abcChương 8:Kiểm thử phần mềm

Mục tiêu

Mục tiêu của chương này là giới thiệu các quy trình kiểm thử phần mềm và kiểm thử phần mềm. Khi bạn đã đọc chương này, bạn sẽ:

1.hiểu các giai đoạn của kiểm thử từ kiểm thử, trong quá trình phát triển đến kiểm thử chấp nhận bởi khách hàng hệ thống;

2.đã được giới thiệu về các kỹ thuật giúp bạn chọn các trường hợp kiểm thử nhằm phát hiện ra các lỗi của chương trình;

3.hiểu phát triển thử nghiệm đầu tiên, nơi bạn thiết kế các thử nghiệm trước khi viết mã và chạy các thử nghiệm này một cách tự động;

4.biết sự khác biệt quan trọng giữa thử nghiệm thành phần, hệ thống và bản phát hành cũng như nhận thức được các quy trình và kỹ thuật thử nghiệm người dùng,

Nội dung

8.1 Kiểm thử phát triển

8.2 Phát triển theo hướng thử nghiệm

8.3 Thử nghiệm phát hành

8.4 Thử nghiệm người dùng

Kiểm tra nhằm chỉ ra rằng một chương trình thực hiện những gì nó dự định làm và để phát hiện ra các lỗi của chương trình trước khi nó được đưa vào sử dụng. Khi bạn kiểm tra phần mềm, bạn thực thi một chương trình bằng dữ liệu nhân tạo. Bạn kiểm tra kết quả của quá trình chạy thử nghiệm để tìm lỗi, sự bất thường hoặc thông tin về các thuộc tính phi chức năng của chương trình.

Quá trình thử nghiệm có hai mục tiêu riêng biệt:

1. Để chứng minh với nhà phát triển và khách hàng rằng phần mềm đáp ứng các yêu cầu của nó. Đối với phần mềm tùy chỉnh, điều này có nghĩa là cần có ít nhất một bài kiểm tra cho mọi yêu cầu trong tài liệu yêu cầu. Đối với các sản phẩm phần mềm chung, điều đó có nghĩa là cần có các bài kiểm tra cho tất cả các tính năng của hệ thống, cộng với sự kết hợp của các tính năng này, sẽ được tích hợp trong bản phát hành sản phẩm

2.Để khám phá các tình huống trong đó hoạt động của phần mềm không chính xác, không mong muốn hoặc không phù hợp với đặc điểm kỹ thuật của nó. Đây là một hệ quả của các lỗi phần mềm. Kiểm tra lỗi có liên quan đến việc loại bỏ các hành vi không mong muốn của hệ thống như sự cố hệ thống, các tương tác không mong muốn với các hệ thống khác tính toán không chính xác và dữ liệu bị hỏng.

Mục tiêu đầu tiên dẫn đến thử nghiệm xác thực, trong đó bạn mong đợi hệ thống hoạt động chính xác bằng cách sử dụng một tập hợp các trường hợp thử nghiệm nhất định phản ánh mức sử dụng dự kiến của hệ thống. Mục tiêu thứ hai dẫn đến kiểm thử khuyết tật, trong đó các trường hợp kiểm thử được thiết kế để bộc lộ khuyết tật. Các trường hợp kiểm thử trong kiểm tra lỗi có thể được cố tình che khuất và không cần phản ánh cách hệ thống được sử dụng bình thường. Tất nhiên, không có ranh giới xác định giữa hai phương pháp thử nghiệm này. Trong quá trình kiểm tra xác nhận. bạn sẽ tìm thấy sự phòng thủ

Sơ đồ thể hiện trong Hình 8.1 có thể giúp giải thích sự khác biệt giữa kiểm tra xác nhận và kiểm tra khuyết tật. Hãy coi hệ thống đang được kiểm tra như một hộp đen. Hệ thống chấp nhận đầu vào từ một số tập hợp đầu vào I và tạo ra các đầu ra trong. Một tập hợp đầu ra O. Một số đầu ra sẽ bị sai. Đây là các đầu ra trong tập O) được tạo ra bởi hệ thống để đáp ứng với các đầu vào trong tập I. Ưu tiên trong kiểm tra khuyết tật là tìm các đầu vào đó trong tập I, vì chúng tiết lộ các vấn đề với th

Những điều này kích thích hệ thống tạo ra kết quả đầu ra đúng như mong đợi.

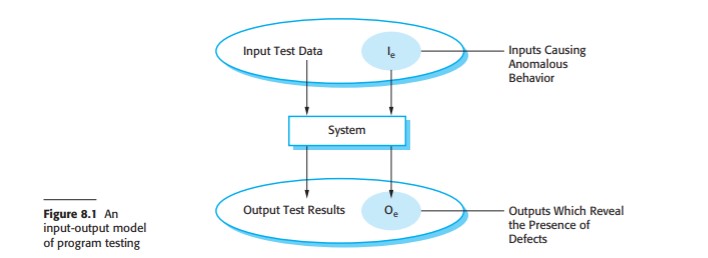
Kiểm tra không thể chứng minh rằng phần mềm không có lỗi hoặc nó sẽ hoạt động như được chỉ định trong mọi trường hợp. Luôn luôn có khả năng một bài kiểm tra mà bạn đã bỏ qua có thể phát hiện ra các vấn đề khác với hệ thống. Như Edsger Dijkstra, người đóng góp ban đầu cho sự phát triển của kỹ thuật phần mềm, hùng hồn

stated (Dijkstra et al., 1972):

Kiểm tra chỉ có thể cho thấy sự hiện diện của lỗi, không phải sự vắng mặt của chúng

Kiểm tra là một phần của quy trình xác minh và xác thực phần mềm (V&V) rộng hơn.

Xác minh và xác thực không giống nhau, chúng thường bị nhầm lẫn.



Barry Boehm, một nhà tiên phong của kỹ thuật phần mềm, đã diễn đạt ngắn gọn sự khác biệt giữa chúng (Bochm, 1979)

* Xác thực: Chúng ta có đang xây dựng sản phẩm phù hợp không?
* Xác minh: Chúng tôi đang xây dựng sản phẩm đúng không?

Quy trình xác minh và xác nhận có liên quan đến việc kiểm tra xem phần mềm đang được phát triển có đáp ứng đặc điểm kỹ thuật của nó và cung cấp chức năng mà những người trả tiền cho phần mềm mong đợi hay không. Các quá trình kiểm tra này bắt đầu ngay khi có các yêu cầu và tiếp tục qua tất cả các bước của quá trình phát triển.

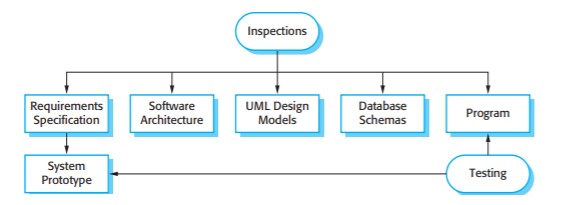
Mục đích của việc xác minh là để kiểm tra xem phần mềm có đáp ứng các yêu cầu chức năng và phi chức năng đã nêu hay không. Xác thực, tuy nhiên, là một quá trình tổng quát hơn. Mục đích của xác nhận là để đảm bảo rằng phần mềm đáp ứng mong đợi của khách hàng, Nó không chỉ đơn giản là kiểm tra sự phù hợp với đặc điểm kỹ thuật để chứng minh rằng phần mềm thực hiện những gì khách hàng mong đợi. Việc xác nhận là điều cần thiết bởi vì, như tôi đã thảo luận trong Chương 4, thông số kỹ thuật yêu cầu không phải lúc nào cũng ref

Mục tiêu cuối cùng của quá trình xác minh và xác nhận là thiết lập sự tin tưởng rằng hệ thống phần mềm là 'phù hợp với mục đích'. Điều này có nghĩa là hệ thống musi đủ tốt cho mục đích sử dụng của nó. Mức độ tin cậy cần thiết phụ thuộc vào mục đích của hệ thống, kỳ vọng của người sử dụng hệ thống và môi trường tiếp thị hiện tại cho hệ thống:

1. Mục đích của phần mềm Phần mềm càng quan trọng, thì điều quan trọng là nó càng đáng tin cậy. Ví dụ. mức độ tin cậy cần thiết đối với phần mềm được sử dụng để kiểm soát một hệ thống quan trọng về an toàn cao hơn nhiều so với mức độ tin cậy cần thiết cho một nguyên mẫu đã được phát triển để chứng minh các ý tưởng sản phẩm mới.

2. Kỳ vọng của người dùng Do trải nghiệm của họ với phần mềm lỗi, không đáng tin cậy, nhiều người dùng có kỳ vọng thấp về chất lượng phần mềm. Họ không ngạc nhiên khi phần mềm của họ bị lỗi. Khi một hệ thống mới được cài đặt, người dùng có thể chịu đựng

hỏng hóc vì lợi ích của việc sử dụng lớn hơn chi phí khắc phục sự cố.Trong những tình huống này, bạn có thể không cần dành nhiều thời gian để kiểm tra phần mềm. Tuy nhiên, khi phần mềm trưởng thành, người dùng mong đợi nó trở nên đáng tin cậy hơn nên có thể cần phải kiểm tra kỹ lưỡng hơn các phiên bản sau.



3.Tiếp thị eninommeni Khi một hệ thống được đưa ra thị trường, những người bán hệ thống đó phải tính đến các sản phẩm cạnh tranh, giá mà khách hàng sẵn sàng trả cho một hệ thống và lịch trình cần thiết để cung cấp hệ thống đó. Trong một môi trường cạnh tranh, một công ty phần mềm có thể quyết định phát hành một chương trình trước khi nó được kiểm tra và gỡ lỗi đầy đủ bởi vì họ muốn là người đầu tiên tham gia vào thị trường. nếu một sản phẩm phần mềm rất rẻ, người dùng có thể sẵn sàng chấp nhận mức độ an toàn thấp hơn

Cũng như kiểm thử phần mềm, quá trình xác minh và xác nhận có thể liên quan đến việc kiểm tra và đánh giá phần mềm, Kiểm tra và đánh giá phân tích và kiểm tra các yêu cầu hệ thống, mô hình thiết kế, mã nguồn chương trình và thậm chí cả các bài kiểm tra hệ thống được đề xuất. Đây được gọi là các kỹ thuật V & V 'tĩnh' mà bạn không cần phải thực thi phần mềm để xác minh nó. Hình 8.2 cho thấy việc kiểm tra và thử nghiệm phần mềm hỗ trợ V & V ở các giai đoạn khác nhau trong quy trình phần mềm. Các mũi tên chỉ ra sử dụng

Việc kiểm tra chủ yếu tập trung vào mã nguồn của hệ thống nhưng có thể kiểm tra bất kỳ phần trình bày nào có thể đọc được của phần mềm, chẳng hạn như các yêu cầu của nó hoặc một mô hình thiết kế. Khi bạn kiểm tra một hệ thống, bạn sử dụng kiến thức về hệ thống, miền ứng dụng của nó và ngôn ngữ lập trình hoặc mô hình hóa để phát hiện ra lỗi.

Có ba ưu điểm của việc kiểm tra phần mềm so với kiểm thử:

1.Trong quá trình thử nghiệm, các lỗi có thể che dấu (ẩn) các lỗi khác. Khi một lỗi dẫn đến kết quả đầu ra không mong muốn, bạn không bao giờ có thể chắc chắn liệu đầu ra bất thường sau này là do lỗi mới hay là tác dụng phụ của lỗi ban đầu. Bởi vì kiểm tra là một quá trình tĩnh, bạn không phải quan tâm đến sự tương tác giữa các lỗi. Do đó, một phiên kiểm tra đơn lẻ có thể phát hiện ra nhiều lỗi trong hệ thống.

Lập kế hoạch kiểm tra

Lập kế hoạch thử nghiệm được gắn liền với việc lập lịch và cung cấp nguồn lực cho tất cả các hoạt động trong quá trình thử nghiệm. Nó liên quan đến việc xác định quy trình thử nghiệm, có tính đến con người và thời gian có sẵn. Thông thường, một kế hoạch kiểm tra sẽ được tạo, trong đó xác định những gì sẽ được kiểm tra, lịch trình kiểm tra dự đoán và cách các bài kiểm tra sẽ được ghi lại. Đối với các hệ thống quan trọng, kế hoạch thử nghiệm cũng có thể bao gồm chi tiết của các thử nghiệm sẽ được chạy trên phần mềm.

<http://ww.SoftwareEngineering-9.com/Web/Testing/Planning.html>

2.Các phiên bản chưa hoàn chỉnh của hệ thống có thể được kiểm tra mà không phải trả thêm chi phí. Nếu một chương trình chưa hoàn thiện, thì bạn cần phát triển các bộ khai thác kiểm tra chuyên biệt để kiểm tra các bộ phận có sẵn. Điều này rõ ràng làm tăng thêm chi phí phát triển hệ thống.

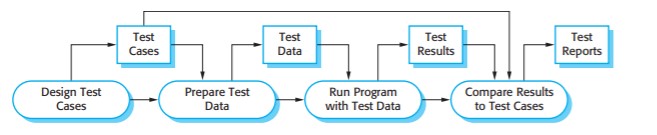
3. Cũng như tìm kiếm các khiếm khuyết của chương trình, việc kiểm tra cũng có thể xem xét các thuộc tính chất lượng rộng hơn của chương trình, chẳng hạn như sự tuân thủ các tiêu chuẩn, tính khả chuyển và khả năng bảo trì. Bạn có thể tìm kiếm sự kém hiệu quả, các thuật toán không phù hợp và phong cách lập trình kém có thể khiến hệ thống khó bảo trì và cập nhật.

Kiểm tra chương trình là một ý tưởng cũ và đã có một số nghiên cứu và thử nghiệm đã chứng minh rằng kiểm tra hiệu quả hơn để phát hiện ra khiếm khuyết hơn là kiểm tra chương trình. Fagan (1986) báo cáo rằng hơn 60% lỗi trong một chương trình có thể được phát hiện bằng cách sử dụng kiểm tra chương trình không chính thức. Trong quy trình Phòng sạch (Prowell và cộng sự, 1999), được khẳng định rằng hơn 90% các khiếm khuyết có thể được phát hiện trong quá trình kiểm tra chương trình.

Tuy nhiên, kiểm tra không thể thay thế kiểm tra phần mềm. Việc kiểm tra không tốt cho việc phát hiện ra những khiếm khuyết phát sinh do những tương tác bất ngờ giữa các bên.

không có các phần của một chương trình, các vấn đề về thời gian hoặc các vấn đề với hệ thống cho phép. Hơn nữa, đặc biệt là trong các công ty nhỏ hoặc các nhóm phát triển, việc lập một nhóm kiểm tra riêng có thể khó khăn và tốn kém vì tất cả các thành viên tiềm năng của nhóm cũng có thể là nhà phát triển phần mềm. Tôi thảo luận chi tiết hơn về việc xem xét và kiểm tra trong Chương 24 (Quản lý chất lượng). Phân tích tĩnh tự động, trong đó văn bản nguồn của chương trình được tự động phân tích để phát hiện ra các điểm bất thường, là giải thích ở chương 15

Hình 8.3 là một mô hình trừu tượng của quy trình kiểm thử 'truyền thống', như được sử dụng trong phát triển plandriven. Các trường hợp thử nghiệm là thông số kỹ thuật của đầu vào cho thử nghiệm và đầu ra dự kiến từ hệ thống (kết quả thử nghiệm), cộng với một tuyên bố về những gì đang được thử nghiệm. Dữ liệu kiểm tra là các đầu vào đã được tạo ra để kiểm tra hệ thống. Dữ liệu thử nghiệm đôi khi có thể được tạo tự động, nhưng việc tạo trường hợp thử nghiệm tự động là không thể.Vì những người hiểu hệ thống phải làm gì phải tham gia để chỉ định kết quả thử nghiệm mong đợi. Tuy nhiên, việc thực thi thử nghiệm có thể được tự động hóa. Kết quả mong đợi được tự động so sánh với kết quả dự đoán nên không cần người tìm kiếm lỗi và điểm bất thường trong quá trình chạy thử nghiệm



Hình 8.3

Thông thường, một hệ thống phần mềm thương mại phải trải qua ba giai đoạn thử nghiệm:

1. Kiểm thử phát triển. nơi hệ thống được kiểm tra trong quá trình phát triển để phát hiện ra các lỗi và khiếm khuyết. Các nhà thiết kế và lập trình hệ thống có khả năng tham gia vào quá trình thử nghiệm tbe.

2. Thử nghiệm phát hành, trong đó một nhóm thử nghiệm riêng biệt sẽ kiểm tra một phiên bản hoàn chỉnh của hệ thống trước khi nó được phát hành cho người dùng. Mục đích của thử nghiệm phát hành là để kiểm tra hệ thống đáp ứng các yêu cầu của các bên liên quan đến hệ thống.

3. Thử nghiệm người dùng. nơi người dùng hoặc người dùng tiềm năng của hệ thống kiểm tra hệ thống trong môi trường của chính họ. Đối với các sản phẩm phần mềm, 'người dùng' có thể là một thị trường liên tục.

nhóm ing người quyết định xem phần mềm có thể được tiếp thị, phát hành và bán hay không.

Thử nghiệm chấp nhận là một loại thử nghiệm đồng nhất trong đó khách hàng chính thức thử nghiệm một hệ thống để quyết định xem nó có nên được chấp nhận từ nhà cung cấp hệ thống hay không hoặc nếu cần phát triển thêm.

Trong thực tế, quá trình thử nghiệm thường lập hóa đơn kết hợp giữa thử nghiệm thủ công và tự động. Trong thử nghiệm thủ công. người thử nghiệm chạy chương trình với một số dữ liệu thử nghiệm và so sánh kết quả với mong đợi của họ. Họ ghi nhận và báo cáo các sai lệch cho các nhà phát triển chương trình. Trong kiểm thử tự động, các bài kiểm tra được mã hóa trong một chương trình được chạy mỗi khi hệ thống đang được phát triển được kiểm tra. Điều này thường nhanh hơn kiểm tra thủ công, đặc biệt khi nó liên quan đến kiểm tra hồi quy-chạy lại kiểm tra để xem các thay đổi đối với chương trình không có lỗi mới

Việc sử dụng thử nghiệm tự động đã tăng lên đáng kể trong vài năm qua.Tuy nhiên, kiểm thử không bao giờ có thể hoàn toàn tự động vì kiểm thử tự động chỉ có thể kiểm tra xem một chương trình có thực hiện những gì nó phải làm hay không. Trên thực tế, không thể sử dụng kiểm thử tự động để kiểm tra các hệ thống phụ thuộc vào giao diện của mọi thứ (ví dụ: giao diện người dùng đồ họa) hoặc để kiểm tra rằng một chương trình không có các tác dụng phụ không mong muốn.

8.1 Kiểm thử phát triển

Gỡ lỗi

Gỡ lỗi là quá trình tìm kiếm các lỗi và sự cố đã được phát hiện bằng cách thử nghiệm. Sử dụng thông tin từ các bài kiểm tra chương trình, người gỡ lỗi sử dụng kiến thức của họ về ngôn ngữ lập trình và kết quả dự kiến của bài kiểm tra để tìm và sửa lỗi chương trình. Quá trình này thường được hỗ trợ bởi các công cụ gỡ lỗi tương tác cung cấp thêm thông tin về việc thực thi chương trình.

<http://www.SoftwareEngineering-9.com/web/Testing/Debugeinghtml>

Kiểm thử phát triển bao gồm tất cả các hoạt động kiểm thử được thực hiện bởi nhóm phát triển hệ thống. Người kiểm tra phần mềm thường là lập trình viên đã phát triển phần mềm đó. mặc dù điều này không phải luôn luôn như vậy. Một số quy trình phát triển sử dụng cặp lập trình viên / người kiểm tra (Cusamano và Selby, 1998) trong đó mỗi

lập trình viên có một người kiểm tra liên quan, người phát triển các bài kiểm tra và hỗ trợ quá trình kiểm tra. Đối với các hệ thống quan trọng, một quy trình chính thức hơn có thể được sử dụng, với một nhóm thử nghiệm riêng biệt trong nhóm phát triển. Họ chịu trách nhiệm phát triển các bài kiểm tra và duy trì hồ sơ chi tiết về kết quả kiểm tra

Trong quá trình phát triển, thử nghiệm có thể được thực hiện ở ba cấp độ chi tiết:

1.Kiểm thử đơn vị, trong đó các đơn vị chương trình riêng lẻ hoặc các lớp đối tượng được kiểm tra. Kiểm thử đơn vị nên tập trung vào kiểm tra chức năng của các đối tượng hoặc phương pháp.

2.Kiểm tra thành phần. trong đó một số đơn vị riêng lẻ được tích hợp để tạo ra các thành phần tổng hợp. Kiểm thử thành phần nên tập trung vào kiểm tra các giao diện thành phần.

3. Kiểm thử hệ thống, trong đó một số hoặc tất cả các thành phần trong hệ thống được tích hợp và toàn bộ hệ thống được kiểm tra. Kiểm thử hệ thống nên tập trung vào kiểm tra các tương tác thành phần.

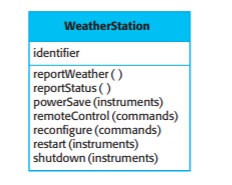
Kiểm thử Developmient chủ yếu là một quá trình kiểm tra lỗi, trong đó mục đích của kiểm thử là phát hiện ra các lỗi trong phần mềm. Do đó, nó thường được xen kẽ với việc gỡ lỗi quá trình xử lý các sự cố định vị với mã và thay đổi chương trình để khắc phục những sự cố này.

8.1.1 Kiểm thử đơn vị

Kiểm thử đơn vị là quá trình kiểm tra các thành phần của chương trình, chẳng hạn như các phương thức hoặc lớp đối tượng. Các hàm hoặc phương thức riêng lẻ là loại thành phần đơn giản nhất. Các bài kiểm tra của bạn phải là các cuộc gọi đến các quy trình này với các tham số đầu vào khác nhau. Bạn có thể sử dụng các cách tiếp cận để thiết kế trường hợp thử nghiệm được thảo luận trong Phần 8.1.2. để thiết kế các bài kiểm tra chức năng hoặc phương pháp

Khi bạn đang kiểm tra các lớp đối tượng, bạn nên thiết kế các bài kiểm tra của mình để cung cấp toàn bộ các tính năng của đối tượng. Điều này có nghĩa là bạn nên:

* kiểm tra tất cả các hoạt động liên kết với đối tượng;
* đặt và kiểm tra giá trị của tất cả các thuộc tính được liên kết với đối tượng
* đưa đối tượng vào tất cả các trạng thái có thể. Điều này có nghĩa là bạn nên mô phỏng tất cả các sự kiện gây ra thay đổi trạng thái



Ví dụ, hãy xem xét đối tượng trạm thời tiết từ ví dụ mà tôi đã thảo luận trong Chương 7. Giao diện của đối tượng này được thể hiện trong Hình 8.4. Nó có một thuộc tính duy nhất.

là định danh của nó. Đây là hằng số được đặt khi trạm thời tiết được cài đặt.

Do đó, bạn chỉ cần một bài kiểm tra để kiểm tra xem nó đã được thiết lập đúng cách chưa. Bạn cần xác định các trường hợp thử nghiệm cho tất cả các phương thức được liên kết với đối tượng như reportWeather, reportStatus, v.v. Tốt nhất, bạn nên thử nghiệm các phương pháp một cách riêng biệt, nhưng trong một số trường hợp, một số trình tự thử nghiệm là cần thiết, chẳng hạn như để kiểm tra phương pháp tắt các công cụ của trạm thời tiết (tắt máy), bạn cần phải thực hiện phương pháp khởi động lại.

Tổng quát hóa hoặc kế thừa làm cho việc kiểm tra lớp đối tượng trở nên phức tạp hơn. Bạn không thể đơn giản kiểm tra một thao tác trong lớp mà nó được định nghĩa và giả định rằng nó sẽ hoạt động như mong đợi trong các lớp con kế thừa thao tác đó. Hoạt động được kế thừa có thể tạo ra các giả định về các hoạt động và thuộc tính khác. Chúng có thể không hợp lệ trong một số lớp con kế thừa hoạt động. Do đó, bạn phải kiểm tra hoạt động kế thừa trong tất cả các ngữ cảnh nơi nó được sử dụng.

Để kiểm tra các trạng thái của trạm thời tiết, bạn sử dụng mô hình trạng thái, chẳng hạn như mô hình hiển thị trong Hình 7.8 trong chương trước. Sử dụng mô hình này, bạn có thể xác định các chuỗi chuyển đổi trạng thái phải được kiểm tra và xác định các chuỗi sự kiện để buộc các chuyển đổi này. Trong nguyên tắc, bạn nên kiểm tra mọi trình tự chuyển đổi trạng thái có thể có, mặc dù trong thực tế, điều này có thể quá đắt. Ví dụ về trình tự trạng thái cần được kiểm tra trong trạm thời tiết bao gồm:

**Shutdown→ Running → Shutdown**

**Configuring → Running → Testing → Transmitting → Running**

**Running → Collecting → Running → Summarizing → Transmitting → Running**

Bất cứ khi nào có thể, bạn nên tự động thử nghiệm đơn vị. Trong thử nghiệm đơn vị tự động.bạn sử dụng khuôn khổ tự động hóa thử nghiệm (chẳng hạn như JUnit) để viết và chạy các thử nghiệm chương trình của mình. Các khuôn khổ kiểm thử đơn vị cung cấp các lớp kiểm thử chung mà bạn mở rộng để tạo các trường hợp kiểm thử cụ thể. Sau đó, họ có thể chạy tất cả các thử nghiệm mà bạn đã triển khai và báo cáo, thường là thông qua một số GUL, về sự thành công hay thất bại của các thử nghiệm. Toàn bộ bộ thử nghiệm thường có thể hoàn thành trong vài giây, do đó bạn có thể thực hiện tất cả các thử nghiệm mỗi khi bạn thực hiện thay đổi đối với chương trình.

Kiểm tra tự động có ba phần:

1. Phần thiết lập, nơi bạn khởi tạo hệ thống với trường hợp thử nghiệm, cụ thể là các đầu vào và đầu ra mong đợi

2. Một phần cuộc gọi, nơi bạn gọi đối tượng hoặc phương thức được kiểm tra

3. Phần xác nhận nơi bạn so sánh kết quả của cuộc gọi với kết quả mong đợi. Nếu khẳng định đánh giá là đúng, thử nghiệm đã thành công; nếu sai.

sau đó nó đã thất bại

Đôi khi đối tượng mà bạn đang kiểm tra có sự phụ thuộc vào các đối tượng khác có thể chưa được viết hoặc làm chậm quá trình kiểm tra nếu chúng được sử dụng.

Ví dụ: nếu đối tượng của bạn gọi một cơ sở dữ liệu, điều này có thể liên quan đến quá trình thiết lập chậm trước khi nó có thể được sử dụng. Trong những trường hợp này, bạn có thể quyết định sử dụng các đối tượng giả. Đối tượng giả là các đối tượng có giao diện giống với các đối tượng bên ngoài đang được sử dụng để mô phỏng chức năng của nó. Do đó, một obiect giả mô phỏng một cơ sở dữ liệu có thể chỉ có một vài mục dữ liệu được tổ chức trong một mảng. Do đó, chúng có thể được truy cập nhanh chóng mà không cần phải gọi cơ sở dữ liệu và truy cập đĩa. Tương tự các đối tượng giả có thể được sử dụng để mô phỏng hoạt động bất thường hoặc các sự kiện hiếm. Ví dụ: nếu hệ thống của bạn dự định thực hiện hành động vào những thời điểm nhất định trong ngày, đối tượng mô phỏng của bạn có thể chỉ cần trả lại những thời điểm đó, bất kể thời gian thực tế của đồng hồ.

8.1.2 Chọn các trường hợp thử nghiệm đơn vị

Việc kiểm tra rất tốn kém và tốn thời gian, vì vậy điều quan trọng là bạn phải chọn các trường hợp kiểm thử đơn vị hiệu quả. Trong trường hợp này, hiệu quả có nghĩa là hai điều:

1. Các trường hợp thử nghiệm phải cho thấy rằng, khi được sử dụng như mong đợi, thành phần mà bạn đang thử nghiệm thực hiện những gì nó phải làm

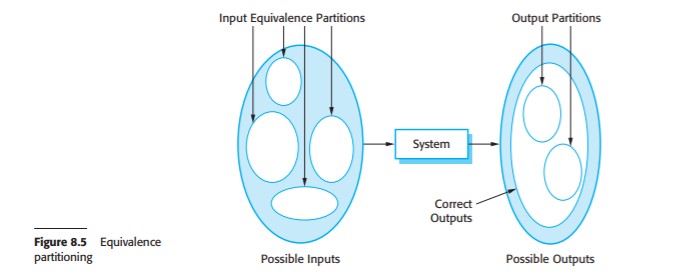
2. Nếu có các lỗi trong thành phần, chúng sẽ được tiết lộ bởi các trường hợp thử nghiệm.

Do đó, bạn nên viết hai loại trường hợp thử nghiệm. Đầu tiên trong số này phải phản ánh hoạt động bình thường của một chương trình và phải cho thấy rằng thành phần hoạt động. Ví dụ: nếu bạn đang kiểm tra một thành phần tạo và khởi tạo bản ghi bệnh nhân mới, thì trường hợp thử nghiệm của bạn sẽ cho thấy rằng bản ghi tồn tại trong cơ sở dữ liệu và các trường của nó đã được đặt như đã chỉ định. Các loại trường hợp thử nghiệm khác nên dựa trên kinh nghiệm thử nghiệm về nơi phát sinh các vấn đề chung. Nó sẽ sử dụng các đầu vào bất thường để

Tôi thảo luận về hai chiến lược khả thi ở đây có thể hiệu quả trong việc giúp bạn chọn các trường hợp thử nghiệm. Đó là:

1 . Kiểm tra phân vùng, nơi bạn xác định các nhóm đầu vào có đặc điểm chung và nên được xử lý theo cùng một cách. Bạn nên chọn các bài kiểm tra từ bên trong các nhóm này.

2. Kiểm thử dựa trên hướng dẫn, nơi bạn sử dụng hướng dẫn kiểm thử để chọn các trường hợp kiểm thử.Các nguyên tắc này phản ánh kinh nghiệm trước đây về các loại lỗi mà lập trình viên thường mắc phải khi phát triển các thành phần



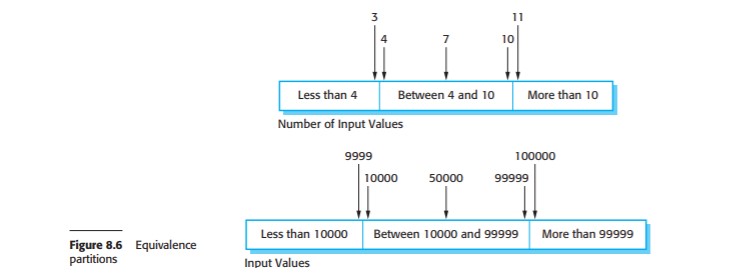
Dữ liệu đầu vào và kết quả đầu ra của một chương trình thường thuộc một số lớp khác nhau với các đặc điểm chung. Ví dụ về các lớp này là số dương, số âm và lựa chọn menu. Các chương trình thường hoạt động theo cách có thể so sánh được cho tất cả các thành viên của một lớp. Nghĩa là, nếu bạn kiểm tra một chương trình thực hiện tính toán và yêu cầu hai số dương, thì bạn sẽ mong đợi chương trình hoạt động theo cùng một cách cho tất cả các số dương.

Do hành vi tương đương này, các lớp này đôi khi được gọi là phân vùng hoặc miền tương đương (Bezier, 1990). Một cách tiếp cận có hệ thống để thiết kế trường hợp kiểm thử dựa trên việc xác định tất cả các phân vùng đầu vào và đầu ra cho một hệ thống hoặc thành phần. Các trường hợp kiểm thử được thiết kế để các đầu vào hoặc đầu ra lic trong các phân vùng này.Kiểm thử phân vùng có thể được sử dụng để thiết kế các trường hợp kiểm thử cho cả hệ thống và thành phần.

Trong Hình 8.5, hình elip tô bóng lớn ở bên trái đại diện cho tập hợp tất cả các đầu vào có thể có cho chương trình đang được kiểm tra. Các hình elip nhỏ hơn không được đánh bóng đại diện cho các phân vùng tương đương. Một chương trình đang được kiểm tra phải xử lý tất cả các thành viên của một phân vùng tương đương đầu vào theo cùng một cách. Phân vùng tương đương đầu ra là phân vùng mà trong đó tất cả các đầu ra có điểm chung. Đôi khi có một ánh xạ 1: 1 giữa các phân vùng tương đương đầu vào và đầu ra. Tuy nhiên.

điều này không phải lúc nào cũng đúng: bạn có thể cần xác định một phân vùng tương đương đầu vào riêng biệt, trong đó đặc điểm chung duy nhất của các đầu vào là chúng tạo ra các đầu ra trong cùng một phân vùng đầu ra. Vùng được tô bóng trong hình elip bên trái đại diện cho các đầu vào không hợp lệ. Khu vực bóng mờ trong hình elip bên phải đại diện cho các trường hợp ngoại lệ có thể xảy ra (tức là phản hồi với các đầu vào không hợp lệ)

Khi bạn đã xác định được một tập hợp các phân vùng, bạn chọn các trường hợp thử nghiệm từ mỗi phân vùng này. Một nguyên tắc nhỏ để lựa chọn trường hợp thử nghiệm là chọn các trường hợp thử nghiệm trên ranh giới của các phân vùng, cộng với các trường hợp gần với điểm giữa của phân vùng.Lý do cho điều này là các nhà thiết kế và lập trình có xu hướng xem xét các giá trị điển hình của đầu vào khi phát triển một hệ thống. Bạn kiểm tra chúng bằng cách chọn điểm giữa của phân vùng. Các giá trị ranh giới thường không điển hình (ví dụ: số 0 có thể hoạt động khác với các số không âm khác) nên đôi khi bị các nhà phát triển bỏ qua. Lỗi chương trình thường xảy ra khi xử lý các giá trị không điển hình này.



Bạn xác định các phân vùng bằng cách sử dụng đặc tả chương trình hoặc tài liệu người dùng và từ kinh nghiệm mà bạn dự đoán các lớp giá trị đầu vào có khả năng phát hiện lỗi. Ví dụ: giả sử một đặc tả chương trình nói rằng chương trình chấp nhận 4 đến 8 đầu vào là các số nguyên năm chữ số lớn hơn 10.000. Bạn sử dụng thông tin này để xác định các phân vùng đầu vào và các giá trị đầu vào kiểm tra có thể có. Chúng được thể hiện trong Hình 8.6.

Khi bạn sử dụng đặc điểm kỹ thuật của hệ thống để xác định các phân vùng tương đương, đây được gọi là "kiểm tra hộp đen". Ở đây, bạn không cần bất kỳ kiến thức nào về cách hệ thống hoạt động. Tuy nhiên, việc bổ sung hộp đen có thể hữu ích kiểm tra với kiểm tra hộp trắng '. nơi bạn xem mã của chương trình để tìm các kiểm tra khả thi khác. Ví dụ: mã của bạn có thể bao gồm các ngoại lệ để xử lý các đầu vào không hợp lệ. Bạn có thể sử dụng kiến thức này để xác định các' phân vùng ngoại lệ '-các phạm vi khác nhau khi giống nhau e

Phân vùng tương đương là một cách tiếp cận hiệu quả để kiểm tra vì nó giúp giải thích các lỗi mà các lập trình viên thường mắc phải khi xử lý đầu vào ở các cạnh của phân vùng. Bạn cũng có thể sử dụng hướng dẫn thử nghiệm để giúp chọn các trường hợp thử nghiệm.

Hướng dẫn gói gọn kiến thức về những loại trường hợp kiểm thử nào hiệu quả để phát hiện lỗi. Ví dụ, khi bạn đang thử nghiệm các chương trình có chuỗi, mảng hoặc các danh sách, nguyên tắc có thể giúp tiết lộ các khiếm khuyết bao gồm:

1. Kiểm tra trình tự waidh soflware chỉ có một giá trị duy nhất. Các lập trình viên tự nhiên nghĩ về các sersences được tạo thành từ một số giá trị và đôi khi họ nhúng giả định này vào chương trình của họ. Do đó, nếu được trình bày với một chuỗi giá trị đơn, chương trình có thể không hoạt động bình thường.

2. Sử dụng các chuỗi khác nhau có kích thước khác nhau trong các thử nghiệm khác nhau. Điều này làm giảm khả năng một chương trình bị lỗi sẽ vô tình tạo ra một đầu ra chính xác do một số đặc điểm tình cờ của đầu vào.

3. Xuất phát các bài kiểm tra sao cho các phần tử đầu tiên, giữa và cuối cùng của dãy được tính. Cách tiếp cận này cho thấy các vấn đề ở ranh giới phân vùng

Kiểm tra đường dẫn

Kiểm thử đường dẫn là một chiến lược kiểm tra nhằm mục đích thực hiện mọi đường dẫn thực thi độc lập thông qua một thành phần hoặc chương trình. Tôi mọi đường dẫn độc lập được thực thi, thì tất cả các bậc trong thành phần phải được thực thi ít nhất một lần. Tất cả các câu lệnh có điều kiện đều được kiểm tra cho cả trường hợp đúng và sai, Trong quá trình phát triển hướng đối tượng, kiểm tra đường dẫn có thể được sử dụng khi kiểm tra các phương pháp liên kết với các đối tượng

<http://www.SoftwareEngineering-9.com/Web/Testing/PathTest.html>

Cuốn sách của Whittaker (2002) bao gồm nhiều ví dụ về các hướng dẫn có thể được sử dụng trong thiết kế trường hợp thử nghiệm. Một số hướng dẫn chung nhất mà ông đề xuất là

* Chọn các đầu vào buộc hệ thống tạo ra tất cả các thông báo lỗi;
* Thiết kế đầu vào gây tràn bộ đệm đầu vào;
* Lặp lại cùng một đầu vào hoặc một loạt đầu vào nhiều lần;
* Buộc tạo ra các đầu ra không hợp lệ;
* Buộc các kết quả tính toán quá lớn hoặc quá nhỏ.

Khi bạn có kinh nghiệm với thử nghiệm. bạn có thể phát triển các hướng dẫn của riêng mình về cách chọn các trường hợp kiểm thử hiệu quả. Tôi đưa ra nhiều ví dụ hơn về hướng dẫn kiểm tra trong phần tiếp theo của chương này.

**8.1.3 Kiểm tra thành phần**

Các thành phần phần mềm thường là các thành phần tổng hợp được tạo thành từ một số đối tượng tương tác. Ví dụ, trong hệ thống trạm thời tiết, thành phần cấu hình lại bao gồm các đối tượng xử lý từng khía cạnh của cấu hình lại. Bạn truy cập chức năng của các đối tượng này thông qua giao diện thành phần đã xác định.

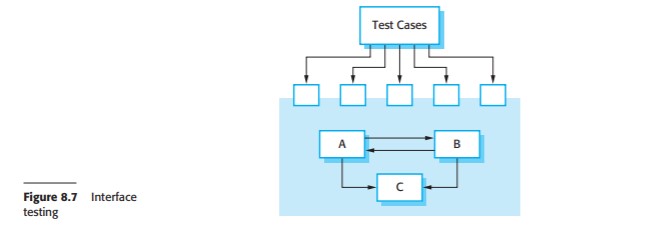
Do đó, kiểm tra các thành phần hỗn hợp nên tập trung vào việc chỉ ra rằng giao diện thành phần hoạt động theo đặc điểm kỹ thuật của nó. Bạn có thể giả định rằng các bài kiểm tra đơn vị trên các đối tượng riêng lẻ trong thành phần đã được hoàn thành.

Hình 8.7 minh họa ý tưởng về kiểm thử giao diện thành phần. Giả sử rằng các thành phần AB và C hiye đã được tích hợp để tạo ra một thành phần hoặc hệ thống con lớn hơn.

Các trường hợp thử nghiệm không được áp dụng cho các thành phần riêng lẻ mà là cho giao diện của thành phần hỗn hợp được tạo ra bằng cách kết hợp các thành phần này. Các lỗi giao diện trong thành phần tổng hợp có thể không phát hiện được bằng cách kiểm tra các đối tượng riêng lẻ vì những lỗi này là kết quả của sự tương tác giữa các đối tượng trong thành phần.

Có nhiều loại giao diện khác nhau giữa các thành phần chương trình và, chắc chắn, các loại lỗi giao diện khác nhau có thể xảy ra?

1. Giao diện Purameter Đây là những giao diện trong đó dữ liệu hoặc đôi khi các tham chiếu chức năng được truyền từ thành phần này sang thành phần khác. Các phương thức trong một đối tượng có một giao diện tham số.



2. Các giao diện ghi nhớ dùng chung Đây là các giao diện trong đó một khối bộ nhớ được chia sẻ giữa các thành phần. Dữ liệu được đặt vào bộ nhớ bởi một hệ thống con và được các hệ thống con khác truy xuất từ đó. Loại giao diện này thường được sử dụng trong các hệ thống nhúng, nơi các cảm biến tạo ra dữ liệu được các thành phần hệ thống khác truy xuất và xử lý.

3. Các giao diện thủ tục Đây là các giao diện trong đó một thành phần đóng gói một tập các thủ tục có thể được gọi bởi các thành phần khác. Các đối tượng và các thành phần có thể tái sử dụng có dạng giao diện này.

4.Các giao diện truyền thông điệp Đây là các giao diện trong đó một thành phần yêu cầu một dịch vụ từ một thành phần khác bằng cách chuyển một thông điệp đến nó. Thông báo retum bao gồm kết quả của việc thực thi dịch vụ. Một số hệ thống hướng đối tượng có dạng giao diện này, cũng như các hệ thống máy khách-máy chủ.

Lỗi giao diện là một trong những dạng lỗi phổ biến nhất trong các hệ thống phức tạp (Lutz, 1993). Các lỗi này thuộc ba lớp:

* Lạm dụng giao diện Một thành phần gọi gọi một số thành phần khác và tạo ra lỗi trong giao diện usTof của nó. Loại lỗi này thường gặp với inter tham số.

mặt, trong đó các tham số có thể không đúng loại hoặc được truyền không đúng thứ tự hoặc có thể truyền sai số lượng tham số.

* Sự hiểu lầm về giao diện Một thành phần đang gọi hiểu sai đặc điểm kỹ thuật của giao diện của thành phần được gọi và đưa ra giả định về hành vi của nó. Thành phần được gọi không hoạt động như mong đợi, sau đó gây ra hành vi không mong muốn trong thành phần đang gọi. Ví dụ, một phương thức tìm kiếm nhị phân có thể được gọi với một tham số là một mảng không có thứ tự. Tìm kiếm sau đó sẽ thất bại.
* Lỗi thời gian Những lỗi này xảy ra trong các hệ thống thời gian thực sử dụng bộ nhớ dùng chung hoặc giao diện truyền thông báo. Người sản xuất dữ liệu và người tiêu dùng dữ liệu có thể hoạt động ở tốc độ khác nhau. Trừ khi được chăm sóc đặc biệt trong thiết kế giao diện người tiêu dùng có thể truy cập thông tin lỗi thời vì nhà sản xuất thông tin chưa cập nhật thông tin giao diện chia sẻ.

Việc kiểm tra các lỗi giao diện rất khó vì lỗi giao diện sone chỉ có thể tự biểu hiện trong các điều kiện bất thường. Ví dụ, giả sử một đối tượng triển khai hàng đợi như một cấu trúc dữ liệu có độ dài cố định. Một đối tượng đang gọi có thể giả định rằng qucue được thực hiện như một cấu trúc dữ liệu vô hạn và có thể không kiểm tra hàng đợi trên luồng khi một mục được nhập. Điều kiện này chỉ có thể được phát hiện trong quá trình thử nghiệm bằng cách thiết kế các trường hợp thử nghiệm buộc hàng đợi bị tràn và gây ra sự tràn đó.

phá vỡ hành vi của đối tượng theo một số cách có thể phát hiện được.

Một vấn đề khác có thể phát sinh do sự tương tác giữa các lỗi trong các mô-đun hoặc đối tượng khác nhau. Lỗi trong một đối tượng chỉ có thể được phát hiện khi một số đối tượng khác hoạt động theo cách không mong muốn. Ví dụ, một đối tượng có thể gọi một đối tượng khác để nhận một số dịch vụ và giả định rằng phản hồi là đúng. Nếu dịch vụ được gọi bị lỗi theo một cách nào đó, yalue được khôi phục có thể hợp lệ nhưng không chính xác. Điều này không được phát hiện ngay lập tức mà chỉ trở nên rõ ràng khi một số tính toán sau đó bị sai

Một số hướng dẫn chung cho kiểm tra giao diện là:

1. Kiểm tra mã được kiểm tra liệt kê rõ ràng từng lệnh gọi đến một thành phần ngoại vi. Thiết kế một tập hợp các thử nghiệm trong đó giá trị của các tham số đối với các thành phần ngoại nhiệt nằm ở đầu cực của phạm vi của chúng. Các giá trị cực đoan này có nhiều khả năng tiết lộ sự không nhất quán về giao diện.

2. Khi các con trỏ đi qua một giao diện, hãy luôn kiểm tra giao diện với các tham số con trỏ rỗng.

3.Nơi một thành phần được gọi thông qua một giao diện thủ tục. kiểm tra thiết kế mà cố tình làm cho thành phần bị hư hỏng. Các giả định về thất bại khác nhau là một trong những hiểu lầm về đặc điểm kỹ thuật phổ biến nhất.

4. Sử dụng kiểm tra căng thẳng trong hệ thống truyền thông báo. Điều này có nghĩa là bạn nên thiết kế các bài kiểm tra để tạo ra nhiều thông báo hơn khả năng xảy ra trong thực tế. Đây là một cách hữu hiệu để tiết lộ các vấn đề về thời gian

5.Nơi một số thành phần tương tác thông qua bộ nhớ được chia sẻ. thiết kế các bài kiểm tra thay đổi thứ tự mà các thành phần này được kích hoạt. Các thử nghiệm này có thể tiết lộ các giả định ngầm được đưa ra bởi lập trình viên về thứ tự sản xuất và tiêu thụ dữ liệu được chia sẻ

Việc kiểm tra và đánh giá đôi khi có thể hiệu quả hơn về chi phí so với việc kiểm tra để phát hiện lỗi giao diện. Việc kiểm tra có thể tập trung vào các giao diện thành phần và các câu hỏi về hành vi giao diện giả định được hỏi trong quá trình kiểm tra. Một ngôn ngữ được đánh máy mạnh như Java cho phép trình biên dịch có nhiều lỗi giao diện. Máy phân tích tĩnh (chương 15) có thể phát hiện một loạt các lỗi giao diện.