TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN
----------o0o---------

Thạc Bình Cường

Bài giảng điện tử môn học

KIỂM THỬ VÀ BẢO ĐẢM CHẤT LƯỢNG PHẦN MỀM

MỞ ĐẦU

Quản lý chất lượng phần mềm là vấn đề không mới nhưng theo một số đánh giá là còn yếu của các công ty phần mềm Việt Nam. Một số công ty trong nước hiện đã đạt các chuẩn quốc tế CMM/CMMI trong nâng cao năng lực và quản lý chất lượng phần mềm, song chỉ đếm được trên đầu ngón tay, và hiện cũng chỉ gói gọn trong vài công ty gia công cho thị trường nước ngoài.

Lâu nay, nói đến chất lượng phần mềm, không ít người nghĩ ngay đến vấn đề là xác định xem phần mềm đó có phát sinh lỗi hay không, có "chạy" đúng như yêu cầu hay không và cuối cùng thường quy về vai trò của hoạt động kiểm thử phần mềm (testing) như là hoạt động chịu trách nhiệm chính.

Với quan điểm của khách hàng, điều này có thể đúng, họ không cần quan tâm nội tình của hoạt động phát triển phần mềm, điều họ cần quan tâm là liệu sản phẩm cuối cùng giao cho họ có đúng hạn hay không và làm việc đúng như họ muốn hay không.

Tuy nhiên theo quan điểm của người phát triển phần mềm, thực tế cho thấy hoạt động kiểm thử phần mềm là quan trọng, nhưng không đủ để đảm bảo sản phẩm sẽ được hoàn thành đúng hạn và đúng yêu cầu. Kiểm thử sau cùng để phát hiện lỗi là điều tất nhiên phải làm, nhưng trong rất nhiều trường hợp, điều đó thường quá trễ và sẽ phải mất rất nhiều thời gian để sửa chữa.

Thực tế cho thấy, để đảm bảo được hai tiêu chí "đơn giản" trên của khách hàng, đòi hỏi tổ chức không chỉ vận hành tốt khâu kiểm thử phần mềm, mà phải tổ chức và duy trì sự hoạt động nhịp nhàng của cả một hệ thống các công việc liên quan đến một dự án phần mềm, từ đây xuất hiện một khái niệm có tên là "hệ thống quản lý chất lượng phần mềm" bao gồm các quy trình được thực thi xuyên suốt chu kỳ phát triển của dự án phần mềm song hành cùng việc kiểm thử phân mềm nhằm đảm bảo chất lượng cho phần mềm khi chuyển giao cho khách hàng.

Với thực tế trên, là những người làm công tác đào tạo mong muốn cung cấp cho sinh viên ngành công nghệ phần mềm - những người sẽ là nguồn nhân lực chủ yếu trong tương lai của các doanh nghiệp phần mềm – những khái niệm, kiến thức và kỹ năng cơ bản ban đầu về kiểm thử phần mềm, về qui trình quản lý chất lượng, đảm bảo chất lượng phần mềm thông qua giáo trình (nội bộ) Kiểm thử và đảm bảo chất lượng phần mềm (Software Testing and Quality Assurrance).

Giáo trình này với mục tiêu cung cấp cho sinh viên công nghệ phần mềm có được kiến thức và kỹ năng về việc kiểm thử phần mềm, các công đoạn kiểm thử, các loại kiểm thử, công cụ kiểm thử, xây dựng tài liệu kiểm thử, dữ liệu kiểm thử …. Và xây qui trình đảm bảo chất lượng phần mềm, giới thiệu tổng quan về hệ thống quản lý chất lượng, nguyên tắc, kỹ thuật … để đảm bảo rằng dự án phần mềm sẽ chuyển giao cho khách hàng đúng hạn, đúng yêu cầu.

Đây là giáo trình sơ khởi, còn nhiều vấn đề chưa đi sâu phân tích và thực hiện, còn mang tính lý thuyết nhiều. Tác giả hy vọng bạn đọc đóng góp ý kiến để phiên bản 2 đáp ứng tốt hơn yêu cầu của nhiều độc giả, của sinh viên và kể cả những người đang công tác tại các phòng phát triển và đảm bảo chất lượng phần mềm.

CHƯƠNG 1: CÁC KHÁI NIỆM

1.1. Các định nghĩa

"Lỗi phần mềm là chuyện hiển nhiên của cuộc sống. Chúng ta dù cố gắng đến mức nào thì thực tế là ngay cả những lập trình viên xuất sắc nhất cũng không có thể lúc nào cũng viết được những đoạn mã không có lỗi. Tính trung bình, ngay cả một lập trình viên loại tốt thì cũng có từ 1 đến 3 lỗi trên 100 dòng lệnh. Người ta ước lượng rằng việc kiểm tra để tìm ra các lỗi này chiếm phân nửa khối lượng công việc phải làm để có được một phần mềm hoạt động được". *(Software Testing Techniques, Second Edition, by Boris Beizer, Van Nostrand Reinhold, 1990, ISBN 1850328803).*

Trên đây là một nhận định về công việc kiểm nghiệm (testing) chương trình.

Thật vậy, ngày nay càng ngày các chương trình (các phần mềm) càng trở lên phức tạp và đồ sộ. Việc tạo ra một sản phẩm có thể bán được trên thị trường đòi hỏi sự nổ lực của hàng chục, hàng trăm thậm chí hàng ngàn nhân viên. Số lượng dòng mã lên đến hàng triệu. Và để tạo ra một sản phẩm thì không phải chỉ do một tổ chức đứng ra làm từ đầu đến cuối, mà đòi hỏi sự liên kết, tích hợp của rất nhiều sản phẩm, thư viện lập trình, … của nhiều tổ chức khác nhau… Từ đó đòi hỏi việc kiểm nghiệm phần mềm càng ngày càng trở nên rất quan trọng và rất phức tạp.

Song song với sự phát triển các công nghệ lập trình, các ngôn ngữ lập trình… thì các công nghệ và kỹ thuật kiểm nghiệm phần mềm ngày càng phát triển và mang tính khoa học. Bài tiểu luận này với mục đích là tập hợp, nghiên cứu, phân tích các kỹ thuật, các công nghệ kiểm nghiệm phần mềm đang được sử dụng và phát triển hiện nay.

1.1.1. Định nghĩa:

Việc kiểm nghiệm là quá trình thực thi một chương trình với mục đích là tìm ra lỗi. (Glen Myers)

Giải thích theo mục đích:

Việc thử nghiệm hiển nhiên là nói đến các lỗi (error), sai sót (fault), hỏng hóc (failure) hoặc các hậu quả (incident). Một phép thử là một cách chạy phần mềm theo các trường hợp thử nghiệm với mục tiêu là:

- Tìm ra sai sót.
- Giải thích sự hoạt động chính xác.

  (Paul Jorgensen)

1.1.2. Các thuật ngữ:

- Lỗi (Error):
  - Là các lỗi lầm do con người gây ra.
- Sai sót (Fault):
  - Sai sót gây ra lỗi. Có thể phân loại như sau:

- Sai sót do đưa ra dư thừa – chúng ta đưa một vài thứ không chính xác vào mô tả yêu cầu phần mềm.
- Sai sót do bỏ sót – Người thiết kế có thể gây ra sai sót do bỏ sót, kết quả là thiếu một số phần đáng ra phải có trong mô tả yêu cầu phần mềm.

- Hỏng hóc (Failure):
  - Xảy ra khi sai sót được thực thi. (Khi thực thi chương trình tại các nơi bị sai thì sẽ xảy ra trạng thái hỏng hóc).
- Kết quả không mong đợi, hậu quả (Incident)
  - Là những kết quả do sai sót đem đến. Hậu quả là các triệu chứng liên kết với một hỏng hóc và báo hiệu cho người dùng biết sự xuất hiện của hỏng hóc.
- Trường hợp thử (Test case)
  - Trường hợp thử được liên kết tương ứng với hoạt động của chương trình. Một trường hợp thử bao một một tập các giá trị đầu vào và một danh sách các kết quả đầu ra mong muốn.
- Thẩm tra (Verification)
  - Thẩm tra là tiến trình nhằm xác định đầu ra của một công đoạn trong việc phát triển phần mềm phù hợp với công đoạn trước đó.
- Xác nhận (Validation)
  - Xác nhận là tiến trình nhằm chỉ ra toàn hệ thống đã phát triển xong phù hợp với tài liệu mô tả yêu cầu.

So sánh giữa Thẩm tra và Xác nhận:

- Thẩm tra: thẩm tra quan tâm đến việc ngăn chặn lỗi giữa các công đoạn.
- Xác nhận: xác nhận quan tâm đến sản phẩm cuối cùng không còn lỗi.

1.2. Vòng đời của việc kiểm nghiệm (testing life cycle):

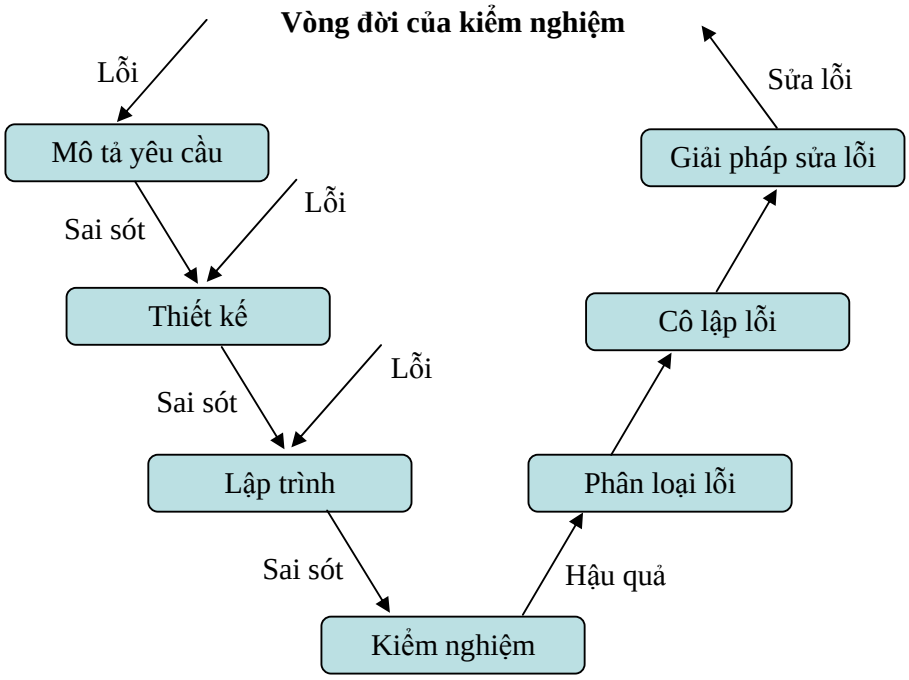
Bảng dưới đây mô tả các công đoạn phát triển một phần mềm và cách khắc phục lỗi.

Lỗi có thể xảy ra trong tất cả các công đoạn từ “Mô tả yêu cầu”, “Thiết kế” đến “Lập trình”.

Từ công đoạn này chuyển sang công đoạn khác thường nảy sinh các sai sót (do dư thừa hoặc thiếu theo mô tả yêu cầu).

Đến công đoạn kiểm nghiệm chúng ta sẽ phát hiện ra các hậu quả (các kết quả không mong muốn).

Quá trình sửa lỗi bao gồm “phân loại lỗi”, “cô lập lỗi” (tìm ra nguyên nhân và nơi gây lỗi), đề ra “giải pháp sửa lỗi” và cuối cùng là khắc phục lỗi.

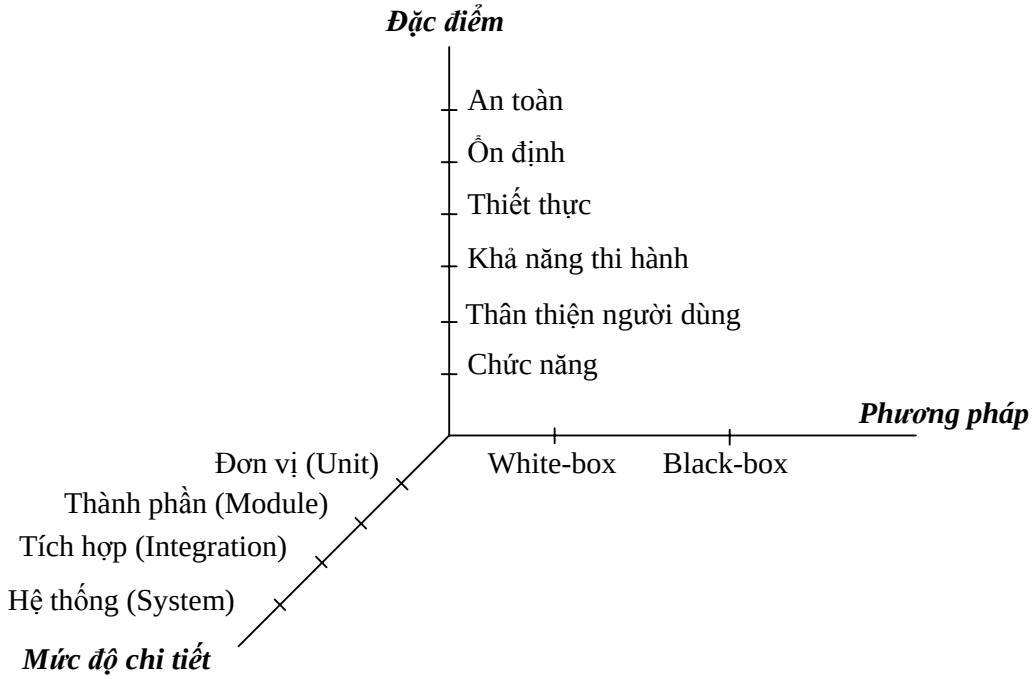


1.3. Phân loại kiểm nghiệm:

Có 2 mức phân loại:

- Một là phân biệt theo mức độ chi tiết của các bộ phận hợp thành phần mềm.
  - Mức kiểm tra đơn vị (Unit)
  - Mức kiểm tra hệ thống (System)
  - Mức kiểm tra tích hợp (Integration)
- Cách phân loại khác là dựa trên phương pháp thử nghiệm (thường dùng ở mức kiểm tra đơn vị)
  - Kiểm nghiệm hộp đen (Black box testing) dùng để kiểm tra chức năng.
  - Kiểm nghiệm hộp trắng (White box testing) dùng để kiểm tra cấu trúc.

Hình bên dưới biểu diễn sự tương quan của “các tiêu chí chất lượng phần mềm”, “mức độ chi tiết đơn vị” và “phương pháp kiểm nghiệm”

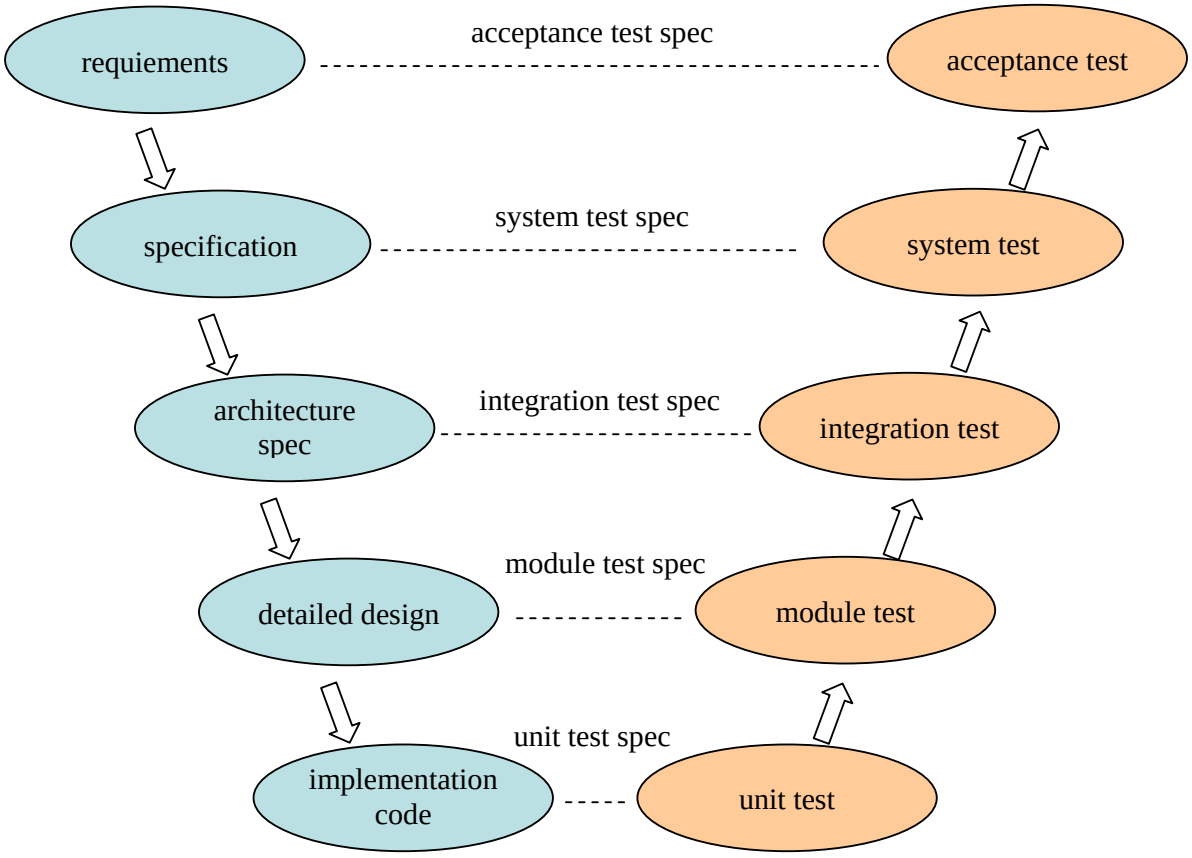


1.4. Sự tương quan giữa các công đoạn xây dựng phần mềm và loại kiểm nghiệm: Mô hình chữ V

Mô hình này nhằm giải thích sự tương quan giữa các công đoạn xây dựng phần mềm và các loại kiểm nghiệm. Ở mỗi công đoạn xây dựng phần mềm sẽ tương ứng với một loại kiểm nghiệm và cần có một hồ sơ kiểm nghiệm tương ứng được thành lập để phục vụ cho việc kiểm nghiệm.

Ví dụ:

- **Công đoạn:** Yêu cầu phần mềm(*requiements*); **Loại kiểm nghiệm:** Kiểm nghiệm chấp nhận (*acceptance test*); **Hồ sơ:** hồ sơ kiểm nghiệm chấp nhận (*acceptance test spec*).
- **Công đoạn:** Mô tả chi tiết phần mềm (*specification*); **Loại kiểm nghiệm:** Kiểm nghiệm hệ thống(*system test*); **Hồ sơ:** hồ sơ kiểm nghiệm hệ thống (*system test spec*).
- **Công đoạn:** Hồ sơ kiến trúc (*architecture spec*); **Loại kiểm nghiệm:** Kiểm nghiệm tích hợp (*integration test*); **Hồ sơ:** hồ sơ kiểm nghiệm tích hợp (*integration test spec*).
- **Công đoạn:** Thiết kế chi tiết (*detailed design*); **Loại kiểm nghiệm:** Kiểm nghiệm khối (*module test*); **Hồ sơ:** hồ sơ kiểm nghiệm khối (*module test spec*).
- **Công đoạn:** Viết mã (*implementation code*); **Loại kiểm nghiệm:** Kiểm nghiệm đơn vị (*unit test*); **Hồ sơ:** hồ sơ kiểm nghiệm đơn vị (*unit test spec*).



1.5. Sơ lượt các kỹ thuật và công đoạn kiểm nghiệm:

Các kỹ thuật và công đoạn kiểm nghiệm có thể chia như sau:

- Kiểm nghiệm tầm hẹp: kiểm nghiệm các bộ phận riêng rẽ.
  - Kiểm nghiệm hộp trắng (White box testing)
  - Kiểm nghiệm hộp đen (Black box testing)
- Kiểm nghiệm tầm rộng:
  - Kiểm nghiệm bộ phận (Module testing): kiểm nhiệm một bộ phận riêng rẽ.
  - Kiểm nghiệm tích hợp (Itegration testing): tích hợp các bộ phận và hệ thống con.
  - Kiểm nghiệm hệ thống (System testing): kiểm nghiệm toàn bộ hệ thống.
  - Kiểm nghiệm chấp nhận (Acceptance testing): thực hiện bởi khách hàng.

1.5.1. Các loại kiểm nghiệm tầm hẹp:

Các loại kiểm nghiệm này đượ thực hiện để kiểm nghiệm đến các đơn vị (unit) hoặc các khối chức năng (module).

a. Kiểm nghiệm hộp trắng (white-box testing)

Còn gọi là kiểm nghiệm cấu trúc. Kiểm nghiệm theo cách này là loại kiểm nghiệm sử dụng các thông tin về cấu trúc bên trong của ứng dụng. Việc kiểm nghiệm này dựa trên quá trình thực hiện xây dựng phần mềm.

Tiêu chuẩn của kiểm nghiệm hộp trắng phải đáp ứng các yêu cầu như sau:

- *Bao phủ dòng lệnh*: mỗi dòng lệnh ít nhất phải được thực thi 1 lần
- *Bao phủ nhánh*: mỗi nhánh trong sơ đồ điều khiển (control graph) phải được đi qua một lần.
- *Bao phủ đường*: tất cả các đường (path) từ điểm khởi tạo đến điểm cuối cùng trong sơ đồ dòng điều khiển phải được đi qua.

b. Kiểm nghiệm hộp đen (black-box testing)

Còn gọi là kiểm nghiệm chức năng. Việc kiểm nghiệm này được thực hiện mà không cần quan tâm đến các thiết kế và viết mã của chương trình. Kiểm nghiệm theo cách này chỉ quan tâm đến chức năng đã đề ra của chương trình. Vì vậy kiểm nghiệm loại này chỉ dựa vào bản mô tả chức năng của chương trình, xem chương trình có thực sự cung cấp đúng chức năng đã mô tả trong bản chức năng hay không mà thôi.

Kiểm nghiệm hộp đen dựa vào các định nghĩa về chức năng của chương trình. Các trường hợp thử nghiệm (test case) sẽ được tạo ra dựa nhiều vào bản mô tả chức năng chứ không phải dựa vào cấu trúc của chương trình.

c. Vấn đề kiểm nghiệm tại biên:

Kiểm nghiệm biên (boundary) là vấn đề được đặt ra trong cả hai loại kiểm nghiệm hộp đen và hộp trắng. Lý do là do lỗi thường xảy ra tại vùng này.

Ví dụ:

if x > y then S1 else S2

Với điều kiện bao phủ, chỉ cần 2 trường hợp thử là $x>y$ và $x<=y$.

Với kiểm nghiệm đường biên thì kiểm tra với các trường hợp thử là $x>y$, $x<y$, $x=y$

Các loại kiểm nghiệm tầm rộng:

Việc kiểm nghiệm này thực hiện trên tầm mức lớn hơn và các khía cạnh khác của phần mềm như kiểm nghiệm hệ thống, kiểm nghiệm sự chấp nhận (của người dùng)...

a. Kiểm nghiệm Module (Module testing)

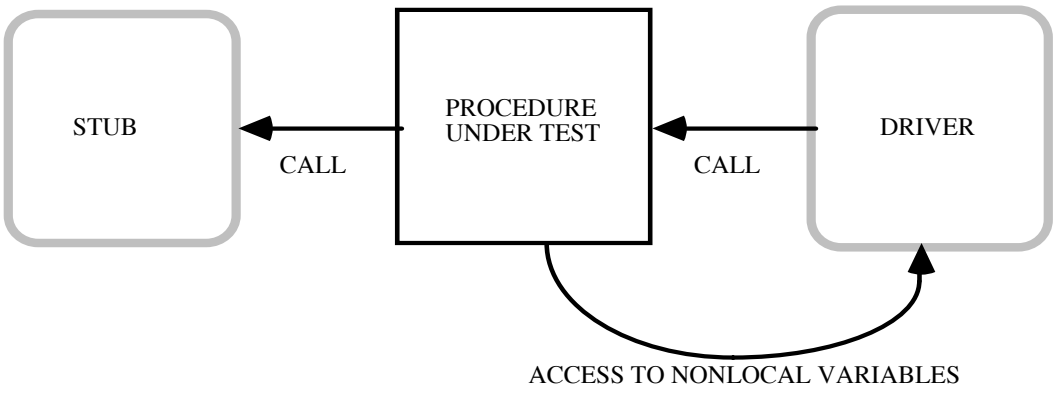
Mục đích: xác minh module đưa ra đã được xây dựng đúng hay chưa?

Vấn đề đặt ra: giả sử module I sử dụng các module H, K. Nhưng các module H và K chưa sẵn sàng. Vậy cách nào để kiểm tra module I một cách độc lập?

Giải pháp đề ra là giả lập môi trường của module H và K.

Thông thường một module có thể gọi một tác vụ (hay một tiến trình) không phải của nó, truy cập các cấu trúc dữ liệu không phải là cục bộ, hay được dùng bởi một module khác.

Hình sau mô tả module được đặt trong môi trường thử nghiệm.



Ghi chú: Driver là module gọi thực thi làm cho module cần kiểm tra hoạt động, nó giả lập các module khác sẽ sử dụng module này. Các tập dữ liệu chia sẻ mà các module khác thiết lập trong thực tế cũng được thiết lập ở drive. Stub là module giả lập các module được module đang kiểm tra sử dụng.

b. Kiểm nghiệm tích hợp:

Là cách kiểm nghiệm bằng cách tích hợp vào hệ thống từng module một và kiểm tra.

Ưu điểm:

- Dễ dàng tìm ra các lỗi vào ngay giai đoạn đầu.
- Dễ dàng khoanh vùng các lỗi (tích hợp n modules, sau đó n + 1 modules).
- Giảm việc sử dụng các stub và Driver

Có thể thực hiệm kiểm nghiệm tích hợp theo cả 2 cách bottom-up và top-down tùy thuộc vào mối quan hệ sử dụng lần nhau giữa các module.

c. Kiểm nghiệm hệ thống:

Bao gồm một loạt các kiểm nghiệm nhằm xác minh toàn bộ các thành phần của hệ thống được tích hợp một cách đúng đắn.

Mục đích của kiểm nghiệm hệ thống là để đảm bảo toàn bộ hệ thống hoạt động như ý mà khách hàng mong muốn.

Bao gồm các loại kiểm nghiệm sau:

- **Kiểm nghiệm chức năng (Function testing)**

  Kiểm tra hệ thống sau khi tích hợp có hoạt động đúng chức năng với yêu cầu đặt ra trong bản mô tả yêu cầu hay không.

  Ví dụ: với hệ thống xử lý văn bản thì kiểm tra các chức năng *tạo tài liệu, sửa tài liệu, xoá tài liệu*… có hoạt động hay không.

- **Kiểm nghiệm hiệu suất (Perfomance testing)**
- **Kiểm nghiệm mức độ đáp ứng (stress testing)**

  Thực thi hệ thống với giả thiết là các tài nguyên hệ thống yêu cầu không đáp ứng được về chất lượng, ổn định và số lượng.

- **Kiểm nghiệm cấu hình (configuration tessting)**

  Phân tích hệ thống với các thiết lập cấu hình khác nhau.

- **Kiểm nghiệm ổn định (robustness tessting)**

  Kiểm nghiệm dưới các điều kiện không mong đợi ví dụ như người dùng gõ lệnh sai, nguồn điện bị ngắt.

- **Kiểm nghiệm hồi phục (recovery testing)**

  Chỉ ra các kết quả trả về khi xảy ra lỗi, mất dữ liệu, thiết bị, dịch vụ… hoặc xoá các dữ liệu hệ thống và xem khả năng phục hồi của nó.

- **Kiểm nghiệm quá tải (overload testing)**

  Đánh giá hệ thống khi nó vượt qua giới hạn cho phép. Ví dụ: một hệ thống giao tác (transaction) được yêu cầu thực thi 20 giao tác/giây. Khi đó sẽ kiểm tra nếu 30 giao tác/giây thì như thế nào?

- **Kiểm nghiệm chất lượng (quality testing)**

  Đánh giá sự tin tưởng, vấn đề duy tu, tính sẵn sàng của hệ thống. Bao gồm cả việc tính toán thời gian trung bình hệ thống sẽ bị hỏng và thời gian trung bình để khắc phục.

- **Kiểm nghiệm cài đặt (Installation testing)**

  Người dùng sử dụng các chức năng của hệ thống và ghi lại các lỗi tại vị trí sử dụng thật sự.

  Ví dụ: một hệ thống được thiết kế để làm việc trên tàu thủy phải đảm bảo không bị ảnh hưởng gì bởi điều kiện thời tiết khác nhau hoặc do sự di chuyển của tàu.

d. Kiểm nghiệm chấp nhận

Nhằm đảm bảo việc người dùng có được hệ thống mà họ yêu cầu. Việc kiểm nghiệm này hoàn thành bởi người dùng phụ thuộc vào các hiểu biết của họ vào các yêu cầu.

CHƯƠNG 2: KIỂM CHỨNG VÀ XÁC NHẬN (V & V)

2.1. Lập kế hoạch V&V (Verification and validation planning)

2.2. Kiểm tra phần mềm (Software inspections)

2.3. Phân tích tĩnh tự động (Automated static analysis)

2.4. Phát triển phần mềm phòng sạch (Cleanroom software development)

2.1. Kiểm chứng và hợp lệ hoá

Kiểm thử phần mềm là một yếu tố trong chủ điểm rộng hơn thường được tham khảo tới như vấn đề kiểm chứng và hợp lệ hoá (V&V). Kiểm chứng nói tới một tập các hành động đảm bảo rằng phần mềm cài đặt đúng cho một chức năng đặc biệt. Hợp lệ hoá nói tới một tập các hoạt động khác đảm bảo rằng phần mềm đã được xây dựng lại theo yêu cầu của khách hàng. Bochm phát biểu điều này theo cách khác:

Kiểm chứng: "Chúng ta có làm ra sản phẩm đúng không? "

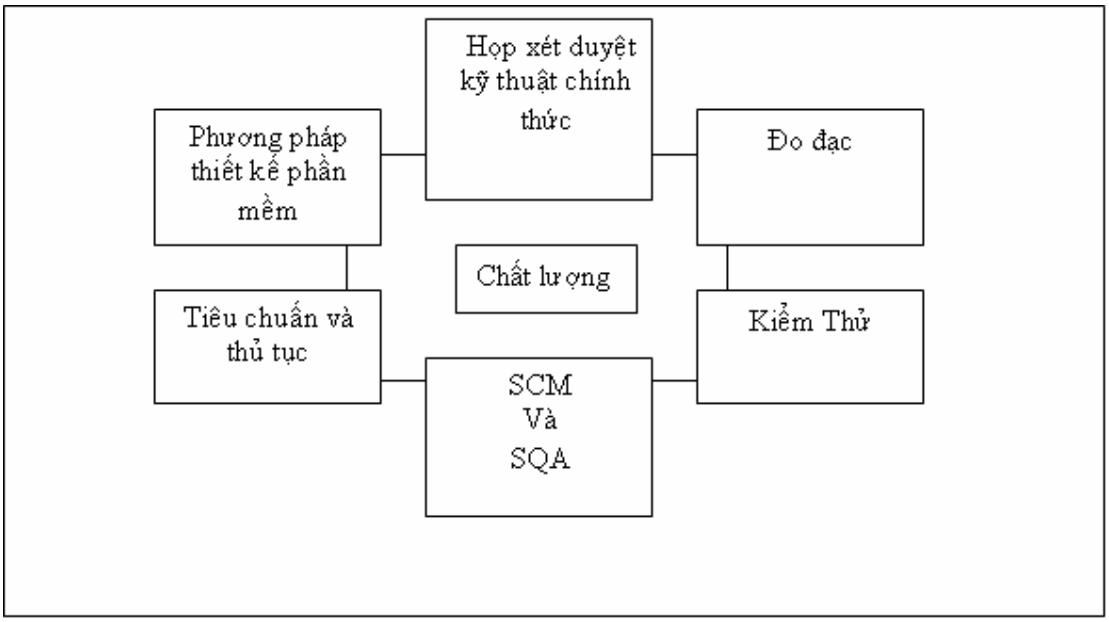
Hợp lệ hoá: "Chúng ta có làm ra đúng sản phẩm không? "

Định nghĩa về V&V bao quát nhiều hoạt động ta đã tham khảo tới như việc đảm bảo chất lượng phần mềm (SQA).

Các phương pháp kỹ nghệ phần mềm cung cấp nền tảng để xây dựng nên chất lượng. Các phương pháp phân tích, thiết kế và thực hiện (mã hoá) làm nâng cao chất lượng bằng cách đưa ra những kỹ thuật thống nhất và kết quả dự kiến được. Các cuộc họp xét duyệt kỹ thuật chính thức (thảo trình) giúp đảm bảo chất lượng của sản phẩm được tạo ra như hệ quả của từng bước kỹ nghệ phần mềm. Qua toàn bộ tiến trình này, việc đo đạc và kiểm soát được áp dụng cho mọi phần tử của cấu hình phần mềm. Các chuẩn và thủ tục giúp đảm bảo tính thống nhất và tiến trình SQA chính thức buộc phải thi hành " triết lý chất lượng toàn bộ ".

Việc kiểm thử cung cấp một thành luỹ cuối cùng để có thể thẩm định về chất lượng, lỗi có thể được phát hiện ra một cách thực tế hơn.

Nhưng không nên coi kiểm thử như một tấm lưới an toàn. Như người ta vẫn nói, " Bạn không thể kiểm thử được chất lượng. Nếu nó không sẵn có trước khi bạn bắt đầu kiểm thử thì nó sẽ chẳng có khi bạn kết thúc kiểm thử." Chất lượng được tổ hợp vào trong phần mềm trong toàn bộ tiến trình kỹ nghệ phần mềm. Việc áp dụng đúng các phương pháp và công cụ, các cuộc họp xét duyệt kỹ thuật chính thức và việc quản lý vững chắc cùng cách đo đạc tất cả dẫn tới chất lượng được xác nhận trong khi kiểm thử.



Hình 2.1 Đạt đến chất lượng phần mềm

Miller kể lại việc kiểm thử phần mềm về đảm bảo chất lượng bằng cách nói rằng "động cơ nền tảng của việc kiểm thử chương trình là để xác nhận chất lượng phần mềm bằng những phương pháp có thể được áp dụng một cách kinh tế và hiệu quả cho cả các hệ thống quy mô lớn và nhỏ."

Điều quan trọng cần lưu ý rằng việc kiểm chứng và hợp lệ hoá bao gồm một phạm vi rộng các hoạt động SQA có chứa cả họp xét duyệt chính thức, kiểm toán chất lượng và cấu hình, điều phối hiệu năng, mô phỏng, nghiên cứu khả thi, xét duyệt tài liệu, xét duyệt cơ sở dữ liệu, phân tích thuật toán, kiểm thử phát triển, kiểm thử chất lượng, kiểm thử cài đăt. Mặc dầu việc kiểm thử đóng một vai trò cực kỳ quan trọng trong V&V, nhiều hoạt động khác cũng còn cần tới.

2.1.1. Tổ chức việc kiểm thử phần mềm

Với mọi dự án phần mềm, có một mâu thuẫn cố hữu về lợi ích xuất hiện ngay khi việc kiểm thử bắt đầu. Người đã xây phần mềm bây giờ được yêu cầu kiểm thử phần mềm. Điều này bản thân nó dường như vô hại; sau rốt, ai biết được chương trình kỹ hơn là người làm ra nó? Nhưng không may, cũng những người phát triển này lại có mối quan tâm chứng minh rằng chương trình là không có lỗi, rằng nó làm việc đúng theo yêu cầu khách hàng, rằng nó sẽ được hoàn tất theo lịch biểu và trong phạm vi ngân sách. Một trong những mối quan tâm này lại làm giảm bớt việc tìm ra lỗi trong toàn bộ tiến trình kiểm thử.

Theo quan điểm tâm lý, việc phân tích và thiết kế phần mềm (cùng với mã hoá) là nhiệm vụ *xây dựng*. Người kỹ sư phần mềm tạo ra một chương trình máy tính, tài liệu về nó và các cấu trúc dữ liệu có liên quan. Giống như bất kỳ người xây dựng nào, người kỹ sư phần mềm tự hào về dinh thự đã được xây dựng và nhìn ngờ vực vào bất

kỳ ai định làm sập đổ nó. Khi việc kiểm thử bắt đầu, có một nỗ lực tinh vi, dứt khoát để "đập vỡ" cái người kỹ sư phần mềm đã xây dựng. Theo quan điểm của người xây dựng, việc kiểm thử có thể được coi như (về tâm lý) có *tính phá hoại*. Cho nên người xây dựng dè dặt đề cập tới việc kiểm thử thiết kế và thực hiện sẽ chứng tỏ rằng chương trình làm việc, thay vì phát hiện lỗi. Điều không may lỗi sẽ hiện hữu. Và nếu người kỹ sư phần mềm không tìm ra chúng thì khách hàng sẽ tìm ra.

Thường có một số nhận thức sai có thể được suy diễn sai lạc từ thảo luận trên: (1) người phát triển phần mềm không nên tiến hành kiểm thử; (2) phần mềm nên được "tung qua tường " cho người lạ làm việc kiểm thử một cách tàn bạo; (3) người kiểm thử nên tham gia vào dự án chỉ khi bước kiểm thử sắp sửa bắt đầu. Từng phát biểu này đều không đúng.

Người phát biểu phần mềm bao giờ cũng có trách nhiệm với việc kiểm thử riêng các đơn vị (mô đun) chương trình, để đảm bảo rằng mỗi mô đun thực hiện đúng chức năng nó đã được thiết kế. Trong nhiều trường hợp, người phát triển cũng tiến hành cả kiểm thử tích hợp - bước kiểm thử dẫn đến việc xây dựng (và kiểm thử) toàn bộ cấu trúc chương trình. Chỉ sau khi kiến trúc phần mềm hoàn tất thì nhóm kiểm thử độc lập mới tham gia vào.

Vai trò của nhóm kiểm thử độc lập (ITG) là loại bỏ vấn đề cố hữu liên quan tới việc để người xây dựng kiểm thử những cái anh ta đã xây dựng ra. Việc kiểm thử độc lập loại bỏ xung khắc lợi ích nếu không có nhóm đó thì có thể hiện hữu. Cuối cùng nhân sự trong nhóm kiểm thử độc lập được trả tiền để tìm ra lỗi.

Tuy nhiên, người phát triển phần mềm không chuyển giao chương trình cho ITG rồi bỏ đi. Người phát triển và ITE làm việc chặt chẽ trong toàn bộ dự án phần mềm để đảm bảo rằng những kiểm thử kỹ lưỡng sẽ được tiến hành. Trong khi tiến hành kiểm thử, người phát triển phải có sẵn để sửa chữa lỗi đã phát hiện ra.

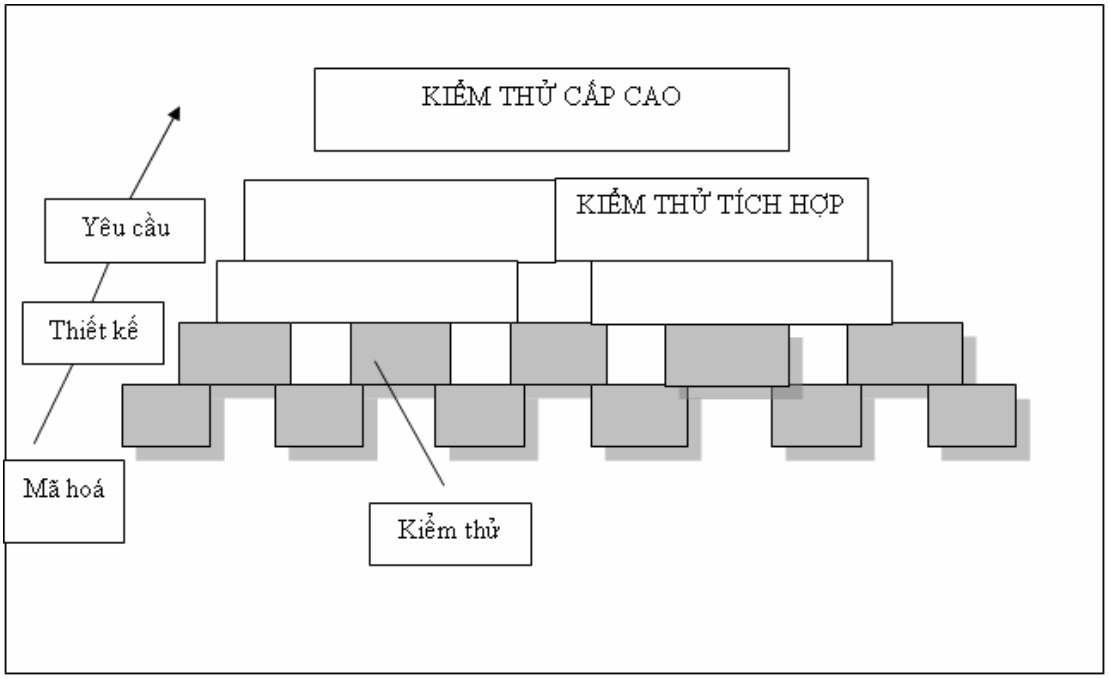
ITG là một phần của nhóm dự án phát triển phần mềm theo nghĩa là nó tham dự trong tiến trình đặc tả và vẫn còn tham dự (lập kế hoạch và xác định các thủ tục kiểm thử) trong toàn bộ dự án lớn. Tuy nhiên, trong nhiều trường hợp ITG báo cáo cho tổ chức đảm bảo chất lượng phần mềm, do đó đạt tới một mức độ độc lập có thể không có được nếu nó là một phần của tổ chức phát triển phần mềm.

2.1.2. Chiến lược kiểm thử phần mềm

Tiến trình kỹ nghệ phần mềm có thể được xét theo vòng xoắn ốc, như được minh hoạ trong Hình 2.2. Ban đầu, kỹ nghệ phần mềm xác định vai trò của phần mềm và đưa tới việc phân tích yêu cầu phần mềm, chỗ thiết lập nên lĩnh vực thông tin, chức năng, hành vi, hiệu năng, ràng buộc và tiêu chuẩn hợp lệ cho phần mềm. Đi vào trong vòng xoắn ốc, chúng ta tới thiết kế và cuối cùng tới mã hoá. Để xây dựng phần mềm máy tính, chúng ta đi dọc theo đường xoắn ốc, mỗi lần mức độ trừu tượng lại giảm dần.

Một chiến lược cho kiểm thử phần mềm cũng có thể xem xét bằng cách đi theo đường xoắn ốc của Hình 2.2 ra ngoài. Việc kiểm thử đơn vị bắt đầu tại tâm xoáy của xoắn ốc và tập chung vào các đơn vị của phần mềm khi được cài đặt trong chương trình gốc. Việc kiểm thử tiến triển bằng cách đi ra theo đường xoắn ốc tới kiểm thử tích hợp, nơi tập trung vào thiết kế và việc xây dựng kiến trúc phần mềm. Đi thêm một vòng xoáy nữa trên đường xoắn ốc chúng ta gặp kiểm thử hợp lệ, nơi các yêu cầu, được

thiết lập như một phần của việc phân tích yêu cầu phần mềm, được hợp lệ hoá theo phần mềm đã được xây dựng. Cuối cùng chúng ta tới kiểm thử hệ thống, nơi phần mềm và các phần tử hệ thống khác được kiểm thử như một toàn bộ. Để kiểm thử phần mềm máy tính, chúng ta theo đường xoáy mở rộng dần phạm vi kiểm thử một lần.



Hình 2.3 Các bước kiểm thử phần mềm

Xem xét tiến trình này theo quan điểm thủ tục vì việc kiểm thử bên trong hoàn cảnh kỹ nghệ phần mềm thực tại là một chuỗi gồm ba bước được thực hiện tuần tự nhau. Các bước này được vẽ trong Hình 2.3. Ban đầu, việc kiểm thử tập trung vào từng mô đun riêng biệt, đảm bảo rằng nó vận hành đúng đắn như một đơn vị. Do đó mới có tên kiểm thử đơn vị. Kiểm thử đơn vị dùng rất nhiều các kỹ thuật kiểm thử hộp trắng, thử các đường đặc biệt trong cấu trúc điều khiển của một mô đun để đảm bảo bao quát đầy đủ và phát hiện ra lỗi tối đa. Tiếp đó các mô đun phải được lắp ghép hay tích hợp lại để tạo nên bộ trình phần mềm hoàn chỉnh. Việc kiểm thử tích hợp đề cập tới các vấn đề có liên quan tới các vấn đề kiểm chứng và xây dựng chương trình. Các kỹ thuật thiết kế kiểm thử hộp đen chiếm đại đa số trong việc tích hợp, mặc dầu một số giới hạn các kiểm thử hộp trắng cũng có thể dược dùng để đảm bảo bao quát đa số các đường điều khiển.

Sau khi phần mềm đã được tích hợp (được xây dựng), một tập các phép kiểm thử cao cấp sẽ được tiến hành. Các tiêu chuẩn hợp lệ (được thiết lập trong phân tích yêu cầu) cũng phải được kiểm thử. Việc kiểm thử hợp lệ đưa ra sự đảm bảo cuối cùng rằng phần mềm đáp ứng cho tất cả các yêu cầu chức năng, hành vi và sự hoàn thiện. Các kỹ thuật kiểm thử hộp đen được dùng chủ yếu trong việc hợp lệ hoá này.

Bước kiểm thử cấp cao cuối cùng rơi ra ngoài phạm vi của kỹ nghệ phần mềm và rơi vào hoàn cảnh rộng hơn của kỹ nghệ hệ thống máy tính. Phần mềm, một khi được hợp lệ hoá, phải được tổ hợp với các phần tử hệ thống khác (như phần cứng, con người, cơ sở dữ liệu). Kiểm thử hệ thống kiểm chứng lại rằng tất cả các yếu tố có

khớp đúng với nhau không và rằng chức năng/ độ hoàn thiện hệ thống toàn bộ đã đạt được.

2.1.3. Tiêu chuẩn hoàn thành kiểm thử

Câu hỏi cổ điển nảy sinh mỗi khi có việc thảo luận về kiểm thử phần mềm là: Khi nào chúng ta thực hiện xong kiểm thử - làm sao ta biết rằng chúng ta đã kiểm thử đủ? Đáng buồn là không có câu trả lời xác định cho câu hỏi này, nhưng có một vài sự đáp ứng thực tế và những nỗ lực ban đầu theo hướng dẫn kinh nghiệm.

Một đáp ứng cho câu hỏi trên là: Bạn chẳng bao giờ hoàn thành việc kiểm thử, gánh nặng đơn giản chuyển từ bạn (người phát triển) sang khách hàng của bạn. Mỗi lúc khách hàng / người dùng thực hiện một chương trình máy tính thì chương trình này lại được kiểm thử trên một tập dữ liệu mới. Sự kiện đúng mức này nhấn mạnh tầm quan trọng của các hoạt động đảm bảo chất lượng phần mềm khác. Một đáp ứng khác (có điều gì đó nhạo báng nhưng dẫu sao cũng chính xác) là : Bạn hoàn thành việc kiểm thử khi bạn hết thời gian hay hết tiền.

Mặc dầu số ít người thực hành sẽ biện minh cho những đáp ứng trên, người kỹ sư phần mềm cần những tiêu chuẩn chặt chẽ hơn để xác định khi nào việc kiểm thử đủ được tiến hành. Musa và Ackerman gợi ý một đáp ứng dựa trên tiêu chuẩn thống kê: " Không, chúng ta không thể tuyệt đối chắc chắn rằng phần mềm sẽ không bao giờ hỏng, nhưng theo mô hình thống kê đúng về lý thuyết và hợp lệ về thực nghiệm thì chúng ta đã hoàn thành kiểm thử đủ để nói với sự tin tưởng tới 95% rằng xác suất của 1000 giờ vận hành CPU không hỏng trong một môi trường được xác định về xác suất là ít nhất 0.995"

Dùng mô hình hoá thống kê và lí thuyết độ tin cậy phần mềm, các mô hình về hỏng hóc phần mềm (được phát hiện trong khi kiểm thử) xem như một hàm của thời gian thực hiện có thể được xây dựng ra. Một bản của mô hình sai hỏng, được gọi là *mô hình thực hiện- thời gian Poisson lô ga rit,* có dạng:

$$f(t) = (\tfrac{1}{p}) x \ln[\, l_0(pt + 1) \,] \qquad (17.1)$$

với *f(t)* = số tích luỹ những hỏng hóc dự kiến xuất hiện một khi phần mềm đã được kiểm thử trong một khoảng thời gian thực hiện t.

l_0 = mật độ hỏng phần mềm ban đầu (số hỏng trên đơn vị thời gian) vào lúc bắt đầu kiểm thử.

P = việc giảm theo hàm mũ trong mật độ hỏng khi lỗi được phát hiện và sữa đổi được tiến hành.

Mật độ hỏng thể nghiệm. l(t), có thể được suy ra bằng cách lấy đạo hàm của f(t):

$$F(t) = \frac{l0}{l0\, pt + 1} \qquad (17.2)$$

Dùng mối quan hệ trong phương trình (2.2), người kiểm thử có thể tiên đoán việc loại bỏ lỗi khi việc kiểm thử tiến triển. Mật độ lỗi thực tại có thể được chấm lên trên đường cong dự kiến (hình 2.4). Nếu dữ liệu thực tài được thu thập trong khi

kiểm thử và mô hình thực hiện - thời gian theo logarit Poisson là xấp xỉ gần nhau với số điểm dữ liệu thì mô hình này có thể được dùng để dự đoán thời gian kiểm thử toàn bộ cần để đạt tới mật độ hỏng thấp chấp nhận được.

Bằng cách thu thập các độ đo trong khi kiểm thử phần mềm và dùng các mô hình về độ tin cậy phần mềm hiện có, có thể phát triển những hướng dẫn có nghĩa để trả lời câu hỏi: Khi nào thì chúng ta hoàn thành việc kiểm thử? Còn ít tranh luận về việc có phải làm công việc thêm nữa hay không trước khi các quy tắc định tính cho kiểm thử có thể xác định, nhưng cách tiếp cận kinh nghiệm hiện đang tồn tại được coi là tốt hơn đáng kể so với trực giác thô.

2.2. Phát triển phần mềm phòng sạch (cleanroom software development)

Cleanroom là một qui trình phát triển phần mềm hơn là một kỹ thuật kiểm thử. Cho đến bây giờ, kỹ thuật này vẫn được xem là một cách mới của việc suy nghĩ về kiểm thử và đảm bảo chất lượng phần mềm. Ý tưởng của cleanroom là nhằm tránh tiêu tốn chi phí cho hoạt động phát hiện và gỡ bỏ các lỗi bằng cách viết mã lệnh chương trình một cách chính xác ngay từ ban đầu với những phương pháp chính thống như kỹ thuật chứng minh tính đúng đắn trước khi kiểm thử.

2.2.1. Nghệ thuật của việc gỡ rối

Kiểm thử phần mềm là một tiến trình có thể được vạch kế hoạc và xác định một cách hệ thống. Việc thiết kế trường hợp kiểm thử có thể tiến hành một chiến lược xác định và có kết quả được tính toán theo thời gian.

Gỡ lỗi xuất hiện như hậu quả của việc kiểm thử thành công. Tức là, khi một trường hợp kiểm thử phát hiện ra lỗi thì việc gỡ lỗi là tiến trình sẽ nảy sinh để loại bỏ lỗi. Mặc dầu việc gỡ lỗi có thể nên là một tiến trình có trật tự, nó phần lớn còn là nghệ thuật. Người kỹ sư phần mềm khi tính các kết quả của phép thử, thường hay phải đương đầu với chỉ dẫn "triệu chứng" và vấn đề phần mềm. Tức là, cái biểu lộ ra bên ngoài của lỗi và nguyên nhân bên trong của lỗi có thể có mối quan hệ không hiển nhiên tới một lỗi khác. Tiến trình tâm trí ít hiểu biết gắn một triệu chứng với nguyên nhân chính việc gỡ lỗi.

2.2.2. Tiến trình gỡ lỗi

Gỡ lỗi không phải là kiểm thử, nhưng bao giờ cũng xuất hiện như một hệ quả kiểm thử. Tham khảo đến hình 2.12 thì tiến trình gỡ lỗi bắt đầu với việc thực hiện kiểm thử. Kết quả được thẩm định và gặp việc thiếu sự tương ứng giữa kết quả trông đợi và thực tế. Trong nhiều trường hợp, dữ liệu không tương ứng là triệu chứng của một nguyên nhân nền tảng còn bị che kín. Tiến trình gỡ lỗi cố gắng ghép triệu chứng với nguyên nhân, từ đó dẫn tới việc sửa lỗi.

Tiến trình gỡ lỗi bao giờ cũng sinh ra một trong hai kết quả logic: (1) Nguyên nhân sẽ được tìm ra, sửa chữa và loại bỏ hay (2) nguyên nhân sẽ không được tìm ra. Trong trường hợp sau, người thực hiện gỡ lỗi có thể hoài nghi một nguyên nhân, thiết kế ra một trường hợp kiểm thử giúp hợp lệ hoá hoài nghi của mình, và việc lầm hướng tới việc sửa lỗi theo cách lặp lại.

Tại sao gỡ lỗi lại khó? Rất có thể tâm lý con người (xem mục sau) có liên quan tới nhiều câu trả lời hơn là công nghệ phần mềm. Tuy nhiên môt vài đặc trưng của lỗi đưa ra vài manh mối:

- Triệu chứng và nguyên nhân có thể xa nhau về mặt địa lý. Tức là, những triệu chứng có thể xuất hiện trong một phần này của chương trình, trong khi nguyên nhân thực tế có thể định vị ở một vị trí xa. Các cấu trúc chương trình đi đôi với nhau làm trầm trọng thêm tình huống này.
- Triệu chứng có thể biến mất (tạm thời) khi một lỗi khác được sửa chữa.
- Triệu chứng thực tế có thể gây ra không lỗi (như do sự không chính xác của việc làm tròn số).
- Triệu chứng có thể được gây ra do lỗi con người không dễ lần dấu vết.
- Triệu chứng có thể là kết quả của vấn đề thời gian, thay vì vấn đề xử lý.
- Có thể khó tái tạo lại chính xác các điều kiện vào (như ứng dụng thời gian thực trong đó thứ tự vào không xác định)
- Triệu chứng có thể có lúc có lúc không. Điều này đặc biệt phổ biến trong các hệ thống nhúng việc đi đôi phần cứng và phần mềm không chặt chẽ.
- Triệu chứng có thể do nguyên nhân được phân bố qua một số các nhiệm vụ chạy trên các bộ xử lý khác nhau.
- Trong khi gỡ lỗi, chúng ta gặp không ít các lỗi chạy từ việc hơi khó chịu (như định dạng cái ra không đúng) tới các thảm hoạ (như hệ thống hỏng, gây ra các thiệt hại kinh tế hay vật lý trầm trọng). Xem như hậu quả của việc tăng lỗi, khối lượng sức ép để tìm ra lỗi cũng tăng thêm. Thông thường, sức ép buộc người phát triển phần mềm phải tìm ra lỗi và đồng thời đưa vào thêm hai lỗi nữa.

2.2.3. Xem xét tâm lý

Không may, dường như có một số bằng cớ là sự tinh thông gỡ lỗi thuộc bẩm sinh con người. Một số người làm việc đó rất giỏi, số khác lại không. Mặc dù bằng cớ kinh nghiệm về gỡ lỗi vẫn còn để mở cho nhiều cách hiểu, nhưng biến thiên lớn nhất trong khả năng gỡ lỗi đã được báo cáo lại đối với các kỹ sư phần mềm có cùng nền tảng kinh nghiệm và giáo dục.

Bình luận về khía cạnh gỡ lỗi của con người, Shneiderman phát biểu:

Gỡ lỗi là một trong những phần chán nhất của lập trình. Nó có yếu tố của việc giải quyết vấn đề hay vấn đề hóc búa, đi đôi với việc thừa nhận khó chịu rằng bạn đã sai lầm. Hay âu lo và không sẵn lòng chấp nhận khả năng lỗi làm tăng khó khăn cho công việc. May mắn là có sự giảm nhẹ và bớt căng thẳng khi lỗi cuối cùng đã được sửa lỗi.

Mặc dầu có thể khó học được việc gỡ lỗi, người ta vẫn đề nghị ra một số cách tiếp cận tới vấn đề. Chúng ta xem xét những vấn đề này trong mục tiếp.

2.2.4. Cách tiếp cận gỡ lỗi

Bất kể tới cách tiếp cận nào được chọn gỡ lỗi có một mục tiêu quan trọng hơn cả: tìm ra và sửa chữa nguyên nhân lỗi phần mềm. Mục tiêu này được thực hiện bằng tổ hợp

các đánh giá có hệ thống, trực giác và may mắn. Bradley mô tả cách tiếp cận gỡ lỗi theo cách này:

Gỡ lỗi là việc ứng dụng trực tiếp phương pháp khó học đã từng được phát triển hơn 2500 năm qua. Cơ sở của việc gỡ lỗi là định vị nguồn gốc của vấn đề [nguyên nhân] bằng việc phân hoạch nhị phân, thông qua các giả thiết làm việc để dự đoán các giá trị mới cần kiểm tra.

Ta hãy lấy một ví dụ không phải phần mềm: Đèn trong nhà tôi không làm việc. Nếu không có gì trong nhà làm việc thì nguyên nhân phải là cầu chì chính hay ở bên ngoài; tôi nhìn quanh để liệu xem hàng xóm có bị tắt đèn hay không. Tôi cắm chiếc đèn nghi ngờ vào ổ cắm khác và cắm một đồ điện khác vào mạch nghi ngờ. Cứ thế tiến hành các phương án giải quyết kiểm thử.

Nói chung, có thể đưa ra ba loại các tiếp cận gỡ lỗi:

- Bó buộc mạnh bạo
- Lật ngược
- Loại bỏ nguyên nhân

Loại bó buộc mạnh bạo có lẽ là phương pháp thông dụng nhất và kém hiệu quả nhất để cô lập nguyên nhân của lỗi phần mềm. Chúng ta áp dụng phương pháp gỡ lỗi bó buộc mạnh bạo khi tất cả các phương pháp khác đều thất bại. Dùng triết lý "cứ để máy tính tìm ra lỗi", người ta cho xổ ra nội dung bộ nhớ, gọi tới chương trình lưu dấu vết khi chạy và nạp chương trình với lệnh WRITE. Chúng ta hy vọng rằng đâu đó trong bãi lầy thông tin dược tạo ra, chúng ta có thể tìm ra được một nguyên nhân của lỗi. Mặc dầu đống thông tin được tạo ra cuối cùng có thể dẫn tới thành công, thường hơn cả là nó dẫn đến phí phạm công sức và thời gian. Phải dành suy nghĩ vào đó trước hết đã.

Lật ngược lại cách tiếp cận khá thông dụng có thể được dùng trong những chương trình nhỏ. Bắt đầu tại chỗ chúng được phát hiện ra, lật ngược theo những chương trình gốc (một cách thủ công) cho tới chỗ tìm ra nguyên nhân. Không may là khi số dòng chương trình gốc tăng lên, số con đường lật ngược tiềm năng có thể trở nên không quản lý nổi.

Cách tiếp cận thứ ba tới gỡ lỗi - loại bỏ nguyên nhân được biểu lộ bằng việc quy nạp hay diễn dịch và đưa vào khái niệm về phân hoạch nhị phân. Dữ liệu có liên quan tới việc xuất hiện lỗi được tổ chức để cô lập ra các nguyên nhân tiềm năng. Một "giả thiết nguyên nhân" được nêu ra và dữ liệu trên được dùng để chứng minh hay bác bỏ giả thiết đó. Một cách khác, ta có thể xây dựng ra một danh sách mọi nguyên nhân đặc biệt có nhiều hứa hẹn thì dữ liệu sẽ được làm mịn thêm để cố gắng cô lập ra lỗi.

Từng cách tiếp cận gỡ lỗi trên đây đều có thể được bổ sung thêm bởi công cụ gỡ lỗi. Chúng ta có thể áp dụng một phạm vi rộng các trình biên dịch gỡ lỗi, nhưng trợ giúp gỡ lỗi động ("Bộ dò dấu vết"), các bộ sinh trường hợp kiểm thử tự động, sổ bộ nhớ và bảng tham khảo chéo. Tuy nhiên các công cụ đều không phải là cách thay thế cho việc đánh giá dựa trên tài liệu thiết kế phần mềm đầy đủ và chương trình gốc rõ ràng.

Trong nhiều trường hợp, việc gỡ lỗi phần mềm máy tính tựa như việc giải quyết vấn đề trong thế giới kinh doanh. Brow và Sampson đã đưa ra một cách tiếp cận gỡ lỗi tên là "Phương pháp", đó là việc thích nghi các kỹ thuật giải quyết vấn đề quản lý. Các tác giả này đề nghị phát triển một bản đặc tả về các độ lệch, mô tả cho vấn đề bằng cách phác họa "cái gì, khi nào, ở đâu và với phạm vi nào?"

Mỗi một trong những vấn đề nêu trên (cái gì, khi nào, ở đâu và với phạm vi nào) đều được chỉ ra thành những đáp ứng là hay không là để phân biệt rõ rệt giữa cái gì đã xảy ra và cái gì đã không xảy ra. Một khi thông tin về lỗi đã được ghi lại thì người ta xây dựng ra một giả thiết nguyên nhân dựa trên các phân biệt quan sát được từ những đáp ứng là hay không là. Việc gỡ lỗi tiếp tục dùng cách tiếp cận qui nạp hay diễn dịch được mô tả phần trên trong mục này.

Bất kỳ thảo luận nào về cách tiếp cận và công cụ gỡ lỗi cũng đều không đầy đủ nếu không nói tới một đồng minh mạnh mẽ: Người khác! Khái niệm về "lập trình vô ngã" của Weinberg (được thảo luận trước đây trong cuốn sách này) nên được mở rộng thành gỡ lỗi vô ngã. Mỗi người chúng ta đều có thể nhớ lại điều gì khó xử khi mất hàng giờ, hàng ngày vì một lỗi dai dẳng. Một đồng nghiệp vẩn vơ đi qua trong nỗi thất vọng rồi chúng tôi giải thích và tung ra bản tin chương trình ra. Lập tức (dường như) nguyên nhân lỗi bị phát hiện ra. Mỉm cười một cách ngạo nghễ, anh bạn đồng nghiệp chúng ta biến mất. Một quan điểm mới mẻ, không bị che phủ bởi hàng giờ thất vọng, có thể tạo ra những điều kỳ diệu. Câu châm ngôn cuối cùng về gỡ lỗi có thể là: Khi tất cả mọi thứ khác đều sai thì hãy nhờ sự giúp đỡ.

Một khi lỗi đã được tìm ra, thì nó phải được sửa chữa. Nhưng khi chúng ta đã lưu ý, việc sửa một lỗi đôi khi có thể lại đưa vào một lỗi khác và do đó lại gây hại hơn là tốt. Van Vleck gợi ý ba câu hỏi đơn giản người kỹ sư phần mềm nên hỏi trước khi tiến hành sửa chữa để loại bỏ nguyên nhân gây lỗi:

- Liệu nguyên nhân gây lỗi này có bị tái tạo ở phần khác của chương trình hay không? Trong nhiều tình huống, một khiếm khuyết chương trình bị gây ra bởi một mẫu hình logic sai sót có thể còn phát sinh ở đâu đó khác nữa. Việc xem xét tường minh về mẫu hình logic này có thể làm phát hiện ra thêm các lỗi khác
- "Lỗi tiếp" có thể bị đưa vào là gì khi tôi chữa lỗi này? Trước khi việc sửa lỗi được tiến hành, chương trình gốc (hay tốt hơn, thiết kế) nên được đánh giá lại để thẩm định việc dính nối các cấu trúc logic dữ liệu. Nếu việc sửa lỗi được tiến hành trong một phần có độ dính nối cao thì càng phải để tâm nhiều khi tiến hành bất kỳ một sự thay đổi nào.

Ta có thể làm gì để ngăn cản lỗi này ngay từ đầu? Câu hỏi này là bước đầu tiên hướng tới việc thiết lập một cách tiếp cận đảm bảo chất lượng phần mềm thống kê. Nếu ta sửa chương trình cũng như sản phẩm thì lỗi sẽ loại bỏ chương trình hiện tại và có thể bị khử bỏ mọi chương trình tương lai

CHƯƠNG 3: KIỂM THỬ PHẦN MỀM

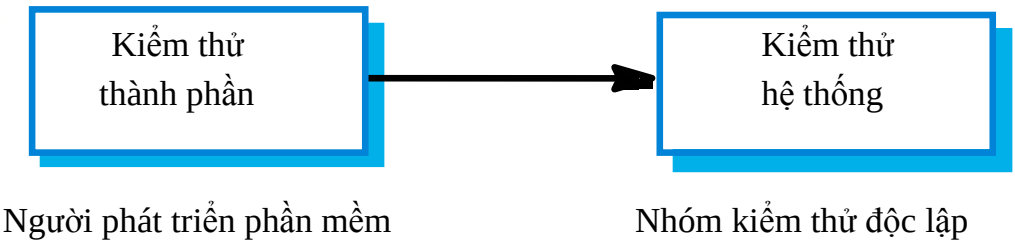
Mục tiêu của chương này là mô tả quá trình kiểm thử phần mềm và đưa ra các kỹ thuật kiểm thử. Khi đọc chương này, bạn sẽ:

- *Hiểu được sự khác biệt giữa kiểm thử hợp lệ và kiểm thử khiếm khuyết.*
- *Hiểu được các nguyên lý của kiểm thử hệ thống và kiểm thử bộ phận.*
- *Hiểu được ba chiến lược có thể sử dụng để sinh các trường hợp kiểm thử hệ thống.*
- *Hiểu được các đặc điểm bản chất của công cụ phần mềm được sử dụng để kiểm thử tự động.*

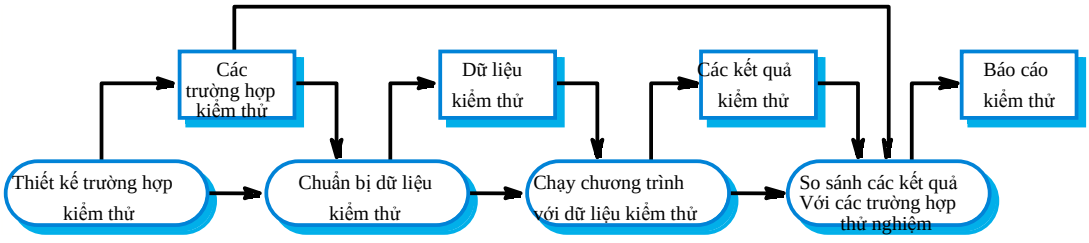
3.1. Quá trình kiểm thử

Quá trình kiểm thử phần mềm có hai mục tiêu riêng biệt:

1. Chứng minh cho người phát triển và khách hàng thấy các yêu cầu của phần mềm. Với phần mềm truyền thống, điều này có nghĩa là bạn có ít nhất một thử nghiệm cho mỗi yêu cầu của người dùng và tài liệu hệ thống yêu cầu.Với các sản phẩm phần mềm chung, điều đó có nghĩa là bạn nên thử nghiệm tất cả các đặc tính của hệ thống sẽ được kết hợp trong sản phẩm phát hành.
2. Phát hiện các lỗi và khiếm khuyết trong phần mềm: phần mềm thực hiện không đúng, không như mong đợi hoặc không làm theo như đặc tả. Kiểm tra khiếm khuyết tập trung vào việc tìm ra tất cả các kiểu thực hiện không như mong đợi của hệ thống, như sự đổ vỡ hệ thống, sự tương tác không mong muốn với hệ thống khác, tính toán sai và sai lạc dữ liệu.



Hình 3.1 Các giai đoạn kiểm thử



Hình 3.2 Một mô hình của quá trình kiểm thử phần mềm

Mục tiêu thứ nhất dẫn đến kiểm thử hợp lệ, sử dụng tập các thử nghiệm phản ánh mong muốn của người dùng để kiểm tra xem hệ thống có thực hiện đúng không. Mục tiêu thứ hai dẫn đến kiểm thử khiếm khuyết: các trường hợp kiểm thử được thiết kế để tìm ra các khiếm khuyết. Các trường hợp kiểm thử có thể được làm không rõ và không cần phản ánh cách hệ thống bình thường được sử dụng. Với kiểm thử hợp lệ, một thử nghiệm thành công là thử nghiệm mà hệ thống thực hiện đúng đắn. Với kiểm thử khiếm khuyết, một thử nghiệm thành công là một thử nghiệm tìm ra một khiếm khuyết, nguyên nhân làm cho hệ thống thực hiện không chính xác.

Kiểm thử có thể không chứng minh được phần mềm không có khiếm khuyết, hoặc nó sẽ thực hiện như đặc tả trong mọi trường hợp. Rất có thể một thử nghiệm bạn bỏ qua có thể phát hiện ra các vấn đề khác trong hệ thống. Như Dijstra, một người đi đầu trong việc phát triển kỹ nghệ phần mềm, đã tuyên bố (1972): kiểm thử chỉ có thể phát hiện ra các lỗi hiện tại, chứ không thể đưa ra tất cả các lỗi.

Nói chung, vì vậy, mục tiêu của kiểm thử phần mềm là thuyết phục người phát triển phần mềm và khách hàng rằng phần mềm là đủ tốt cho các thao tác sử dụng. Kiểm thử là một quá trình được dùng để tạo nên sự tin tưởng trong phần mềm.

Mô hình tổng quát của quá trình kiểm thử được mô tả trong hình 3.2. Các trường hợp kiểm thử sự chỉ rõ của đầu vào để thử nghiệm và đầu ra mong đợi từ hệ thống cộng với một bản báo cáo sản phẩm đã được kiểm thử. Dữ liệu kiểm thử là đầu vào, được nghĩ ra để kiểm thử hệ thống. Dữ liệu kiểm thử thỉnh thoảng có thể được tự động sinh ra. Sinh các trường hợp kiểm thử tự động là điều không làm được. Đầu ra của thử nghiệm chỉ có thể được dự đoán bởi người hiểm biết về hoạt động của hệ thống.

Kiểm thử toàn diện: mọi chương trình có thể thực hiện tuần tự được kiểm tra, là điều không thể làm được. Vì vậy, kiểm thử, phải được thực hiện trên một tập con các trường hợp kiểm thử có thể xảy ra. Trong lý tưởng, các công ty phần mềm có những điều khoản để lựa chọn tập con này hơn là giao nó cho đội phát triển. Những điều khoản này có thể dựa trên những điều khoản kiểm thử chung, như một điều khoản là tất cả các câu lệnh trong chương trình nên được thực thi ít nhất một lần. Một sự lựa chọn là những điều khoản kiểm thử có thể sự trên kinh nghiệm sử dụng hệ thống, và có thể tập trung vào kiểm thử các đặc trưng hoạt động của hệ thống. Ví dụ:

1. Tất cả các đặc trưng của hệ thống được truy cập thông qua thực đơn nên được kiểm thử.
2. Kết hợp các chức năng (ví dụ định dạng văn bản) được truy cập thông qua cùng thực đơn phải được kiểm thử.
3. Khi đầu vào được đưa vào, tất cả các chức năng phải được kiểm thử với cùng một thử nghiệm đúng đắn và thử nghiệm không đúng đắn.

Điều đó rõ ràng từ kinh nghiệm với sản phẩm phần mềm lớn như phần mềm xử lý văn bản, hoặc bảng tính có thể so sánh các nguyên tắc thông thường được sử dụng trong lúc kiểm thử sản phẩm. Khi các đặc trưng của phần mềm được sử dụng cô lập, chúng làm việc bình thuờng. Các vấn đề phát sinh, như Whittaker giải thích (Whittaker, 2002), khi liên kết các đặc trưng không được kiểm thử cùng nhau. Ông đã đưa ra một ví dụ, khi sử dụng phần mềm xử lý văn bản sử dụng sử dụng lời chú thích ở cuối trang với cách sắp xếp nhiều cột làm cho văn bản trình bày không đúng.

Khi một phần của quá trình lập kế hoạch V & V, người quản lý phải đưa ra các quyết định ai là người chịu trách nhiệm trong từng bước kiểm thử khác nhau. Với hầu hết các hệ thống, các lập trình viên chịu trách nhiệm kiểm thử các thành phần

mà họ đã triển khai. Khi các lập trình viên đã hoàn thành các công việc đó, công việc được giao cho đội tổng hợp, họ sẽ tích hợp các môđun từ những người phát triển khác nhau để tạo nên phần mềm và kiểm thử toàn bộ hệ thống. Với hệ thống quan trọng, một quá trình theo nghi thức có thể được sử dụng, các người thử độc lập chịu trách nhiệm về tất cả các bước của quá trình kiểm thử. Trong kiểm thử hệ thống quan trọng, các thử nghiệm được kiểm thử riêng biệt và hồ sơ chi tiết của kết quả kiểm thử được duy trì.

Kiểm thử các thành phần được thực hiện bởi những người phát triển thường dựa trên hiểu biết trực giác về cách hoạt động của các thành phần. Tuy nhiên, kiểm thử hệ thống phải dựa trên văn bản đặc tả hệ thống. Đó có thể là một đặc tả chi tiết yêu cầu hệ thống, hoặc nó có thể là đặc tả hướng người sử dụng ở mức cao của các đặc tính được thực hiện trong hệ thống. Thường có một đội độc lập chịu trách nhiệm kiểm thử hệ thống, đội kiểm thử hệ thống làm việc từ người sử dụng và tài liệu yêu cầu hệ thống để lập kế hoạch kiểm thử hệ thống.

Hầu hết các thảo luận về kiểm thử bắt đầu với kiểm thử thành phần và sau đó chuyển đến kiểm thử hệ thống. Tôi đã đảo ngược thử tự các thảo luận trong chương này bởi vì rất nhiều quá trình phát triển phần mềm bao gồm việc tích hợp các thành phần sử dụng lại và được lắp vào phần mềm để tạo nên các yêu cầu cụ thể. Tất cả các kiểm thử trong trường hợp này là kiểm thử hệ thống, và không có sự tách rời trong quá trình kiểm thử thành phần.

3.2. Kiểm thử hệ thống

Hệ thống gồm hai hoặc nhiều thành phần tích hợp nhằm thực hiện các chức năng hoặc đặc tính của hệ thống. Sau khi tích hợp các thành phần tạo nên hệ thống, quá trình kiểm thử hệ thống được tiến hành. Trong quá trình phát triển lặp đi lặp lại, kiểm thử hệ thống liên quan với kiểm thử một lượng công việc ngày càng tăng để phân phối cho khách hàng; trong quá trình thác nước, kiểm thử hệ thống liên quan với kiểm thử toàn bộ hệ thống.

Với hầu hết các hệ thống phức tạp, kiểm thử hệ thống gồm hai giai đoạn riêng biệt:

1. Kiểm thử tích hợp: đội kiểm thử nhận mã nguồn của hệ thống. Khi một vấn đề được phát hiện, đội tích hợp thử tìm nguồn gốc của vấn đề và nhận biết thành phần cần phải gỡ lỗi. Kiểm thử tích hợp hầu như liên quan với việc tìm các khiếm khuyết của hệ thống.
2. Kiểm thử phát hành: Một phiên bản của hệ thống có thể được phát hành tới người dùng được kiểm thử. Đội kiểm thử tập trung vào việc hợp lệ các yêu cầu của hệ thống và đảm bảo tính tin cậy của hệ thống. Kiểm thử phát hành thường là kiểm thử "hộp đen", đội kiểm thử tập trung vào mô tả các đặc tính hệ thống có thể làm được hoặc không làm được. Các vấn đề được báo cáo cho đội phát triển để gỡ lỗi chương trình. Khách hàng được bao hàm trong kiểm thử phát hành, thường được gọi là kiểm thử chấp nhận. Nếu hệ thống phát hành đủ tốt, khách hàng có thể chấp nhận nó để sử dụng.

Về cơ bản, bạn có thể nghĩ kiểm thử tích hợp như là kiểm thử hệ thống chưa đầy đủ bao gồm một nhóm các thành phần. Kiểm thử phát hành liên quan đến kiểm thử hệ thống phát hành có ý định phân phối tới khách hàng. Tất nhiên, có sự gối chồng lên

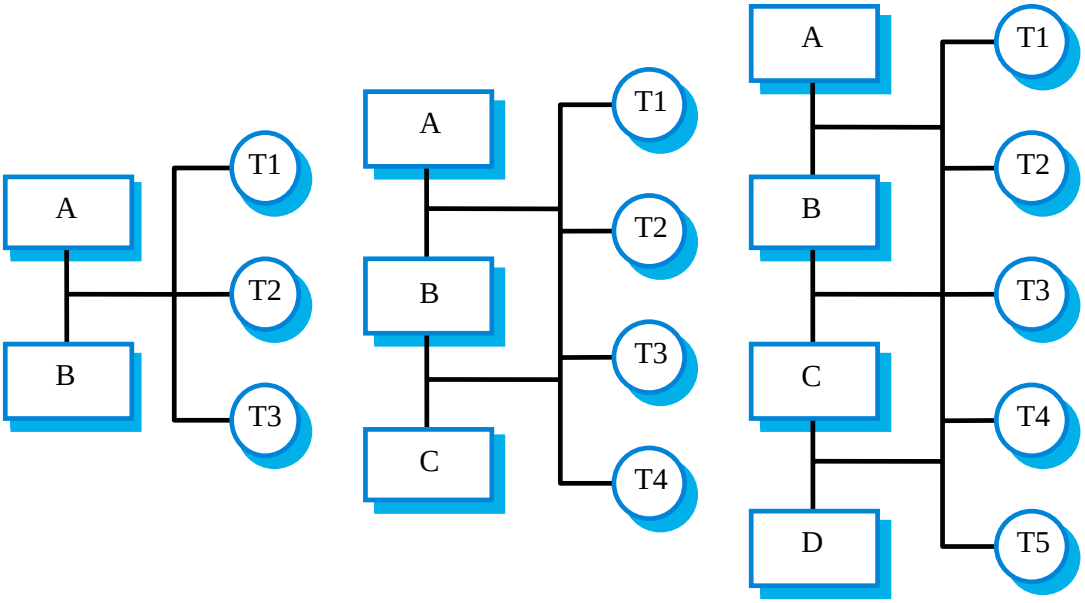
nhau, đặc biệt khi phát triển hệ thống và hệ thống được phát hành khi chưa hoàn thành. Thông thường, sự ưu tiên hàng đầu trong kiểm thử tích hợp là phát hiện ra khiếm khuyết trong hệ thống và sự ưu tiên hàng đầu trong kiểm thử hệ thống là làm hợp lệ các yêu cầu của hệ thống. Tuy nhiên trong thực tế, có vài kiểm thử hợp lệ và vài kiểm thử khiếm khuyết trong các quá trình.

3.3. Kiểm thử tích hợp

Quá trình kiểm thử tích hợp bao gồm việc xây dựng hệ thống từ các thành phần và kiểm thử hệ thống tổng hợp với các vấn đề phát sinh từ sự tương tác giữa các thành phần. Các thành phần được tích hợp có thể trùng với chính nó, các thành phần có thể dùng lại được có thể thêm vào các hệ thống riêng biệt hoặc thành phần mới được phát triển. Với rất nhiều hệ thống lớn, có tất cả 3 loại thành phần được sử dụng. Kiểm thử tích hợp kiểm tra trên thực tế các thành phần làm việc với nhau, được gọi là chính xác và truyền dữ liệu đúng vào lúc thời gian đúng thông qua giao diện của chúng.

Hệ thống tích hợp bao gồm một nhóm các thành phần thực hiện vài chức năng của hệ thống và được tích hợp với nhau bằng cách gộp các mã để chúng làm việc cùng với nhau. Thỉnh thoảng, đầu tiên toàn bộ khung của hệ thống được phát triển, sau đó các thành phần được gộp lại để tạo nên hệ thống. Phương pháp này được gọi là tích hợp từ trên xuống (top-down). Một cách lựa chọn khác là đầu tiên bạn tích hợp các thành phần cơ sở cung cấp các dịch vụ chung, như mạng, truy cập cơ sở dữ liệu, sau đó các thành phần chức năng được thêm vào. Phương pháp này được gọi là tích hợp từ dưới lên (bottom-up). Trong thực tế, với rất nhiều hệ thống, chiến lược tích hợp là sự pha trộn các phương pháp trên. Trong cả hai phương pháp top-down và bottom-up, bạn thường phải thêm các mã để mô phỏng các thành phần khác và cho phép hệ thống thực hiện.

Một vấn đề chủ yếu nảy sinh trong lúc kiểm thử tích hợp là các lỗi cục bộ. Có nhiều sự tương tác phức tạp giữa các thành phần của hệ thống, và khi một đầu ra bất thường được phát hiện, bạn có thể khó nhận ra nơi mà lỗi xuất hiện. Để việc tìm lỗi cục bộ được dễ dàng, bạn nên thường xuyên tích hợp các thành phần của hệ thống và kiểm thử chúng. Ban đầu, bạn nên tích hợp một hệ thống cấu hình tối thiểu và kiểm thử hệ thống này. Sau đó bạn thêm dần các thành phần vào hệ thống đó và kiểm thử sau mỗi bước thêm vào.



Hình 3.3 Kiểm thử tích hợp lớn dần

Trong ví dụ trên hình 2.3, A,B,C,D là các thành phần và T1, T2, T3, T4, T5 là tập các thử nghiệm kết hợp các đặc trưng của hệ thống. Đầu tiên, các thành phần A và B được kết hợp để tạo nên hệ thống (hệ thống cấu hình tối thiểu), và các thử nghiệm T1, T2, T3 được thực hiện. Nếu phát hiện có khiếm khuyết, nó sẽ được hiệu chỉnh. Sau đó, thành phần C được tích hợp và các thử nghiệm T1, T2 và T3 được làm lặp lại để đảm bảo nó không tạo nên các kết quả không mong muốn khi tương tác với A và B. Nếu có vấn đề nảy sinh trong các kiểm thử này, nó hầu như chắc chắn do sự tương tác với các thành phần mới. Nguồn gốc của vấn đề đã được khoanh vùng, vì vậy làm đơn giản việc tìm và sửa lỗi. Tập thử nghiệm T4 cũng được thực hiện trên hệ thống. Cuối cùng, thành phần D được tích hợp vào hệ thống và kiểm thử được thực hiện trên các thử nghiệm đã có và các thử nghiệm mới.

Khi lập kế hoạch tích hợp, bạn phải quyết định thứ tự tích hợp các thành phần. Trong một quá trình như XP, khách hàng cũng tham gia trong quá phát triển, khách hàng quyết định các chức năng nên được thêm vào trong mỗi bước tích hợp hệ thống. Do đó, tích hợp hệ thống được điều khiển bởi sự ưu tiên của khách hàng. Trong cách tiếp cận khác để phát triển hệ thống, khi các thành phần và các thành phần riêng biệt được tích hợp, khách hàng có thể không tham gia vào quá trình tích hợp hệ thống và đội tích hợp quyết định thứ tự tích hợp các thành phần.

Trong trường hợp này, một quy tắc tốt là đầu tiên tích hợp các thành phần thực hiện hầu hết các chức năng thường sử dụng của hệ thống. Điều này có nghĩa là các thành phần thường được sử dụng hầu hết đã được kiểm thử. Ví dụ, trong hệ thống thư viện, LIBSYS, đầu tiên bạn nên tích hợp chức năng tìm kiếm trong hệ thống tối thiểu, để người dùng có thể tìm kiếm các tài mà họ cần. Sau đó, bạn nên tích hợp các chức năng cho phép người dùng tải tài liệu từ trên Internet và dần thêm các thành phần thực hiện các chức năng khác của hệ thống.

Tất nhiên, thực tế ít khi đơn giản như mô hình trên. Sự thực hiện các chức năng của hệ thống có thể liên quan đến nhiều thành phần. Để kiểm thử một đặc tính mới, bạn có thể phải tích hợp một vài thành phần khác nhau. Kiểm thử có thể phát hiện lỗi trong khi tương tác giữa các thành phần riêng biệt và các phần khác của hệ thống. Việc sửa lỗi có thể khó khăn bởi vì một nhóm các thành phần thực hiện chức năng đó có thể phải thay đổi. Hơn nữa, tích hợp và kiểm thử một thành phần mới có thể thay đổi tương tác giữa các thành phần đã được kiểm thử. Các lỗi có thể được phát hiện có thể đã không được phát hiện trong khi kiểm thử hệ thống cấu hình đơn giản.

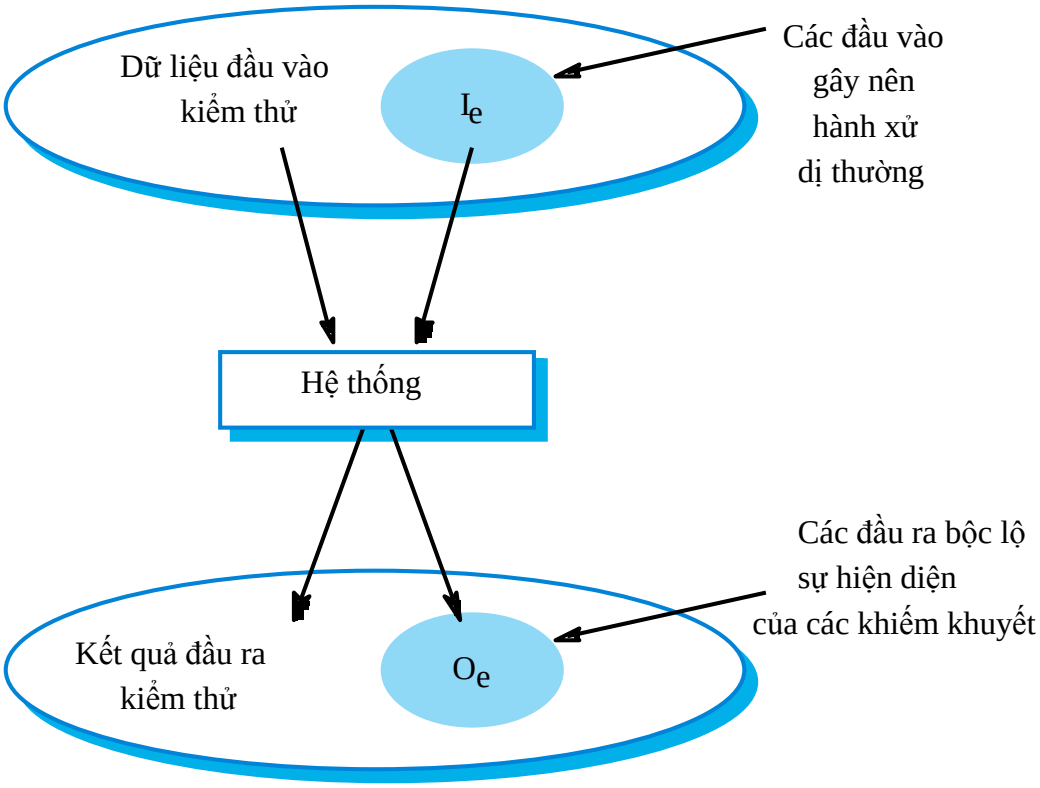
Những vấn đề này có nghĩa là khi một hệ thống tích hợp mới được tạo ra, cần phải chạy lại các thử nghiệm trong hệ thống tích hợp cũ để đảm bảo các yêu cầu các thử nghiệm đó vẫn thực hiện tốt, và các kiểm thử mới thực hiện tốt cá chức năng mới của hệ thống. Việc thực hiện kiểm thử lại tập các thử nghiệm cũ gọi là kiểm thử hồi quy. Nếu kiểm thử hồi quy phát hiện có vấn đề, thì bạn phải kiểm tra có lỗi trong hệ thống cũ hay không mà hệ thống mới đã phát hiện ra, hoặc có lỗi do thêm các chức năng mới.

Rõ ràng, kiểm thử hồi quy là quá trình tốn kém, không khả thi nếu không có sự hỗ trợ tự động. Trong lập trình cực độ, tất cả các thử nghiệm được viết như mã có thể thực thi, các đầu vào thử nghiệm và kết quả mong đợi được xác định rõ và được tự động kiểm tra. Khi được sử dụng cùng với một khung kiểm thử tự động như Junit (Massol và Husted, 2003), điều này có nghĩa là các thử nghiệm có thể được tự động thực hiện lại. Đây là nguyên lý cơ bản của lập trình cực độ, khi tập các thử nghiệm toàn diện được thực hiện bất cứ lúc nào mã mới được tích hợp và các mã mới này không được chấp nhận cho đến khi tất cả các thử nghiệm được thực hiện thành công.

3.4. Kiểm thử phát hành

Kiểm thử phát hành là quá trình kiểm thử một hệ thống sẽ được phân phối tới các khách hàng. Mục tiêu đầu tiên của quá trình này là làm tăng sự tin cậy của nhà cung cấp rằng sản phẩm họ cung cấp có đầy đủ các yêu cầu. Nếu thỏa mãn, hệ thống có thể được phát hành như một sản phẩm hoặc được phân phối đến các khách hàng. Để chứng tỏ hệ thống có đầy đủ các yêu cầu, bạn phải chỉ ra nó có các chức năng đặc tả, hiệu năng, và tính tin cậy cao, nó không gặp sai sót trong khi được sử dụng bình thường.

Kiểm thử phát hành thường là quá trình kiểm thử hộp đen, các thử nghiệm được lấy từ đặc tả hệ thống. Hệ thống được đối xử như chiếc hộp đen, các hoạt động của nó chỉ có thể được nhận biết qua việc nghiên cứu đầu vào và đầu ra của nó. Một tên khác của quá trình này là kiểm thử chức năng, bởi vì người kiểm tra chỉ tập trung xem xét các chức năng và không quan tâm sự thực thi của phần mềm.



Hình 3.4 Kiểm thử hộp đen

Hình 3.4 minh họa mô hình một hệ thống được kiểm thử bằng phương pháp kiểm thử hộp đen. Người kiểm tra đưa đầu vào vào thành phần hoặc hệ thống và kiểm tra đầu ra tương ứng. Nếu đầu ra không như dự báo trước (ví dụ, nếu đầu ra thuộc tập O_e), kiểm thử phát hiện một lỗi trong phần mềm.

Khi hệ thống kiểm thử được thực hiện, bạn nên thử mổ sẻ phần mềm bằng cách lựa chọn các trường hợp thử nghiệm trong tập I_e (trong hình 3.4). Bởi vì, mục đích của chúng ta là lựa chọn các đầu vào có xác suất sinh ra lỗi cao (đầu ra nằm trong tập O_e). Bạn sử dụng các kinh nghiệm thành công trước đó và các nguyên tắc kiểm thử để đưa ra các lựa chọn.

Các tác giả như Whittaker (Whittaker, 2002) đã tóm lược những kinh nghiệm kiểm thử của họ trong một tập các nguyên tắc nhằm tăng khả năng tìm ra các thử nghiệm khiếm khuyết. Dưới đây là một vài nguyên tắc:

1. Lựa chọn những đầu vào làm cho hệ thống sinh ra tất cả các thông báo lỗi.
2. Thiết kế đầu vào làm cho bộ đệm đầu vào bị tràn.
3. Làm lặp lại với các đầu vào như nhau hoặc một dãy các đầu vào nhiều lần.
4. Làm sao để đầu ra không đúng được sinh ra.
5. Tính toán kết quả ra rất lớn hoặc rất nhỏ.

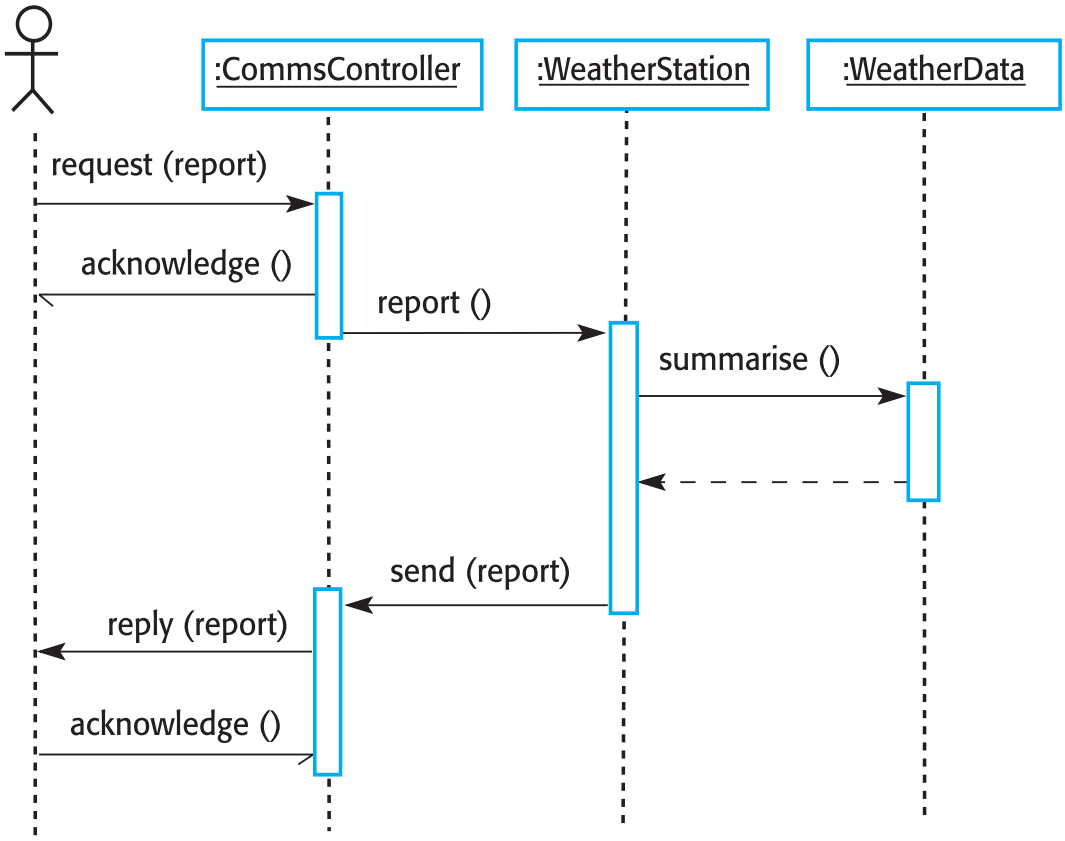
Để xác nhận hệ thống thực hiện chính xác các yêu cầu, cách tiếp cận tốt nhất vấn đề này là kiểm thử dựa trên kịch bản, bạn đưa ra một số kịch bản và tạo nên các trường

hợp thử nghiệm từ các kịch bản đó. Ví dụ, kịch bản dưới đây có thể mô tả cách hệ thống thư viện LIBSYS, đã thảo luận trong chương trước, có thể được sử dụng:

Một sinh viên ở Scốt-len nghiên cứu lịch sử nước Mỹ đã được yêu cầu viết một bài luận về "Tâm lý của người miền Tây nước Mỹ từ năm 1840 đến năm 1880". Để làm việc đó, cô ấy cần tìm các tài liệu từ nhiều thư viện. Cô ấy đăng nhập vào hệ thống LIBSYS và sử dụng chức năng tìm kiếm để tìm xem cô ấy có được truy cập vào các tài liệu gốc trong khoảng thời gian ấy không. Cô ấy tìm được các nguồn tài liệu từ rất nhiều thư viện của các trường đại học của Mỹ, và cô ấy tải một vài bản sao các tài liệu đó. Tuy nhiên, với một vài tài liệu, cô ấy cần phải có sự xác nhận từ trường đại học của cô ấy rằng cô ấy thật sự là một sinh viên và các tài liệu được sử cho những mục đích phi thương mại. Sau đó, sinh viên đó sử dụng các phương tiện của LIBSYS để yêu cầu sự cho phép và đăng ký các yêu cầu của họ. Nếu được xác nhận, các tài liệu đó sẽ được tải xuống từ máy chủ của thư viện và sau đó được in. Cô ấy nhận được một thông báo từ LIBSYS nói rằng cô ấy sẽ nhận được một e-mail khi các tài liệu đã in có giá trị để tập hợp.

Từ kịch bản trên, chúng ta có thể áp dụng một số thử nghiệm để tìm ra mục đích của LIBSYS:

1. Kiểm thử cơ chế đăng nhập bằng cách thực hiện các đăng nhập đúng và đăng nhập sai để kiểm tra người dùng hợp lệ được chấp nhận và người dùng không hợp lệ không được chấp nhận.
2. Kiểm thử cơ chế tìm kiếm bằng cách sử dụng các câu hỏi đã biết các tài liệu cần tìm để kiểm tra xem cơ chế tìm kiếm có thực sự tìm thấy các tài liệu đó.
3. Kiểm thử sự trình bày hệ thống để kiểm tra các thông tin về tài liệu có được hiển thị đúng không.
4. Kiểm thử cơ chế cho phép yêu cầu tải tài liệu xuống.
5. Kiểm thử e-mail trả lời cho biết tài liệu đã tải xuống là sẵn sàng sử dụng.



Hình 3.5 Biểu đồ dãy tập hợp dữ liệu về thời tiết

Với mỗi thử nghiệm, bạn nên thiết kế một tập các thử nghiệm bao gồm các đầu vào hợp lệ và đầu vào không hợp lệ để sinh ra các đầu ra hợp lệ và đầu ra không hợp lệ. Bạn cũng nên tổ chức kiểm thử dựa trên kịch bản, vì thế đầu tiên các kịch bản thích hợp được thử nghiệm, sau đó các kịch bản khác thường và ngoại lệ được xem xét, vì vậy sự cố gắng của bạn dành cho các phần mà hệ thống thường được sử dụng.

Nếu bạn đã sử dụng trường hợp người dùng để mô tả các yêu cầu của hệ thống, các trường hợp người dùng đó và biểu đồ liên kết nối tiếp có thể là cơ sở để kiểm thử hệ thống. Để minh họa điều này, tôi sử dụng một ví dụ từ hệ thống trạm dự báo thời tiết,

Hình 3.5 chỉ ra các thao tác lần lượt được thực hiện tại trạm dự báo thời tiết khi nó đáp ứng một yêu cầu để tập hợp dữ liệu cho hệ thống bản vẽ. Bạn có thể sử dụ biểu đồ này để nhận biết các thao tác sẽ được thử nghiệm và giúp cho việc thiết kế các trường hợp thử nghiệm để thực hiện các thử nghiệm. Vì vậy để đưa ra một yêu cầu cho một báo cáo sẽ dẫn đến sự thực hiện của một chuỗi các thao tác sau:

CommsController:request → WheatherStation:report → WeatherData:summarise

Biểu đồ đó có thể được sử dụng để nhận biết đầu vào và đầu ra cần tạo ra cho các thử nghiệm:

1. Một đầu vào của một yêu cầu báo cáo nên có một sự thừa nhận và cuối cùng báo cáo nên xuất phát từ yêu cầu. Trong lúc kiểm thử, bạn nên tạo ra dữ liệu tóm tắt, nó có thể được dùng để kiểm tra xem báo cáo được tổ chức chính xác.

2. Một yêu cầu đầu vào cho một báo cáo về kết quả của WeatherStation trong một báo cáo tóm tắt được sinh ra. Bạn có thể kiểm thử điều này một cách cô lập bằng cách tạo ra các dữ liệu thô tương ứng với bản tóm tắt, bạn đã chuẩn bị để kiểm tra CommosController và kiểm tra đối tượng WeatherStation đã được đưa ra chính xác trong bản tóm tắt.
3. Dữ liệu thô trên cũng được sử dụng để kiểm thử đối tượng WeatherData.

Tất nhiên, tôi đã làm đơn giản biểu đồ trong hình 3.5 vì nó không chỉ ra các ngoại lệ. Một kịch bản kiểm thử hoàn chỉnh cũng phải có trong bản kê khai và đảm bảo nắm bắt được đúng các ngoại lệ.

3.5. Kiểm thử hiệu năng

Ngay khi một hệ thống đã được tích hợp đầy đủ, hệ thống có thể được kiểm tra các thuộc tính nổi bất như hiệu năng và độ tin cậy. Kiểm thử hiệu năng phải được thiết kế để đảm bảo hệ thống có thể xử lý như mong muốn. Nó thường bao gồm việc lập một dãy các thử nghiệm, gánh nặng sẽ được tăng cho nên khi hệ thống không thể chấp nhận được nữa.

Cùng với các loại kiểm thử khác, kiểm thử hiệu năng liên quan đến cả việc kiểm chứng các yêu cầu của hệ thống và phát hiện các vấn đề và khiếm khuyết trong hệ thống. Để kiểm thử các yêu cầu hiệu năng đạt được, bạn phải xây dụng mô tả sơ lược thao tác. Mô tả sơ lược thao tác là tập các thử nghiệm phản ánh sự hòa trộn các công việc sẽ được thực hiện bởi hệ thống. Vì vậy, nếu 90% giao dịch trong hệ thống có kiểu A, 5% kiểu B và phần còn lại có kiểu C, D và E, thì chúng ta phải thiết kế mô tả sơ lược thao tác phần lớn tập trung vào kiểm thử kiểu A. Nếu không thì bạn sẽ không có được thử nghiệm chính xác về hiệu năng hoạt động của hệ thống.

Tất nhiên, cách tiếp cận này không nhất thiết là tốt để kiểm thử khiếm khuyết. Như tôi đã thảo luận, theo kinh nghiệm đã chỉ ra cách hiệu quả để phát hiện khiếm khuyết là thiết kế các thử nghiệm xung quanh giới hạn của hệ thống. Trong kiểm thử hiệu năng, điều này có nghĩa là nhấn mạnh hệ thống (vì thế nó có tên là kiểm thử nhấn mạnh) bằng cách tạo ra những đòi hỏi bên ngoài giới hạn thiết kế của phần mềm.

Ví dụ, một hệ thống xử lý các giao dịch có thể được thiết kế để xử lý đến 300 giao dịch mỗi giây; một hệ thống điều khiển có thể được thiết kế để điều khiển tới 1000 thiết bị đầu cuối khác nhau. Kiểm thử nhấn mạnh tiếp tục các thử nghiệm bên cạnh việc thiết kế lớn nhất được nạp vào hệ thống cho đến khi hệ thống gặp lỗi. Loại kiểm thử này có 2 chức năng:

1. Nó kiểm thử việc thực hiện lỗi của hệ thống. Trường hợp này có thể xuất hiện qua việc phối hợp các sự kiện không mong muốn bằng cách nạp vượt quá khả năng của hệ thống. Trong trường hợp này, sai sót của hệ thống làm cho dữ liệu bị hư hỏng hoặc không đáp ứng được yêu cầu của người dùng. Kiểm thử nhấn mạnh kiểm tra sự quá tải của hệ thống dẫn tới ‘thất bại mềm’ hơn là làm sụp đổ dưới lượng tải của nó.
2. Nó nhấn mạnh hệ thống và có thể gây nên khiếm khuyết trở nên rõ ràng mà bình thường không phát hiện ra. Mặc dù, nó chứng tỏ những khiếm khuyết không thể dẫn đến sự sai sót của hệ thống trong khi sử dụng bình thường, có thể hiếm gặp trong trường hợp bình thường mà kiểm thử gay cấn tái tạo.

Kiểm thử gay cấn có liên quan đặc biệt đến việc phân phối hệ thống dựa trên một một mạng lưới máy xử lý. Các hệ thống thường đưa ra đòi hỏi cao khi chúng phải thực hiện nhiều công việc. Mạng trở thành bị làm mất tác dụng với dữ liệu kết hợp mà các quá trình khác nhau phải trao đổi, vì vậy các quá trình trở nên chậm hơn, như khi nó đợi dữ liệu yêu cầu từ quá trình khác.

3.6. Kiểm thử thành phần

Kiểm thử thành phần (thỉnh thoảng được gọi là kiểm thử đơn vị) là quá trình kiểm thử các thành phần riêng biệt của hệ thống. Đây là quá trình kiểm thử khiếm khuyết vì vậy mục tiêu của nó là tìm ra lỗi trong các thành phần. Khi thảo luận trong phần giới thiệu, với hầu hết các hệ thống, người phát triển các thành phần chịu trách nhiệm kiểm thử các thành phần. Có nhiều loại thành phần khác nhau, ta có thể kiểm thử chúng theo các bước sau:

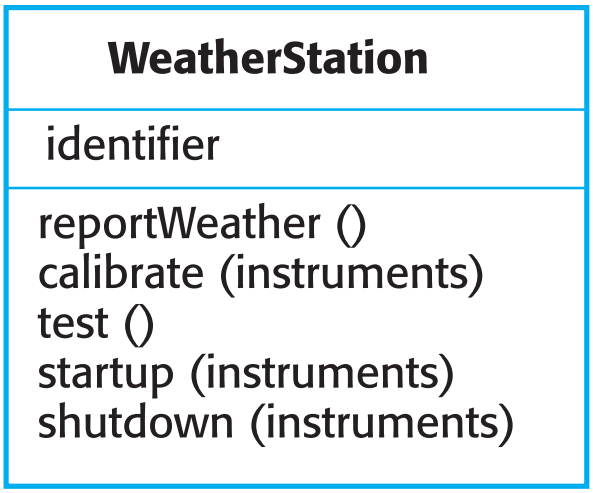
1. Các chức năng và cách thức riêng biệt bên trong đối tượng.
2. Các lớp đối tượng có một vài thuộc tính và phương thức.
3. Kết hợp các thành phần để tạo nên các đối tượng và chức năng khác nhau. Các thành phần hỗn hợp có một giao diện rõ ràng được sử dụng để truy cập các chức năng của chúng.

Các chức năng và phương thức riêng lẻ là loại thành phần đơn giản nhất và các thử nghiệm của bạn là một tập các lời gọi tới các thủ tục với tham số đầu vào khác nhau. Bạn có thể sử dụng cách tiếp cận này để thiết kế trường hợp kiểm thử (được thảo luận trong phần sau), để thiết kế các thử nghiệm chức năng và phương thức.

Khi bạn kiểm thử các lớp đối tượng, bạn nên thiết kế các thử nghiệm để cung cấp tất cả các chức năng của đối tượng. Do đó, kiểm thử lớp đối tượng nên bao gồm:

1. Kiểm thử tất cả các thao tác cô lập liên kết tạo thành đối tượng.
2. Bố trí và kiểm tra tất cả các thuộc tính liên kết tạo thành đối tượng.
3. Kiểm tra tất cả các trạng thái của đối tượng. Điều này có nghĩa là tất cả các sự kiện gây ra các trạng thái khác nhau của đối tượng nên được mô phỏng.

Hình 3.6 Giao diện của đối tượng WeatherStation



Ví dụ, trạm dự báo thời tiết có giao diện trình bày trên hình 3.6. Nó chỉ có một thuộc tính, là định danh của nó. Nó có một hằng số là tập thông số khi trạm dự báo thời tiết được thiết

đặt. Do đó, bạn chỉ cần một thử nghiệm để kiểm tra nó đã được thiết đặt hay chưa. Bạn cần xác định các trường hợp kiểm thử để kiểm tra reportWeather, calibrate, test, startup và shutdown. Trong trường hợp lý tưởng, bạn nên kiểm thử các phương thức riêng biệt, nhưng trong một vài trường hợp, cần có vài thử nghiệm liên tiếp. Ví dụ để kiểm thử phương thức shutdown bạn cần thực hiện phương thức startup.

Sử dụng mô hình này, bạn có thể nhận biết thứ tự của các trạng thái chuyển tiếp phải được kiểm thử và xác định thứ tự chuyển tiếp các sự kiện. Trong nguyên tắc này, bạn nên kiểm thử mọi trạng thái chuyển tiếp có thể xảy ra, mặc dù trong thực tế, điều này có thể rất tốn kém.Ví dụ dãy trạng thái nên kiểm thử trong trạm dự báo thời tiết bao gồm:

Shutdown → Waiting → Shutdown

Waiting → Calibrating → Testing → Transmitting → Waiting

Waiting → Collecting → Waiting → Summarising → Transmitting → Waiting

Nếu bạn sử dụng sự kế thừa sẽ làm cho việc thiết kế lớp đối tượng kiểm thử khó khăn hơn. Một lớp cha cung cấp các thao tác sẽ được kế thừa bởi một số lớp con, tất cả các lớp con nên được kiểm thử tất cả các thao tác kế thừa. Lý do là các thao tác kế thừa có thể đã thay đổi các thao tác và thuộc tính sau khi được kế thừa. Khi một thao tác của lớp cha được định nghĩa lại, thì nó phải được kiểm thử.

Khái niệm lớp tương đương, được thảo luận trong phần 23.3.2, có thể cũng được áp dụng cho các lớp đối tượng. Kiểm thử các lớp tương đương giống nhau có thể sử dụng các thuôc tính của đối tượng. Do đó, các lớp tương đương nên được nhận biết như sự khởi tạo, truy cập và cập nhật tất cả thuộc tính của lớp đối tượng.

3.7. Kiểm thử giao diện

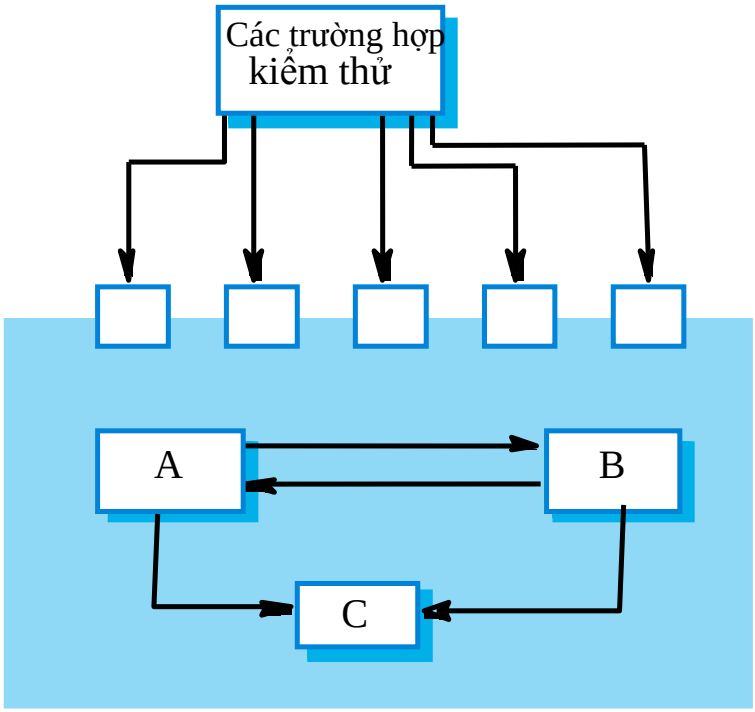
Nhiều thành phần trong một hệ thống là sự kết hợp các thành phần tạo nên bởi sự tương tác của một vài đối tượng. Kiểm thử các thành phần hỗn hợp chủ yếu liên quan đến kiểm thử hoạt động giao diện của chúng thông qua các đặc tả.

Hình 3.7 minh họa quá trình kiểm thử giao diện. Giả sử các thành phần A, B, và C đã được tích hợp để tạo nên một thành phần lớn hoặc một hệ thống con. Các thử nghiệm không chỉ áp dụng vào các thành phần riêng lẻ mà còn được áp dụng vào giao diện của các thành phần hỗn hợp được tạo nên bằng cách kết hợp các thành phần đó.

Kiểm thử giao diện đặc biệt quan trọng trong việc phát triển phần mềm hướng đối tượng và các thành phần cơ sở. Các đối tượng và các thành phần được xác định qua giao diện của chúng và có thể được sử dụng lại khi liên kết với các thành phần khác trong các hệ thống khác nhau. Các lỗi giao diện trong thành phần hỗn hợp không thể được phát hiện qua việc kiểm thử các đối tượng và các thành phần riêng lẻ. Sự tương tác giữa các thành phần trong thành phần hỗn hợp có thể phát sinh lỗi.

Có nhiều kiểu giao diện giữa các thành phần chương trình, do đó có thể xuất hiện các kiểu lỗi giao diện khác nhau:

Hình 3.7 Kiểm thử giao diện



1. Giao diện tham số: Khi dữ liệu hoặc tham chiếu chức năng được đưa từ thành phần này tới thành phần khác.
2. Giao diện chia sẻ bộ nhớ: Khi một khối bộ nhớ được chia sẻ giữa các thành phần. Dữ liệu được để trong bộ nhớ bởi một hệ thống con và được truy xuất bởi một hệ thống khác.
3. Giao diện thủ tục: Một thành phần bao gồm một tập các thủ tục có thể được gọi bởi các thành phần khác. Các đối tượng và các thành phần dùng lại có dạng giao diện này.
4. Giao diện truyền thông điệp: Một thành phần yêu cầu một dịch vụ từ một thành phần khác bằng cách gởi một thông điệp tới thành phần đó. Thông điệp trả lại bao gồm các kết quả thực hiện dịch vụ. Một vài hệ thống hướng đối tượng có dạng giao diện này như trong hệ thống chủ-khách (client-server).

Các lỗi giao diện là một dạng lỗi thường gặp trong các hệ thống phức tạp (Lutz, 1993). Các lỗi này được chia làm 3 loại:

1. Dùng sai giao diện: Một thành phần gọi tới thành phần khác và tạo nên một lỗi trong giao diện của chúng. Đây là loại lỗi rất thường gặp trong giao diện tham số: các tham số có thể được truyền sai kiểu, sai thứ tự hoặc sai số lượng tham số.
2. Hiểu sai giao diện: Một thành phần gọi tới thành phần khác nhưng hiểu sai các đặc tả giao diện của thành phần được gọi và làm sai hành vi của thành phần được gọi. Thành phần được gọi không hoạt động như mong đợi và làm cho thành phần gọi cũng hoạt động không như mong đợi. Ví dụ, một thủ tục tìm kiếm nhị phân có thể được gọi thực hiện trên một mảng chưa được xếp theo thứ tự, kết quả tìm kiếm sẽ không đúng.

3. Các lỗi trong bộ đếm thời gian: Các lỗi này xuất hiện trong các hệ thống thời gian thực sử dụng giao diện chia sẻ bộ nhớ hoặc giao diện truyền thông điệp. Dữ liệu của nhà sản xuất và dữ liệu của khách hàng có thể được điều khiển với các tốc độ khác nhau. Nếu không chú ý đến trong thiết kế giao diện, thì khách hàng có thể truy cập thông tin lỗi thời bởi vì thông tin của nhà sản xuất chưa được cập nhật trong giao diện chia sẻ.

Kiểm thử những khiếm khuyết trong giao diện rất khó khăn bởi vì một số lỗi giao diện chỉ biểu lộ trong những điều kiện đặc biệt. Ví dụ, một đối tượng có chứa một danh sách hàng đợi với cấu trúc dữ liệu có chiều dài cố định. Giả sử danh sách hàng đợi này được thực hiện với một cấu trúc dữ liệu vô hạn và không kiểm tra việc tràn hàng đợi khi một mục được thêm vào. Trường hợp này chỉ có thể phát hiện khi kiểm thử với những thử nghiệm làm cho tràn hàng đợi và làm sai hành vi của đối tượng theo những cách có thể nhận biết được.

Những lỗi khác có thể xuất hiện do sự tương tác giữa các lỗi trong các môđun và đối tượng khác nhau. Những lỗi trong một đối tượng có thể chỉ được phát hiện khi một vài đối tượng khác hoạt động không như mong muốn. Ví dụ, một đối tượng có thể gọi một đối tượng khác để nhận được một vài dịch vụ và giả sử được đáp ứng chính xác. Nếu nó đã hiểu sai về giá trị được tính, thì giá trị trả về là hợp lệ nhưng không đúng. Điều này chỉ được phát hiện khi các tính toán sau đó có kết quả sai.

Sau đây là một vài nguyên tắc để kiểm thử giao diện:

1. Khảo sát những mã đã được kiểm thử và danh sách lời gọi tới các thành phần bên ngoài.
2. Với những tham số trong một giao diện, kiểm thử giao diện với tham số đưa vào rỗng.
3. Khi một thành phần được gọi thông qua một giao diện thủ tục, thiết kế thử nghiệm sao cho thành phần này bị sai. Các lỗi khác hầu như là do hiểu sai đặc tả chung.
4. Sử dụng kiểm thử gay cấn, như đã thảo luận ở phần trước, trong hệ thống truyền thông điệp. Thiết kế thử nghiệm sinh nhiều thông điệp hơn trong thực tế. Vấn đề bộ đếm thời gian có thể được phát hiện theo cách này.
5. Khi một vài thành phần tương tác thông qua chia sẻ bộ nhớ, thiết kế thử nghiệm với thứ tự các thành phần được kích hoạt thay đổi. Những thử nghiệm này có thể phát hiện những giả sử ngầm của các lập trình viên về thứ tự dữ liệu chia sẻ được sử dụng và được giải phóng.

Kỹ thuật hợp lệ tĩnh thường hiệu quả hơn kiểm thử để phát hiện lỗi giao diện. Một ngôn ngữ định kiểu chặt chẽ như JAVA cho phép ngăn chặn nhiều lỗi giao diện bởi trình biên dịch. Khi một ngôn ngữ không chặt chẽ như C được sử dụng, một phân tích tĩnh như LINT có thể phát hiện các lỗi giao diện. Sự kiểm tra chương trình có thể tập trung vào các giao diện giữa các thành phần và câu hỏi về hành vi giao diện xảy ra trong quá trình kiểm tra.

3.8. Thiết kế trường hợp thử (Test case design)

Thiết kế trường hợp thử nghiệm là một phần của kiểm thử hệ thống và kiểm thử thành phần, bạn sẽ thiết kế các trường hợp thử nghiệm (đầu vào và đầu ra dự đoán) để kiểm thử hệ thống. Mục tiêu của quá trình thiết kế trường hợp kiểm thử là tạo ra một tập

các trường hợp thử nghiệm có hiệu quả để phát hiện khiếm khuyết của chương trình và chỉ ra các yêu cầu của hệ thống.

Để thiết kế một trường hợp thử nghiệm, bạn chon một chức năng của hệ thống hoặc của thành phần mà bạn sẽ kiểm thử. Sau đó bạn chọn một tập các đầu thực hiện các chức năng đó, và cung cấp tài liệu về đầu ra mong muốn và giới hạn của đầu ra, và điểm mà có thể thiết kế tự động để kiểm tra thử nghiệm với đầu ra thực tế và đầu ra mong đợi vẫn như thế.

Có nhiều phương pháp khác nhau giúp bạn có thể thiết kế các trường hợp thử nghiệm:

1. Kiểm thử dựa trên các yêu cầu: Các trường hợp thử nghiệm được thiết kế để kiểm thử các yêu cầu hệ thống. Nó được sử dụng trong hầu hết các bước kiểm thử hệ thống bởi vì các yêu cầu hệ thống thường được thực hiện bởi một vài thành phần. Với mỗi yêu cầu, bạn xác định các trường hợp thử nghiệm để có thể chứng tỏ được hệ thống có yêu cầu đó.
2. Kiểm thử phân hoạch: bạn xác định các phân hoạch đầu vào và phân hoạch đầu ra và thiết kể thử nghiệm, vì vậy hệ thống thực hiện với đầu vào từ tất cả các phân hoạch và sinh ra đầu ra trong tất cả các phân hoạch. Các phân hoạch là các nhóm dữ liệu có chung đặc tính như tất cả các số đều âm, tất cả tên đều có đều có độ dài nhỏ hơn 30 ký tự, tất cả các sự kiện phát sinh từ việc chọn các mục trên thực đơn…
3. Kiểm thử cấu trúc: Bạn sử dụng những hiểu biết về cấu trúc chương trình để thiết kế các thử nghiệm thực hiện tất cả các phần của chương trình. Về cơ bản, khi kiểm thử một chương trình, bạn nên kiểm tra thực thi mỗi câu lệnh ít nhất một lần. Kiểm thử cấu trúc giúp cho việc xác định các trường hợp thử nghiệm.

Thông thường, khi thiết kế các trường hợp thử nghiệm, bạn nên bắt đầu với các thử nghiệm mức cao nhất của các yêu cầu, sau đó thêm dần các thử nghiệm chi tiết bằng cách sử dụng kiểm thử phân hoạch và kiểm thử cấu trúc.

3.8.1. Kiểm thử dựa trên các yêu cầu

Một nguyên lý chung của các yêu cầu kỹ nghệ là các yêu cầu phải có khả năng kiểm thử được. Các yêu cầu nên được viết theo cách mà một thử nghiệm có thể được thiết kế, do đó quan sát viên có thể kiểm tra xem yêu cầu đó đã thỏa mãn chưa. Vì vậy, kiểm thử dựa trên các yêu cầu là một tiếp cận có hệ thống để thiết kế trường hợp thử nghiệm giúp cho bạn xem xét mỗi yêu cầu và tìm ra các thử nghiệm. Kiểm thử dựa trên các yêu cầu có hiệu quả hơn kiểm thử khiếm khuyết – bạn đang chứng tỏ hệ thống thực hiện được đầy đủ các yêu cầu.

Ví dụ, hãy xem xét các yêu cầu cho hệ thống LIBSYS .

1. Người dùng có thể tìm kiếm hoặc tất cả các tập ban đầu của cơ sở dữ liệu hoặc lựa chọn một tập con từ đó.
2. Hệ thống sẽ cung cấp các khung nhìn hợp lý cho người dùng để đọc tài liệu trong kho tài liệu.
3. Mọi yêu cầu sẽ được cấp phát một định danh duy nhất (ORDER_ID) để người dùng có thể được phép sao chép qua tài khoản của vùng lưu trữ thường trực.

Giả sử chức năng tìm kiếm đã được kiểm thử, thì các thử nghiệm có thể chấp nhận được cho yêu cầu thứ nhất là:

- Ban đầu, người dùng tìm kiếm các mục mà đã biết sự có mặt và đã biết không có trong tập cơ sở dữ liệu chỉ gồm có một cở sở dữ liệu.

- Ban đầu, người dùng tìm kiếm các mục mà đã biết sự có mặt và đã biết không có trong tập cơ sở dữ liệu gồm có hai cở sở dữ liệu.
- Ban đầu, người dùng tìm kiếm các mục mà đã biết sự có mặt và đã biết không có trong tập cơ sở dữ liệu gồm có nhiều hơn hai cở sở dữ liệu.
- Lựa chọn một cơ sở dữ liệu từ tập cơ sở dữ liệu, người dùng tìm kiếm các mục mà đã biết sự có mặt và đã biết không có trong cơ sở dữ liệu đó.
- Lựa chọn nhiều hơn một cơ sở dữ liệu từ tập cơ sở dữ liệu, người dùng tìm kiếm các mục mà đã biết sự có mặt và đã biết không có trong cơ sở dữ liệu đó.

Từ đó, bạn có thể thấy kiểm thử một yêu cầu không có nghĩa là chỉ thực hiện kiểm thử trên một thử nghiệm. Thông thường, bạn phải thực kiểm thử nghiệm trên một vài thử nghiệm để đảm bảo bạn đã kiểm soát được yêu cầu đó.

Kiểm thử các yêu cầu khác trong hệ thống LIBSYS có thể được thực hiện theo giống như trên. Với yêu cầu thứ hai, bạn sẽ soạn ra các thử nghiệm để phân phối tất cả các kiểu tài liệu có thể được xử lý bởi hệ thống và kiểm tra sự hiển thị các tài liệu đó. Với yêu cầu thứ ba, bạn giả vờ đưa vào một vài yêu cầu, sau đó kiểm tra định danh yêu cầu được hiển thị trong giấy chứng nhận của người dùng, và kiểm tra định danh yêu cầu đó có là duy nhất hay không.

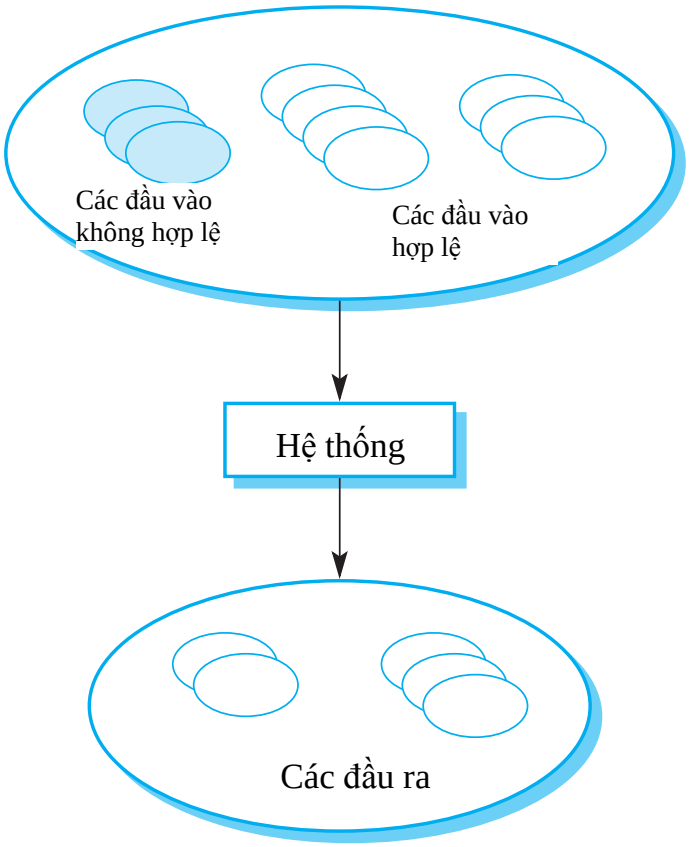
3.8.2. Kiểm thử phân hoạch

Dữ liệu đầu vào và kết quả đầu ra của chương trình thường được phân thành một số loại khác nhau, mỗi loại có những đặc trưng chung, như các số đều dương, các số đều âm, và các thực đơn lựa chọn. Thông thường, các chương trình thực hiện theo cách có thể so sánh được với tất cả thành viên của một lớp. Do đó, nếu chương trình được kiểm thử thực hiện những tính toán và yêu cầu hai số dương, thì bạn sẽ mong muốn chương trình thực hiện theo cách như nhau với tất cả các số dương.

Bởi vì cách thực hiện là tương đương, các loại này còn được gọi là phân hoạch tương đương hay miền tương đương (Bezier, 1990). Một cách tiếp cận có hệ thống để thiết kế các trường hợp kiểm thử là dựa trên sự định danh của tất cả các phân hoạch trong một hệ thống hoặc một thành phần. Các trường hợp thử nghiệm được thiết kế sao cho đầu vào và đầu ra nằm trong phân hoạch đó. Kiểm thử phân hoạch có thể được sử dụng để thiết kế các trường hợp thử nghiệm cho các hệ thống và các thành phần.

Trong hình 3.8, mỗi phân hoạch tương đương được biểu thị như một elíp. Đầu vào các phân hoạch tương đương là những tập dữ liệu, tất cả các tập thành viên nên được xử lý một cách tương đương. Đầu ra phân hoạch tương là đầu ra của chương trình và chúng có các đặc trưng chung, vì vậy chúng có thể được kiểm tra như một lớp riêng biệt. Bạn cũng xác định các phân hoạch có đầu vào ở bên ngoài các phân hoạch khác. Kiểm tra các thử nghiệm mà chương trình sử dụng đầu vào không hợp lệ có thực hiện đúng cách thức không. Các đầu vào hợp lệ và đầu vào không hợp lệ cũng được tổ chức thành các phân hoạch tương đương.

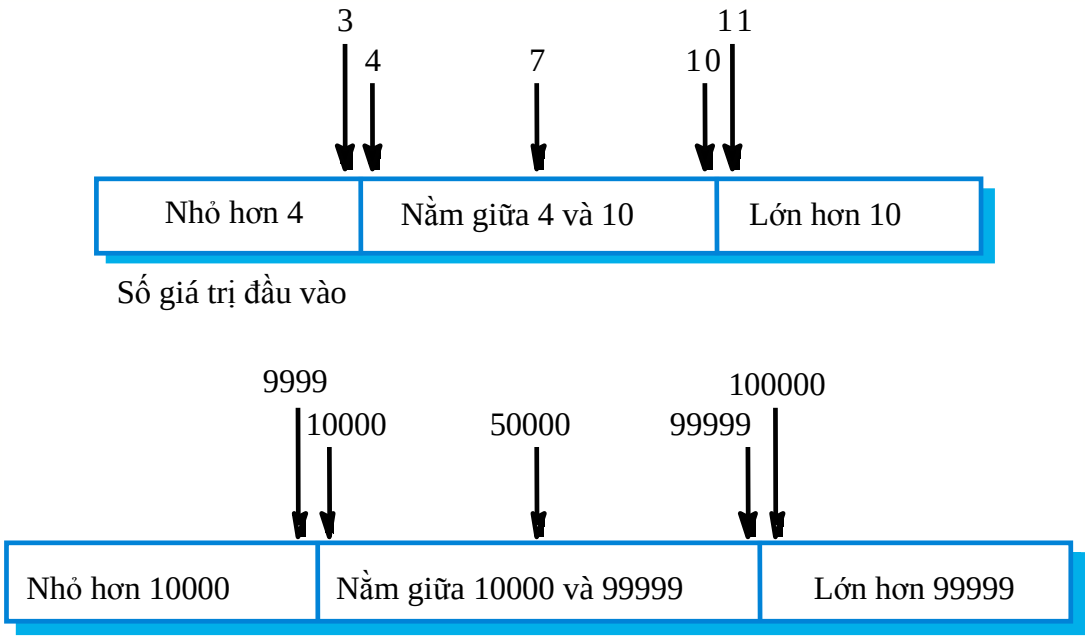
Hình 3.8
Phân hoạch
tương đương



Khi bạn đã xác định được tập các phân hoạch, bạn có thể lựa chọn các trường hợp thử nghiệm cho mỗi phân hoạch đó. Một quy tắc tốt để lựa chọn trường hợp thử nghiệm là lựa chọn các trường hợp thử nghiệm trên các giới hạn của phân hoạch cùng với các thử nghiệm gần với điểm giữa của phân hoạch. Lý do căn bản là người thiết kế và lập trình viên thường xem xét các giá trị đầu vào điển hình khi phát triển một hệ thống. Bạn kiểm thử điều đó bằng cách lựa chọn điểm giữa của hệ thống. Các giá trị giới hạn thường không điển hình (ví dụ, số 0 có thể được sử dụng khác nhau trong các tập các số không âm), vì vậy nó không được người phát triển chú ý tới. Các lỗi của chương trình thường xuất hiện khi nó xử lý các giá trị không điển hình.

Bạn xác định các phân hoạch bằng cách sử dụng đặc tả chương trình hoặc tài liệu hướng dẫn sử dụng, và từ kinh nghiệm của mình, bạn dự đoán các loại giá trị đầu vào thích hợp để phát hiện lỗi. Ví dụ, từ đặc trưng của chương trình: chương trình chấp nhận từ 4 đến 8 đầu vào là các số nguyên có 5 chữ số lớn hơn 10 000. Hình 3.9 chỉ ra các phân hoạch cho tình huống này và các giá trị đầu vào có thể xảy ra.

Để minh họa cho nguồn gốc của những trường hợp thử nghiệm này, sử dụng các đặc tả của thành phần tìm kiếm (trên hình 3.10). Thành phần này tìm kiếm trên một dãy các phần tử để đưa ra phần tử mong muốn (phần tử khóa). Nó trả lại vị trí của phần tử đó trong dãy. Tôi đã chỉ rõ đây là một cách trừu tượng để xác định các điều kiện tiên quyết phải đúng trước khi thành phần đó được gọi, và các hậu điều kiện phải đúng sau khi thực hiện.



Hình 3.9 Các phân hoạch tương đương

Điều kiện tiên quyết: Thử tục tìm kiếm sẽ chỉ làm việc với các dãy không rỗng. Hậu điều kiện: biến **Found** được thiết đặt nếu phần tử khóa thuộc dãy. Phần tử khóa có chỉ số L. Giá trị chỉ số không được xác định nếu phần tử đó không thuộc dãy.

Từ đặc trưng đó, bạn có thể nhận ra hai phân hoạch tương đương:

1. Các đầu vào có phần tử khóa là một phần tử của dãy (Found = true).
2. Các đầu vào có phần tử khóa không phải là một phần tử của dãy (Found = false).

procedure Search (Key : ELEM ; T: SEQ of ELEM;
Found : **in out** BOOLEAN; L: **in out** ELEM_INDEX) ;

Tiền điều kiện

-- Dãy có ít nhất một phần tử
T'FIRST <= T'LAST

Hậu điều kiện

-- Phần tử được tìm thấy và được chỉ bởi L
(Found and T (L) = Key)

hoặc

-- Phần tử không thuộc dãy
(**not** Found **and**
not (**exists** i, T'FIRST >= i <= T'LAST, T (i) = Key))

Hình 3.10 Đặc tả chương trình tìm kiếm

Dãy	Phần tử
Có một giá trị	Thuộc dãy
Có một giá trị	Không thuộc dãy
Nhiều hơn một giá trị	Là phần tử đầu tiên trong dãy
Nhiều hơn một giá trị	Là phần tử cuối cùng trong dãy
Nhiều hơn một giá trị	Là phần tử nằm giữa trong dãy
Nhiều hơn một giá trị	Không thuộc dãy

Dãy đầu vào	Khóa (Key)	Đầu ra (Found,L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Hình 3.11 Các phân hoạch tương đương cho chương trình tìm kiếm

Khi bạn thử nghiệm chương trình với các dãy, mảng hoặc danh sách, một số nguyên tắc thường được sử dụng để thiết kế các trường hợp kiểm thử:

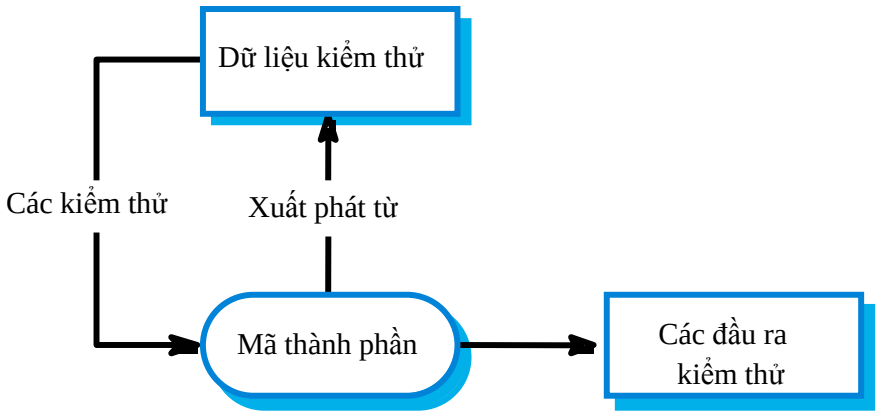
1. Kiểm thử phần mềm với dãy chỉ có một giá trị. Lập trình viên thường nghĩ các dãy gồm vài giá trị, và thình thoảng, họ cho rằng điều này luôn xảy ra trong các chương trình của họ. Vì vậy, chương trình có thể không làm việc chính xác khi dãy được đưa vào chỉ có một giá trị.
2. Sử dụng các dãy với các kích thước khác nhau trong các thử nghiệm khác nhau. Điều này làm giảm cơ hội một chương trình khiếm khuyết sẽ ngẫu nhiên đưa ra đầu ra chính xác bởi vì các đầu vào có các đặc tính ngẫu nhiên.
3. Xuất phát từ các thử nghiệm có phần tử đầu tiên, phần tử ở giữa, và phần tử cuối cùng được truy cập. Cách tiếp cận này bộc lộ các vấn đề tại các giới hạn phân hoạch.

Từ các nguyên tắc trên, hai phân hoạch tương đương có thể được xác định:

1. Dãy đầu vào có một giá trị.
2. Số phần tử trong dãy đầu vào lớn hơn 1.

Sau khi, bạn xác định thêm các phân hoạch bằng cách kết hợp các phân hoạch đã có, ví dụ, kết hợp phân hoạch có số phần tử trong dãy lớn hơn 1 và phần tử khóa không thuộc

dãy. Hình 3.11 đưa ra các phân hoạch mà bạn đã xác định để kiểm thử thành phần tìm kiếm.

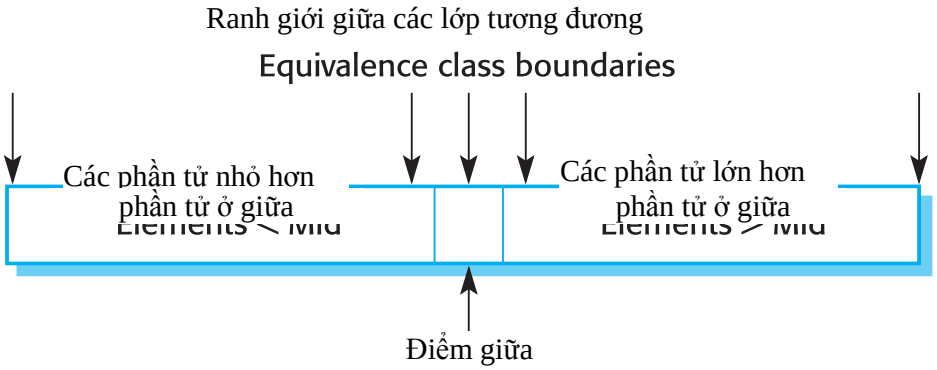


Hình 3.12 Kiểm thử cấu trúc

Một tập các trường hợp thử nghiệm có thể dựa trên các phân hoạch đó cũng được đưa ra trên hình 3.11. Nếu phần tử khóa không thuộc dãy, giá trị của L là không xác định ('??'). Nguyên tắc "các dãy với số kích thước khác nhau nên được sử dụng" đã được áp dụng trong các trường hợp thử nghiệm này.

Tập các giá trị đầu vào sử dụng để kiểm thử thủ tục tìm kiếm không bao giờ hết. Thủ tục này có thể gặp lỗi nếu dãy đầu vào tình cờ gồm các phần tử 1, 2, 3 và 4. Tuy nhiên, điều đó là hợp lý để giả sử: nếu thử nghiệm không phát hiện khiếm khuyết khi một thành viên của một loại được xử lý, không có thành viên khác của lớp sẽ xác định các khiếm khuyết. Tất nhiên, các khiếm khuyết sẽ vẫn tồn tại. Một vài phân hoạch tương đương có thể không được xác định, các lỗi có thể đã được tạo ra trong phân hoạch tương đương hoặc dữ liệu thử nghiệm có thể đã được chuẩn bị không đúng.

3.8.3. Kiểm thử cấu trúc



Hình 3.13 Các lớp tương đương trong tìm kiếm nhị phân

Kiểm thử cấu trúc (hình 3.12) là một cách tiếp cận để thiết kế các trường hợp kiểm thử, các thử nghiệm được xác định từ sự hiểu biết về cấu trúc và sự thực hiện của phần mềm. Cách tiếp cận này thỉnh thoảng còn được gọi là kiểm thử "hộp trắng", "hộp kính", hoặc kiểm thử "hộp trong" để phân biệt với kiểm thử hộp đen.

```
Class BinSearch {

// Đây là một hàm tìm kiếm nhị phân được thực hiện trên một dãy các
// đối tượng đã có thứ tự và một khóa, trả về một đối tượng với 2 thuộc
// tính là:
// index – giá trị chỉ số của khóa trong dãy
// found – có kiểu logic cho biết có hay không có khóa trong dãy
// Một đối tượng được trả về bởi vì trong Java không thể thông qua các
// kiểu cơ bản bằng tham chiếu tới một hàm và trả về hai giá trị
// Giá trị index  = -1 nếu khóa không có trong dãy

        public static void search( int key, int[] elemArray, Result r)
        {
1.          int bottom = 0;
2.          int top =  elemArray.length – 1;
            int mid;
3.          r.found = false;
4.          r.index = -1;
5.          while (bottom <= top)
                {
6.                  mid = (top + bottom) / 2;
7.                  if (elemArray[mid] = key)
                    {
8.                      r.index = mid;
9.                      r.found = true;
10.                     return;
                    } // if part
                    else
                    {
11.                     if (elemArray[mid] < key)
12.                             bottom = mid + 1;
                        else
13.                             top = mid -1;
                    }
                } // while loop
14.     }// search
}// BinSearch
```

Hình 3.14 Chương trình tìm kiếm nhị phân được viết bằng Java

Hiểu được cách sử dụng thuật toán trong một thành phần có thể giúp bạn xác định thêm các phân hoạch và các trường hợp thử nghiệm. Để minh họa điều này, tôi đã thực hiện cách đặc tả thủ tục tìm kiếm (hình 3.10) như một thủ tục tìm kiếm nhị phân (hình 3.14). Tất nhiên, điều kiện tiên quyết đã được bảo đảm nghiêm ngặt. Dãy được thực thi

Dãy đầu vào (T)	Khóa (Key)	Đầu ra (Found,L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30,31,41,45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Hình 3.15 Các trường hợp kiểm thử cho chương trình tìm kiếm

như một mảng và mảng này phải được sắp xếp và giá trị giới hạn dưới phải nhỏ hơn giá trị giới hạn trên.

Để kiểm tra mã của thủ tục tìm kiếm, bạn có thể xem việc tìm kiếm nhị phân chia không gian tìm kiếm thành 3 phần. Mỗi phần được tạo bởi một phân hoạch tương đương (hình 3.13). Sau đó, bạn thiết kế các trường hợp thử nghiệm có phần tử khóa nằm tại các giới hạn của mỗi phân hoạch.

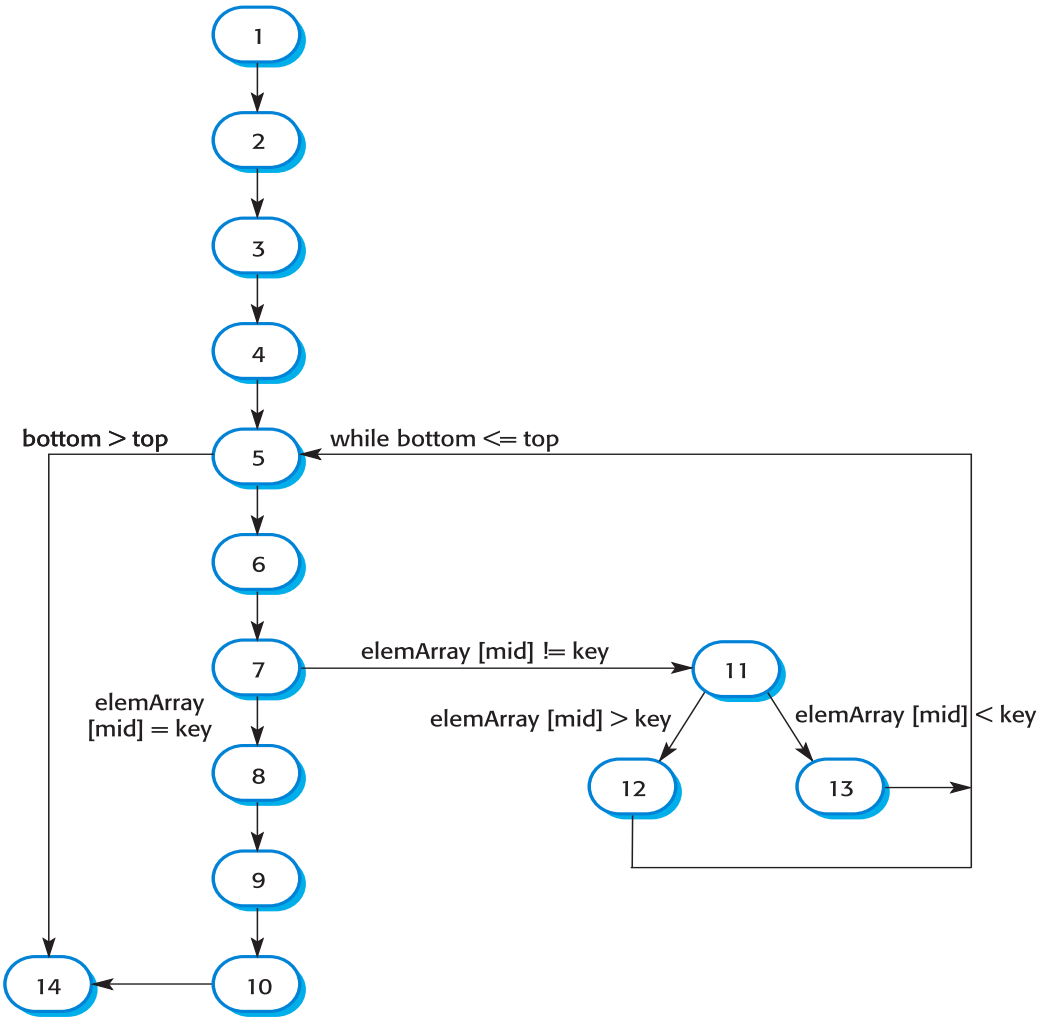
Điều này đưa đến một tập sửa lại của các trường hợp thử nghiệm cho thủ tục tìm kiếm, như trên hình 3.15. Chú ý, đã sửa đổi mảng đầu vào vì vậy nó đã được sắp xếp theo thứ tự tăng dần và đã thêm các thử nghiệm có phần tử khóa kề với phần tử giữa của mảng.

3.8.4. Kiểm thử đường dẫn

Kiểm thử đường dẫn là một chiến lược kiểm thử cấu trúc. Mục tiêu của kiểm thử đường dẫn là thực hiện mọi đường dẫn thực hiện độc lập thông qua một thành phần hoặc chương trình. Nếu mọi đường dẫn thực hiện độc lập được thực hiện, thì tất cả các câu lệnh trong thành phần đó phải được thực hiện ít nhất một lần. Hơn nữa, tất cả câu lệnh điều kiện phải được kiểm thử với cả trường hợp đúng và sai. Trong quá trình phát triển hướng đối tượng, kiểm thử đường dẫn có thể được sử dụng khi kiểm thử các phương thức liên kết với các đối tượng.

Số lượng đường dẫn qua một chương trình thường tỷ lệ với kích thước của nó. Khi tất cả các môđun được tích hợp trong hệ thống, nó trở nên không khả thi để sử dụng kỹ thuật kiểm thử cấu trúc. Vì thế, kỹ thuật kiểm thử đường dẫn hầu như được sử dụng trong lúc kiểm thử thành phần.

Kiểm thử đường dẫn không kiểm tra tất cả các kết hợp có thể của của các đường dẫn qua chương trình. Với bất kỳ thành phần nào ngoài các thành phần rất tầm thường không có vòng lặp, đây là mục tiêu không khả thi. Trong chương trình có các vòng lặp sẽ có một số vô hạn khả năng kết hợp đường dẫn. Thậm chí, khi tất cả các lệnh của chương trình đã được thực hiện ít nhất một lần, các khiếm khuyết của chương trình vẫn có thể được đưa ra khi các đường dẫn đặc biệt được kết hợp.



Hình 2.16 Đồ thị luồng của chương trình tìm kiếm nhị phân

Điểm xuất phát để kiểm thử đường dẫn là đồ thị luồng chương trình. Đây là mô hình khung của tất cả đường dẫn qua chương trình. Một đồ thị luồng chứa các nút miêu tả các quyết định và các cạnh trình bày luồng điều khiển. Đồ thị luồng được xây dựng bằng cách thay đổi các câu lệnh điều khiển chương trình sử dụng biểu đồ tương đương. Nếu không có các câu lệnh goto trong chương trình, đó là một quá trình đơn giản xuất phát từ đồ thị luồng. Mỗi nhánh trong câu lệnh điều kiện (if-then-else hoặc case) được miêu tả như một đường dẫn riêng biệt. Mỗi mũi tên trở lại nút điều kiện miêu tả một vòng lặp. Tôi đã vẽ đồ thị luồng cho phương thức tìm kiếm nhị phân trên hình 3.16. Để tạo nên sự tương ứng giữa đồ thị này và chương trình trên hình 3.14 được rõ ràng, tôi đã miêu tả mỗi câu lệnh như một nút riêng biệt, các số trong mỗi nút tương ứng với số dòng trong chương trình.

Mục đích của kiểm thử đường dẫn là đảm bảo mỗi đường dẫn độc lập qua chương trình được thực hiện ít nhất một lần. Một đường dẫn chương trình độc lập là một đường đi ngang qua ít nhất một cạnh mới trong đồ thị luồng. Cả nhánh đúng và nhánh sai của các điều kiện phải được thực hiện.

Đồ thị luồng cho thủ tục tìm kiếm nhị phân được miêu tả trên hình 3.16, mỗi nút biểu diễn một dòng trong chương trình với một câu lệnh có thể thực hiện được. Do đó, bằng

cách lần vết trên đồ thị luồng, bạn có thể nhận ra các đường dẫn qua đồ thị luồng tìm kiếm nhị phân:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14

1, 2, 3, 4, 5, 14

1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …

1, 2, 3, 4, 5, 6, 7, 11, 13, 5, …

Nếu tất cả các đường dẫn được thực hiện, chúng ta có thể đảm bảo mọi câu lệnh trong phương thức đã được thực hiện ít nhất một lần và mỗi nhánh đã được thực hiện với các điều kiện đúng và sai.

Bạn có thể tìm được số lượng các đường dẫn độc lập trong một chương trình bằng tính toán vòng liên hợp (McCabe, 1976) trong đồ thị luồng chương trình. Với chương trình không có câu lệnh goto, giá trị vòng liên hợp là nhiều hơn số câu lệnh điều kiện trong chương trình. Một điều kiện đơn là một biểu thức lôgíc không có các liên kết “and” hoặc “or”. Nếu chương trình bao gồm các điều kiện phức hợp, là các biểu thức lôgíc bao gồm các liên kết “and” và “or”, thì bạn phải đếm số điều kiện đơn trong các điều kiện phức hợp khi tính số vòng liên hợp.

Vì vậy, nếu có 6 câu lệnh “if” và 1 vòng lặp “while” và các biểu thức điều kiện là đơn, thì số vòng liên hợp là 8. Nếu một biểu thức điều kiện là biểu thức phức hợp như “if A and B or C”, thì bạn tính nó như 3 điều kiện đơn. Do đó, số vòng liên hợp là 10. Số vòng liên hợp của thuật toán tìm kiếm nhị phân (hình 3.14) là 4 bởi vì nó có 3 điều kiện đơn tại các dòng 5, 7, 11.

Sau khi tính được số đường dẫn độc lập qua mã chương trình bằng tính toán số vòng liên hợp, bạn thiết kế các trường hợp thử nghiệm để thực hiện mỗi đường dẫn đó. Số lượng trường hợp thử nghiệm nhỏ nhất bạn cần để kiểm tra tất cả các đường dẫn tỏng chương trình bằng số vòng liên hợp.

Thiết kế trường hợp thử nghiệm không khó khăn trong trường hợp chương trình là thủ tục tìm kiếm nhị phân. Tuy nhiên, khi chương trình có cấu trúc nhánh phức tạp, có thể rất khó khăn để dự đoán có bao nhiêu thử nghiệm đã được thực hiện. Trong trường hợp đó, một người phân tích chương trình năng động có thể được sử dụng để phát hiện sơ thảo sự thực thi của chương trình.

Những người phân tích chương trình năng động là các công cụ kiểm thử, cùng làm việc với trình biên dịch. Trong lúc biên dịch, những người phân tích này thêm các chỉ thị phụ để sinh ra mã. Chúng đếm số lần mỗi câu lệnh đã được thực hiện. Sau khi chương trình đã thực hiện, một bản sơ thảo thực thi có thể được in ra. Nó chỉ ra những phần chương trình đã thực thi và đã không thực thi bằng cách sử dụng các trường hợp thử nghiệm đặc biệt. Vì vậy, bản sơ thảo thực thi cho phép phát hiện các phần chương trình không được kiểm thử.

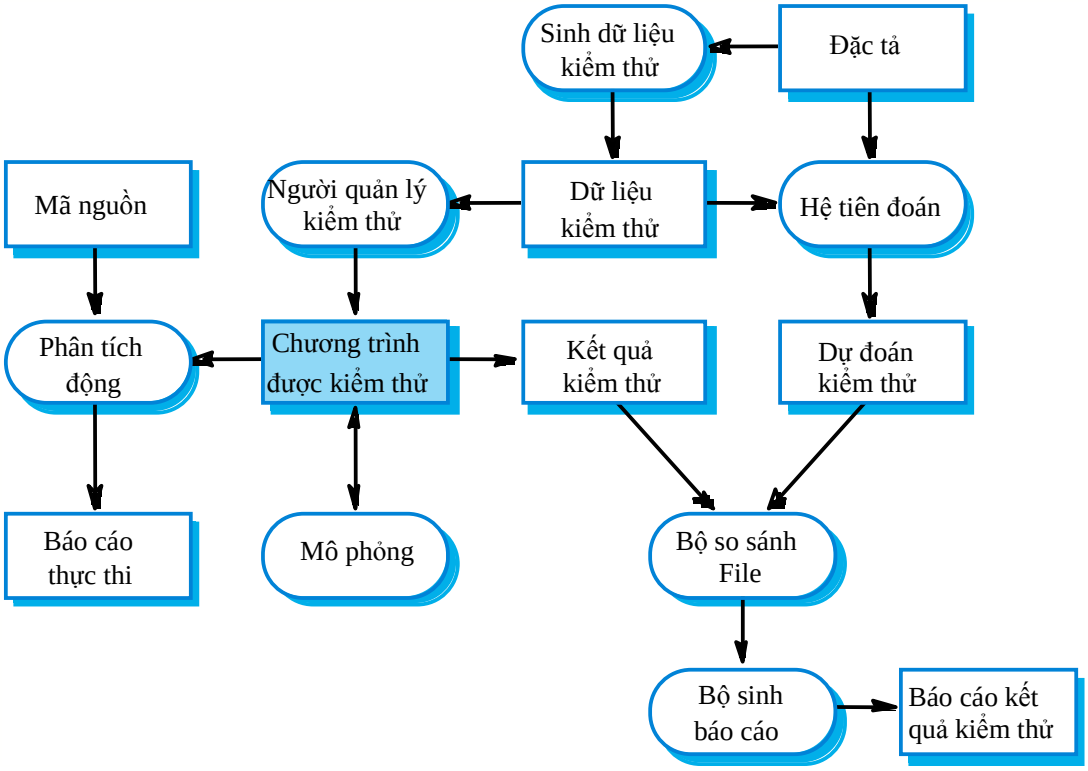
3.9. Tự động hóa kiểm thử (Test automation)

Kiểm thử là một giai đoạn tốn kém và nặng nề trong quy trình phần mềm. Kết quả là những công cụ kiểm thử là một trong những công cụ phần mềm đầu tiên được phát triển. Hiện nay, các công cụ này đã bộc lộ nhiều sự thuận tiện và chúng làm giảm đáng kể chi phí kiểm thử.

Tôi đã thảo luận một cách tiếp cận để tự động hóa kiểm thử (Mosley và Posey, 2002) với một khung kiểm thử như JUnit (Massol và Husted, 2003) được sử dụng kiểm thử phục hồi. JUnit là một tập các lớp Java được người dùng mở rộng để tạo nên môi trường kiểm thử tự động. Mỗi thử nghiệm riêng lẻ được thực hiện như một đối tượng và một chương trình đang chạy thử nghiệm chạy tất cả các thử nghiệm đó. Các thử nghiệm đó nên được viết theo cách để chúng chỉ ra hệ thống kiểm thử có thực hiện như mong muốn không.

Một phần mềm kiểm thử workbench là một tập tích hợp các công cụ để phục vụ cho quá trình kiểm thử. Hơn nữa với các khung kiểm thử cho phép thực hiện kiểm thử tự động, một workbench có thể bao gồm các công cụ để mô phỏng các phần khác của hệ thống và để sinh ra dữ liệu thử nghiệm hệ thống. Hình 3.17 đưa ra một vài công cụ có thể bao gồm trong một workbench kiểm thử:

1. Người quản lý kiểm thử: quản lý quá trình chạy các thử nghiệm. Họ giữ vết của dữ liệu thử nghiệm, các kết quả mong đợi và chương trình dễ dàng kiểm thử. Các khung kiểm tụ động hóa thử nghiệm như JUnit là ví dụ của các người quản lý thử nghiệm.
2. Máy sinh dữ liệu thử nghiệm: sinh các dữ liệu để thử nghiệm chương trình. Điều này có thể thực hiện bằng cách lựa chọn dữ liệu từ cơ sở dữ liệu hoặc sử dụng các mẫu để sinh ngẫu nhiên dữ liệu với khuôn dạng đúng đắn.
3. Hệ tiên đoán (Oracle): đưa ra các dự đoán về kết quả kiểm thử mong muốn. Các hệ tiên đoán có thể là phiên bản trước của chương trình hoặc hệ thống bản mẫu. Kiểm thử back-to-back , bao gồm việc thực hiện kiểm thử song song hệ tiên đoán và chương trình đó. Các khác biệt trong các đầu ra của chúng được làm nổi bật.
4. Hệ so sánh tập tin: so sánh các kết quả thử nghiệm chương trình với các kết quả thử nghiệm trước đó và báo cáo các khác biệt giữa chúng. Các hệ so sánh được sử dụng trong kiểm thử hồi quy (các kết quả thực hiện trong các phiên bản khác nhau được so sánh). Khi kiểm thử tự động được sử dụng, hệ so sánh có thể được gọi từ bên trong các kiểm thử đó.
5. Hệ sinh báo cáo: cung cấp các báo cáo để xác định và đưa ra các tiện lợi cho kết quả thử nghiệm.
6. Hệ phân tích động: thêm mã vào chương trình để đếm số lần mỗi câu lệnh đã được thực thi. Sau khi kiểmthử, một bản sơ thảo thực thi được sinh ra sẽ cho biết mỗi câu lệnh trong chương trình đã được thực hiện bao nhiêu lần.
7. Hệ mô phỏng (Simulator): Các loại hệ mô phỏng khác nhau có thể được cung cấp. Mục đích của các hệ mô phỏng là mô phỏng các máy khi chương trình được thực thi. Hệ mô phỏng giao diện người dùng là các chương trình điều khiển kịch bản mô phỏng nhiều tương tác đồng thời của người dùng. Sử dụng hệ mô phỏng cho I/O có nghĩa là bộ định thời gian của dãy giao dịch có thể được lặp đi lặp lại.



Hình 3.17 Một workbench kiểm thử

Khi sử dụng cho kiểm thử hệ thống lớn, các công cụ đó phải được định dạng và phù hợp với hệ thống cụ thể. Ví dụ:

1. Các công cụ mới có thể được thêm vào để kiểm thử các đặc trưng ứng dụng cụ thể, một vài công cụ hiện có có thể không cần đến.
2. Các kịch bản có thể được viết cho hệ mô phỏng giao diện người dùng và các mẫu đã xác định cho hệ sinh dữ liệu thử nghiệm. Các khuôn dạng báo cáo có thể cũng phải được xác định.
3. Các tập kết quả thử nghiệm mong muốn có thể phải chuẩn bị bằng tay nếu không một phiên bản chương trình nào trước đó có thể dùng được như một hệ tiên đoán.
4. Hệ so sánh tập tin mục đích đặc biệt có thể được viết bao gồm hiểu biết về cấu trúc của kết quả thử nghiệm trên tập tin.

Một lượng lớn thời gian và công sức thường cần để tạo nên một workbench thử nghiệm toàn diện. Do đó, các workbench hoàn chỉnh, như trên hình 3.17, chỉ được sử dụng khi phát triển các hệ thống lớn. Với các hệ thống đó, toàn bộ chi phí kiểm thử có thể lên tới 50% tổng giá trị phát triển, vì vậy, nó là hiệu quả để đầu tư cho công cụ chất lượng cao CASE hỗ trợ việc kiểm thử. Tuy nhiên, vì các loại hệ thống khác nhau yêu cầu sự hỗ trợ các loại kiểm thử khác nhau, các công cụ kiểm thử có thể không sãn có để dùng. Rankin (Rankin, 2002) đã thảo luận một tình huống trong IBM và miêu tả thiết kế của hệ thống hỗ trợ kiểm thử, mà họ đã phát triển cho máy chủ kinh doanh điện tử.

Các điểm chính:

- Kiểm thử có thể chỉ ra sự hiện diện của các lỗi trong chương trình. Nó không thử chứng tỏ không còn lỗi trong chương trình.
- Kiểm thử thành phần là trách nhiệm của người phát triển thành phần. Một đội kiểm thử khác thường thực hiện kiểm thử hệ thống.
- Kiểm thử tích hợp là hoạt động kiểm thử hệ thống ban đầu khi bạn kiểm thử khiếm khuyết của các thành phần tích hợp. Kiểm thử phát hành liên quan đến kiểm thử của khách hàng và kiểm thử phát hành nên xác nhận hệ thống được phân phối có đầy đủ các yêu cầu.
- Khi kiểm thử hệ thống, bạn nên có gắng "phá" hệ thống bằng cách sử dụng kinh nghiệm và các nguyên tắc để lựa chọn các kiểu thử nghiệm có hiệu quả để phát hiện khiếm khuyết trong hệ thống.
- Kiểm thử giao diện dùng để phát hiện các khiếm khuyết trong giao diện của các thành phần hỗn hợp. Các khiếm khuyết trong giao diện có thể nảy sinh bởi lỗi trong khi đọc các đặc tả chương trình, hiểu sai các đặc tả chương trình, các lỗi khác hoặc do thừa nhận bộ đếm thời gian không hợp lệ.
- Phân hoạch tương đương là một cách xác định các thử nghiệm. Nó phụ thuộc vào việc xác định các phân hoạch trong tập dữ liệu đầu vào và đầu ra, sự thực hiện chương trình với các giá trị từ các phân hoạch đó. Thông thường, các giá trị đó là giá trị tại giới hạn của phân hoạch.
- Kiểm thử cấu trúc dựa trên phân tích chương trình để phát hiện đường dẫn qua chương trình và sử dụng những phân tích để lựa chọn các thử nghiệm.
- Tự động hóa thử nghiệm làm giảm chi phí kiểm thử bằng cách hỗ trợ quá trình kiểm thử bằng cách công cụ phần mềm.

CHƯƠNG 4: CÁC PHƯƠNG PHÁP KIỂM THỬ

Chương này tập trung vào các kỹ thuật để tạo ra các trường hợp kiểm thử tốt và ít chi phí nhất, tất cả chúng phải thoả những mục tiêu kiểm thử ở chương trước. Nhắc lại các mục tiểu kiểm thử phần mềm là thiết kế các trường hợp kiểm thử có khả năng tìm kiếm nhiều lỗi nhất trong phần mềm và với ít thòi gian và công sức nhất.

Hiện tại phát triển rất nhiều phương thức thiết kế các trường hợp kiểm thử cho phần mềm. Những phương pháp này đều cung cấp một hướng kiểm thử có tính hệ thống. Qua trọng hơn nữa là chúng cung cấp một hệ thống có thể giúp đảm bảo sự hoàn chỉnh của các trường hợp kiểm thử phát hiên lỗi cho phần mềm.

Một sản phẩm đều có thể được kiểm thử theo 2 cách:

- Hiểu rõ một chức năng cụ thể của một hàm hay một module. Các trường hợp kiểm thử có thể xây dựng để kiểm thử tất cả các thao tác đó.
- Hiểu rõ cách hoạt động của một hàm/module hay sản phẩm. Các trường hợp kiểm thử có thể được xây dựng để đảm bảo tất cả các thành phần con khớp với nhau. Đó là tất cả các thao tác nội bộ của hàm dựa vào các mô tả và tất cả các thành phần nôi bộ đã được kiểm thủ một cách thoả đáng.

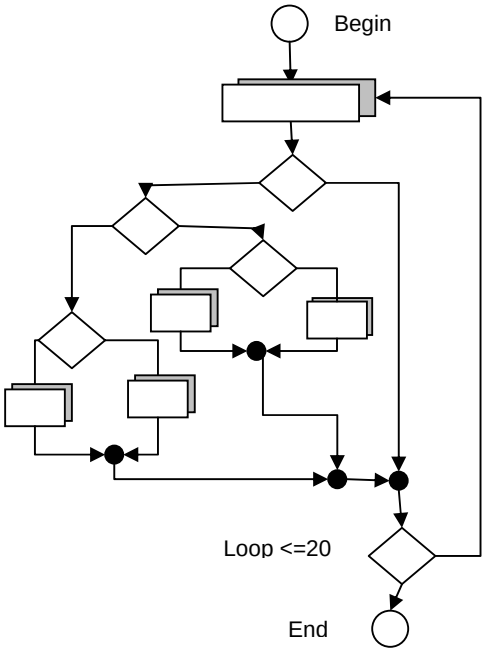
Cách tiếp cận đầu tiên được gọi là kiểm thử hộp đen (black box testing) và cách tiếp cận thứ hai là gọi là kiểm thử hộp trắng (white box testing).

Khi đề cập đến kiểm thử phần mềm, black box testing còn được biết như là kiểm thử ở mức giao diện (interface). Mặc dù thật sự thì chúng được thiết kế để phát hiện lỗi. Black box testing còn được sử dùng để chứng minh khả năng hoạt động của hàm hay module chương trình và có thể cả một chương trình lớn: các thông số đầu vào được chấp nhận như mô tả của hàm, giá trị trả về cũng hoạt đông tốt, đảm bảo các dữ liệu từ bên ngoài ví dụ như file dữ liệu được giữ/đảm bảo tính nguyên vẹn của dữ liệu khi thực thi hàm.

While box testing là kỹ thuật tập trung vào khảo sát chặc chẻ thủ tục một cách chi tiết. Tất cả những đường diễn tiến logic trong chương trình được kiểm tra bằng những trường hợp kiểm thử kiểm tra trên các tập điều kiện và cấu trúc lặp cụ thể. kỹ thuật này sẽ kiểm tra trạng thái của chương trình tại rất nhiều điểm trong chương trình nhằm xác giá trị mong đợi tại các điểm nay có khớp với giá trị thực tế hay không.

Với tất cả các mục tiêu kiểm định trên thì kỹ thuật while box testing có lẽ sẻ dẫn đến một chương trình chính xác tuyệt đối. Tất cả những gì chúng ta cần bây giờ là thiết kế tất cả các đường logic của chương trình và sau đó là cài đặt tất cả các trường hợp kiểm định có được. Tuy nhiên việc kiểm định một cách thấu đáo tất cả các trường hợp là một bài toán quá lớn và tốn rất nhiều chi phi. Chúng ta hay xem xét ví dụ sau

Hình 4.1. FlowChart



Bên trái là flowchart cho một chương trình đơn giản được viết bằng khoản 100 dòng mã với một vòng lặp chính thực thi đoạn mã bên trong và lăp lại không quá 20 lần. Tuy nhiên khi tính toán cho thấy đối với chương trình này có đến khoảng 10^{14} đường có thể được thực hiện.

Chúng ta làm tiếp một phép tính nhanh để thấy được chi phí dùng để kiểm thử đoạn chương trình nay một cách thấu đáo và chi tiết. Ta giả sử rằng để kiểm định một trường hợp cần chạy trung bình tồn một giây. Và chương trình kiểm thử sẽ được chạy 24 giờ một ngày và chạy suốt 365 ngày một năm. Vây thì để chạy kiểm thử cho tất cả các trường hợp này cũng cần phải tốn khoản 3170 năm.

Do đó kiểm thử một cách thấu đáo là một việc bất khả thi cho những hệ thống lớn.

Mặc dù kỹ thuật này không thể hiện thực được trong thực tế với lượng tài nguyên có hạn, tuy nhiên với một số lượng có giới hạn các đường diễn tiến logic quan trọng có chọn lựa trước để kiểm thử. Phương pháp này có thể là rất khả thi

Ngoài ra các trường hợp kiểm thử còn có thể là sự kết hợp của cả hai kỹ thuật trên nhằm đạt được các mục tiêu của việc kiểm thử.

Và bây giờ chúng ta sẽ đi và chi tiết thảo luận về kỹ thuật kiểm thử hộp trắng

4.1. Phương pháp white-box:

Là phương pháp kiểm nghiệm dựa vào cấu trúc/mã lệnh chương trình. Phương pháp white-box kiểm nghiệm một chương trình (một phần chương trình, hay một hệ thống, một phần của hệ thống) đáp ứng tốt tất cả các giá trị input bao gồm cả các giá trị không đúng hay không theo dự định của chương trình.

Phương pháp kiểm nghiệm white-box dựa trên:

- Các câu lệnh (statement)
- Đường dẫn (path)
- Các điều kiện (condition)
- Vòng lặp (loop)
- Ngã rẽ (branch)

4.1.1 Mô tả một số cấu trúc theo lược đồ:

Trong các phương pháp kiểm tra tính đúng đắn của chương trình, lược đồ được dùng để:

- Trừu tượng hóa cú pháp của mã lệnh.