CHƯƠNG 6

KIỂM TRA CHẤT LƯỢNG PHẦN MỀM

Như đã nói ở trước, sản phẩm phần mềm được gọi là đúng nếu nó thực hiện được chính xác những tiêu chuẩn mà người thiết kế đã đặt ra. Để có một đánh giá chính xác về cấp độ đúng của phần mềm, ta phải kiểm tra chất lượng phần mềm. Như thế, kiểm tra là quá trình tìm lỗi và nó là một đánh giá cuối cùng về các đặc tả, thiết kế và mã hoá. Mục đích của kiểm tra là đảm bảo rằng tất cả các thành phần của ứng dụng ăn khớp, vận hành như mong đợi và phù hợp các tiêu chuẩn thiết kế.

Trong chương này, chúng ta thảo luận các chiến lược kiểm tra phần mềm và các kỹ thuật, phương pháp hiệu quả cho mỗi mức độ kiểm tra. Cuối cùng, các công cụ hỗ trợ kiểm tra tự động và các công cụ hỗ trợ kiểm tra độc lập được trình bày để hỗ trợ cho quá trình kiểm tra.

6.1. ĐỘ TIN CẬY CỦA PHẦN MỀM

6.1.1. Chất lượng phần mềm và việc đảm bảo chất lượng phần mềm

Kiểm tra chất lượng phần mềm là một hoạt động khó khăn để chấp nhận về mặt ý thức vì chúng ta đang cân nhắc công việc của chúng ta hoặc của đồng nghiệp để tìm lỗi. Sau quá trình làm việc trong nhóm và trở thành thành viên, chúng ta ngại tìm ra lỗi và không phát hiện được ra chúng thông qua kiểm tra. Khi một người nào đó tiến hành kiểm tra lại không phải là thành viên của dự án, ví dụ một chuyên gia kiểm tra, họ được nhìn nhận như là một kẻ thù.

Thêm vào đó, kiểm tra chất lượng phần mềm lại là một hoạt động khó được chấp nhận đối với việc quản lý vì nó tốn kém, mất thời gian và hiếm khi phát hiện được lỗi. Kết quả là phần lớn các ứng dụng không được kiểm tra đầy đủ và được phát hành với lỗi tiềm ẩn.

Tuy vậy, chất lượng phần mềm cao là một mục tiêu quan trọng của nhóm phát triển phần mềm. Do vậy, cần và phải đảm bảo các tiêu chuẩn của phần mềm như đã đề cập ở chương 2. Đảm bảo chất lượng phần mềm là một hoạt động có hệ thống và kế hoạch. Nó bao gồm nhiều nhiệm vụ liên kết với các hoạt động chính sau:

+ Áp dụng các phương pháp kỹ thuật,
+ Tiến hành các cuộc xét duyệt kỹ thuật chính thức,
+ Kiểm thử phần mềm,
+ Buộc tôn trọng các chuẩn,
+ Kiểm soat thay đổi,
+ Đo chất lượng,
+ Báo cáo, lưu giữ kết quả.

Theo chuẩn ANSI/IEEE, kế hoạch đảm bảo chất lượng phần mềm như sau:

I. Mục đích của kế hoạch
II. Tham khảo
III. Quản lý
  A. Tổ chức
  B. Nhiệm vụ
  C. Trách nhiệm
IV. Tài liệu
  A. Mục đích
  B. Tài liệu công nghệ phần mềm cần thiết
  C. Các tài liệu khác
V. Chuẩn, thực hành và quy ước
  A. Mục đích
  B. Quy ước
VI. Xét duyệt và kiểm toán
  A. Mục đích
  B. Các yêu cầu xét duyệt
    1. Xét duyệt yêu cầu phần mềm
    2. Xét duyệt thiết kế
    3. Kiểm chứng phần mềm và xét duyệt hợp lệ
    4. Kiểm toán chức năng
    5. Kiểm toán vật lý
    6. Kiểm toán trong tiến trình
    7. Xét duyệt quản lý
VII. Quản lý cấu hình phần mềm
VIII. Báo cáo vấn đề và cách sửa chữa
IX. Công cụ, kỹ thuật và phương pháp luận
X. Kiểm soát mã
XI. Kiểm soát phương tiện
XII. Kiểm soát người cung cấp
XIII. Thu thập bảo trì và ghi nhớ báo cáo

Việc đảm bảo chất lượng phần mềm là một hoạt động bản chất cho bất kỳ nhóm phát triển phần mềm nào sản xuất ra phần mềm cho người sử dụng.

6.1.2. Độ tin cậy của phần mềm

6.1.2.1. Các lỗi thường gặp

Khi phân tích chất lượng, phần mềm thường gặp một số lỗi như:

+ Lỗi chiến lược: ý đồ thiết kế sai
+ Phân tích các yêu cầu không đầy đủ hoặc lệch lạc
+ Hiểu sai về các chức năng
+ Vi phạm nguyên lý đối tượng
+ Lỗi tại các thủ tục chịu tải, đây là những lỗi nặng.
+ Lỗi lây lan: lỗi được truyền từ chương trình này sang chương trình khác
+ Lỗi cú pháp: viết sai quy định của ngôn ngữ.

+ Hiệu ứng phụ: lỗi xảy ra khi một đơn vị chương trình làm thay đổi giá trị của một biến ngoài ý kiến của lập trình viên.

Các lỗi của phần mềm tuân theo nguyên lý mức độ lỗi:

a) Mức chịu tải tăng theo chiều đi xuống: lỗi phát ra ở mức dưới được xem là nặng hơn ở mức trên.
b) Lỗi nặng nhất nằm ở mức cao nhất (ý đồ thiết kế) và ở mức thấp nhất (thủ tục chịu tải lớn nhất)

Do vậy, khi phát triển phần mềm, cần đảm bảo nguyên lý an toàn là: Mọi lỗi dù nhỏ lớn đều phải được phát hiện ở một bước nào đó của chương trình, trước khi lỗi đó hoành hành.

6.1.2.2. Độ tin cậy của phần mềm

Độ tin cậy của một hệ phần mềm là độ đo về mức độ tốt của các dịch vụ mà hệ cung cấp cho máy tính. Cần chú ý là người dùng không xét rằng các dịch vụ là quạn trọng như nhau: chẳng hạn một hệ điều khiển máy bay có thể rất, rất hiếm khi thất bại, nhưng nếu chúng có thất bại gây ra tai nạn máy bay thì các người bị nạn và thân nhân người bị nạn không thể xem hệ đó là đáng tin.

Độ tin cậy là một đặc trưng động của hệ thống, nó là một hàm của số các thất bại phần mềm. Một thất bại phần mềm là một sự kiện thi hành mà khi đó phần mềm hành xử không như người ta mong đợi. Chú ý rằng một thất bại phần mềm khác nột hư hỏng phần mềm. Hư hỏng phần mềm là một đặc trưng tĩnh, và nó sẽ gây ra thất bại phần mềm khi mà mã lỗi được thi hành với một tập hợp đặc biệt các thông tin vào. Các hư hỏng không phải luôn luôn xuất đầu lộ diện, vì vậy đọ tin cậy phụ thuộc vào việc sử dụng hệ thống như thế nào. Không thể đưa ra một phát biểu đơn giản và khái quát về độ tin cậy phần mềm.

Các hư hỏng phần mềm không phải là các khuyết tật của chương trình. Một hành xử bất ngờ có thể xảy ra khi mà phần mềm phù hợp với các yêu cầu của nó, nhưng mà chính các yếu tố đó lại không đầy đủ. Các sai sót trong các tư liệu phần mềm cũng có thể dẫn đến các hành vi bất ngờ mặc dầu rằng phần mềm không có khiếm khuyết.

Có công trình nghiên cứu đã chỉ ra rằng có thể rút bỏ 60% các khiếm khuyết mà chỉ có thể cải tạo được 3% độ tin cậy. Cũng có người đã chú ý rằng nhiều khiếm khuyết trong sản phẩm chỉ là kết quả của hàng trăm hoặc hàng nghìn tháng sử dụng.

6.1.2.3. Một số đánh giá vì độ tin cậy

1. *Đặc tả độ tin cậy phần mềm:* gồm các bước

+ Phân tích hệ quả của các thất bại.
+ Chia các thất bại thành các nhóm khác nhau.
+ Thiết lập các yêu cầu về độ tin cậy bằng cách sử dụng các độ đo thích hợp cho từng loại.

2. Đo độ tin cậy: theo một vài cách đo như sau

+ Xác suất thất bại tính theo đòi hỏi.
+ Tỷ lệ xuất hiện thất bại
+ Thời gian trung bình giữa hai thất bại kế tiếp nhau.
+ Độ đo mức sẵn sàng hoạt động của hệ.

3. Thử nghiệm tĩnh

Mục tiêu chủ yếu của thử nghiệm tĩnh là xác định độ tin cậy của phần mềm chứ không phải là xác định các hư hỏng phần mềm.

Quá trình thử nghiệm tĩnh liên quan đến 4 bước sau:

i) Xác định độ đo thao tác phần mềm. Độ đo thao tác là một mẫu sử dụng phần mềm và xác định mẫu đó liên quan đến việc phát hiện các lớp thông tin vào của chương trình và ước tính xác suất của chúng.

ii) Chọn ra hoặc sinh ra một tập các dữ liệu thử tương ứng với độ đo đó.

iii) Áp dụng các trường hợp thử chương trình, ghi lại độ dài thời gian thi hành giữa mỗi cặp thất bại quan sát được. Thích hợp hơn là dùng thời gian thô, với đơn vị thời gian thích hợp cho độ đo mức tin cậy.

iv) Tính toán độ đo mức tin cậy sau một số đáng kể (về mặt thống kê) các thất bại đã quan sát được.

4. An toàn phần mềm

Có những hệ thống mà thất bại của nó có thể gây ra một mối đe dọa tính mạng con người. Thí dụ về hệ thống an toàn sinh mệnh như vậy là hệ thống điều khiển máy bay.

Có hai lớp phần mềm an toàn sinh mệnh.

i) Các phần mềm an toàn sinh mệnh sơ cấp: các phần mềm lồng nhúng trong một hệ phần cứng dùng để điều khiển quá trình khác mà sự làm việc sai sót của nó có thể trực tiếp gây ra thương vong hoặc phá hủy môi trường sống của con người.

ii) Các phần mềm an toàn sinh mệnh thứ cấp: các phần mềm có thể gián tiếp gây ra thương vong. Thí dụ hệ thống phần mềm trợ giúp thiết kế kỹ thuật, hệ thống cơ sở dữ liệu y tế liên quan đến các chất độc bảng A.

5. Thử nghiệm khiếm khuyết

Thử nghiệm chương trình có hai mục đích: thứ nhất là chỉ ra rằng hệ thống là phù hợp với các đặc tả của nó, thứ hai là thực hành hệ thống theo một cách sao cho các khuyêt tật được phơi ra. Các thử nghiệm với mục đích thứ nhất chính là các thẩm định, nó là các thử nghiệm để chấp nhận. Các thử nghiệm cho mục đích thứ hai lại khác hẳn: thử nghiệm thành công nhất là thử nghiệm phơi ra được nhiều khuyết tật nhất.

Các thử nghiệm có thể được phát triển song song với việc thiết kế và thực hiện bởi những người không dính dáng tới việc thiết kế.

6.1.2.4. Lập trình vì độ tin cậy

Lập trình là một là một công đoạn phụ thuộc nhiều vào kỹ xảo cá nhân, sự chú ý đến các chi tiết và kiến thức về việc làm như thế nào để sử dụng các công cụ sẵn có theo cách thức tốt nhất. Nhu cầu các hệ thống đáng tin là đang tăng lên vì vậy cần có các kỹ thuật chuyên biệt nhằm đạt được một hệ thống tin cậy được. Hiện nay, có hai kỹ thuật để tăng độ tin cậy phần mềm khi viết các chương trình ứng dụng là: tránh lỗi và tha thứ lỗi.

1. Tránh lỗi

Tất cả các kỹ sư phần mềm hẳn đều muốn sản ra các phần mềm không có lỗi. Một quá trình phát triển dựa vào việc phát hiện lỗi và trừ khử lỗi chứ không phải là tránh lỗi là một quá trình kém cõi và lỗi thời.

Phần mềm không có lỗi ở đây là phần mềm tuân theo đúng đặc tả. Nói chung, nó có thể có lỗi trong đặc tả hoặc có thể không phản ánh đúng các nhu cầu của người sử dụng vậy là phần mềm không có lỗi không nhất thiết là các phần mềm luôn luôn hành xử như người dùng dự đoán.

Việc phát triển phần mềm không có lỗi là một việc rất đắt đỏ, và khi mà một số lỗi đã được tháo khỏi chương trình thì giá cả cho việc tìm và tháo lỗi còn lại có xu hướng tăng theo hàm số mũ. Do đó một tổ chức có thể quyết định chấp nhận một vài lỗi còn lưu lại. Tính về mặt giá cả thì thà rằng chịu tiền chi trả cho các thất bại của hệ thống do các lỗi đó gây ra còn hơn là đi phát hiện tháo gỡ các lỗi đó trước khi phân phối.

Tránh lỗi và phát triển phần mềm vô lỗi dựa trên:

+ Sản phẩm của một đặc tả hệ thống chính xác.

+ Chấp nhận một cách tiếp cận thiết kế phần mềm dựa trên việc che dấu thông tin và bao gói thông tin.

+ Tăng cường việc duyệt lại trong quá trình phát triển và thẩm định hệ phần mềm.

+ Chấp nhận triết lý chất lượng tổ chức: chất lượng là bánh lái của quá trình phát triển phần mềm.

+ Việc lập kế hoạch cẩn thận cho việc thử nghiệm hệ thống để trưng ra các lỗi mà các lỗi này chưa được phát hiện trong quá trình duyệt lại và để định lượng độ tin cậy của hệ thống.

2. Thứ lỗi

Ngay với một hệ vô lỗi thì vẫn cần một tiện ích thứ lỗi: đó là vì có thể có các lỗi đặc tả. Một tiện ích thứ lỗi là cần thiết cho một hệ thống đáng tin cậy.

Có bốn hoạt động cần phải tiến hành nếu hệ thống phải là thứ lỗi:

+ Phát hiện lỗi

+ Định ra mức độ thiệt hại

+ Hồi phục sau khi gặp lỗi. Hệ thống phải hồi phục về trạng thái mà nó biết là an toàn. Cũng có thể là chỉnh lý trạng thái bị hủy hoại (hồi phục tiến), cũng có thể là lui về một trạng thái trước mà an toàn (hồi phục lùi).

+ Chữa lỗi. Cải tiến hệ thống để cho lỗi đó không xuất hiện nữa. Trong nhiều trường hợp sự thất bại của phần mềm là tàng hình và gây ra bởi một tổ hợp đặc biệt của thông tin vào.

3. Xử lý bất thường

Một sai loại nào đó hoặc một sự cố bất ngờ xuất hiện thì ta gọi chúng là một bất thường. Các bất thường có thể do phần cứng cũng có thể do phần mềm. Khi mà một bất thường không được dự đoán thì bộ điều khiển sẽ chuyển cho cơ chế xử lý bất thường hệ thống. Nếu một bất thường đã được dự đoán thì mã phải bao gồm cả việc phát hiện và việc xử lý bất thường đó.

Hầu hết các ngôn ngữ lập trình là không có các tiện ích để phát hiện và xử lý bất thường.

Các bất thường có thể được ghi lại bằng cách dùng một biến Logic nhằm chỉ ra rằng có một bất thường đã xuất hiện.

4. Lập trình phòng thủ

Lập trình phòng thủ là cách phát triển chương trình mà người lập trình giả định rằng các mâu thuẫn hoặc các lỗi chưa được phát hiện có thể tồn tại trong chương trình. Mã sẽ có phần kiểm tra trạng thái hệ thống sau khi biến đổi và phải đảm bảo rằng sự biến đổi trạng thái là kiên định. nếu phát hiện một mâu thuẫn thì việc biến đổi trạng thái là phải rút lại và trạng thái phải trở về trạng thái đúng đắn trước đó.

Lập trình phòng thủ là một cách thứ lỗi, mà được tiến hành không cần bộ điều khiển thứ lỗi. Về cơ bản quá trình vẫn là: phát hiện lỗi, đánh giá lỗi, và phục hồi sau lỗi. Nói chung một lỗi gây ra một sự sụp đổ trạng thái: các biến trạng thái được gắn các trị không hợp luật. Ngôn ngữ lập trình như Ada cho phép phát hiện các lỗi đó ngay trong thời gian biên dịch. Tuy nhiên việc kiểm tra biên dịch chỉ hạn chế cho các giá trị tĩnh và một vài phép kiểm tra thời gian thực là không thể tránh được.

Một cách để phát hiện lỗi trong chương trình Ada là dùng cơ chế xử lý bất thường kết hợp với đặc tả miền trị.

Hồi phục lỗi là một quá trình cải biên không gian trạng thái của hệ thống sao cho hiệu ứng của lỗi là nhỏ nhất và hệ thống có thể tiếp tục vận hành, có lẽ là trong một mức suy giảm. Hồi phục tiến liên quan đến việc cố gắng chỉnh lại trạng thái hệ thống. Hồi phục lùi liên quan đến việc lưu trạng thái của hệ thống ở một trạng thái đúng đã biết.

Hồi phục tiến thường là một chuyên biệt ứng dụng, mà kiến thức miền được sử dụng để tính ra các sự chỉnh lý có thể. Tuy nhiên có hai tình thế chung mà khi đó hồi phục tiến có thể thành công:

1) Khi dữ liệu mã đã bị sụp. Việc sử dụng mã hóa thích hợp bằng cách thêm các dữ liệu dư thừa vào dữ liệu cho phép sửa sai khi phát hiện lỗi.

2) Khi cấu trúc nối bị sụp. Nếu các con trỏ tiến và lùi đã có trong cấu trúc dữ liệu thì cấu trúc đó có thể tái tạo nếu như còn đủ các con trỏ chưa bị sụp. Kỹ thuật này thường được dùng cho việc sửa chữa hệ thống tệp và cơ sở dữ liệu.

Hồi phục lùi là một kỹ thuật đơn giản liên quan đến việc duy trì các chi tiết của trạng thái an toàn và cất giữ trạng thái đó khi mà một sai đã bị phát hiện. Hầu hết các hệ quản trị cơ sở dữ liệu đều có bộ phục hồi lỗi. Cơ sở dữ liệu chỉ cập nhật dữ liệu một khi giao dịch đã hoàn tất và không phát hiện được vấn đề gì. Nếu giao dịch thất bại thì cơ sở dữ liệu không được cập nhật.

Một kỹ thuật khác là thiết lập các điểm kiểm tra thường kỳ mà chúng là các bản sao của trạng thái hệ thống. Khi mà một lỗi được phát hiện thì trạng thái an toàn đó được tái lưu kho từ điểm kiểm tra gần nhất.

Không may là khi hệ thống dính líu tới nhiều quá trình hợp tác thì dãy các thao tác có thể là các điểm kiểm tra của các quá trình đó là không đồng bộ và để phục hồi thì mỗi quá trình phải lần trở lại trạng thái ban đầu của nó.

6.2. KIỂM TRA VÀ CÁC CHIẾN LƯỢC KIỂM TRA PHẦN MỀM

6.2.1. Đặc điểm của kiểm tra phần mềm

Xác minh và thẩm định một hệ phần mềm là một quá trình liên tục xuyên suốt mọi giai đoạn của quá trình phần mềm. Xác minh và thẩm định là từ chung cho các quá trình kiểm tra để đảm bảo rằng phần mềm thỏa mãn các yêu cầu của chúng và các yêu cầu đó thỏa mãn các nhu cầu của người sắm phần mềm.

Xác minh và thẩm định là một quá trình kéo dài suốt vòng đời. Nó bắt đầu khi duyệt xét yêu cầu. Xác minh và thẩm định có hai mục tiêu:

i) Phát hiện các khuyết tật trong hệ thống.

ii) Đánh giá xem hệ thống liệu có dùng được hay không?

Sự khác nhau giữa xác minh và thẩm định là:

i) Thẩm định: Xem xét cái được xây dựng có *là sản phẩm đúng* không? Tức là kiểm tra xem chương trình có được như mong đợi của người dùng hay không.

ii) Xác minh: Xem xét cái được xây dựng có *đúng là sản phẩm* không? Như thế, xác minh là kiểm tra chương trình có phù hợp với đặc tả hay không.

Như trên, kiểm tra là một quá trình của tìm kiếm lỗi. Một kiểm tra tốt có khả năng cao tìm kiếm các lỗi chưa được phát hiện. Một kiểm tra thành công là kiểm tra tìm ra các lỗi mới, một kiểm tra tồi là kiểm tra mà không tìm được lỗi.

Có hai kiểu lỗi trong ứng dụng và các kiểm tra tốt sẽ xác định cả hai loại đó. Cụ thể:

- Không làm những điều cần phải làm: lỗi bỏ sót thường xuất hiện đối với ứng dụng mới được phát triển.
- Làm những điều không cần làm: lỗi của lệnh đã cư trú trước trong các ứng dụng bảo trì.

Các kiểm tra có mặt tại các mức khác nhau và được tiến hành bởi các cá nhân khác nhau trong quá trình phát triển ứng dụng. Các kiểm tra được tiến hành bởi đội ngũ dự án và kiểm tra được tiến hành bởi các cơ quan bên ngoài để chấp nhận ứng dụng.

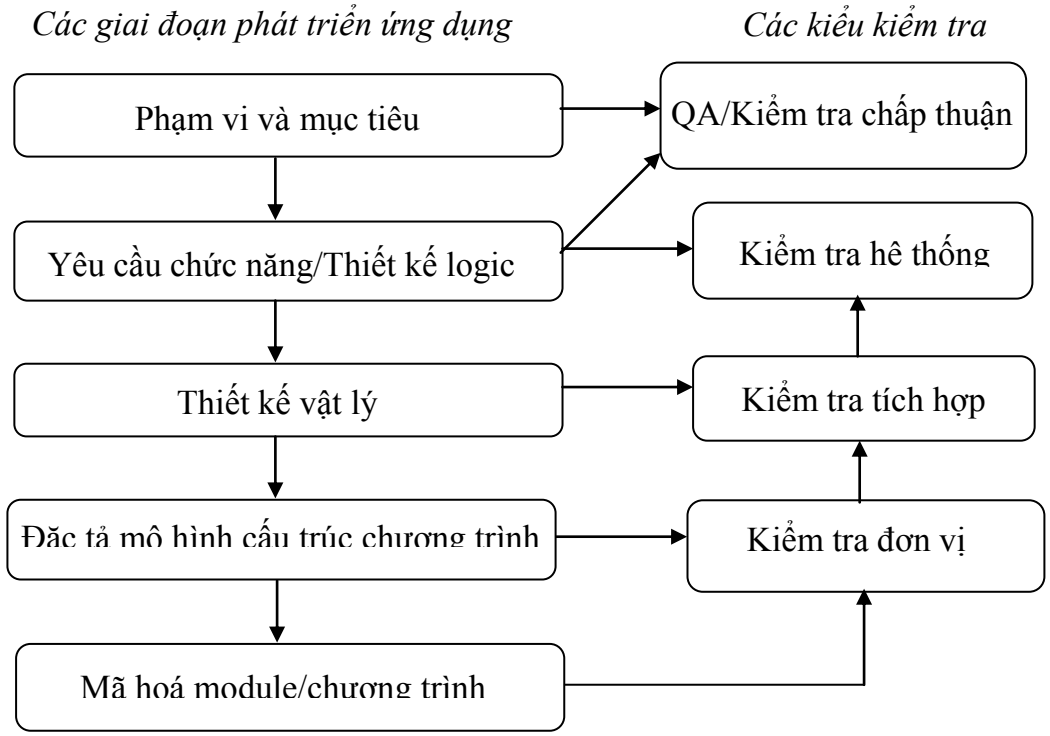
Các kiểm tra của đội dự án được gọi là kiểm tra phát triển (Development test). Kiểm tra phát triển bao gồm kiểm tra đơn vị, kiểm tra hệ thống con, kiểm tra tích hợp và các kiểm tra hệ thống.

- Kiểm tra đơn vị (Unit test) được tiến hành cho mỗi đơn vị mã nhỏ nhất.
- Kiểm tra tích hợp (Subsystem integration test) kiểm tra mặt logic và xử lý cho phù hợp của các khối, kiểm tra việc truyền tin giữa chúng.
- Kiểm tra hệ thống (System test) đánh giá các đặc tả chức năng có được đáp ứng, các thao tác giao diện người có giống thiết kế không, và các công việc của ứng dụng trong môi trường thao tác mong muốn, đối với các ràng buộc.

Các kiểm tra bởi các cơ quan bên ngoài được gọi là đảm bảo chất lượng (Quality assurance-QA) và kiểm tra chấp nhận (Acceptance test). Người ngoài có thể là người sử dụng hoặc đại diện người dùng, nó tạo một mục đích, đánh giá khách quan về ứng dụng và cơ quan bên ngoài được coi là có mục đích hơn các thành viên nhóm.

Kiểm tra đảm bảo chất lượng tương tự các kiểm tra hệ thống về mặt mục đích và tiến hành, nhưng nó khác là họ nằm ngoài sự điều khiển của dự án trưởng. Các báo cáo kiểm tra đảm bảo chất lượng được gửi thường xuyên tới nhóm phát triển và quản lý dự án. Các kiểm tra viên đảm bảo chất lượng lập kế hoạch cho chiến lược của mình và tiến hành kiểm tra để đảm bảo các ứng dụng thực hiện tất cả các chức năng cần thiết. Kiểm tra đảm bảo chất lượng là kiểm tra cuối cùng được làm trước khi ứng dụng được đưa sản xuất đại trà.

Quan hệ giữa các mức kiểm tra và các giai đoạn trong vòng đời của chương trình được trình bày như sau:



Mỗi mức kiểm tra đòi hỏi xác định chiến lược kiểm tra. Chiến lược kiểm tra hộp trắng hoặc kiểm tra hộp đen.

- Dữ liệu vào được tạo theo thiết kế để sinh ra các dữ liệu ra khác nhau mà không chú ý tới các chức năng logic thực hiện thế nào. Các kết quả được dự đoán và so sánh với các kết quả thực tế để đánh giá mức độ thành công.
- Chiến lược White-box mở hộp và nhìn vào các logic đặc tả của ứng dụng để kiểm tra nó làm thế nào. Kiểm tra sử dụng các đặc tả logic để tạo ra các xử lý khác nhau và dự đoán các kết quả ra. Các kết quả trung gian và đầu ra cuối cùng có thể dự đoán và định lượng nhờ kiểm tra white-box.

Chiến lược kiểm tra top–down hay bottom–up: xác định các kiểm tra và phát triển mã sẽ được tiến hành như thế nào:

- Kiểm tra top–down cho rằng mã điều khiển tới hạn và các chức năng sẽ được phát triển và kiểm tra đầu tiên. Tiếp theo là các chức năng thứ cấp và các hàm hỗ trợ. Lý thuyết là càng có nhiều module tới hạn được kiểm tra, thì càng ổn định về chương trình.
- Kiểm tra bottom–up cho rằng càng ít thay đổi trong các module khả năng sinh lỗi càng ít. Toàn bộ các module được mã và đơn vị được kiểm tra. Sau đó kiểm tra được tiến hành ở mức độ tích hợp.

Các chiến lược kiểm tra không loại trừ lẫn nhau, chúng có thể được sử dụng đơn lẻ hoặc đồng thời. Lý tưởng là kiểm tra cho một ứng dụng bao gồm nhiều chiến lược để phát hiện được hết các lỗi.

Sau khi chiến lược đã được xác định, nó được áp dụng để tạo các trường hợp kiểm tra cụ thể. Test case: là các giao dịch riêng biệt hoặc các bản ghi dữ liệu tạo các logic được kiểm tra. Cho mọi trường hợp kiểm tra tất cả các kết quả của xử lý được dự đoán. Đối với ứng dụng on-line và off-line tài liệu hoá, test scripts các hội thoại tạo ra giữa người dùng và ứng dụng và các thay đổi từ hội thoại. Test plan: tư liệu hoá các chiến lược, kiểu, các trường hợp và mô tả cho kiểm tra vài khối ứng dụng. Tất cả các kế hoạch tạo thành kế hoạch tổng thể cho ứng dụng.

Kiểm tra được lặp lại cho đến khi không còn lỗi, hoặc đạt được mức độ chấp nhận.

- Bước đầu tiên của xử lý kiểm tra, đầu vào kiểm tra, cấu hình và mã ứng dụng được đòi hỏi để tạo các kiểm tra.
- Bước thứ hai là so sánh các kết quả kiểm tra với kết quả dự tính và định lượng sự khác biệt.
- Bước tiếp theo là loại trừ các lỗi, hoặc gỡ rối mã. Khi việc mã hoá lại hoàn thành, kiểm tra lại được tiến hành.

Quá trình tiến hành kiểm tra bắt đầu từ khi thiết kế. Cộng tác viên thực hiện việc kiểm tra nên có khả năng của phân tích viên và lập trình viên để có thể hiểu các đòi hỏi của ứng dụng và kiểu cách tiến hành kiểm tra. Chương trình càng lớn và phức tạp thì càng cần người có kinh nghiệm, đôi khi cần có một đội ngũ kiểm tra. Đội ngũ kiểm tra sử dụng yêu cầu chức năng từ giai đoạn phân tích và đặc tả chương trình, đặc tả thiết kế từ giai đoạn thiết kế như là đầu vào để phát triển chiến lược kiểm tra hệ thống. Khi chiến lược đã được hoàn tất, các buổi thảo luận được tiến hành để kiểm tra lại chiến lược và thông báo tới toàn thể đội. Các nhiệm vụ tại các mức sẽ được ấn định. Thời gian được ước lượng. Đội ngũ kiểm tra làm việc độc lập và song song với đội ngũ lập trình. Họ làm việc với quản trị CSDL để phát triển cơ sở dữ liệu mà có thể

hỗ trợ tất cả các mức kiểm tra. Đối với kiểm tra đơn vị, đội kiểm tra kiểm tra kết quả, các chấp nhận các module và chương trình cho kiểm tra tích hợp (*integration test*). Đội ngũ kiểm tra tiến hành và định lượng các kiểm tra tích hợp và kiểm tra hệ thống.

6.2.2. Chiến lược kiểm tra phần mềm

Như đã nói ở trên, có hai kiểu chiến lược kiểm tra. Kiểu thứ nhất liên quan logic được kiểm tra thế nào trong ứng dụng. Chiến lược kiểm tra logic có thể là black-box hoặc white-box. Chiến lược kiểm tra black-box cho ràng module của chương trình hoặc hệ thống liên quan tới đầu vào và đầu ra. Các chi tiết logic chi tiết được che dấu và không cần phân tích. Chiến lược black-box có tính hướng dữ liệu. White-box hướng tới việc cho rằng logic đặc trưng là quan trọng và cần phải kiểm tra. White-box đánh giá một vài hoặc tất cả mặt logic để kiểm tra được tính đúng đắn của chức năng. White-box hướng về logic - giải thuật.

Kiểu thứ hai liên quan tới việc kiểm tra được tiến hành thế nào, không quan tâm chiến lược kiểm tra logic. Nó là top-down hoặc bottom-up. Top-down coi chương trình chính là quan trọng nhất nên cần phải phát triển và kiểm tra trước và tiếp tục trong quá trình phát triển. Bottom-up cho rằng các module và chương trình riêng sẽ được phát triển hoàn toàn như standalone. Vậy chúng được kiểm tra riêng rẽ sau đó được kết hợp thành kiểm tra tổ hợp.

6.2.2.1. Kiểm tra Black-box

Kiểm tra hộp đen, black-box, coi xử lý kết quả như là minh chứng bởi dữ liệu. Khái niệm kiểm tra là black-box không quan tâm logic. Khuynh hướng này hiệu quả đối với các modul chức năng đơn và các hệ thống cấp cao. Ba phương pháp chính là:

- *Phân hoạch cân bằng:* Mục đích của phương pháp này là tối thiểu các trường hợp kiểm tra cho trước, các mục dữ liệu vào được chia thành các nhóm dữ liệu có vai trò cân bằng nhau, mỗi nhóm đại diện cho một tập dữ liệu. Nguyên tắc là bằng cách kiểm tra triệt để một mục của mỗi tập hợp, chúng ta có thể chấp nhận tất cả các mục tương đương khác của tập hợp đó cũng sẽ được kiểm tra một cách kỹ càng.
- *Phân tích cực biên:* Phân tích giá trị cực biên là một dạng nghiêm ngặt của phân hoạch cân bằng có sử dụng các giá trị biên hơn là bất cứ giá trị nào trong tập cân bằng. Ví dụ: miền giá trị của tháng là 1 – 12 và ngoài là 0 và 13. Thì 4 kiểm tra ứng với bốn trường hợp sẽ được kiểm tra phân tích cực biên thường xuyên được dùng tại các mức modul để xác định các mục dữ liệu đặc trưng cho testing.
- *Đoán lỗi:* Dựa trên cơ sở trực giác và kinh nghiệm, các chuyên gia có thể dễ dàng kiểm tra các điều kiện lỗi bằng cách đoán cái nào dễ xảy ra nhất. Ví dụ, chia 0, nếu modul có phép chia, nên dùng phép chia 0. Vì dựa trên trực giác, nên phép thử này khó tìm hết các lỗi.

Một nhược điểm của phân hoạch cân bằng và phân tích cực biên là tổ hợp của các miền hợp không được xác định. Để bổ sung, người ta dùng sơ đồ nguyên nhân – kết quả (Cause – Effect Graphing). Sơ đồ nguyên nhân – kết quả chỉ ra các đầu ra và thông tin di chuyển như là hệ quả và các đầu vào gây ra hệ quả trên. Các ký hiệu

graphic xác định tương tác, lựa chọn, logic và các điều kiện tương đương. Một vòng đại diện một dãy các thao tác không có điểm quyết định hoặc điều khiển. Mỗi dòng biểu diễn một lớp cân bằng của dữ liệu và các điều kiện sử dụng nó. Mỗi lần graph được thực hiện thì ít nhất một giá trị có nghĩa và một không có nghĩa cho mỗi tập được thử.

6.2.2.2. White-box testing

Có ba loại kiểm tra hộp trắng là kiểm tra logic -logic test, chứng minh toán học -mathematical proof và Cleanroom testing.

1. Logic test

Logic test có thể được chi tiết đến mức lệnh. Trong khi thực hiện của mọi lệnh là mục đích đáng khen ngợi, nó có thể không kiểm tra tất cả các điều kiện thuộc chương trình. Ví dụ, ít nhất 2 kiểm tra cho một kiểm tra 2 điều kiện (1 đúng và 1 sai). Cố gắng kiểm tra tất cả các điều kiện lẽ dĩ nhiên là không thể thực hiện được về thực tiễn. Ví dụ trong 1 module có 10 thao tác qua 4 vòng lặp thì có 5,5 triệu trường hợp kiểm tra. Do đó phải có phương pháp lựa chọn.

Logic kiểm tra nhìn vào mỗi quyết định trong module và sản sinh các dữ liệu để tạo tất cả các kết quả ra có thể. Có một vấn đề với logic test tại mức này là chúng không test tình trạng của module đối với đặc tả. Nếu kiểm tra được phát triển dựa trên đặc tả, mà đặc tả được dịch khác nhau bởi người lập trình thì kiểm tra sẽ không đúng. Giải pháp là đòi hỏi đặc tả chương trình đủ chi tiết và logic. Điều này có thể phù hợp với ngôn ngữ thế hệ 1 và 2.

Các kiểm tra nhiều điều kiện tạo mỗi kết quả ra của tiêu chuẩn đa quyết định và nhiều điểm vào module. Các kiểm tra đòi hỏi việc phân tích xác định được bên quyết định đa tiêu chuẩn. Nếu các biên được xác định không chính xác, thì kiểm tra không hiệu quả. Khi được thiết kế phù hợp, test logic đa điều kiện có thể tối thiểu hoá các trường hợp kiểm tra để kiểm tra nhiều nhất các điều kiện. Sự sử dụng kỹ thuật này đòi hỏi luyện tập và kỹ năng.

2. Chứng minh bằng toán học

Một phương pháp theo cách tiếp cận giảm thiếu sót về 0 là áp dụng suy diễn toán học cho đòi hỏi logic, chứng minh tính đúng đắn của chương trình. Phương pháp này đòi hỏi đặc tả ngôn ngữ dạng hình thức. Kỹ thuật chứng minh tính đúng đắn của chương trình được đề cập ở 6.4.

3. Cleanroom testing

Cleanroom testing là mở rộng của phương pháp chứng minh bằng toán học. Lý thuyết của kỹ thuật này là bằng cách ngăn chặn các lỗi tại các đầu vào của quá trình xử lý, giá thành sẽ giảm, độ tin cậy tăng lên và tiệm cận tới không có lỗi.

Phương pháp này được phát triển tại IBM đầu những năm 1980 bởi Mills, Dyer, Linger. Các đặc tả hình thức được dùng và việc kiểm tra thủ công được tiến hành bởi các đội kiểm tra. Các chương trình khó đọc sẽ được viết lại và kiểm tra hoàn toàn trên giấy.

Cleanroom testing là kỹ thuật kiểm tra toán học hình thức và hội ý (walkthrough). Mục đích là phân tích mọi module thành các chức năng và liên kết. Các phép kiểm tra chức năng dùng kỹ thuật toán học, các kiểm tra liên kết sử dụng thuyết tập hợp.

Sau khi thực hiện kiểm tra (verification), đội kiểm tra độc lập sẽ dịch và thực hiện mã. Dữ liệu kiểm tra được dịch bởi các phân tích đặc tả chức năng và được thực hiện thể hiện các tỷ lệ xác suất của dữ liệu được giả định cho hệ thống thực. Thêm vào các dữ liệu chuẩn thì các loại lỗi nghiêm trọng được tạo để kiểm tra ứng dụng.

Bất lợi của phương pháp này bắt buộc là đòi hỏi kỹ năng cao về: toán, thống kê, logic và ngôn ngữ đặc tả hình thức.

6.2.2.3. Kiểm tra top-down

Phương pháp kiểm tra top-down cần một mã ngoài, được hiểu như là một bộ khung để gắn các chức năng gốc, các modul, và các phần khác của ứng dụng. Bộ khung này thường bắt đầu với ngôn ngữ điều khiển công việc và logic chính của ứng dụng. Logic chính được kiểm tra và lập khung theo các hệ thống phân rã. Đầu tiên chỉ có các thủ tục thiết yếu và các logic điều khiển được kiểm tra.

Khi các module thiết yếu nhất đã được kiểm tra và chạy tốt thì mã của các modul ít quan trọng hơn sẽ được gắn vào khung và tiếp tục kiểm tra. Về lý thuyết thì, top-down sẽ tìm thấy các lỗi thiết kế sớm hơn trong kiểm tra thao tác (testing process) hơn các khuynh hướng khác. Phương pháp này ít hiệu quả trong việc cải thiện chất lượng của các phần mềm chuyển giao vì bản chất tương tác của kiểm tra.

Kiểm tra top-down dễ dàng hỗ trợ giao diện người dùng và thiết kế màn hình. Trong các ứng dụng tương tác, thường là bộ điều khiển màn hình được kiểm tra đầu tiên. Người dùng có thể kiểm tra sự điều khiển thông qua màn hình với các phát triển tạo mẫu.

6.2.2.4. Kiểm tra bottom-up

Nguyên tắc của bottom-up là kiểm tra mọi thay đổi tại module có thể ảnh hưởng tới chức năng của nó. Trong kiểm tra bottom-up, toàn bộ khối là đơn vị để đánh giá. Tất cả các module được mã hoá và kiểm tra riêng rẽ.

Các trường hợp kiểm tra: các trường hợp kiểm tra là dữ liệu vào được tạo để thể hiện từng khối và toàn bộ hệ thống thoả mãn tất cả các yêu cầu thiết kế.

Mỗi trường hợp kiểm tra nên được phát triển để kiểm tra nghiệm các đòi hỏi thiết kế đặc trưng, thiết kế chức năng, hoặc mã đã được thoả mãn. Hơn nữa các trường hợp kiểm tra cần dự đoán các output.

Mỗi đơn nguyên của ứng dụng (Ví dụ: module, subroutine, program, utility,...) phải được kiểm tra với ít nhất hai trường hợp kiểm tra: một trường hợp chạy tốt và một trường hợp không chạy. Trong trường hợp chạy sai hệ phải đưa được thông báo, quay lại (rollback) được trạng thái ban đầu của giao dịch.

Để chắc chắn rằng các trường hợp được toàn diện nhất, người ta thường dùng ma trận. Chúng được dùng cho:

- Kiểm tra đơn khối để định nhánh logic, điều kiện logic, các phần dữ liệu hoặc biên dữ liệu để kiểm tra trên cơ sở đặc tả chương trình.
- Kiểm tra tổ hợp để định ra yêu cầu về dữ liệu và quan hệ trong số các tương tác.
- Kiểm tra hệ thống để xác định yêu cầu về người dùng và hệ thống từ các yêu cầu chức năng và các yêu cầu chấp nhận.

Phối hợp các kiểm tra trong một chiến lược: mục đích của các nhà kiểm tra là tìm ra sự cân bằng của các chiến lược cho phép họ chứng minh được ứng dụng chạy tốt mà tối thiểu hoá chi phí máy tính và nhân lực. Sử dụng duy nhất một chiến lược là rất nguy hiểm. Không có cái nào là duy nhất đúng. Nếu chỉ có white-box thì tài nguyên nhân lực và máy là rất tốn kém, nếu chỉ có black-box các vấn đề logic đặc trưng có thể chưa được khám phá.

6.3. KỸ THUẬT KIỂM THỬ PHẦN MỀM VÀ ĐẶC ĐIỂM

6.3.1. Khái niệm

Kiểm thử một sản phẩm phần mềm là xây dựng một cách có chủ đích những tập dữ liệu và dãy thao tác nhằm đánh giá một số hoặc toàn bộ các tiêu chuẩn của sản phẩm phần mềm đó.

Thử nghiệm có hai mục đích: chỉ ra hệ thống phù hợp với đặc tả và phơi ra được các khuyết tật của hệ thống.

6.3.2. Đặc điểm của kiểm thử

6.3.2.1. Các hạn chế của kiểm thử

- Do kiểm thử là chạy thử chương trình với tập dữ liệu giả nên không thể khẳng định tính đúng của chương trình do bản chất quy nạp không hoàn toàn của nó.
- Trong nhiều trường hợp, việc kiểm thử thường được thực hiện từ những giai đoạn đầu của quá trình cài đặt sản phẩm.
- Các chương trình nên được kiểm chứng theo hai kỹ thuật: kiểm thử và chứng minh. Và nếu có thể nên khẳng định tính đúng của chương trình thông qua văn bản chương trình

Như vây, một chương trình tuyệt đối đúng phải được thực hiện thông qua: tính đúng đắn của thuật toán và tính tương đương của chương trình với thuật toán (được thể hiện ở chứng minh thông qua văn bản chương trình).

Việc kiểm thử chương trình chỉ là nhìn sự kiện đưa ra kết luận do vậy không thể khẳng định một chương trình tuyệt đối đúng bằng kiểm thử. Tuy vậy, bộ dữ liệu kiểm thử phải phủ kín mọi trường hợp cần đánh giá.

Thêm vào đó, trong quá trình kiểm thử, ta thường mắc phải các đặc trưng của nguyên lý chủ quan như sau:

- Bộ dữ liệu Test không thay đổi trong quá trình xây dựng phần mềm
- Chỉ Test các trường hợp chính thống, hợp lệ, không quan tâm đến các cận và các sự cố
- Cài đặt chức năng nào thì chỉ Test riêng chức năng đó, không chỉ Test tổng hợp chức năng vừa cài đặt với các chức năng đã cài đặt trước đó.
- Người Test đồng thời là người xây dựng phần mềm tức vừa đá bóng, vừa thổi còi.

6.3.2.2. Các loại hình kiểm thử

- Kiểm thử lược đồ hệ thống: chỉ quan tâm đến các bản chọn (menu) đánh giá tính hợp lý, khả năng chọn một mục, khả năng di chuyển qua mục khác, tính đủ, tính khoa học của các chức năng.
- Kiểm thử cận dưới
- Kiểm thử cận trên: cho hệ thống thực hiện đến mức tối hạn.
- Kiểm thử qua sự cố: tạo ra các sự cố để kiểm thử phần mềm.

6.3.2.3. Nguyên tắc kiểm thử

- Nguyên tắc khách quan: người kiểm thử không phải là tác giả của phần mềm đang kiểm thử
- Nguyên tắc ngẫu nhiên: dữ liệu và chức năng được chọn, tuy có chủ đích nhưng không phải xuất hiện theo thứ tự nhất định.
- Nguyên tắc "người sử dụng kém": hệ thống được một người sử dụng có trình độ thấp (ở mức chấp nhận được) dùng thử. (Người này có thể gây các sự cố có thể không lường trước được của hệ thống)
- Nguyên tắc "kẻ phá hoại": hệ thống rơi vào tay có trình độ nghiệp vụ cao, chủ ý phá hoại. "Trình độ" ở đây thuộc lĩnh vực công nghệ thông tin hoặc lĩnh vực phần mềm đang hướng tới.

6.3.2.4. Kỹ thuật kiểm thử

- Kỹ thuật đối xứng: dựa vào tính đối xứng của các thao tác hoặc tập dữ liệu để xây dựng bộ dữ liệu Test.
- Kỹ thuật đám đông
- Kỹ thuật kiểm thử trên dữ liệu thật: cho hệ thống vận hành với các tập dữ liệu thật đã thu được từ trước để so sánh và đánh giá kết quả
- Kỹ thuật kiểm thử trên thị trường thật: cho hệ thống vận hành trên thị trường thật (không chính thức) để so sánh với các hệ thống chính được dùng và đánh giá kết quả.
- Kỹ thuật đối sánh: cho thực hiện với một vài sản phẩm khác với cùng các chức năng giống nhau và trên cùng các tập dữ liệu rồi lập bảng so sánh các chức năng.

6.3.2.5. Quá trình kiểm thử

Trừ hệ thống nhỏ, nói chung không nên kiểm thử nguyên cả khối; quá trình kiểm thử có thể chia 5 giai đoạn:

1. Thử đơn vị
2. Thử module
3. Thử hệ con
4. Thử hệ thống
5. Thử nghiệm thu: còn gọi thử anpha.

Khi hệ thống được đem bán còn phép thử beta: phân phối hệ thống cho một số người dùng đồng ý dùng thử và báo cáo lại các vấn đề cho người phát triển hệ thống.

6.3.3. Kế hoạch thử nghiệm

Thử hệ thống là rất đắt đỏ, đối với một vài hệ thời gian thực có các ràng buộc thời gian phức tạp thì việc thử có thể ngốn hết khoảng nửa tổng chi phí phát triển.Vì thế mà phải lập kế hoạch thử và khống chế chi phí thử.

Cần chú ý là việc thử liên quan đến việc thiết lập ra các mẫu cho quá trình thử nhiều hơn là mô tả các phép thử.

6.3.4. Phân loại một số công cụ kiểm thử tự động

Vì kiểm thử phần mềm thường chiếm tới 40% tất cả các nổ lực dành cho một dự án xây dựng phần mềm, nên công cụ có thể làm giảm thời gian kiểm thử (không làm giảm tính kỹ lưỡng) sẽ rất có giá trị. Thừa nhận lợi ích tiềm năng này, các nhà nghiên cứu và người thực hành đã phát triểnmột số thế hệ các công cụ kiểm thử tự động:

- *Bộ phân tích tĩnh*. Các hệ thống phân tích chương trình này hỗ trợ cho "việc chứng minh" các lý lẽ tĩnh - những mệnh đề yếu kém về cấu trúc và định dạng của chương trình.

- *Bộ kiểm toán mã*. Những bộ lọc chuyên dụng này được dùng để kiểm tra chất lượng của phần mềm để đảm bảo rằng nó đáp ứng các chuẩn mã hoá tối thiểu.

- *Bộ xử lý khẳng định*. Những hệ thống tiền xử lý/hậu xử lý này được sử dụng để cho biết liệu những phát biểu do người lập trình nêu, được gọi là các khẳng định, về hành vi của chương trình có thực sự được đáp ứng trong việc thực hiện chương trình thực hay không.

- *Bộ sinh tệp kiểm thử*. Những bộ xử lý này sinh ra, và điền các giá trị đã xác định, vào các tệp đọc vào điển hình cho chương trình đang được kiểm thử.

- *Bộ sinh dữ liệu kiểm thử*. Những hệ thống phân tích tự động này hỗ trợ cho người dùng trong việc chọn dữ liệu kiểm thử làm cho chương trình hành xử theo một cách đặc biệt.

- *Bộ kiểm chứng kiểm thử*. Những công cụ này đo mức bao quát kiểm thử bên trong, thường được diễn tả dưới dạng có liên quan tới cấu trúc điều khiển của sự vật kiểm thử, và báo cáo về giá trị bao quát cho chuyên gia đảm bảo chất lượng.

- *Dụng cụ kiểm thử*. Lớp các công cụ này hỗ trợ cho việc xử lý các phép kiểm thử bằng cách làm gần như không khó khăn để (1) thiết lập một chương trình ứng cử viên trong môi trường kiểm thử, (2) nạp dữ liệu vào, và (3) mô phỏng bằng các cuống cho hành vi của các module phụ.

- *Bộ so sánh cái ra*. Công cụ này làm cho người ta có thể so sánh một tập cái ra từ một chương trình này với một tập cái ra khác (đã được lưu giữ trước) để xác định sự khác biệt giữa chúng.

- *Hệ thống thực hiện ký hiệu*. Công cụ này thực hiện việc kiểm thử chương trình bằng cách dùng cái vào đại số, thay vì giá trị dữ liệu số. Phần mềm được kiểm thử vậy xuất hiện để kiểm thử các lớp dữ liệu, thay vì chỉ là một trường hợp kiểm thử đặc biệt. Cái ra là đại số và có thể được so sánh với kết quả trông đợi cũng được xác định dưới dạng đại số.

- *Bộ mô phỏng môi trường*. Công cụ này là một hệ thống dựa trên máy tính giúp người kiểm thử mô hình hoá môi trường bên ngoài của phần mềm thời gian thực và rồi mô phỏng các điều kiện vận hành thực tại một cách động.

- *Bộ phân tích luồng dữ liệu*. Công cụ này theo dõi dấu vết luồng dữ liệu đi qua hệ thống (tương tự về nhiều khía cạnh với bộ phân tích đường đi) và cố gắng tìm ra những tham khảo dữ liệu không xác định, đặt chỉ số sai và các lỗi khác có liên quan tới dữ liệu.

Hiện nay việc dùng các công cụ tự động hoá cho kiểm thử phần mềm đang phát triển, và rất có thể là ứng dụng đó sẽ phát triển nhanh trong thập kỷ tới. Các công cụ kiểm thử có thể sẽ gây ra những thay đổi lớn trong cách chúng ta kiểm thử phần mềm và do đó cải tiến độ tin cậy của các hệ thống dựa trên máy tính.

6.4. CHỨNG MINH TOÁN HỌC TÍNH ĐÚNG ĐẮN CỦA CHƯƠNG TRÌNH

Như đã đề cập ở trên, mục tiêu của chứng minh toán học là để có thể khẳng định tính đúng của chương trình thông qua chính văn bản của chương trình.

6.4.1. Khái niệm chung

Như ta đã biết, chương trình P là một bộ biến đổi tuần tự P để chuyển cái vào x thành ra cái y; ở đây x và y hoàn toàn được xác định trước.

Như vậy, một chương trình P được gọi là đúng nếu nó thực hiện chính xác những mục tiêu do người thiết kế đặc ra. Ta gọi:

+ Giả thiết $\{A\}$ là mệnh đề được phát biểu để thể hiện tính chất của cái vào, gọi tắt là mệnh đề dữ liệu vào.

+ Kết luận ⟨B⟩ là mệnh đề được phát biểu để tính chất cần có của dữ liệu ra, gọi tắt là mệnh đề dữ liệu ra.

Do P có tính tuần tự và hữu hạn nên có thể biểu diễn P là một dãy liên tiếp các cấu trúc điều khiển P_1, P_2,....,P_n. Do vậy, bằng cách nào đó mà ta khẳng định được:

P_1 biến đổi ⟨A⟩ thành ⟨A_1⟩

P_2 biến đổi ⟨A_1⟩ thành ⟨A2⟩

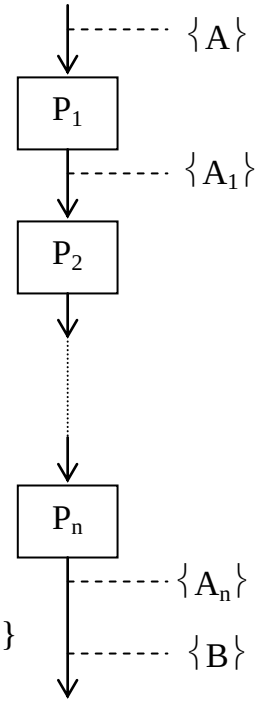
....

P_n biến đổi ⟨A_{n-1}⟩ thành ⟨A_n⟩

Và dựa vào quy tắc toán học, ⟨A_n⟩ có thể suy ra ⟨B⟩ thì ta có thể nói rằng P là đúng với cái vào ⟨A⟩ và cái ra ⟨B⟩. Lúc này ký hiệu ⟨A⟩P⟨B⟩ hay $\langle A\rangle \overset{P}{=\!>} \langle B\rangle$.

Cần chú ý rằng $\langle A\rangle \overset{P}{=\!>} \langle B\rangle$ là khác với $\langle A\rangle \overset{L}{=\!>} \langle B\rangle$:mệnh đề {A} suy diễn ra mệnh đề {B} dựa vào các quy tắc toán học.

Nói cách khác, để chứng minh P là đúng, ta chứng minh theo sơ đồ sau:



⟨A⟩ P_1 ⟨A_1⟩

⟨A_1⟩ P_2 ⟨A_2⟩

.....

.....

⟨A_{n-1}⟩P_n⟨A_n⟩

$\langle A_n\rangle \overset{L}{=\!>} \langle B\rangle$

Ở đây, cần để ý là tính chất ⟨A⟩ và tính chất ⟨B⟩ có thể không liên quan đến nhau.

Ví dụ 1: Cho mệnh đề dữ liệu vào {A: x,y∈R; 0<x<1}

Đoạn trình P =$P_1 \cup P_2 \cup P_3 \cup P_4$ như sau:

x:=1/x+1; (P_1)

y:=y+1; (P_2)

x:=x+2; (P_3)

x:=x+y; (P_4)

và mệnh đề dữ liệu ra {B: x,y∈R; x>y+3}

Lúc này ta có dãy biến đổi tính chất dữ liệu vào/ ra như sau:

{A} P_1{A_1: x,y∈R; x>2}

{A_1}P_2{A_2: x,y∈R; x>2}

{A_2}P_3{A_3: x,y∈R; x>4}

{A_3}P_4{A_4: x,y∈R; x>y+4}

và $\langle A_4\rangle \overset{L}{=\!>} \langle B\rangle$

Vậy ta có kết luận {A}P{B} hay nói cách khác là P đúng với dữ liệu vào {A} và dữ liệu ra {B}.

Cần để ý rằng khí ta có dãy biến đổi tính chất dữ liệu vào và ra như sau:

$\{A\}\ P_1\ \{A_1\}$

$\{A_1\}\ P_2\ \{A_2\}$

.....

.....

$\{A_{n-1}\}P_n\{A_n\}$

$\{A_n\}\overset{L}{\not\Rightarrow}\{B\}$

Thì chưa thể kết luận được điều gì vì còn tuỳ thuộc vào các mệnh đề trung gian thu được $\{A_1\},\{A_2\},....\{A_n\}$ là đã "mạnh nhất" hay chưa.

Xét ví dụ đã cho ở trên, ta có dãy biến đổi như sau:

$\{A\}\ P_1\{A'_1: x,y\in R;\ x>0\}$

$\{A'_1\}P_2\{A'_2: x,y\in R;\ x>0\}$

$\{A'_2\}P_3\{A'_3: x,y\in R;\ x>2\}$

$\{A'_3\}P_4\{A'_4: x,y\in R;\ x>y+2\}$

Rõ ràng ta có: $\{A'_4\}\overset{L}{\not\Rightarrow}\{B\}$ nhưng theo trên ta vẫn có kết luận $\{A\}P\{B\}$. Trong trường hợp này, ta thấy các mệnh đề $\{A'_1\}\{A'_2\}\{A'_3\}\{A'_4\}$ rõ ràng là các mệnh đề hệ quả của các mệnh đề $\{A_1\}\{A_2\}\{A_3\}\{A_4\}$.

Ví dụ 2: Cho mệnh đề dữ liệu vào $\{A: x,y\in N;\ x=3y\}$, đoạn trình $P = P_1 \cup P_2$ như sau:

x:=x+5; (P_1)

y:=y+5; (P_2)

và mệnh đề dữ liệu ra $\{B: x,y\in R;\ x=3y\}$. Ở đây, rõ ràng ta có $\{A\}\overset{P}{\not\Rightarrow}\{B\}$

6.4.2. Hệ tiên đề Hoare

1. Tiên đề 1: Tiên đề tuần tự

Nếu mệnh đề $\{A\}$ sau khi chịu tác động của khối cấu trúc điều khiển P ta được $\{B\}$ và mệnh đề $\{B\}$ sau khi chịu tác động của cấu trúc điều khiển Q ta được $\{C\}$ thì $\{A\}$ chịu tác động tuần tự P,Q sẽ thu được $\{C\}$

Hay nói cách khác, đây chính là tiên đề về dãy thao tác: Nếu $\{A\}\ P\ \{B\}$ và $\{B\}\ Q\ \{C\}$ thì $\{A\}\ P,Q\ \{C\}$

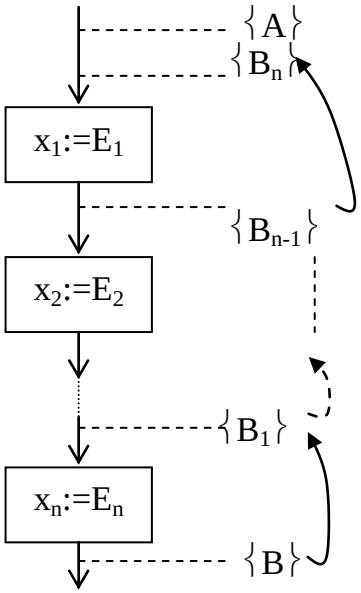
2. Tiên đề gán: tính chất của phép gán

Điều kiện để có mệnh đề $\{B\}$ sau khi thực hiện lệnh gán x: = E (với E là một biểu thức) từ mệnh đề $\{A\}$ thì trước đó ta phải có $\{A\}$ suy dẫn được ra $\{B[x|E]\}$.

Mệnh đề $\{B[x|E]\}$ là mệnh đề thu được từ $\{B\}$ bằng phép thay thế mọi xuất hiện của x trong $\{B\}$ bởi E. Tức là: $\{A\}\ x: = E\ \{B\}$ thì $\{A\}\overset{L}{\Rightarrow}\{B[x|E]\}$

- *Kỹ thuật lần ngược của tiên đề gán*

Cho đoạn trình P gồm n phép gán $x_1:=E_1$; $x_2:=E_2$;.....$x_n:=E_n$; để $\{A\}P\{B\}$ thì ta phải có $\{A\} \overset{L}{\Rightarrow} \{B_n\}$. Trong đó $\{B_n\}$ được xác định như sau



Trong đó các mệnh đề $\{B_i\}$ được xác định như sau:

$\{B_1\}$ là mệnh đề $\{B[x_n|E_n]\}$

$\{B_{n-1}\}$ là mệnh đề $\{B_{n-2}[x_2|E_2]\}$

$\{B_n\}$ là mệnh đề $\{B_{n-1}[x_1|E_1]\}$

Trong trường hợp $\{A\} \overset{L}{\not\Rightarrow} \{B_n\}$ thì ta nói P là có lỗi.

Ví dụ 3: (Xét ví dụ 1) Cho mệnh đề dữ liệu vào {A: x,y∈R; 0<x<1},
Đoạn trình P =$P_1 \cup P_2 \cup P_3 \cup P_4$ như sau:

x:=1/x+1; (P_1)

y:=y+1; (P_2)

x:=x+2; (P_3)

x:=x+y; (P_4)

và mệnh đề dữ liệu ra {B: x,y∈R; x>y+3}. Hãy khảo sát $\{A\}P\{B\}$ hay không?

Ta có

$\{B[x|x+y]\} \equiv \{B_1 : x+y, y \in R;\ x+y>y+3\}$

$\{B_1[x|x+2]\} \equiv \{B_2 : (x+2)+y, y \in R;\ (x+2)+y>y+3\}$

$\{B_2[y|y+1]\} \equiv \{B_3 : (x+2)+(y+1), (y+1) \in R;\ (x+2)+(y+1)>(y+1)+3\}$

$\{B_3[x|1/x+1]\} \equiv \{B_4 : ((1/x+1)+2)+(y+1), (y+1) \in R;\ ((1/x+1)+2)+(y+1)>(y+1)+3\}$

Rõ ràng ta có $\{A\} \overset{L}{\Rightarrow} \{B_4\}$, nên $\{A\}P\{B\}$.

3. Tiên đề rẽ nhánh

i. Với mệnh đề dữ liệu vào {A}, mệnh đề dữ liệu ra {B}, biểu thức logic E, và đoạn trình P. Nếu ta có {A, E}P{B} và $\{A,!E\} \xRightarrow{L} \{B\}$ thì ta nói rằng mệnh đề {A} và {B} tuân theo cấu trúc rẽ nhánh dạng khuyết với cấu trúc P và điều kiện lựa chọn E; tức là: {A} *if* E *then* P*;* {B}.

ii. Với mệnh đề dữ liệu vào {A}, mệnh đề dữ liệu ra {B}, biểu thức logic E, và các đoạn trình P, Q. Nếu ta có {A, E}P{B} và {A,!E}Q{B} thì ta nói rằng mệnh đề {A} và {B} tuân theo cấu trúc rẽ nhánh dạng đủ với cấu trúc P, Q và điều kiện lựa chọn E; tức là: {A} *if* E *then* P *else* Q*;* {B}.

Ví dụ 4: Cho mệnh đề dữ liệu vào {A: x,y,q,r∈N, x=qy+r, 0≤r<2y}, đoạn trình P như sau:

If y≤r *then*
Begin
q:=q+1;
r:=r-y;
End;

Và mệnh đề dữ liệu ra {B: x,y,q,r∈N, x=qy+r, 0≤r<y}. Hãy xem {A}P{B}?

Áp dụng tính chất của phép gán, ta có:

i. {A,E: x,y,q,r∈N, x=qy+r, 0≤ r<2y, y≤ r}q:=q+1;r:=r-y;{B}

ii. $\{A,!E: x,y,q,r\in N, x=qy+r, 0\le r<2y, y>r\} \xRightarrow{L} \{B\}$

do đó suy ra {A}P{B}.

4. Tính bất biến của chương trình

Cho mệnh đề dữ liệu vào {A} và đoạn trình P. Nếu ta có {A}P{A} thì ta nói rằng tính chất dữ liệu của mệnh đề {A} không thay đổi khi chịu sự tác động của đoạn trình P và lúc này người ta nói rằng mệnh đề {A} là bất biến đối với P, tức ta có: {A}P {A}.

Ví dụ 5: Ta có mệnh đề {A: x∈R, x>0} là bất biến đối với đoạn trình P: x:=x*x; vì ta có {A}P{A}.

5. Tiên đề lặp

Cho mệnh đề dữ liệu vào {A}, biểu thức logic E và đoạn trình P. Nếu mệnh đề {A} tuân theo cấu trúc lặp P với điều kiện lặp E thì mệnh đề {A} sẽ bất biến đối với P trong điều kiện E, tức là {A,E}P{A}, kết thúc vòng lặp ta có mệnh đề {A,!E}. Lúc này ta viết: {A} *while* E *do* P; {A,!E}.

Ví dụ 6: Cho x,y,z là 3 số nguyên không âm. Hãy viết chương trình để tính z=xy, biết rằng x,y được nhập từ bàn phím. Hãy khẳng định tính đúng của chương trình.

Ta có đoạn trình như sau:

Vào: x,y,z∈N; x=a; y=b;

Ra: x,y,z∈N; z=ab;

Chương trình P được viết:

```
z:=0;
while x>0 do
        Begin
                If (x mod 2)≠0 then z:=z+y;
                x=x div 2;
                y:=y*2;
        End;
Return z;
```

Ta cần phải khẳng định chương trình trên đúng với yêu cầu đặt ra.

Thật vậy, gọi mệnh đề thể hiện tính chất dữ liệu vào của chương trình {A} và mệnh đề thể hiện tính chất dữ liệu ra cần có {B}, ta có

{A: x,y,z∈N; x=a; y=b;} và {B: x,y,z∈N; z=ab;}

Ta cần chứng tỏ {A}P {B}.

+ Xét mệnh đề {C: x,y,z∈N; ab=z+xy;}
+ Ta có {A} z:=0;{C}
+ Để chứng tỏ {C} là bất biến của đoạn trình

```
while x>0 do
        Begin
                If (x mod 2)≠0 then z:=z+y;
                x=x div 2;
                y:=y*2;
        End;
```

Ta cần có: {C,E: x,y,z∈N; ab=z+xy;x>0}Q{C}, với đoạn trình Q như sau:

```
If (x mod 2)=0 then z:=z+y;
x=x div 2;
y:=y*2;
```

Theo tính chất của phép gán, ta có:

$\{C_1\}\equiv\{C[y|y*2]: x,y*2,z\in N; ab=z+x(y*2);\}$

$\{C_2\}\equiv\{C_1[x|(x \text{ div } 2)]: (x \text{ div } 2),y*2,z\in N; ab=z+(x \text{ div } 2)(y*2);\}$

Nên cần chứng tỏ:

{C,E: x,y,z∈N; ab=z+xy;x>0} *If* (x mod 2)≠0 *then* z:=z+y; $\{C_2\}$

Dễ dàng ta có

i. {C,E,F: x,y,z∈N; ab=z+xy;x>0,(x mod 2)≠0} z:=z+y $\{C_2\}$; và

ii..{C,E,!F: x,y,z∈N; ab=z+xy;x>0,(x mod 2)=0} $\overset{L}{=>}\{C_2\}$;

Vậy {C} là bất biến của Q. Nên kết thúc Q, ta có mệnh đề {C,!E}.

+ Dễ dàng chứng tỏ: $\{C,!E\} \overset{L}{=>} \{B\}$

Vậy ta có {A}P{B}, hay chương trình trên là đúng.

Để ý rằng: do {A,E}P{A} nên trong trường hợp $\{A\} \overset{L}{=>} E$ thì vòng lặp là vô hạn và không tồn tại mệnh đề {A, !E}.

Câu hỏi

1. Độ tin cậy của phần mềm? Vì sao phải đảm bảo chất lượng phần mềm.
2. Đặc điểm của việc kiểm tra phần mềm? Sự khác nhau của kiểm tra hộp trắng và hộp đen.
3. Thế nào là kiểm thử phần mềm? Các đặc điểm của việc kiểm thử phần mềm.
4. Thế nào là kiểm tra tính đúng đắn của văn bản chương trình.
5. Đánh giá các công cụ hỗ trợ kiểm thử phần mềm