực hiện kiểm thử tổ hợp?

Bài 3.9. Kinh nghiệm áp dụng các kỹ thuật kiểm thử hàm?

Bài 3.10. Thiết kế testcase kiểm thử chức năng: Đăng nhập, đăng ký sử dụng các phương pháp kiểm thử hàm?

CHƯƠNG 4: KỸ THUẬT KIỂM THỬ HỘP TRẮNG

Nội dung chính của chương

Chương 4 trình bày những kiến thức cơ bản về hai phương pháp được sử dụng trong kiểm thử hộp trắng là kiểm thử dòng điều khiển (control flow testing) và kiểm thử dòng dữ liệu (data 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. 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.

Mục tiêu cần đạt được của chương

Biết cách thiết kế testcase theo quy trình kiểm thử dòng điều khiển và dòng dữ liệu dựa trên các độ đo kiểm thử.

Vận dụng vào kiểm thử các phần mềm cụ thể.

Bài 5: Kiểm thử dòng điều khiển (Số tiết: 03 tiết)

4.1 Tổng quan về kiểm thử hộp trắng

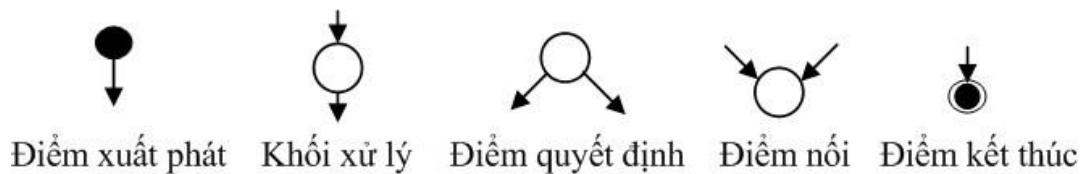
Kiểm thử hộp trắng sử dụng các chiến lược cụ thể và sử dụng mã nguồn của chương trình/đơn vị phần mềm cần kiểm thử nhằm kiểm tra xem chương trình/đơn vị phần mềm có thực hiện đúng so với thiết kế và đặc tả hay không. Trong khi các phương pháp kiểm thử hộp đen hay kiểm thử hàm/chức năng chỉ cho phép phát hiện các lỗi/khiếm khuyết có thể quan sát được, kiểm thử hộp trắng cho phép phát hiện các lỗi/khiếm khuyết tiềm ẩn bên trong chương trình/đơn vị phần mềm. Các lỗi này thường khó phát hiện bởi các phương pháp kiểm thử hộp đen. 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/đơn vị 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.

4.2 Kiểm thử dựa trên dòng điều khiển

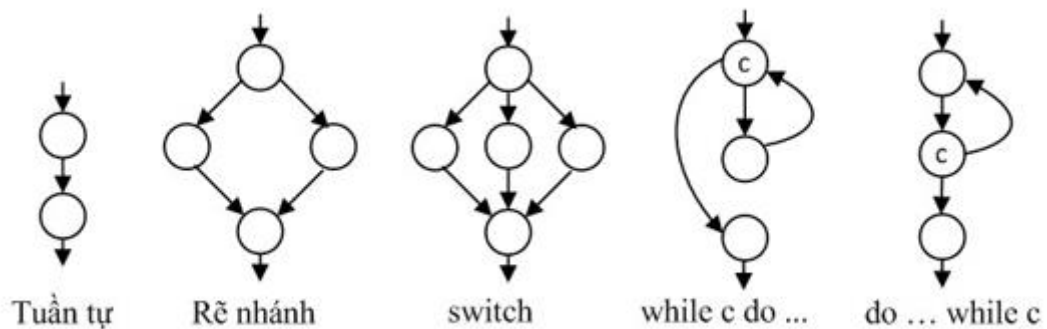
4.2.1 Đồ thị dòng điều khiển

Phương pháp kiểm thử dòng điều khiển dựa trên khái niệm đồ thị dòng điều khiển. Đồ thị này được xây dựng từ mã nguồn của chương trình/đơn vị chương trình. Đồ thị dòng điều khiển là một đồ thị có hướng gồm các đỉnh tương ứng với các câu lệnh/nhóm câu lệnh và các cạnh là các dòng điều khiển giữa các câu lệnh/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 lệnh tương ứng với i .

Xây dựng một đồ thị dòng điều khiển từ một chương trình/đơn vị chương trình khá đơn giản. Hình 4.1 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 kiện trong các khối lệnh rẽ nhánh hoặc lặp, điểm nối ứng với các câu lệnh ngay sau các lệnh rẽ nhánh, và điểm kết thúc ứng với điểm kết thúc của đơn vị chương trình. Các cấu trúc điều khiển phổ biến của chương trình được mô tả trong Hình 4.2. 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 mọi ngôn ngữ lập trình.



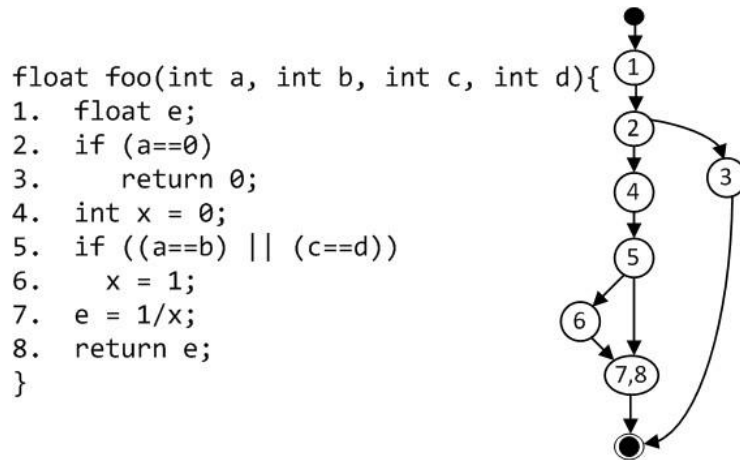
Hình 4.1: Các thành phần cơ bản của đồ thị chương trình.



Hình 4.2: Các cấu trúc điều khiển phổ biến của chương trình.

Chúng ta thử xem cách dựng đồ thị dòng điều khiển cho đơn vị chương trình có mã nguồn bằng ngôn ngữ C như Hình 4.3. 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 trong khi các đỉnh

5 và 6 ứng với câu lệnh if. Đỉnh 7,8 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.



Hình 4.3: Mã nguồn của hàm foo và đồ thị dòng điều khiển của nó.

4.2.2 Các độ đo kiểm thử

Kiểm thử hàm (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 bao 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.

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 4.3, ta chỉ cần hai ca kiểm thử như Bảng 4.1 là đạt 100% độ phủ cho độ đo 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).

Bảng 4.1: Các ca kiểm thử cho độ đo C1 của hàm foo

ID	Inputs	EO	RO	Note
tc1	0, 1, 2, 3	0		
tc2	1, 1, 2, 3	1		

Độ đo kiểm thử cấp 2 (C2): các điểm quyết định trong đồ thị dòng điều khiển của đơn vị kiểm thử đều được thực hiện ít nhất một lần cả hai nhánh đúng và sai. Ví dụ, Bảng 4.2 mô tả các trường hợp cần kiểm thử để đạt được 100% độ phủ của độ đo C2 ứng với hàm foo được mô tả trong Hình 4.3.

Như vậy, với hai ca kiểm thử trong độ đo kiểm thử cấp 1 (tc1 và tc2), ta chỉ kiểm thử được $3/4 = 75\%$ ứng với độ đo kiểm thử cấp 2. Chúng ta cần một ca kiểm thử nữa ứng với trường hợp sai của điều kiện $(a == b) \parallel (c == d)$ nhằm đạt được 100% độ phủ của độ đo C2. Bảng 4.2 mô tả các ca kiểm thử cho mục đích này.

Bảng 4.2: Các ca kiểm thử cho độ đo C2 của hàm foo

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		

Độ đo 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. Ví dụ, 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 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. Ví dụ, Bảng 4.3 mô tả các trường hợp cần kiểm thử để đạt được 100% độ phủ của độ đo C3 ứng với hàm foo được mô tả trong Hình 4.3.

Như vậy, với ba ca kiểm thử trong độ đo kiểm thử cấp 2 (tc1, tc2 và tc3), ta chỉ kiểm thử được $7/8 = 87,5\%$ ứng với độ đo kiểm thử cấp 3. Chúng ta cần một ca kiểm thử nữa ứng với trường hợp sai của điều kiện con cơ bản $(c == d)$ nhằm đạt được 100% độ phủ của độ đo C3. Bảng 4.4 mô tả các ca kiểm thử cho mục đích này.

Bảng 4.3: Các trường hợp cần kiểm thử của độ đo C3 với hàm foo

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

Bảng 4.4: Các ca kiểm thử cho độ đo C3 của hàm foo

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		
tc4	1, 2, 1, 1	1		

Câu hỏi cuối bài:

Câu 1. Kiểm thử hộp trắng là gì?

Câu 2. So sánh kiểm thử hộp đen và kiểm thử hộp trắng?

Câu 3. Định nghĩa đồ thị dòng điều khiển?

Câu 4. Nêu các độ đo kiểm thử?

Câu 5. Viết hàm kiểm tra 1 số nguyên có phải là số nguyên tố hay không. Vẽ đồ thị dòng điều khiển cho hàm vừa viết.

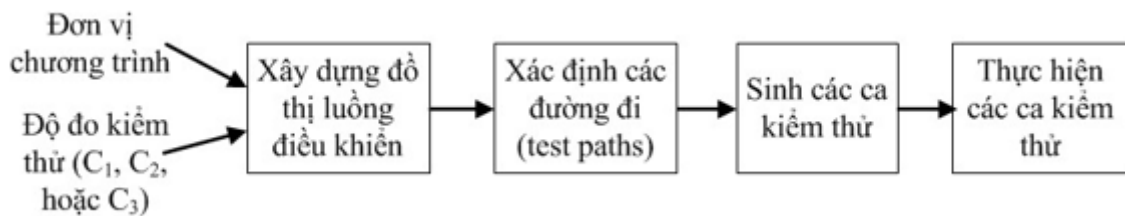
Bài 6: Kiểm thử dòng điều khiển dựa trên độ đo (Số tiết: 03 tiết)

4.2.3 Kiểm thử dựa trên độ đo

Kiểm thử dựa trên độ đo là phương pháp chạy mã nguồn sao cho bao phủ một độ đo nào đó. Hình 4.4 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 ứng với độ đo C3. 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

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. Cuối cùng, các ca kiểm thử được thực hiện trên đơn vị chương trình nhằm phát hiện các lỗi.



Hình 4.4: Quy trình kiểm thử đơn vị chương trình dựa trên độ đo

a. Kiểm thử cho độ đo C1

Xét lại hàm foo có mã nguồn như Hình 4.5. Chúng ta xây dựng đồ thị dòng điều khiển ứng với độ phủ C1 cho hàm này như Hình 4.5. 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/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; 4; 5; 6; 7,8
2. 1; 2; 3

Với đường đi 1; 2; 4; 5; 6; 7,8, ta sẽ sinh 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 ứng với điểm quyết định 2 ($a == 0$) là sai và điều kiện ứng với điểm quyết định 5 ($(a == b) \parallel (c == d)$) 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; 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 4.5 là một ví dụ về hai ca kiểm thử được sinh ra bằng ý tưởng trên.

Bảng 4.5: Các ca kiểm thử cho độ đo C1 của hàm foo

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		

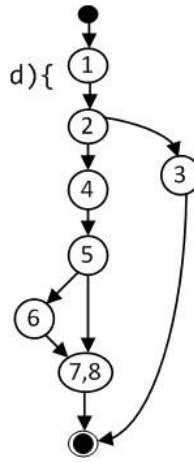
b. Kiểm thử cho độ đo C2

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 4.5 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 4.2.3, chúng ta có hai cách để tính được con số này.

```

float foo(int a, int b, int c, int d){
1. float e;
2. if (a==0)
3.     return 0;
4. int x = 0;
5. if ((a==b) || (c==d))
6.     x = 1;
7. e = 1/x;
8. return e;
}

```



Hình 4.5: Hàm foo và đồ thị dòng điều khiển của nó.

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 2 và 5 đều được kiểm tra.

1. 1; 2; 4; 5; 6; 7,8
2. 1; 2; 3
3. 1; 2; 4; 5; 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. Bảng 4.6 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).

Bảng 4.6: Các ca kiểm thử cho độ đo C2 của hàm foo

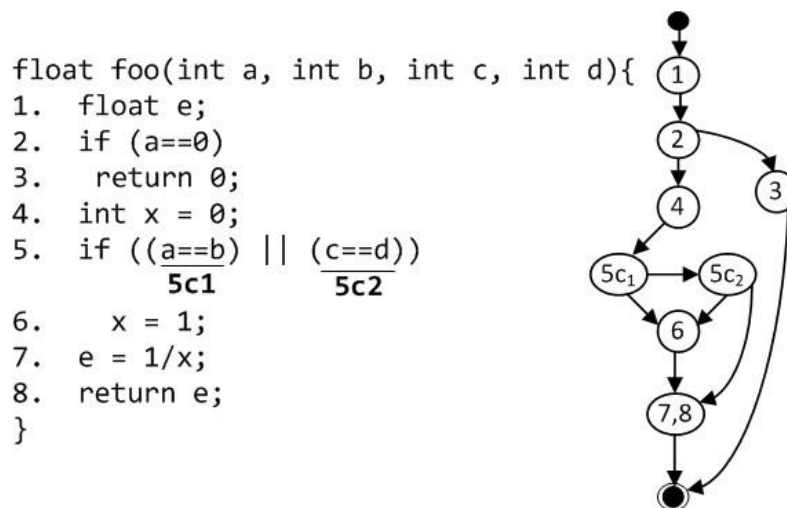
ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		
tc3	1; 2; 4; 5; 7,8	2,3,4,5	lỗi chia cho 0		

Độ đo C1 đảm bảo các câu lệnh được “viếng thăm” 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 4.6, 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 4.7.

c. Kiểm thử cho độ đo C3

Như đã trình bày ở phía trên, ứng với mỗi đơn vị chương trình, 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. Ví dụ, đồ thị dòng điều khiển của hàm foo ứng với độ đo C3 được xây dựng như Hình 4.6. 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 hai 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 càng 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 4.6: Hàm foo và đồ thị dòng điều khiển của nó ứng với độ đo C3

Đồ thị dòng điều khiển của hàm foo ứng với độ đo C3 như Hình 4.6 có ba 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; 4; 5c1; 6; 7,8
2. 1; 2; 4; 5c1; 5c2; 6; 7,8

3. 1; 2; 4; 5c1; 5c2; 7,8

4. 1; 2; 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. Bảng 4.7 là một ví dụ về các ca kiểm thử cho hàm foo ứng với độ đo C3.

Bảng 4.7: Các ca kiểm thử cho độ đo C3 của hàm foo

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5c1; 6; 7,8	0, 2, 3, 5	0		
tc2	1; 2; 4; 5c1; 5c2; 6; 7,8	2, 2, 2, 7	1		
tc3	1; 2; 4; 5c1; 5c2; 7,8	2,3,4,5	lỗi chia cho 0		
tc4	1; 2; 3	2,3,4,4	1		

d. 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. Trong thực tế, lỗi hay xảy ra ở 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 ba trường hợp sau:

- **Lệnh lặp đơn giản:** đơn vị chương trình chỉ chứa đúng một vòng lặp (thân của vòng lặp không chứa các vòng lặp khác).
- **Lệnh lặp liên kế:** đơn vị chương trình chỉ chứa các lệnh lặp kế tiếp nhau.
- **Lệnh lặp lồng nhau:** đơn vị chương trình chỉ chứa các vòng lặp chứa các lệnh lặp khác.

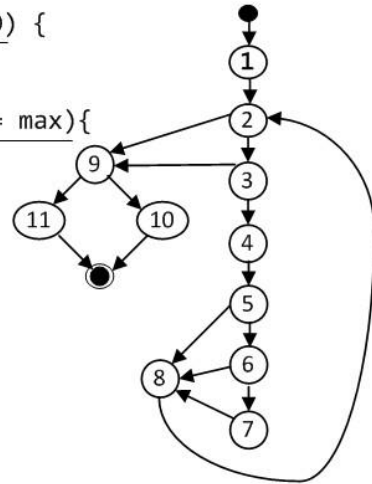
Để kiểm thử các đơn vị chương trình chỉ có lệnh lặp đơn giản, ta xét hàm average với mã nguồn đồ thị dòng điều khiển tương ứng với độ đo C3 như Hình 4.7.

Đồ thị dòng điều khiển của hàm average ứng với độ đo C3 như Hình 4.7 có năm điểm quyết định là 2, 3, 5, 6 và 9 nên chúng ta cần $5 + 1 = 6$ đường đi để được 100% độ phủ của độ đo C3.

```

double average(double value[], double min, double
max, int& tcnt, int& vcnt) {
double sum = 0;
int i = 1;
tcnt = vcnt = 0;
while (value[i] <> -999 && tcnt <100) {
    tcnt++;
    if (min<=value[i] && value[i]<= max){
        sum += value[i];
        vcnt ++;
    }
    i++;
} //end while
if (vcnt > 0)
    return sum/vcnt;
return -999;
} //end

```



Hình 4.7: Hàm average và đồ thị dòng điều khiển của nó ứng với độ đo C3.

Các đường đi cần thiết được liệt kê như sau:

1. 1; 2; 9; 10
2. 1; 2; 9; 11
3. 1; 2; 3; 9; 10
4. 1; 2; 3; 4; 5; 8; 2; 9; 11
5. 1; 2; 3; 4; 5; 6; 8; 2; 9; 11
6. 1; 2; 3; 4; 5; 6; 7; 8; 2; 9; 10

Bảng 4.8: Các ca kiểm thử cho độ đo C₃ của hàm average

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 9; 10				tc6
tc2	1; 2; 9; 11	[-999, ...], 1, 2	-999		
tc3	1; 2; 3; 9; 10				tc6
tc4	1; 2; 3; 4; 5; 8; 2; 9; 11	[0,-999], 1, 2	-999		
tc5	1; 2; 3; 4; 5; 6; 8; 2; 9; 11	[3,-999], 1, 2	-999		
tc6	1; 2; 3; 4; 5; 6; 7; 8; 2; 9; 10	[1,-999], 1, 2	1		

Với mỗi đường đi, chúng ta sẽ sinh một ca kiểm thử tương ứng. Bảng 4.8 mô tả các ca kiểm thử cho hàm average ứng với độ đo C3. Với mỗi ca kiểm thử, bộ dữ liệu đầu vào (inputs) gồm ba thành phần: double value[], double min, và double max. Với đường đi 1; 2; 9; 10, chúng ta không thể tìm được bộ dữ liệu đầu vào để đường đi này được thực thi. Thực vậy, điều kiện để thực thi đường đi này là value[0] = -999 (tức là điều kiện 2 là sai) và điều kiện 9 đúng (tức là vcnt > 0). Điều này không bao giờ xảy ra vì

nếu $value[0] = -999$ thì $vcnt = 0$. Tương tự, chúng ta cũng không thể sinh bộ kiểm thử với đường đi 1; 2; 3; 9; 10. Trong các trường hợp này, chúng ta không cần sinh các ca kiểm thử cho những đường đi này. Chúng sẽ được kiểm thử bởi các đường đi khác (Ví dụ: đường đi 1; 2; 3; 4; 5; 6; 7; 8; 2; 9; 10 chứa các đỉnh của các đường dẫn trên).

Với các đường đi trên, vòng lặp while của hàm average chỉ được thực hiện tối đa một lần lặp nên chúng ta rất khó để phát hiện các lỗi tiềm ẩn (có thể có) bên trong vòng lặp này. Các lỗi này có thể xảy ra khi vòng lặp này được thực hiện nhiều lần lặp. Ví dụ trên đã chỉ ra những hạn chế của phương pháp kiểm thử dòng điều khiển khi áp dụng cho các chương trình/đơn vị chương trình có chứa vòng lặp. Để giải quyết vấn đề này, chúng ta cần sinh thêm bảy ca kiểm thử ứng với bảy trường hợp sau:

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

Chú ý rằng trong một số trường hợp chúng ta có thể không xác định được số lần lặp tối đa của các vòng lặp. Trong trường hợp này, chúng ta chỉ cần sinh bốn ca kiểm thử đầu tiên. Tương tự, trong một số các trường hợp khác, chúng ta không thể sinh ca kiểm thử để vòng lặp thực hiện n + 1 lần (trường hợp thứ 7). Khi đó, chúng ta chỉ cần sinh sáu ca kiểm thử còn lại (các trường hợp từ 1–6). Ví dụ, với vòng lặp while trong hàm average như Hình 4.7, 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 4.9 nhằm kiểm thử vòng lặp này.

Bảng 4.9: Các ca kiểm thử cho cho kiểm thử vòng lặp while của hàm average

ID	Số 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)					

Với các chương trình/đơn vị chương trình có các vòng lặp liên kế, 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 (như đã mô tả ở trên). 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 bảy ca kiểm thử như đã mô tả ở trên).

4.2.4 Tổng kết

Kiểm thử dòng điều khiển là một trong những phương pháp kiểm thử quan trọng nhất của chiến lược kiểm thử hộp trắng cho các chương trình/đơn vị chương trình. Phương pháp này cho phép phát hiện ra các lỗi (có thể có) tiềm ẩn bên trong chương trình/đơn vị chương trình bằng cách kiểm thử các đường đi của nó tương ứng với các dòng điều khiển có thể có. Để áp dụng phương pháp này, chúng ta cần xác định độ đo kiểm thử (cần kiểm thử với độ đo nào?). Tiếp theo, đồ thị dòng điều khiển của chương trình/đơn vị chương trình ứng với độ đo kiểm thử sẽ được tạo ra. Dựa vào đồ thị này, chúng ta sẽ sinh ra các đường đi độc lập. Số đường đi độc lập này là các trường hợp tối thiểu nhất để đảm bảo 100% độ bao phủ ứng với độ đo yêu cầu. Với mỗi đường đi, chúng ta sẽ sinh ra một ca kiểm thử sao cho khi nó được dùng để kiểm thử thì đường đi này được thực thi. Việc sinh các đầu vào cho các ca kiểm thử này là một bài toán thú vị. Chúng ta sẽ chọn một bộ đầu vào sao cho thỏa mãn các điểm quyết định có trong đường đi tương ứng. Giá trị đầu ra mong muốn ứng với mỗi bộ đầu vào của mỗi ca kiểm thử cũng sẽ được tính toán. Đây là bài toán khó và thường chỉ có các chuyên gia phân tích chương trình mới có thể trả lời chính xác giá trị này. Cuối cùng, các ca kiểm thử được chạy nhằm phát hiện ra các lỗi của chương trình/đơn vị chương trình cần kiểm thử. Khi một lỗi được phát hiện bởi một ca kiểm thử nào đó, nó sẽ được thông báo tới lập trình viên tương ứng. Lập trình viên sẽ tiến hành sửa lỗi (phát hiện vị trí của lỗi ở câu lệnh nào và sửa nó). Trong trường hợp này, chúng ta không chỉ thực hiện lại ca kiểm thử phát hiện ra lỗi này mà phải thực hiện lại tất cả các ca kiểm thử của đơn vị chương trình. Lý do chúng ta phải thực hiện công việc này là vì khi sửa lỗi này có thể gây ra một số lỗi khác.

Việc áp dụng phương pháp kiểm thử dòng điều khiển là khó và tốn kém hơn các phương pháp kiểm thử hộp đen (phân hoạch tương đương, phân tích giá trị biên, bảng quyết định, ...). Để áp dụng kỹ thuật này, chúng ta cần đội ngũ nhân lực về kiểm thử có kiến thức và kỹ năng tốt. Hơn nữa, chúng ta cần một sự đầu tư lớn về các nguồn lực

khác (tài chính, thời gian, ...) mới có thể thực hiện tốt phương pháp này. Đây là một yêu cầu khó và không nhiều công ty phần mềm đáp ứng được. Kiểm thử dòng điều khiển tự động hứa hẹn sẽ là một giải pháp tốt nhằm giúp cho các công ty giải quyết những khó khăn này. Hiện nay, đã có nhiều công cụ hỗ trợ một phần hoặc hoàn toàn các bước trong phương pháp này. Một môn trường lập trình mới nơi mà các công cụ lập trình trực quan được tích hợp với công cụ kiểm thử tự động nhằm giải quyết những khó khăn nêu trên. Eclipse1 là một ví dụ điển hình về xu hướng này. Trong cá phiên bản hiện nay của môi trường này đã được tích hợp công cụ kiểm thử tự động có tên là JUnit2. Công cụ kiểm thử này chưa hỗ trợ việc sinh các ca kiểm thử tự động nhưng nó cho phép chúng ta viết các kịch bản kiểm thử độc lập với mã nguồn và thực thi chúng một cách tự động trên một môi trường thống nhất.

Câu hỏi cuối bài:

Câu 1. Thế nào là độ đo kiểm thử?

Câu 2. Trình bày độ đo C1? Cho ví dụ?

Câu 3. Trình bày độ đo C2? Cho ví dụ?

Câu 4. Trình bày độ đo C3? Cho ví dụ?

Câu 5. Trình bày độ đo vòng lặp? Cho ví dụ?

Bài 7: Kiểm thử dòng dữ liệu (Số tiết: 03 tiết)

4.3 Kiểm thử dòng dữ liệu

Kiểm thử dòng điều khiển (control flow testing) và kiểm thử dòng dữ liệu (data flow testing) được xem là hai phương pháp chủ yếu trong chiến lược kiểm thử hộp trắng nhằm phát hiện các lỗi tiềm tàng bên trong các chương trình/đơn vị chương trình. Phương pháp kiểm thử dòng điều khiển cho phép sinh ra các ca kiểm thử tương ứng với các đường đi (dòng điều khiển) của chương trình. Tuy nhiên, chỉ áp dụng phương pháp này là chưa đủ để phát hiện tất cả các lỗi tiềm ẩn bên trong chương trình. Trong thực tế, các lỗi thường hay xuất hiện tại các biến được sử dụng trong chương trình/đơn vị chương trình. Kiểm thử dòng dữ liệu cho phép ta phát hiện những lỗi này. Bằng cách áp dụng cả hai phương pháp này, chúng ta khá tự tin về chất lượng của sản phẩm phần mềm.

4.3.1 Các vấn đề phổ biến về dòng dữ liệu

Trong quá trình lập trình, các lập trình viên có thể viết các câu lệnh “bất thường” hoặc không tuân theo chuẩn lập trình. Chúng ta gọi những bất thường liên quan đến việc khai báo, khởi tạo giá trị cho các biến và sử dụng chúng là các vấn đề về dòng dữ liệu của đơn vị chương trình. Ví dụ, một lập trình viên có thể sử dụng một biến mà không khởi tạo giá trị sau khi khai báo nó (`int x; if(x == 100){ ...}`).

Các vấn đề phổ biến về dòng dữ liệu có thể được phát hiện bằng phương pháp kiểm thử dòng dữ liệu tĩnh. Theo Fosdick và Osterweil, các vấn đề này được chia thành ba loại như sau:

- **Gán giá trị rồi gán tiếp giá trị (Loại 1):** Ví dụ, Hình 4.8 chứa hai câu lệnh tuần tự $x = f1(y); x = f2(z);$ với $f1$ và $f2$ là các hàm đã định nghĩa trước và y, z lần lượt là các tham số đầu vào của các hàm này. Chúng ta có thể xem xét hai câu lệnh tuần tự này với các tình huống sau:
 - Khi câu lệnh thứ hai được thực hiện, giá trị của biến x được gán và câu lệnh đầu không có ý nghĩa.
 - Lập trình viên có thể có nhầm lẫn ở câu lệnh đầu. Câu lệnh này có thể là gán giá trị cho một biến khác như là $w = f1(y).$
 - Có thể có nhầm lẫn ở câu lệnh thứ hai. Lập trình viên định gán giá trị cho một biến khác như là $w = f2(z).$
 - Một hoặc một số câu lệnh giữa hai câu lệnh này bị thiếu. Ví dụ như câu lệnh $w = f3(x).$

```

      .
      .
      x = f1(y);
      x = f2(z);
      .
      .
  
```

Hình 4.8: Tuần tự các câu lệnh có vấn đề thuộc loại 1

Chỉ có lập trình viên và một số thành viên khác trong dự án mới có thể trả lời một cách chính xác vấn đề trên thuộc trường hợp nào trong bốn tình huống trên. Mặc dù vậy, những vấn đề tương tự như ví dụ này là khá phổ biến và chúng ta cần phân tích mã nguồn để phát hiện ra chúng.

- **Chưa gán giá trị nhưng được sử dụng (Loại 2):** Ví dụ, Hình 4.9 chứa ba câu lệnh tuần tự với y là một biến đã được khai báo và gán giá trị ($y = f(x1);$). Trong trường hợp này, biến z chưa được gán giá trị khởi tạo nhưng đã được sử dụng

trong câu lệnh để tính giá trị của biến x ($x = y + z$). Chúng ta cũng có thể lý giải vấn đề này theo các tình huống sau:

- Lập trình viên có thể bỏ quên lệnh gán giá trị cho biến z trước câu lệnh tính toán giá trị cho biến x . Ví dụ, $z = f_2(x_2)$, với f_2 là một hàm đã được định nghĩa và x_2 là một biến đã được khai báo và gán giá trị.
- Có thể có sự nhầm lẫn giữa biến z với một biến đã được khai báo và gán giá trị. Ví dụ, $x = y + x_2$

```
⋮  
⋮  
y=f(x1);  
int z;  
x=y+z;  
⋮  
⋮
```

Hình 4.9: Tuần tự các câu lệnh có vấn đề thuộc loại 2.

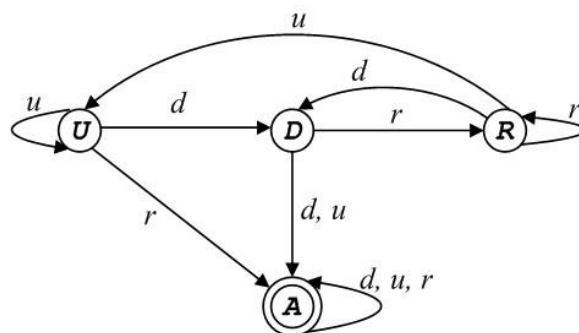
- **Đã được khai báo và gán giá trị nhưng không được sử dụng (Loại 3):** Nếu một biến đã được khai báo và gán giá trị nhưng không hề được sử dụng (trong các câu lệnh tính toán hoặc trong các biểu thức điều kiện), chúng ta cần xem xét cẩn thận vấn đề này. Tương tự như các trường hợp trên, các tình huống sau có thể được sử dụng để lý giải cho vấn đề này:
 - Có sự nhầm lẫn giữa biến này và một số biến khác được sử dụng trong chương trình. Trong thiết kế, biến này được sử dụng nhưng nó đã bị thay thế (do nhầm lẫn) bởi một biến khác.
 - Biến này thực sự không được sử dụng trong chương trình. Lúc đầu lập trình viên định sử dụng nó như là một biến tạm thời hoặc biến trung gian nhưng sau đó lại không cần dùng. Lập trình viên này đã quên xóa các câu lệnh khai báo và gán giá trị cho biến này.

Huang đã giới thiệu một phương pháp để xác định những bất thường trong việc sử dụng các biến dữ liệu bằng cách sử dụng sơ đồ chuyển trạng thái ứng với mỗi biến dữ liệu của chương trình. Các thành phần của sơ đồ chuyển trạng thái của một chương trình ứng với mỗi biến gồm:

- Các trạng thái, gồm:
 - U : biến chưa được gán giá trị
 - D: biến đã được gán giá trị nhưng chưa được sử dụng

- R: biến đã được sử dụng
- A: trạng thái lỗi
- Các hành động, gồm:
 - d: biến được gán giá trị
 - r: biến được sử dụng
 - u: biến chưa được gán giá trị hoặc được khai báo lại và chưa được gán giá trị

Hình 4.10 mô tả sơ đồ chuyển trạng thái của một biến trong một chương trình/đơn vị chương trình. Ban đầu, biến này đã được khai báo và chưa được gán giá trị nên trạng thái của chương trình là U. Tại trạng thái này, nếu biến này được sử dụng (hành động r) thì chương trình có vấn đề và trạng thái của chương trình là A. Ngược lại, trạng thái U vẫn được giữ nguyên nếu các câu lệnh tiếp theo vẫn chưa chứa lệnh gán giá trị cho biến này (hành động u). Cho đến khi gặp câu lệnh gán giá trị cho biến này (hành động d), trạng thái của chương trình được chuyển thành D. Nếu biến này được sử dụng ở các câu lệnh tiếp theo (hành động r) thì trạng thái của chương trình chuyển thành R. Ngược lại, nếu các câu lệnh tiếp theo lại gán lại giá trị cho biến (hành động d) hoặc khai báo lại biến này và không gán giá trị cho nó (hành động u) thì xảy ra vấn đề và trạng thái của chương trình là A. Tại trạng thái này, mọi hành động (d, u và r) xảy ra đều không thay đổi trạng thái của chương trình. Tại trạng thái R, nếu biến này vẫn tiếp tục được sử dụng ở các lệnh tiếp theo (hành động r) thì trạng thái của chương trình vẫn không thay đổi. Ngược lại, nếu xuất hiện câu lệnh gán lại giá trị cho biến (hành động d) thì trạng thái của chương trình quay về D. Trong trường hợp xuất hiện câu lệnh khai báo lại biến này và không gán giá trị cho nó (hành động u) thì chương trình được chuyển từ trạng thái R sang trạng thái U.



Hình 4.10: Sơ đồ chuyển trạng thái của một biến.

Như vậy, các vấn đề với dòng dữ liệu thuộc loại 1 ứng với trường hợp dd xảy ra trong sơ đồ chuyển trạng thái. Các vấn đề thuộc loại 2 ứng với trường hợp ur và loại 3

ứng với trường hợp du. Để phát hiện các vấn đề này, chúng ta sẽ tiến hành xây dựng sơ đồ chuyển trạng thái ứng với mỗi biến như Hình 4.10. Nếu trạng thái A xuất hiện thì chương trình có vấn đề về dòng dữ liệu. Trong trường hợp này, chúng ta cần kiểm tra lại mã nguồn, tìm nguyên nhân của tình huống này và sửa lỗi. Tuy nhiên, cho dù trạng thái lỗi (trạng thái A) không xuất hiện trong quá trình phân tích chương trình, chúng ta vẫn không đảm bảo được rằng chương trình không còn lỗi. Các lỗi có thể xảy ra trong quá trình gán/gán lại giá trị cho các biến và trong quá trình sử dụng chúng (trong các câu lệnh tính toán, trong các biểu thức điều kiện, ...). Để phát hiện những lỗi này, chúng ta cần phương pháp kiểm thử dòng dữ liệu động. Phương pháp này sẽ được trình bày chi tiết ở các mục tiếp theo.

4.3.2 Tổng quan về kiểm thử dòng dữ liệu

Kiểm thử dòng dữ liệu tĩnh không đảm bảo phát hiện tất cả các lỗi liên quan đến việc khởi tạo, gán giá trị mới và sử dụng các biến (trong các câu lệnh tính toán và các biểu thức điều kiện như trong các lệnh rẽ nhánh và lặp). Nó được xem như một bước tiền xử lý mã nguồn trước khi áp dụng phương pháp kiểm thử dòng dữ liệu động. Chúng ta hy vọng các lỗi còn lại (không được phát hiện bởi kiểm thử dòng dữ liệu tĩnh) sẽ được phát hiện bởi phương pháp này.

Các biến xuất hiện trong một chương trình theo một số trường hợp như: khởi tạo, gán giá trị mới, tính toán, và dùng làm điều khiển trong các biểu thức điều kiện của các lệnh rẽ nhánh và lặp. Có hai lý do chính khiến chúng ta phải tiến hành kiểm thử dòng dữ liệu của chương trình:

- Chúng ta cần chắc chắn rằng một biến phải được gán đúng giá trị, tức là chúng ta phải xác định được một đường đi của biến từ một điểm bắt đầu nơi nó được định nghĩa đến điểm mà biến đó được sử dụng. Mỗi khi chưa có các ca kiểm thử để kiểm tra đường đi này, chúng ta không thể tự tin khẳng định là biến này đã được gán giá trị đúng.
- Ngay cả khi gán đúng giá trị cho biến thì các giá trị được sinh ra chưa chắc đã chính xác do tính toán hoặc các biểu thức điều kiện sai (biến được sử dụng sai).

Để áp dụng phương pháp kiểm thử dòng dữ liệu động, chúng ta phải xác định các đường dẫn chương trình có một điểm đầu vào và một điểm đầu ra sao cho nó bao phủ việc gán

giá trị và sử dụng mỗi biến của chương trình/đơn vị chương trình cần kiểm thử. Cụ thể, chúng ta cần thực hiện các bước sau:

- Xây dựng đồ thị dòng dữ liệu của chương trình/đơn vị chương trình
- Chọn một hoặc một số tiêu chí kiểm thử dòng dữ liệu
- Xác định các đường dẫn chương trình phù hợp với tiêu chí kiểm thử đã chọn
- Lấy ra các biểu thức điều kiện từ tập các đường đi, thực hiện giải các biểu thức điều kiện để có được các giá trị đầu vào cho các ca kiểm thử tương ứng với các đường đi này và tính toán giá trị đầu ra mong đợi của mỗi ca kiểm thử.
- Thực hiện các ca kiểm thử để xác định các lỗi (có thể có) của chương trình.
- Sửa các lỗi (nếu có) và thực hiện lại tất cả các ca kiểm thử trong trường hợp bước trên phát hiện ra lỗi.

Như vậy, các bước trong quy trình kiểm thử dòng dữ liệu động cũng tương tự như các bước trong quy trình kiểm thử dòng điều khiển. Tuy nhiên, đồ thị dòng điều khiển hoàn toàn khác với đồ thị dòng dữ liệu. Hơn nữa, các tiêu chí kiểm thử (độ đo kiểm thử) ứng với hai kỹ thuật này cũng khác nhau. Kết quả là, các đường đi và phương pháp chọn chúng từ đồ thị cũng khác nhau.

4.3.3 Đồ thị dòng dữ liệu

Trong mục này, chúng ta sẽ tìm hiểu về đồ thị dòng dữ liệu và cách xây dựng nó từ đơn vị chương trình. Trong thực tế, chúng ta có thể sử dụng các tiện ích của các chương trình dịch để xây dựng đồ thị này một cách tự động. Đồ thị dòng dữ liệu của một chương trình/đơn vị chương trình sử dụng các khái niệm liên quan đến việc định nghĩa và sử dụng các biến.

Định nghĩa 4.1. (Định nghĩa của một biến.) Một câu lệnh thực hiện việc gán giá trị cho một biến được gọi là câu lệnh định nghĩa của biến đó (ký hiệu là *def*). Ví dụ, trong hàm `VarTypes` viết bằng ngôn ngữ C như Hình 4.11, câu lệnh `i = x` (dòng 3) là một ví dụ về việc định nghĩa của biến `i`.

Định nghĩa 4.2. (Chưa định nghĩa một biến.) Một câu lệnh khai báo một biến nhưng chưa thực hiện việc gán giá trị được gọi là *undef* với biến đó. Ví dụ, trong hàm `VarTypes` viết bằng ngôn ngữ C như Hình 4.11, câu lệnh `iptr = malloc(sizeof(int));` (dòng 4) là một ví dụ về việc chưa định nghĩa biến `iptr`. Câu lệnh tiếp theo (`*iptr = i + x;` (dòng 5)) là một định nghĩa của biến này. Tuy nhiên, câu lệnh ở dòng 9 lại định nghĩa lại biến `iptr` và vì vậy câu lệnh này là một ví dụ khác về việc chưa định nghĩa biến này.

Định nghĩa 4.3. (Biến được sử dụng.) Một câu lệnh sử dụng một biến (để tính toán hoặc để kiểm tra các điều kiện) được gọi là use của biến đó.

Định nghĩa 4.4. (Biến được sử dụng để tính toán.) Một câu lệnh sử dụng một biến để tính toán giá trị của một biến khác được gọi là c-use với biến đó. Ví dụ, trong hàm VarTypes như Hình 4.11, câu lệnh `*iptr = i + x;` (dòng 5) là c-use ứng với biến `i` và biến `x`.

Định nghĩa 4.5. (Biến được sử dụng để kiểm tra các điều kiện.) Một câu lệnh sử dụng một biến trong các biểu thức điều kiện (câu lệnh rẽ nhánh, lặp, ...) được gọi là p-use với biến đó. Ví dụ, trong hàm VarTypes như Hình 4.11, câu lệnh `if (*iptr > y) ...` (dòng 6) là p-use ứng với biến `iptr` và biến `y`.

```
int VarTypes(int x, int y){
1.  int i;
2.  int *iptr;
3.  i = x;
4.  iptr = malloc(sizeof(int));
5.  *iptr = i + x;
6.  if (*iptr > y)
7.      return (x);
8.  else {
9.      iptr = malloc(sizeof(int));
10.     *iptr = x + y;
11.     return(*iptr);
12. } //end if
} //the end
```

Hình 4.11: Ví dụ về định nghĩa và sử dụng các biến.

Định nghĩa 4.6. (Đồ thị dòng dữ liệu.) Đồ thị dòng dữ liệu của một chương trình/đơn vị chương trình là một đồ thị có hướng $G = \langle N, E \rangle$, với:

N là tập các đỉnh tương ứng với các câu lệnh `def` hoặc c-use của các biến được sử dụng trong đơn vị chương trình. Đồ thị G có hai đỉnh đặc biệt là đỉnh bắt đầu (tương ứng với lệnh `def` của các biến tham số) và đỉnh kết thúc đơn vị chương trình.

- E là tập các cạnh tương ứng với các câu lệnh p-use của các biến.

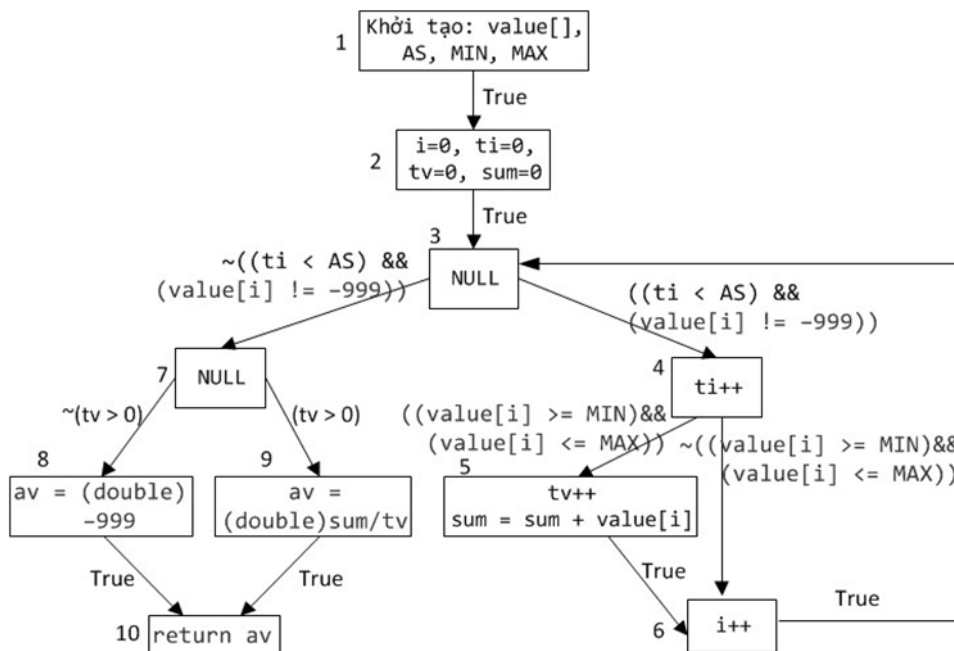
```

double ReturnAverage(int value[], int AS, int MIN, int MAX){
    int i, ti, tv, sum;
    double av;
    i = 0; ti = 0; tv = 0; sum = 0;
    while (ti < AS && value[i] != -999) {
        ti++;
        if (value[i] >= MIN && value[i] <= MAX) {
            tv++;
            sum = sum + value[i];
        }
        i++;
    } //end while
    if (tv > 0)
        av = (double)sum/tv;
    else
        av = (double) -999;
    return (av);
} //the end

```

Hình 4.12: Mã nguồn của hàm ReturnAverage bằng ngôn ngữ C.

Chúng ta sẽ tìm hiểu cách xây dựng đồ thị dòng dữ liệu của một đơn vị chương trình thông qua ví dụ về hàm ReturnAverage bằng ngôn ngữ C trong Hình 4.12. Đồ thị dòng dữ liệu của hàm này được trình bày ở Hình 4.13.



Hình 4.13: Đồ thị dòng dữ liệu của hàm ReturnAverage trong Hình 4.12.

Trong đồ thị này, đỉnh bắt đầu (đỉnh 1) thể hiện việc định nghĩa (def) các biến tham số value[], AS, MIN, MAX. Đỉnh 2 tương ứng với việc định nghĩa các biến cục bộ i, ti, tv, sum. Tiếp đến, chúng ta sử dụng đỉnh NULL (đỉnh 3) với mục đích đây là điểm bắt đầu của vòng lặp while. Chú ý rằng đỉnh NULL không thực hiện bất cứ tính toán hay định

nghĩa cho biến nào cả. Đỉnh này cũng sẽ là điểm kết thúc của vòng lặp này sau khi biểu thức điều kiện của vòng lặp là sai. Câu lệnh `ti++`; được biểu diễn bằng đỉnh 4. Cạnh (3, 4) biểu diễn biểu thức điều kiện của vòng lặp `((ti < AS) && (value[i] != -999))`. Ý nghĩa của cạnh này là nếu biểu thức này đúng thì chuyển từ đỉnh 3 sang thực hiện đỉnh 4. Các câu lệnh `tv++`; và `sum = sum + value[i]`; được biểu diễn bởi đỉnh 5. Để chuyển từ đỉnh 4 sang đỉnh 5 (cạnh (4, 5)), điều kiện tương ứng với cạnh này phải đúng `((value[i] >= MIN) && (value[i] <= MAX))` là đúng). Tiếp theo, đỉnh 6 biểu diễn câu lệnh `i++`; . Trong trường hợp biểu thức sau là sai `((value[i] >= MIN) && (value[i] <= MAX))`, đồ thị sẽ chuyển từ đỉnh 4 sang đỉnh 6 (cạnh (4, 6)). Từ đỉnh 5, đồ thị sẽ chuyển đến đỉnh 6 (cạnh (5, 6)) vì không có biểu thức điều kiện của cạnh này. Vòng lặp `while` sẽ kết thúc khi điều kiện `((ti < AS) && (value[i] != -999))` là sai. Khi đó, đồ thị sẽ chuyển từ đỉnh 3 sang đỉnh 7 (cạnh (3, 7)). Đỉnh 7 là đỉnh NULL tương ứng với câu lệnh `if (tv > 0)`. Đỉnh 8 là biểu diễn của câu lệnh `av = (double) -999`; trong khi câu lệnh `av = (double)sum/tv`; được biểu diễn bằng đỉnh 9. Từ đỉnh 7, nếu điều kiện `tv > 0` là đúng thì đồ thị chuyển sang đỉnh 9 (cạnh (7, 9)). Ngược lại, đồ thị chuyển từ đỉnh 7 sang đỉnh 8 (cạnh (7, 8)). Cuối cùng, đỉnh 10 được sử dụng để biểu diễn câu lệnh `return(av)`; . Từ các đỉnh 8 và 9, đồ thị chuyển đến đỉnh 10 vì không có biểu thức điều kiện cho các cạnh (8, 10) và (9, 10).

Câu hỏi cuối bài:

Câu 1. Trình bày các vấn đề phổ biến về dòng dữ liệu?

Câu 2. Thế nào là kiểm thử dòng dữ liệu tĩnh? Kiểm thử dòng dữ liệu động?

Câu 3. Vẽ sơ đồ chuyển trạng thái tổng quát?

Câu 4. Định nghĩa đồ thị dòng dữ liệu?

Câu 5. Các bước thực hiện kiểm thử dòng dữ liệu?

Bài 8: Các độ đo cho kiểm thử dòng dữ liệu (Số tiết: 03 tiết)

4.3.4 Các khái niệm về dòng dữ liệu

Sau khi xây dựng đồ thị dòng dữ liệu của đơn vị chương trình, chúng ta cần xác định các đường đi của đơn vị chương trình của mỗi biến dữ liệu ứng với các độ đo kiểm thử. Trong mỗi đường dẫn này, biến dữ liệu được định nghĩa tại một đỉnh nào đó và được sử

dụng tại các câu lệnh tiếp theo ứng với các đỉnh hoặc các cạnh của đường đi này. Trong mục này, chúng ta sẽ tìm hiểu một số khái niệm cơ bản về dòng dữ liệu. Các khái niệm này sẽ được sử dụng để phục vụ mục đích trên.

Định nghĩa 4.7. (Global c-use.) Giả sử biến x được sử dụng để tính toán (c-use) tại đỉnh i của đồ thị dòng dữ liệu. Việc sử dụng biến x tại đỉnh i được gọi là Global c-use nếu x đã được định nghĩa ở các đỉnh trước đó.

Ví dụ: Trong đồ thị dòng dữ liệu của hàm ReturnAverage được mô tả ở Hình 4.12, việc sử dụng biến tv tại đỉnh 9 là Global c-use vì biến này đã được định nghĩa tại các đỉnh 2 và 5.

Định nghĩa 4.8. (Def-clear path.) Giả sử biến x được định nghĩa (def) tại đỉnh i và được sử dụng tại đỉnh j . Một đường đi từ i đến j ký hiệu là $(i - n_1 - \dots - n_m - j)$ với $m \geq 0$ được gọi là Def-clear path ứng với biến x nếu biến này không được định nghĩa tại các đỉnh từ n_1 đến n_m .

Ví dụ: Trong đồ thị dòng dữ liệu của hàm ReturnAverage được mô tả ở Hình 4.13, đường đi $(2 - 3 - 4 - 5)$ là một Def-clear path ứng với biến tv vì biến này được định nghĩa tại đỉnh 2, được sử dụng tại đỉnh 5, và không được định nghĩa tại các đỉnh 3 và 4. Tương tự, đường đi $(2 - 3 - 4 - 6 - 3 - 4 - 6 - 3 - 4 - 5)$ cũng là một Def-clear path ứng với biến tv .

Định nghĩa 4.9. (Global def.) Một đỉnh i được gọi là Global def của biến x nếu đỉnh này định nghĩa biến x (def) và có một Def-clear path của x từ đỉnh i tới đỉnh chứa một Global c-use hoặc cạnh chứa một p-use của biến này.

Bảng 4.11 liệt kê tất cả các Global def và Global c use xuất hiện trong đồ thị dòng dữ liệu ở Hình 4.13. Trong bảng này, def(i) được sử dụng để chỉ tập các biến có Global def tại đỉnh i . Tương tự, c-use(i) chỉ tập các biến có Global c-use tại đỉnh i . Tất cả các điều kiện ứng với các cạnh và các p-use xuất hiện trong đồ thị dòng dữ liệu ở Hình 4.13 cũng được liệt kê trong Bảng 4.12, với p-use(i, j) là tập các biến có p-use tại cạnh (i, j) .

Định nghĩa 4.10. (Simple path.) Một đường đi trong đồ thị dòng dữ liệu được gọi là một Simple path nếu các đỉnh chỉ xuất hiện đúng một lần trừ đỉnh đầu và đỉnh cuối.

Ví dụ: Trong đồ thị dòng dữ liệu của hàm ReturnAverage được mô tả ở Hình 4.13, các đường đi $(2 - 3 - 4 - 5)$ và $(3 - 4 - 6 - 3)$ là các Simple paths.

Định nghĩa 4.11. (Loop-free path.) Một đường đi trong đồ thị dòng dữ liệu được gọi là một Loop-free path nếu các đỉnh chỉ xuất hiện đúng một lần.

Định nghĩa 4.12. (Complete-path.) Một đường đi được gọi là một Complete-path nếu nó có điểm bắt đầu và điểm kết thúc chính là điểm bắt đầu và điểm kết thúc của đồ thị dòng dữ liệu.

Định nghĩa 4.13. (Du-path.) Một đường đi $(n_1 - n_2 - \dots - n_j - n_k)$ được gọi là một Du-path (definition-use path) ứng với biến x nếu đỉnh n_1 là Global def của biến x và:

- đỉnh n_k có một Global c-use với biến x và $(n_1 - n_2 - \dots - n_j - n_k)$ là một Def-clear simple path với biến x , hoặc
- cạnh (n_j, n_k) có p-use với biến x và $(n_1 - n_2 - \dots - n_j)$ là Def-clear loop-free path với biến này.

Ví dụ: Trong đồ thị dòng dữ liệu của hàm ReturnAverage (Hình 4.13), đường đi $(2 - 3 - 4 - 5)$ là một Du-path ứng với biến tv vì đỉnh 2 là Global def của biến tv , đỉnh 5 có Global c-use với biến này và $(2 - 3 - 4 - 5)$ là một Def-clear simple path với biến tv . Một ví dụ khác, đường đi $(2 - 3 - 7 - 9)$ cũng là một Du-path ứng với biến tv vì đỉnh 2 là Global def của biến này, cạnh $(7, 9)$ có p-use với biến tv và $(2 - 3 - 7)$ là một Def-clear loop-free path với tv .

4.3.5 Các độ đo cho kiểm thử dòng dữ liệu

Các độ đo hay các tiêu chí kiểm thử dòng dữ liệu là đầu vào cùng với đồ thị dòng dữ liệu nhằm xác định các đường đi cho mục đích kiểm thử tương ứng. Trong mục này, chúng ta sẽ tìm hiểu các độ đo phổ biến đang được sử dụng trong kiểm thử dòng dữ liệu. Trước hết ta có ba độ đo đơn giản là All-defs, All-c-uses, và All-p-uses tương ứng với tất cả các đường đi, đường đi qua tất cả các đỉnh, và đường đi qua tất cả các cạnh. Các độ đo tiếp theo sẽ giả sử ta đã xác định được các đỉnh xác định và sử dụng và các Du-path cho từng biến. Các định nghĩa sau cũng giả sử T là một tập hợp các (đoạn) đường trong đồ thị dòng dữ liệu $G = \langle N, E \rangle$ và V là tập các biến được sử dụng trong đơn vị chương trình.

All-defs: Mỗi một biến $x \in V$ và mỗi đỉnh $i \in N$, giả sử x có một Global def tại i , chọn một Complete-path chứa một Def-clear path từ đỉnh i tới đỉnh j sao cho tại j có chứa một Global c-use của x , hoặc cạnh (j, k) chứa một p-use của biến x . Điều này có nghĩa là đối với mỗi một định nghĩa (def) của x tại một đỉnh ta cần ít nhất một đường đi xuất phát từ đỉnh đó tới một đỉnh khác sử dụng biến x sao cho đường đi này chứa một Def-clear path của biến đó và thuộc về một Complete-path nào đó.

Ví dụ: Biến tv có hai Global def tại các đỉnh 2 và 5 (Hình 4.13, Bảng 4.11 và Bảng 4.12). Trước hết, ta quan tâm đến Global def tại đỉnh 2. Chúng ta thấy rằng có một Global c-use của biến tv tại đỉnh 5 và tồn tại một Def-clear path (2 - 3 - 4 - 5) từ đỉnh 2 tới đỉnh 5. Ta cũng có được một Complete-path (1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 9 - 10) chứa đường đi này. Do vậy ta kết luận (2 - 3 - 4 - 5) thỏa mãn độ đo All-defs. Tương tự, đường đi (2 - 3 - 7 - 8) của biến tv cũng thỏa mãn độ đo All-defs do có một Global def tại đỉnh 2, có một p-use tại cạnh (7, 8), có một Def-clear path là (2 - 3 - 7 - 8) từ đỉnh 2 tới cạnh (7, 8) và nó thuộc về một Complete-path (1 - 2 - 3 - 7 - 8 - 10). Quan tâm đến Global def tại đỉnh 5 và có một Global c-use tại đỉnh 9, (5 - 6 - 3 - 7 - 9) là một Def-clear path và tồn tại một Complete-path là (1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 9 - 10) chứa đường đi này. Vì vậy (5 - 6 - 3 - 7 - 9) cũng thỏa mãn độ đo All-defs. Để quá trình kiểm thử dòng dữ liệu thỏa mãn độ đo All-defs, chúng ta cần tiến hành một cách tương tự với các biến còn lại.

All-c-uses: Với mỗi một biến x và mỗi đỉnh i sao cho i là Global def với biến x, chọn các Complete-path bao gồm các Def-clear path từ đỉnh i tới tất cả các đỉnh j sao cho j là Global c-use của x. Điều này có nghĩa là cứ mỗi định nghĩa (def) của x ta tìm tất cả các đường đi xuất phát từ def của x tới tất cả các c-use của biến x sao cho các đường đi này có chứa một Def-clear path của x và thuộc về một Complete-path nào đó.

Ví dụ: Ta sẽ tìm tất cả các đường đi thỏa mãn độ đo All-c-uses ứng với biến ti (Hình 4.13, Bảng 4.11, và Bảng 4.12). Chúng ta tìm thấy hai Global def của biến này tại các đỉnh 2 và 4. Với đỉnh 2, có một Global c-use của biến ti tại đỉnh 4. Tuy nhiên, với Global def của biến này tại đỉnh 4, ta không tìm thấy Global c-use nào của biến ti. Từ đỉnh 2, ta có một Def-clear path tới đỉnh 4 là (2 - 3 - 4). Chúng ta có thể tìm thấy bốn Complete-path từ đồ thị dòng dữ liệu (Hình 4.13) chứa đường đi này như sau:

- (1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 8 - 10),
- (1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 9 - 10),
- (1 - 2 - 3 - 4 - 6 - 3 - 7 - 8 - 10), và
- (1 - 2 - 3 - 4 - 6 - 3 - 7 - 9 - 10).

Chúng ta có thể chọn một hoặc một số trong bốn đường đi trên để đảm bảo độ đo này ứng với biến ti. Để quá trình kiểm thử dòng dữ liệu thỏa mãn độ đo All-c-uses, chúng ta cần tiến hành một cách tương tự với các biến còn lại (i, tv, và sum).

All-p-uses: Với mỗi một biến x và mỗi đỉnh i sao cho i là Global def với biến x , chọn các Complete-path bao gồm các Def-clear path từ đỉnh i tới tất cả các cạnh (j, k) sao cho có một p-use của x tại cạnh này. Điều này có nghĩa là cứ mỗi định nghĩa (def) của x ta tìm tất cả các đường đi xuất phát từ def của x tới tất cả các p-use của biến đó sao cho các đường đi này có chứa một Def-clear path của x và thuộc về một Complete-path nào đó.

Ví dụ: Ta sẽ tìm tất cả các đường đi thỏa mãn độ đo All-p-uses ứng với biến tv (Hình 4.13, Bảng 4.11, và Bảng 4.12). Biến tv có hai Global def tại các đỉnh 2 và 5. Tại đỉnh 2, ta có hai p-use của biến này tại các cạnh $(7, 8)$ và $(7, 9)$. Dễ thấy $(2 - 3 - 7 - 8)$ là một Def-clear path của tv từ đỉnh 2 đến cạnh $(7, 8)$ và $(2 - 3 - 7 - 9)$ là một Def-clear path của tv từ đỉnh 2 đến cạnh $(7, 9)$. Tương tự, $(5 - 6 - 3 - 7 - 8)$ là một Def-clear path của tv từ đỉnh 5 đến cạnh $(7, 8)$ và $(5 - 6 - 3 - 7 - 9)$ là một Def-clear path của tv từ đỉnh 5 đến cạnh $(7, 9)$. Chúng ta có thể tìm thấy bốn Complete-path từ đồ thị dòng dữ liệu (Hình 4.13) chứa các đường đi này như sau:

- $(1 - 2 - 3 - 7 - 8 - 10)$,
- $(1 - 2 - 3 - 7 - 9 - 10)$,
- $(1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 8 - 10)$, và
- $(1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 9 - 10)$.

All-p-uses/Some-c-uses: Độ đo này tương tự như độ đo All-p-uses ngoại trừ trường hợp khi có một nghĩa (def) của biến x mà không có một p-use của biến này. Trong trường hợp này, ta sử dụng độ đo Some-c-uses được định nghĩa như sau.

Định nghĩa 4.14. (Some-c-uses.) Với mỗi một biến x và mỗi đỉnh i sao cho i là Global def với biến x , chọn các Complete-path bao gồm các Def-clear path từ đỉnh i tới một số đỉnh j sao cho j là Global c-use của x .

Ví dụ: Ta sẽ tìm tất cả các đường đi thỏa mãn độ đo All-p-uses/Some-c-uses ứng với biến i (Hình 4.13, Bảng 4.11, và Bảng 4.12). Biến i có hai Global def tại các đỉnh 2 và đỉnh 6. Dễ thấy rằng không có p-use của biến này (Hình 4.13). Vì vậy, ta quan tâm đến độ đo Some-c-uses của biến i . Tại đỉnh 2, có một Global c-use của i tại đỉnh 6 và có một Def-clear path $(2 - 3 - 4 - 5 - 6)$. Vì vậy, để thỏa mãn độ đo All-p-uses/Some-c-uses với biến này, ta chọn Complete-path $(1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 9 - 10)$ chứa đường đi trên.

All-c-uses/Some-p-uses: Độ đo này tương tự như độ đo All-c-uses ngoại trừ trường hợp khi có một nghĩa (def) của biến x mà không có một Global c-use của biến này. Trong trường hợp này, ta sử dụng độ đo Some-p-uses được định nghĩa như sau.

Định nghĩa 4.15. (Some-p-uses.) Với mỗi một biến x và mỗi đỉnh i sao cho i là Global def với biến x , chọn các Complete-path bao gồm các Def-clear path từ đỉnh i tới một số cạnh (j, k) sao cho có một p-use của x tại cạnh này.

Ví dụ: Ta sẽ tìm tất cả các đường đi thỏa mãn độ đo All-c-uses/Some-p-uses ứng với biến AS (Hình 4.13, Bảng 4.11, và Bảng 4.12). Biến AS chỉ có một Global def tại đỉnh 1. Dễ thấy rằng không có Global c-use của biến này (Hình 4.13). Vì vậy, ta quan tâm đến độ đo Some-p-uses của biến AS. Từ đỉnh 1, có các p-use của AS tại các cạnh $(3, 7)$ và $(3, 4)$ và có các Def-clear path tương ứng với hai cạnh này là $(1 - 2 - 3 - 7)$ và $(1 - 2 - 3 - 4)$. Có rất nhiều Complete-path chứa hai đường đi này. Ví dụ $(1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 9 - 10)$.

All-uses: Độ đo này bao gồm các đường đi được sinh ra từ các độ đo All-p-uses và All-c-uses (như đã định nghĩa ở trên). Điều này có nghĩa là với mỗi việc sử dụng (c-use hoặc p-use) của một biến thì có một đường đi từ định nghĩa (def) của biến đó tới các sử dụng của nó.

All-du-paths: Với mỗi một biến x và mỗi đỉnh i sao cho i là Global def với biến x , chọn các Complete-path chứa các tất cả các Du-path từ đỉnh i tới:

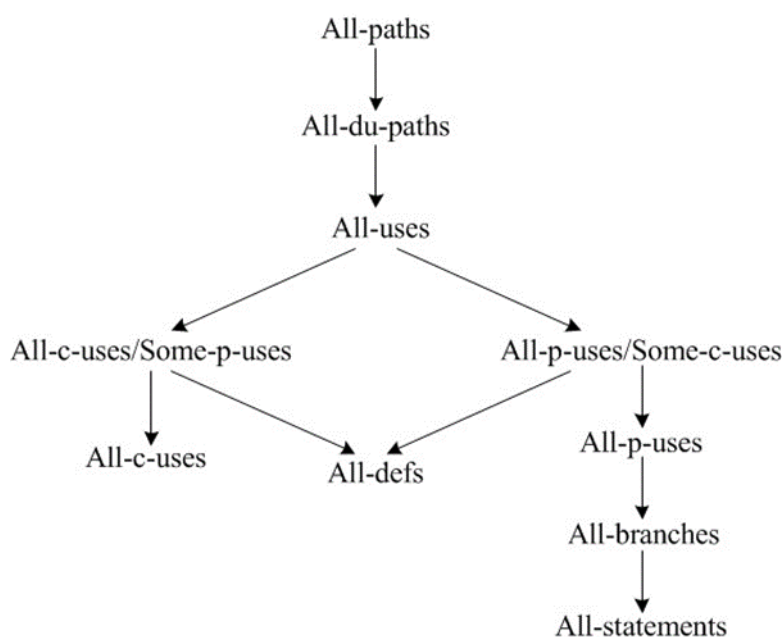
- tất cả các đỉnh j sao cho có một Global c-use của biến x tại j , và
- tất cả các cạnh (j, k) sao cho có một p-use của biến x tại (j, k) .

So sánh các độ đo của kiểm thử dòng dữ liệu: Sau khi tìm hiểu các độ đo về kiểm thử dòng dữ liệu, chúng ta cần so sánh mối quan hệ giữa chúng để trả lời câu hỏi “chúng ta nên chọn độ đo nào cho kiểm thử dòng dữ liệu?”. Với mỗi cặp độ đo ở trên, chúng có thể so sánh với nhau hoặc không. Rapps và Weyuker đã định nghĩa mối quan hệ “bao gồm” giữa hai độ đo như sau.

Định nghĩa 4.16. (Mối quan hệ bao gồm.) Cho hai độ đo $c1$ và $c2$, ta nói $c1$ bao gồm $c2$ nếu mọi đường đi đầy đủ (Complete-paths) sinh ra từ đồ thị dòng dữ liệu thỏa mãn $c1$ thì cũng thỏa mãn $c2$.

Định nghĩa 4.17. (Mối quan hệ bao gồm chặt.) Cho hai độ đo $c1$ và $c2$, ta nói $c1$ bao gồm chặt $c2$, ký hiệu là $c1 \rightarrow c2$, nếu $c1$ bao gồm $c2$ và tồn tại một số đường đi đầy đủ sinh ra từ đồ thị dòng dữ liệu thỏa mãn $c2$ nhưng không thỏa mãn $c1$.

Để thấy mối quan hệ bao gồm chặt (\rightarrow) có tính bắc cầu. Hơn nữa, với hai độ đo $c1$ và $c2$, có thể $c1$ không bao gồm chặt $c2$ và $c2$ cũng không bao gồm chặt $c1$. Trong trường hợp này ta nói hai độ đo này là không so sánh được. Frankl và Weyuker đã tổng kết các mối quan hệ giữa các độ đo cho kiểm thử dòng dữ liệu trong Hình 4.13. Với các độ đo như đã trình bày ở trên, All-du-paths là độ đo tốt nhất. Độ đo này bao gồm chặt độ đo All-uses. Tương tự, độ đo All-uses bao gồm chặt hai độ đo All-p-uses/Some-c-uses và All-c-uses/Some-p-uses trong khi chúng bao gồm chặt độ đo All-defs. Hơn nữa, độ đo All-p-uses/Some-c-uses bao gồm chặt độ đo All-p-uses trong khi độ đo All-c-uses/Some-p-uses bao gồm chặt độ đo All-c-uses. Tuy nhiên, chúng ta không thể tìm thấy mối quan hệ bao gồm chặt giữa hai độ đo All-c-uses và All-p-uses.



Hình 4.14: Mối quan hệ giữa các độ đo cho kiểm thử dòng dữ liệu.

4.3.6 Sinh các ca kiểm thử

Để tiến hành phương pháp kiểm thử dòng dữ liệu, trước hết chúng ta phải sinh đồ thị dòng dữ liệu của đơn vị chương trình. Với độ đo kiểm thử C , chúng ta sẽ xác định tất cả các đường đi đầy đủ (Complete-paths) thỏa mãn độ đo này. Tuy nhiên, không phải đường đi nào cũng có thể tìm được một bộ dữ liệu đầu vào để nó được thực thi khi chạy đơn vị chương trình. Nếu tồn tại một bộ dữ liệu đầu vào như vậy thì đường đi tương ứng được gọi là đường đi thực thi được. Như vậy, bài toán còn lại hiện nay là làm thế nào để sinh được bộ đầu vào cho từng đường đi đầy đủ trên. Bộ đầu vào này cùng với giá trị đầu ra mong đợi sẽ là ca kiểm thử cho đường đi này. Để trả lời câu hỏi này, ta xét ví dụ sau.

Ví dụ: Xét đường đi đầy đủ (1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 9) từ đồ thị dòng dữ liệu trong Hình 4.13. Từ đường đi này, ta sẽ xác định các biểu thức thuộc các p-uses nằm trên các cạnh của nó. Cụ thể, ta có các biểu thức sau:

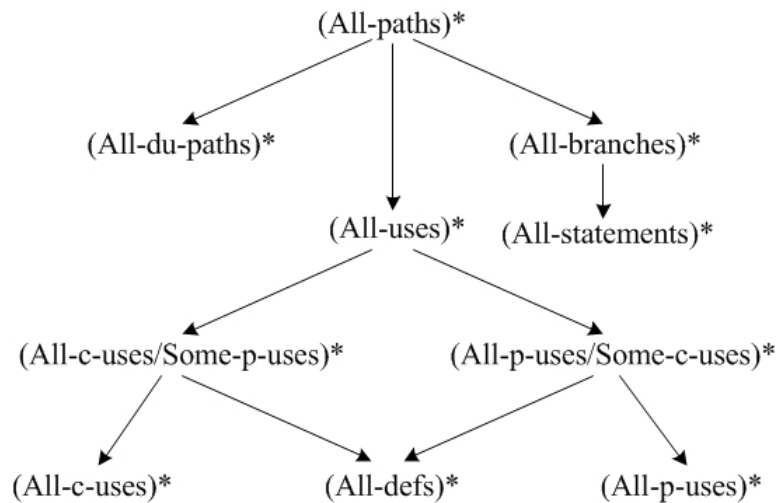
1. $((ti < AS) \ \&\& \ (value[i] \neq -999))$ (thuộc cạnh (3, 4)),
2. $((value[i] \geq MIN) \ \&\& \ (value[i] \leq MAX))$ (thuộc cạnh (4, 5)),
3. $\sim((ti < AS) \ \&\& \ (value[i] \neq -999))$ (thuộc cạnh (3, 7)), và
4. $tv > 0$ (thuộc cạnh (7, 9))

Chúng ta sẽ sinh ra các giá trị đầu vào và kiểm tra các giá trị đó sao cho thỏa mãn hết các điều kiện trên. Chú ý, các giá trị đầu vào ở đây là các tham số của hàm ReturnAverage (Hình 4.12), giá trị của biến i ở điều kiện (3) lớn hơn 1 so với giá trị nó tại các biểu thức (1) và (2), và giá trị của biến ti ở các điều kiện (2) và (3) lớn hơn 1 so với giá trị nó tại biểu thức (1). Ví dụ, bộ đầu vào $([1, -999], 2, 1, 2)$ thỏa mãn tất cả các điều kiện trên.

Để lựa chọn các bộ đầu vào ứng với các đường đi đầy đủ thỏa mãn một độ đo cho trước, chúng ta phải đảm bảo rằng các đường đi này là thực thi được. Giả sử PC là tập các đường đi đầy đủ của đơn vị chương trình ứng với độ đo C. Độ đo này chỉ có ích khi tồn tại ít nhất một đường đi thuộc PC sao cho nó thực thi được. Trong trường hợp này, chúng ta gọi C là độ đo dòng dữ liệu thực thi được, ký hiệu là (C) . Ví dụ, $(All-c-uses)$ được sử dụng để chỉ tất cả các đường đi đầy đủ thỏa mãn độ đo $All-c-uses$ sao cho chúng thực thi được. Cụ thể, $(All-c-uses)^*$ được định nghĩa như sau.

Định nghĩa 4.18. $((All-c-uses) .)$ Với mỗi một biến x và mỗi đỉnh i sao cho i là Global def với biến x , chọn các đường đi đầy đủ thực thi được bao gồm các Def-clear path từ đỉnh i tới tất cả các đỉnh j sao cho j là Global c-use của x .

Tương tự, chúng ta có thể định nghĩa $(All-paths)$, $(All-du-paths)$, $(All-uses)$, $(All-p-uses)$, $(All-p-uses/Some-c-uses)$, $(All-c-uses/Some-p-uses)$, $(All-defs)$, $(All-branches)$, và $(All-statements)$. Mọi quan hệ bao gồm chặt giữa từng cặp này đã được Frankl và Weyuker đã tổng kết trong hình 4.15



Hình 4.15: Mối quan hệ giữa các độ đo dòng dữ liệu thực thi được

Bài tập cuối chương

- Bài 4.1. So sánh kiểm thử hộp trắng và kiểm thử hộp đen?
- Bài 4.2. Tại sao kiểm thử hộp trắng lại tốn chi phí cao hơn kiểm thử hộp đen?
- Bài 4.3 Cách tính độ phức tạp của đồ thị dòng điều khiển?
- Bài 4.4. Quy trình kiểm thử dựa trên độ đo?
- Bài 4.5. Phân biệt các độ đo kiểm thử dòng điều khiển?
- Bài 4.6. Trình bày các vấn đề phổ biến về dòng dữ liệu?
- Bài 4.7. Trình bày các khái niệm cơ bản về dòng dữ liệu?
- Bài 4.8. Trình bày các độ đo cho kiểm thử dòng dữ liệu?
- Bài 4.9. So sánh kiểm thử dòng điều khiển và kiểm thử dòng dữ liệu?
- Bài 4.10. Thiết kế testcase kiểm thử cho đoạn chương trình dưới đây sử dụng các phương pháp dòng điều khiển và dòng dữ liệu?

```

int Binsearch(int X, int V[], int n){
    int low, high, mid;
    low = 0;
    high = n - 1;
    while (low <= high) {
        mid = (low + high)/2;
        if (X < V[mid])
            high = mid - 1;
        else
            if (X > V[mid])
                low = mid + 1;
            else
                return mid;
    }//end while
    return -1;
}//the end
  
```


CHƯƠNG 5: QUẢN LÝ LỖI PHẦN MỀM VÀ BÁO CÁO KIỂM THỬ

Nội dung chính của chương

Chương 5 trình bày về các nội dung chính sau:

- Các thành phần của lỗi
- Tổng quan về TestReport
- Quy trình TestReport
- Cấu trúc của TestReport

Mục tiêu cần đạt được của chương

- Biết các kiến thức cơ bản về ứng dụng Windows Form
- Áp dụng vào xây dựng ứng dụng Windows Form.

Bài 9: Quản lý lỗi phần mềm và báo cáo kiểm thử (Số tiết: 03 tiết)

5.1 Các thành phần của lỗi

5.1.1 Giới thiệu

Lỗi phần mềm là một lỗi hay hỏng hóc trong chương trình hoặc hệ thống máy tính khiến nó tạo ra kết quả không chính xác hoặc không mong muốn hoặc hành xử theo những cách không lường trước được. Quá trình tìm và sửa lỗi được gọi là "gỡ lỗi" và thường sử dụng các kỹ thuật hoặc công cụ chính thức để xác định lỗi và từ những năm 1950, một số hệ thống máy tính đã được thiết kế để ngăn chặn, phát hiện hoặc tự động sửa các lỗi máy tính khác nhau trong quá trình hoạt động.

Hầu hết các lỗi phát sinh từ các lỗi và sai lầm được tạo ra trong mã nguồn của chương trình hoặc thiết kế của chương trình hoặc trong các thành phần và hệ điều hành được sử dụng bởi các chương trình đó. Một số ít các lỗi được gây ra bởi trình biên dịch sản xuất mã không chính xác. Một chương trình có chứa nhiều lỗi (bug) hoặc lỗi ảnh hưởng nghiêm trọng đến chức năng của nó, (buggy). Lỗi có thể kích hoạt các lỗi khác tạo ra hiệu ứng gợn. Lỗi có thể có hiệu ứng hoặc khiến chương trình bị sập hoặc treo máy tính. Các lỗi khác như chỉnh sửa điều kiện truy cập là lỗi bảo mật và có thể giúp cho phép một số người dùng qua được các hàng rào bảo mật để truy cập một số trang web trái phép hoặc mua bán qua các nền tảng công nghệ bị cấm bởi chính phủ.

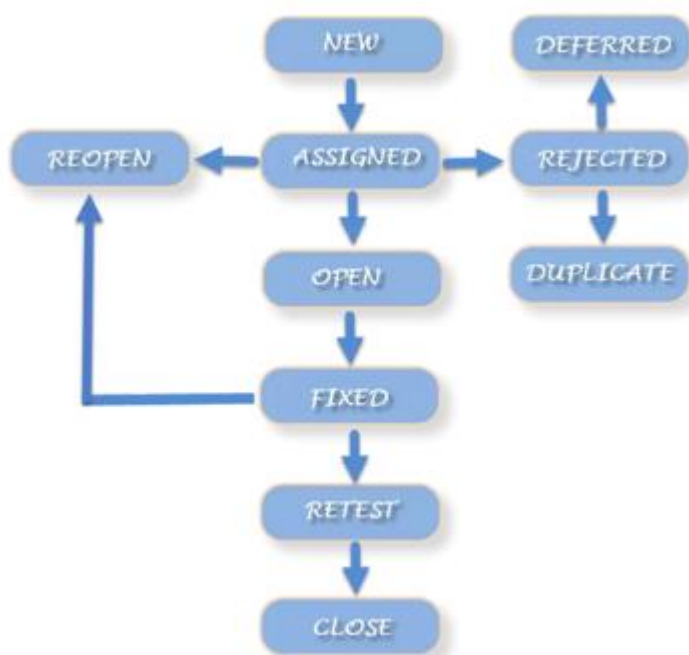
Một số lỗi phần mềm có thể nghiêm trọng tới mức thảm họa. Lỗi trong mã điều khiển máy xạ trị Therac-25 đã trực tiếp khiến bệnh nhân tử vong trong những năm 1980. Năm 1996, tên lửa Ariane 5 của Cơ quan Vũ trụ Châu Âu trị giá 1 tỷ USD đã bị phá hủy chưa đầy một phút sau khi phóng do lỗi trong chương trình máy tính hoa tiêu cài đặt trên tàu. Vào tháng 6 năm 1994, một máy bay trực thăng Chinook của Không quân

Hoàng gia đã đâm vào Mull of Kintyre, giết chết 29 người. Điều này ban đầu được coi là lỗi phi công, nhưng một cuộc điều tra của Computer Weekly đã thuyết phục một cuộc điều tra của House of Lords rằng nó có thể là do lỗi phần mềm trong máy tính điều khiển động cơ của máy bay.

5.1.2 Vòng đời của lỗi

Vòng đời của lỗi là một hành trình mà lỗi đi qua trong suốt cuộc đời của nó. Chúng thay đổi từ tổ chức này sang tổ chức khác, từ dự án này đến dự án khác và được điều chỉnh bởi quy trình kiểm thử phần mềm.

Vòng đời của lỗi bao gồm các trạng thái được thể hiện trong hình 5.1



Hình 5.1 Vòng đời của lỗi phần mềm

New: Khi mà lần lỗi được log lên lần đầu tiên bởi người kiểm thử

Assigned: Khi lỗi đã được đăng lên và chỉ định cho lập trình viên nào đó

Open: Khi lập trình viên đang sửa lỗi

Fixed: Khi lập trình viên đã hoàn thành việc sửa lỗi

Retest: Người kiểm thử kiểm tra lại lỗi đã được sửa hay chưa, có phát sinh thêm lỗi mới nào không

Reopened: Nếu lỗi vẫn còn, người kiểm thử sẽ trả lại cho lập trình viên. Lỗi phải đi lại một vòng đời như cũ.

Deferred: Lỗi sẽ được sửa trong bản phát hành tiếp theo. Lý do có thể là độ ưu tiên

của lỗi có thể là thấp, thiếu thời gian để phát hành hoặc lỗi có thể không có ảnh hưởng lớn đến phần mềm.

Rejected: Nếu lập trình viên cho rằng không phải là lỗi, họ có thể chuyển sang trạng thái này

Duplicate: Lỗi được đăng trùng với nhau

Closed: Khi người kiểm thử đã thấy lỗi được sửa triệt để

Not a bug/Enhancement: Một số thay đổi trong ứng dụng, không phải là lỗi

5.1.3 Phương pháp viết báo cáo lỗi tốt

Một trong những kỹ năng quan trọng nhất mà chúng ta cần có trong hộp công cụ của người kiểm thử nghiệm là khả năng viết một báo cáo lỗi tốt. Việc tìm kiếm lỗi chỉ là một phần của công việc, bởi vì nếu các nhà phát triển không thể phát hiện ra lỗi chúng ta tìm thấy, họ sẽ gặp khó khăn để khắc phục chúng. Phần mềm theo dõi lỗi của chúng ta nên bao gồm một số trường bắt buộc để đảm bảo rằng người kiểm thử đưa ra một bản báo cáo đầy đủ về lỗi mà họ gặp phải. Và người kiểm thử nên trau dồi kỹ năng mô tả của họ.

Báo cáo lỗi tốt bao gồm:

Tiêu đề: Mọi thứ bắt đầu với một tiêu đề. Tiêu đề phải rõ ràng để người đọc có một cái nhìn tổng quát ngay trong nháy mắt

Nội dung lỗi: Mô tả phải rõ ràng và súc tích. Nên nhớ rằng người đọc báo cáo của chúng ta đã không thấy lỗi trước đó.

Các bước tái hiện lỗi: Cần mô tả cụ thể các bước để tái hiện lỗi. Có thể dùng các công cụ để chụp ảnh, quay phim để làm cho các bước này rõ ràng hơn.

Khi lỗi không xảy ra với tỉ lệ 100% theo các bước chúng ta mô tả, thì chúng ta phải cung cấp thông tin đó cho lập trình viên được biết.

Kết quả hiện tại: Chúng ta phải làm rõ kết quả hiện tại như thế nào, khác với kì vọng ra sao. Trường này giúp ngăn ngừa bất kỳ sự hiểu nhầm nào và cho lập trình viên biết chuyện gì đã xảy ra.

Kết quả mong muốn: Chúng ta cần nêu ra những yêu cầu cụ thể theo như đặc tả ban đầu khi thực hiện các bước chúng ta đã nêu ra.

Phiên bản: Chúng ta cũng cần phải có được phiên bản phần mềm đúng. Đôi khi một lỗi sẽ được khắc phục khi một lỗi khác được giải quyết, hoặc chỉ đơn giản bởi một số thay

đổi trong phiên bản mới nhất của phần mềm. Nếu sai phiên bản, lập trình viên có thể mất rất nhiều thời gian để tìm ra lỗi đó.

Thông tin chi tiết: Chúng ta đang sử dụng thiết bị nào? Hệ điều hành nào đang chạy? Chúng ta đã sử dụng trình duyệt nào? Mọi chi tiết chúng ta có thể đưa ra về nền tảng sẽ giúp ích cho các lập trình viên nhanh chóng tái hiện lỗi.

Mức độ ưu tiên và mức độ nghiêm trọng: Đề cập đến Mức độ ưu tiên và Mức độ nghiêm trọng của lỗi giúp quản lý dự án và lập trình viên biết nên ưu tiên sửa lỗi nào trước.

Tài liệu minh họa: Những tài liệu đính kèm, thường là ảnh chụp màn hình hoặc video, thường được các lập trình viên xem đầu tiên, trước khi đọc các bước mô tả của chúng ta. Vì vậy ảnh hoặc video minh họa đính kèm sẽ giúp các lập trình viên tiết kiệm nhiều thời gian quý báu!

Tags & links: Chúng ta nên sử dụng các thẻ mô tả cho phép chúng ta lọc cơ sở dữ liệu và tìm các nhóm lỗi liên quan. Đôi khi chúng ta muốn đưa một ID lỗi hoặc một liên kết trong báo cáo lỗi của chúng ta lên một cái gì đó mà chúng ta cảm thấy có liên quan chặt chẽ hoặc tương tự nhưng không tương tự như một bản sao.

Assignee: Tùy vào quy trình của dự án chúng ta đang làm, quy định sẽ chỉ định lỗi cho ai.

5.2 Tổng quan về TestReport

5.2.1 Khái niệm thế nào là Test Report?

Test Report là bản báo cáo quan trọng nó bao gồm toàn bộ dữ liệu liên quan tới mục tiêu kiểm thử, hoạt động kiểm thử và kết quả của toàn bộ quá trình kiểm thử đó. Không những thế, Test Report còn là tiền đề giúp cho các bộ phận thực thi như developers, tester... dễ dàng đánh giá lại dự án phần mềm đó và liệu rằng sản phẩm này đã đủ điều kiện để đi vào vận hành hay chưa.

Bên cạnh mục đích giúp đánh giá chất lượng thì test report còn giúp cho lập trình viên có thể hiểu hơn về quy trình thực hiện bài test. Bài test đó đã thực sự đi theo kế hoạch ban đầu? Chức năng đã thực sự ổn định? Có vấn đề lỗi nào xảy ra hay không?

Chính vì thế, ngoài việc giúp kiểm tra chất lượng phần mềm thì test report còn giúp tăng tốc quá trình hoàn thiện sản phẩm với hiệu quả cao nhất.

5.2.2 Tầm quan trọng của bản Test Report?

Rất khó có thể hình dung được tầm quan trọng của test report nếu bạn chỉ đọc về lý thuyết, do đó hãy quan sát ví dụ thực tế với kịch bản sau đây nhé:

Nhóm bạn được giao hoàn thiện dự án phần mềm bất kỳ. Trước thời gian cần giao lại cho khách sếp cho hỏi nhóm bạn về phần mềm đó đã đủ điều kiện vận hành chưa? Sau phần trả lời thuyết phục của nhóm, sếp đã quyết định phát hành sản phẩm và giao lại cho khách hàng. Nhưng chỉ thời gian ngắn sau khách liên tục phản hồi về lỗi hiệu suất hoặc lỗi liên quan tới vấn đề vận hành.

Sau tình huống này, bạn đã thực sự hiểu nguyên nhân chính của sự việc này chưa? Vì sao sản phẩm phần mềm đó vẫn bị lỗi ngay cả khi bạn đã rất tự tin về nó?

Vấn đề có thể là do bạn đã bỏ qua phân tích báo cáo và đánh giá lại sản phẩm trong có trong phần test report. Sếp chỉ tin vào những gì bạn nói chứ chưa thực sự thông qua tài liệu để kiểm chứng để đánh giá lại chất lượng sản phẩm đồng nghĩa với việc sản phẩm được phát hành nhưng không cần biết về hiệu suất hoạt động thực tế của nó.

Vì vậy, khi thực hiện test report sẽ đem tới cho bạn 1 vài lợi ích cơ bản sau:

- Nhanh chóng đánh giá được chất lượng sản phẩm ở thời điểm hiện tại cũng như tiến độ thực hiện nhanh hay chậm.
- Hỗ trợ các bộ phận thực hiện có liên quan và đưa ra đánh giá kịp thời
- Có vai trò quan trọng giúp xác định báo cáo đó đã đủ điều kiện để bàn giao cho khách hay chưa?

Ngoài ra, nó còn đem tới cho dự án test automation những lợi ích nhất định như:

- **Đối với leader:** Là cơ sở giúp đánh giá lại sản phẩm trước khi đưa ra vận hành
- **Với tester:** Đưa ra cái nhìn cụ thể hơn về test case. Số lượng test case đã được automation trong thực tế. Số test case cần phải manual. Và thông qua test report sẽ giúp tester nhanh chóng phản hồi lại được các yêu cầu trên.
- **Đối với developer:** Giúp nhanh chóng thêm các tính năng mới cũng như tái cấu trúc mã code mà không làm ảnh hưởng tới tính năng cũ
- **Với các thành viên khác trong dự án:** nắm được tình hình phát triển của hệ thống, bao gồm các vấn đề của network, môi trường và test case.

Chính vì thế, test report là bản báo cáo quan trọng không thể thiếu trong quá trình phát triển dự án phần mềm đặc biệt là những dự án test automation.

5.2.3 Nội dung cần có trong 1 bản Test Report hoàn chỉnh

Mỗi dự án sẽ có yêu cầu khác nhau vì thế nội dung trong đó cũng sẽ khác nhau. Tuy nhiên, để bản test report đầy đủ và chi tiết nhất thì bạn nên đảm bảo các nội dung cơ bản cần có dưới đây:

Thông tin về dự án phần mềm

Thông tin dự án là cơ sở quan trọng cho việc phát triển phần mềm sau này. Do đó với mỗi dự án thì đều phải có tên dự án/ sản phẩm, dữ liệu mô tả về sản phẩm, dự kiến thời gian bắt đầu và thời gian kết thúc, loại dự án.....

Mục tiêu kiểm thử

Mỗi phần test report bắt buộc cần có nội dung cho từng giai đoạn kiểm thử ví dụ performance test, unit test, system test. Bên cạnh đó trong phần này còn bao gồm mục tiêu test mà dự án hướng tới.

Tóm tắt phần kiểm thử – Test Summary

Trong phần test summary sẽ bao gồm toàn bộ thông tin vận hành trong dự án, cụ thể như:

- Số lượng test case đã được thực thi
- Test case Pass hoặc Fail chiếm số lượng ra sao?
- Tỷ lệ phần trăm đạt được theo kế hoạch.
- Tỷ lệ phần trăm thất bại sau khi kiểm thử.
- Thu thập phản hồi, đánh giá từ các bộ phận có liên quan.

Kết luận về lỗi – Defects

Đây là nội dung quan trọng nhất trong bản báo cáo test report, với nội dung chính nói về bug và phần đề xuất khắc phục lỗi. Trong đó :

- Tổng số lượng lỗi trong dự án đó
- Tình trạng hiện tại của lỗi
- Số lượng đã giải quyết thành công

- Đánh giá mức độ nghiêm trọng của lỗi và đề xuất những ưu tiên cần khắc phục trước.

Có thể bạn quan tâm: [Phân biệt Confirmation Testing và Regression Testing](#)

Mẹo để viết Test Report chi tiết và dễ đọc nhất

Test Report có thể được xem là sợi dây kết nối giữa manager với các bộ phận thực hiện trong dự án. Dựa vào phần báo cáo này, các bộ phận có thể nhanh chóng đánh giá được tình hình cụ thể của dự án, về tiến độ cũng như chất lượng hiện tại.

Nếu bản Test report của bạn chưa đủ rõ ràng thì rất khó để các bộ phận khác như devs, phân tích..... có thể hiểu và khắc phục được vấn đề đang gặp phải. Do đó, để có bản test report hoàn chỉnh hãy lưu ý 1 số vấn đề sau:

Thông tin trình bày chi tiết, ngắn gọn

Bản test report chứa các thông tin càng chi tiết sẽ giúp quá trình khắc phục nhanh chóng hơn, bạn không chỉ điền các con số vào báo cáo mà nên mô tả lại quá trình test như các loại test được sử dụng trong dự án, test failed và nguyên nhân failed.

Thêm vào đó, hoạt động thử nghiệm cũng là dữ liệu rất cần thiết giúp cho bản test report trở nên dễ hiểu hơn.

Rõ ràng

Mọi thông tin phải được viết rõ ràng và ngắn gọn nhất, tránh trình bày dài dòng không đúng trọng tâm và không nêu được vấn đề cần giải quyết.

Tiêu chuẩn

Một báo cáo đầy đủ thì tốt nhất bạn nên viết theo tiêu chuẩn nhất định, mặc dù trong dự án có nhiều người cùng hợp tác, mỗi người sẽ có cách thức làm việc riêng nhưng nếu viết báo cáo theo tiêu chuẩn nhất định sẽ giúp quá trình theo dõi trở nên thuận tiện hơn.

Bên cạnh đó, việc viết báo cáo theo nhiều mẫu khác nhau sẽ làm mất nhiều thời gian cho việc kiểm soát lỗi cũng như mất cân bằng giữa các bên.

5.2.4 Những khó khăn trong quá trình phân tích Test Report

Thời gian tổng hợp kết quả

Với các bản báo cáo truyền thống thông thường sẽ được biên tập, phát hành theo mẫu dưới dạng bảng tính. Giống với mô hình thác nước, mỗi dự án sẽ trải qua nhiều giai đoạn

tuy nhiên luôn hạn chế về mặt thời gian. Do đó, quá trình tổng hợp thông tin, tạo báo cáo test report cũng sẽ gặp nhiều khó khăn.

Quá trình phân tích test report phải được thực hiện nhanh chóng nhưng vẫn đảm bảo được chất lượng kèm theo, do đó thời gian sẽ không tính theo tháng mà nó sẽ tính theo tuần, vài ngày thậm chí là trong vài tiếng đồng hồ. Nếu bạn phân tích test report diễn ra chậm theo kế hoạch thì chắc chắn dự án sẽ bị trì trệ và chậm tiến độ so với kế hoạch.

Khối lượng dữ liệu lớn

Dữ liệu cần kiểm thử sẽ được tạo ra hàng giờ, các dữ liệu thường được tạo ra trong quá trình test automation. Nhưng khi dữ liệu cần phân tích quá lớn sẽ gây khó khăn trong việc xác định lỗi của test case hoặc vấn đề lỗi liên quan tới môi trường kiểm thử..... Vì thế phần lớn khi viết test report bạn sẽ gặp vấn đề với các dữ liệu không liên quan.

Divide Data

Một trong những khó khăn bạn sẽ gặp trong khi phân tích test report là về phần dữ liệu divide data. Cụ thể:

- Có quá nhiều dữ liệu cần kiểm thử trong dự án phần mềm cụ thể.
- Dữ liệu cần kiểm thử sẽ được chuyển tới từ các bộ phận như developers, API tester, Business Tester...
- Tồn tại dưới nhiều định dạng khác nhau như Appium, Selenium...

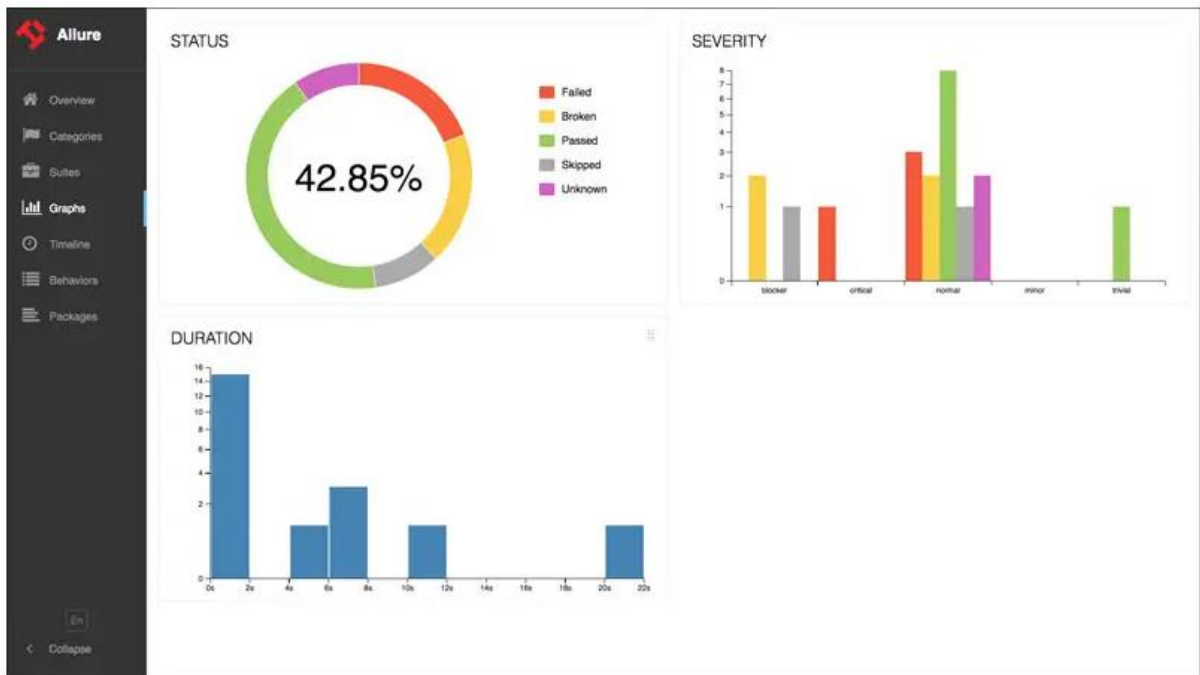
5.2.5 Tiêu chí cơ bản để lựa chọn Report Tool

Hiện nay, hầu hết các dự án phần mềm đều phải thông qua sự hỗ trợ từ built-in report. Nhưng cũng có 1 vài trường hợp built-in report không thể hoàn tất các yêu cầu của dự án như xuất file dữ liệu báo cáo bằng PDF/HTML, gửi test report qua gmail... Vì vậy, quá trình lựa chọn report tool cũng cần đảm bảo các tiêu chí cơ bản sau:

- Luôn hiển thị các chỉ số cơ bản như số lượng test case đã thành công hoặc failed, thời gian thực hiện.....
- Đưa ra bản test report cụ thể trong trường hợp test case failed.
- Tạo dữ liệu test report với nhiều định dạng văn bản khác nhau như HTML hoặc PDF.
- Có thể chạy dữ liệu test report bằng đồ họa.

- Đảm bảo về thời gian thực hiện test case dưới dạng bản đồ chi tiết từ đó mới có thể dễ dàng đánh giá lại tốc độ thực tế giữa các lần chạy vận hành thử.
- Recoding lại phần test case khi chúng đang được thực thi.

Tùy vào yêu cầu và mong muốn của từng dự án để lựa chọn kết hợp giữa 1 hoặc nhiều report tool trong cùng khoảng thời gian điển hình như: JUnit Plugin + Jenkins, Allure Test Report...



Hình 5.2 Lựa chọn công cụ báo cáo

Công nghệ thông tin ngày càng phát triển kèm theo đó là một số thay đổi của test report nhưng chắc chắn phần nguyên lý cơ bản sẽ không bị thay đổi. Chính vì thế, để nhanh chóng tìm ra lỗi và khắc phục sớm nhất thì việc cho ra đời bản test report là điều rất cần thiết.

5.3 Cấu trúc của TestReport

Hình ảnh phía dưới đây miêu tả những thành phần cần có trong một bản Test Report (báo cáo kiểm thử).



Hình 5.3 Cấu trúc TestReport

5.3.1 Thông tin dự án

Thông tin dự án là cơ sở quan trọng cho việc phát triển phần mềm sau này. Do đó với mỗi dự án thì đều phải có tên dự án/ sản phẩm, dữ liệu mô tả về sản phẩm, dự kiến thời gian bắt đầu và thời gian kết thúc, loại dự án...

Project Overview				
PROJECT BASIC INFORMATION				
Project Name	Guru99 Bank			
Name of product (Product Number)	Banking website www.demo.guru99.com			
Product Description	The banking website			
Project Description	<Mission of project> Conduct testing to verify the quality of this website Ensure the website is released without any defects <Project's output product> Test Summary Report & Evaluation			
	Project Type	Testing/Verification		
Project Duration	Start date	10/1/2013	End date	10/31/2013

Hình 5.4 Thông tin dự án

5.3.2 Mục tiêu kiểm thử

Mục tiêu của từng giai đoạn trong quy trình kiểm thử phần mềm (kiểm tra chức năng, kiểm tra hiệu năng, kiểm tra giao diện người dùng, vv) cần phải được mô tả trong Test Report. Mục tiêu là loại kiểm thử mà nhóm và những tester đã thực hiện và tại sao. Ví dụ nếu test report bao gồm kiểm thử chức năng, hồi quy... thì người viết báo cáo sẽ cần mô tả cho từng loại kiểm thử trên.

Trong hầu hết các trường hợp, kiểm thử hồi quy là mục đích chính của việc thực hiện kiểm thử phần mềm. Mục tiêu của kiểm thử hồi quy có thể khác nhau, nhưng một nhóm thường thực hiện phương pháp này để tìm kiếm các lỗi sau khi nhà phát triển thêm code của tính năng vào cơ sở code hiện có.

5.3.3 Tóm tắt kiểm thử

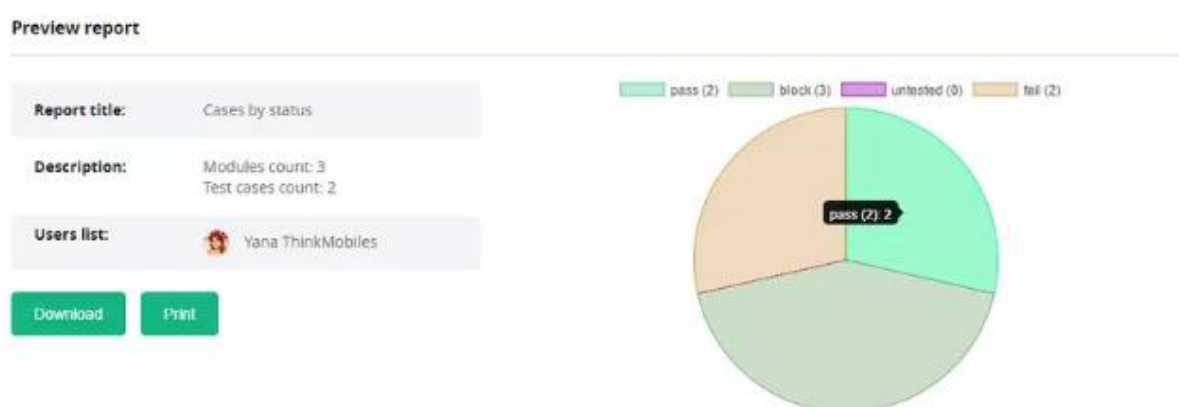
Phần này nên bao gồm những thông tin về sản phẩm vận hành trong bài test như thế nào, test nào pass và test nào failed, hay có test nào không được thực hiện hay không?

Điều tiếp theo bắt buộc phải được chỉ ra như phần dưới đây:

- Số lượng các trường hợp kiểm thử đã thực hiện
- Số lượng các trường hợp kiểm thử thành công
- Số lượng các trường hợp kiểm tra không thành công
- Tỷ lệ phần trăm các trường hợp kiểm thử thành công
- Tỷ lệ phần trăm các trường hợp kiểm thử không thành công
- Các bình luận liên quan

Việc trình bày sẽ tốt hơn nếu chúng ta thể hiện các thông tin một cách trực quan. Dùng các chỉ dẫn màu sắc, các biểu đồ, và các bảng biểu được đánh dấu nổi bật.

Hình ảnh dưới đây là ví dụ về Test Report cho các trường hợp kiểm thử (Test cases)



Hình 5.5 Ví dụ về TestReport

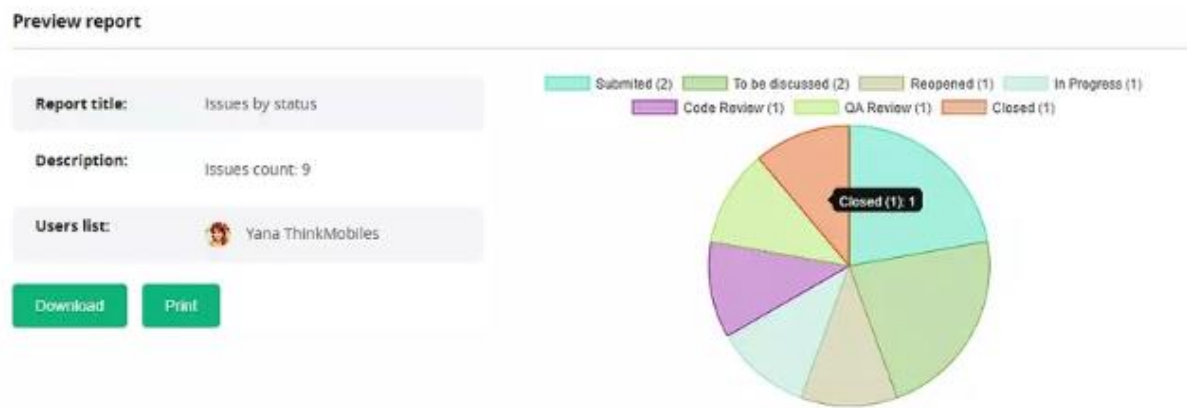
5.3.4 Kết luận về thiếu sót

Đây có thể được coi là phần quan trọng nhất trong Test Report. Nội dung chính của phần này sẽ nói về trạng thái và ưu tiên cần phải làm. Trong đó, có thể nói về số bugs (lỗi) đã được fix, những việc cần làm, v.v... Để cho sinh động hơn, chúng ta có thể cân nhắc sử dụng biểu đồ, bảng được hoạt hoá, v.v.. cho dễ nhìn.

Phần này nên chứa những thông tin sau:

- Tổng số lỗi đã phát hiện
- Trạng thái mỗi lỗi (mở, đóng, đã sửa lỗi, v.v.) - (open, closed, fixed etc.)
- Số lỗi theo từng trạng thái (mở, đóng, đã sửa lỗi, v.v.) - (open, closed, fixed etc.)
- Phân tích mức độ ưu tiên và mức độ xảy ra lỗi - (Severity and priority)

Hình ảnh phía dưới đây minh họa bản Test Report cho các lỗi

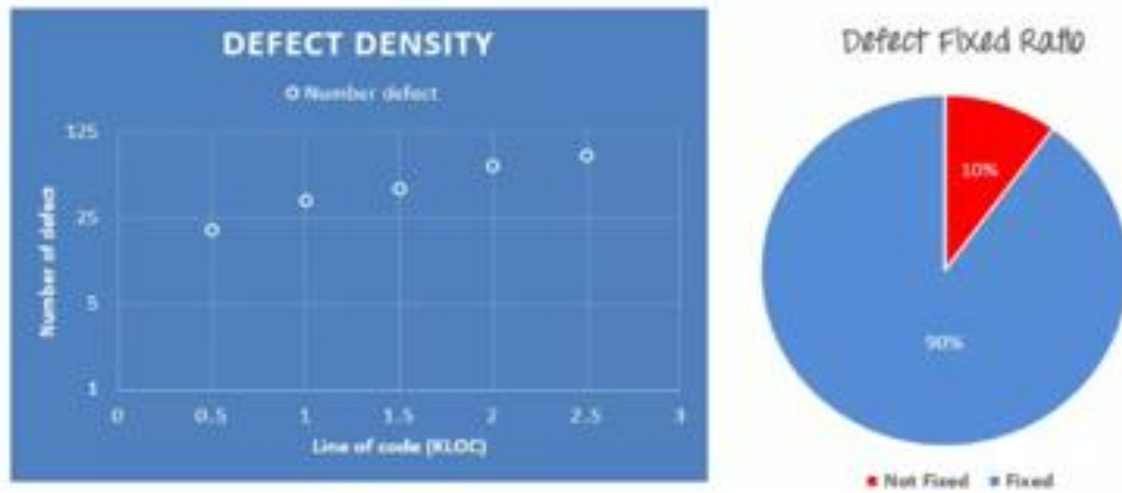


Hình 5.6 Ví dụ minh họa bản Test Report cho các lỗi

Giả sử nhóm dự án đã gửi cho chúng ta thông tin về lỗi như sau:

- Mật độ lỗi là trung bình 20 lỗi/1000 dòng mã code
- Tổng số 90% lỗi đã được sửa
- Chi tiết của các lỗi được mô tả trong trình theo dõi lỗi

Vậy chúng ta có thể biểu diễn dữ liệu về lỗi như biểu đồ sau:



Hình 5.7 Biểu đồ biểu diễn dữ liệu lỗi

Bài tập cuối chương

Bài 5.1 Trình bày các thành phần của lỗi?

Bài 5.2 Trình bày vòng đời của lỗi?

Bài 5.3 Phương pháp viết báo cáo lỗi?

Bài 5.4 Khái niệm Test Report?

Bài 5.5 Tầm quan trọng của Test Report?

Bài 5.6 Nội dung cần có trong một bản Test Report hoàn chỉnh?

Bài 5.7 Những khó khăn trong quá trình phân tích Test Report?

Bài 5.8 Các tiêu chí cơ bản để lựa chọn công cụ Test Report?

Bài 5.9 Trình bày cấu trúc của Test Report?

Bài 5.10 Cho ví dụ về một bản Test Report cụ thể?

CHƯƠNG 6: CÁC THÀNH PHẦN VÀ CÁC CHUẨN ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

Nội dung chính

Nội dung chính của chương 6 bao gồm:

- Độ đo chất lượng phần mềm
- Giá thành của chất lượng phần mềm
- SQA trong các tiêu chuẩn ISO
- SQA trong các tiêu chuẩn IEEE
- Một số hệ thống tiêu chuẩn khác

Mục tiêu cần đạt được của chương

- Phân biệt các độ đo chất lượng phần mềm
- Biết các chuẩn đảm bảo chất lượng phần mềm

Bài 10: Đảm bảo chất lượng phần mềm (Số tiết: 03 tiết)

6.1. Độ đo chất lượng phần mềm

Hai cách khác nhưng được định nghĩa bổ sung (IEEE, 1990) miêu tả độ đo chất lượng phần mềm như một phạm trù của công cụ SQA.

(1) Một sự đo lường định lượng mức độ để một khoản mục có một thuộc tính chất lượng đã quy định.

(2) Đầu vào của một chức năng là dữ liệu phần mềm và đầu ra của nó là một giá trị số duy nhất có thể được giải thích như mức độ để các phần mềm có một thuộc tính chất lượng đã quy định.

Tức là, định nghĩa thứ hai đề cập đến quá trình đưa ra các độ đo chất lượng trong khi định nghĩa đầu tiên đề cập đến kết quả của quá trình nói trên.

Thông thường người ta tin rằng độ đo chất lượng phần mềm nên chứa trong phần mềm, như trong các ngành công nghiệp khác, giữa các công cụ cơ bản được dùng để giúp đỡ sự quản lý trong ba lĩnh vực cơ bản sau: điều khiển phát triển và bảo trì phần mềm, hỗ trợ đưa ra quyết định, khởi tạo các hoạt động hiệu chỉnh. Phân tích thống kê các dữ liệu độ đo được mong đợi để xác định (sinh động và có ý nghĩa thống kê) những thay đổi đã được khởi tạo như một kết quả của ứng dụng của công cụ phát triển mới, thủ tục thay đổi và những sự can thiệp khác.

Phạm vi của độ đo chất lượng phần mềm đã được mở rộng đáng kể qua một vài thập niên trước đây. Chúng ta sẽ xem xét lại một số việc bảo trì và phát triển phần mềm thích

hợp nhất đưa ra ở trọng tâm của chương này. Độ đo như là một công cụ đảm bảo chất lượng phần mềm, không may, không được ứng dụng ở mức độ đầy đủ trong công nghiệp phần mềm, cũng không được cung cấp lợi ích ở mức độ mong đợi. Chỉ một phần nhỏ của tổ chức phát triển phần mềm áp dụng độ đo chất lượng phần mềm một cách có hệ thống.

6.1.1 Mục tiêu đo lường phần mềm

Mục tiêu chính của đo lường chất lượng phần mềm

- Để thuận tiện cho việc điều khiển quản lý cũng như lập kế hoạch và thực thi của sự can thiệp quản lý thích hợp. Đạt được mục đích này dựa trên sự tính toán của độ đo đối với:
 - Độ chênh lệch giữa sự thực thi chức năng (chất lượng) thực tế từ sự thực thi đã lập kế hoạch.
 - Độ chênh lệch của thực hiện ngân sách và thời gian biểu thực tế từ sự thực thi đã lập kế hoạch.
 - Để xác định trạng thái (situation) yêu cầu hoặc cho phép cải tiến quy trình phát triển hay bảo trì dưới dạng các hoạt động ngăn ngừa và sửa đổi được đưa ra trong suốt tổ chức. Đạt mục tiêu này dựa trên:

Tích lũy thông tin về độ đo đối với sự thực hiện của cả nhóm, đơn vị,...

Việc so sánh sẽ cung cấp thực tế cơ bản cho ứng dụng quản lý của độ đo và sự tiên bộ của SQA nói chung. Độ đo được sử dụng cho việc so sánh dữ liệu thực thi với chỉ dẫn (indicator), giá trị định lượng như sau:

- Xác định các chuẩn chất lượng phần mềm.
- Tập hợp mục tiêu chất lượng cho tổ chức và cá nhân.
- Thành tựu chất lượng trong năm trước.
- Thành tựu chất lượng của dự án trước.
- Mức độ chất lượng trung bình đạt được bởi các nhóm khác áp dụng cùng công cụ phát triển trong môi trường phát triển giống nhau.
- Thành tựu chất lượng trung bình của tổ chức.
- Thực tiễn công nghiệp cho sự thỏa mãn yêu cầu chất lượng.

6.1.2 Phân loại độ đo chất lượng phần mềm

Độ đo chất lượng phần mềm có thể chia thành một số nhóm. ở đây chúng tôi sử dụng một hệ thống 2 mức độ:

Thứ nhất, phân loại danh mục phân biệt giữa vòng đời và các pha khác nhau của hệ thống phần mềm:

- Độ đo quy trình, liên quan đến quá trình phát triển phần mềm
- Độ đo Sản phẩm, liên quan đến bảo trì phần mềm.

Thứ hai, phân loại danh mục đề cập đến các chủ đề về đo lường:

- Chất lượng.
- Thời gian biểu.
- Hiệu quả (của lỗi và xóa bỏ các dịch vụ bảo trì).
- Năng suất.

Các mục này được chia với với các nhóm riêng biệt.

Một số lượng lớn độ đo chất lượng phần mềm liên quan đến một trong hai cách đo lường cho kích thước hệ thống như sau:

KLOC - thước đo cổ điển này đo kích cỡ của phần mềm bởi hàng ngàn dòng lệnh. Vì số lượng các dòng mã cần thiết cho lập trình một tác vụ đã quy định về cơ bản là khác nhau đáng kể với mỗi công cụ lập trình, phép đo này là cụ thể cho ngôn ngữ lập trình và công cụ phát triển được sử dụng. Việc áp dụng độ đo đó bao gồm KLOC được giới hạn trong các hệ thống phần mềm phát triển bằng cách sử dụng cùng một ngôn ngữ lập trình hay công cụ phát triển.

Điểm chức năng – một phép đo các tài nguyên phát triển (nguồn lực con người) được yêu cầu để phát triển một chương trình, dựa trên chức năng đã xác định cho hệ thống phần mềm.

Các độ đo thỏa mãn khách hàng sẽ không được trình bày ở đây, các đọc giả có thể tìm thấy phạm vi mở rộng của chủ đề này trong bài giảng marketing.

6.2. Giá thành của chất lượng phần mềm

Các mục tiêu tính giá thành các độ đo chất lượng phần mềm

Mục tiêu của các phép đo chi phí của chất lượng phần mềm

Việc áp dụng các phép đo chi phí cho đảm bảo chất lượng phần mềm cho phép có được sự quản lý tài chính đối với các hoạt động SQA và các kết quả. Các mục tiêu cụ thể là:

- Quản lý tổ chức – xây dựng chi phí để ngăn ngừa và dò tìm lỗi phần mềm
- Ước tính thiệt hại kinh tế của thất bại phần mềm để làm cơ sở xét duyệt ngân sách cho SQA

- Ước lượng các kế hoạch để tăng hoặc giảm các hoạt động SQA hoặc để đầu tư mới hoặc cập nhật cơ sở hạ tầng SQA dựa trên cơ sở hiệu năng kinh tế trong quá khứ

Mô hình truyền thống tính giá chất lượng phần mềm

Mô hình chi phí chất lượng phần mềm được phát triển vào đầu những năm 1950 bởi Feigenbaum và một số người khác (Xem trong Feigenbaum, 1991), cung cấp một phương pháp luận cho việc phân loại chi phí kết hợp với đảm bảo chất lượng sản phẩm từ quan điểm về kinh tế. Được phát triển để phù hợp hoàn cảnh chất lượng dựa trên các tổ chức sản xuất, mô hình đã được thực hiện một cách rộng rãi.

Mô hình phân loại chi phí liên quan đến chất lượng sản phẩm vào hai lớp chung:

- Chi phí quản lý: bao gồm chi phí phải trả cho ngăn ngừa và dò tìm lỗi phần mềm để giảm thiểu chúng đến một mức độ được chấp nhận.
- Chi phí thất bại của quản lý: bao gồm chi phí của những thất bại xảy ra bởi vì thất bại trong việc ngăn ngừa và dò tìm lỗi phần mềm.

Mô hình chia nhỏ hơn nữa thành những lớp con. Chi phí quản lý được chia cho lớp con ngăn ngừa hoặc lớp con chi phí đánh giá:

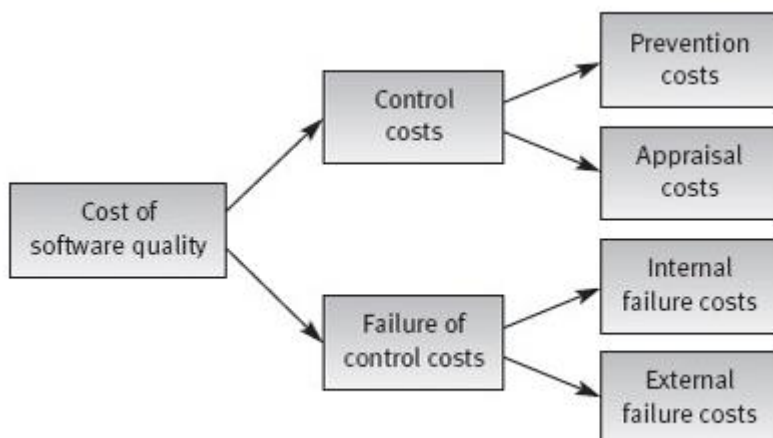
- Chi phí ngăn ngừa bao gồm các khoản đầu tư vào cơ sở hạ tầng chất lượng và hoạt động chất lượng không được nhằm vào một dự án hoặc hệ thống riêng biệt, là chung cho cả tổ chức.
- Chi phí đánh giá bao gồm chi phí các hoạt động được thực hiện cho những dự án cụ thể hoặc hệ thống phần mềm cho mục đích dò tìm lỗi phần mềm.

Chi phí thất bại của quản lý được chia nhỏ hơn thành chi phí thất bại bên trong và chi phí thất bại bên ngoài:

- Chi phí thất bại bên trong bao gồm chi phí sửa chữa những lỗi được tìm thấy khi xem xét lại thiết kế, kiểm tra phần mềm và kiểm tra chấp nhận (được thực hiện bởi khách hàng) và được hoàn thành trước khi phần mềm được cài đặt ở phía khách hàng.
- Chi phí thất bại bên ngoài bao gồm tất cả các chi phí sửa chữa những lỗi được tìm thấy bởi khách hàng hoặc đội bảo trì sau khi phần mềm đã được cài đặt.

Mô hình cổ điển của chi phí chất lượng phần mềm được biểu diễn ở hình 6.1. Mặc dù sự cố gắng áp dụng mô hình cổ điển vào việc phát triển và bảo trì phần mềm đã được ghi nhận là thành công nhưng vẫn không hoàn chỉnh. Nguyên nhân cho những khó khăn

phải đối mặt sẽ được thảo luận phần sau của chương này. Nhưng trước khi tiếp tục, ta xem lại mô hình.



Hình 6.1 Mô hình tính chi phí truyền thống

Mô hình mở rộng tính giá chất lượng phần mềm

Các phân tích chi phí cho đảm bảo chất lượng phần mềm theo mô hình cổ điển cho thấy rằng có một số chi phí rất lớn đã bị bỏ qua. Những chi phí này hoặc là chỉ có trong công nghiệp phần mềm, hoặc không đáng kể trong các ngành công nghiệp khác. Ta xét ví dụ về hai chi phí tiêu biểu phải mất do các lỗi trong đảm bảo chất lượng phần mềm như sau:

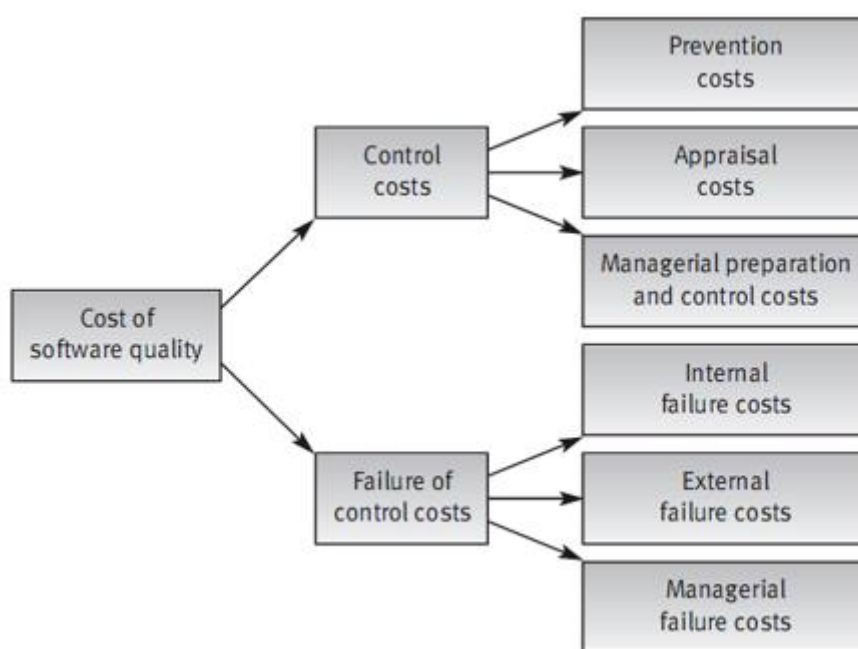
- Chi phí đền bù do hoàn thành dự án muộn, nguyên nhân xuất phát từ việc lập lịch biểu một cách không thực tế.
- Chi phí đền bù vì hoàn thành dự án muộn do thất bại trong việc tuyển nhân viên mới có đủ khả năng.

Hai thất bại trên đây đều không phải do bất cứ một hoạt động cụ thể nào của nhóm phát triển hay do thiếu chuyên môn nghiệp vụ, mà là do việc quản lý gây ra. Do đó, những người quản lý dự án có thể thực hiện một vài hoạt động để ngăn ngừa hoặc chí ít là giảm thiểu chi phí mà các loại thất bại trên gây ra:

- Xem xét lại hợp đồng (xem xét lại các đề xuất sơ bộ và hợp đồng sơ bộ). Chi phí của các hoạt động này thường không đáng kể đối với các hợp đồng trong công nghiệp sản xuất. Tuy nhiên trong công nghiệp phần mềm, người ta cần một lượng công việc chuyên môn đáng kể để đảm bảo rằng các đề xuất của dự án được dựa trên các đánh giá cẩn thận và các ước lượng dễ hiểu (rõ ràng). Sự khác biệt về yêu cầu tài nguyên cho cùng công việc xem xét lại hợp đồng giữa công nghiệp phần mềm và công nghiệp sản xuất xuất phát từ bản chất của sản phẩm và tiến

trình sản xuất được đề cập đến trong hợp đồng. Trong khi hợp đồng trong công nghiệp sản xuất xử lý việc tạo ra nhiều lần các sản phẩm trong danh sách cố định, thì hợp đồng trong công nghiệp thường chỉ xử lý việc phát triển một hệ thống phần mềm mới, duy nhất.

- Điều khiển tiến độ dự án phần mềm một cách kỹ lưỡng và thích hợp. Trong khi việc điều khiển quá trình sản xuất trong công nghiệp sản xuất là một công việc lặp đi lặp lại và thường được thực hiện tự động bằng máy móc, thì việc điều khiển tiến độ phát triển phần mềm quản lý thiết kế tác vụ và các hoạt động coding được thực hiện bởi nhóm phát triển dự án.



Hình 6.2 Mô hình tính chi phí mở rộng

Các vấn đề trong áp dụng tính giá các độ đo chất lượng phần mềm

Ứng dụng của mô hình chi phí chất lượng phần mềm thường đi cùng với các vấn đề cần phải giải quyết ở bất cứ ngành công nghiệp nào. Những vấn đề này tác động lên độ chính xác và đầy đủ của dữ liệu chi phí chất lượng do:

- Sự thiếu chính xác và thiếu đầy đủ của việc xác định và phân loại các chi phí chất lượng.
- Các báo cáo cầu thả của các thành viên nhóm và người ngoài nhóm.
- Việc báo cáo lệch lạc các chi phí phần mềm.
- Các bản ghi lệch lạc về các chi phí do thất bại bên ngoài, xuất phát gián tiếp từ các khoản đền bù được ngay trang cho khách hàng (ví dụ: hạ giá các dịch vụ

tương lại, cung cấp các dịch vụ miễn phí...). Chi phí cho các thất bại cần đền bù đó không được ghi nhận như là chi phí do thất bại bên ngoài.

Những vấn đề trên đây nảy sinh trong ngữ cảnh công nghiệp phần mềm nhưng cũng có những vấn đề khác nữa. Một số trong chúng chỉ có trong sản xuất phần mềm mà thôi. Chúng ta sẽ tập trung vào các vấn đề gặp phải khi ghi nhận các chi phí chuẩn bị và điều khiển công việc quản lý cùng các chi phí do thất bại trong quản lý bởi vì những khoản mục này ảnh hưởng tới chi phí và tính dễ hiểu của tổng chi phí cho đảm bảo chất lượng phần mềm, đặc biệt là khi áp dụng mô hình chi phí mở rộng.

Các vấn đề nảy sinh khi thu thập dữ liệu trong chuẩn bị và điều khiển việc quản lý chi phí bao gồm:

Xác định trách nhiệm khi có thất bại trong thực hiện lịch biểu. Những chi phí này có thể gán cho khách hàng (trong trường hợp khách hàng được yêu cầu bồi thường cho nhà thầu), nhóm phát triển (được xem như chi phí do thất bại bên ngoài) hay ban quản lý (được xem như chi phí cho thất bại trong quản lý). Chi phí cho thất bại lịch biểu thường được cân nhắc cho một giai đoạn đủ dài vì các nguyên nhân trực tiếp hay các đóng góp của từng thành phần trong dự án dẫn tới thất bại khó được xác định.

6.3. SQA trong các tiêu chuẩn ISO

6.3.1 ISO 9001 và ISO 9000-3

ISO 9000-3, được đưa ra bởi tổ chức ISO, trình bày cách cài đặt thực hiện phương pháp luận chung của tiêu chuẩn quản lý chất lượng ISO 9000 trong các trường hợp đặc biệt của phát triển và duy trì phần mềm. Cả ISO 9001 và ISO 9000-3 được xem xét lại và cập nhật liên tục khoảng 5 – 8 năm 1 lần, với những các cuộc thảo luận riêng rẽ.

Giống như ISO 9000-3 dựa trên cơ sở những gì mà tiêu chuẩn ISO 9001 đưa ra, việc công bố các sửa chữa trong các nguyên tắc cũng dựa trên sự công bố các tiêu chuẩn đã được xem xét lại sau khoảng 1 vài năm. Chẳng hạn như, phiên bản 1997 của ISO 9000-3 dựa trên phiên bản 1994 của ISO 1994. Phiên bản 1997 của ISO 9000-3 kết hợp ISO 9001 cùng các đặc trưng của ISO 9000-3 thành tiêu chuẩn “tất cả trong một” cho ngành công nghiệp phần mềm.

8 nguyên lý hướng dẫn tiêu chuẩn ISO 9000 – 3 mới, có nguồn gốc từ tiêu chuẩn ISO9000:2000 như sau:

- Tập trung khách hàng: Tổ chức dựa trên khách hàng của họ và từ đó nên hiểu được các yêu cầu hiện tại và trong tương lai của khách hàng

- Lãnh đạo: Những người lãnh đạo thiết lập tầm nhìn của tổ chức. Họ nên tạo ra và duy trì môi trường bên trong mà mọi người có thể phát triển nhằm đạt được mục tiêu của tổ chức qua các lộ trình định sẵn
- Sự phát triển của con người: Con người là cốt lõi của tổ chức. Tại mọi mức độ của tổ chức, sự phát triển của nhân lực cho phép kỹ năng của họ góp phần làm tăng lợi nhuận của tổ chức
- Tiếp cận tiến trình: Kết quả mong muốn giành được 1 cách hiệu quả hơn khi các hoạt động và tài nguyên được quản lý như 1 tiến trình
- Tiếp cận hệ thống để quản lý: Xác định, hiểu được và quản lý tiến trình, nếu nhìn như 1 hệ thống sẽ góp phần nâng cao hiệu năng tổ chức.
- Tiếp tục cải thiện: Liên tục cải thiện tất cả các hiệu suất nên được đề cao trong các cuộc họp của tổ chức
- Tiếp cận sự thực để ra quyết định: Các quyết định hiệu quả dựa trên việc phân tích các thông tin
- Mỗi quan hệ hỗ trợ lẫn nhau giữa những nhà cung cấp: 1 tổ chức và nhà cung cấp của nó là độc lập với nhau. Tuy nhiên mỗi quan hệ hỗ trợ lẫn nhau có thể nâng cao kỹ năng của cả 2 để tạo thêm ra giá trị gia tăng

Tiêu chuẩn ISO 9000-3 (ISO 1997) gồm 20 yêu cầu liên quan đến rất nhiều khía cạnh của hệ thống quản lý chất lượng phần mềm. Phiên bản mới hơn ISO 2001 đề nghị 1 cấu trúc mới với 22 yêu cầu được phân thành 5 nhóm cơ bản:

- Hệ thống quản lý chất lượng:
 - Những yêu cầu chung
 - Các tài liệu yêu cầu
- Trách nhiệm quản lý:
 - Sự tận tâm trong quản lý
 - Đối tượng khách hàng cần tập trung
 - Điều khoản về chất lượng
 - Lập kế hoạch
 - Trách nhiệm, thẩm quyền và cách truyền đạt
 - Xem xét lại về quản lý
- Quản lý tài nguyên:
 - Tài nguyên cung cấp

- Nhân lực
- Các thiết bị cơ bản
- Môi trường làm việc
- Hiểu rõ sản phẩm:
 - Hiểu rõ kế hoạch của sản phẩm
 - Quá trình quan hệ với khách hàng có liên quan
 - Thiết kế và phát triển
 - Khả năng chi trả
 - Dự liệu về sản phẩm và các dịch vụ
 - Điều khiển thiết bị đo lường và kiểm tra
- Quản lý , phân tích và cải thiện:
 - Kiểm tra và đo lường
 - Điều khiển các sản phẩm không phù hợp
 - Phân tích dữ liệu
 - Cải thiện chất lượng

ISO 9001 – từ ứng dụng đến phần mềm: Sáng kiến TickIT

Được bắt đầu vào cuối thập kỷ 1980 , ngành công nghiệp phần mềm Vương quốc Anh cộng tác với Bộ công thương UK đã nâng phương pháp luận phát triển cho ISO 9001 thành các đặc điểm riêng cho công nghiệp phần mềm được biết đến như sáng kiến TickIT. Tại thời điểm nó bắt đầu, ISO 9001 đã được áp dụng thành công vào công nghiệp chế tạo tuy nhiên không có phương pháp luận nào có ý nghĩa cho các đặc điểm riêng biệt của công nghiệp phần mềm. Vài năm sau đó, TickIT cùng với sự nỗ lực nghiên cứu trong phát triển ISO 9000-3 đã đạt được thành quả này .

6.3.2 Các mô hình tăng trưởng khả năng – phương pháp đánh giá CMM và CMMI

Đánh giá CMM dựa trên các khái niệm và nguyên lý sau:

- Ứng dụng của nhiều phương thức quản lý phức tạp dựa trên cách tiếp cận định lượng nhằm tăng khả năng của tổ chức để điều khiển chất lượng và cải thiện hiệu quả của quá trình phát triển phần mềm.
- Phương tiện để đánh giá việc phát triển phần mềm bao gồm 5 mức của mô hình đánh giá khả năng hoàn thành công việc đúng thời hạn. Mô hình cho phép một tổ chức đánh giá hiệu quả hoạt động của nó và xác định các yêu cầu cần thiết để có

thể đạt tới mức tiếp theo bằng cách xác định các vùng xử lý được yêu cầu cho việc cải thiện.

- Vùng xử lý được dùng chung, chúng định nghĩa “cái gì” chứ không phải” bằng cách nào”. Đây là cách tiếp cận cho phép mô hình có thể được áp dụng cho một cho một miền rộng lớn của việc thực thi tổ chức, bởi vì:
 - Nó cho phép sử dụng một số mô hình vòng đời.
 - Nó cho phép sử dụng một số phương pháp luận, công cụ để phát triển phần mềm và ngôn ngữ lập trình.
 - Nó không chỉ rõ một tài liệu chuẩn cụ thể nào.

Mô hình CMMI cũng giống như mô hình gốc CMM bao gồm 5 mức. Các mức của CMMI cũng giống như các mức trong bản gốc của nó, chỉ có một sự thay đổi nhỏ ở mức 4, được đặt tên như sau:

- Capability maturity level 1: khởi tạo
- Capability maturity level 2: Lặp lại
- Capability maturity level 3: Xác định
- Capability maturity level 4: Quản lí
- Capability maturity level 5: Tối ưu hóa

6.4. SQA trong các hệ tiêu chuẩn IEEE

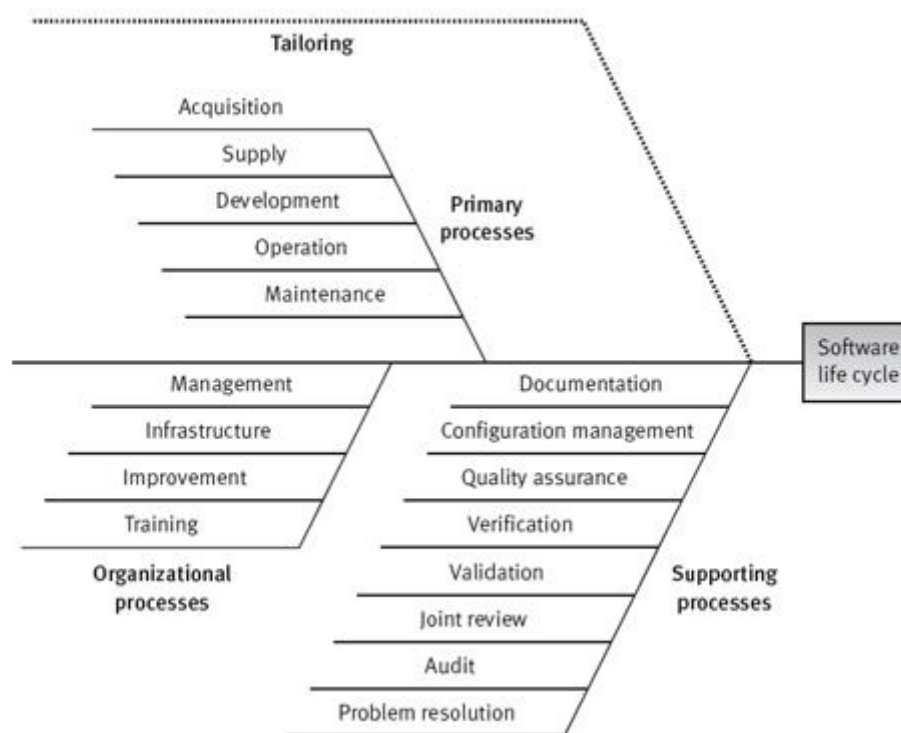
6.4.1 IEEE/EIA Std 12207- các tiến trình vòng đời phần mềm

Chuẩn IEEE/EIA Std 12207 cung cấp framework chung cho phát triển và quản lý phần mềm

- Tổ chức phát triển chuẩn:
 - US Department of Defense (MIL-STD-498:1994)
 - ANSI, IEEE và EIA (Joint Standard 016 (J-Std-016-1995))
 - Tổ chức tiêu chuẩn quốc tế (ISO) và IEC – ISO/IEC 12207 Standard
- Chuẩn gồm 3 phần – three-part standard
 - IEEE/EIA Std 12207.0-1996 (IEEE/EIA, 1996): bao gồm ISO/IEC 12207 gốc và các phần thêm vào
 - IEEE/EIA Std 12207.1-1997 (IEEE/EIA, 1997a): Hướng dẫn – dữ liệu vòng đời
 - IEEE/EIA Std 12207.2-1997 (IEEE/EIA, 1997b): Hướng dẫn – Xem xét cài đặt

Mục tiêu của IEEE/EIA Std 12207 là:

- Thiết lập mô hình các quá trình vòng đời phần mềm chung được chấp nhận trên toàn cầu
- Đẩy mạnh sự am hiểu giữa các nhóm nghiệp vụ bằng việc áp dụng các quá trình, các hoạt động và các nhiệm vụ đã được chấp nhận chung
 - ❖ Kiến trúc vòng đời phần mềm IEEE/EIA 12207 bao gồm:
 - 4 mức
 - Process classes: Các lớp quy trình
 - Processes: Các tiến trình
 - Activities: Các hoạt động
 - Tasks: Các công việc
 - 3 quy trình
 - Primary life cycle processes
 - Supporting life cycle processes
 - Organizational life cycle processes



Hình 6.3 Biểu đồ xương cá- Những tiến trình vòng đời phần mềm theo chuẩn IEEE/EIA 12207

- Các khái niệm chung – general concepts
 - Khả năng áp dụng chuẩn nói chung và sự thích ứng của nó bằng việc điều chỉnh cho một nhu cầu cụ thể

- Khả năng áp dụng cho tất cả các bên tham gia trong vòng đời phần mềm
- Tính mềm dẻo và tính đáp ứng với thay đổi công nghệ
- Phần mềm liên kết với hệ thống
- Tính nhất quán của quản lý chất lượng toàn phần mềm
- Không có các yêu cầu xác minh
- Baselining
- Các khái niệm liên quan tới công việc – task-related concepts
 - Gán trách nhiệm cho mỗi hoạt động và mỗi công việc
 - Modun của các thành phần trong vòng đời phần mềm
 - Các mức phù hợp được yêu cầu
 - Bản chất của công việc ước lượng

6.4.2 IEEE Std 1012 – xác minh và thẩm định

Mục tiêu của chuẩn IEEE Std 1012 là:

- Thiết lập framework chung cho các hoạt động xác minh và thẩm định(V & V)
- Xác định đầu vào – đầu ra của việc V & V
- Xác định các mức toàn vẹn phần mềm, công việc V & V cho mỗi mức
- Xác định nội dung của tài liệu lập kế hoạch V&V Các khái niệm cơ bản của IEEE 1012
- Định nghĩa rộng về các hoạt động V&V: Cho phép chuẩn bao quát tất cả các hoạt động V&V thực hiện xuyên suốt vòng đời phần mềm
- Tuân thủ và tương thích với các chuẩn quốc tế
- Các mức toàn vẹn của phần mềm và các yêu cầu V&V cho từng mức
- Độc lập của các hoạt động V&V
- Tính độc lập về quản lý: nhóm V&V tự quyết định phương thức V&V nào được áp dụng.
- Tính độc lập về công nghệ: các thành viên trong nhóm V&V không liên quan đến việc phát triển phần mềm
- Tính độc lập về tài chính: quyền điều hành về ngân sách của V&V không được trao cho nhóm phát triển

Quy trình xác minh và thẩm định bao gồm:

- Quy trình quản lý
- Quy trình thu thập

- Quy trình cung cấp
- Quy trình phát triển
- Quy trình vận hành
- Quy trình bảo trì

6.4.3 IEEE Std 1028 – rà soát

- Mục tiêu của IEEE Std 1028 là

Định nghĩa các thủ tục của rà soát một cách hệ thống để:

- Thích hợp cho việc xem xét lại được thực hiện trong suốt vòng đời phần mềm
- Phù hợp với những yêu cầu việc xem xét lại được định nghĩa bởi những tiêu chuẩn khác

Có 5 kiểu rà soát trong IEEE Std 1028:

- Xem xét lại quản lý.
- Xem xét lại kỹ thuật
- Kiểm tra.
- Walkthroughs
- Kiểm toán Ba khái niệm cơ bản được đặc trưng
- Hình thức cao - High formality
- Follow-up
- Sự tuân theo tiêu chuẩn quốc tế và IEEE IEEE Std 1028-. 1997 Phần chính của IEEE

Std 1028-1997 đòi hỏi:

- Trình bày định nghĩa chi tiết của những yêu cầu của việc xem xét lại
- Phụ lục trình bày những mối quan hệ của tiêu chuẩn với những quá trình vòng đời

Thành phần của yêu cầu rà soát:

- Giới thiệu:
- Những mục đích của mỗi kiểu xem xét lại.
- Những ví dụ tiêu biểu của mỗi kiểu sản phẩm phần mềm
- Các trách nhiệm: Chuẩn cung cấp một danh sách những người tham gia, có thể là:
- Người tham gia ủy quyền (ví dụ: lãnh đạo, nhân viên kỹ thuật...)
- Hoặc những người tham gia tùy chọn khác (ví dụ: nhà quản lí, khách hàng...)
- Đầu vào dữ liệu được chia thành
- Những mục bắt buộc: tập trung vào mục đích của việc xem xét lại (vd: phát biểu mục tiêu, các tiêu chuẩn...)

- Tù y chọn (vd: nhữ ng điều chỉnh, nhữ ng bất thườ ng)
- Tiêu chuẩn: Là nhữ ng điều kiện ban đầu để cho phép và thực hiện việc xem xét lại, bao gồm:
 - Phát biểu tất cả các mục tiêu của việc xem xét lại
 - Tính sẵn sàng của dữ liệu đầu vào được yêu cầu
- Thủ tục gao gồm:
 - Chuẩn bị quản lý
 - Lập kế hoạch để xem xét lại
 - Sắp xếp thành viên trong nhóm
 - Kiểm tra sản phẩm phần mềm
 - Theo dõi các hoạt động chỉnh sửa
- Các tiêu chuẩn đã tồn tại xác định nhữ ng việc phải được thực hiện trước khi việc xem xét lại kết thúc hoàn toàn, bao gồm:
 - Hoàn thành các hoạt động theo thủ tục
 - Theo dõi và phê chuẩn của các mục hành động đã được hoàn thành hoặc các hoạt động sửa chữa và ngăn chặn
 - Hoàn thành tài liệu của việc xem xét lại.
- Đầu ra: gồm:
 - Các tiêu chuẩn xác định các đầu ra cho mỗi loại xem xét lại
 - Các mục bổ sung có thể được yêu cầu bởi tổ chức, bởi các thủ tục bộ phận khác, hoặc trong các trường hợp cụ thể.
- Các khuyến nghị thu thập dữ liệu:

nhóm kiểm tra và walkthrough thu thập dữ liệu liên quan tới nhữ ng bất thườ ng đã gặp mỗi trường hợp được phân loại và xếp hạng theo mức độ nghiêm trọng.

dữ liệu này được sử dụng để

nghiên cứu tính hiệu quả của quá trình thực hiện hiện tại

chúng cũng được sử dụng để khuyến khích cải tiến các phương pháp và thủ tục.
- Cải tiến: dữ liệu kiểm tra và walkthrough sẽ được phân tích theo:
 - Hoàn thiện các thủ tục
 - Cập nhật danh sách kiểm tra (checklists) được sử dụng bởi nhữ ng người tham gia
 - Cải thiện các quy trình phát triển phần mềm

6.5. Một số hệ tiêu chuẩn khác

Các chuẩn tiến trình dự án SQA tập trung vào phương pháp luận để thực hiện việc phát triển phần mềm và bảo trì phần mềm trong đó:

- Mô tả mỗi bước của một quá trình
- Yêu cầu kèm theo: tài liệu thiết kế, các nội dung thiết kế, xem xét lại thiết kế và các vấn đề xem xét lại, kiểm tra phần mềm và các mục tiêu của nó...
- Các tổ chức phát triển chuẩn: IEEE, EIA, ISO....
- Các chuẩn IEEE có thể được chia thành 3 lớp chính
 - Các chuẩn khái niệm (conceptual standards)
IEEE 610.12, IEEE 1061, IEEE 1320.2, IEEE 1420.1a, IEEE/EIA 12207.0
 - Các chuẩn yêu cầu cho sự tương thích (Prescriptive standards of conformance)
IEEE 828, IEEE 829, IEEE 1012, IEEE 1028, IEEE 1042.1
 - Các chuẩn hướng dẫn (Guidance standards):
IEEE 1233, IEEE/EIA 12207.1, IEEE/EIA 12207.2

Bài tập cuối chương

Bài 6.1 Trình bày các thành phần của lỗi?

Bài 6.2 Trình bày vòng đời của lỗi?

Bài 6.3 Phương pháp viết báo cáo lỗi?

Bài 6.4 Khái niệm Test Report?

Bài 6.5 Tầm quan trọng của Test Report?

Bài 6.6 Nội dung cần có trong một bản Test Report hoàn chỉnh?

Bài 6.7 Những khó khăn trong quá trình phân tích Test Report?

Bài 5.8 Các tiêu chí cơ bản để lựa chọn công cụ Test Report?

Bài 6.9 Trình bày cấu trúc của Test Report?

Bài 6.10 Cho ví dụ về một bản Test Report cụ thể?

TÀI LIỆU THAM KHẢO

- [1] Phạm Ngọc Hùng, Trương Anh Hoàng, Đặng Văn Hưng, 2014, *Giáo trình kiểm thử phần mềm*, ĐHCN-ĐHQGHN.
- [2] Phan Thị Hoài Phương, 2010, *Bài giảng Đảm bảo chất lượng phần mềm*
- [3] Kshirasagar Naik, *Software testing and quality assurance, theory and practice*, Wiley
- [4] Ian Sommerville (2015), *Software Engineering*; 9th Edition, Addison Wesley
- [5] Gerald D. Everett, Raymond McLeod, Jr. (2007), *Software Testing. Testing Across the Entire Software Development Life Cycle*, IEEE press, Wiley-interscience , A John Wiley&Sons, Inc, Publication.

CÁC CÂU HỎI THƯỜNG GẶP

- Câu 1. Các định nghĩa và thuật ngữ cơ bản về kiểm thử phần mềm?
- Câu 2. Phân loại lỗi?
- Câu 3. Phân biệt các mức kiểm thử?
- Câu 4. Trình bày các nguyên tắc kiểm thử?
- Câu 5. So sánh kiểm thử hộp trắng và kiểm thử hộp đen?
- Câu 6. Tại sao kiểm thử hộp trắng lại tốn chi phí cao hơn kiểm thử hộp đen?
- Câu 7. Quy trình kiểm thử dựa trên độ đo?
- Câu 8. Phân biệt các độ đo kiểm thử dòng điều khiển?
- Câu 9. Trình bày các vấn đề phổ biến về dòng dữ liệu?
- Câu 10. Trình bày các khái niệm cơ bản về dòng dữ liệu?
- Câu 11. So sánh kiểm thử dòng điều khiển và kiểm thử dòng dữ liệu?
- Câu 12. Trình bày nguyên tắc chọn dữ liệu thử?
- Câu 13. Trình bày nguyên tắc phân hoạch lớp tương đương?
- Câu 14. Trình bày các thành phần của bảng quyết định?
- Câu 15. Các bước thực hiện kiểm thử giá trị biên?
- Câu 16. Các bước thực hiện kiểm thử lớp tương đương?
- Câu 17. Các bước thực hiện kiểm thử bằng bảng quyết định?
- Câu 18. Các bước thực hiện kiểm thử tổ hợp?
- Câu 19. Nêu quy trình kiểm thử phần mềm?
- Câu 20. Các bước lập kế hoạch kiểm thử?
- Câu 21. Nêu quy trình kiểm thử phần mềm?
- Câu 22. Các bước lập kế hoạch kiểm thử?
- Câu 23. Khái niệm Test Report?
- Câu 24. Nội dung cần có trong một bản Test Report hoàn chỉnh?
- Câu 25. Trình bày cấu trúc của Test Report?

BÀI TẬP THỰC HÀNH

Bài thực hành số 1 (Số tiết: 05 tiết)

1. Mục tiêu kiến thức

- Giúp sinh viên làm quen với việc lập kế hoạch kiểm thử
- Bước đầu thiết kế các testcase chuẩn bị cho kiểm thử chức năng

2. Yêu cầu:

- + Yêu cầu về điều kiện thực hành: máy tính, công cụ kiểm thử tự động
- + Yêu cầu sinh viên: nắm được các bước lập kế hoạch kiểm thử, biết cách thiết kế testcase, biết sử dụng công cụ kiểm thử tự động để thực thi testcase,...

3. Nội dung

3.1. Bài thực hành mẫu

Bài 1.1: Cho một chương trình xác nhận một trường số đếm như sau: Các giá trị nhỏ hơn 10 bị loại bỏ, các giá trị giữa 10 và 21 được chấp nhận, các giá trị lớn hơn hoặc bằng 22 bị loại bỏ.

Yêu cầu: Sử dụng phương pháp test giá trị biên xác định các giá trị đầu vào của các giá trị biên?

Hướng dẫn giải:

Bước 1: Phân chia miền dữ liệu vào thành các lớp tương đương

stt	Dữ liệu đầu vào	Các lớp tương đương hợp lệ	Các lớp tương đương không hợp lệ
1	Số kiểu number	Giá trị thuộc [10,21]	Giá trị <10, 21<Giá trị

Bước 2: Xác định giá trị biên cho mỗi lớp tương đương

-Lớp 1 : 10,21.

-Lớp 2 : <10,>21

Bước 3: Sinh dữ liệu vào cho các lớp tương đương.

-TC1 :10

-TC2 :11

-TC3 :15

-TC4 :20

-TC5 :21

-TC6 :9

-TC7 :22

3.2. Các bài thực hành cơ bản

Bài 1.2: Lập kế hoạch kiểm thử cho nhóm.

Bài 1.3: Số thứ tự trên hệ thống điều khiển cổ phiếu có thể sắp xếp từ 10000 đến 99999 (bao gồm cả 10000 và 99999).

Yêu cầu: Hãy chỉ ra các đầu vào nào có thể là một kết quả của thiết kế kiểm thử chỉ cho các lớp tương đương đúng và các lớp biên đúng?

Bài 1.4: Một hệ thống được thiết kế để đóng thuế như sau: “Một nhân viên có lương £4000 thì được miễn thuế. £1500 tiếp theo bị đánh thuế 10%. £28000 tiếp theo bị đánh thuế 22%. Bất kỳ lượng tiền lớn hơn sau đó đều bị đánh thuế 44%”.

Yêu cầu: Giá trị nào là một trường hợp phân tích giá trị biên đúng cho giá trị tiền lương thấp nhất bị đánh phần trăm thuế lớn nhất?

3.3. Các bài thực hành nâng cao

Bài 1.5:

Thiết kế testcase cho chức năng Đăng nhập sử dụng kỹ thuật kiểm thử lớp tương đương

User

Pass

Input: dữ liệu kiểu text, độ dài ký tự: từ 10-15

Bài 6: Sử dụng kỹ thuật kiểm thử lớp tương đương sinh testcase kiểm thử các chức năng: Đăng kí, Tìm kiếm,...

Bài 7: Hãy áp dụng các kỹ thuật kiểm thử lớp tương đương cho chương trình giải phương trình bậc hai.

Bài thực hành số 2 (số tiết: 05 tiết)

1. Mục tiêu kiến thức

- Giúp sinh viên làm quen với việc lập kế hoạch kiểm thử
- Bước đầu thiết kế các testcase chuẩn bị cho kiểm thử chức năng

2. Yêu cầu:

- + Yêu cầu về điều kiện thực hành: máy tính, công cụ kiểm thử tự động
- + Yêu cầu sinh viên: nắm được các bước lập kế hoạch kiểm thử, biết cách thiết kế testcase, biết sử dụng công cụ kiểm thử tự động để thực thi testcase,...

3. Nội dung

3.1. Bài thực hành mẫu

Bài 2.1: Thiết kế testcase cho chức năng đăng nhập.

- Bước 1:
 - XD điều kiện: Gồm 2 điều kiện
 - Điều kiện 1: Nhập User
 - Điều kiện 2: Nhập Password
 - Giá trị của điều kiện: 3 giá trị: đúng (T), sai (F), để trống (B)
 - Hành động của hệ thống: Đăng nhập thành công (T) hay không (F)
- Bước 2:

Đk	TH1	Th2	TH3	TH4	TH5	TH6	TH7	TH8	TH9
Nhập user	T	T	T	F	F	F	B	B	B
Nhập Password	T	F	B	T	F	B	T	F	B

- Bước 3, Bước 4

	Các trường hợp				
Điều kiện	TH1	TH2	TH3	TH4	TH5
Nhập user	T	T	T	F	B
Nhập pass	T	F	B	-	-
Hành động của hệ thống					
Đăng nhập thành công hay không	T	F	F	F	F

- Bước 5:

Id	Tiêu đề	Mô tả kịch bản	Kết quả mong đợi		
TC1	Đăng nhập thành công	1. Mở chức năng đăng nhập 2. Nhập user đúng 3. Nhập password đúng 4. ấn phím đăng nhập	Đăng nhập thành công		
TC2	Đăng nhập không thành công	1. Mở chức năng đăng nhập 2. Nhập user đúng 3. Nhập password sai.	Hệ thống thông báo lỗi sai password		

		4. Ấn phím đăng nhập			
TC3	Đăng nhập không thành công	1. Mở chức năng đăng nhập 2. Nhập user đúng 3. Không nhập password 4. Ấn phím đăng nhập	Hệ thống thông báo lỗi để trống password		
TC4	Đăng nhập không thành công	1. Mở chức năng đăng nhập 2. Nhập user sai 3. Ấn phím đăng nhập	Hệ thống thông báo lỗi nhập sai user		
TC5	Đăng nhập không thành công	1. Mở chức năng đăng nhập 2. Bỏ trống user 3. Ấn phím đăng nhập	Hệ thống thông báo lỗi bỏ trống user		

3.2. Các bài thực hành cơ bản

Bài 2.2: Thiết kế testcase cho chức năng cập nhật giỏ hàng.

- Bảng quyết định chức năng cập nhật giỏ hàng

Điều kiện	TH1	TH2	TH3	TH4
Màu sắc	T	B	T	B
Size	T	T	B	B
Hành động				
Kết quả	T	F	F	F

- Kịch bản test chức năng cập nhật giỏ hàng

ID-TC	Tiêu đề	Kịch bản	EO	RO	Kết luận
1	Cập nhật giỏ hàng thành công	- Nhấn vào nút “chọn màu” để thay đổi - Nhấn vào nút “chọn size” để thay đổi - Nhấn nút “cập nhật giỏ hàng”	Hệ thống cho biết cập nhật giỏ hàng thành công		
2	Chưa cập nhật được vào giỏ hàng	- Nhấn vào nút “chọn màu” để thay đổi - Nút “chọn size” bỏ trống - Nhấn nút “cập nhật giỏ hàng”	Hệ thống hiển thị thông báo "Bạn vui lòng chọn size"		
3	Chưa cập nhật được	- Nút “chọn màu” bỏ trống	Hệ thống hiển thị thông báo "Bạn vui lòng chọn màu"		

	vào giỏ hàng	- Nhấn nút “chọn size” để thay đổi - Nhấn nút “cập nhật giỏ hàng”			
4	Chưa cập nhật được vào giỏ hàng	- Nút “chọn màu” bỏ trống - Nút “chọn size” bỏ trống - Nhấn nút “cập nhật giỏ hàng”	Hệ thống hiển thị thông báo "Bạn vui lòng chọn màu và size"		

ID-TC	Tiêu đề	Kịch bản	EO	RO	Kết luận
1	Cập nhật giỏ hàng thành công	Màu: đen -> Màu: trắng Size: 26 -> Size:25	Hệ thống cho biết cập nhật giỏ hàng thành công	Hệ thống cho biết cập nhật giỏ hàng thành công	Pass
2	Chưa cập nhật được vào giỏ hàng	Màu: đen -> Màu: trắng Size: 26 -> Size:	Hệ thống hiển thị thông báo "Bạn vui lòng chọn size"	Hệ thống hiển thị thông báo "Bạn vui lòng chọn size"	Pass
	Chưa cập nhật được vào giỏ hàng	Màu: đen -> Màu: Size: 26 -> Size:25	Hệ thống hiển thị thông báo "Bạn vui lòng chọn màu"	Hệ thống hiển thị thông báo "Bạn vui lòng chọn màu"	Pass
	Chưa cập nhật được vào giỏ hàng	Màu: đen -> Màu: Size: 26 -> Size:	Hệ thống hiển thị thông báo "Bạn vui lòng chọn size và màu"	Hệ thống hiển thị thông báo "Bạn vui lòng chọn size và màu"	Pass

3.3. Các bài thực hành nâng cao

Bài 2.3. Xây dựng bảng quyết định cho PreviousDate, hàm ngược của bài toàn NextDate.

Bài 2.4. Xây dựng bảng quyết định cho bài toán giải phương trình bậc hai.

Bài 2.5. Để có thể rút được tiền từ máy ATM, người dùng cần một trong hai điều kiện: tiền trong tài khoản vẫn còn và lớn hơn số tiền muốn rút (Đối với thẻ debit) hoặc là người dùng được cấp cho một khoản tín dụng từ trước (Đối với thẻ Credit).

Yêu cầu: Thiết kế testcase kiểm thử chức năng Rút tiền từ máy ATM.

Bài thực hành số 3 (số tiết: 05 tiết)

1. Mục tiêu:

- + Vẽ đồ thị dòng điều khiển cho đoạn mã nguồn cần kiểm thử
- + Biết cách tính độ phức tạp của đồ thị
- + Liệt kê đủ số đường thi hành tuyến tính độc lập
- + Sinh testcase đảm bảo các độ đo kiểm thử

2. Yêu cầu:

- + Yêu cầu về điều kiện thực hành: máy PC (laptop), công cụ kiểm thử tự động
- + Yêu cầu sinh viên: Thiết kế tescae đảm bảo các độ đo, thực thi kiểm thử với công cụ

3. Nội dung

3.1. Bài thực hành mẫu

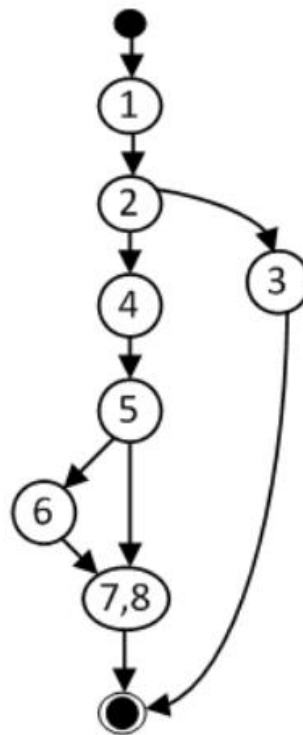
Bài 1. Cho đoạn chương trình sau:

```
float foo(int a, int b, int c, int d){  
1. float e;  
2. if (a==0)  
3.     return 0;  
4. int x = 0;  
5. if ((a==b) || (c==d))  
6.     x = 1;  
7. e = 1/x;  
8. return e;  
}
```

1. Vẽ đồ thị dòng điều khiển cho đoạn chương trình trên
2. Tính hằng số cyclomatic (kí hiệu C) cho sơ đồ vừa vẽ?
3. Sinh các đường đi và các testcase đảm bảo độ đo C1, C2, C3

Hướng dẫn:

1. Vẽ đồ thị:



2. Tính $C = P + 1 = 3$

3. Sinh các đường đi và các testcase đảm bảo độ đo C1, C2, C3

Độ đo C1:

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		

Độ đo C2:

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		
tc3	1; 2; 4; 5; 7,8	2,3,4,5	lỗi chia cho 0		

Độ đo 3:

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5c1; 6; 7,8	0, 2, 3, 5	0		
tc2	1; 2; 4; 5c1; 5c2; 6; 7,8	2, 2, 2, 7	1		
tc3	1; 2; 4; 5c1; 5c2; 7,8	2,3,4,5	lỗi chia cho 0		
tc4	1; 2; 3	2,3,4,4	1		

3.2. Các bài thực hành cơ bản

Bài 2. Cho hàm được viết bằng ngôn ngữ C như sau:

```
//tra ve 1 neu year la nam nhuan, nguoc lai tra ve 0
int LaNamNhuan(int year){
    if( (year%400==0) || (year%4==0 && year%100!=0))
        return 1;

    return 0;
}
```

1. Hãy xây dựng đồ thị dòng điều khiển cho hàm LaNamNhuan
2. Hãy sinh các đường đi và các ca kiểm thử với độ đo C1.
3. Hãy sinh các đường đi và các ca kiểm thử với độ đo C2.
4. Hãy xây dựng đồ thị dòng điều khiển cho hàm LaNamNhuan ứng với độ đo C3.
5. Hãy sinh các đường đi và các ca kiểm thử với độ đo C3.

3.3. Các bài thực hành nâng cao

Bài 3. Cho hàm được viết bằng ngôn ngữ C như sau:

```
//tra lai so ky tu trong xau
int SoKyTu(char a[]){
    int i;
    int total = 0;
    int len = strlen(a); //lay do dai cua xau a
    for (i = 0; i<len; i++){
        if(((a[i]>='A')&&(a[i]<='Z'))||((a[i]>='a')&&(a[i]<='z'))))
            total ++;
    }//end for
    return total;
}
```

1. Hãy xây dựng đồ thị dòng điều khiển cho hàm SoKyTu ứng với độ đo C1 và C2.
2. Hãy sinh các đường đi và các ca kiểm thử với độ đo C1.
3. Hãy sinh các đường đi và các ca kiểm thử với độ đo C2.
4. Hãy sinh các đường đi và các ca kiểm thử với độ đo C3.
5. Hãy sinh các ca kiểm thử để kiểm thử vòng lặp for.

Bài thực hành số 4 (số tiết: 05 tiết)

1. Mục tiêu:

- + Vẽ đồ thị dòng điều khiển cho đoạn mã nguồn cần kiểm thử
- + Biết cách tính độ phức tạp của đồ thị
- + Liệt kê đủ số đường thi hành tuyến tính độc lập
- + Sinh testcase đảm bảo các độ đo kiểm thử

2. Yêu cầu:

- + Yêu cầu về điều kiện thực hành: máy PC (laptop), công cụ kiểm thử tự động
- + Yêu cầu sinh viên: Thiết kế tescae đảm bảo các độ đo, thực thi kiểm thử với công cụ

3. Nội dung

3.1. Bài thực hành mẫu

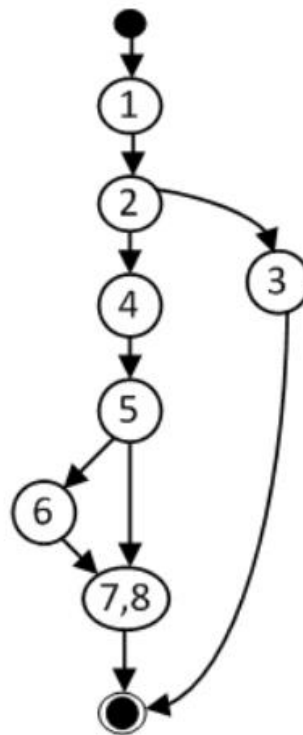
Bài 1. Cho đoạn chương trình sau:

```
float foo(int a, int b, int c, int d){
1.  float e;
2.  if (a==0)
3.      return 0;
4.  int x = 0;
5.  if ((a==b) || (c==d))
6.      x = 1;
7.  e = 1/x;
8.  return e;
}
```

1. Vẽ đồ thị dòng điều khiển cho đoạn chương trình trên
2. Tính hằng số cyclomatic (kí hiệu C) cho sơ đồ vừa vẽ?
3. Sinh các đường đi và các testcase đảm bảo độ đo C1, C2, C3

Hướng dẫn:

1. Vẽ đồ thị:



2. Tính $C = P + 1 = 3$

3. Sinh các đường đi và các testcase đảm bảo độ đo C1, C2, C3

Độ đo C1:

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		

Độ đo C2:

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5; 6; 7,8	2, 2, 3, 5	1		
tc2	1; 2; 3	0, 3, 2, 7	0		
tc3	1; 2; 4; 5; 7,8	2,3,4,5	lỗi chia cho 0		

Độ đo 3:

ID	Test Path	Inputs	EO	RO	Note
tc1	1; 2; 4; 5c1; 6; 7,8	0, 2, 3, 5	0		
tc2	1; 2; 4; 5c1; 5c2; 6; 7,8	2, 2, 2, 7	1		
tc3	1; 2; 4; 5c1; 5c2; 7,8	2,3,4,5	lỗi chia cho 0		
tc4	1; 2; 3	2,3,4,4	1		

3.2. Các bài thực hành cơ bản

Bài 2. Cho đoạn chương trình sau:

```
double Power(double x, int n)
{
    double result = 1;
    for(int i=1; i<n; i++)
        result *= x;

    return result;
}
```

1. Vẽ đồ thị dòng điều khiển cho đoạn chương trình trên
2. Tính hằng số cyclomatic (kí hiệu C) cho sơ đồ vừa vẽ?
3. Sinh các đường đi và các testcase đảm bảo độ đo C1, C2, C3

Bài 3. Cho đoạn chương trình sau:

```
int sum(int a[], int n){
    int i;
    int total = 0;
    for(i=0; i<n; i++)
        total = total + a[i];
    return total;
}
```

1. Vẽ đồ thị dòng điều khiển cho đoạn chương trình trên
2. Tính hằng số cyclomatic (kí hiệu C) cho sơ đồ vừa vẽ?
3. Sinh các đường đi và các testcase đảm bảo độ đo C1, C2, C3
4. Sinh các đường đi và các testcase đảm bảo độ đo cho vòng lặp

3.3. Các bài thực hành nâng cao

Bài 4. Hãy sinh các ca kiểm thử để kiểm thử các vòng lặp lồng nhau của hàm

SelectionSort

```
void SelectionSort(int a[], int size){
    int I,j;
    for(i=0; i<size-1; i++){
        int min = I;
        for(j=i+1; j<size; j++)
            if(a[j]<a[min])
                min = j;
        int tem = a[i];
        a[i] = a[min];
        a[min] = tem;
    }//end for
}//the end
```

Bài thực hành số 5 (số tiết: 05 tiết)

1. Mục tiêu:

- + Vẽ đồ thị dòng dữ liệu cho đoạn mã nguồn cần kiểm thử
- + Xác định tất cả các *Complete-path* từ đồ thị
- + Xác định tất cả các *Def-clear-path* của các biến
- + Xác định các *All-p-uses/Some-c-uses* và *All-c-uses/Some-p-uses*

2. Yêu cầu:

- + Yêu cầu về điều kiện thực hành: máy PC (laptop), công cụ kiểm thử tự động
- + Yêu cầu sinh viên: Thiết kế tescae đảm bảo các độ đo, thực thi kiểm thử với công cụ

3. Nội dung

3.1. Bài thực hành mẫu

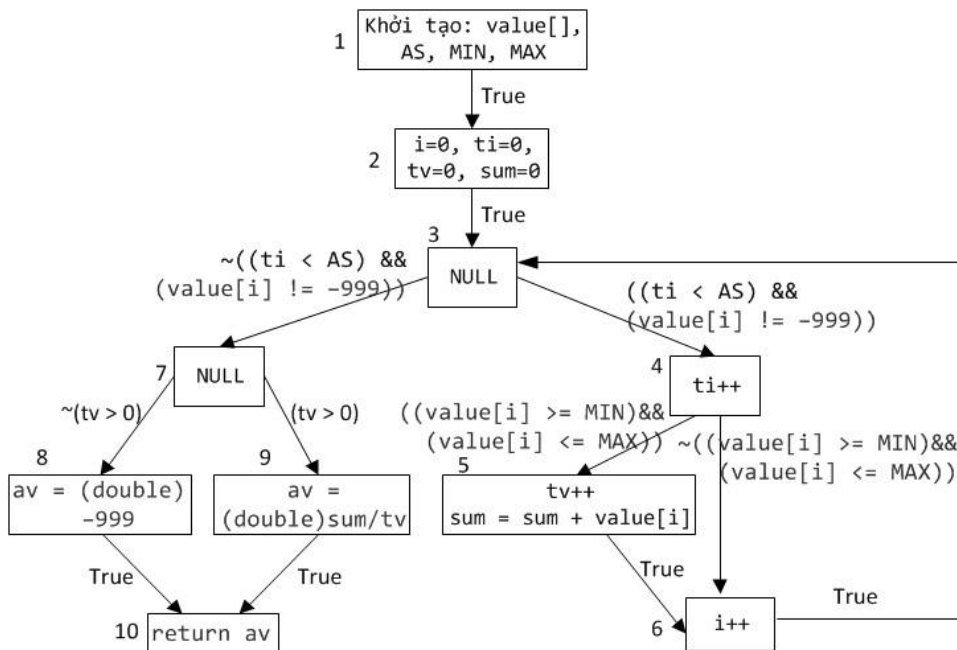
Bài 1. Cho đoạn mã nguồn như sau:

```
double ReturnAverage(int value[], int AS, int MIN, int MAX){
    int i, ti, tv, sum;
    double av;
    i = 0; ti = 0; tv = 0; sum = 0;
    while (ti < AS && value[i] != -999) {
        ti++;
        if (value[i] >= MIN && value[i] <= MAX) {
            tv++;
            sum = sum + value[i];
        }
        i++;
    } //end while
    if (tv > 0)
        av = (double)sum/tv;
    else
        av = (double) -999;
    return (av);
} //the end
```

1. Vẽ đồ thị dòng dữ liệu cho đoạn mã nguồn trên
2. Xác định các def và c-use của các đỉnh
3. Xác định các Complete-path từ đồ thị dòng dữ liệu

Hướng dẫn:

1. Vẽ đồ thị dòng dữ liệu:



2. Xác định các def và c-use của các đỉnh

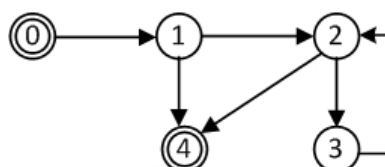
Đỉnh i	$def(i)$	$c-use(i)$
1	{value, AS, MIN, MAX}	{}
2	{i, ti, tv, sum}	{}
3	{}	{}
4	{ti}	{ti}
5	{tv, sum}	{tv, i, sum, value}
6	{i}	{i}
7	{}	{}
8	{av}	{}
9	{av}	{sum, tv}
10	{}	{av}

3. Xác định các Complete-path từ đồ thị dòng dữ liệu

- (1 - 2 - 3 - 4 - 5 - 6 - 3 - 7 - 8 - 10)
- (1 - 2 - 3 - 4 - 6 - 3 - 7 - 8 - 10)
- (1 - 2 - 3 - 4 - 6 - 3 - 7 - 8 - 10)
- (1 - 2 - 3 - 4 - 6 - 3 - 7 - 9 - 10)

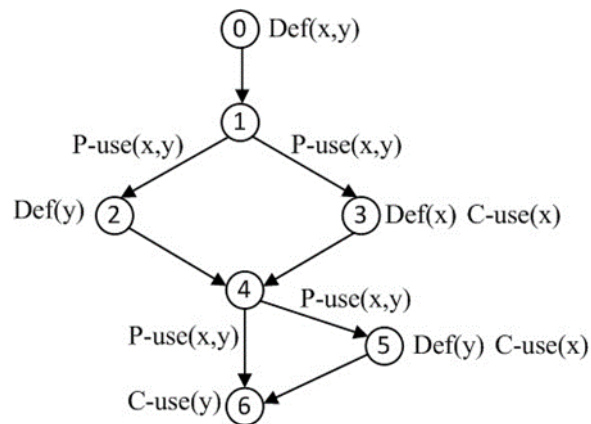
3.2. Các bài thực hành cơ bản

Bài 2. Cho đồ thị dòng dữ liệu như sau:



1. Hãy xác định tất cả các *Complete-path* từ đồ thị này.
2. Các đường đi (0 - 1 - 2 - 3), (1 - 2 - 3 - 2 - 4), (3 - 2 - 4), (2 - 3 - 2) và (0 - 1 - 4) có phải là các *simple-paths* không? Giải thích?
3. Các đường đi (2 - 3 - 2), (2 - 3) và (3 - 2 - 4) có phải là các *loop-free-paths* không? Giải thích?

Bài 3. Cho đồ thị dòng dữ liệu như sau:



1. Hãy xác định tất cả các *Def-clear-path* của các biến x và y.
2. Hãy xác định tất cả các *du-paths* của các biến x và y.
3. Dựa vào các chuẩn của kiểm thử dòng dữ liệu hãy xác định tất cả các *All-p-uses/Some-c-uses* và *All-c-uses/Some-p-uses*.

3.3. Các bài thực hành nâng cao

Bài 4. Cho đoạn lệnh như sau:

```

1 #define MAX=50
2 int intValue=0;
3 int iCount=0;
4 for(int i=0;i<MAX;i++){
5     if(i%2==0){
6         intValue = intValue+i;
7     }
8     else {
9         intValue = intValue*i;
10    } //end if-else
11 } //end for
  
```

1. Biến i trong vòng lặp for có ảnh hưởng đến giá trị của biến intValue không? Tại sao?
2. Hãy xác định các câu lệnh ảnh hưởng đến giá trị của biến intValue tại các câu lệnh 6 và 8.

Bài 5. Cho đoạn lệnh như sau:

```
int returnSumArray(int intArr[], int size){
    int intCount, intSum;
    intCount = 10, intSum = 0;
    while(intCount < size){
        intSum = intSum + intArr[intCount];
        intCount++;
    }//end while
    return intSum;
}
```

1. Hãy xây dựng hàm phân mảnh của hàm này.
2. Xây dựng đồ thị dòng dữ liệu và xác định các định nghĩa (def) và sử dụng (use) của tất cả các biến trong chương trình.

Bài thực hành số 6 (số tiết: 05 tiết)

1. Mục tiêu:

- + Biết cách sử dụng công cụ kiểm thử phi chức năng
- + Thực thi kiểm thử phi chức năng
- + Ghi nhận kết quả và báo cáo kiểm thử

2. Yêu cầu:

- + Yêu cầu về điều kiện thực hành: máy PC (laptop), công cụ kiểm thử phi chức năng
- + Yêu cầu sinh viên: sử dụng thành thạo công cụ kiểm thử phi chức năng.

3. Nội dung

3.1. Bài thực hành mẫu

Bài 1. Kiểm thử hiệu năng website

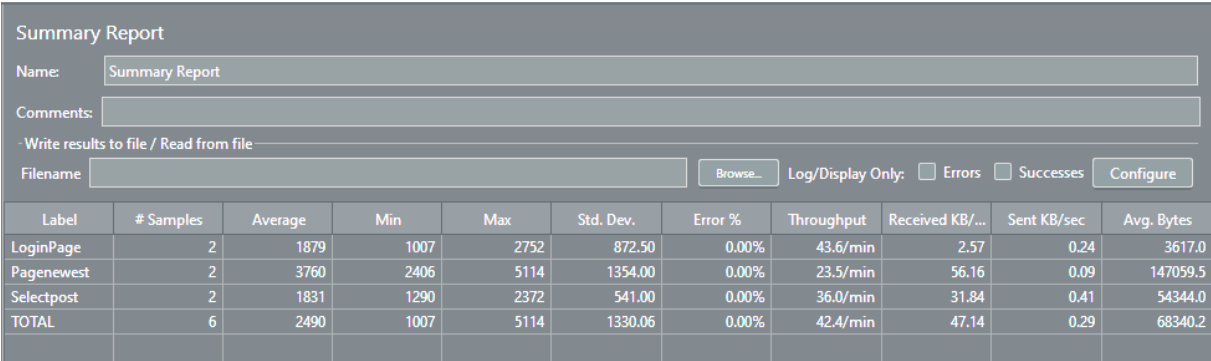
1. Kiểm thử *stress test*
2. Kiểm thử *load test*

Hướng dẫn

1. Kiểm thử *stress test*

Các cấu hình quan trọng nhất của Thread Group là số lượng Threads , Ramp-up time và loop count. Số lượng Threads set cho số lượng người dùng mô phỏng, Ramp-up time set cho thời gian để JMeter bắt đầu thực hiện tất cả các Threads và loop count là số lần kịch bản kiểm tra sẽ được lặp lại.

Với số số lượng user thử nghiệm là ban đầu là 2 user



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Log/Display Only: Errors Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
LoginPage	2	1879	1007	2752	872.50	0.00%	43.6/min	2.57	0.24	3617.0
Pagenewest	2	3760	2406	5114	1354.00	0.00%	23.5/min	56.16	0.09	147059.5
Selectpost	2	1831	1290	2372	541.00	0.00%	36.0/min	31.84	0.41	54344.0
TOTAL	6	2490	1007	5114	1330.06	0.00%	42.4/min	47.14	0.29	68340.2

Kết quả cho chúng ta thấy rằng thời gian xử lý yêu cầu cho page đăng nhập : 2752 mili giây và đối với page chi tiết dự án: 5114 mili giây. Do đó, chúng ta có thể kết luận rằng rất nhiều tài nguyên được sử dụng trong quá trình tải trang newest.

Sau đó, chúng ta có thể bắt đầu tăng số lượng Threads đang thử nghiệm. Chúng ta chuyển sang 10 Threads, 0 ramp-up và 1 vòng lặp. Vì chúng ta vẫn đang kiểm tra một số lượng nhỏ các Threads nên có thể sử dụng JMeter trong chế độ GUI và xem thử nghiệm thông qua trình listener.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: Errors Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
LoginPage	10	1557	916	4155	955.65	0.00%	2.4/sec	8.50	0.79	3620.0
Pagenewest	10	3424	2682	4085	445.34	0.00%	1.6/sec	232.09	0.38	147108.8
Selectpost	10	1333	778	2128	397.01	0.00%	2.1/sec	109.95	1.41	54348.9
TOTAL	30	2105	778	4155	1140.88	0.00%	3.5/sec	234.18	1.45	68359.2

2. Kiểm thử load test

Thiết lập kịch bản test:

- Kịch bản 1: 1000 user test liên tục trong 10 phút. Yêu cầu hệ thống phản hồi < 4000 ms. Tỷ lệ lỗi < 1 %.
- Kịch bản 2: 2000 user test liên tục trong 10 phút. Yêu cầu hệ thống phản hồi < 3000 ms. Tỷ lệ lỗi < 2 %.

- Thêm listener để theo dõi kết quả

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: Errors Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
TOTAL	0	0	#N/A	#N/A	0.00	0.00%	.0/hour	0.00	0.00	.0

3.2. Các bài thực hành cơ bản

Bài 2. Kiểm thử hiệu năng 1 số website vừa và nhỏ

3.3. Các bài thực hành nâng cao

Bài 3. Kiểm thử hiệu năng 1 số website thương mại điện tử, hệ thống bán hàng.