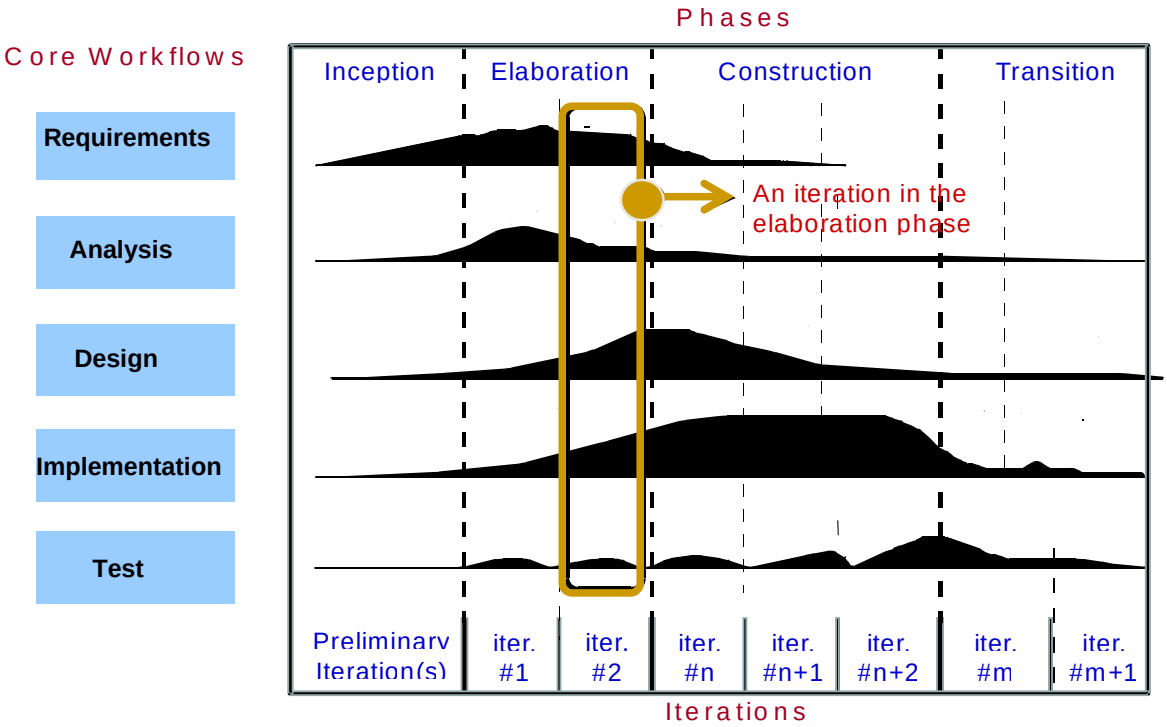
Chương 1

Tổng quát về kiểm thử phần mềm

1.1 Qui trình phát triển phần mềm RUP



Chu kỳ phần mềm được tính từ lúc có yêu cầu (mới hoặc nâng cấp) đến lúc phần mềm đáp ứng đúng yêu cầu được phân phối.

Trong mỗi chu kỳ, người ta tiến hành nhiều công đoạn : khởi động, chi tiết hóa, hiện thực và chuyển giao.

Mỗi công đoạn thường được thực hiện theo cơ chế lặp nhiều lần để kết quả ngày càng hoàn hảo hơn.

Trong từng bước lặp, chúng ta thường thực hiện nhiều workflows đồng thời (để tận dụng nguồn nhân lực hiệu quả nhất) : nắm bắt yêu cầu, phân tích chức năng, thiết kế, hiện thực và kiểm thử.

Sau mỗi lần lặp thực hiện 1 công việc nào đó, ta phải tạo ra kết quả (artifacts), kết quả của bước/công việc này là dữ liệu đầu

vào của bước/công việc khác. Nếu thông tin không tốt sẽ ảnh hưởng nghiêm trọng đến kết quả của các bước/hoạt động sau đó.

Một số vấn đề thường gặp trong phát triển phần mềm :

- tính toán không đúng, hiệu chỉnh sai dữ liệu.
- trộn dữ liệu không đúng.
- Tìm kiếm dữ liệu sai yêu cầu.
- Xử lý sai mối quan hệ giữa các dữ liệu.
- Coding/hiện thực sai các qui luật nghiệp vụ.
- Hiệu suất của phần mềm còn thấp.
- Kết quả hoặc hiệu suất phần mềm không tin cậy.
- Hỗ trợ chưa đủ các nhu cầu nghiệp vụ.
- Giao tiếp với hệ thống khác chưa đúng hay chưa đủ.
- Kiểm soát an ninh phần mềm chưa đủ.

1.2 Vài định nghĩa về kiểm thử phần mềm

- Kiểm thử phần mềm là qui trình chứng minh phần mềm không có lỗi.
- Mục đích của kiểm thử phần mềm là chỉ ra rằng phần mềm thực hiện đúng các chức năng mong muốn.
- Kiểm thử phần mềm là qui trình thiết lập sự tin tưởng về việc phần mềm hay hệ thống thực hiện được điều mà nó hỗ trợ.
- Kiểm thử phần mềm là qui trình thi hành phần mềm với ý định tìm kiếm các lỗi của nó.
- Kiểm thử phần mềm được xem là qui trình cố gắng tìm kiếm các lỗi của phần mềm theo tinh thần "hủy diệt".

Các mục tiêu chính của kiểm thử phần mềm :

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

Kiểm thử phần mềm là 1 thành phần trong lĩnh vực rộng hơn, đó là Verification & Validation (V &V), ta tạm dịch là Thanh kiểm tra và Kiểm định phần mềm.

Thanh kiểm tra phần mềm là qui trình xác định xem sản phẩm của 1 công đoạn trong qui trình phát triển phần mềm có thoả mãn các yêu cầu đặt ra trong công đoạn trước không (Ta có đang xây dựng đúng đắn sản phẩm không ?)

Thanh kiểm tra phần mềm thường là hoạt động kỹ thuật vì nó dùng các kiến thức về các artifacts, các yêu cầu, các đặc tả rời rạc của phần mềm.

Các hoạt động Thanh kiểm tra phần mềm bao gồm kiểm thử (testing) và xem lại (reviews).

Kiểm định phần mềm là qui trình đánh giá phần mềm ở cuối chu kỳ phát triển để đảm bảo sự bằng lòng sử dụng của khách hàng (Ta có xây dựng phần mềm đúng theo yêu cầu khách hàng ?).

Các hoạt động kiểm định được dùng để đánh giá xem các tính chất được hiện thực trong phần mềm có thỏa mãn các yêu cầu khách hàng và có thể theo dõi với các yêu cầu khách hàng không ?

Kiểm định phần mềm thường phụ thuộc vào kiến thức của lĩnh vực mà phần mềm xử lý.

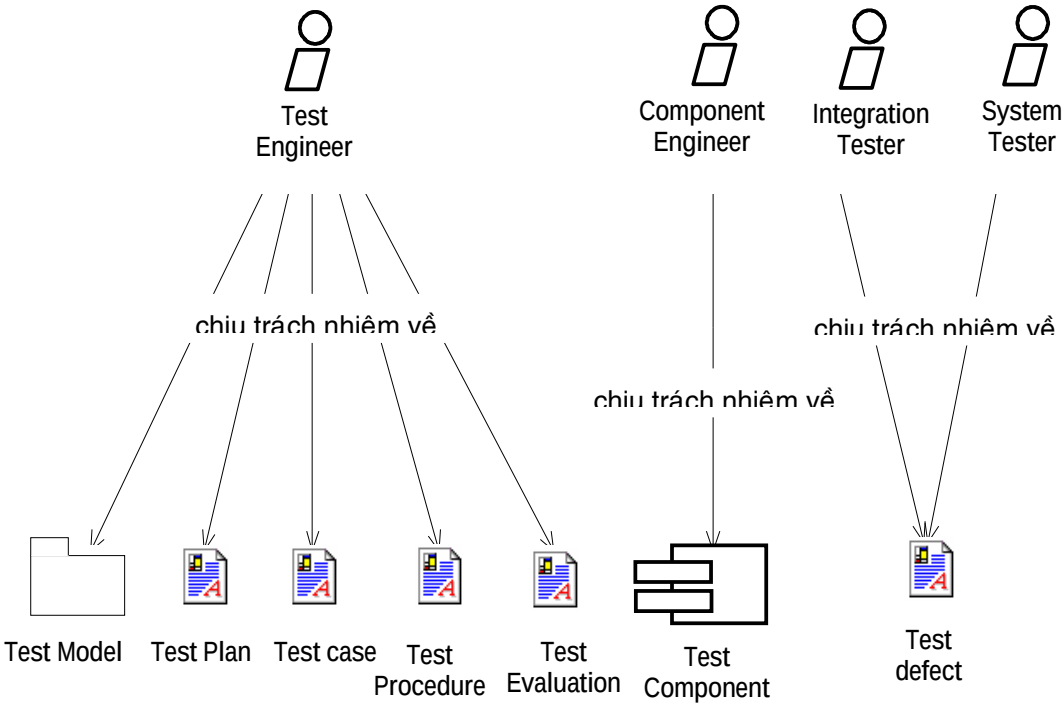
Verification
Are we building the product **right**?

"I landed on "Go" but didn't get my $200!"

Validation
Are we building the **right** product?

"I know this game has money and players and "Go" – but this is not the game I wanted."

1.3 Kiểm thử : các worker và qui trình



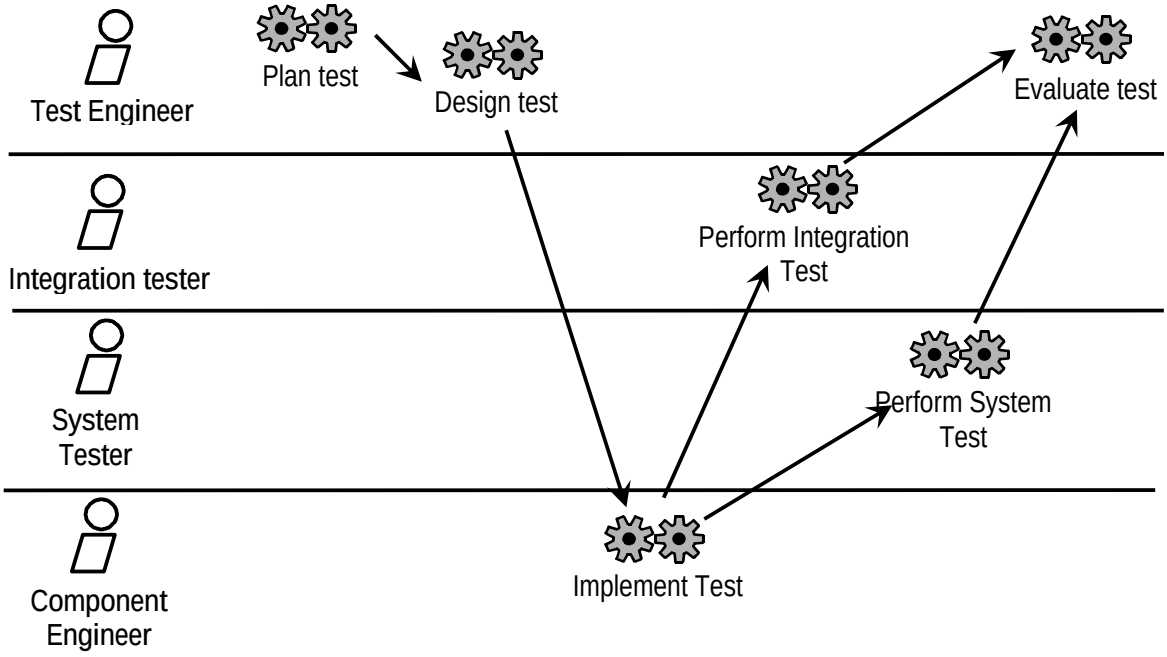
Kỹ sư kiểm thử :

- người chuyên về IT, chịu trách nhiệm về nhiều hoạt động kỹ thuật liên quan đến kiểm thử.

- định nghĩa các testcase, viết các đặc tả và thủ tục kiểm thử.
- phân tích kết quả, báo cáo kết quả cho các người phát triển và quản lý biết.
- ...

Người quản lý kiểm thử :

- Thiết lập chiến lược và qui trình kiểm thử, tương tác với các người quản lý về các hoạt động khác trong project, giúp đỡ kỹ sư kiểm thử thực hiện công việc của họ.



Tự động 1 số hoạt động kiểm thử

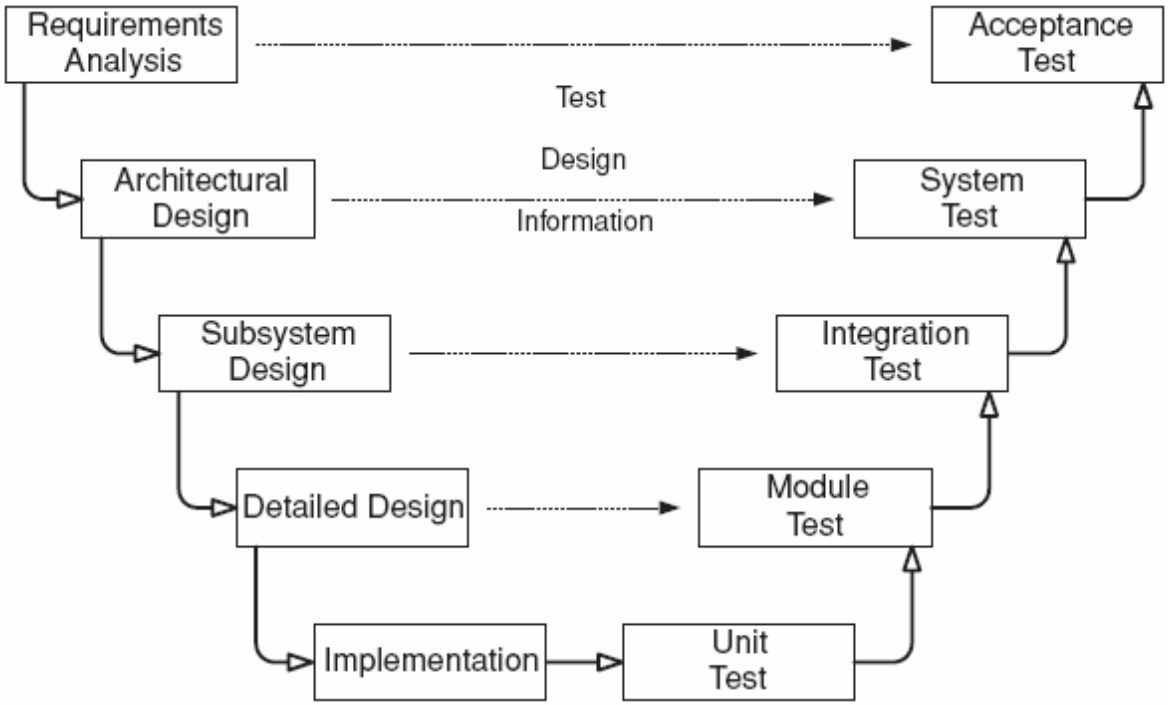
Kiểm thử phần mềm tốn nhiều chi phí nhân công, thời gian. Trong 1 số dự án, kiểm thử phần mềm tiêu hao trên 50% tổng giá phát triển phần mềm. Nếu cần ứng dụng an toàn hơn, chi phí kiểm thử còn cao hơn nữa.

Do đó 1 trong các mục tiêu của kiểm thử là tự động hóa nhiều như có thể, nhờ đó mà giảm thiểu chi phí rất nhiều, tối thiểu hóa các lỗi do người gây ra, đặc biệt giúp việc kiểm thử hồi qui dễ dàng và nhanh chóng hơn.

Tự động hóa việc kiểm thử là dùng phần mềm điều khiển việc thi hành kiểm thử, so sánh kết quả có được với kết quả kỳ vọng, thiết lập các điều kiện đầu vào, các kiểm soát kiểm thử và các chức năng báo cáo kết quả...

Thí dụ các tiện ích phục vụ tự động kiểm thử như : Stress Test, Selenium, TestComplete, IBM Rational Functional Tester.

1.4 Các mức độ kiểm thử phần mềm



- Kiểm thử đơn vị (Unit Testing) : kiểm thử sự hiện thực chi tiết của từng đơn vị nhỏ (hàm, class,...) có hoạt động đúng không ?
- Kiểm thử module (Module Testing) : kiểm thử các dịch vụ của module có phù hợp với đặc tả của module đó không ?
- Kiểm thử tích hợp (Integration Testing) : kiểm thử xem từng phân hệ của phần mềm có đảm bảo với đặc tả thiết kế của phân hệ đó không ?
- Kiểm thử hệ thống (System Testing) : kiểm thử các yêu cầu không chức năng của phần mềm như hiệu suất, bảo mật, làm việc trong môi trường căng thẳng,...

- Kiểm thử độ chấp nhận của người dùng (Acceptance Testing) : kiểm tra xem người dùng có chấp thuận sử dụng phần mềm không ?
- Kiểm thử hồi qui : được làm mỗi khi có sự hiệu chỉnh, nâng cấp phần mềm với mục đích xem phần mềm mới có đảm bảo thực hiện đúng các chức năng trước khi hiệu chỉnh không ?

1.5 Testcase

Mỗi testcase chứa các thông tin cần thiết để kiểm thử thành phần phần mềm theo 1 mục tiêu xác định. Thường testcase gồm bộ 3 thông tin {tập dữ liệu đầu vào, trạng thái của thành phần phầm mềm, tập kết quả kỳ vọng}

Tập dữ liệu đầu vào (Input): gồm các giá trị dữ liệu cần thiết để thành phần phầm mềm dùng và xử lý.

Tập kết quả kỳ vọng : kết quả mong muốn sau khi thành phần phần mềm xử lý dữ liệu nhập.

Trạng thái thành phần phần mềm : được tạo ra bởi các giá trị prefix và postfix.

Tập các testcase : tập hợp các testcase mà ta có ý định dùng để kiểm thử thành phần phần mềm để minh chứng rằng TPPM có đúng các hành vi mong muốn.

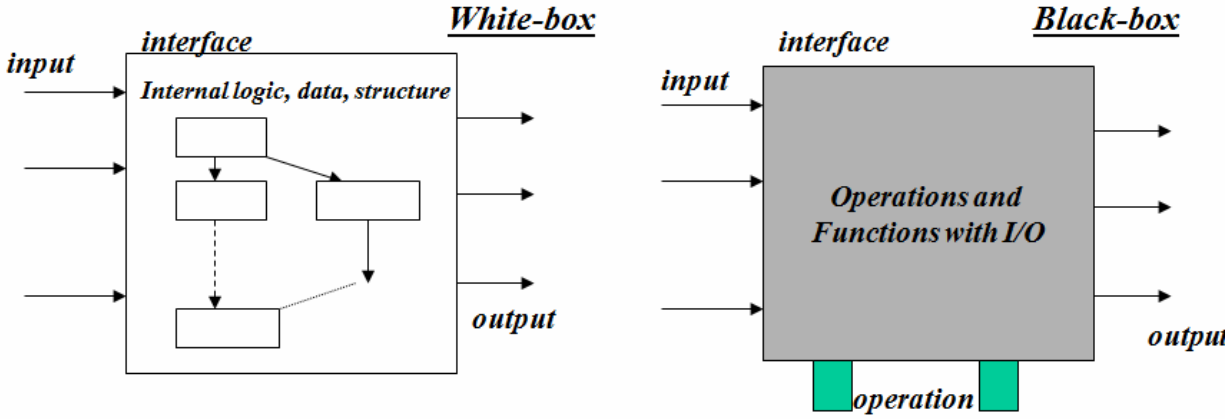
Các phương pháp thiết kế testcase

Bất kỳ sản phẩm kỹ thuật nào (phần mềm không phải là ngoại lệ) đều có thể được kiểm thử bởi 1 trong 2 cách :

- Kiểm thử hộp đen (Black box testing) : theo góc nhìn sử dụng
  - Không cần kiến thức về chi tiết thiết kế và hiện thực bên trong.

- Kiểm thử dựa trên các yêu cầu và đặc tả sử dụng TPPM.

- Kiểm thử hộp trắng (White box testing) : theo góc nhìn hiện thực
  - cần kiến thức về chi tiết thiết kế và hiện thực bên trong.
  - Kiểm thử dựa vào phủ các lệnh, phủ các nhánh, phủ các điều kiện con,...



Kiểu kiểm thử	Kỹ thuật kiểm thử được dùng
Unit Testing	White Box, Black Box
Integration Testing	Black Box, White Box
Functional Testing	Black Box
System Testing	Black Box
Accceptance Testing	Black Box

1.6 Các nguyên tắc cơ bản về kiểm thử

Thông tin thiết yếu của mỗi testcase là kết quả hay dữ liệu xuất kỳ vọng.

Nếu kết quả kỳ vọng của testcase không được định nghĩa rõ ràng, người ta sẽ giải thích kết quả sai (plausible) thành kết quả đúng bởi vì hiện tượng “the eye seeing what it wants to see.”

=> 1 test case phải chứa 2 thành phần thiết yếu :

- đặc tả về điều kiện dữ liệu nhập.
- đặc tả chính xác về kết quả đúng của chương trình tương ứng với dữ liệu nhập.

Việc kiểm thử đòi hỏi tính độc lập : lập trình viên nên tránh việc kiểm thử các TPPM do mình viết.

Các issues tâm lý :

- Chương trình có thể chứa các lỗi do lập trình viên hiểu sai về đặc tả/phát biểu vấn đề.
- Tổ chức lập trình không nên kiểm thử các chương trình của tổ chức mình viết.
- Thanh tra 1 cách xuyên suốt các kết quả kiểm thử.

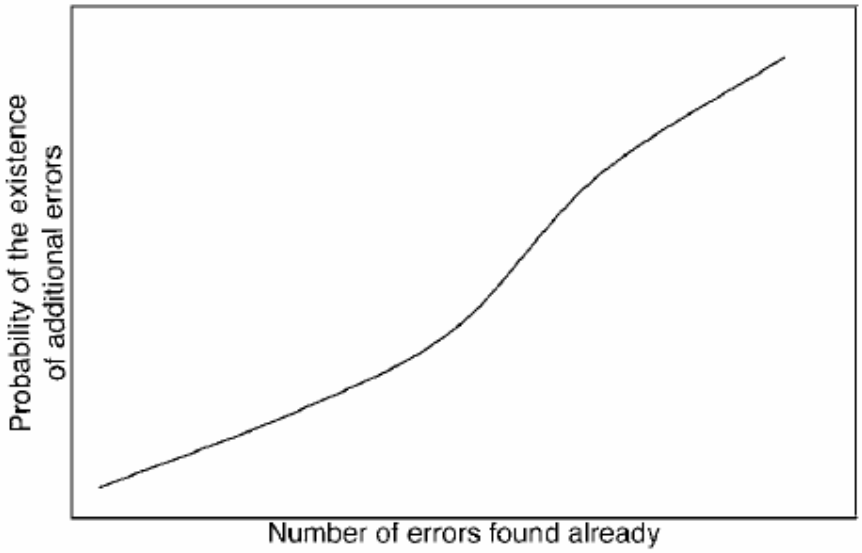
Phải thiết kế đủ các test case cho cả 2 trường hợp : dữ liệu đầu vào hợp lệ và dữ liệu đầu vào không hợp lệ và chờ đợi.

Xem xét chương trình xem nó không thực hiện những điều mong muốn, xem nó có làm những điều không mong muốn ?

Tránh các testcase "throwaway" trừ phi chương trình thật sự là "throwaway".

Không nên lập kế hoạch nỗ lực kiểm thử dựa trên giả định ngầm rằng phần mềm không có lỗi.

Xác xuất xuất hiện nhiều lỗi hơn trong 1 section phần mềm tỉ lệ thuận với số lỗi đã phát hiện được trong section đó.



Kiểm thử là 1 tác vụ rất thách thức đòi hỏi sự sáng tạo và trí tuệ.

Kiểm thử phần mềm nên bắt đầu từ các thành phần nhỏ đơn giản rồi đến các thành phần ngày càng lớn hơn.

Kiểm thử theo kiểu vét cạn là không thể.

Nên hoạch định qui trình kiểm thử trước khi bắt đầu thực hiện kiểm thử.

1.7 Các ý tưởng không đúng về kiểm thử

- Ta có thể kiểm thử phần mềm đầy đủ, nghĩa là đã vét cạn mọi hoạt động kiểm thử cần thiết.
- Ta có thể tìm tất cả lỗi nếu kỹ sư kiểm thử làm tốt công việc của mình.
- Tập các testcase tốt phải chứa rất nhiều testcase để bao phủ rất nhiều tình huống.
- Testcase tốt luôn là testcase có độ phức tạp cao.
- Tự động kiểm thử có thể thay thế kỹ sư kiểm thử để kiểm thử phần mềm 1 cách tốt đẹp.
- Kiểm thử phần mềm thì đơn giản và dễ dàng. Ai cũng có thể làm, không cần phải qua huấn luyện.

1.8 Các hạn chế của việc kiểm thử

- Ta không thể chắc là các đặc tả phần mềm đều đúng 100%.
- Ta không thể chắc rằng hệ thống hay tool kiểm thử là đúng.
- Không có tool kiểm thử nào thích hợp cho mọi phần mềm.
- Kỹ sư kiểm thử không chắc rằng họ hiểu đầy đủ về sản phẩm phần mềm.
- Ta không bao giờ có đủ tài nguyên để thực hiệm kiểm thử đầy đủ phần mềm.
- Ta không bao giờ chắc rằng ta đạt đủ 100% hoạt động kiểm thử phần mềm.

1.9 Kết chương

Chương này đã ôn lại qui trình phát triển phần mềm được dùng phổ biến nhất hiện nay, đó là qui trình RUP (Rational Unified Process), từ đó giới thiệu các lý do cần phải kiểm thử phần mềm, các thuật ngữ cơ bản trong hoạt động kiểm thử phần mềm.

Chương này cũng đã giới thiệu vai trò của các worker trong qui trình kiểm thử phần mềm, các mức độ kiểm thử phần mềm khác nhau, các nguyên tắc cơ bản của kiểm thử phần mềm.

Chương này cũng đã giới thiệu 1 số ý tưởng không đúng đắn về kiểm thử phần mềm, những hạn chế của hoạt động kiểm thử phần mềm.

Chương 2

Qui trình & Kế hoạch kiểm thử phần mềm

2.1 Giới thiệu

1. Qui trình kiểm thử phần mềm là gì ?

- Chế độ kiểm thử được định nghĩa bởi tổ chức phát triển phần mềm là gì.
- Cần có chiến lược kiểm thử và nó sẽ lý giải tại sao tổ chức phần mềm kiểm thử các thành phần mà mình tạo ra.
- Cần nhận dạng cái gì là quan trọng đối với tổ chức (chi phí, chất lượng, thời gian, phạm vi,..) và cách nào, bởi ai và khi nào việc kiểm thử sẽ được thực hiện.
- Tất cả các thông tin trên sẽ được lập thành tài liệu cho hoạt động kiểm thử và ta có thể gọi qui trình tạo lập tài liệu này là qui trình kiểm thử phần mềm (Test Process).

cuu duong than cong . com

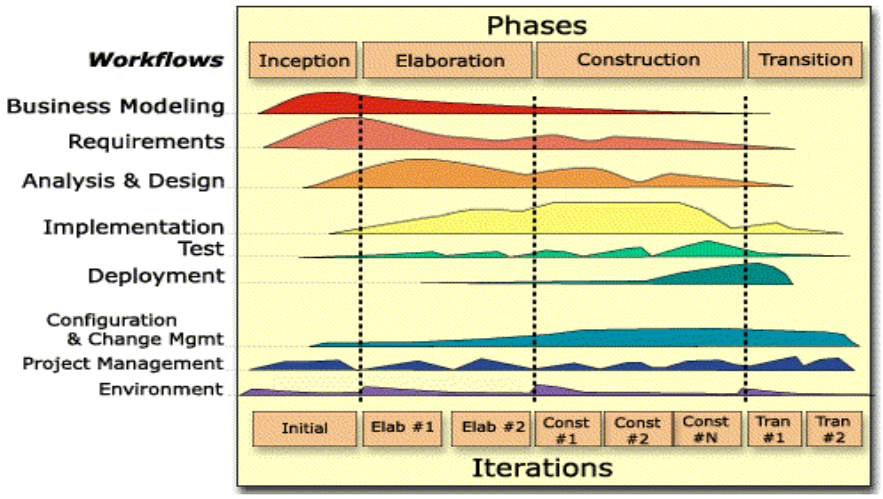
2. Tạo sao cần phải thực hiện qui trình kiểm thử phần mềm ?

- Cần làm rõ vai trò và trách nhiệm của việc kiểm thử phần mềm.
- Cần làm rõ các công đoạn, các bước kiểm thử.
- Cần phải hiểu và phân biệt các tính chất kiểm thử (tạo sao phải kiểm thử), các bước kiểm thử (khi nào kiểm thử), và các kỹ thuật kiểm thử (kiểm thử bằng cách nào).

3. Chúng ta phải kiểm thử phần mềm khi nào ?

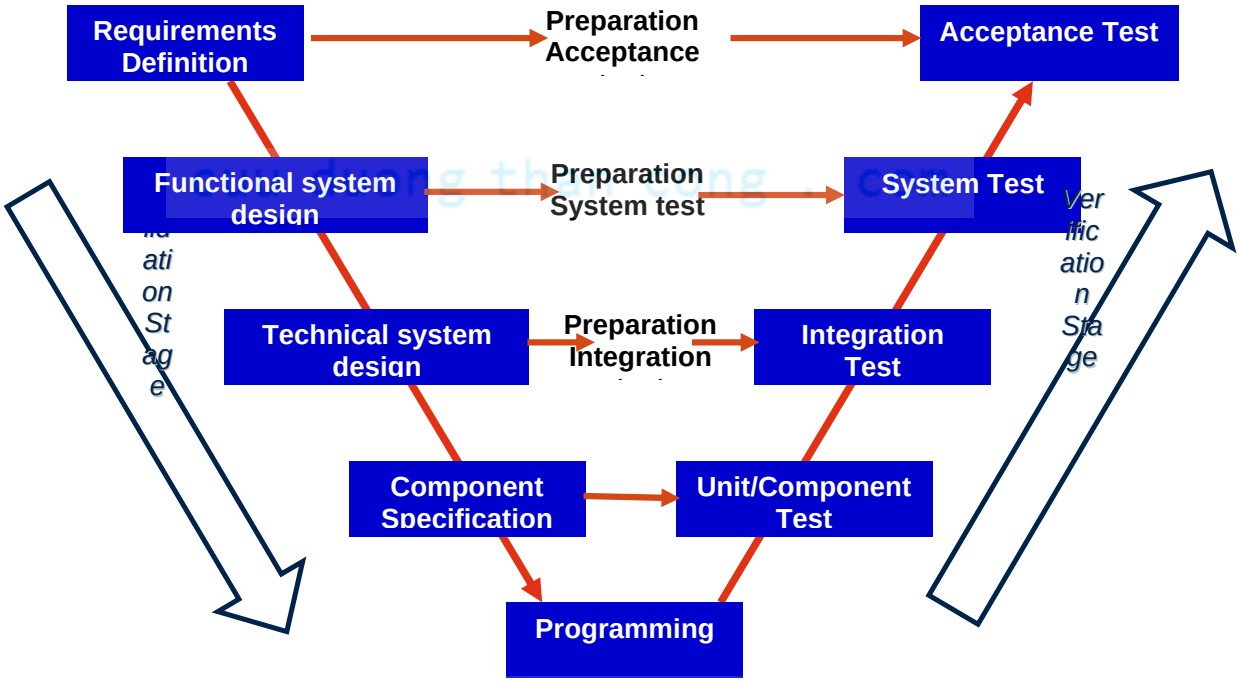
cuu duong than cong . com

RUP Life Cycle



Kiểm thử sẽ được thực hiện sau mỗi bước lặp.

Mô hình phát triển và kiểm thử phần mềm hình chữ V



Các tính chất cần ghi nhận trên mô hình chữ V :

- Các hoạt động hiện thực và các hoạt động kiểm thử được tách biệt nhưng độ quan trọng là như nhau.
- Chữ V minh họa các khía cạnh của hoạt động Verification và Validation.

- Cần phân biệt giữa các mức độ kiểm thử ở đó mỗi mức kiểm thử là kiểm thử trên mức phát triển phần mềm tương ứng.

Mô hình phát triển tăng tiến-tương tác :

- Qui trình thiết lập các yêu cầu phần mềm, thiết kế, xây dựng, kiểm thử hệ thống phần mềm được thực hiện như 1 chuỗi các chu kỳ phát triển ngắn hơn.
- Hệ thống có được từ 1 bước lặp được kiểm thử ở nhiều cấp trong việc phát triển hệ thống đó.
- Kiểm thử hồi quy có độ quan trọng tăng dần theo các bước lặp (không cần trong bước đầu tiên).
- Thanh kiểm tra và kiểm định có thể được thực hiện theo kiểu tăng dần trên từng bước lặp.

Các tính chất của qui trình kiểm thử tốt :

cuu duong than cong . com

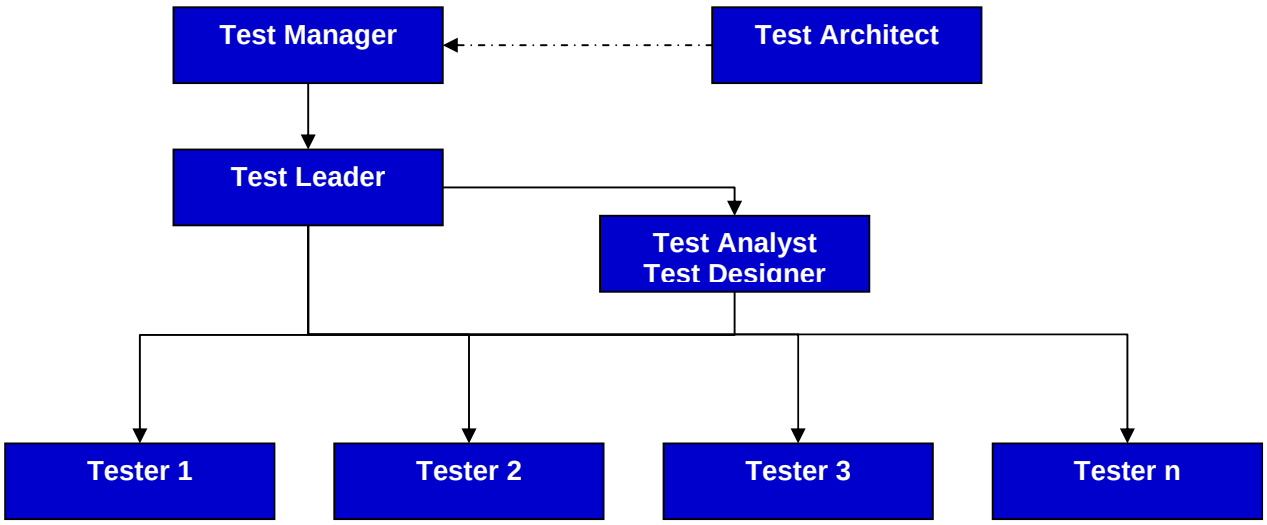
- Cần có 1 mức độ kiểm thử cho mỗi công đoạn phát triển phần mềm.
- Các mục tiêu kiểm thử sẽ bị thay đổi, mỗi mức kiểm thử nên có các mục tiêu đặc thù của mình.
- Việc phân tích và thiết kế testcase cho 1 mức độ kiểm thử nên bắt đầu sớm nhất như có thể có.
- Các tester nên xem xét các tài liệu sớm như có thể có, ngay sau khi các tài liệu này được tạo ra trong chu kỳ phát triển phần mềm.

cuu duong than cong . com

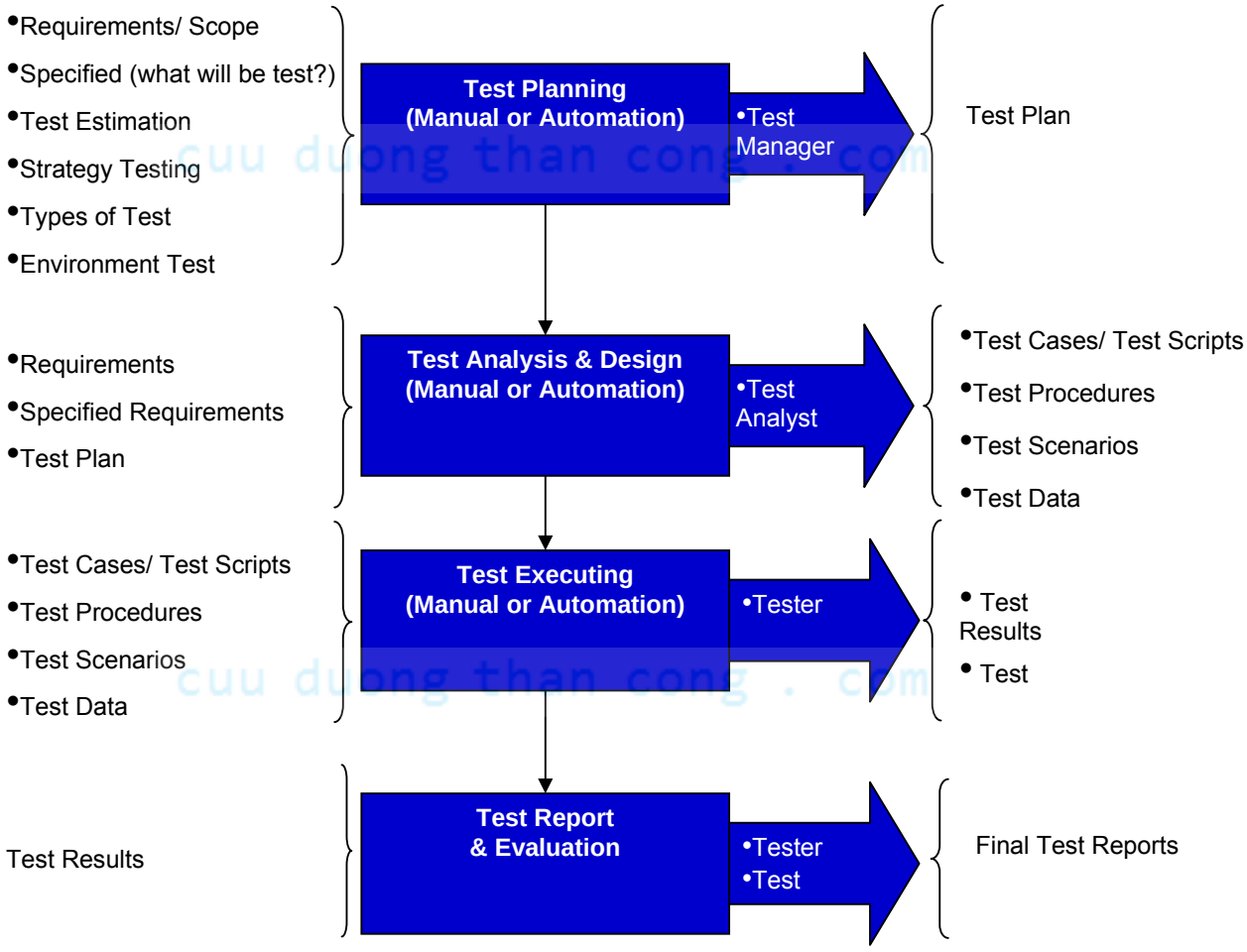
- Số lượng và cường độ của các mức kiểm thử được điều khiển theo các yêu cầu đặc thù của project phần mềm đó.

Sơ đồ tổ chức phổ biến của đội kiểm thử

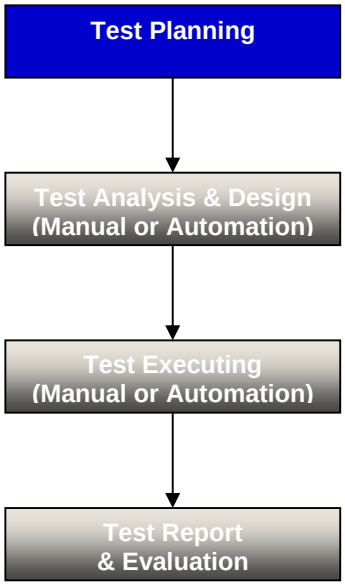
4. Ai liên quan đến việc kiểm thử phần mềm ?



2.2 Qui trình kiểm thử tổng quát



Xây dựng kế hoạch kiểm thử



Test Manager hoặc Test Leader sẽ xây dựng kế hoạch ban đầu về kiểm thử.

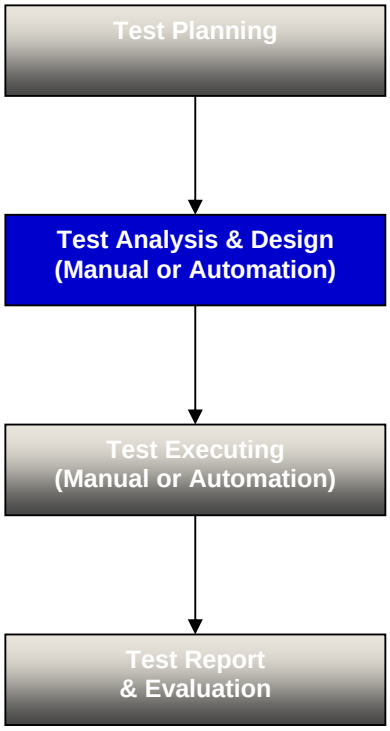
- Định nghĩa phạm vi kiểm thử
- Định nghĩa các chiến lược kiểm thử
- Nhận dạng các rủi ro và yếu tố bất ngờ
- Nhận dạng các hoạt động kiểm thử nào là thủ công, kiểm thử nào là tự động hay cả hai.
- Ước lượng chi phí kiểm thử và xây dựng lịch kiểm thử.
- Nhận dạng môi trường kiểm thử.
- ...

Kế hoạch kiểm thử cần được :

- xem lại bởi QC team, Developers, Business Analysis. TA (if need), PM and Customer
- Chấp thuận bởi : Project Manager and Customer
- Hiệu chỉnh trong suốt chu kỳ kiểm thử để phản ánh các thay đổi nếu cần thiết.

Phân tích & thiết kế kiểm thử

cuu duong than cong . com



Test Analyst hoặc Test Designer sẽ thiết kế (định nghĩa) các testcase từ các yêu cầu liên quan (thí dụ từ thông tin trong usecase).

cuu duong than cong . com

- sẽ thiết kế (định nghĩa) các testcase từ các yêu cầu chức năng và các yêu cầu không chức năng của phần mềm.
- Các testcase cần bao phủ tất cả khía cạnh kiểm thử cho từng yêu cầu phần mềm.
- Các testcase cần bao phủ tất cả yêu cầu trong các chiến lược kiểm thử.
- Nếu cần kiểm thử tự động, Test Designer sẽ xây dựng các kịch bản dựa trên các testcase/Test procedures.

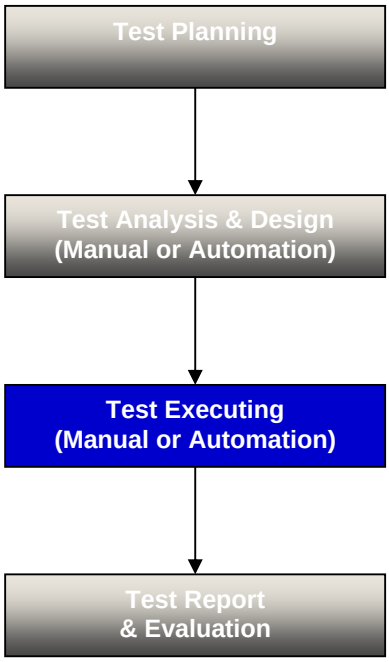
cuu duong than cong . com

Các testcase cần được :

- Xem xét lại bởi Project Leader, Developer có liên quan, các Testers khác, Test Leader, Business Analysis và Customer.
- Chấp thuận bởi Test Leader hoặc Customer

- Hiệu chỉnh/cập nhật nếu Tester đã tìm được những lỗi mà không nằm trong các testcase hiện có.

Thi hành kiểm thử

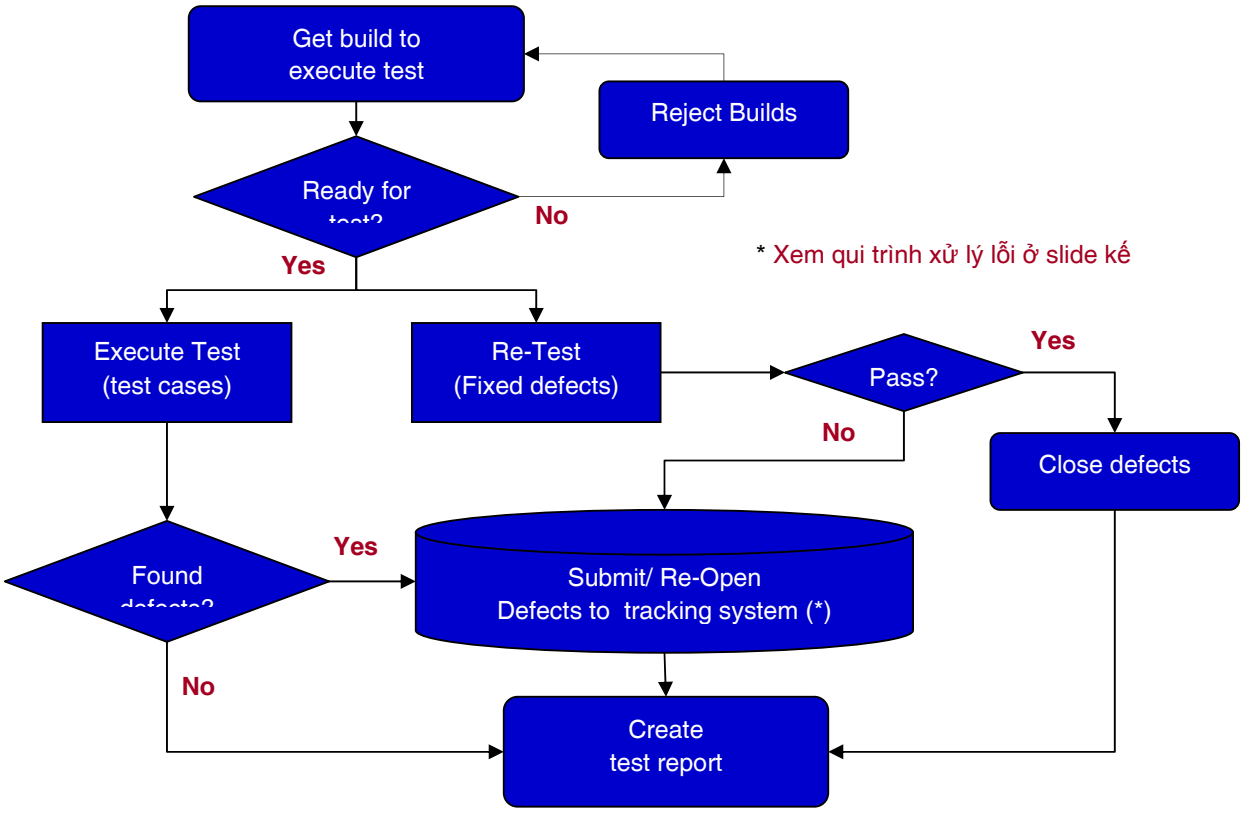


Testers sẽ được bố trí công việc bởi Test Leader để thi hành kiểm thử.

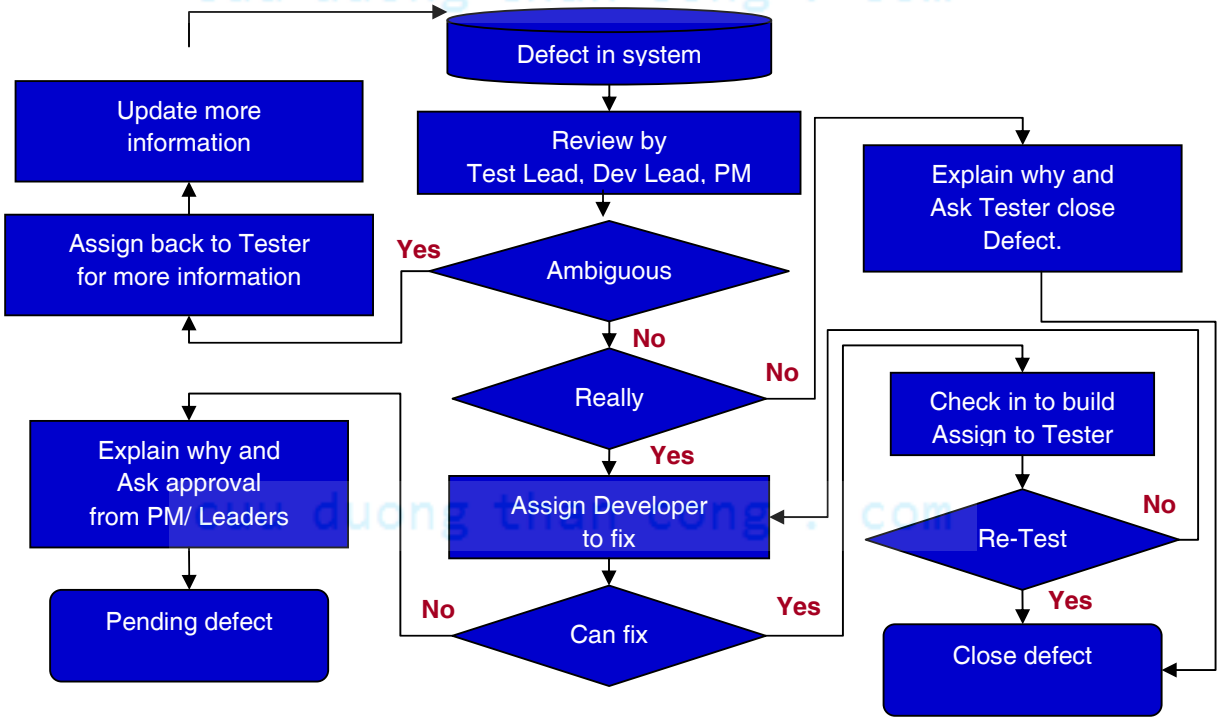
- Thi hành kiểm thử theo từng testcase.
- Thực hiện kiểm thử đặc biệt (ad-hoc)
- Thực hiện kịch bản kiểm thử mà không được định nghĩa trong testcase.
- Kiểm thử lại các lỗi đã được sửa.
- Tester sẽ tạo các báo cáo về lỗi trong suốt quá trình kiểm lỗi và theo dõi chúng cho đến khi chúng đã được xử lý.
- Ở công đoạn kiểm thử độ chấp thuận, Customer sẽ thi hành kiểm thử để kiểm định xem hệ thống phần mềm có thỏa mãn các nhu cầu người dùng không ?

Test Execution Workflow

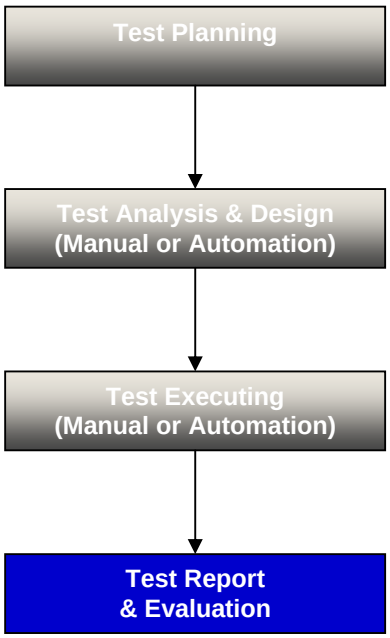
CuuDuongThanCong.com https://fb.com/tailieudientucntt



Defects Workflow



Test Report and Evaluation



Test Manager hoặc Test Leader sẽ phân tích các lỗi trong hệ thống theo dõi các lỗi.

- Tạo các báo cáo lỗi.
- Đánh giá các kết quả kiểm thử, thống kê các yêu cầu thay đổi.
- Tính và phân phối các thông tin đo lường hoạt động kiểm thử.
- Tạo bảng tổng kết đánh giá hoạt động kiểm lỗi.
- Xác định xem đã đạt tiêu chí thành công và hoàn thành kiểm thử chưa.

2.3 Kế hoạch kiểm thử

1. Định nghĩa : Kế hoạch kiểm thử thường được để trong 1 file và chứa các kết quả của các hoạt động sau :

- Nhận dạng các chiến lược được dùng để kiểm tra và đảm bảo rằng sản phẩm thỏa mãn đặc tả thiết kế phần mềm và các yêu cầu khác về phần mềm.
- Định nghĩa các mục tiêu và phạm vi của nỗ lực kiểm thử.

CuuDuongThanCong.com https://fb.com/tailieudientucntt

- Nhận dạng phương pháp luận mà đội kiểm thử sẽ dùng để thực hiện công việc kiểm thử.
- Nhận dạng phần cứng, phần mềm và các tiện ích cần cho kiểm thử.
- Nhận dạng các tính chất và chức năng sẽ được kiểm thử.
- Xác định các hệ số rủi ro gây nguy hại cho việc kiểm thử.
- Lập lịch kiểm thử và phân phối công việc cho mỗi thành viên tham gia.
- …

Test Manager hoặc Test Leader sẽ xây dựng kế hoạch kiểm thử.

2. Nhu cầu cần phải có kế hoạch kiểm thử : Kế hoạch kiểm thử cần phải được xây dựng sớm như có thể có trong mỗi chu kỳ phát triển phần mềm để :

cuu duong than cong . com

- Tập hợp và tổ chức các thông tin kiểm thử cần thiết.
- Cung cấp thông tin về qui trình kiểm thử sẽ xảy ra trong tổ chức kiểm thử.
- Cho mỗi thành viên trong đội kiểm thử có hướng đi đúng.
- Gán các trách nhiệm rõ ràng cụ thể cho mỗi thành viên đội kiểm thử.
- Có lịch biểu làm việc rõ ràng và các thành viên có thể làm việc với nhau tốt.

cuu duong than cong . com

3. Kế hoạch kiểm thử cần chứa các thông tin sau đây :

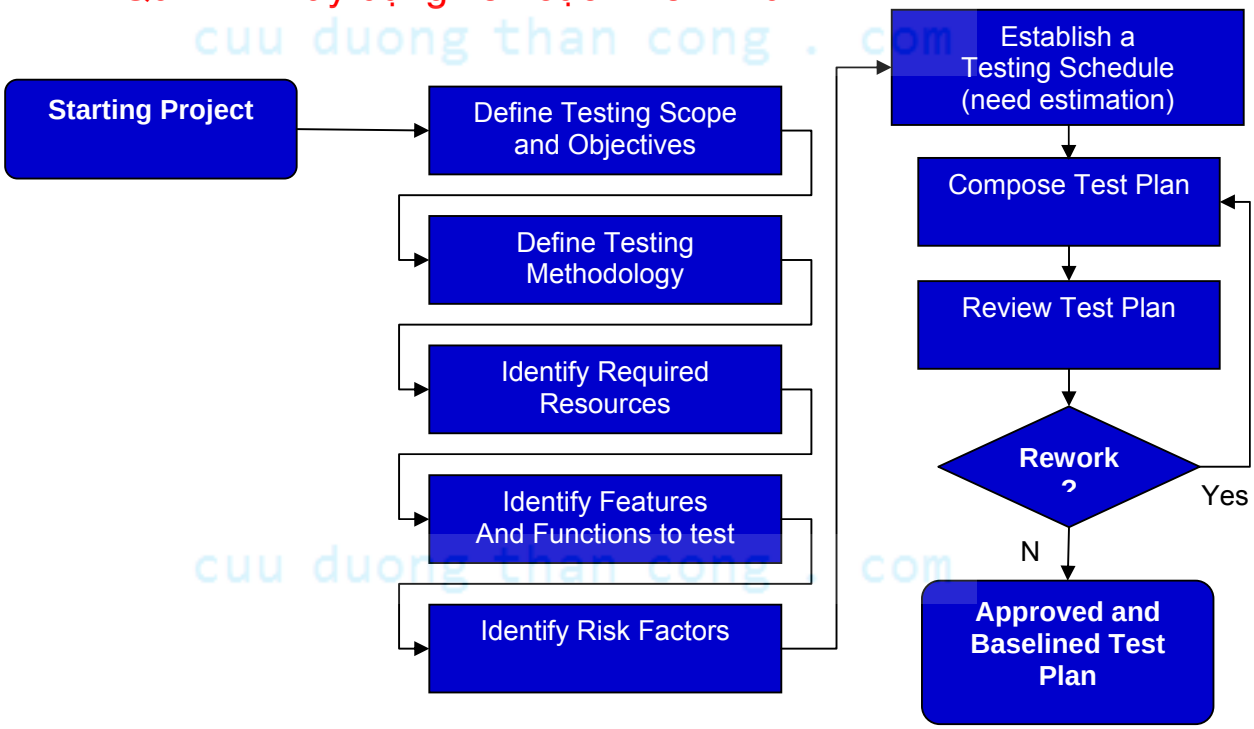
- Phạm vi/mục tiêu kiểm thử
- Các chiến lược được dùng
- Các tài nguyên phần cứng và phần mềm phục vụ kiểm thử.
- Các nhu cầu về nhân viên và huấn luyện nhân viên.

- Các tính chất cần được kiểm thử.
- Các tính chất không cần kiểm thử.
- Các rủi ro & sự cố bất ngờ.
- Lịch kiểm thử cụ thể.
- Các kênh thông tin liên lạc.
- Cấu hình cho từng phần tử như kế hoạch kiểm thử, testcase, thủ tục kiểm thử,...
- Môi trường kiểm thử (Test bed)
- Tiêu chí đầu vào và tiêu chí dừng kiểm thử.
- Các kết quả phân phối.

Test Plan Workflow

4. Qui trình xây dựng kế hoạch kiểm thử :

cuu duong than cong . com



cuu duong than cong . com

Ghi chú quan trọng :

Sau khi xây dựng xong kế hoạch kiểm thử, ta có thể thay đổi nó nhưng phải tuân thủ qui trình yêu cầu thay đổi.

Main activities

5. Các hoạt động chính trong việc xây dựng kế hoạch kiểm thử :

- Định nghĩa mục đích, phạm vi, chiến lược, cách tiếp cận, các điều kiện chuyển, các rủi ro, kế hoạch giảm nhẹ và tiêu chí chấp thuận.
- Định nghĩa cách thức thiết lập môi trường và các tài nguyên được dùng cho việc kiểm thử.
- Thiết lập cơ chế theo dõi lỗi phát hiện.
- Chuẩn bị ma trận theo dõi bao phủ mọi yêu cầu phần mềm.
- Báo cáo trạng thái kiểm thử.
- Phát hành leo thang (Escalating Issues)
- Raising Testing related PIR (Process Improvement Request) / PCR (Process Change Request)

cuu duong than cong . com

2.4 Các thành phần chính trong kế hoạch kiểm thử

1. Mục đích và phạm vi kiểm thử :

- Đặc tả mục đích của tài liệu về kế hoạch kiểm thử.
- Cung cấp vắn tắt về phạm vi mà project được hỗ trợ như platform, loại database, hay danh sách vắn tắt về các loại project con in project kiểm thử.

cuu duong than cong . com

- Thí dụ :

Purpose

The purpose of this document is to give stakeholders, as well as team members, a detailed quality assurance plan covering test requirements for the release. Test plan objectives include, but are not limited to:

- Identify existing project information and the software components that should be tested
- List the recommended test requirements
- Recommend and describe the testing strategies to be employed
- Include traceability from Requirement to Test Case
- List the deliverable elements of the test project

Testing scope

This section to provide test requirements, strategies as below:

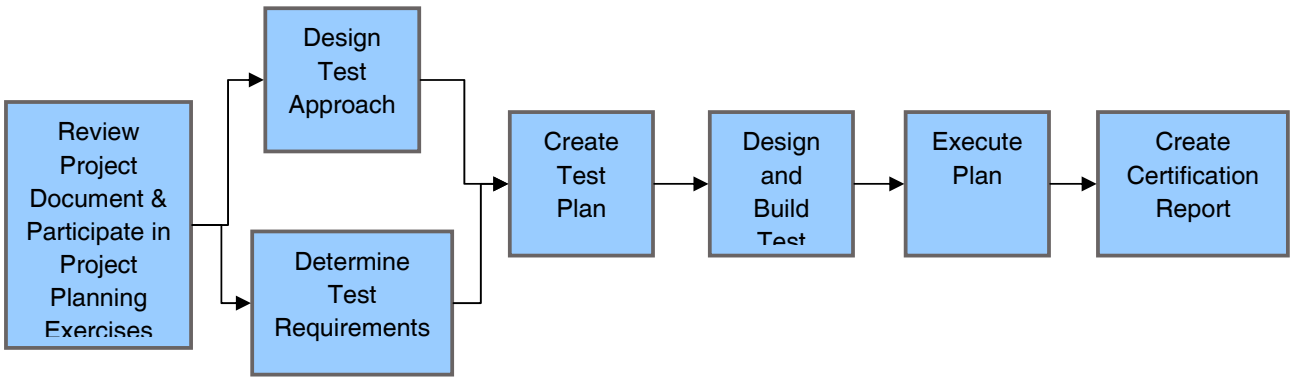
- Operation will be tested: Windows XP SP2, SP3 + Latest security updated from Microsoft.
- Database type: Microsoft SQL Server 2005
- Browsers: Internet Explorer 7
- The sub-products will be tested as below:
  - Quality Monitoring 9.0 SP3
  - Agent Capture
  - UST/BUIT
  - Media Testing
  - Documents verification
  - Installation/Upgrade testing

cuu duong than cong . com

2. Cách tiếp cận & các chiến lược được dùng :

- Đặc tả về phương phạm luận kiểm thử sẽ được dùng để thực hiện kiểm thử.
- Thí dụ : General Testing Process Approach for Project ABC

cuu duong than cong . com



Đề cập các cấp độ kiểm thử cần thực hiện

Các kỹ thuật được dùng cho mỗi kiểu kiểm thử trong project :

- Kiểm thử tích hợp (Integration Testing)
- Kiểm thử hệ thống (System Testing)
- Kiểm thử độ chấp thuận (Acceptance Testing)
- Kiểm thử chức năng của người dùng (Functionality Testing)
- Kiểm thử hồi qui (Regression Testing)
- Kiểm thử việc phục hồi sau lỗi (Failover and Recovery Testing)
- Kiểm thử việc kiểm soát an minh và truy xuất (Security and Access Control Testing)
- Kiểm thử việc cấu hình và cài đặt (Configuration and Installation Testing)
- Kiểm thử đặc biệt (Ad-hoc Testing)
- Kiểm thử hiệu suất (Performance Testing)

3. Các tính chất cần được kiểm thử :

- Danh sách các tính chất của phần mềm cần được kiểm thử, đây là 1 catalog chứa tất cả các testcase (bao gồm

cuu duong than cong . com

chỉ số testcase, tiêu đề testcase) cũng như tất cả trạng thái cơ bản.

- Thí dụ :

New Feature and Requirements

List all features in scope. Requirement to Test Case Traceability will be updated in Quality Center.

Planned Iteration	Req ID	Requirement	Test Case(s) (should map to QC)	Reviewer
1	ABC-2001	Support ABC Client in 64-bit Citrix/TS	ABC-2001 - Support ABC Client in 64-bit Citrix/TS	
1	ABC-2002	Enhance Supervisor Installation for JRE Selection	ABC-2002 - Enhance Supervisor Installation for JRE Selection	
2	ABC-1065	Dynamic Workspaces	ABC-1065 - Dynamic Workspaces	
3	ABC-1108	Configurable JRE Versioning	ABC-1108 - Configurable JRE Versioning	
3	ABC-1093	Remove DCOM Dependency	ABC-1093 - Remove DCOM Dependency	
3	ABC-1106	DSE (VTG) Card Replacement	ABC-1106 - DSE (VTG) Card Replacement	

cuu duong than cong . com

Regression

ID	Test Case Name (Should Map to QC)	Automated?
123	__FTC - Abandoned Call	N
124	__FTC - Basic Call	N
125	__FTC - Conference Call	N
126	__FTC - Consultation Call	N
127	__FTC - Race Condition	N
128	__FTC - Transfer Call	N
454	__Inters - Playback	N
455	__Inters - Agent List Security	N

cuu duong than cong . com

4. Các tính chất không cần được kiểm thử :

- Danh sách các vùng phần mềm được loại trừ khỏi kiểm thử, cũng như các testcase đã được định nghĩa nhưng không cần kiểm thử.
- Thí dụ :

De-scoped Test Cases / Out of Scope Testing

ID	Test Case	Reason
	General	Low priority
	Install	Low priority
	Error Handling	Low priority
	JRE	Low priority
	Native Player	Low priority
	Mini Regression	Low priority

5. Rủi ro và các sự cố bất ngờ

- Danh sách tất cả rủi ro có thể xảy ra trong chu kỳ kiểm thử.
- Phương pháp mà ta cần thực hiện để tối thiểu hóa hay sống chung với rủi ro.
- Thí dụ :

cuu duong than cong . com

#	Risks	Contingency Plan	Level
1	Requirement changes affect to human resources and test strategy.	Need to do re-planning with a realistic schedule to deal with changed requirement: either extend schedule or/and increase resource if need to execute testing for new requirement or previous requirement if changing requirements effect on previous requirements	High
2	Build is not ready on time as project schedule.	Development team must follow schedule to create build strictly. If not, the escalation will be issued by Quality Control Lead	High
3	Requirements are not base-lined on time.	Onshore team must follow-up to get requirements baselined on time.	High
4	Late response on issues	The feedback must be provided in 5 working days when an issue is raised.	Medium
5	Virus effectively	High light this to everyone in team, update the newest version for anti virus program. Turn on the auto-update for Windows service patch/	High

6. Tiêu chí đình chỉ & phục hồi kiểm thử :

- Tiêu chí đình chỉ kiểm thử là các điều kiện mà nếu thoả mãn thì kiểm thử sẽ dừng lại.

- Tiêu chí phục hồi là những điều kiện được đòi hỏi để tiếp tục việc kiểm thử đã bị ngừng trước đó.
- Thí dụ :

1.6 Các nguyên tắc cơ bản về kiểm thử

Suspension Criteria

The testing will be halted if these criteria below happen:

- No build notes or it is not clear
- There are some Fatal errors in smoke test build without work around solutions

Resumption Criteria

The testing will be resumed if the build has:

- Build notes clearly
- Any fatal errors with work around solutions
- Test cases had been baselined

7. Môi trường kiểm thử

- Đặc tả đầy đủ về các môi trường kiểm thử, bao gồm đăc5 tả phần cứng, mạng, database, phần mềm, hệ điều hành và các thuộc tính môi trường khác ảnh hưởng đến kiểm thử.
- Thí dụ

cuu duong than cong . com

Web Server

Covered	Not Covered
Tomcat 5.5.9 with JRE 1.5.0 Update 17	

JRE

Covered	Not Covered
JRE 1.6.0 Update 07	
JRE 1.5.0 Update 17 (Default JRE installed)	
JRE 1.4.2 _Update 19	

Web Browsers

Covered	Not Covered
Microsoft Internet Explorer 7.0	
Microsoft Internet Explorer 6.0, Service Pack 3 for Windows XP	Microsoft Internet Explorer 6.0, Service Pack 1 for Windows 2003
Microsoft Internet Explorer 8.0 (if available)	Note: Popup blockers are not supported

Server OS

Covered	Not Covered
Microsoft Windows 2003 Enterprise Edition 32 bit, Service Pack 2	Microsoft Windows 2003 Standard Edition 32 bit, Service Pack 2

Client OS (Supervisor, Agent)

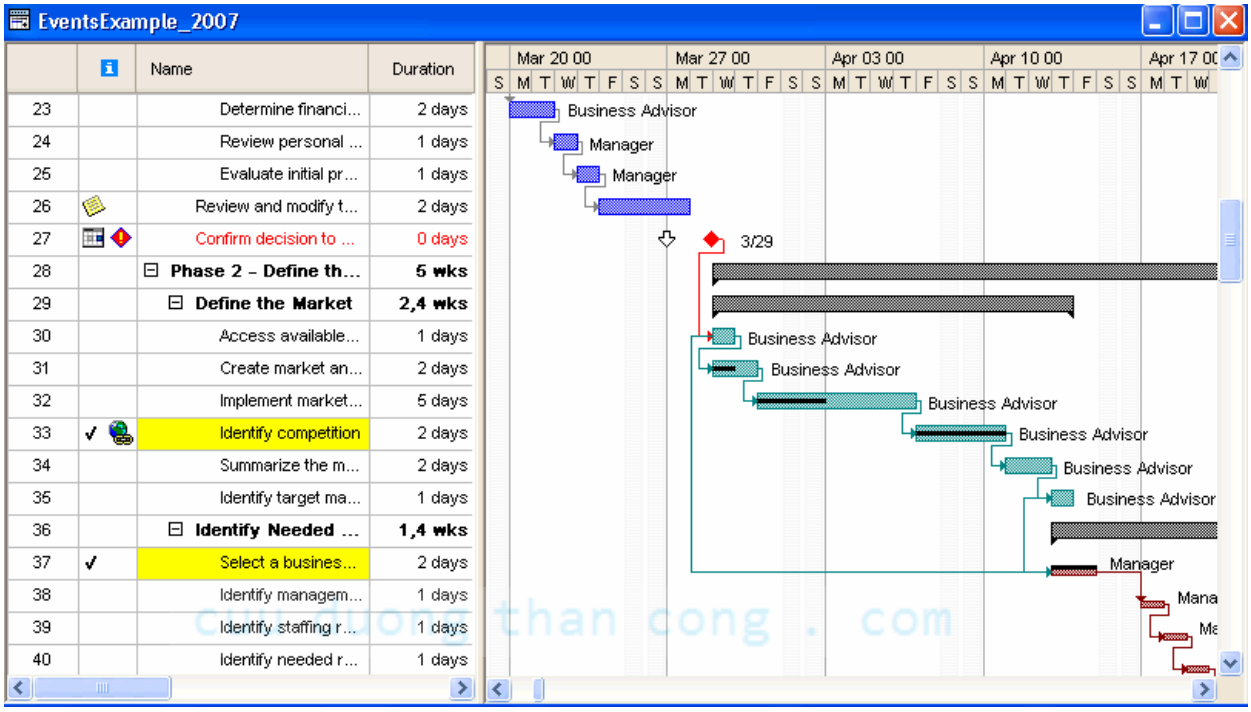
Covered	Not Covered
Microsoft Windows XP Professional 32 bit, Service Pack 2 (Use SP3 if available) (25%)	Microsoft Vista Enterprise 32 bit, Service Pack 1
Microsoft Vista Business 32 bit, Service Pack 1 (25%)	Microsoft Vista Ultimate 32 bit, Service Pack 1
MS 2000 Professional 32 bit, SP 4 (25%)	Microsoft Windows 2003 Standard Edition 32 bit, Service Pack 2

Database

Covered	Not Covered
Microsoft SQL Server 2005 Service Pack 3 Enterprise Edition	Microsoft SQL Server 2005 Service Pack 2 Standard Edition
Oracle 10g Release 2 (10.2.0.1.0) Enterprise Edition	Microsoft SQL Server 2005 Service Pack 2 Enterprise Edition
	Support for SQL Server Database Cluster environment - active-passive

8. Lịch kiểm thử :

- Lịch kiểm thử ở dạng ước lượng, nên chứa các thông tin : các cột mốc với ngày xác định + Kết quả phân phối của từng cột mốc.
- Thí dụ :



9. Tiêu chí dừng kiểm thử & chấp thuận :

Bất kỳ chuẩn chất lượng mong muốn nào mà phần mềm phải thỏa mãn hầu sẵn sàng cho việc phân phối đến khách hàng. Có thể bao gồm các thứ sau :

- Các yêu cầu mà phần mềm phải được kiểm thử dưới các môi trường xáx định.
- Số lỗi tối thiểu ở cấp an ninh và ưu tiên khác nhau, số phủ kiểm thử tối thiểu,...
- Stakeholder sign-off and consensus
- Thí dụ :

cuu duong than cong . com

Exit Criteria

- 100% of test cases have been executed and passed.
- Stability requirements have been met.
- Defect criteria has been met as follows
  - Critical - 0
  - High - 0
  - Medium - 20
  - Low - 50

10. Nhân sự :

Vai trò và trách nhiệm từng người :

- Danh sách các vai trò xác định của các thành viên đội kiểm thử trong hoạt động kiểm thử.
- Các trách nhiệm của từng vai trò.
- Công tác huấn luyện.
- Danh sách các huấn luyện cần thiết cho các QC
- Thí dụ : xem slide kế

cuu duong than cong . com

cuu duong than cong . com

Worker	Minimum Resources Recommended	Specific Responsibilities/Comments
Test Leader A Nguyen	 20%	Provides management oversight Responsibilities: • Provide technical direction • Acquire appropriate resources • Management reporting • Ensures test environment and assets are managed and maintained.
Test Designer A Nguyen B Tran C Ngo D Tran	 40% 50% 50% 50%	Identifies, prioritizes, and implements test cases Responsibilities: • Generate test cases • Evaluate effectiveness of test effort • Prepare test data
Tester A Nguyen B Tran C Ngo D Tran	 40% 50% 50% 50%	Executes the tests Responsibilities: • Execute tests • Log results • Recover from errors • Document defects

cuu duong than cong . com

11. Các tiện ích phục vụ kiểm thử :

- danh sách tất cả các tiện ích cần dùng trong suốt chu kỳ kiểm thử.
- Với project kiểm thử tự động, các tiện ích cần được liệt kê với chỉ số version cùng thông tin license.
- Thí dụ :

Test Tools

During the test cycle, following tools will be used with its purposes:

cuu duong than cong . com

- HP Quality Center: for storing Test cases, Defects and Test cases status
- Guru site: repository for the latest requirements, test metrics

Automation Test tool

Because there is requested for performance test, tool below will be used with license had been payment fully.

- HP LoadRunner 9.2

12. Các kết quả phân phối :

- danh sách tất cả tài liệu hay artifacts dự định phân phối nội bộ sau khi mỗi cột mốc kết thúc hay sau khi project kết thúc.
- Thí dụ :

Test Deliverables

Following is documents, artifacts that will be delivered at the end of testing life cycle

- Test cases
- Status for each Test cases under tested
- Defect reports
- Defect metric will be delivered weekly along with weekly report of project
- Test Scripts for performance test with its results
- Certification test

2.5 Một số điểm chính cần nhớ

cuu duong than cong . com

- Mục đích & phạm vi kiểm thử
- Cách tiếp cận & các chiến lược kiểm thử được dùng.
- Các tính chất cần được kiểm thử/ không cần kiểm thử
- Lịch kiểm thử
- Nhân sự
- Môi trường kiểm thử
- Tạm dừng/Tiếp tục kiểm thử
- Kiểm thử độ chấp thuận
- Các tiện ích kiểm thử cần dùng
- Rủi ro và yếu tố bất ngờ
- Các kết quả phân phối nội bộ

cuu duong than cong . com

2.6 Kết chương

Chương này đã giới thiệu 1 số vấn đề cơ bản liên quan đến qui trình kiểm thử phần mềm như qui trình kiểm thử là gì, tạo sao phải kiểm thử phần mềm, khi nào kiểm thử, ai có liên quan đến việc kiểm thử, qui trình kiểm thử gồm các bước chi tiết nào ?

Chúng ta cũng đã trình bày các nội dung trong bản kế hoạch kiểm thử phần mềm như mục đích & phạm vi kiểm thử, cách tiếp cận & các chiến lược kiểm thử được dùng, các tính chất cần được kiểm thử/ không cần kiểm thử, lịch kiểm thử, nhân sự, môi trường kiểm thử, khi nào tạm dừng/tiếp tục kiểm thử, kiểm thử độ chấp thuận, các tiện ích kiểm thử cần dùng, rủi ro và yếu tố bất ngờ, các kết quả phân phối nội bộ...

cuu duong than cong . com

cuu duong than cong . com

Chương 3

Kỹ thuật kiểm thử hộp trắng

3.1 Tổng quát về kiểm thử hộp trắng

Đối tượng được kiểm thử là 1 thành phần phần mềm (TPPM). TPPM có thể là 1 hàm chức năng, 1 module chức năng, 1 phân hệ chức năng…

Kiểm thử hộp trắng dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu bên trong của đơn vị phần mềm cần kiểm thử để xác định đơn vị phần mềm đó có thực hiện đúng không.

Do đó người kiểm thử hộp trắng phải có kỹ năng, kiến thức nhất định về ngôn ngữ lập trình được dùng, về thuật giải được dùng trong TPPM để có thể thông hiểu chi tiết về đoạn code cần kiểm thử.

Thường tốn rất nhiều thời gian và công sức nếu TPPM quá lớn (thí dụ trong kiểm thử tích hợp hay kiểm thử chức năng).

Do đó kỹ thuật này chủ yếu được dùng để kiểm thử đơn vị. Trong lập trình hướng đối tượng, kiểm thử đơn vị là kiểm thử từng tác vụ của 1 class chức năng nào đó.

Có 2 hoạt động kiểm thử hộp trắng :

- Kiểm thử luồng điều khiển : tập trung kiểm thử thuật giải chức năng.
- Kiểm thử dòng dữ liệu : tập trung kiểm thử đời sống của từng biến dữ liệu được dùng trong thuật giải.

Trong chương 3 này, chúng ta tập trung giới thiệu kiến thức về hoạt động kiểm thử luồng điều khiển của TPPM và trong chương 4, chúng ta tập trung giới thiệu các kiến thức về hoạt động kiểm thử dòng dữ liệu.

3.2 Một số thuật ngữ về kiểm thử luồng điều khiển

Đường thi hành (Execution path) : là 1 kịch bản thi hành đơn vị phần mềm tương ứng, cụ thể nó là danh sách có thứ tự các lệnh được thi hành ứng với 1 lần chạy cụ thể của đơn vị phần mềm, bắt đầu từ điểm nhập của đơn vị phần mềm đến điểm kết thúc của đơn vị phần mềm.

Mỗi TPPM có từ 1 đến n (có thể rất lớn) đường thi hành khác nhau. Mục tiêu của phương pháp kiểm thử luồng điều khiển là đảm bảo mọi đường thi hành của đơn vị phần mềm cần kiểm thử đều chạy đúng. Rất tiếc trong thực tế, công sức và thời gian để đạt mục tiêu trên đây là rất lớn, ngay cả trên những đơn vị phần mềm nhỏ.

Thí dụ đoạn code sau :

```
for (i=1; i<=1000; i++)
  for (j=1; j<=1000; j++)
        for (k=1; k<=1000; k++)
              doSomethingWith(i,j,k);
```

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

Còn đoạn code gồm 32 lệnh if else độc lập sau :

```
if (c1) s11 else s12;
if (c2) s21 else s22;
if (c3) s31 else s32;
...
if (c32) s321 else s322;
```

có 2^32 = 4 tỉ đường thi hành khác nhau.

Mà cho dù có kiểm thử hết được toàn bộ các đường thi hành thì vẫn không thể phát hiện những đường thi hành cần có nhưng không (chưa) được hiện thực :

```
if (a>0) doIsGreater();
if (a==0) doIsEqual();
// thiếu việc xử lý trường hợp a < 0 - if (a<0) doIsLess();
```

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

```
int phanso (int a, int b) {
   return a/b;
}
```

khi kiểm tra, ta chọn b <> 0 thì chạy đúng, nhưng khi dùng thật trong trường hợp b = 0 thì hàm phanso bị lỗi.

3.3 Các cấp phủ kiểm thử (Coverage)

Do đó, ta nên kiểm thử 1 số test case tối thiểu mà kết quả độ tin cậy tối đa. Nhưng làm sao xác định được số test case tối thiểu nào có thể đem lại kết quả có độ tin cậy tối đa ?

Phủ kiểm thử (Coverage) : là tỉ lệ các thành phần thực sự được kiểm thử so với tổng thể sau khi đã kiểm thử các test case được chọn. Phủ càng lớn thì độ tin cậy càng cao.

Thành phần liên quan có thể là lệnh thực thi, điểm quyết định, điều kiện con hay là sự kết hợp của chúng.

Phủ cấp 0 : kiểm thử những gì có thể kiểm thử được, phần còn lại để người dùng phát hiện và báo lại sau. Đây là mức độ kiểm thử không thực sự có trách nhiệm.

Phủ cấp 1 : kiểm thử sao cho mỗi lệnh được thực thi ít nhất 1 lần.

Phân tích hàm foo sau đây :

```
1   float foo(int a, int b, int c, int d) {
2     float e;
3     if (a==0)
4       return 0;
5     int x = 0;
6     if ((a==b) || ((c==d) && bug(a)))
7       x = 1;
8     e = 1/x;
9     return e;
10  }
```

Với hàm foo trên, ta chỉ cần 2 test case sau đây là đạt 100% phủ cấp 1 :

1. foo(0,0,0,0), trả về 0

2. foo(1,1,1,1), trả về 1

nhưng không phát hiện lỗi chia 0 ở hàng lệnh 8.

Phủ cấp 2 : kiểm thử sao cho mỗi điểm quyết định luận lý đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các nhánh (Branch coverage). Phủ các nhánh đảm bảo phủ các lệnh.

Line	Predicate	True	False
3	(a == 0)	Test Case 1 foo(0, 0, 0, 0) return 0	Test Case 2 foo(1, 1, 1, 1) return 1
6	((a==b) OR ((c == d) AND bug(a)))	Test Case 2 foo(1, 1, 1, 1) return 1	Test Case 3 foo(1, 2, 1, 2) division by zero!

Với 2 test case xác định trong slide trước, ta chỉ đạt được 3/4 = 75% phủ các nhánh. Nếu thêm test case 3 :

3. foo(1,2,1,2), thì mới đạt 100% phủ các nhánh.

Phủ cấp 3 : kiểm thử sao cho mỗi điều kiện luận lý con (subcondition) của từng điểm quyết định đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các điều kiện con (subcondition coverage). Phủ các điều kiện con chưa chắc đảm bảo phủ các nhánh & ngược lại.

Predicate	True	False
a ==0	TC 1 : foo(0, 0, 0, 0) return 0	TC 2 : foo(1, 1, 1, 1) return 1
(a==b)	TC 2 : foo(1, 1, 1, 1) return value 0	TC 3 : foo(1, 2, 1, 2) division by zero!
(c==d)	TC 4 : foo(1, 2, 1, 1) Return 1	TC 3 : foo(1, 2, 1, 2) division by zero!

bug(a)	TC 4 : foo(1, 2, 1, 1) Return 1	TC 5 : foo(2,1, 1, 1) division by zero!

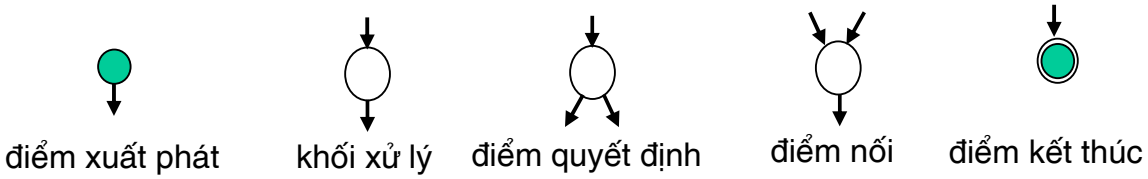
Phủ cấp 4 : kiểm thử sao cho mỗi điều kiện luận lý con (subcondition) của từng điểm quyết định đều được thực hiện ít nhất 1 lần cho trường hợp TRUE lẫn FALSE & điểm quyết định cũng được kiểm thử cho cả 2 nhánh TRUE lẫn FALSE. Ta gọi mức kiểm thử này là phủ các nhánh & các điều kiện con (branch & subcondition coverage). Đây là mức độ phủ kiểm thử tốt nhất trong thực tế. Phần còn lại của chương này sẽ giới thiệu qui trình kỹ thuật để định nghĩa các testcase sao cho nếu kiểm thử hết các testcase được định nghĩa này, ta sẽ đạt phủ kiểm thử cấp 4.

3.4 Đồ thị dòng điều khiển

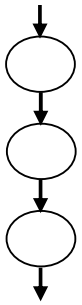
Là một trong nhiều phương pháp miêu tả thuật giải. Đây là phương pháp trực quan cho chúng ta thấy dễ dàng các thành phần của thuật giải và mối quan hệ trong việc thực hiện các thành phần này.

Gồm 2 loại thành phần : các nút và các cung nối kết giữa chúng.

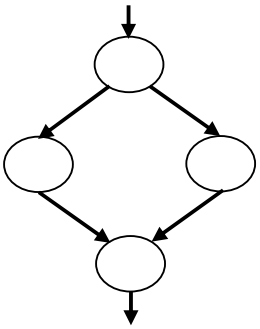
Các loại nút trong đồ thị dòng điều khiển :



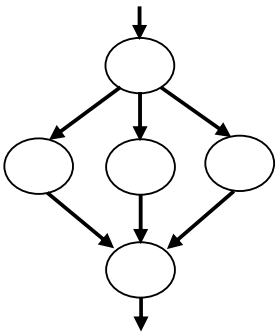
Miêu tả các cấu trúc điều khiển phổ dụng :



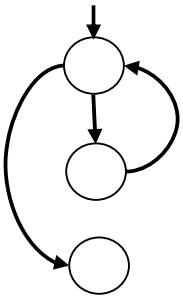
tuần tự



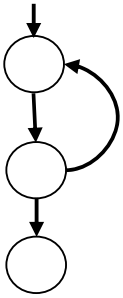
If



switch



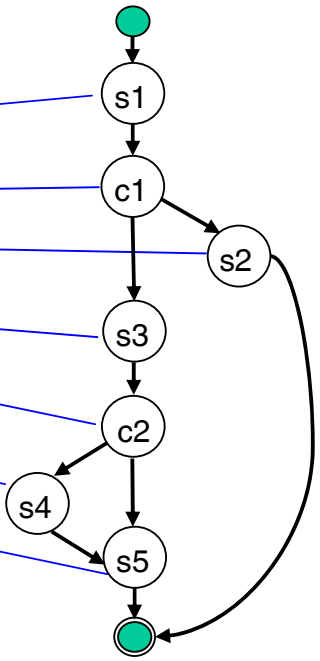
while c do...



do ... while c

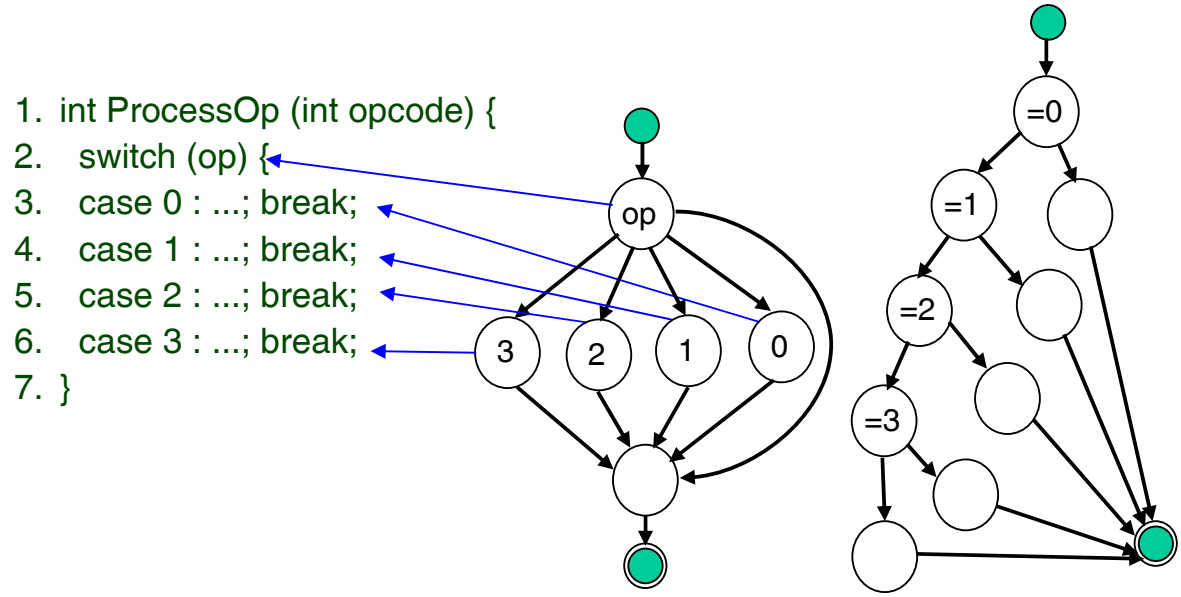
Thí dụ :

```
1.  float foo(int a, int b, int c, int d) {
2.    float e;
3.    if (a==0)
4.       return 0;
5.    int x = 0;
6.    if ((a==b) || ((c==d) && bug(a)))
7.      x = 1;
8.    e = 1/x;
9.    return e;
10.}
```



Nếu đồ thị dòng điều khiển chỉ chứa các nút quyết định nhị phân thì ta gọi nó là đồ thị dòng điều khiển nhị phân.

Ta luôn có thể chi tiết hóa 1 đồ thị dòng điều khiển bất kỳ thành đồ thị dòng điều khiển nhị phân.



Độ phức tạp Cyclomatic C

Độ phức tạp Cyclomatic C = V(G) của đồ thị dòng điều khiển được tính bởi 1 trong các công thức sau :

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

Độ phức tạp Cyclomatic C chính là số đường thi hành tuyến tính độc lập của TPPM cần kiểm thử.

Nếu chúng ta chọn lựa được đúng C đường thi hành tuyến tính độc lập của TPPM cần kiểm thử và kiểm thử tất cả các đường thi hành này thì sẽ đạt được phủ kiểm thử cấp 3 như đã trình bày trong các slide trước.

3.5 Đồ thị dòng điều khiển cơ bản

Xét đồ thị dòng điều khiển nhị phân : nếu từng nút quyết định (nhị phân) đều miêu tả 1 điều kiện con luận lý thì ta nói đồ thị này là đồ thi dòng điểu khiển cơ bản.

Ta luôn có thể chi tiết hóa 1 đồ thị dòng điều khiển bất kỳ thành đồ thị dòng điều khiển nhị phân. Tương tự, ta luôn có thể chi tiết hóa 1 đồ thị dòng điều khiển nhị phân bất kỳ thành đồ thị dòng điều khiển cơ bản.

Tóm lại, ta luôn có thể chi tiết hóa 1 đồ thị dòng điều khiển bất kỳ thành đồ thị dòng điều khiển cơ bản.

Độ phức tạp Cyclomatic C của đồ thị dòng điều khiển cơ bản chính là số đường thi hành tuyến tính độc lập cơ bản của TPPM cần kiểm thử.

Nếu chúng ta chọn lựa được đúng C đường thi hành tuyến tính độc lập cơ bản của TPPM cần kiểm thử và kiểm thử tất cả các đường thi hành này thì sẽ đạt được phủ kiểm thử cấp 4 như đã trình bày trong các slide trước.

3.6 Qui trình kiểm thử hộp trắng

Tom McCabe đề nghị qui trình kiểm thử TPPM gồm các bước công việc sau :

1. Từ TPPM cần kiểm thử, xây dựng đồ thị dòng điều khiển tương ứng, rồi chuyển thành đồ thị dòng điều khiển nhị phân, rồi chuyển thành đồ thị dòng điều khiển cơ bản.
2. Tính độ phức tạp Cyclomatic của đồ thị (C = P +1).
3. Xác định C đường thi hành tuyến tính độc lập cơ bản cần kiểm thử (theo thuật giải chi tiết ở slide kế).
4. Tạo từng test case cho từng đường thi hành tuyến tính độc lập cơ bản.
5. Thực hiện kiểm thử trên từng test case.

6. So sánh kết quả có được với kết quả được kỳ vọng.
7. Lập báo cáo kết quả để phản hồi cho những người có liên qu

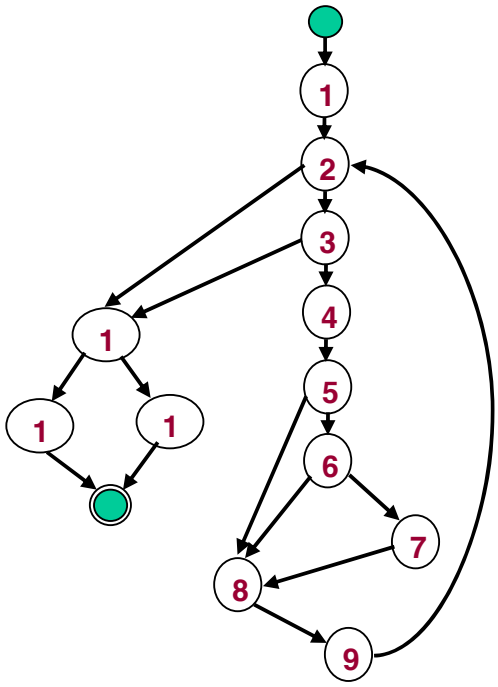
Qui trình xác định các đường tuyến tính độc lập

Tom McCabe đề nghị qui trình xác định C đường tuyến tính độc lập gồm các bước :

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

3.7 Thí dụ

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



Đồ thị bên có 5 nút quyết định nhị phân nên có độ phức tạp C = 5+1 = 6.

6 đường thi hành tuyến tính độc lập cơ bản là :

1. 1→2→10→11
2. 1→2→3→10→11
3. 1→2→3→4→5→8→9
4. 1→2→3→4→5→6→8→9
5. 1→2→3→4→5→6→7→8→9
6. 1→2→10→12

Thiết kế các test case

Phân tích mã nguồn của hàm average, ta định nghĩa 6 testcase kết hợp với 6 đường thi hành tuyến tính độc lập cơ bản như sau :

Test case cho đường 1 :

value(k) <>-999, với 1< k < i

value(i) = -999 với 2 ≤ i ≤ 100

Kết quả kỳ vọng : (1) average=Giá trị trung bình của i-1 giá trị hợp lệ. (2) tcnt = i-1. (3) vcnt = i-1

Chú ý : không thể kiểm thử đường 1 này riêng biệt mà phải kiểm thử chung với đường 4 hay 5 hay 6.

Test case cho đường 2 :

value(k) <>-999, với $\forall k < i$, i >100

Kết quả kỳ vọng : (1) average=Giá trị trung bình của 100 giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = 100

Test case cho đường 3 :

value(1) = -999

Kết quả kỳ vọng : (1) average = -999. (2) tcnt = 0 (3) vcnt = 0

Test case cho đường 4 :

value(i) <> -999 $\forall i <= 100$

và value(k) < min với k < i

Kết quả kỳ vọng : (1) average=Giá trị trung bình của n giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = n (số lượng giá trị hợp lệ)

Test case cho đường 5 :

value(i) <>-999 với $\forall i <= 100$

và value(k) > max với k <= i

Kết quả kỳ vọng : (1) average=Giá trị trung bình của n giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = n (số lượng giá trị hợp lệ)

Test case cho đường 6 :

value(i) <>-999 và min <= value(i) <= max với $\forall i <= 100$

Kết quả kỳ vọng : (1) average=Giá trị trung bình của 100 giá trị hợp lệ. (2) tcnt = 100. (3) vcnt = 100

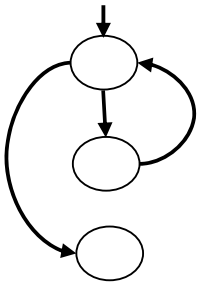
3.8 Kiểm thử vòng lặp

Thường thân của 1 lệnh lặp sẽ được thực hiện nhiều lần (có thể rất lớn). Chi phí kiểm thử đầy đủ rất tốn kém, nên chúng ta sẽ chỉ kiểm thử ở những lần lặp mà theo thống kê dễ gây lỗi nhất. Ta xét từng loại lệnh lặp, có 4 loại :

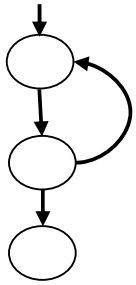
1. lệnh lặp đơn giản : thân của nó chỉ chứa các lệnh khác chứ không chứa lệnh lặp khác.
2. lệnh lặp lồng nhau : thân của nó có chứa ít nhất lệnh lặp khác...

3. lệnh lặp liền kề : 2 hay nhiều lệnh lặp kế tiếp nhau
4. lệnh lặp giao nhau : 2 hay nhiều lệnh lặp giao nhau.

1. Kiểm thử loại vòng lặp n lần đơn giản :



while c do...

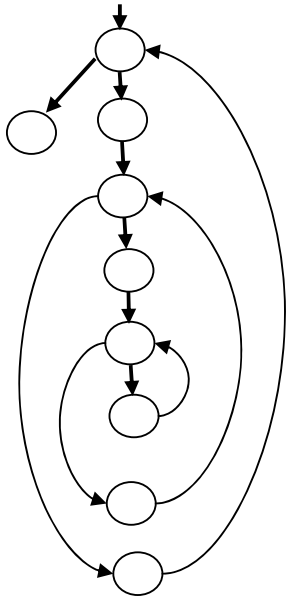


do ... while c

Nên chọn các test case để kiểm thử thân lệnh lặp ở các vị trí sau :

- chạy 0 bước.
- chạy 1 bước.
- chạy 2 bước.
- chạy k bước, k là giá trị nào đó thỏa $2 < k < n-1$.
- chạy n-1 bước
- chạy n bước
- chạy n+1 bước.

2. Kiểm thử vòng lặp lồng nhau :



Kiểm thử tuần tự từng vòng lặp từ trong ra ngoài theo đề nghị sau đây :

- kiểm thử vòng lặp trong cùng : cho các vòng ngoài chạy với giá trị min, kiểm thử vòng lặp trong cùng bằng 7 test case đã giới thiệu ở silde trước.
- kiểm thử từng vòng lặp còn lại : cho các vòng ngoài nó chạy với giá trị min, còn các vòng bên trong nó chạy với giá trị điển hình, kiểm thử nó bằng 7 test case đã giới thiệu ở slide trước.

3. Kiểm thử các vòng lặp liền kề : Kiểm thử tuần tự từng vòng lặp từ trên xuống, mỗi vòng thực hiện kiểm thử bằng 7 test case đã giới thiệu.

4. Riêng các vòng lặp giao nhau thì thường do việc viết code chưa tốt tạo ra $\Rightarrow$ nên cấu trúc lại đoạn code sao cho không chứa dạng giao nhau này.

3.9 Kết chương

Chương này đã giới thiệu 1 kỹ thuật thiết yếu để kiểm thử hộp trắng TPPM, đó là kỹ thuật kiểm thử dòng điều khiển.

Chúng ta đã giới thiệu các cấp độ phủ kiểm thử khác nhau, giới thiệu đồ thị dòng điều khiển và đồ thị dòng điều khiển cơ bản

của TPPM, độ phức tạp Cyclomatic C, qui trình tổng quát để kiểm thử dòng điều khiển.

Chương này cũng đã giới thiệu 1 thí dụ cụ thể về qui trình kiểm thử dòng điều khiển trên 1 TPPM.

Chương 4

Kỹ thuật kiểm thử hộp trắng (tt)

4.1 Tổng quát về kiểm thử dòng dữ liệu

Mục tiêu của chương trình là xử lý dữ liệu. Dữ liệu của chương trình là tập nhiều biến độc lập. Phương pháp kiểm thử dòng dữ liệu sẽ kiểm thử đời sống của từng biến dữ liệu có "tốt lành" trong từng luồng thi hành của chương trình.

Phương pháp kiểm thử dòng dữ liệu là 1 công cụ mạnh để phát hiện việc dùng không hợp lý các biến do lỗi coding phần mềm gây ra :

- Phát biểu gán hay nhập dữ liệu vào biến không đúng.
- Thiếu định nghĩa biến trước khi dùng
- Tiên đề sai (do thi hành sai luồng thi hành).
- ...

Mỗi biến nên có chu kỳ sống tốt lành thông qua trình tự 3 bước : được tạo ra, được dùng và được xóa đi.

Chỉ có những lệnh nằm trong tầm vực truy xuất biến mới có thể truy xuất/xử lý được biến. Tầm vực truy xuất biến là tập các lệnh được phép truy xuất biến đó.

Thường các ngôn ngữ lập trình cho phép định nghĩa tầm vực cho mỗi biến thuộc 1 trong 3 mức chính yếu : toàn cục, cục bộ trong từng module, cục bộ trong từng hàm chức năng.

```
int x, y;
void func1() {  //thân hàm
    int x;     // định nghĩa biến x mới cục bộ trong hàm
    ...;        // mỗi lần truy xuất x là x cục bộ trong hàm
    {          // khối lệnh bên trong bắt đầu
       int y;   // định nghĩa biến y mới cục bộ trong lệnh phức hợp
```

```
        ...;      //mỗi lần truy xuất y là y cục bộ trong lệnh phức hợp
    }       // y bên trong tự động bị xóa
    ...;    //truy xuất y ngoài cùng, x cục bộ trong hàm
}   // x cục bộ trong hàm bị xóa tự động
```

4.2 Phân tích đời sống của 1 biến

Các lệnh truy xuất 1 biến thông qua 1 trong 3 hành động sau :

- d : định nghĩa biến, gán giá trị xác định cho biến (nhập dữ liệu vào biến cũng là hoạt động gán trị cho biến).
- u : tham khảo trị của biến (thường thông qua biểu thức).
- k : hủy (xóa bỏ) biến đi.

Như vậy nếu ký hiệu ~ là miêu tả trạng thái mà ở đó biến chưa tồn tại, ta có 3 khả năng xử lý đầu tiên trên 1 biến :

- ~d : biến chưa tồn tại rồi được định nghĩa với giá trị xác định.
- ~u : biến chưa tồn tại rồi được dùng ngay (trị nào ?)
- ~k : biến chưa tồn tại rồi bị hủy (lạ lùng).

3 hoạt động xử lý biến khác nhau kết hợp lại tạo ra 9 cặp đôi hoạt động xử lý biến theo thứ tự :

- dd : biến được định nghĩa rồi định nghĩa nữa : hơi lạ, có thể đúng và chấp nhận được, nhưng cũng có thể có lỗi lập trình.
- du : biến được định nghĩa rồi được dùng : trình tự đúng và bình thường.
- dk : biến được định nghĩa rồi bị xóa bỏ : hơi lạ, có thể đúng và chấp nhận được, nhưng cũng có thể có lỗi lập trình.
- ud : biến được dùng rồi định nghĩa giá trị mới : hợp lý.
- uu : biến được dùng rồi dùng tiếp : hợp lý.
- uk : biến được dùng rồi bị hủy : hợp lý.

- kd : biến bị xóa bỏ rồi được định nghĩa lại : chập nhận được.
- ku : biến bị xóa bỏ rồi được dùng : đây luôn là lỗi.
- kk : biến bị xóa bỏ rồi bị xóa nữa : có lẽ là lỗi lập trình.

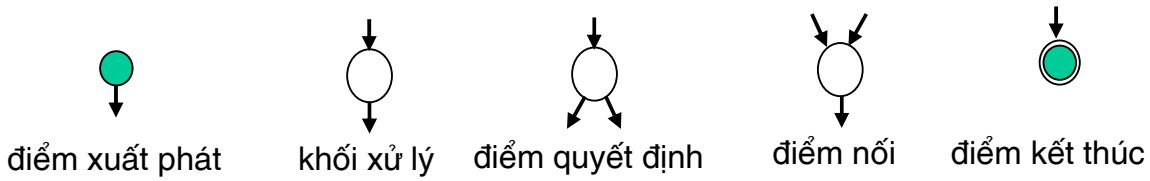
4.3 Đồ thị dòng dữ liệu

Là một trong nhiều phương pháp miêu tả các kịch bản đời sống khác nhau của các biến.

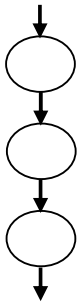
Qui trình xây dựng đồ thị dòng dữ liệu dựa trên qui trình xây dựng đồ thị dòng điều khiển của TPPM cần kiểm thử.

Gồm 2 loại thành phần : các nút và các cung nối kết giữa chúng.

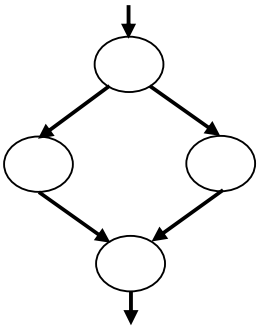
Các loại nút trong đồ thị dòng điều khiển :



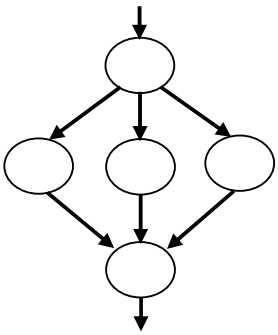
Miêu tả các cấu trúc điều khiển phổ dụng :



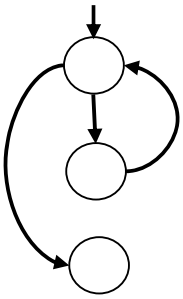
tuần tự



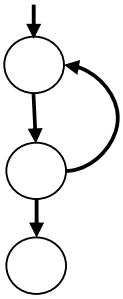
If



switch



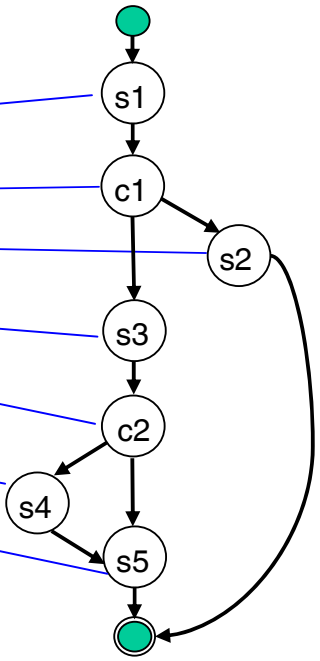
while c do...



do ... while c

Thí dụ :

```
1.  float foo(int a, int b, int c, int d) {
2.    float e;
3.    if (a==0)
4.       return 0;
5.    int x = 0;
6.    if ((a==b) || ((c==d) && bug(a)))
7.      x = 1;
8.    e = 1/x;
9.    return e;
10.}
```



Độ phức tạp Cyclomatic C

Ta cũng dùng độ phức tạp Cyclomatic C = V(G) của đồ thị dòng điều khiển của TPPM cần kiểm thử để xác định số đường thi hành tuyến tính độc lập của TPPM cần kiểm thử.

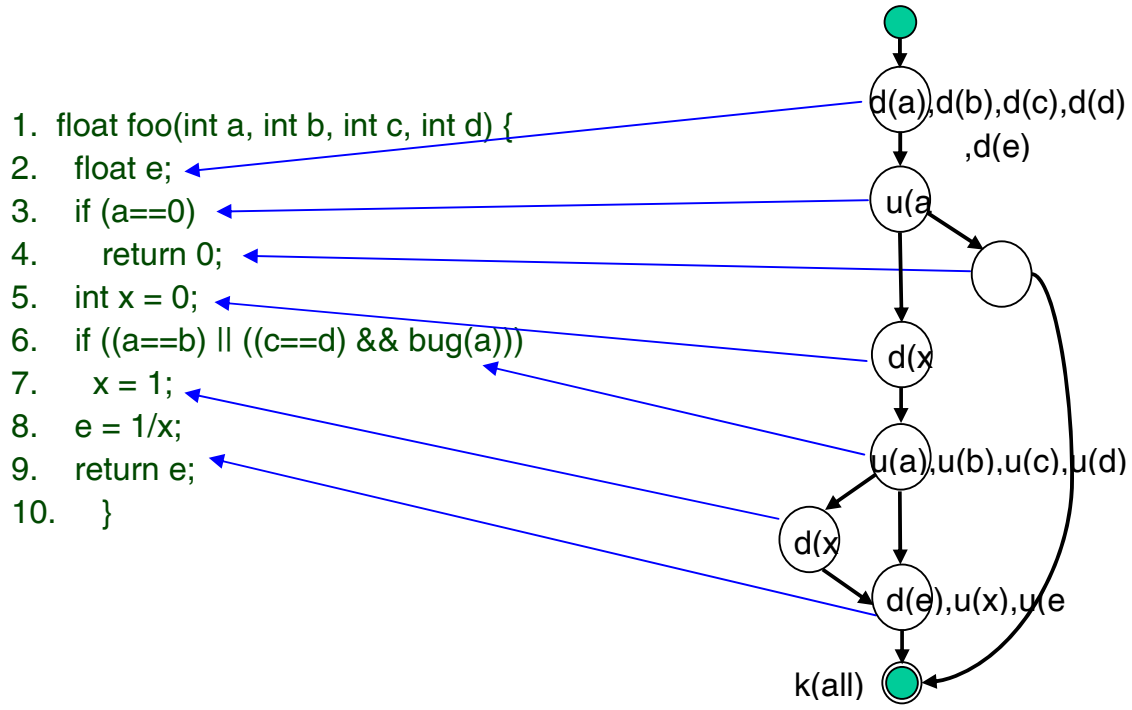
Mục tiêu của kiểm thử dòng dữ liệu là chọn lựa được đúng C đường thi hành tuyến tính độc lập của TPPM cần kiểm thử rồi kiểm thử đời sống của từng biến trên từng đường thi hành này xem có lỗi gì không.

4.4 Qui trình kiểm thử dòng dữ liệu

Qui trình kiểm thử dòng dữ liệu của 1 TPPM gồm các bước công việc sau :

- Từ TPPM cần kiểm thử, xây dựng đồ thị dòng điều khiển tương ứng, rồi chuyển thành đồ thị dòng điều khiển nhị phân, rồi chuyển thành đồ thị dòng dữ liệu.
- Tính độ phức tạp Cyclomatic của đồ thị (C = P +1).
- Xác định C đường thi hành tuyến tính độc lập cơ bản cần kiểm thử (theo thuật giải chi tiết ở chương 3).
- Lặp kiểm thử đời sống từng biến dữ liệu :
  - mỗi biến có thể có tối đa C kịch bản đời sống khác nhau.
  - trong từng kịch bản đời sống của 1 biến, kiểm thử xem có tồn tại cặp đôi hoạt động không bình thường nào không ? Nếu có hãy ghi nhận để lập báo cáo kết quả và phản hồi cho những người có liên quan.

4.5 Thí dụ

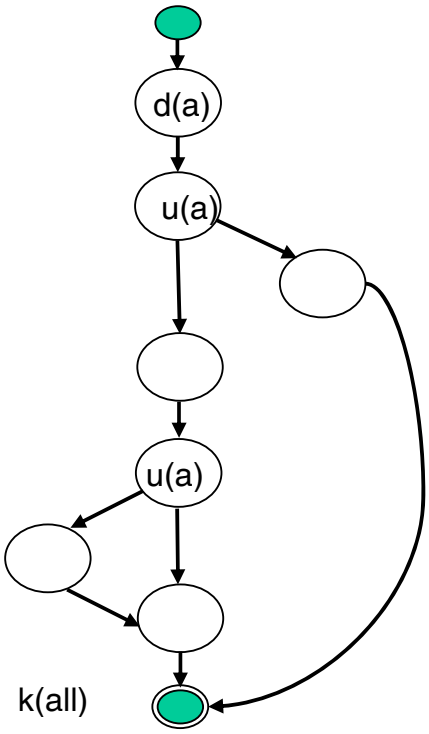


Đồ thị ở slide trước có 2 nút quyết định nhị phân nên có độ phức tạp C = 2 +1 = 3.

Nó có 4 biến đầu vào (tham số) và 2 biến cục bộ.

Hãy lặp kiểm thử đời sống từng biến a, b, c, d, e, x.

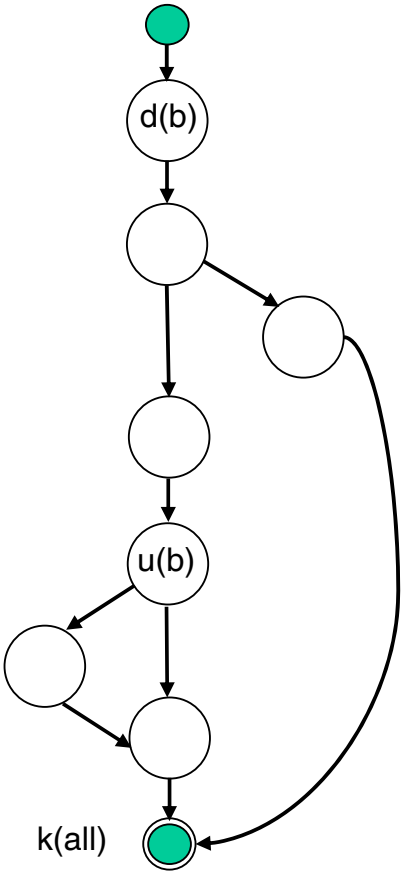
Kiểm thử đời sống biến a



- Kịch bản 1 : ~duuk
- Kịch bản 2 : ~duuk (giống kịch bản 1).
- Kịch bản 3 : ~duk

Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thường cả.

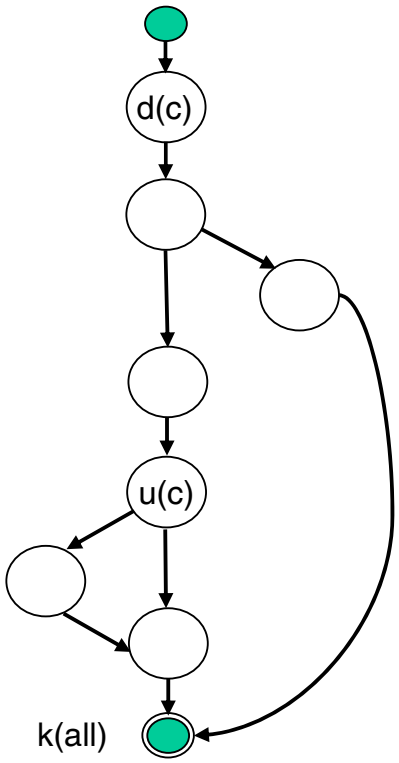
Kiểm thử đời sống biến b



- Kịch bản 1 : ~duk
- Kịch bản 2 : ~duk (giống kịch bản 1).
- Kịch bản 3 : ~dk

Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thýờng cả.

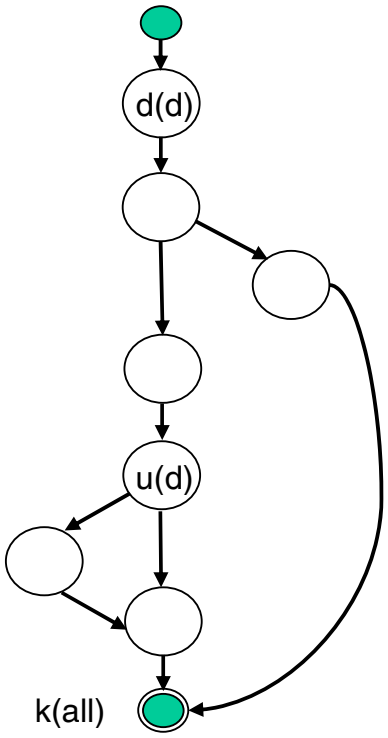
Kiểm thử đời sống biến c



- Kịch bản 1 : ~duk
- Kịch bản 2 : ~duk (giống kịch bản 1).
- Kịch bản 3 : ~dk

Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thýờng cả.

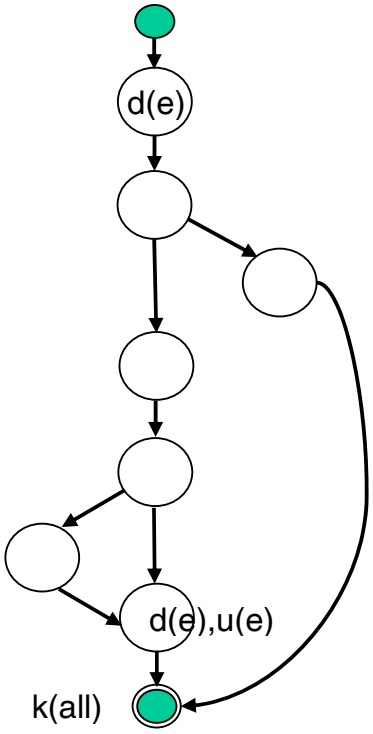
Kiểm thử đời sống biến d



- Kịch bản 1 : ~duk
- Kịch bản 2 : ~duk (giống kịch bản 1).
- Kịch bản 3 : ~dk

Cả 3 kịch bản trên đều không chứa cặp đôi hoạt động nào bất thýờng cả.

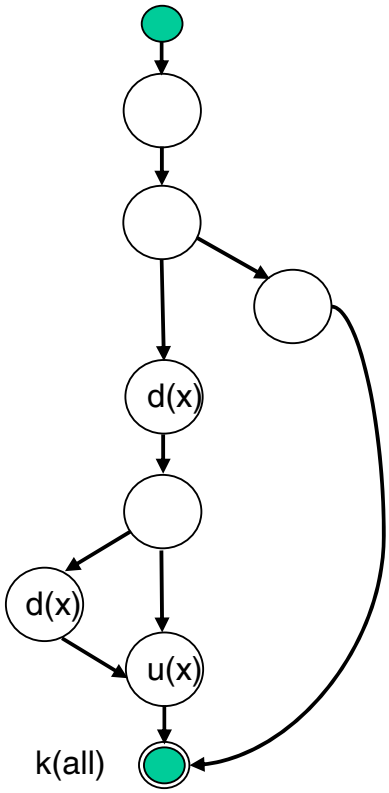
Kiểm thử đời sống biến e



- Kịch bản 1 : ~dduk
- Kịch bản 2 : ~dduk (giống kịch bản 1).
- Kịch bản 3 : ~dk

Trong 3 kịch bản trên, kịch bản 1 & 2 có chứa cặp đôi dd bất thuờng nên cần tập trung chú ý kiểm tra xem có phải là lỗi không.

Kiểm thử đời sống biến x



- Kịch bản 1 : ~dduk
- Kịch bản 2 : ~duk
- Kịch bản 3 : ~

Trong 3 kịch bản trên, chỉ có kịch bản 1 có chứa cặp đôi dd bất thuờng nên cần tập trung chú ý kiểm tra xem có phải là lỗi không.

4.6 Kết chương

Chương này đã giới thiệu tiếp 1 kỹ thuật khác để kiểm thử hộp trắng TPPM, đó là kỹ thuật kiểm thử dòng dữ liệu.

Chúng ta đã phân tích đời sống của biến dữ liệu, các cặp đôi hoạt động trên biến được gọi là hợp lệ hay nghi ngờ có lỗi hay tệ hơn là chắn chắn gây lỗi.

Chương này cũng đã giới thiệu 1 thí dụ cụ thể về qui trình kiểm thử dòng dữ liệu trên 1 TPPM.

Chương 5

Kỹ thuật kiểm thử hộp đen

5.1 Tổng quát về kiểm thử hộp đen

Đối tượng được kiểm thử là 1 thành phần phần mềm (TPPM). TPPM có thể là 1 hàm chức năng, 1 module chức năng, 1 phân hệ chức năng… Nói chung, chiến lược kiểm thử hộp đen thích hợp cho mọi cấp độ kiểm thử từ kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống, kiếm thử độ chấp nhận của người dùng.

Kiểm thử hộp đen (black-box testing) là chiến lược kiểm thử TPPM dựa vào thông tin duy nhất là các đặc tả về yêu cầu chức năng của TPPM tương ứng.

Đây là chiến lược kiểm thử theo góc nhìn từ ngoài vào, các người tham gia kiểm thử hộp đen không cần có kiến thức nào về thông tin hiện thực TPPM cần kiểm thử (mã nguồn của thành phần phần mềm, thuật giải được dùng, các dữ liệu được xử lý…).

Qui trình kiểm thử hộp đen tổng quát gồm các bước chính :

- Phân tích đặc tả về các yêu cầu chức năng mà TPPM cần thực hiện.
- Dùng 1 kỹ thuật định nghĩa các testcase xác định (sẽ giới thiệu sau) để định nghĩa các testcase. Định nghĩa mỗi testcase là xác định 3 thông tin sau :
  - Giá trị dữ liệu nhập để TPPM xử lý (hoặc hợp lệ hoặc không hợp lệ).
  - Trạng thái của TPPM cần có để thực hiện testcase.
  - Giá trị dữ liệu xuất mà TPPM phải tạo được.
- Kiểm thử các testcase đã định nghĩa.
- So sánh kết quả thu được với kết quả kỳ vọng trong từng testcase, từ đó lập báo cáo về kết quả kiểm thử.

Vì chiến lược kiểm thử hộp đen thích hợp cho mọi mức độ kiểm thử nên nhiều người đã nghiên cứu tìm hiểu và đưa ra nhiều kỹ thuật kiểm thử khác nhau, chúng ta sẽ chọn ra 8 kỹ thuật có nhiều ưu điểm nhất và được dùng phổ biến nhất, đó là :

1. Kỹ thuật phân lớp tương đương (Equivalence Class Partitioning).
2. Kỹ thuật phân tích các giá trị biên (Boundary value analysis).
3. Kỹ thuật dùng các bảng quyết định (Decision Tables)
4. Kỹ thuật kiểm thử các bộ n thần kỳ (Pairwise)
5. Kỹ thuật dùng bảng chuyển trạng thái (State Transition)
6. Kỹ thật phân tích vùng miền (domain analysis)
7. Kỹ thuật dựa trên đặc tả Use Case (Use case)
8. Kỹ thuật dùng lược đồ quan hệ nhân quả (Cause-Effect Diagram)

5.2 Kỹ thuật phân lớp tương đương

Tinh thần của kỹ thuật này là cố gắng phân các testcase ra thành nhiều nhóm (họ) khác nhau : các testcase trong mỗi họ sẽ kích hoạt TPPM thực hiện cùng 1 hành vi. Mỗi nhóm testcase thỏa mãn tiêu chuẩn trên được gọi là 1 lớp tương đương, ta chỉ cần xác định 1 testcase đại diện cho nhóm và dùng testcase này để kiểm thử TPPM. Như vậy ta đã giảm rất nhiều testcase cần định nghĩa và kiểm thử, nhưng chất lượng kiểm thử không bị giảm sút bao nhiêu so với vét cạn. Điều này là dựa vào kỳ vọng rất hợp lý sau đây :

- Nếu 1 testcase trong lớp tương đương nào đó gây lỗi TPPM thì các testcase trong lớp này cũng sẽ gây lỗi như vậy.

- Nếu 1 testcase trong lớp tương đương nào đó không gây lỗi TPPM thì các testcase trong lớp này cũng sẽ không gây lỗi.

Vấn đề kế tiếp là có cần định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả hay không ? Điều này phụ thuộc vào tinh thần kiểm thử :

- Nếu ta dùng tinh thần kiểm thử theo hợp đồng (Testing-by-Contract) thì không cần định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả vì không cần thiết.
- Còn nếu ta dùng tinh thần kiểm thử phòng vệ (Defensive Testing), nghĩa là kiểm thử hoàn hảo, thì phải định nghĩa các lớp tương đương đại diện các testcase chứa các giá trị không hợp lệ theo đặc tả để xem TPPM phản ứng như thế nào với những testcase này.

Thí dụ ta cần kiểm thử 1 TPPM “quản lý nguồn nhân lực” với đặc tả chức năng như sau : mỗi lần nhận 1 hồ sơ xin việc, TPPM sẽ ra quyết định dựa vào tuổi ứng viên theo bảng sau :

Tuổi ứng viên	Kết quả
0-16	Không thuê
16-18	Thuê dạng bán thời gian
18-55	Thuê toàn thời gian
55-99	Không thuê

Lưu ý rằng bảng đặc tả chức năng phía trên có lỗi ở các giá trị đầu và/hoặc cuối trong từng luật, và giả sử chúng ta chưa phát hiện lỗi này. Chúng ta sẽ thấy bằng cách nào sẽ phát hiện dễ dàng lỗi này.

Phân tích đặc tả chức năng của TPPM cần kiểm thử của slide trước, ta thấy có 4 lớp tương đương, mỗi lớp chứa các testcase ứng với 1 chế độ xử lý của TPPM : không thuê vì quá trẻ, thuê dạng bán thời gian, thuê toàn thời gian, không thuê vì quá già.

Ứng với mỗi lớp tương đương, ta định nghĩa 1 testcase đại diện, thí dụ ta chọn 4 testcase sau :

1. Testcase 1 : {Input : 2 tuổi, Output : không thuê}
2. Testcase 2 : {Input : 17 tuổi, Output : thuê bán thời gian}
3. Testcase 3 : {Input : 35 tuổi, Output : thuê toàn thời gian}
4. Testcase 4 : {Input : 90 tuổi, Output : không thuê}

Trong thí dụ trên, thay vì phải kiểm thử vét cạn 100 testcase, ta chỉ kiểm thử 4 testcase → chí phí giảm rất lớn, nhưng chất lượng kiểm thử hy vọng không bị giảm sút là bao.

Tại sao chúng ta hy vọng chất lượng kiểm thử dùng lớp tương đương không giảm sút nhiều ? Hãy xét đoạn code mà những người lập trình bình thường sẽ viết khi xử lý TPPM cần kiểm thử của slide trước :

```
if (applicantAge >= 0 && applicantAge <=16) qd ="NO";
if (applicantAge >= 16 && applicantAge <=18) qd ="PART";
if (applicantAge >= 18 && applicantAge <=55) qd ="FULL";
if (applicantAge >= 55 && applicantAge <=99) qd ="NO";
```

Ở góc nhìn kiểm thử hộp trắng, nếu dùng 4 testcase đại diện của 4 lớp tương đương, ta sẽ kiểm thử được ở phủ cấp 3, cấp phủ rất tốt vì đã kiểm thử 100% các lệnh mã nguồn, 100% các nhánh quyết định.

Tuy nhiên nếu người lập trình hiện thực như sau (rất cá biệt vì đây là người lập trình rất yếu tay nghề) :

```
if (applicantAge == 0) qd ="NO";
…
if (applicantAge == 16) qd ="PART";
…
if (applicantAge == 53) qd ="FULL";
…
if (applicantAge == 99) qd ="NO";
```

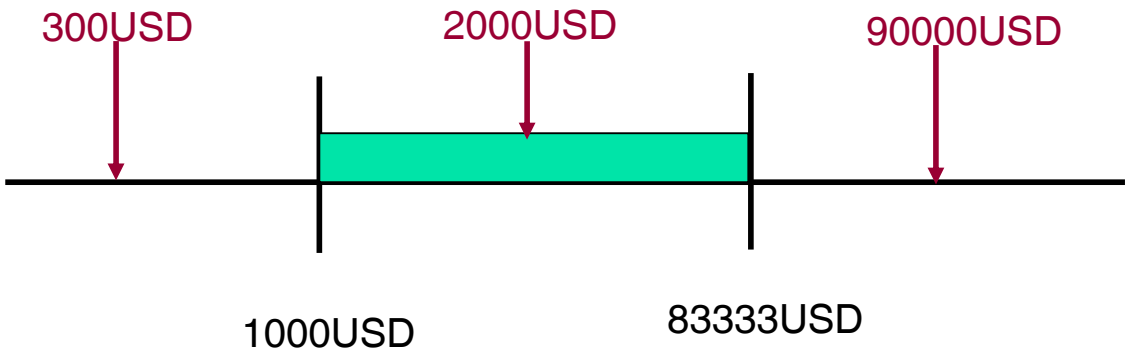
Thì nếu dùng 4 testcase đại diện của 4 lớp tương đương, ta mới kiểm thử được 4/100 lệnh mã nguồn của TPPM, mức độ phủ này chưa thể nói lên gì nhiều về TPPM!

Làm sao chọn testcase đại diện cho lớp tương đương ? Điều này phụ thuộc vào kiểu dữ liệu nhập. Ta hãy lần lượt xét 1 số kiểu dữ liệu nhập phổ biến.

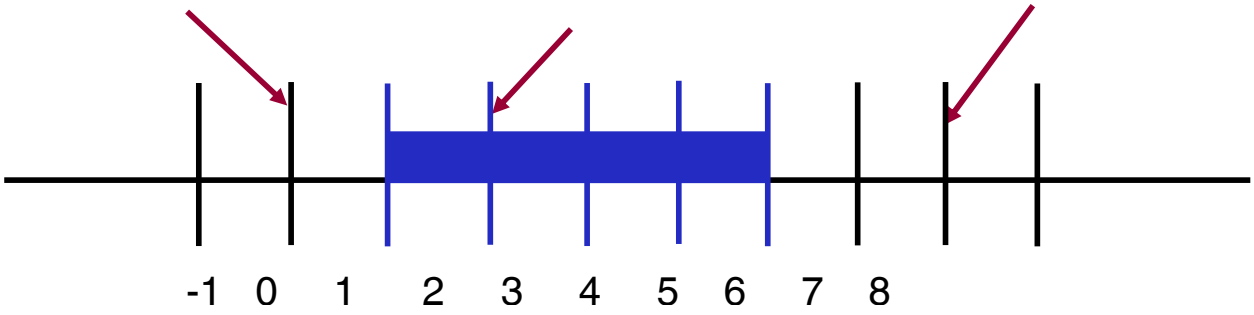
Thí dụ ta cần kiểm thử 1 TPPM "xét đơn cầm cố nhà" với đặc tả chức năng như sau : mỗi lần nhận 1 đơn xin cầm cố, TPPM sẽ ra quyết định chấp thuận nếu 4 điều kiện sau đều thỏa mãn :

1. Thu nhập hàng tháng của đương đơn nằm trong khoảng từ 1000$ đến 83333$.
2. số nhà xin cầm cố từ 1 đến 5.
3. Đương đơn phải là cá nhân, không được là hội, công ty hay người được ủy nhiệm (partnership, trust, corporation).
4. Loại nhà cầm cố phải là loại nhà cố định (single family, condo, townhouse), không xét loại nhà di động (treehouse, duplex, mobile home).

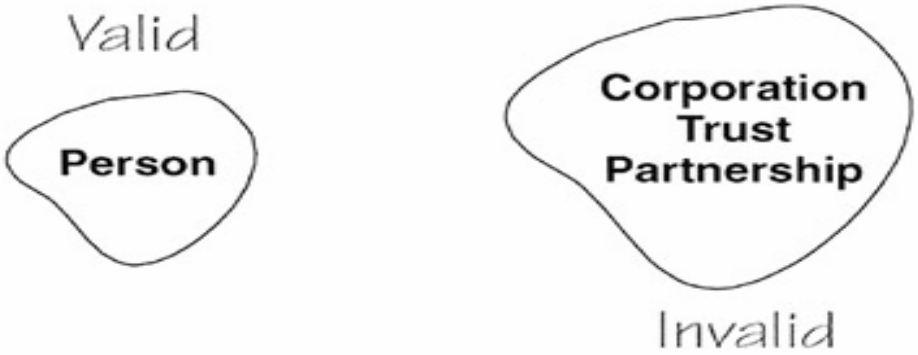
1. Nếu lớp tương đương được xác định bởi các dữ liệu nhập là số thực liên tục, thì ta chọn 1 testcase đại diện có giá trị nhập hợp lệ nằm trong khoảng liên tục các giá trị hợp lệ, và nếu muốn, 2 testcase miêu tả giá trị không hợp lệ nằm phía dưới và phía trên khoảng trị hợp lệ (số testcase cho mỗi lớp tương đương là từ 1 tới 3).



2. Nếu lớp tương đương được xác định bởi các dữ liệu nhập là số nguyên liên tục, trong trường hợp này ta chọn 1 testcase đại diện có giá trị nhập hợp lệ nằm trong khoảng liên tục các giá trị hợp lệ, và nếu muốn, 2 testcase miêu tả giá trị không hợp lệ nằm phía dưới và phía trên khoảng trị hợp lệ (số testcase cho mỗi lớp tương đương là từ 1 tới 3).

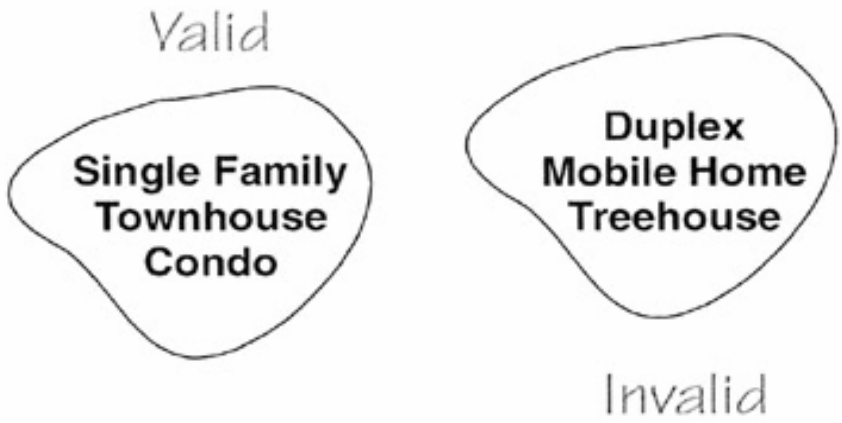


3. Nếu lớp tương đương được xác định bởi các dữ liệu dạng liệt kê rời rạc và không có mối quan hệ lẫn nhau gồm 1 trị hợp lệ và nhiều trị không hợp lệ. Trong trường hợp này ta chọn 1 testcase có giá trị nhập hợp lệ và nếu muốn, 2 testcase miêu tả 2 giá trị không hợp lệ nào đó, nhưng cho dù chọn 2 testcase nào cũng không đại diện tốt cho các trường hợp không hợp lệ còn lại (số testcase cho mỗi lớp tương đương là từ 1 tới 3).



4. Nếu lớp tương đương được xác định bởi các dữ liệu dạng liệt kê rời rạc và không có mối quan hệ lẫn nhau gồm n trị hợp lệ và m trị không hợp lệ. Trong trường hợp này ta chọn 1 testcase có giá trị nhập hợp lệ nào đó và nếu muốn, 2 testcase miêu tả 2 giá trị không hợp lệ nào đó, nhưng cho dù chọn các testcase nào cũng

không đại diện tốt cho các trường hợp hợp lệ và không hợp lệ còn lại (số testcase cho mỗi lớp tương đương là từ 1 tới 3).



Khi TPPM cần kiểm thử nhận nhiều dữ liệu nhập (thí dụ TPPM xét đơn cầm cố nhà ở slide trước có 4 loại dữ liệu nhập), ta định nghĩa các testcase độc lập cho các dữ liệu hay testcase dựa trên tổng hợp các dữ liệu nhập ?

Nếu định nghĩa các testcase độc lập trên từng loại dữ liệu nhập, số lượng testcase cần kiểm thử sẽ nhiều. Trong TPPM xét đơn cầm cố nhà, ta phải xử lý ít nhất là 3 testcase cho từng loại dữ liệu * 4 loại dữ liệu = 12 testcase.

Để giảm thiểu số lượng testcase nhưng vẫn đảm bảo chất lượng kiểm thử, người ta đề nghị chọn tescase như sau :

- 1 testcase cho tổ hợp các giá trị hợp lệ.
- n testcase cho tổ hợp các giá trị trong đó có 1 giá trị không hợp lệ, các giá trị còn lại hợp lệ, nên thay đổi các giá trị hợp lệ trong tổ hợp cho mỗi testcase.

Thí dụ TPPM xét đơn cầm cố nhà ở slide trước có 4 loại dữ liệu nhập), ta định nghĩa các testcase dựa trên tổ hợp các dữ liệu nhập như sau :

Thu nhập /tháng	Số lượng nhà	Đương đơn	Kiểu nhà	Kết quả
5.000	2	Person	Condo	Valid
100	1	Person	Single	Invalid

1.342	0	Person	Condo	Invalid
5.432	3	Corporation	Townhouse	Invalid
10.000	2	Person	Treehouse	Invalid

5.3 Kỹ thuật phân tích các giá trị ở biên

Kỹ thuật kiểm thử phân lớp tương đương là kỹ thuật cơ bản nhất, nó còn gợi ý chúng ta đến 1 kỹ thuật kiểm thử khác : phân tích các giá trị ở biên.

Kinh nghiệm trong cuộc sống đời thường cũng như trong lập trình các giải thuật lặp cho chúng ta biết rằng lỗi thường nằm ở biên (đầu hay cuối) của 1 khoảng liên tục nào đó (lớp tương đương). Do đó ta sẽ tập trung tạo các testcase ứng với những giá trị ở biên này.

Thí dụ xét đặc tả TPPM “quản lý nguồn nhân lực” ở slide 8, ta thấy đặc tả các luật đều bị lỗi ở các biên, thí dụ luật 1 qui định không thuê những người có tuổi từ 0 – 16, còn luật 2 qui định sẽ thuê bán thời gian những người từ 16-18 tuổi. Vậy người 16 tuổi được xử lý như thế nào bởi hệ thống ? Đã có nhập nhằng và mâu thuẩn trong các luật. Lỗi này do nắm bắt yêu cầu phần mềm sai.

Giả sử ta đã chỉnh sửa lại yêu cầu phần mềm như sau :

Tuổi ứng viên	Kết quả
0-15	Không thuê
16-17	Thuê dạng bán thời gian
18-54	Thuê toàn thời gian
55-99	Không thuê

Và đoạn code hiện thực sau :

```
if (0 < applicantAge && applicantAge < 15) kq ="NO";
if (16 < applicantAge && applicantAge <17) kq ="PART";
if (18 < applicantAge && applicantAge <54) kq ="FULL";
if (55 < applicantAge && applicantAge <99) kq ="NO";
```

Đoạn code hiện thực ở slide trước bị lỗi ở các gía trị biên (đúng ra là phải dùng điều kiện <= chứ không phải là <). Lỗi này thuộc về người hiện thực chương trình.

Cách đơn giản và hiệu quả để phát hiện lỗi ở slide trước là thanh tra mã nguồn (code inspection), mà ta sẽ đề cập trong chương 7. Ở đây ta chỉ trình bày kỹ thuật kiểm thử dựa trên các giá trị biên để phát hiện các lỗi này.

Ý tưởng của kỹ thuật kiểm thử dựa trên các trị biên là chỉ định nghĩa các testcase ứng với các giá trị ngay trên biên hay lân cận biên của từng lớp tương đương. Do đó kỹ thuật này chỉ thích hợp với các lớp tương đương xác định bởi các giá trị liên tục (số nguyên, số thực), chứ nó không thích hợp với lớp tương đương được xác định bởi các giá trị liệt kê mà không có mối quan hệ lẫn nhau.

Qui trình cụ thể để thực hiện kiểm thử dựa trên các giá trị ở biên :

- Nhận dạng các lớp tương đương dựa trên đặc tả về yêu cầu chức năng của TPPM.
- Nhận dạng 2 biên của mỗi lớp tương đương.
- Tạo các testcase cho mỗi biên của mỗi lớp tương đương :

  - 1 testcase cho giá trị biên.

  - 1 testcase ngay dưới biên.

  - 1 testcase ngay trên biên.
- Ý nghĩa ngay trên và ngay dưới biên phụ thuộc vào đơn vị đo lường cụ thể :

  - nếu là số nguyên, ngay trên và ngay dưới lệch biên 1 đơn vị.
  - nếu đơn vị tính là “$ và cent” thì ngay dưới của biên 5$ là 4.99$, ngay trên là 5.01$.

- nếu đơn vị là $ thì ngay dưới của 5$ là 4$, ngay trên 5$ là 6$.

Thí dụ dựa vào đặc tả của TPPM “quản lý nguồn nhân lực” :

Tuổi ứng viên	Kết quả
0-15	Không thuê
16-17	Thuê dạng bán thời gian
18-54	Thuê toàn thời gian
55-99	Không thuê

Ta sẽ định nghĩa các testcase tương ứng với các tuổi sau : {-1,0,1}, {14,15,16}, {15,16,17}, {16,17,18}, {17,18,19}, {53,54,55}, {54,55,56}, {98,99,100}.

Có nhiều testcase trùng nhau, nếu loại bỏ các testcase trùng nhau, ta còn : -1, 0, 1, 14, 15, 16, 17, 18, 19, 53, 54, 55, 56, 98, 99, 100 (16 testcase so với hàng trăm testcase nếu vẹt cạn).

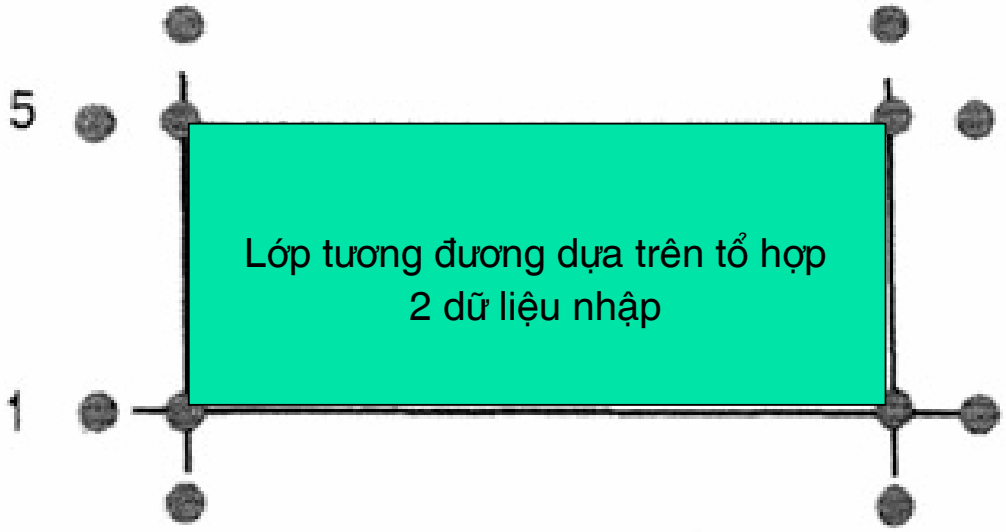
Khi TPPM cần kiểm thử nhận nhiều dữ liệu nhập (thí dụ TPPM xét đơn cầm cố nhà ở slide trước có 4 loại dữ liệu nhập), ta định nghĩa các testcase độc lập cho các dữ liệu hay testcase dựa trên tổng hợp các dữ liệu nhập ?

Nếu định nghĩa các testcase độc lập trên từng loại dữ liệu nhập, số lượng testcase cần kiểm thử sẽ nhiều. Trong TPPM xét đơn cầm cố nhà, ta phải xử lý ít nhất là 6 testcase cho từng loại dữ liệu * 4 loại dữ liệu = 24 testcase.

Để giảm thiểu số lượng testcase nhưng vẫn đảm bảo chất lượng kiểm thử, người ta đề nghị chọn tescase như sau :

- 1 số testcase cho các tổ hợp các giá trị biên.
- 1 số testcase cho các tổ hợp các giá trị ngay dưới và ngay trên biên.

TPPM xét đơn cầm cố nhà ở slide trước có 2 dữ liệu nhập liên tục là thu nhập hàng tháng và số lượng nhà. Tổng hợp 2 loại dữ liệu này theo góc nhìn đồ họa trực quan, ta thấy cần định nghĩa các testcase cho các trường hợp sau :



Thu nhập /tháng	Số lượng nhà	Kết quả	Diễn giải
$1,000	1	Valid	Min Thu nhập, min SoLN
$83,333	1	Valid	Max Thu nhập, min SoLN
$1,000	5	Valid	Min Thu nhập, max SoLN
$83,333	5	Valid	Max Thu nhập, max SoLN
$1,000	0	Invalid	Min Thu nhập, below min SoLN
$1,000	6	Invalid	Min Thu nhập, above max SoLN
$83,333	0	Invalid	Max Thu nhập, below min SoLN
$83,333	6	Invalid	Max Thu nhập, above max SoLN
$999	1	Invalid	Below min Thu nhập, min SoLN
$83,334	1	Invalid	Above max Thu nhập, min SoLN
$999	5	Invalid	Below min Thu nhập, max SoLN
$83,334	5	Invalid	Above max Thu nhập, max SoLN

5.4 Kỹ thuật dùng bảng quyết định (decision table)

Bảng quyết định là 1 công cụ rất hữu ích để đặc tả các yêu cầu phần mềm hoặc để đặc tả bảng thiết kế hệ thống phần mềm. Nó miêu tả các qui tắc nghiệp vụ phức tạp mà phần mềm phải thực hiện dưới dạng dễ đọc và dễ kiểm soát :

	Rule 1	Rule 2	…	Rule p
Conditions				
Condition-1				
…				
Condition-m				

Actions				
Action-1				
…				
Action-n				

Condition-1 tới Condition-m miêu tả m điều kiện dữ liệu nhập khác nhau có thể có. Action-1 tới Action-n miêu tả n hoạt động khác nhau mà hệ thống có thể thực hiện phụ thuộc vào tổ hợp điều kiện dữ liệu nhập nào. Mỗi cột miêu tả 1 luật cụ thể : tổ hợp điều kiện nhập cụ thể và các hoạt động cụ thể cần thực hiện.

Lưu ý các hoạt động cần thực hiện không phụ thuộc vào thứ tự các điều kiện nhập, nó chỉ phụ thuộc vào giá trị các điều kiện nhập.

Tương tự, các hoạt động cần thực hiện không phụ thuộc vào trạng thái hiện hành của TPPM, chúng cũng không phụ thuộc vào các điều kiện nhập đã có trước đó.

Chúng ta sẽ lấy 1 thí dụ cụ thể để làm rõ bảng quyết định. Giả sử TPPM cần kiểm thử là phân hệ chức năng nhỏ của công ty bảo hiểm : nó sẽ khuyến mãi cho những chủ xe (cũng là tài xế) nếu họ thỏa ít nhất 1 trong 2 điều kiện : đã lập gia đình / là sinh viên giỏi. Mỗi dữ liệu nhập là 1 giá trị luận lý, nên bảng quyết định chỉ cần có 4 cột, miêu tả 4 luật khác nhau :

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
Married?	Yes	Yes	No	No
Good Student?	Yes	No	Yes	No
Actions				
Discount ($)	60	25	50	0

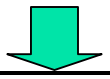
Qui trình cụ thể để thực hiện kiểm thử dùng bảng quyết định :

1. Tìm bảng quyết định từ đặc tả về yêu cầu chức năng của TPPM hay từ bảng thiết kế TPPM. Nếu chưa có thì xây dựng nó dựa vào đặc tả về yêu cầu chức năng hay dựa vào bảng thiết kế TPPM.

2. Từ bảng quyết định chuyển thành bảng các testcase trong đó mỗi cột miêu tả 1 luật được chuyển thành 1 đến n cột miêu tả các testcase tương ứng với luật đó :

   - nếu điều kiện nhập là trị luận lý thì mỗi cột luật được chuyển thành 1 cột testcase.
   - nếu điều kiện nhập là 1 lớp tương đương (nhiều giá trị liên tục) thì mỗi cột luật được chuyển thành nhiều testcase dựa trên kỹ thuật lớp tương đương hay kỹ thuật giá trị biên.

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
Cond 1	Yes	Yes	No	No
Cond 2	Yes	No	Yes	No
Actions				
Action-1	60	25	50	0



	TC1	**TC2**	**TC3**	**TC4**
Conditions				
Married?	Yes	Yes	No	No
Good Student?	Yes	No	Yes	No
Actions				
Discount ($)	60	25	50	0

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
Cond 1	0–1	1–10	10–100	100–1000
Cond 2	<5	5	6 or 7	>7
Actions				
Action-1	Do X	Do Y	Do X	Do Z

	TC1	TC2	TC3	TC4
Conditions				
Cond 1	0	5	50	500
Cond 2	3	5	7	10
Actions				
Action-1	Do X	Do Y	Do X	Do Z

5.5 Kỹ thuật kiểm thử các bộ n thần kỳ (pairwise)

Chúng ta hãy kiểm thử 1 website với đặc tả yêu cầu như sau :

1. Phải chạy tốt trên 8 trình duyệt khác nhau : Internet Explorer 5.0, 5.5, and 6.0, Netscape 6.0, 6.1, and 7.0, Mozilla 1.1, and Opera 7.
2. Phải chạy tốt ở 3 chế độ plug-ins : RealPlayer, MediaPlayer, none.
3. Phải chạy tốt trên 6 HĐH máy client : Windows 95, 98, ME, NT, 2000, and XP.
4. Phải chạy tốt trên 3 web server khác nhau : IIS, Apache, and WebLogic.
5. Phải chạy tốt trên 3 HĐH máy server : Windows NT, 2000, and Linux.

Như vậy để kiểm thử đầy đủ đặc tả yêu cầu, ta phải kiểm thử website trên 8*3*6*3*3 = 1296 cấu hình khác nhau.

Một thí dụ khác : hãy kiểm thử 1 hệ thống xử lý dữ liệu ngân hàng có đặc tả yêu cầu như sau :

1. Phải xử lý tốt 4 loại khách hàng là consumers, very important consumers, businesses, and non-profits.
2. Phải xử lý tốt 5 loại tài khoản : checking, savings, mortgages, consumer loans, and commercial loans.

3. Phải chạy tốt ở 6 bang khác nhau với chế độ khác nhau : California, Nevada, Utah, Idaho, Arizona, and New Mexico.

Như vậy để kiểm thử đầy đủ đặc tả yêu cầu, ta phải kiểm thử hệ thống xử lý dữ liệu ngân hàng trên 4*5*6 = 120 cấu hình khác nhau.

Một thí dụ khác : hãy kiểm thử 1 phần mềm hướng đối tượng có chi tiết thiết kế như sau : Đối tượng class A sẽ gởi thông điệp đến đối tượng class X dùng tham số là đối tượng class P.

1. Có 3 class con của A là B, C, D.
2. Có 4 class con của P là Q, R, S, T.
3. Có 2 class con của X là Y, Z.

Như vậy để kiểm thử đầy đủ đặc tả thiết kế, ta phải kiểm thử phần mềm trên với 4*5*3 = 60 trường hợp khác nhau.

Thông qua 3 thí dụ kiểm thử vừa giới thiệu, ta thấy số lượng kiểm thử là rất lớn, thường ta không có đủ tài nguyên về con người, về trang thiết bị và thời gian để kiểm thử hết tất cả. Buộc lòng ta phải tìm tập con các trường hợp cần kiểm thử, nhưng làm sao xác định được tập con cần kiểm thử thỏa điều kiện là số lượng càng ít càng tốt mà chất lượng kiểm thử không bị suy giảm nhiều.

Thật ra có nhiều chiến lược kiểm thử khác nhau :

1. Không kiểm thử bộ n nào cả (vì khiếp sợ số lượng quá lớn).
2. Kiểm thử tất cả bộ n, như vậy sẽ delay dự án quá lâu và làm mất thị trường.
3. Kiểm thử 1 hay 2 bộ n và hy vọng là đủ.
4. Chọn các bộ n đã kiểm thử rồi (cho project khác).
5. Chọn các bộ n dễ dàng tạo ra và kiểm thử nhất, mà không để ý chất lượng của chúng ra sao.

6. Tạo tất cả bộ n và chọn 1 ít bộ n đầu tiên trong danh sách.
7. Tạo tất cả bộ n và chọn 1 ít bộ n theo cơ chế ngẫu nhiên.
8. Chọn 1 tập con đủ nhỏ bộ n mà tạo được điều kỳ diệu là chất lượng kiểm thử không bị giảm sút. Bằng cách nào, nếu có ?

Có 2 phương pháp xác định được tập con các bộ n thần kỳ cần kiểm thử :

1. Dùng ma trận trực giao (orthogonal array).
2. Dùng tiện ích Allpairs.

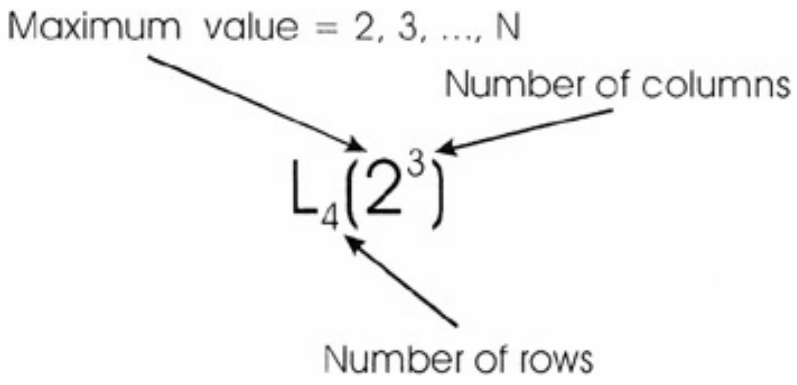
Ma trận trực giao là 1 bảng 2 chiều gồm n hàng * m cột các giá trị số nguyên với 1 đặc tính rất đặc biệt là : 2 cột bất kỳ trong ma trận đều chứa đúng các bộ đôi xác định trước.

Thí dụ, xét ma trận trực giao sau 4 hàng * 3 cột sau :

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Ta thấy 2 cột bất kỳ trong ma trận đều chứa đúng 4 bộ đôi sau đây (1,1), (1,2), (2,1), (2,2).

Ta miêu tả 1 ma trận trực giao như sau :



Số lượng các giá trị trong các cột không nhất thiết phải giống nhau, chúng ta có thể trộn nhiều loại cột chứa phạm vi giá trị khác nhau. Thí dụ ta dùng ký hiệu L18(21,37) để miêu tả ma trận trực giao có 18 hàng, mỗi hàng chứa 1 cột 2 giá trị và 7 cột chứa 3 giá trị.

Qui trình kiểm thử dùng các bộ n thần kỳ dựa vào ma trận trực giao gồm các bước chính :

1. Nhận dạng các biến dữ liệu. Thí dụ bài toán kiểm thử website trong các slide trước chứa 5 biến là Browser, Plug-in, Client operating system, Web Server, và Server operating system.
2. Xác định các giá trị cho từng biến. Thí dụ biến Browser có 8 giá trị lần lượt là : IE 5.0, IE 5.5, IE 6.0, Netscape 6.0, 6.1, 7.0, Mozilla 1.1, và Opera 7.
3. Tìm ma trận trực giao (trên mạng hay trong thư viện cá nhân) có mỗi cột cho mỗi biến, và số giá trị trong cột tương ứng với số giá trị của biến tương ứng. Thí dụ ta cần tìm ma trận trực giao Lx (816133) cho bài toán kiểm thử website.
4. Nếu không có ma trận thỏa mãn chính xác yêu cầu thì chọn ma trận trực giao có số cột tương đương, nhưng số lượng giá trị trong từng cột lớn hơn 1 chút cũng được. Cụ thể chưa ai tạo ra được ma trận trực giao có kích thước Lx (816133), do đó ta có thể tìm ma trận trực giao với kích thước L64(8243).
5. Ánh xạ bài toán kiểm thử vào ma trận trực giao bằng cách thay thế giá trị số bằng giá trị ngữ nghĩa. Thí dụ ta dùng cột 3 chứa 4 giá trị 1-4 để miêu tả biến Web Server, trong trường hợp này ta thế các giá trị 1-4 thành 4 giá trị ngữ nghĩa là IIS, Apache, WebLogic, Not used.
6. Xây dựng các test cases và kiểm thử chúng.

Thay vì dùng ma trận trực giao để nhận dạng các testcase, ta có thể dùng 1 thuật giải để sinh tự động các testcase.

Thí dụ James Bach đã cung cấp 1 tool tạo tự động tất cả các tổ hợp bộ n, bạn có thể download tool này từ địa chỉ http://www.satisfice.com và cái tiện ích này vào máy.

Để chuẩn bị dữ liệu nhập cho tiện ích, ta có thể Excel định nghĩa các biến dữ liệu cùng tập các giá trị nhập có thể có của từng biến theo dạng cột/hàng, sau đó lưu kết quả dạng văn bản thô *.txt. Chạy tiện ích theo dạng hàng lệnh như sau :

```
Allpairs input.txt > output.txt
```

Dùng 1 trình soạn trhảo văn bản mở file output.txt và xem kết quả.

5.6 Kết chương

Chương này đã giới thiệu những vấn đề tổng quát về kiểm thử hộp đen 1 TPPM.

Chúng ta đã giới thiệu chi tiết cụ thể về 4 kỹ thuật kiểm thử hộp đen được dùng phổ biến là kỹ thuật phân lớp tương đương, kỹ thuật phân tích các giá trị ở biên, kỹ thuật dùng bảng quyết định, và kỹ thuật kiểm thử các bộ n thần kỳ.

Ứng với mỗi kỹ thuật kiểm thử, chúng ta cũng đã giới thiệu 1 thí dụ cụ thể để demo qui trình thực hiện kỹ thuật kiểm thử tương ứng.

Chương 6

Kỹ thuật kiểm thử hộp đen (tt)

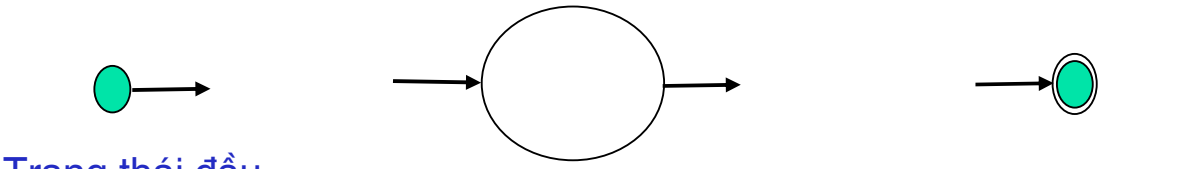
6.1 Kỹ thuật dùng lược đồ chuyển trạng thái

Cũng giống như bảng quyết định, lược đồ chuyển trạng thái là 1 công cụ rất hữu ích để đặc tả các yêu cầu phần mềm hoặc để đặc tả bảng thiết kế hệ thống phần mềm.

Thay vì miêu tả các qui tắc nghiệp vụ phức tạp mà phần mềm phải thực hiện dưới dạng dễ đọc và dễ kiểm soát như bảng quyết định, lược đồ chuyển trạng thái ghi nhận các sự kiện xảy ra, rồi được hệ thống xử lý cũng như những đáp ứng của hệ thống.

Khi hệ thống phải nhớ trạng thái trước đó của mình, hay phải biết trình tự các hoạt động nào là hợp lệ, trình tự nào là không hợp lệ thì lược đồ chuyển trạng thái là rất thích hợp.

Lược đồ chuyển trạng thái được cấu thành từ các thành phần cơ bản sau đây :



Trạng thái đầu

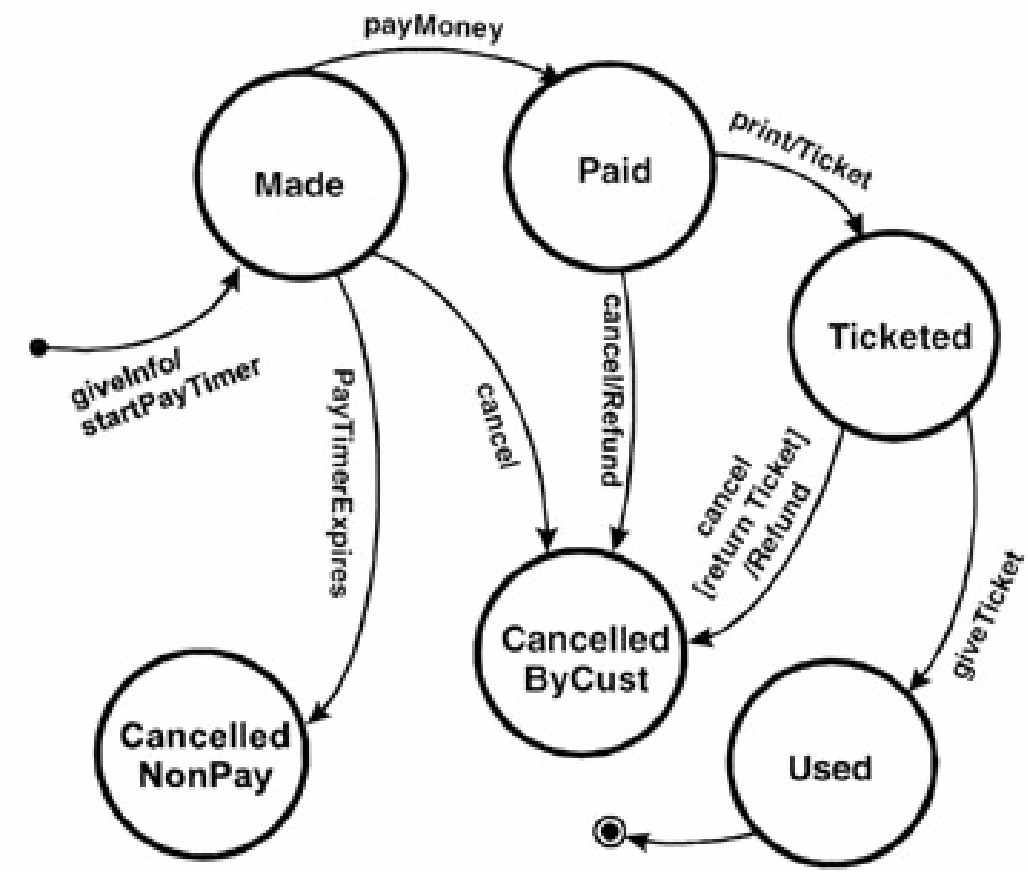
Trạng thái trung gian

Trạng thái cuối

Ta có thể đặt tên nhận dạng cho từng trạng thái trung gian, miêu tả điều kiện chuyển trạng thái kèm theo từng cung chuyển trạng thái.

Ta có thể miêu tả hành động cần thực hiện kết hợp với việc chuyển trạng thái.

Lược đồ chuyển trạng thái của TPPM đặt mua vé máy bay :



TPPM đặt mua vé máy bay có 6 trạng thái khác nhau :

1. Made :
   - điều kiện chuyển đến : sau khi người dùng đã nhập thông tin khách hàng.
   - Hành động cần thực hiện kèm theo : khởi động timer T0 đếm thời gian giữ trạng thái.
2. Cancelled (NonPay) :
   - điều kiện chuyển đến : sau khi timer T0 đã hết.
   - Hành động cần thực hiện kèm theo : null.
3. Paid :
   - điều kiện chuyển đến : sau khi người dùng đã thanh toán tiền.

- Hành động cần thực hiện kèm theo : null.

4. Cancelled (ByCustomer) :
   - điều kiện chuyển đến : sau khi người dùng đã cancel.
   - Hành động cần thực hiện kèm theo : null.
5. Ticketed :
   - điều kiện chuyển đến : sau khi in vé xong.
   - Kết quả kèm theo : vé máy bay.
6. Used :
   - điều kiện chuyển đến : sau khi người dùng đã dùng vé.
   - Hành động cần thực hiện kèm theo : null.

Trong khi lược đồ chuyển trạng thái là cách thức miêu tả hành vi của TPPM dễ hiểu và dễ đọc thì 1 dạng khác - bảng chuyển trạng thái – có thể miêu tả hành vi của TPPM hệ thống hơn và dễ xử lý tự động hơn.

Bảng chuyển trạng thái gồm 4 cột : trạng thái hiện hành, sự kiện xảy ra, hành động cần thực hiện/kết quả thu được, trạng thái kế tiếp.

Thí dụ lược đồ chuyển trạng thái ở slide trước có thể chuyển thành bảng chuyển trạng thái :

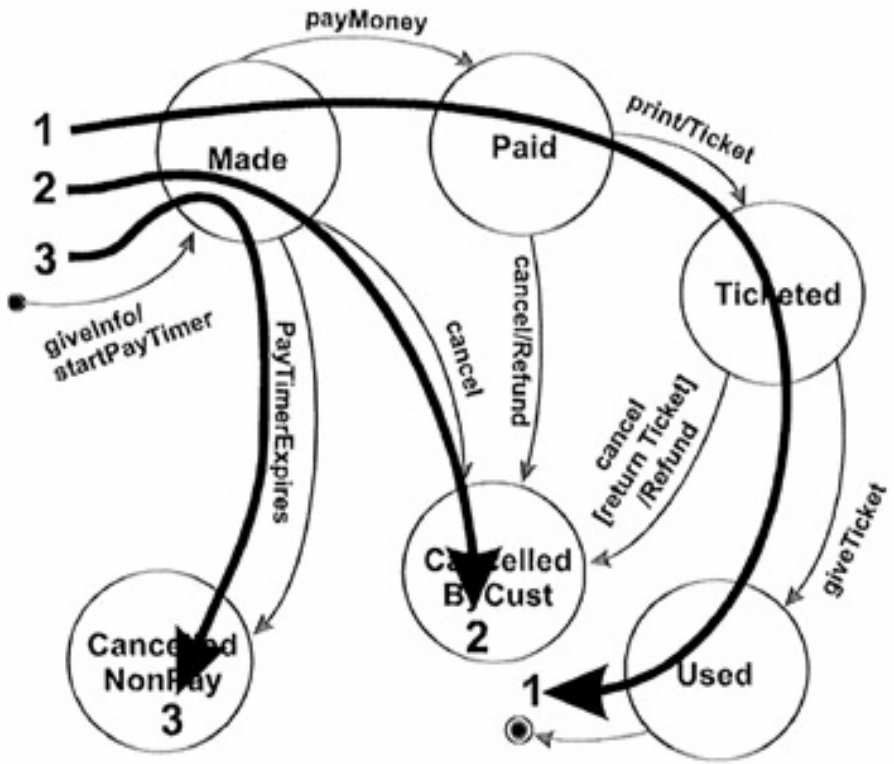
Current State	Event	Action	Next State
null	giveInfo	startPayTimer	Made
null	payMoney	--	null
null	print	--	null
null	giveTicket	--	null
null	cancel	--	null
null	PayTimerExpires	--	null
Made	giveInfo	--	Made

Current State	Event	Action	Next State
Made	payMoney	--	Paid
Made	print	--	Made
Made	giveTicket	--	Made
Made	cancel	--	Can-Cust
Made	PayTimerExpires	--	Can-NonPay
Paid	giveInfo	--	Paid
Paid	payMoney	--	Paid
Paid	print	Ticket	Ticketed
Paid	giveTicket	--	Paid
Paid	cancel	Refund	Can-Cust
Paid	PayTimerExpires	--	Paid
Ticketed	giveInfo	--	Ticketed
Ticketed	payMoney	--	Ticketed
Ticketed	print	--	Ticketed
Ticketed	giveTicket	--	Used
Ticketed	cancel	Refund	Can-Cust
Ticketed	PayTimerExpires	--	Ticketed
Used	giveInfo	--	Used
Used	payMoney	--	Used
Used	print	--	Used
Used	giveTicket	--	Used
Used	cancel	--	Used
Used	PayTimerExpires	--	Used
Can-NonPay	giveInfo	--	Can-NonPay
Can-NonPay	payMoney	--	Can-NonPay
Can-NonPay	print	--	Can-NonPay
Can-NonPay	giveTicket	--	Can-NonPay
Can-NonPay	cancel	--	Can-NonPay
Can-NonPay	PayTimerExpires	--	Can-NonPay

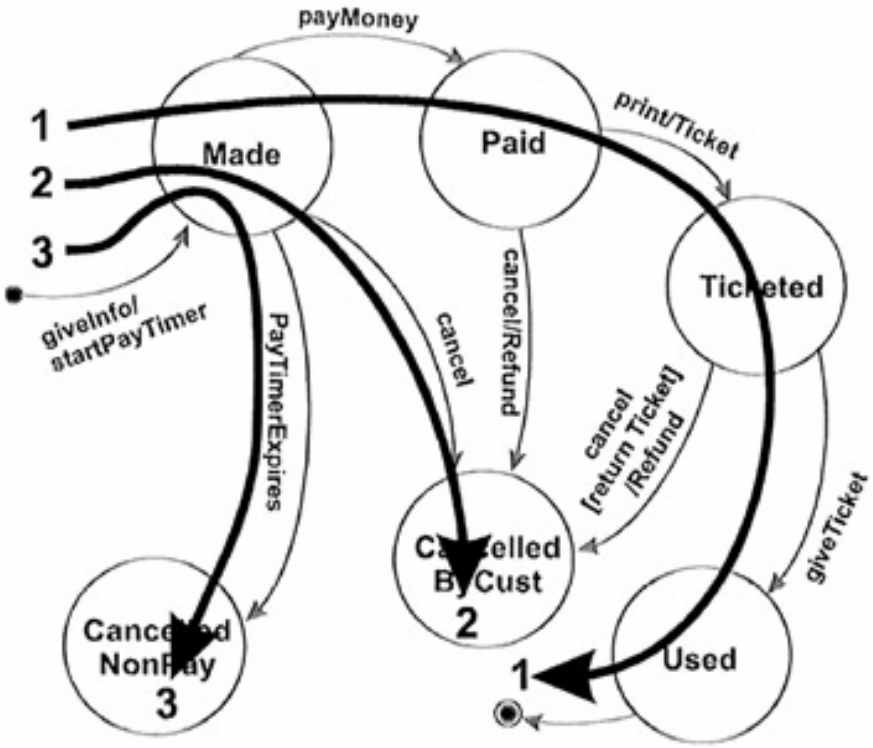
Current State	Event	Action	Next State
Can-Cust	giveInfo	--	Can-Cust
Can-Cust	payMoney	--	Can-Cust
Can-Cust	print	--	Can-Cust
Can-Cust	giveTicket	--	Can-Cust
Can-Cust	cancel	--	Can-Cust
Can-Cust	PayTimerExpires	--	Can-Cust

Dựa vào lược đồ chuyển trạng thái, ta có thể dễ dàng định nghĩa các testcase.

1. Phủ cấp 1 : tạo các testcase sao cho mỗi trạng thái đều xảy ra ít nhất 1 lần. Thí dụ 3 tescase sau sẽ kiểm thử được TPPM đạt phủ cấp 1 :



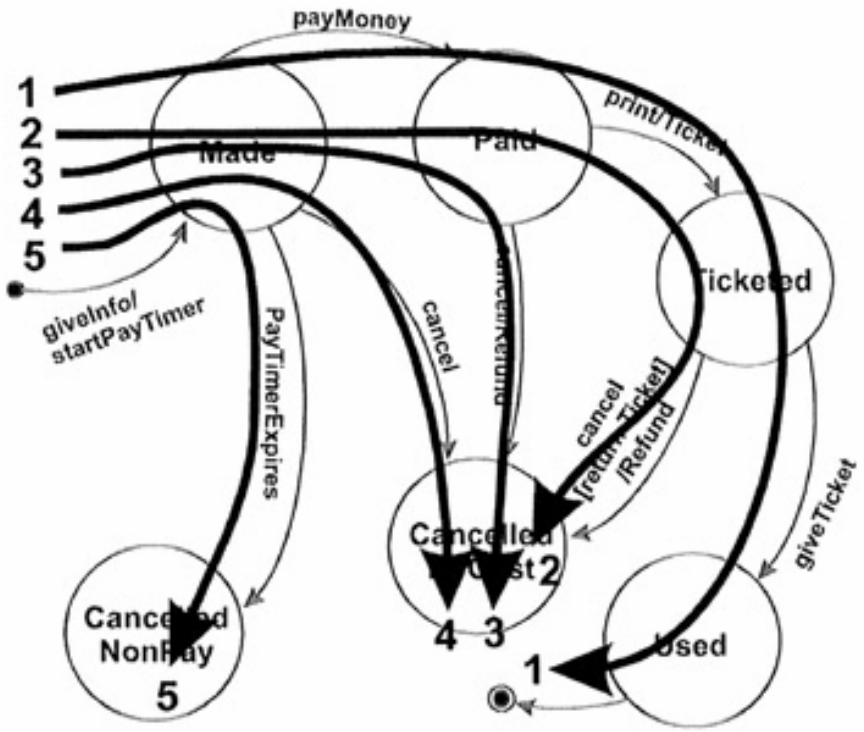
2. Phủ cấp 2 : tạo các testcase sao cho mỗi sự kiện đều xảy ra ít nhất 1 lần. Thí dụ 3 tescase sau sẽ kiểm thử được TPPM đạt phủ cấp 2 :



3. Phủ cấp 3 : tạo các testcase sao cho tất cả các path chuyển đều được kiểm thử. 1 path chuyển là 1 đường chuyển trạng thái xác định, bắt đầu từ trạng thái nhập và kết thúc ở trạng thái kết thúc.

   Đây là phủ tốt nhất vì đã vét cạn mọi khả năng hoạt động của TPPM, tuy nhiên không khả thi vì 1 path chuyển có thể lặp vòng.

4. Phủ cấp 4 : tạo các testcase sao cho mỗi path chuyển tuyến tính đều xảy ra ít nhất 1 lần. Thí dụ 5 tescase sau sẽ kiểm thử được TPPM đạt phủ cấp 4 :



Dựa vào bảng chuyển trạng thái, ta cũng có thể dễ dàng định nghĩa các testcase.

Current State	Event	Action	Next State
null	giveInfo	startPayTimer	Made
null	payMoney	--	null
null	print	--	null
null	giveTicket	--	null
null	cancel	--	null
null	PayTimerExpires	--	null
Made	giveInfo	--	Made
Made	payMoney	--	Paid
Made	print	--	Made
Made	giveTicket	--	Made
Made	cancel	--	Can-Cust
Made	PayTimerExpires	--	Can-NonPay
Paid	giveInfo	--	Paid

Current State	Event	Action	Next State
Paid	payMoney	--	Paid
Paid	print	Ticket	Ticketed
Paid	giveTicket	--	Paid
Paid	cancel	Refund	Can-Cust
Paid	PayTimerExpires	--	Paid
Ticketed	giveInfo	--	Ticketed
Ticketed	payMoney	--	Ticketed
Ticketed	print	--	Ticketed
Ticketed	giveTicket	--	Used
Ticketed	cancel	Refund	Can-Cust
Ticketed	PayTimerExpires	--	Ticketed
Used	giveInfo	--	Used
Used	payMoney	--	Used
Used	print	--	Used
Used	giveTicket	--	Used
Used	cancel	--	Used
Used	PayTimerExpires	--	Used
Can-NonPay	giveInfo	--	Can-NonPay
Can-NonPay	payMoney	--	Can-NonPay
Can-NonPay	print	--	Can-NonPay
Can-NonPay	giveTicket	--	Can-NonPay
Can-NonPay	cancel	--	Can-NonPay
Can-NonPay	PayTimerExpires	--	Can-NonPay
Can-Cust	giveInfo	--	Can-Cust
Can-Cust	payMoney	--	Can-Cust
Can-Cust	print	--	Can-Cust

Current State	Event	Action	Next State
Can-Cust	giveTicket	--	Can-Cust
Can-Cust	cancel	--	Can-Cust
Can-Cust	PayTimerExpires	--	Can-Cust

6.2 Kỹ thuật phân tích vùng (Domain Analysis)

Như ta đã biết, 2 kỹ thuật kiểm thử phân lớp tương đương và phân tích giá trị biên chủ yếu xử lý các biến dữ liệu độc lập, rời rạc. Tuy nhiên thường thì các biến dữ liệu có mối quan hệ với nhau, do đó cách tốt nhất là nên tổ hợp chúng để kiểm thử :

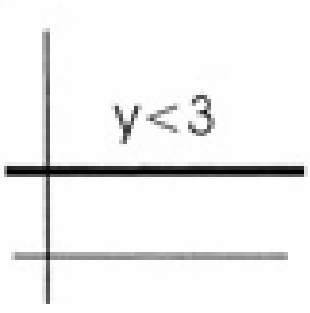
- Nếu tạo các testcase cho từng biến dữ liệu độc lập thì số lượng testcase sẽ quá nhiều.
- Các biến dữ liệu thường tương tác nhau, giá trị của biến này có thể ràng buộc giá trị của biến kia, do đó nếu kiểm thử chúng độc lập thì không thể phát hiện lỗi liên quan đến sự ràng buộc này.

Kỹ thuật phân tích vùng rất thích hợp trong việc xác định các testcase hiệu quả khi các biến dữ liệu có sự tương tác lẫn nhau. Nó được xây dựng trên 2 kỹ thuật kiểm thử được đề cập ở trên và tổng quát hóa chúng để kiểm thử đồng thời n biến dữ liệu.

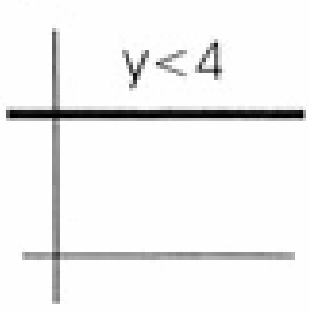
Xét trường hợp 2 biến dữ liệu tương tác nhau, ta thấy có 4 loại lỗi sau :

1. Biên ngang bị dịch lên hay xuống

Đúng

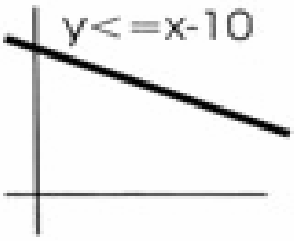


Sai

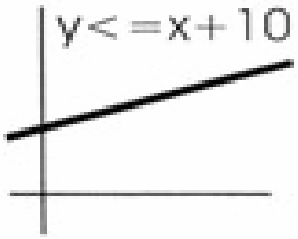


2. Biên nghiêng sai góc

Đúng

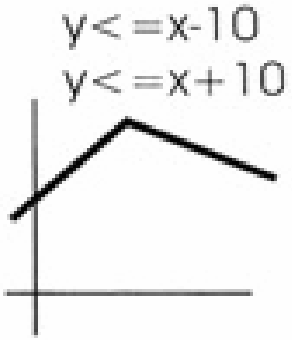


Sai

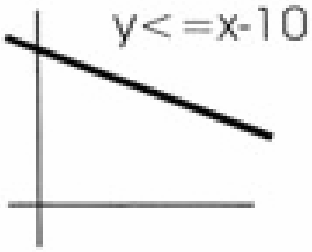


3. Thiếu biên

Đúng

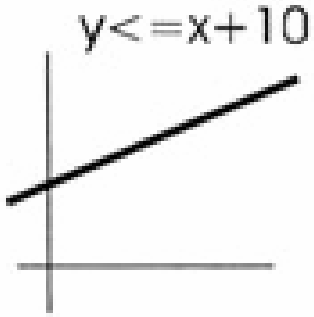


Sai

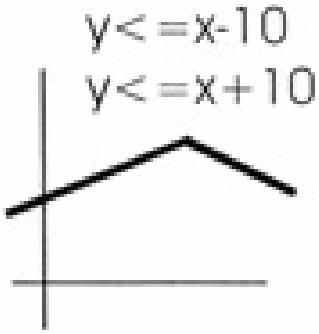


4. Thừa biên

Đúng



Sai



Ta định nghĩa 1 số thuật ngữ :

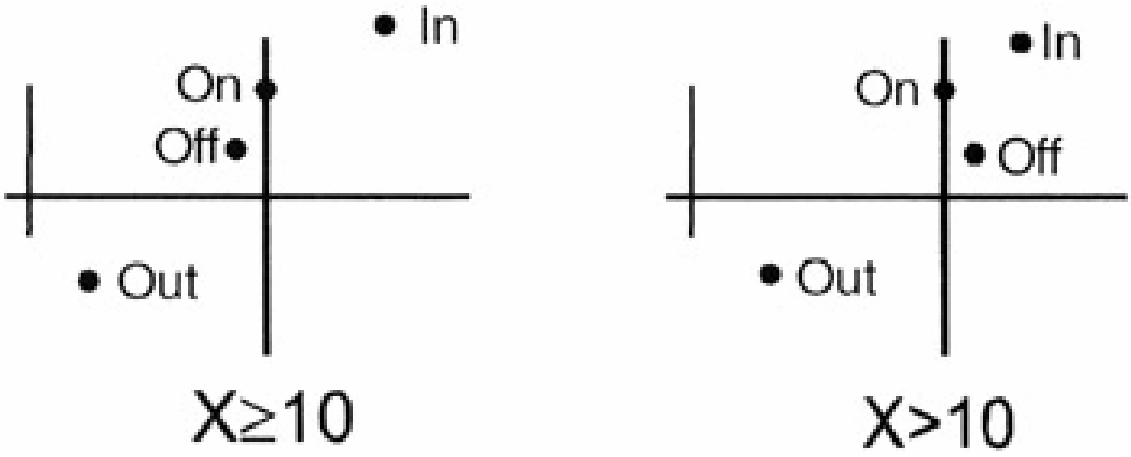
1. Điểm on : là điểm nằm trên biên

2. Điểm off : là điểm không nằm trên biên
3. Điểm in : là điểm thỏa mọi điều kiện biên nhưng không nằm trên biên.
4. Điểm out : là điểm không thỏa bất kỳ điều kiện biên.

Việc chọn điểm on và off thường phức tạp hơn chúng ta nghĩ :

- Nếu biên đóng (dùng toán tử so sánh có yếu tố =), thì điểm on nằm trên biên và thuộc vùng xử lý. Trong trường hợp này, ta chọn điểm off nằm ngoài vùng xử lý.
- Nếu biên mở (dùng toán tử so sánh không có yếu tố =), thì điểm on nằm trên biên nhưng không thuộc vùng xử lý. Trong trường hợp này ta chọn điểm off nằm trong vùng xử lý.

Thí dụ về các điểm on, off, in và out :



Kỹ thuật phân tích vùng yêu cầu chúng ta chọn các tescase theo cách thức sau :

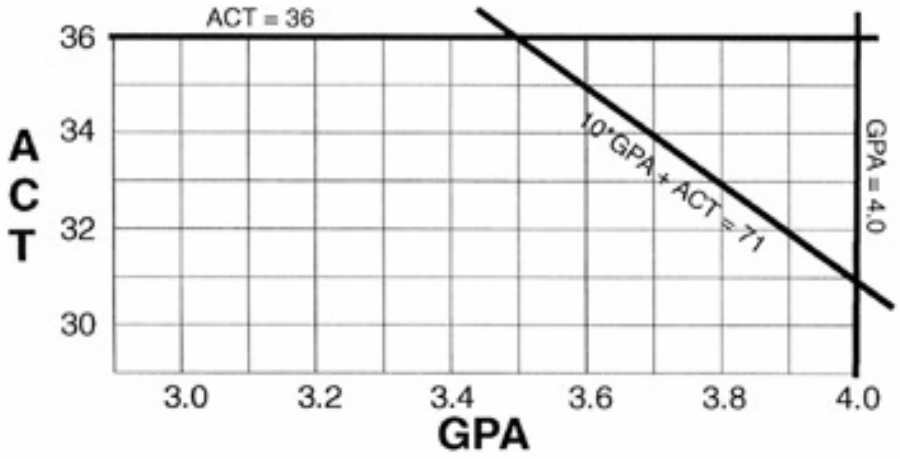
1. Ứng với mỗi điều kiện <, >, ≤, ≥, chọn 1 điểm on và 1 điểm off.
2. Ứng với mỗi điều kiện =, ≠, chọn 1 điểm on, 2 điểm off ngay trên và ngay dưới điểm on.

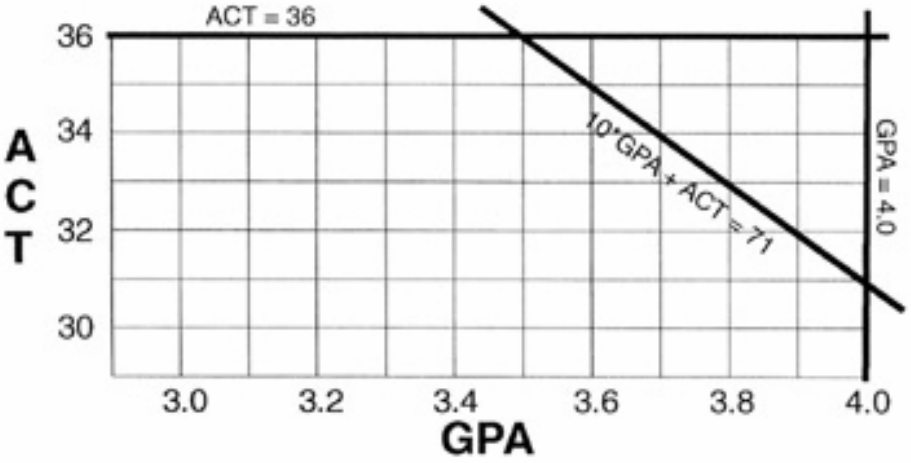
Binder đề nghị 1 bảng rất hữu ích – ma trận kiểm thử vùng :

Variable/ Condition Type			Test Cases															
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X1	C11	On																
		Off																
	C12	On																
		Off																
	...	On																
		Off																
	C1m	On																
		Off																
	Typical	In																
X2	C21	On																
		Off																
	C22	On																
		Off																
	...	On																
		Off																
	C2m	On																
		Off																
	Typical	In																
Expected Result																		

Thí dụ, TPPM xét kết quả đậu đại học theo tiêu chuẩn sau :

- 10*GPA + ACT >= 71
- GPA : điểm trung bình tích lũy của lớp phổ thông (<=4.0)
- ACT : điểm thi tuyển vào đại học (<= 36).





		GPA						
		0.0 – 3.4	3.5	3.6	3.7	3.8	3.9	4.0
ACT Score	36							
	35							
	34							
	33							
	32							
	31							
	0 - 30							

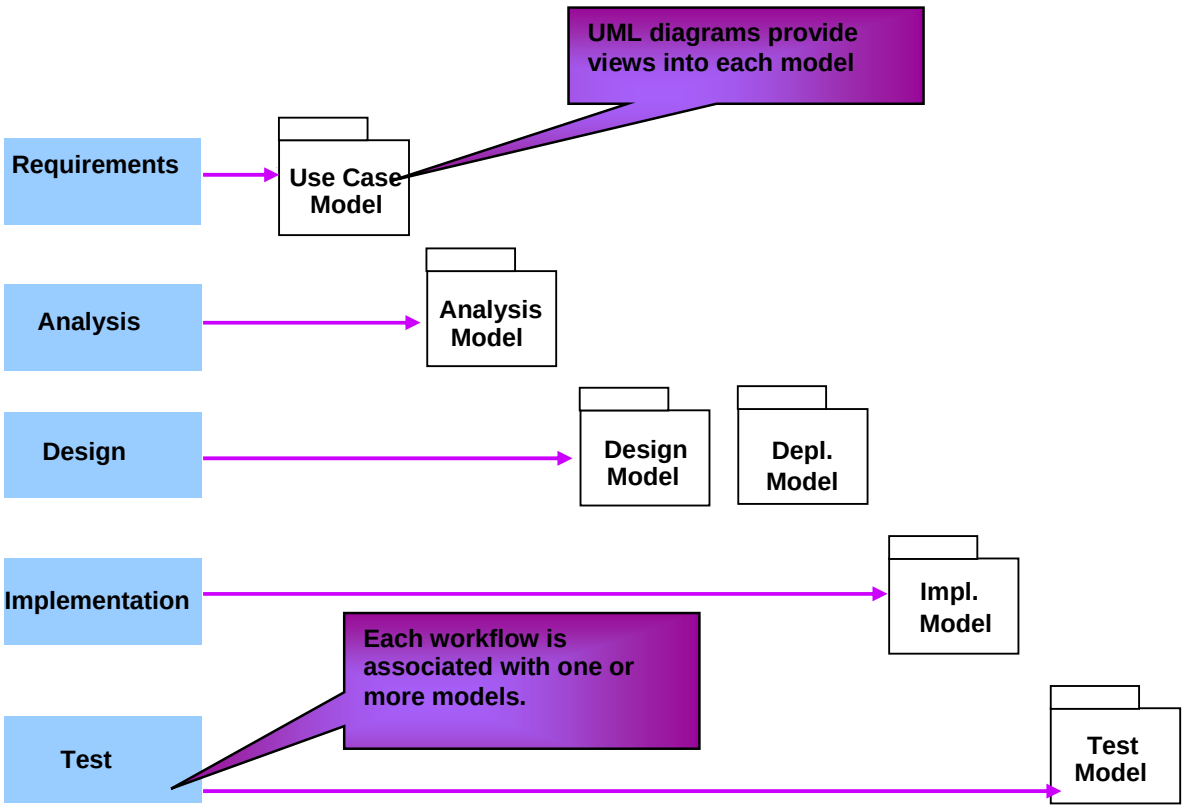
			1	2	3	4	5	6
GPA	GPA ≤ 4.0	On	4.0					
		Off		4.1				
	Typical	In			3.7	3.8	3.8	3.9
ACT	ACT ≤ 36	On			36			
		Off				37		
	Typical	In	34	33			32	35
GPA/ACT	10*GPA + ACT ≥ 71	On						
		Off						
	Typical	In	3.9/35	3.8/34	3.6/36	3.8/34	3.7/34	3.8/32
Expected Result			Admit	Reject	Admit	Reject	Admit	Reject

6.3 Kỹ thuật dùng thông tin trong use-case

Trong qui trình phát triển phần mềm hợp nhất, ta thực hiện nhiều workflows khác nhau : nắm bắt yêu cầu phần mềm, phân tích từng yêu cầu, thiết kế chi tiết để giải quyết từng yêu cầu, hiện thực từng phần bảng thiết kế, kiểm thử kết quả.

Mỗi workflows, thậm chí mỗi lần lập thực hiện 1 workflow, ta phải có kết quả, kết quả này phải được miêu tả ở dạng dễ đọc, dễ hiểu bởi nhiều người và phải cố gắng đơn nghĩa để tránh nhập nhằng.

Ta dùng khái niệm mô hình để miêu tả hệ thống phần mềm theo một góc nhìn nào đó.

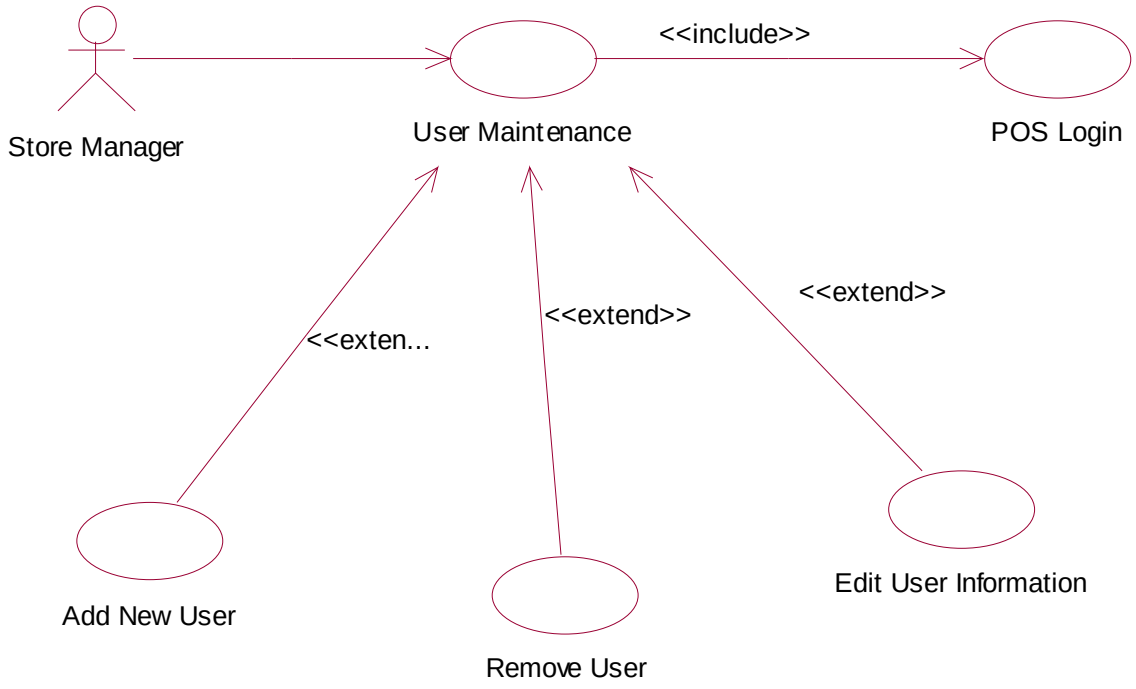


Về nguyên tắc, khi kiểm thử 1 thành phần phần mềm, ta có thể tận dụng bất kỳ thông tin nào trong bất kỳ mô hình nào. Kỹ thuật kiểm thử dùng thông tin trong use-case là kỹ thuật định nghĩa các testcase dựa vào các kịch bản thực hiện usecase.

Như chúng ta biết, mô hình usecase miêu tả hệ thống phần mềm theo góc nhìn bên ngoài : nó cung cấp các chức năng nào cho những “user” nào. Thành phần thiết yếu nhất của mô hình usecase là các lược đồ usecase.

Mỗi lược đồ usecsae thể hiện 1 bộ phận nhỏ của phần mềm : nó bao gồm nhiều chức năng và các chức năng này tương tác với các actor nào.

Thí dụ lược đồ usecase liên quan đến bộ phận chức năng quản lý khách hàng trong hệ thống thương mại điện tử.



Trong lược đồ usecase, mỗi usecase thể hiện 1 chức năng mà bên ngoài có thể truy xuất, tuy nhiên mỗi usecase chỉ được miêu tả ở dạng tối giản : gồm 1 hình ellipse và tên gợi nhớ sơ bộ về chức năng của usecase.

Để hiểu đầy đủ hơn về usecase, người ta cần đặc tả usecase ở 1 dạng chi tiết nào đó. Rất tiết là hiện nay, mỗi nơi mỗi khác, chưa có 1 chuẩn nào được mọi người chấp thuận.

Ở đây, ta hãy dùng khuôn mẫu chi tiết để đặc tả usecase do Alistair Cockburn đề nghị trong sách "Writing Effective Use Cases".

Use Case Component	**Description**
Use Case Number or Identifier	A unique identifier for this use case
Use Case Name	The name should be the goal stated as a short active verb phrase
Goal in Context	A more detailed statement of the goal if necessary
Scope	Corporate I System I Subsystem
Level	Summary I Primary task I Subfunction

Use Case Component	Description	
Primary Actor	Role name or description of the primary actor	
Preconditions	The required state of the system before the use case is triggered	
Success End Conditions	The state of the system upon successful completion of this use case	
Failed End Conditions	The state of the system if the use case cannot execute to completion	
Trigger	The action that initiates the execution of the use case	
Main Success Scenario	Step	Action
	1	
	2	
	...	
Extensions	Conditions under which the main success scenario will vary and a description of those variations	
Sub-Variations	Variations that do not affect the main flow but that must be considered	
Priority	Criticality	
Response Time	Time available to execute this use case	
Frequency	How often this use case is executed	
Channels to Primary Actor	Interactive I File I Database I ...	
Secondary Actors	Other actors needed to accomplish this use case	
Channels to Secondary Actors	Interactive I File I Database I ...	
Date Due	Schedule information	
Completeness Level	Use Case identified (0.1)I Main scenario defined (0.5) I All extensions defined (0.8) I All fields complete (1.0)	
Open Issues	Unresolved issues awaiting decisions	

Thí dụ bảng đặc tả usecase "đăng ký môn học" trong phần mềm quản lý học vụ có nội dung chi tiết như sau :

Use Case Component	Description	
Use Case Number or Identifier	SURS1138	
Use Case Name	Register for a course (a class taught by a faculty member)	
Goal in Context		
Scope	System	
Level	Primary task	
Primary Actor	Student	
Preconditions	None	
Success End Conditions	The student is registered for the course—the course has been added to the student's course list	
Failed End Conditions	The student's course list is unchanged	
Trigger	Student selects a course and "Registers"	
Main Success Scenario A: Actor S: System	Step	Action
	1	A: Selects "Register for a course"
	2	A: Selects course (e.g. Math 1060)
	3	S: Displays course description
	4	A: Selects section (Mon & Wed 9:00am)
	5	S: Displays section days and times
	6	A: Accepts
	7	S: Adds course/section to student's course list
Extensions	2a	Course does not exist S: Display message and exit
	4a	Section does not exist S: Display message and exit
	4b	Section is full S: Display message and exit
	6a	Student does not accept S: Display message and exit

Use Case Component	Description
Sub-Variations	Student may use • Web • Phone
Priority	Critical
Response Time	10 seconds or less
Frequency	Approximately 5 courses x 10,000 students over a 4-week period
Channels to Primary Actor	Interactive
Secondary Actors	None
Channels to Secondary Actors	N/A
Date Due	1 Feb
Completeness Level	0.5
Open Issues	None

Dựa vào đặc tả về kịch bản chính và về các nới rộng của kịch bản, ta sẽ thiết kế các testcase theo ý tưởng như sau :

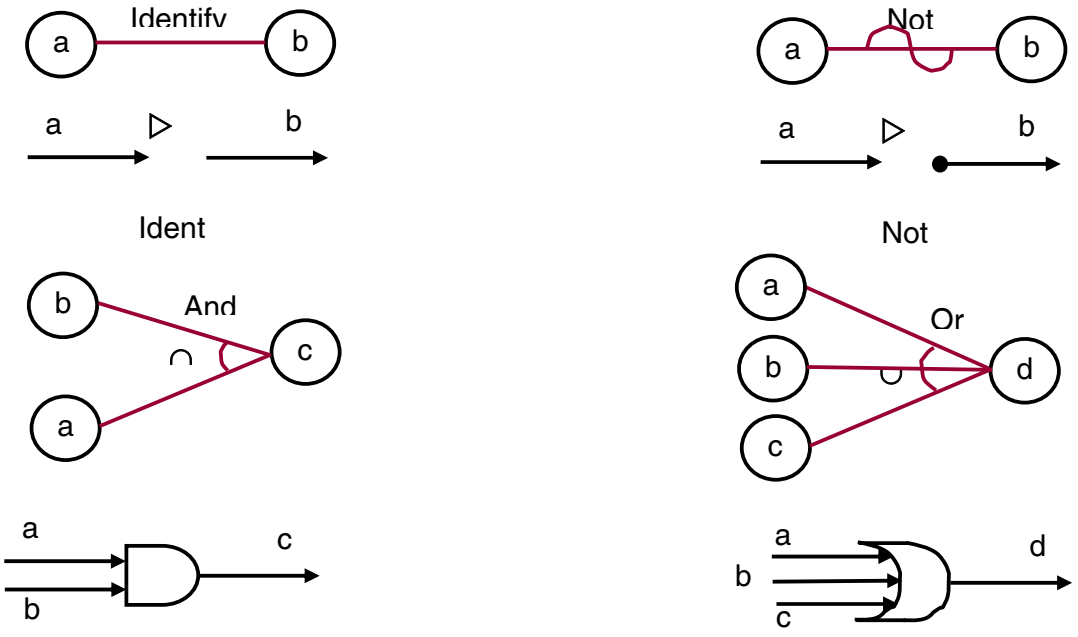
- Ít nhất 1 testcase để kiểm thử kịch bản chính.
- Ít nhất 1 testcase cho từng nới rộng có thể có.
- Nếu kịch bản chính hay 1 nới rộng nào đó bị loop thì không cần thiết phải kiểm thử phần loop lại.

6.4 Kỹ thuật dùng đồ thị nhân quả (Cause-Effect Diagram)

Đồ thị nhân quả là 1 dạng khác của mạng luận lý tổ hợp mà phần cứng thường dùng. Các phần tử cấu thành đồ thị nhân quả là :

- các nút : mỗi nút miêu tả 1 hậu quả (1 hay nhiều hoạt động + 1 hay nhiều kết quả).
- các đoạn thẳng : mỗi đoạn thẳng miêu tả 1 nguyên nhân (1 điều kiện dữ liệu nhập ở dạng nhị phân)

- các ký hiệu : mỗi ký hiệu miêu tả 1 phép toán luận lý.
- các phần tử ràng buộc, mỗi phần tử miêu tả 1 ràng buộc xác định nào đó.



Giả sử đặc tả 1 TPPM như sau : dữ liệu đầu vào là tên file gồm 2 ký tự, ký tự đầu là A hay B, ký tự còn lại là ký số, TPPM sẽ cập nhật file, nếu ký tự đầu không phải là A hay B thì TPPM báo lỗi X1, nếu ký tự thứ 2 không phải là số thì báo lỗi X2.

Duyệt đọc đặc tả và phân tích đặc tả, ta tìm được các điều kiện đầu vào là :

1 : Ký tự đầu là A.

2 : Ký tự đầu là B.

3 : Ký tự thứ hai là ký số.

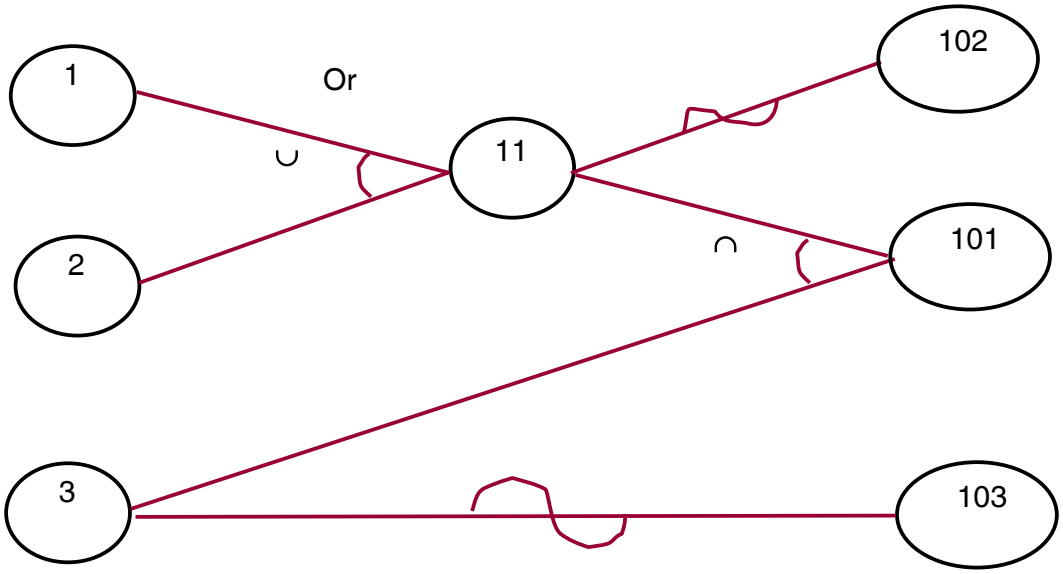
Và các hậu quả ở đầu ra là :

101 : cập nhật file.

102 : báo lỗi X1.

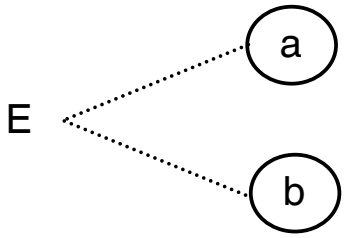
103 : báo lỗi X2.

Đồ thị nhân quả của TPPM ở silde trước là :

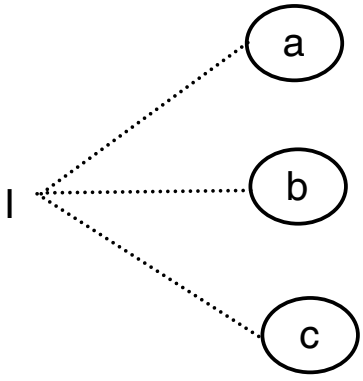


Trong nhiều trường hợp, tồn tại tổ hợp điều kiện nhập không thể xảy ra, thí dụ ở slide trước, điều kiện 1 và 2 không thể xảy ra đồng thời vì ký tự đầu không thể vừa là A vừa là B. Để miêu tả các ràng buộc này, ta dùng các ký hiệu ràng buộc sau :

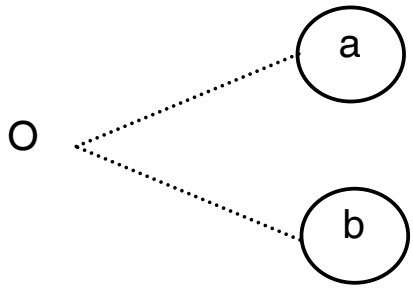
1. E : không thể đồng thời xảy ra.



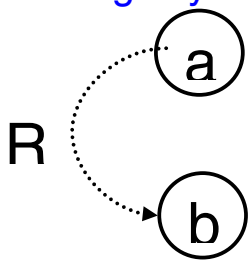
2. I : phải ít nhất 1 điều kiện xảy ra.



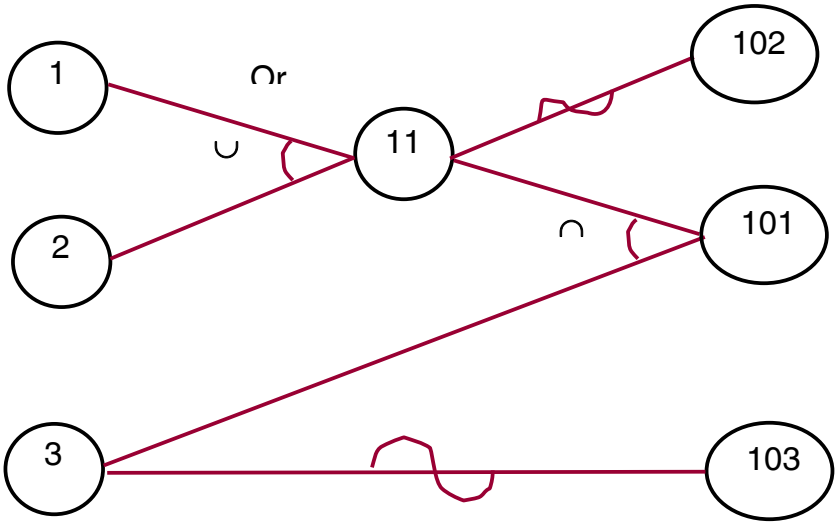
3. O : 1 và chỉ 1 điều kiện xảy ra.



4. R : nếu a xảy ra thì b cũng xảy ra.



Đồ thị nhân quả của TPPM ở slide 34 được hoàn chỉnh là :



Qui trình định nghĩa các testcase dùng kỹ thuật đồ thị nhân quả gồm các bước :

1. Đặc tả của TPPM được chia nhỏ ra nhiều phần nhỏ để có thể làm việc dễ dàng (nếu không thì đồ thị nhân quả sẽ rất phức tạp).
2. Nhận dạng các nguyên nhân và hậu quả của phần nhỏ đang xử lý.

3. Tìm mối quan hệ giữa các nguyên nhân và hậu quả, mỗi mối quan hệ được vẽ thành 1 đường nối.
4. Xác định các ràng buộc giữa các nguyên nhân và chú thích chúng vào đồ thị.
5. Chuyển đồ thị nhân quả về bảng quyết định.
6. Chuyển bảng quyết định thành bảng các testcase.

6.5 Kết chương

Chương này đã tiếp tục giới thiệu chi tiết cụ thể về 4 kỹ thuật kiểm thử hộp đen được dùng phổ biến khác là kỹ thuật dùng lược đồ chuyển trạng thái, kỹ thuật phân tích vùng, kỹ thuật dùng thông tin trong use-case, và kỹ thuật dùng đồ thị nhân quả.

Ứng với mỗi kỹ thuật kiểm thử, chúng ta cũng đã giới thiệu 1 thí dụ cụ thể để demo qui trình thực hiện kỹ thuật kiểm thử tương ứng.

Chương 7

Thanh tra, chạy thử & xem xét mã nguồn

7.1 Giới thiệu

Trong các chương 3, 4, 5, 6 chúng ta đã giới thiệu nhiều kỹ thuật kiểm thử hộp đen lẫn hộp trắng. Điểm chung của các kỹ thuật này là phải chạy thật phần mềm trên máy tính với môi trường phù hợp để tìm lỗi của phần mềm.

Nhưng trong những thế hệ đầu tiên của máy tính, máy tính còn rất yếu và rất đắt, người lập trình chưa có cơ hội làm việc trực tiếp trên máy tính, họ chỉ viết chương trình trên giấy và đem chồng giấy miêu tả chương trình và dữ liệu cần xử lý đến trung tâm máy tính để đăng ký chạy. Khi nhận được kết quả, nếu thấy không vừa ý, họ sẽ phải tự mình đọc và xem xét mã nguồn trên giấy để tìm lỗi và sửa lỗi.

cuu duong than cong . com

Hiện nay, không phải người kiểm thử nào cũng đọc mã nguồn, nhưng ý tưởng nghiên cứu mã nguồn vẫn được chấp nhận rộng rãi như là 1 nỗ lực kiểm thử hữu hiệu vì những lý do sau :

- kích thước và độ phức tạp về thuật giải của chương trình.
- kích thước của đội phát triển phần mềm.
- thời gian qui định cho việc phát triển phần mềm.
- nền tảng và văn hóa của đội ngũ lập trình.

cuu duong than cong . com

Qui trình kiểm thử thủ công (chỉ dùng sức người, không dùng máy tính) được gọi là kiểm thử tĩnh, qui trình này có 1 số tính chất chính :

- Rất hữu hiệu trong việc tìm lỗi nên mỗi project phần mềm nên dùng 1 hay nhiều kỹ thuật này trong việc kiểm thử phần mềm.

- Dùng các kỹ thuật kiểm thử tĩnh trong khoảng thời gian từ lúc phần mềm được viết đến khi phần mềm có thể được kiểm thử bằng máy tính.
- Không có nhiều kết quả toán học đánh giá về các kỹ thuật kiểm thử tĩnh vì chúng chỉ liên quan đến con người.

Kiểm thử thủ công TPPM cũng đã đóng góp 1 phần cho tính tin cậy, công nghiệp của hoạt động kiểm thử thành công TPPM :

- Các lỗi được phát hiện càng sớm càng giúp giảm chi phí sữa lỗi và càng giúp nâng cao xác xuất sữa lỗi đúng đắn.
- Lập trình viên dễ dàng chuẩn bị tinh thần khi các kỹ thuật kiểm thử bằng máy tính bắt đầu.

Có nhiều phương pháp kiểm thử thủ công TPPM, trong đó 2 phương pháp quan trọng nhất là thanh kiểm tra mã nguồn (Code Inspections) và chạy thủ công mã nguồn (Walkthroughs). Hai phương pháp này có nhiều điểm giống nhau :

- Cần 1 nhóm người đọc hay thanh kiểm tra trực quan TPPM.
- Các thành viên phải có chuẩn bị trước, không khí cuộc họp phải là "họp các ý kiến thẳng thắn, chân thành".
- Mục tiêu của cuộc họp là tìm các lỗi chứ không phải tìm biện pháp giải quyết lỗi.
- Chúng là sự cải tiến, tăng cường của 1 phương pháp kiểm thử cũ hơn là phương pháp "desk-checking" mà chúng ta sẽ giới thiệu sau.

Hai phương pháp thanh kiểm tra mả nguồn và chạy thủ công mã nguốn có thể phát hiện được từ 30 tới 70% các lỗi viết mã nguồn và lỗi thiết kế luận lý của TPPM bình thường.

Tuy nhiên 2 phương pháp này không hiệu quả trong việc phát hiện các lỗi thiết kế cấp cao như lỗi do hoạt động phân tích yêu cầu TPPM :

cuu duong than cong . com

- Các hoạt động con người thường chỉ tìm được các lỗi dễ.
- Do đó 2 phương pháp kiểm thử tĩnh này chỉ bổ trợ thêm cho các kỹ thuật kiểm thử chính qui bằng máy tính.

7.2 Phương pháp thanh kiểm tra mã nguồn

Bao gồm các thủ tục và các kỹ thuật phát hiện lỗi nhờ 1 nhóm người cùng đọc mã nguồn. Các vấn đề bàn cải liên quan đến phương pháp thanh kiểm tra là các thủ tục vận hành, các form kết quả cần tạo ra.

- Một nhóm thanh kiểm tra mã nguồn thường gồm 4 người :
- người điều hành (chủ tịch hội đồng), thường là kỹ sư QC
- lập trình viên TPPM cần kiểm thử.
- Kỹ sư thiết kế TPPM (nếu không phải là lập trình viên TPPM này)
- Chuyên gia kiểm thử

cuu duong than cong . com

1. Người điều hành :

- nên là người lập trình kinh nghiệm, uy tín.
- không nên là tác giả TPPM cần kiểm thử và không cần biết chi tiết về TPPM cần kiểm thử.

2. Các nhiệm vụ :

- Phân phối các tài liệu cho các thành viên khác trước khi cuộc họp diễn ra. Lập lịch cho buổi họp thanh kiểm tra.

cuu duong than cong . com

- Điều khiển cuộc họp thanh kiểm tra.
- Ghi nhận các lỗi phát hiện được bởi các thành viên.

Công việc chuẩn bị :

- Thời gian và địa điểm buổi họp : làm sao tránh được các ngắt quảng từ ngoài.
- Thời lượng tối ưu cho mỗi buổi họp là từ 90 tới 120 phút.

- Mỗi thành viên cần chuẩn bị thái độ khách quan, nhẹ nhàng trong buổi họp.

Các tài liệu cần có cho mỗi thành viên (đã phân phát trước khi cuộc họp diễn ra) :

- Mã gnuồn TPPM cần kiểm thử.
- Danh sách các lỗi quá khứ thường gặp (checklist).

Trong suốt cuộc họp, 2 hoạt động chính sẽ xảy ra :

1. Hoạt động 1 :

- Người lập trình sẽ giới thiệu tuần tự từng hàng lệnh cùng luận lý của TPPM cho các thành viên khác nghe.
- Trong khi thảo luận, các thanh viên khác đưa ra các câu hỏi và theo dõi phần trả lời để xác định có lỗi ở hàng lệnh nào không ? (Tốt nhất là người lập trình, chứ không phải thành viên khác) sẽ phát hiện được nhiều lỗi trong phần giới thiệu mã nguồn này).

2. Hoạt động 2 :

- Các thành viên cùng phân tích TPPM dựa trên danh sách các lỗi lập trình thường gặp trong quá khứ.

Sau cuộc họp thanh kiểm tra mã nguồn :

- Người điều hành sẽ giao cho người lập trình TPPM 1 danh sách chứa các lỗi mà nhóm đã tìm được.
- Nếu số lỗi là nhiều hay nếu lỗi phát hiện đòi hỏi sự hiệu chỉnh lớn, người điều hành sẽ sắp xếp 1 buổi thanh kiểm tra khác sau khi TPPM đã được sửa lỗi.

Chú ý : Các lỗi phát hiện được cũng sẽ được phân tích, phân loại và được dùng để tinh chỉnh lại danh sách các lỗi quá khứ để cải tiến độ hiệu quả cho các lần thanh kiểm tra sau này.

Các hiệu ứng lề tích cực cho từng thành viên :

cuu duong than cong . com

- Người lập trình thường nhận được các style lập trình tốt mà mình chưa biết, cách thức chọn lựa giải thuật tốt để giải quyết bài toán, các kỹ thuật lập trình tốt,..
- Các thành viên khác cũng vậy.
- Quá trình thanh kiểm tra mã nguồn là 1 cách nhận dạng sớm nhất các vùng code chứa nhiều lỗi, giúp ta tập trung sự chú ý hơn vào các vùng code này trong suốt quá trình kiểm thử dựa trên mày tính sau này.

7.3 Checklist được dùng để thanh tra mã nguồn

Checklist liệt kê các lỗi mà người lập trình thường phạm phải. Đây là kết quả của 1 lịch sử thanh tra mã nguồn bởi nhiều người, và ta có thể bỏ bớt/thêm mới các lỗi nếu thấy cần thiết. Các lỗi mà người lập trình thường phạm phải được phân loại thành các nhóm chính :

1. Các lỗi truy xuất dữ liệu (Data Reference Errors)
2. Các lỗi định nghĩa/khai báo dữ liệu (Data-Declaration Errors)
3. Các lỗi tính toán (Computation Errors)
4. Các lỗi so sánh (Comparison Errors)
5. Các lỗi luồng điều khiển (Control-Flow Errors)
6. Các lỗi giao tiếp (Interface Errors)
7. Các lỗi nhập/xuất (Input/Output Errors)
8. Các lỗi khác (Other Checks)

Các lỗi truy xuất dữ liệu (Data Reference Errors)

1. Dùng biến chưa có giá trị xác định ?
   ```
   int i, count;
   for (i = 0; i < count; i++) {...}
   ```
2. Dùng chỉ số của biến array nằm ngoài phạm vi ?

cuu duong than cong . com

```
int list[10];
if (list[10] == 0) {...}
```

3. Dùng chỉ số không thuộc kiểu nguyên của biến array ?

```
int list[10];
double idx=3.1416;
if (list[idx] == 0) {...}
```

4. Tham khảo đến dữ liệu không tồn tại (dangling references)?

```
int *pi;
if (*pi == 10) {...}   //pi đang tham khảo đến địa chỉ không hợp lệ - Null
int *pi = new int;
...
delete (pi);
if (*pi = 10) {...}    //pi đang tham khảo đến địa chỉ
                       //mà không còn dùng để chứa số nguyên
```

5. Truy xuất dữ liệu thông qua alias có đảm bảo thuộc tính dữ liệu đúng ?

```
int pi[10];
pi[1] = 25;
char* pc = pi;
if (pc[1] == 25) {...}          //pc[1] khác với pi[1];
```

6. Thuộc tính của field dữ liệu trong record có đúng với nội dung gốc không ?

```
struct {int i; double d;} T_Rec;
T_Rec rec;
read(fdin,&rec, sizeof(T_Rec);
if (rec.i ==10) {...}           //lỗi nếu field d nằm trước i
                                  //trong record gốc trên file
```

7. Cấu trúc kiểu record có tương thích giữa client/server không ?

```
Private Type OSVERSIONINFO
        dwOSVersionInfoSize As Long
```

cuu duong than cong . com

```
        dwMajorVersion As Long
        dwMinorVersion As Long
        dwBuildNumber As Long
        dwPlatformId As Long
        szCSDVersion As String * 128 ' Maintenance string
    End Type
    Private Declare Function GetVersionEx Lib "kernel32" _
    Alias "GetVersionExA" (lpVersionInformation As
OSVERSIONINFO) As Long
```

8. Dùng chỉ số bị lệch ?

```
int i, pi[10];
for (i = 1; i <= 10; i++) pi [i] = i;
```

9. Class có hiện thực đủ các tác vụ trong interface mà nó hiện thực không ?

10. Class có hiện thực đủ các tác vụ "pure virtual" của class cha mà nó thừa kế không ?

cuu duong than cong . com

Các lỗi khai báo dữ liệu

1. Tất cả các biến đều được định nghĩa hay khai báo tường minh chưa?

```
int i;
extern double d;
d = i*10;
```

2. Định nghĩa hay khai báo đầy đủ các thuộc tính của biến dữ liệu chưa?

cuu duong than cong . com

```
static int i = 10;
```

3. Biến array hay biến chuỗi được khởi động đúng chưa ?

```
int pi[10] = {1, 5,7,9} ;
```

4. Kiểu và độ dài từng biến đã được định nghĩa đúng theo yêu cầu chưa ?

```
short IPAddress;
byte Port;
```

5. Giá trị khởi động có tương thích với kiểu biến ?
   short IPAddress = inet_addr("203.7.85.98");
   byte Port = 65535;
6. Có dùng các biến ý nghĩa khác nhau nhưng tên rất giống nhau không?
   int count, counts;

Các lỗi tính toán (Computation Errors)

1. Thực hiện phép toán trên toán hạng không phải là số?
   CString s1, s2;
   int ketqua = s1/s2;
2. Thực hiện phép toán trên các toán hạng có kiểu không tương thích ?
   byte b;
   int i;
   double d;
   b = i * d;
3. Thực hiện phép toán trên các toán hạng có độ dài khác nhau ?
   byte b;
   int i;
   b = i * 500;
4. Gán dữ liệu vào biến có độ dài nhỏ hơn ?
   byte b;
   int i;
   b = i * 500;
5. Kết quả trung gian bị tràn?
   byte i, j, k;
   i = 100; j = 4;
   k = i * j / 5;
6. Phép chia có mẫu bằng 0 ?
   byte i, k;

cuu duong than cong . com
cuu duong than cong . com

```
i = 100 / k;
```

7. Mất độ chính xác khi mã hóa/giải mã số thập phân/số nhị phân ?
8. Giá trị biến nằm ngoài phạm vi ngữ nghĩa ?
```
int tuoi = 3450;
tuoi = -80;
```
9. Thứ tự thực hiện các phép toán trong biểu thức mà người lập trình mong muốn có tương thích với thứ tự mà máy thực hiện? Người lập trình hiểu đúng về thứ tự ưu tiên các phép toán chưa ?
```
double x1 = (-b-sqrt(delta)) / 2*a;
```
10. Kết quả phép chia nguyên có chính xác theo yêu cầu không ?
```
int i = 3;
if (i/2*2) == i) {...}
```

cuu duong than cong . com

Các lỗi so sánh (Comparison Errors)

1. So sánh 2 dữ liệu có kiểu không tương thích ?
```
int ival;
char sval[20];
if (ival == sval) {...}
```
2. So sánh 2 dữ liệu có kiểu không cùng độ dài ?
```
int ival;
char cval;
if (ival == cval) {...}
```

cuu duong than cong . com

3. Toán tử so sánh đúng ngữ nghĩa mong muốn? Dễ lộn giữa = và ≠, <= và >=, and và or...
4. Có nhầm lẫn giữa biểu thức Bool và biểu thức so sánh ?
```
if (2 < i < 10) {...}
if (2 < i  &&  i < 10) {...}
```
5. Có hiểu rõ thứ tự ưu tiên các phép toán ?

if(a==2 && b==2 || c==3) {...}

6. Cách thức tính biểu thức Bool của chương trình dịch như thế nào ?
   if(y==0 || (x/y > z))

Các lỗi luồng điều khiển (Control-Flow Errors)

1. Thiếu thực hiện 1 số nhánh trong lệnh quyết định theo điều kiện số học ?
   switch (i) {
   case 1: ... //cần hay không cần lệnh break;
   case 2: ...
   case 3: ...
   }
2. Mỗi vòng lặp thực hiện ít nhất 1 lần hay sẽ kết thúc ?
   for (i=x ; i<=z; i++) {...} //nếu x > z ngay từ đầu thì sao ?
   for (i = 1; i <= 10; i--) {...} //có dừng được không ?
3. Biên của vòng lặp có bị lệch ?
   for (i = 0; i <= 10; i++) {...} //hay i < 10 ?
4. Có đủ và đúng cặp token begin/end, {} ?

Các lỗi giao tiếp (Interface Errors)

1. Số lượng tham số cụ thể được truyền có = số tham số hình thức của hàm được gọi ?
2. thứ tự các tham số có đúng không ?
3. thuộc tính của từng tham số thực có tương thích với thuộc tính của tham số hình thức tương ứng của hàm được gọi ?
   char* str = "Nguyen Van A";
   MessageBox (hWnd, str,"Error", MB_OK); //sẽ bị lỗi khi dịch ở chế độ Unicode
4. Đơn vị đo lường của tham số thực giống với tham số hình thức ?
   double d = cos (90);

cuu duong than cong . com

5. Tham số read-only có bị thay đổi nội dung bởi hàm không ?
6. Định nghĩa biến toàn cục có tương thích giữa các module chức năng không ?

Các lỗi nhập/xuất (Input/Output Errors)

1. Lệnh mở/tạo file có đúng chế độ và định dạng truy xuất file?

```
if ((fdout = open ("tmp0", O_WRONLY| O_CREAT| O_BINARY, S_IREAD| S_IWRITE)) < 0)
        pr_error_exit("Khong the mo file tmp0 de ghi");
if ((fdtmp = open ("tmp2", O_RDWR | O_CREAT | O_BINARY, S_IREAD | S_IWRITE)) < 0)
```

2. Kích thước của buffer có đủ chứa dữ liệu đọc vào không ?

```
char buffin[100];
sl = read(fd, bufin, MAXBIN); //MAXBIN <= 100 ?
```

3. Có mở file trước khi truy xuất không ?
4. Có đóng file lại sau khi dùng không ? Có xử lý điều kiện hết file ?
5. Có xử lý lỗi khi truy xuất file không ?
6. Chuỗi xuất có bị lỗi từ vựng và cú pháp không ?

Các lỗi khác (Other Checks)

1. Có biến nào không được tham khảo trong danh sách tham khảo chéo (cross-reference)?
2. Cái gì được kỳ vọng trong danh sách thuộc tính ?
3. Có các cảnh báo hay thông báo thông tin ?
4. Có kiểm tra tính xác thực của dữ liệu nhập chưa ?
5. Có thiếu hàm chức năng ?

cuu duong than cong . com

7.4 Phương pháp chạy thủ công mã nguồn

Giống như phương pháp thanh kiểm tra mã nguồn, phương pháp này bao gồm 1 tập các thủ tục và kỹ thuật phát hiện lỗi dành cho 1 nhóm người đọc mã nguồn.

Cần 1 cuộc họp dài từ 1 tới 4 giờ và không được ngắt quảng giữa chừng.

Nhóm chạy thủ công gồm 3 tới 5 người :

- Lập trình viên nhiều kinh nghiệm
- Chuyên gia về ngôn ngữ lập trình được dùng để viết mã nguồn
- Lập trình viên mới
- 1 người mà sẽ bảo trì phần mềm
- 1 người từ project khác, 1 người cùng nhóm với lập trình viên mã nguồn cần kiểm thử.

cuu duong than cong . com

Các vai trò trong nhóm :

- Chủ tịch điều hành
- Thư ký (người ghi lại các lỗi phát hiện được)
- Người kiểm thử

Thủ tục ban đầu cũng giống như thủ tục ban đầu của phương pháp thanh kiểm tra mã nguồn.

Thủ tục trong cuộc họp :

cuu duong than cong . com

- Thay vì chỉ đọc chương trình hay dùng danh sách lỗi thường gặp, các thành viên phải biến mình làm CPU để chạy thủ công mã nguồn.
- Người kiểm thử được cung cấp 1 tập các testcase.
- Trong cuộc họp, người kiểm thử sẽ thực thi từng testcase thủ công. Trạng thái chương trình (nội dung các biến) sẽ được ghi và theo dõi trên giấy hay trên bảng.

Lưu ý :

- Các testcase thường ở mức đơn giản để phục vụ như là phương tiện để bắt đầu và gợi câu hỏi người lập trình về logic thuật giải cũng như những giả định của anh ta.
- Thái độ của từng người tham dự là quan trọng.
- Các chú thích nên hướng đến chương trình thay vì đến người lập trình.
- Chạy thử thủ công nên có qui trình theo sau tương tự như đã diễn tả cho phương pháp thanh kiểm tra.
- Phương pháp chạy thủ công mã nguồn cũng tạo được các hệu ứng lề y như phương pháp thanh kiểm tra.

7.5 Phương pháp kiểm tra thủ công (Desk-checking)

Phương pháp kiểm tra thủ công có thể được xem như là phương pháp thanh kiểm tra hay phương pháp chạy thủ công mà chỉ có 1 người tham gia thực hiện : người này sẽ tự đọc mã nguồn, tự kiểm tra theo danh sách lỗi thường gặp hay chạy thủ công và theo dõi nội dung các biến dữ liệu.

Đối với hầu hết mọi người, phương pháp này không được công nghiệp cho lắm :

- Đây là 1 qui trình hoàn toàn không "undisciplined".
- Ta thường không thể kiểm thử hiệu quả chương trình do mình viết vì chủ quan, thiên vị....⇒ nên được thực hiện bởi người khác, chứ không phải là tác giả của phần mềm.

Kém hiệu quả hơn nhiều so với 2 phương pháp trước. Hiệu ứng synergistic của 2 phương pháp trước.

7.6 Phương pháp so sánh với phần mềm tương tự (Peer Ratings)

Phương pháp này không kiểm thử trực tiếp phần mềm, mục tiêu của nó không phải để tìm lỗi của phần mềm.

Đây là kỹ thuật đánh giá, so sánh các tính chất của các chương trình tương tự như chất lượng tổng thể, độ bảo trì, độ mở rộng, độ sự dụng, độ trong sáng...

Mục đích của phương pháp này là cung cấp cho người lập trình 1 sự tự đánh giá.

7.7 Kết chương

Chương này đã trình bày lý do vì sao chúng ta cần kiểm thử TPPM một cách tĩnh, nghĩa là không cần dùng máy tính chạy trực tiếp TPPM mà chỉ khảo sát xem xét TPPM thủ công thông qua mắt người.

Chúng ta đã trình bày các phương pháp kiểm thử tĩnh TPPM như Desk Checking, thanh kiểm tra, chạy thủ công, peer rating. Ứng với mỗi phương pháp, chúng ta đã trình bày các tính chất cơ bản của phương pháp đó, nguồn nhân lực cần thiết và qui trình thực hiện kiểm thử.

cuu duong than cong . com

cuu duong than cong . com

Chương 7

Thanh tra, chạy thử & xem xét mã nguồn

7.1 Giới thiệu

Trong các chương 3, 4, 5, 6 chúng ta đã giới thiệu nhiều kỹ thuật kiểm thử hộp đen lẫn hộp trắng. Điểm chung của các kỹ thuật này là phải chạy thật phần mềm trên máy tính với môi trường phù hợp để tìm lỗi của phần mềm.

Nhưng trong những thế hệ đầu tiên của máy tính, máy tính còn rất yếu và rất đắt, người lập trình chưa có cơ hội làm việc trực tiếp trên máy tính, họ chỉ viết chương trình trên giấy và đem chồng giấy miêu tả chương trình và dữ liệu cần xử lý đến trung tâm máy tính để đăng ký chạy. Khi nhận được kết quả, nếu thấy không vừa ý, họ sẽ phải tự mình đọc và xem xét mã nguồn trên giấy để tìm lỗi và sửa lỗi.

Hiện nay, không phải người kiểm thử nào cũng đọc mã nguồn, nhưng ý tưởng nghiên cứu mã nguồn vẫn được chấp nhận rộng rãi như là 1 nỗ lực kiểm thử hữu hiệu vì những lý do sau :

- kích thước và độ phức tạp về thuật giải của chương trình.
- kích thước của đội phát triển phần mềm.
- thời gian qui định cho việc phát triển phần mềm.
- nền tảng và văn hóa của đội ngũ lập trình.

Qui trình kiểm thử thủ công (chỉ dùng sức người, không dùng máy tính) được gọi là kiểm thử tĩnh, qui trình này có 1 số tính chất chính :

- Rất hữu hiệu trong việc tìm lỗi nên mỗi project phần mềm nên dùng 1 hay nhiều kỹ thuật này trong việc kiểm thử phần mềm.

- Dùng các kỹ thuật kiểm thử tĩnh trong khoảng thời gian từ lúc phần mềm được viết đến khi phần mềm có thể được kiểm thử bằng máy tính.
- Không có nhiều kết quả toán học đánh giá về các kỹ thuật kiểm thử tĩnh vì chúng chỉ liên quan đến con người.

Kiểm thử thủ công TPPM cũng đã đóng góp 1 phần cho tính tin cậy, công nghiệp của hoạt động kiểm thử thành công TPPM :

- Các lỗi được phát hiện càng sớm càng giúp giảm chi phí sữa lỗi và càng giúp nâng cao xác xuất sữa lỗi đúng đắn.
- Lập trình viên dễ dàng chuẩn bị tinh thần khi các kỹ thuật kiểm thử bằng máy tính bắt đầu.

Có nhiều phương pháp kiểm thử thủ công TPPM, trong đó 2 phương pháp quan trọng nhất là thanh kiểm tra mã nguồn (Code Inspections) và chạy thủ công mã nguồn (Walkthroughs). Hai phương pháp này có nhiều điểm giống nhau :

- Cần 1 nhóm người đọc hay thanh kiểm tra trực quan TPPM.
- Các thành viên phải có chuẩn bị trước, không khí cuộc họp phải là "họp các ý kiến thẳng thắn, chân thành".
- Mục tiêu của cuộc họp là tìm các lỗi chứ không phải tìm biện pháp giải quyết lỗi.
- Chúng là sự cải tiến, tăng cường của 1 phương pháp kiểm thử cũ hơn là phương pháp "desk-checking" mà chúng ta sẽ giới thiệu sau.

Hai phương pháp thanh kiểm tra mả nguồn và chạy thủ công mã nguốn có thể phát hiện được từ 30 tới 70% các lỗi viết mã nguồn và lỗi thiết kế luận lý của TPPM bình thường.

Tuy nhiên 2 phương pháp này không hiệu quả trong việc phát hiện các lỗi thiết kế cấp cao như lỗi do hoạt động phân tích yêu cầu TPPM :

- Các hoạt động con người thường chỉ tìm được các lỗi dễ.
- Do đó 2 phương pháp kiểm thử tĩnh này chỉ bổ trợ thêm cho các kỹ thuật kiểm thử chính qui bằng máy tính.

7.2 Phương pháp thanh kiểm tra mã nguồn

Bao gồm các thủ tục và các kỹ thuật phát hiện lỗi nhờ 1 nhóm người cùng đọc mã nguồn. Các vấn đề bàn cải liên quan đến phương pháp thanh kiểm tra là các thủ tục vận hành, các form kết quả cần tạo ra.

- Một nhóm thanh kiểm tra mã nguồn thường gồm 4 người :
- người điều hành (chủ tịch hội đồng), thường là kỹ sư QC
- lập trình viên TPPM cần kiểm thử.
- Kỹ sư thiết kế TPPM (nếu không phải là lập trình viên TPPM này)
- Chuyên gia kiểm thử

1. Người điều hành :

- nên là người lập trình kinh nghiệm, uy tín.
- không nên là tác giả TPPM cần kiểm thử và không cần biết chi tiết về TPPM cần kiểm thử.

2. Các nhiệm vụ :

- Phân phối các tài liệu cho các thành viên khác trước khi cuộc họp diễn ra. Lập lịch cho buổi họp thanh kiểm tra.
- Điều khiển cuộc họp thanh kiểm tra.
- Ghi nhận các lỗi phát hiện được bởi các thành viên.

Công việc chuẩn bị :

- Thời gian và địa điểm buổi họp : làm sao tránh được các ngắt quảng từ ngoài.
- Thời lượng tối ưu cho mỗi buổi họp là từ 90 tới 120 phút.

- Mỗi thành viên cần chuẩn bị thái độ khách quan, nhẹ nhàng trong buổi họp.

Các tài liệu cần có cho mỗi thành viên (đã phân phát trước khi cuộc họp diễn ra) :

- Mã gnuồn TPPM cần kiểm thử.
- Danh sách các lỗi quá khứ thường gặp (checklist).

Trong suốt cuộc họp, 2 hoạt động chính sẽ xảy ra :

1. Hoạt động 1 :

- Người lập trình sẽ giới thiệu tuần tự từng hàng lệnh cùng luận lý của TPPM cho các thành viên khác nghe.
- Trong khi thảo luận, các thanh viên khác đưa ra các câu hỏi và theo dõi phần trả lời để xác định có lỗi ở hàng lệnh nào không ? (Tốt nhất là người lập trình, chứ không phải thành viên khác) sẽ phát hiện được nhiều lỗi trong phần giới thiệu mã nguồn này).

2. Hoạt động 2 :

- Các thành viên cùng phân tích TPPM dựa trên danh sách các lỗi lập trình thường gặp trong quá khứ.

Sau cuộc họp thanh kiểm tra mã nguồn :

- Người điều hành sẽ giao cho người lập trình TPPM 1 danh sách chứa các lỗi mà nhóm đã tìm được.
- Nếu số lỗi là nhiều hay nếu lỗi phát hiện đòi hỏi sự hiệu chỉnh lớn, người điều hành sẽ sắp xếp 1 buổi thanh kiểm tra khác sau khi TPPM đã được sửa lỗi.

Chú ý : Các lỗi phát hiện được cũng sẽ được phân tích, phân loại và được dùng để tinh chỉnh lại danh sách các lỗi quá khứ để cải tiến độ hiệu quả cho các lần thanh kiểm tra sau này.

Các hiệu ứng lề tích cực cho từng thành viên :

- Người lập trình thường nhận được các style lập trình tốt mà mình chưa biết, cách thức chọn lựa giải thuật tốt để giải quyết bài toán, các kỹ thuật lập trình tốt,..
- Các thành viên khác cũng vậy.
- Quá trình thanh kiểm tra mã nguồn là 1 cách nhận dạng sớm nhất các vùng code chứa nhiều lỗi, giúp ta tập trung sự chú ý hơn vào các vùng code này trong suốt quá trình kiểm thử dựa trên mày tính sau này.

7.3 Checklist được dùng để thanh tra mã nguồn

Checklist liệt kê các lỗi mà người lập trình thường phạm phải. Đây là kết quả của 1 lịch sử thanh tra mã nguồn bởi nhiều người, và ta có thể bỏ bớt/thêm mới các lỗi nếu thấy cần thiết. Các lỗi mà người lập trình thường phạm phải được phân loại thành các nhóm chính :

1. Các lỗi truy xuất dữ liệu (Data Reference Errors)
2. Các lỗi định nghĩa/khai báo dữ liệu (Data-Declaration Errors)
3. Các lỗi tính toán (Computation Errors)
4. Các lỗi so sánh (Comparison Errors)
5. Các lỗi luồng điều khiển (Control-Flow Errors)
6. Các lỗi giao tiếp (Interface Errors)
7. Các lỗi nhập/xuất (Input/Output Errors)
8. Các lỗi khác (Other Checks)

Các lỗi truy xuất dữ liệu (Data Reference Errors)

1. Dùng biến chưa có giá trị xác định ?
   ```
   int i, count;
   for (i = 0; i < count; i++) {...}
   ```
2. Dùng chỉ số của biến array nằm ngoài phạm vi ?

```
int list[10];
if (list[10] == 0) {...}
```

3. Dùng chỉ số không thuộc kiểu nguyên của biến array ?

```
int list[10];
double idx=3.1416;
if (list[idx] == 0) {...}
```

4. Tham khảo đến dữ liệu không tồn tại (dangling references)?

```
int *pi;
if (*pi == 10) {...}   //pi đang tham khảo đến địa chỉ không hợp lệ - Null
int *pi = new int;
...
delete (pi);
if (*pi = 10) {...}     //pi đang tham khảo đến địa chỉ
                        //mà không còn dùng để chứa số nguyên
```

5. Truy xuất dữ liệu thông qua alias có đảm bảo thuộc tính dữ liệu đúng ?

```
int pi[10];
pi[1] = 25;
char* pc = pi;
if (pc[1] == 25) {...}          //pc[1] khác với pi[1];
```

6. Thuộc tính của field dữ liệu trong record có đúng với nội dung gốc không ?

```
struct {int i; double d;} T_Rec;
T_Rec rec;
read(fdin,&rec, sizeof(T_Rec);
if (rec.i ==10) {...}           //lỗi nếu field d nằm trước i
                                  //trong record gốc trên file
```

7. Cấu trúc kiểu record có tương thích giữa client/server không ?

```
Private Type OSVERSIONINFO
        dwOSVersionInfoSize As Long
```

```
        dwMajorVersion As Long
        dwMinorVersion As Long
        dwBuildNumber As Long
        dwPlatformId As Long
        szCSDVersion As String * 128 ' Maintenance string
    End Type
    Private Declare Function GetVersionEx Lib "kernel32" _
    Alias "GetVersionExA" (lpVersionInformation As
OSVERSIONINFO) As Long
```

8. Dùng chỉ số bị lệch ?

```
int i, pi[10];
for (i = 1; i <= 10; i++) pi [i] = i;
```

9. Class có hiện thực đủ các tác vụ trong interface mà nó hiện thực không ?

10. Class có hiện thực đủ các tác vụ "pure virtual" của class cha mà nó thừa kế không ?

Các lỗi khai báo dữ liệu

1. Tất cả các biến đều được định nghĩa hay khai báo tường minh chưa?

```
int i;
extern double d;
d = i*10;
```

2. Định nghĩa hay khai báo đầy đủ các thuộc tính của biến dữ liệu chưa?

```
static int i = 10;
```

3. Biến array hay biến chuỗi được khởi động đúng chưa ?

```
int pi[10] = {1, 5,7,9} ;
```

4. Kiểu và độ dài từng biến đã được định nghĩa đúng theo yêu cầu chưa ?

```
short IPAddress;
byte Port;
```

5. Giá trị khởi động có tương thích với kiểu biến ?
   short IPAddress = inet_addr("203.7.85.98");
   byte Port = 65535;
6. Có dùng các biến ý nghĩa khác nhau nhưng tên rất giống nhau không?
   int count, counts;

Các lỗi tính toán (Computation Errors)

1. Thực hiện phép toán trên toán hạng không phải là số?
   CString s1, s2;
   int ketqua = s1/s2;
2. Thực hiện phép toán trên các toán hạng có kiểu không tương thích ?
   byte b;
   int i;
   double d;
   b = i * d;
3. Thực hiện phép toán trên các toán hạng có độ dài khác nhau ?
   byte b;
   int i;
   b = i * 500;
4. Gán dữ liệu vào biến có độ dài nhỏ hơn ?
   byte b;
   int i;
   b = i * 500;
5. Kết quả trung gian bị tràn?
   byte i, j, k;
   i = 100; j = 4;
   k = i * j / 5;
6. Phép chia có mẫu bằng 0 ?
   byte i, k;

```
i = 100 / k;
```

7. Mất độ chính xác khi mã hóa/giải mã số thập phân/số nhị phân ?
8. Giá trị biến nằm ngoài phạm vi ngữ nghĩa ?
```
int tuoi = 3450;
tuoi = -80;
```
9. Thứ tự thực hiện các phép toán trong biểu thức mà người lập trình mong muốn có tương thích với thứ tự mà máy thực hiện? Người lập trình hiểu đúng về thứ tự ưu tiên các phép toán chưa ?
```
double x1 = (-b-sqrt(delta)) / 2*a;
```
10. Kết quả phép chia nguyên có chính xác theo yêu cầu không ?
```
int i = 3;
if (i/2*2) == i) {...}
```

Các lỗi so sánh (Comparison Errors)

1. So sánh 2 dữ liệu có kiểu không tương thích ?
```
int ival;
char sval[20];
if (ival == sval) {...}
```
2. So sánh 2 dữ liệu có kiểu không cùng độ dài ?
```
int ival;
char cval;
if (ival == cval) {...}
```
3. Toán tử so sánh đúng ngữ nghĩa mong muốn? Dễ lộn giữa = và ≠, <= và >=, and và or...
4. Có nhầm lẫn giữa biểu thức Bool và biểu thức so sánh ?
```
if (2 < i < 10) {...}
if (2 < i  &&  i < 10) {...}
```
5. Có hiểu rõ thứ tự ưu tiên các phép toán ?

if(a==2 && b==2 || c==3) {...}

6. Cách thức tính biểu thức Bool của chương trình dịch như thế nào ?
   if(y==0 || (x/y > z))

Các lỗi luồng điều khiển (Control-Flow Errors)

1. Thiếu thực hiện 1 số nhánh trong lệnh quyết định theo điều kiện số học ?
   switch (i) {
   case 1: ... //cần hay không cần lệnh break;
   case 2: ...
   case 3: ...
   }
2. Mỗi vòng lặp thực hiện ít nhất 1 lần hay sẽ kết thúc ?
   for (i=x ; i<=z; i++) {...} //nếu x > z ngay từ đầu thì sao ?
   for (i = 1; i <= 10; i--) {...} //có dừng được không ?
3. Biên của vòng lặp có bị lệch ?
   for (i = 0; i <= 10; i++) {...} //hay i < 10 ?
4. Có đủ và đúng cặp token begin/end, {} ?

Các lỗi giao tiếp (Interface Errors)

1. Số lượng tham số cụ thể được truyền có = số tham số hình thức của hàm được gọi ?
2. thứ tự các tham số có đúng không ?
3. thuộc tính của từng tham số thực có tương thích với thuộc tính của tham số hình thức tương ứng của hàm được gọi ?
   char* str = "Nguyen Van A";
   MessageBox (hWnd, str,"Error", MB_OK); //sẽ bị lỗi khi dịch ở chế độ Unicode
4. Đơn vị đo lường của tham số thực giống với tham số hình thức ?
   double d = cos (90);

5. Tham số read-only có bị thay đổi nội dung bởi hàm không ?
6. Định nghĩa biến toàn cục có tương thích giữa các module chức năng không ?

Các lỗi nhập/xuất (Input/Output Errors)

1. Lệnh mở/tạo file có đúng chế độ và định dạng truy xuất file?

```
if ((fdout = open ("tmp0", O_WRONLY| O_CREAT| O_BINARY, S_IREAD| S_IWRITE)) < 0)
        pr_error_exit("Khong the mo file tmp0 de ghi");
if ((fdtmp = open ("tmp2", O_RDWR | O_CREAT | O_BINARY, S_IREAD | S_IWRITE)) < 0)
```

2. Kích thước của buffer có đủ chứa dữ liệu đọc vào không ?

```
char buffin[100];
sl = read(fd, bufin, MAXBIN); //MAXBIN <= 100 ?
```

3. Có mở file trước khi truy xuất không ?
4. Có đóng file lại sau khi dùng không ? Có xử lý điều kiện hết file ?
5. Có xử lý lỗi khi truy xuất file không ?
6. Chuỗi xuất có bị lỗi từ vựng và cú pháp không ?

Các lỗi khác (Other Checks)

1. Có biến nào không được tham khảo trong danh sách tham khảo chéo (cross-reference)?
2. Cái gì được kỳ vọng trong danh sách thuộc tính ?
3. Có các cảnh báo hay thông báo thông tin ?
4. Có kiểm tra tính xác thực của dữ liệu nhập chưa ?
5. Có thiếu hàm chức năng ?

7.4 Phương pháp chạy thủ công mã nguồn

Giống như phương pháp thanh kiểm tra mã nguồn, phương pháp này bao gồm 1 tập các thủ tục và kỹ thuật phát hiện lỗi dành cho 1 nhóm người đọc mã nguồn.

Cần 1 cuộc họp dài từ 1 tới 4 giờ và không được ngắt quảng giữa chừng.

Nhóm chạy thủ công gồm 3 tới 5 người :

- Lập trình viên nhiều kinh nghiệm
- Chuyên gia về ngôn ngữ lập trình được dùng để viết mã nguồn
- Lập trình viên mới
- 1 người mà sẽ bảo trì phần mềm
- 1 người từ project khác, 1 người cùng nhóm với lập trình viên mã nguồn cần kiểm thử.

Các vai trò trong nhóm :

- Chủ tịch điều hành
- Thư ký (người ghi lại các lỗi phát hiện được)
- Người kiểm thử

Thủ tục ban đầu cũng giống như thủ tục ban đầu của phương pháp thanh kiểm tra mã nguồn.

Thủ tục trong cuộc họp :

- Thay vì chỉ đọc chương trình hay dùng danh sách lỗi thường gặp, các thành viên phải biến mình làm CPU để chạy thủ công mã nguồn.
- Người kiểm thử được cung cấp 1 tập các testcase.
- Trong cuộc họp, người kiểm thử sẽ thực thi từng testcase thủ công. Trạng thái chương trình (nội dung các biến) sẽ được ghi và theo dõi trên giấy hay trên bảng.

Lưu ý :

- Các testcase thường ở mức đơn giản để phục vụ như là phương tiện để bắt đầu và gợi câu hỏi người lập trình về logic thuật giải cũng như những giả định của anh ta.
- Thái độ của từng người tham dự là quan trọng.
- Các chú thích nên hướng đến chương trình thay vì đến người lập trình.
- Chạy thử thủ công nên có qui trình theo sau tương tự như đã diễn tả cho phương pháp thanh kiểm tra.
- Phương pháp chạy thủ công mã nguồn cũng tạo được các hệu ứng lề y như phương pháp thanh kiểm tra.

7.5 Phương pháp kiểm tra thủ công (Desk-checking)

Phương pháp kiểm tra thủ công có thể được xem như là phương pháp thanh kiểm tra hay phương pháp chạy thủ công mà chỉ có 1 người tham gia thực hiện : người này sẽ tự đọc mã nguồn, tự kiểm tra theo danh sách lỗi thường gặp hay chạy thủ công và theo dõi nội dung các biến dữ liệu.

Đối với hầu hết mọi người, phương pháp này không được công nghiệp cho lắm :

- Đây là 1 qui trình hoàn toàn không "undisciplined".
- Ta thường không thể kiểm thử hiệu quả chương trình do mình viết vì chủ quan, thiên vị....$\Rightarrow$ nên được thực hiện bởi người khác, chứ không phải là tác giả của phần mềm.

Kém hiệu quả hơn nhiều so với 2 phương pháp trước. Hiệu ứng synergistic của 2 phương pháp trước.

7.6 Phương pháp so sánh với phần mềm tương tự (Peer Ratings)

Phương pháp này không kiểm thử trực tiếp phần mềm, mục tiêu của nó không phải để tìm lỗi của phần mềm.

Đây là kỹ thuật đánh giá, so sánh các tính chất của các chương trình tương tự như chất lượng tổng thể, độ bảo trì, độ mở rộng, độ sự dụng, độ trong sáng...

Mục đích của phương pháp này là cung cấp cho người lập trình 1 sự tự đánh giá.

7.7 Kết chương

Chương này đã trình bày lý do vì sao chúng ta cần kiểm thử TPPM một cách tĩnh, nghĩa là không cần dùng máy tính chạy trực tiếp TPPM mà chỉ khảo sát xem xét TPPM thủ công thông qua mắt người.

Chúng ta đã trình bày các phương pháp kiểm thử tĩnh TPPM như Desk Checking, thanh kiểm tra, chạy thủ công, peer rating. Ứng với mỗi phương pháp, chúng ta đã trình bày các tính chất cơ bản của phương pháp đó, nguồn nhân lực cần thiết và qui trình thực hiện kiểm thử.

Chương 8

Kiểm thử module (đơn vị)

8.1 Giới thiệu

Kiểm thử module (hay kiểm thử đơn vị) là quá trình kiểm thử từng chương trình con, từng thủ tục nhỏ trong chương trình. Một số động cơ của việc kiểm thử đơn vị :

- Kiểm thử đơn vị là 1 cách quản lý nhiều phần tử cần kiểm thử, bắt đầu tập trung chú ý trên từng phần tử nhỏ của chương trình.
- Kiểm thử đơn vị giúp dễ dàng việc debug chương trình.
- Kiểm thử đơn vị tạo cơ hội tốt nhất cho ta thực hiện kiểm thử đồng thời bởi nhiều người.

Mục đích của kiểm thử đơn vị : so sánh chức năng thực tế của từng module với đặc tả chức năng hay đặc tả interface của module đó. Sự so sánh này có tính chất :

1. Không chỉ ra việc module có thoả mãn đầy đủ đặc tả chức năng ?
2. Mà chỉ ra việc module có làm điều khác biệt gì so với đặc tả của module.

8.2 Thiết kế testcase

Hai tài nguyên thiết yếu sau sẽ cần thiết cho việc thiết kế các testcase :

- Đặc tả chức năng module : nêu rõ các thông số đầu vào, đầu ra và các chức năng cụ thể chi tiết của module.
- Mã nguồn của module.

Tính chất các testcase là dựa chủ yếu vào kỹ thuật kiểm thử hợp trắng :

- Khi kiểm thử phần tử ngày càng lớn hơn thì kỹ thuật kiểm thử hộp trắng ít khả thi hơn.
- Việc kiểm thử sau đó thường hướng đến việc tìm ra các kiểu lỗi (lỗi phân tích, lỗi nắm bắt yêu cầu phần mềm).

Thủ tục thiết kế testcase

Phân tích luận lý của module dựa vào 1 trong các kỹ thuật kiểm thử hộp trắng.

Áp dụng các kỹ thuật kiểm thử hộp đen vào đặc tả của module để bổ sung thêm các testcase khác.

8.3 Kiểm thử không tăng tiến

Để thực hiện qui trình kiểm thử các module, hãy để ý 2 điểm chính :

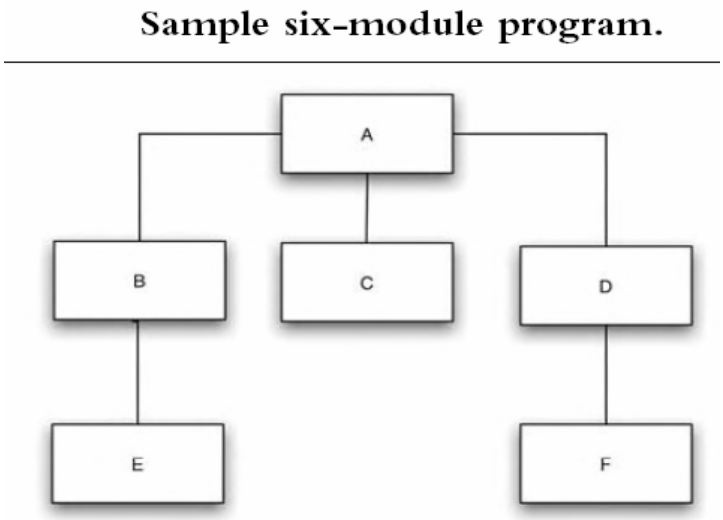
1. Làm sao thiết kế được 1 tập các testcase hiệu quả.
2. Cách thức và thứ tự tích hợp các module lại để tạo ra phần mềm chức năng :
    - Viết testcase cho module nào ?
    - Dùng loại tiện ích nào cho kiểm thử ?
    - Coding và kiểm thử các module theo thứ tự nào ?
    - Chi phí tạo ra các testcase ?
    - Chi phí debug để tìm và sửa lỗi ?

Có 2 phương án để kiểm thử các module :

1. Kiểm thử không tăng tiến hay kiểm thử độc lập (Non-incremental testing) : kiểm thử các module chức năng độc lập nhau, sau đó kết hợp chúng lại để tạo ra chương trình.
2. Kiểm thử tăng tiến (Incremental testing) : kết hợp module cần kiểm thử vào bộ phận đã kiểm thử (lúc đầu là null) để kiểm thử module cần kiểm thử trong ngữ cảnh.

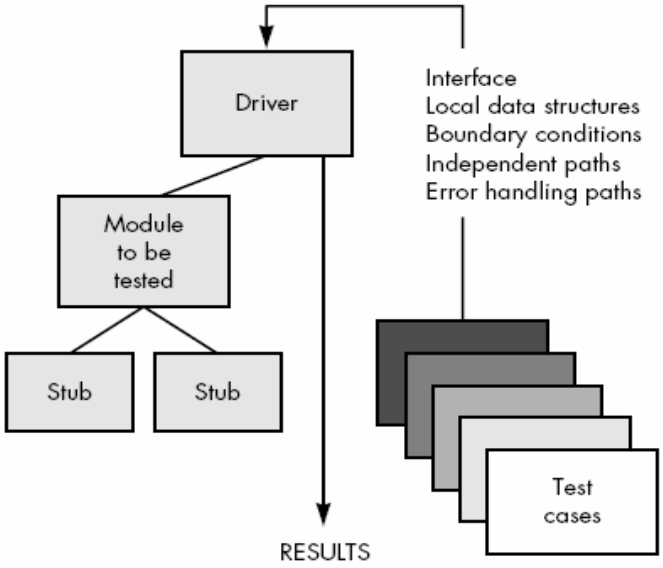
Các bước kiểm thử không tăng tiến :

1. Kiểm thử từng module chức năng 1 cách độc lập.
2. Tích hợp chúng lại thành chương trình.
3. Để kiểm thử 1 module độc lập, ta cần viết 1 Driver và nhiều Stub cho nó.



Driver là module có nhiệm vụ kích hoạt các testcase để kiểm thử module đang cần kiểm thử.

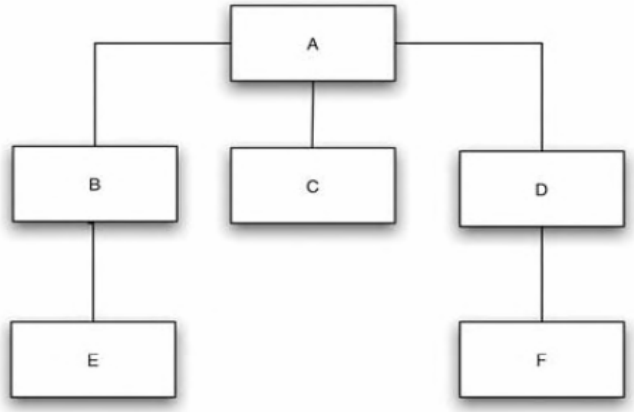
Stub là một hiện thực ở mức độ tối thiểu nào đó cho 1 module chức năng được dùng bởi module đang cần kiểm thử.



Các ý tưởng kiểm thử tăng tiến :

- Trước khi kiểm thử module mới, ta tích hợp nó vào tập các module đã kiểm thử rồi.
- Tích hợp các module theo thứ tự nào ? Từ trên xuống (top-down) hay từ dưới lên (bottom-up).
- Có phải kiểm thử tăng tiến tốt hơn kiểm thử không tăng tiến ?

Sample six-module program.



Một số ý quan sát :

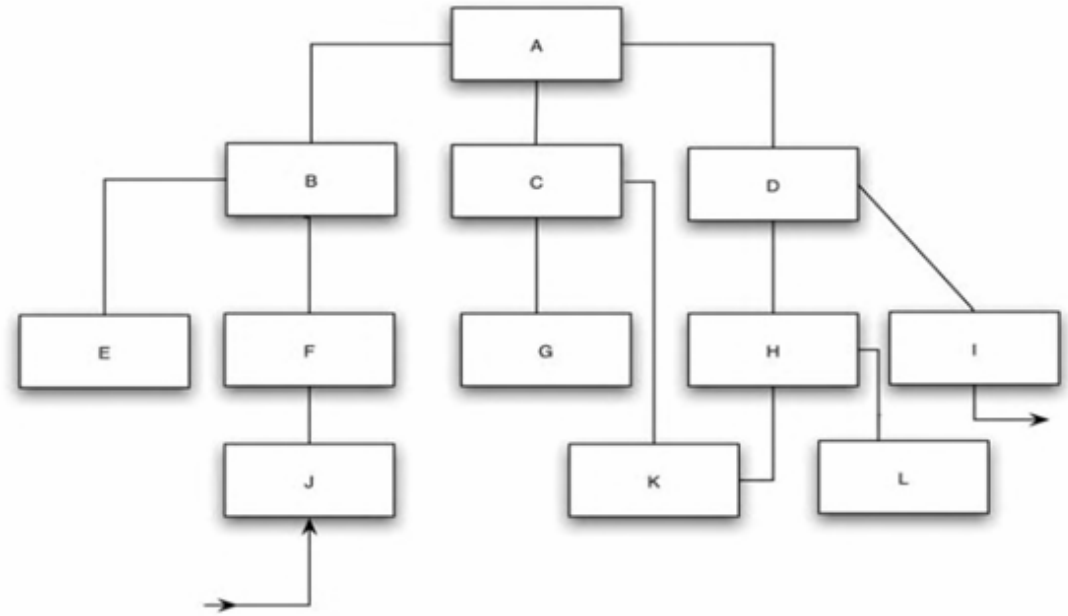
- Kiểm thử tăng tiến cần nhiều công sức hơn kiểm thử không tăng tiến.

- Các lỗi liên quan đến giao tiếp giữa các module sẽ được phát hiện sớn hơn vì việc tích hợp các module xảy ra sớm hơn so với kiểm thử không tăng tiến.
- Kiểm thử tăng tiến cũng sẽ giúp debug dễ dàng hơn.
- Kiểm thử tăng tiến sẽ hiệu quả hơn.
- Kiểm thử không tăng tiến dùng ít thời gian máy hơn (vì chỉ tiến hành trên từng module độc lập).
- Ở đầu chu kỳ kiểm thử module, kiểm thử không tăng tiến tạo cơ hội tốt cho việc kiểm thử đồng thời trên nhiều module khác nhau.

8.4 Kiểm thử từ trên xuống (top-down)

Gồm các bước theo thứ tự :

1. Bắt đầu từ module gốc ở trên cùng của cây cấu trúc chương trình.
2. Sau khi kiểm thử xong module hiện hành, ta chọn module kế tiếp theo ý tưởng :
    - Module kế tiếp phải được dùng trực tiếp bởi module được kiểm thử rồi.
    - Vì có nhiều module cùng thỏa mãn điều kiện trên, nên ta chọn module thực hiện nhiều hoạt động I/O trước.
    - Rồi chọn module "critical", là module dùng thuật giải phức tạp, tiềm ẩn nhiều lỗi và/hoặc lỗi nặng nhất.



Kiểm thử module A trước. Để kiểm thử module A, ta cần phải xây dựng các Stub cho 3 module mà A phụ thuộc trực tiếp là B, C, D.

Tạo các testcase cho module A như thế nào ?

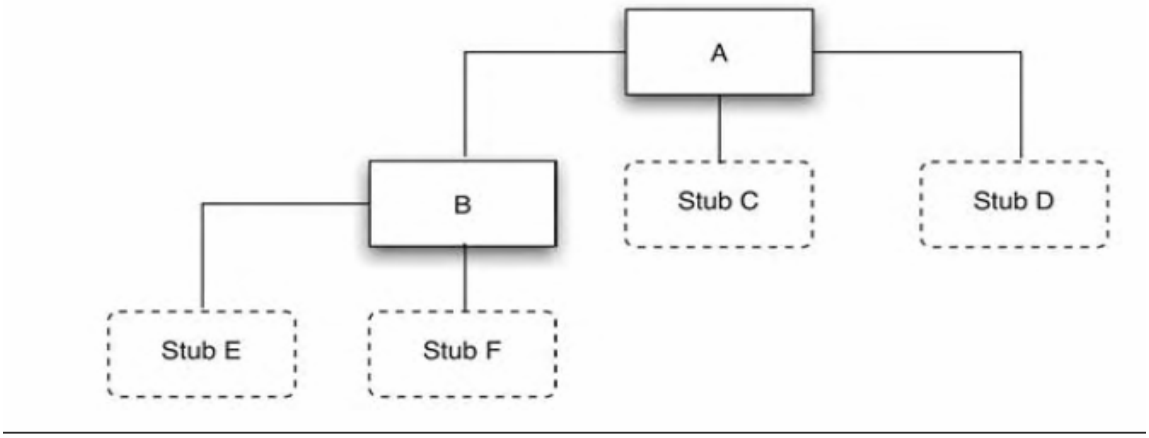
- Các Stubs sẽ có nhiệm vụ cung cấp dữ liệu cho module cần kiểm thử :
  - B—cung cấp thông tin tổng kết về giao tác.
  - C—xác định trạng thái hàng tuần có thỏa quota không?
  - D—tạo báo cáo tổng kết hàng tuần.
- Như vậy 1 testcase cho A là tổng kết giao tác từ B gởi về, Stub D có thể chứa các lệnh để xuất dữ liệu ra máy in để ta có thể xem xét kết quả của mỗi test case.

Nếu module A gọi module B chỉ 1 lần, làm sao cung cấp nhiều dữ liệu test khác nhau cho A ?

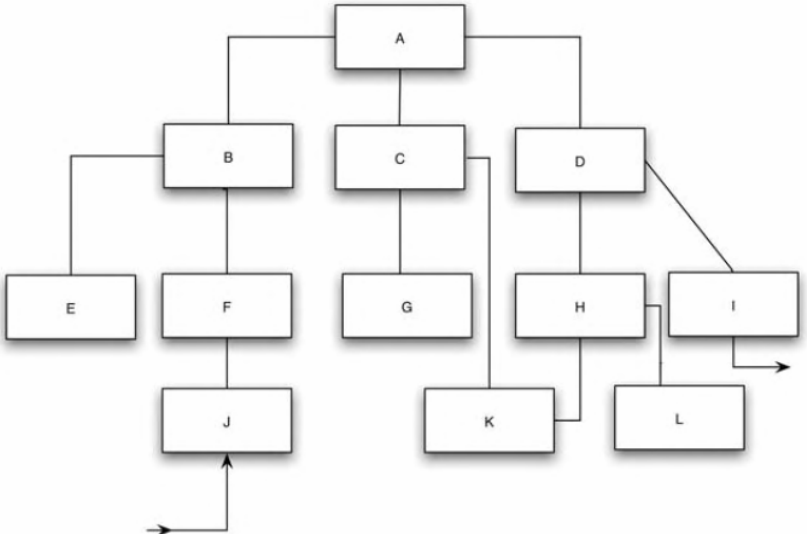
- Viết nhiều version khác nhau cho Stub B, mỗi version cung cấp 1 dữ liệu test xác định cho A.
- Đặt dữ liệu test ở file bên ngoài B, Stub B đọc vào và return cho A.

Sau khi kiểm thử xong module hiện hành, ta chọn 1 trong các Stub và thay thế nó bằng module thật rồi kiểm thử module thật này :

Second step in the top-down test.



Có nhiều thứ tự kiểm thử khác nhau có thể được chọn như dưới đây, và nếu cần kiểm thử đồng thời, cũng có nhiều thứ tự khác nữa.

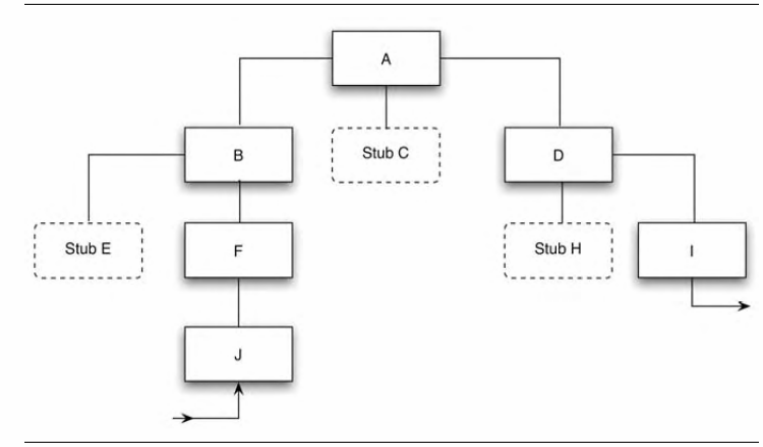


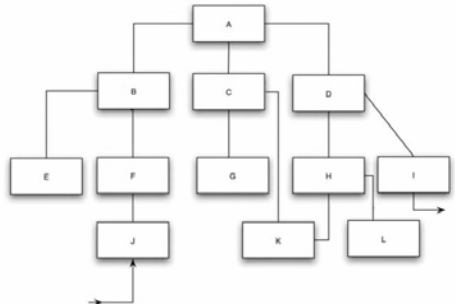
1. A B C D E F G H I J K L
2. A B E F J C G K D H L I
3. A D H I K L C G B F J E
4. A B F J D I E C G K H L
5. ...

Nếu module J và I thực hiện nhập/xuất dữ liệu, còn module G là critical, ta nên chọn thứ tự kiểm thử tăng tiến sau đây :

A B F J D I C G E K H L

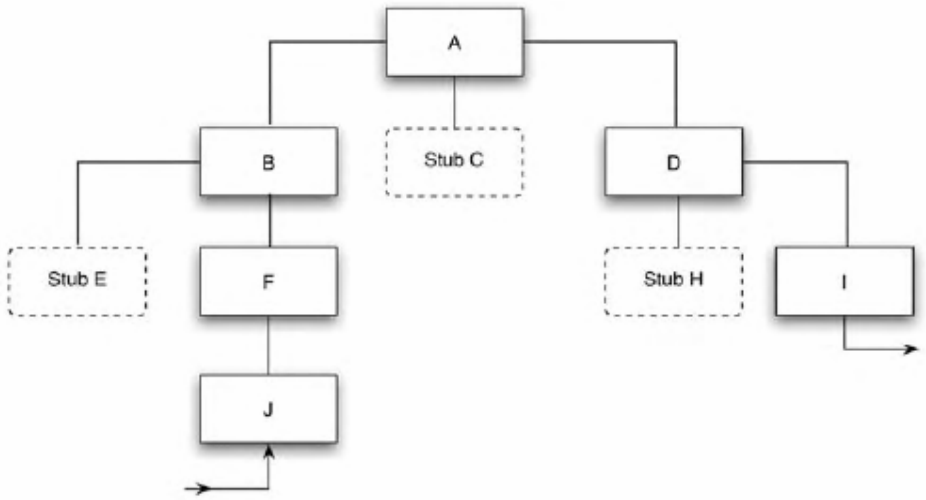
Intermediate state in the top-down test.





Một khi đã kiểm thử đến trạng thái như hình dưới đây :

- Việc miêu tả các testcase và việc thanh tra kết quả sẽ đơn giản.
- Ta đã có 1 version chạy được của chương trình, nó thực hiện được các hoạt động nhập/xuất dữ liệu.
- Ta có cảm giác chương trình hoàn chỉnh đến rất gần rồi.



Ưu điểm của phương pháp top-down

- Nếu các lỗi xảy ra có khuynh hướng nằm trong các module mức cao thì phương pháp top-down sẽ giúp phát hiện sớm chúng.
- Một khi các hoạt động nhập/xuất dữ liệu đã được thêm vào hệ thống thì việc miêu tả các test case sẽ dễ dàng hơn.
- Chương trình khung sườn sớm hình thành để demo và tiếp thêm sức mạnh tinh thần cho những người phát triển phần mềm.

Khuyết điểm của phương pháp top-down

- Phải viết các Stub để kiểm thử module dùng chúng, và viết Stub thường phức tạp hơn nhiều so với suy nghĩ của chúng ta.
- Trước khi các hoạt động nhập/xuất được tích hợp vào hệ thống, việc miêu tả các testcase trong các Stub có thể gặp khó khăn.
- Việc tạo ra điều kiện kiểm thử sẽ rất khó và nhiều lúc là không khả thi : Do có khoảng cách khá xa giữa module cần test và module nhập dữ liệu cung cấp cho module cần test nên rất khó để cung cấp dữ liệu gì để có thể kiểm thử 1 tình huống xác định của module cần kiểm thử.

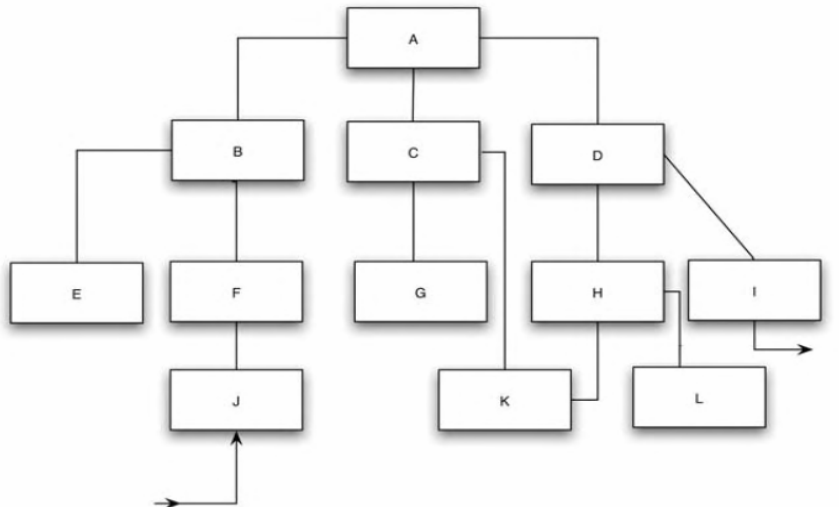
- Quan sát kết quả kiểm thử sẽ gặp khó khăn : Tương tự, xem xét sự tương quan dữ liệu xuất của 1 module và dữ liệu nhập tạo dữ liệu xuất này (nhưng ở trong module có khoảng cách khá xa) sẽ rất khó khăn.
- Nó làm ta nghĩ rằng việc thiết kế và kiểm thử có cùng thứ tự thực hiện : ta sẽ cảm nhận rằng hoạt động kiểm thử và hoạt động thiết kế gối đầu nhau : thiết kế tới đâu thì kiểm thử tới đó. Điều này thật nguy hiểm vì nếu ta tiến hành thiết kế và kiểm thử gối đầu nhau như vậy thì khi kiểm thử tới các module phía dưới mà đòi hỏi hiệu chỉnh bản thiết kế của module phía trên thì sẽ gây lãng phí rất lớn (vì phải huỷ bỏ kết quả đã có và thiết kế lại từ đầumodule phía trên).
- Nó trì hoãn việc kiểm thử 1 số module.
- Nó làm ta dễ quên hiện thực module chức năng vì đã có Stub thay thế.
- Khó lòng kiểm thử đầy đủ module cần kiểm thử trước khi tiến hành kiểm thử module khác. Điều này là do 2 lý do chính sau đây :
  - Các module Stub khó lòng tạo được tất cả dữ liệu thật mà module thực sự tương ứng sẽ tạo ra.
  - Các module cấp trên của cấu trúc chương trình thường chứa các đoạn code tạo, thiết lập trạng thái đầu của các tài nguyên mà sẽ được dùng trong các module phía dưới, nhưng hiện nay module phía dưới chưa được kiểm thử nên không thể kiểm thử các đoạn code thiết lập tài nguyên này được.

8.5 Kiểm thử từ dưới lên (bottom-up)

Gồm các bước theo thứ tự :

1. Bắt đầu từ 1 hay nhiều module lá : module mà không gọi module nào khác.

2. Sau khi kiểm thử xong module hiện hành, ta chọn module kế tiếp theo ý tưởng :
    - Module kế tiếp phải dùng trực tiếp 1 hay nhiều module được kiểm thử rồi.
    - Vì có nhiều module cùng thỏa mãn điều kiện trên, nên ta chọn module thực hiện nhiều hoạt động I/O trước.
    - Rồi chọn module "critical", là module dùng thuật giải phức tạp, tiềm ẩn nhiều lỗi và/hoặc lỗi nặng nhất.

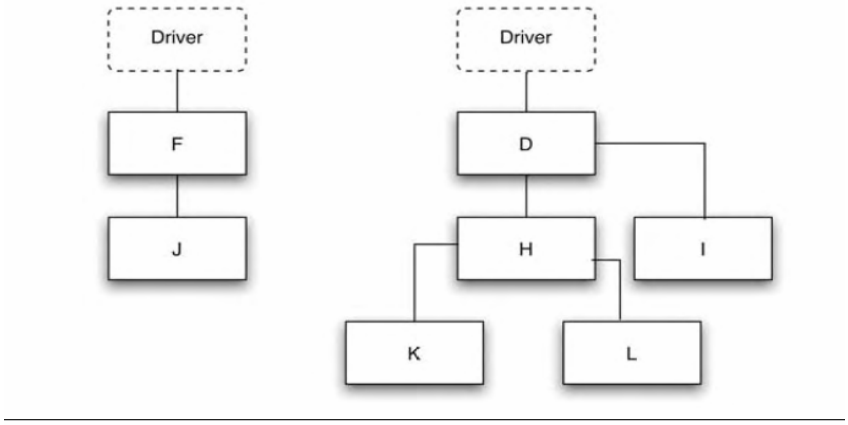


Các module E, J, G, K, L và I được kiểm thử trước.

Để kiểm thử 1 module, ta phải viết driver cho nó. Không cần phải viết nhiều driver khác nhau cho cùng 1 module. Trong đại đa số trường hợp, viết driver dễ hơn nhiều so với viết Stub.

Nếu D và F là 2 module critical nhất, thì ta nên kiểm thử theo trình tự của hình vẽ dưới đây :

Intermediate state in the bottom-up test.



Ưu & khuyết điểm của phương pháp bottom-up

- Ưu :
  - Nếu các lỗi xảy ra có khuynh hướng nằm trong các module mức thấp thì phương pháp bottom-up sẽ giúp phát hiện sớm chúng.
  - Việc tạo các điều kiện kiểm thử sẽ dễ dàng hơn.
  - Việc quan sát các kết quả kiểm thử cũng dễ dàng hơn.
- Khuyết :
  - Phải viết các module driver, mặc dù việc viết các module này khá dễ dàng.
  - Chương trình khung sườn chưa tồn tại 1 thời gian dài cho đến khi module cuối cùng được tích hợp vào hệ thống.

8.6 Kết chương

Chương này đã trình bày các vấn đề cơ bản về hoạt động kiểm thử đơn vị, hay còn gọi là kiểm thử module.

Chúng ta cũng đã trình bày các kỹ thuật kiểm thử đơn vị thường dùng như kỹ thuật kiểm thử không tăng tiến, kỹ thuật kiểm thử tăng tiến từ trên xuống, kỹ thuật kiểm thử tăng tiến từ dưới lên cùng các ưu/khuyết điểm của từng kỹ thuật.

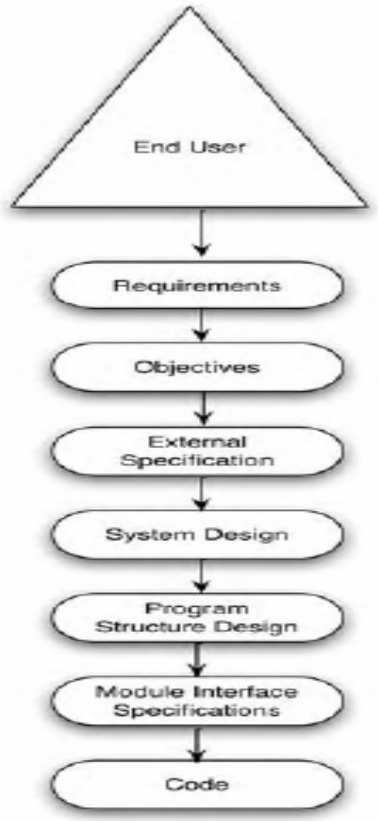
Chương 9

Các hoạt động kiểm thử khác

9.1 Giới thiệu

Sau khi kiểm thử mọi đơn vị chức năng phần mềm và sửa lỗi hoàn chỉnh cho chúng, ta cũng không thể đảm bảo là đã tìm hết lỗi trong phần mềm. Thật vậy, còn nhiều lỗi khác mà kiểm thử đơn vị chưa phát hiện được. Tại sao vậy ?

Như chúng ta biết trong qui trình phát triển phần mềm, ta đã thực hiện 1 số workflows như :

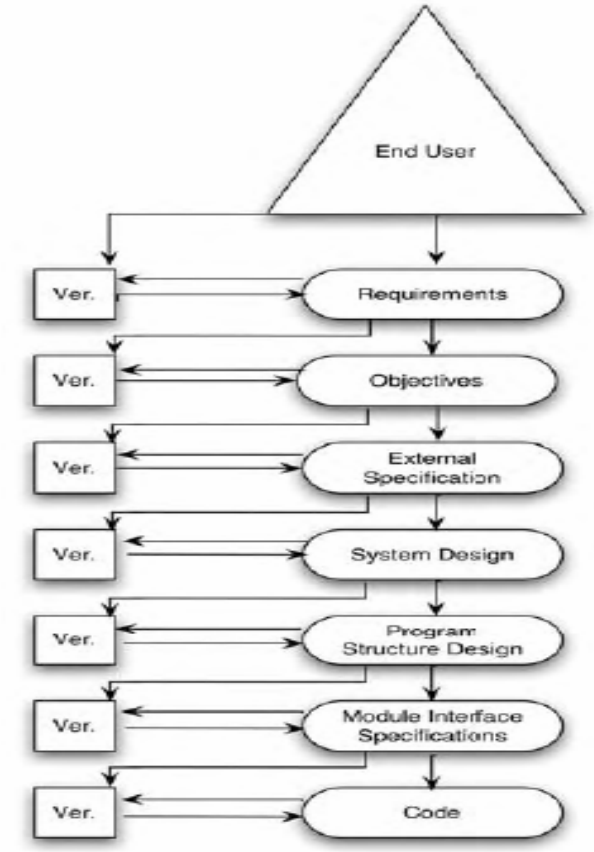


1. Xác định các yêu cầu để biết rõ tạo sao phần mềm là cần thiết.
2. Xác định các mục tiêu của phần mềm để biết rõ những gì phần mềm phải thực hiện và mức độ thực hiện chúng như thế nào ?

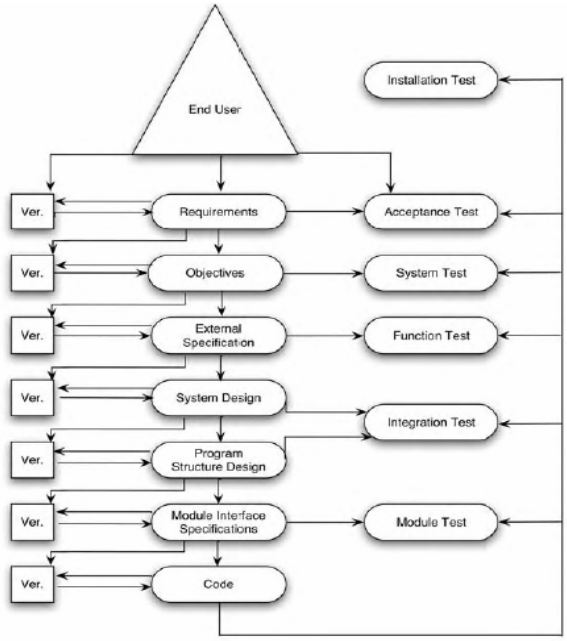
3. Đặc tả các chức năng mà người dùng thấy về phần mềm.
4. Thiết kế hệ thống và thiết kế cấu trúc cụ thể và chi tiết của phần mềm.
5. Đặc tả giao tiếp của từng module chức năng.
6. Hiện thực chi tiết các chức năng của từng module.

Về nguyên tắc, con người có những hạn chế nhất định, kết quả của 1 công việc nào đó đều có thể có lỗi, và nếu dùng kết quả này làm dữ liệu đầu vào cho hoạt động kế tiếp thì kết quả của hoạt động kế cũng sẽ bị lỗi,... Ta thường dùng tổ hợp 2 biện pháp sau đây để hạn chế/ngăn ngừa các lỗi :

- Xác định lại cho rõ ràng và chi tiết hơn từng workflows của qui trình phát triển phần mềm.
- Ở cuối việc thực hiện 1 workflows bất kỳ, cần thêm 1 hoạt động được gọi là "thanh kiểm tra kết quả" để đảm bảo chất lượng kết quả này trước khi dùng nó để thực hiện workflow kế tiếp.



Ứng với mỗi workflow khác nhau, ta xác định và dùng chiến lược kiểm thử phù hợp để dễ dàng xác định các loại lỗi đặc thù của workflow đó.



Mục đích của kiểm thử đơn vị là phát hiện sự khác biệt giữa đặc tả giao tiếp của đơn vị và thực tế mà đơn vị này cung cấp.

Mục đích của kiểm thử chức năng là chỉ ra rằng chương trình không tương thích với các đặc tả bên ngoài của nó.

Mục đích của kiểm thử hệ thống là chỉ ra rằng chương trình không tương thích với các mục tiêu ban đầu của nó.

Các lợi ích :

- tránh kiểm thử dư thừa.
- ngăn chặn sự quan tâm nhiều vào quá nhiều loại lỗi tại từng thời điểm.

Chú ý : trình tự các hoạt động kiểm thử trong hình ở slide trước không nhất thiết ám chỉ trình tự thời gian kiểm thử tương ứng.

9.2 Kiểm thử chức năng

Qui trình cố gắng tìm ra các khác biệt giữa đặc tả bên ngoài của phần mềm và thực tế mà phần mềm cung cấp.

Đặc tả bên ngoài của phần mềm là đặc tả chính xác về hành vi của phần mềm theo góc nhìn của người dùng thấy.

Kiểm thử chức năng thường sử dụng 1 kỹ thuật kiểm thử hộp đen nào đó :

- Kỹ thuật phân lớp tương đương (Equivalence Class Partitioning).
- Kỹ thuật dùng các bảng quyết định (Decision Tables)
- Kỹ thuật kiểm thử các bộ n thần kỳ (Pairwise)
- Kỹ thật phân tích vùng miền (domain analysis)
- Kỹ thuật dựa trên đặc tả Use Case (Use case)
- ...

Các cách tiếp cận để kiểm thử chức năng phần mềm :

- User Navigation Testing
- Transaction Screen Testing

- Transaction Flow Testing
- Report Screen Testing
- Report Flow Testing
- Database Create/Retrieve/Update/Delete Testing

1. Kiểm thử khả năng duyệt chức năng của người dùng (User Navigation Test)

Các màn hình phục vụ duyệt thực hiện chức năng là màn hình log on/log off, menu bar và hệ thống cây phân cấp các option để thực hiện chức năng, toolbar, tất cả các mối liên kết từ màn hình này tới màn hình khác để thể hiện sự liên tục của hoạt động nghiệp vụ đang cần thực hiện.

Kiểm thử khả năng duyệt chức năng của người dùng tập trung trên :

- khả năng người dùng login vào hệ thống với quyền hạn thích hợp.
- di chuyển qua các màn hình "giao tác" mong muốn 1 cách đúng đắn và logout khỏi phần mềm.

2. Kiểm thử màn hình giao tác (Transaction screen Test)

Màn hình giao tác có các field nhập liệu, list chọn option, các options, các button chức năng (Add, Change, Delete, Submit, Cancel, OK...).

Một vài loại kết quả có thể được hiển thị trên màn hình giao tác sau khi người dùng click button chức năng nào đó.

Công việc của người kiểm thử :

- Thiết kế testcase để xác thực hoạt động của mỗi field dữ liệu, list, option và button trên màn hình giao tác theo các yêu cầu nghiệp vụ, tài liệu người dùng và tài liệu người quản trị.

- Nếu kết quả được hiển thị trên màn hình giao tác, thì kỹ thuật kiểm thử hộp đen với testcase gồm (data input, ouput kỳ vọng) sẽ được dùng để xác thực kết quả hiển thị.

3. Kiểm thử luồng giao tác (Transaction Flow Test)

Kiểm tra kết quả tổng hợp của nhiều màn hình giao tác theo thứ tự duyệt đúng có hoàn thành hoạt động nghiệp vụ tương ứng không ?

Thí dụ nghiệp vụ cập nhật profile khách hàng gồm các màn hình giao tác sau :

- màn hình 1 cập nhật tên, địa chỉ, contact. Màn hình 2 cập nhật credit. Màn hình 3 cập nhật thông tin thanh toán và khuyến mãi. Màn hình 4 tổng kết profile và thực hiện cập nhật. Màn hình 5 để xem kết quả profile đã cập nhật.
- Kết quả cuối cùng của trình tự các màn hình là file hay database sẽ được cập nhật để chứa các thông tin mà người dùng đã cập nhật thông qua các màn hình giao tác.

Nhiệm vụ của người kiểm thử :

- Xác thực rằng nếu người dùng thực hiện đúng trình tự các màn hình giao tác và hoàn tất được chúng thì hệ thống se cung cấp kết quả đúng.
- Ngược lại, nếu người dùng không tuân thủ bất kỳ 1 qui luật nghiệp vụ nào trong 1 màn hình giao tác nào thì hệ thống sẽ không cung cấp kết quả gì cho người dùng.

4. Kiểm thử màn hình báo biểu (Report screen Test)

màn hình báo biểu cho phép tìm kiếm dữ liệu và hiển thị kết quả (không cần nhập dữ liệu như màn hình giao tác).

Khó khăn trong kiểm thử màn hình bào biểu nằm ở chỗ có nhiều cách mà người dùng có thể đặc tả dữ liệu cần được tìm kiếm (tiêu chuẩn) và cách thức dữ liệu này được hiển thị (sắp xếp và định dạng).

Công việc của người kiểm thử là chú ý đặc biệt vào dữ liệu tìm kiếm và hiển thị vì người dùng có thể chọn sai dữ liệu hay tệ hơn là không có kết quả nào được hiển thị.

5. Kiểm thử luồng báo biểu (Report Flow Test)

Kiểm thử các khác biệt giữa kết quả hiển thị trong màn hình báo biểu và các phương thức báo biểu khác (như máy in, file,..).

Nhiệm vụ của người kiểm thử :

- Xác định xem phần mềm gởi cùng kết quả ra màn hình report và máy in ?
- Xác thực kết quả báo biểu trên tất cả phương thức báo cáo khác nhau được hỗ trợ bởi phần mềm.
- Xác định xem khả năng máy in có hỗ trợ font, vùng chọn được người dùng xác định trong màn hình báo biểu ?

6. Kiểm thử việc Create/Retrieve/Update/Delete database

Thường được thực hiện thông qua 2 bước :

- Kiểm thử việc thiết kế, khởi tạo database ban đầu thông qua tiện ích bên ngoài phần mềm ứng dụng cần kiểm thử.
- Kiểm thử việc phần mềm sử dụng database đã được thiết kế và khởi tạo đúng.

Đòi hỏi sự hợp tác và cộng tác giữa người kiểm thử và người quản trị database.

9.3 Kiểm thử hệ thống

Kiểm thử hệ thống không phải là qui trình kiểm thử toàn bộ chức năng của 1 chương trình hay của 1 hệ thống phần mềm đầy đủ.

Mục đích của kiểm thử hệ thống là so sánh hệ thống hay chương trình với các mục tiêu ban đầu của nó.

Kiểm thử hệ thống không bị hạn chế với các hệ thống phần mềm. Nếu sản phẩm cần kiểm thử là 1 chương trình, kiểm thử hệ thống là qui trình cố gắng chứng minh cách mà toàn bộ phần mềm không thỏa mản các mục tiêu của nó.

Theo định nghĩa trên, kiểm thử hệ thống không thể xảy ra được nếu ta không viết ra rõ ràng các thông tin đo đạt được về các mục tiêu của chương trình.

Thí dụ về đặc tả mục tiêu của chương trình :

- Hãy hiện thực 1 hàng lệnh để từ cửa sổ text-mode, người dùng xem các nội dung chi tiết về các ô nhớ trong bộ nhớ chính của phần mềm.
- Cú pháp của hàng lệnh nên nhất quán với cú pháp của các lệnh khác mà hệ thống cung cấp.
- Người dùng nên có thể đặc tả vùng nhớ thông qua 2 địa chỉ đầu cuối hay thông qua địa chỉ đầu và số lượng ô nhớ cần xem.
- Các toán hạng của lệnh nên có nhiệm ý gợi nhớ.
- Kết quả nên xuất trên nhiều hàng, nội dung của từng ổ nhớ ở dạng hex cách nhau bởi 1 hay nhiều khoảng trắng.
- Mỗi hàng nên chứa địa chỉ của ô nhớ đầu hàng đó.
- Lệnh là bình thường, nghĩa là nếu máy đang chạy bình thường, nó sẽ bắt đầu xuất kết quả trong vài giây và kết quả xuất không có thời gian chờ giữa các ô nhớ trong hàng hay giữa các hàng.
- Lỗi lập trình ít nhất nên làm cho lệnh bị dừng, hệ thống và session người dùng không bị ảnh hưởng gì hết.
- Sau khi hệ thống bắt đầu xuất kết quả, bộ xử lý lệnh không nên có hơn 1 lỗi do người dùng phát hiện được.

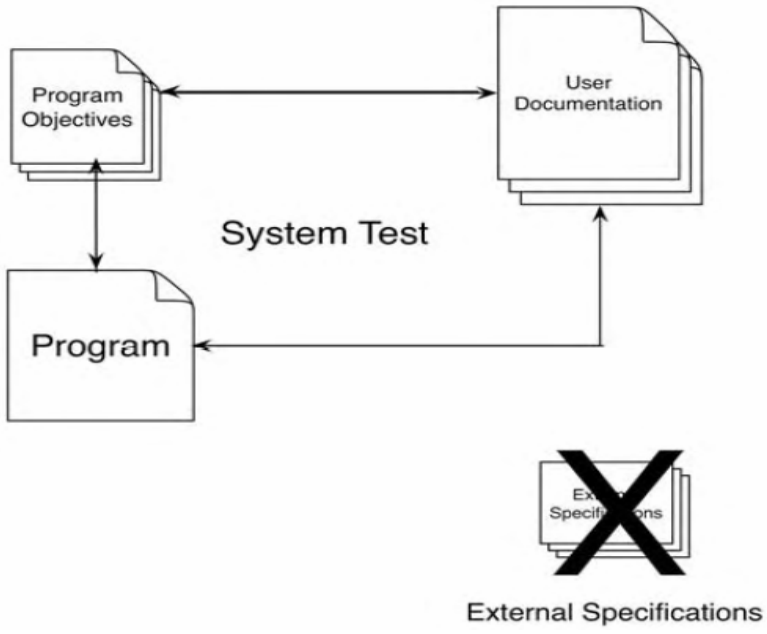
Kiểm thử hệ thống là qui trình kiểm thử quan trọng.

Các chất liệu để tạo testcase cho kiểm thử hệ thống :

- Chúng ta không chỉ dùng đặc tả theo góc nhìn người dùng để suy ra các testcase.
- Tài liệu về mục tiêu chương trình, tự nó cũng không thể được dùng để tạo ra các testcase, vì theo định nghĩa, nó không chứa các miêu tả chính xác, rõ ràng về giao tiếp từ ngoài vào chương trình.
- ta sẽ dùng tài liệu sử dụng và những công bố cho người dùng.

Thiết kế testcase bằng cách phân tích các mục tiêu.

Tạo các testcase bằng cách phân tích tài liệu dành cho người dùng.



Kiểm thử hệ thống là hoạt động kiểm thử khó khăn nhất

Phải so sánh chương trình với các mục tiêu ban đầu : nên không có phương pháp luận thiết kế testcase tường minh.

Dùng cách tiếp cận khác để thiết kế testcase :

- Thay vì miêu tả phương pháp luận, các loại testcase riêng biệt sẽ được đề cập.

- Do không có phương pháp luận, kiểm thử hệ thống đòi hòi rất nhiều sự năng động và sáng tạo.

Kiểm thử hệ thống tập trung vào kiểm thử các yêu cầu không chức năng. Có 15 yêu cầu không chức năng sau đây có thể cần kiểm thử (nhưng không phải phần mềm nào cũng đòi hỏi đủ 15 yêu cầu này) :

- Facility Testing
- Volume Testing
- Stress Testing
- Usability Testing
- Security Testing
- Performance Testing
- Storage Testing
- Configuration Testing
- Compatibility/Configuration/Conversion Testing
- Installability Testing
- Reliability Testing
- Recovery Testing
- Serviceability Testing
- Documentation Testing
- Procedure Testing

1. Kiểm thử phương tiện (Facility Test)

xác định xem mỗi phương tiện được đề cập trong phần mục tiêu của chương trình đã được hiện thực thực sự chưa.

Qui trình kiểm thử :

- Dò nội dung, từng câu một, miêu tả mục tiêu.

- Khi 1 câu miêu tả cái gì, xác định chương trình đã thỏa mãn cái đó chưa.

Thường ta có thể thực hiện kiểm thử phương tiện mà không cần chạy máy tính, so sánh bằng trí óc các mục tiêu với tài liệu sử dụng đôi khi đủ rồi.

2. Kiểm thử dung lượng (Volume Test)

Mục đích của kiểm thử dung lượng là chỉ ra rằng chương trình không thể xử lý khối lượng dữ liệu lớn được đặc tả trong bảng đặc tả mục tiêu chương trình.

Thí dụ :

- Chương trình dịch không thể dịch file mã nguồn dài 10MB.
- Trình liên kết không thể liên kết 1000 module chức năng khác nhau lại.
- Trình xem phim không thể chiếu file film dài 15GB.

Kiểm thử dung lượng thường đòi hỏi rất nhiều tài nguyên, con người lẫn máy tính.

3. Kiểm thử tình trạng căng thẳng (StressTest)

Mục đích của kiểm thử tình trạng căng thẳng là chỉ ra rằng chương trình sẽ không thể hoạt động được hay hoạt động không tốt trong tình huống căng thẳng : quá nhiều yêu cầu đồng thời, quá nhiều chương trình khác đang cạnh tranh tài nguyên,...

Thí dụ :

- web server sẽ bế tắc nếu có 100000 yêu cầu truy xuất trang web đồng thời.
- HĐH không thể quản lý 1000 process chạy đồng thời.
- Trình chiếu phim sẽ không chiếu phim mượt và tốt nếu có nhiều chương trình khác cần rất nhiều tài nguyên đang chạy.

4. Kiểm thử độ khả dụng (UsabililyTest)

Mục đích của kiểm thử độ khả dụng là chỉ ra các phương tiện/kết quả nhập/xuất không phù hợp, thân thiện với người dùng :

- Mỗi đối tượng giao diện có thân thiện, tự nhiên và dễ dùng không ?
- Kết quả xuất có ngắn gọn, trong sáng, nghĩa dễ hiểu không ?
- Các cảnh báo có dễ hiểu không ? "IEK022A OPEN ERROR ON FILE 'SYSIN' ABEND CODE=102?"
- Nói chung tất cả các kết quả, các cảnh báo đều phải nhất quán, đồng nhất về cú pháp, về định dạng, ngay cả các từ viết tắt được dùng.

Một số chú ý :

- Khi độ chính xác là rất quan trọng như trong hệ thống quản lý ngân hàng, thì thông tin nhập có tính dư thừa đủ không ?
- Hệ thống có quá nhiều nhiệm ý hay các nhiệm ý được người dùng thích dùng không ?
- Hệ thống có trả về đủ đáp ứng với mọi hoạt động nhập ?
- Chương trình có dễ dùng và thân thiện ?

5. Kiểm thử các dịch vụ cộng thêm (Serviceability Test)

Trong mục tiêu của phần mềm có thể đề cập đến 1 số dịch vụ cộng thêm, thí dụ như :

- Chương trình chẩn đoán và xuất nội dung thô của bộ nhớ chương trình.
- Thời gian trung bình để debug 1 vấn đề rõ ràng.
- Các thủ tục bảo trì.
- Chất lượng của tài liệu luận lý bên trong.

Các mục tiêu trên, nếu có đề cập trong mục tiêu chương trình thì cần phải được kiểm thử.

6. Kiểm thử tính an ninh (Security Test)

An ninh phần mềm gồm 3 vấn đề chính là bảo mật, tính toàn vẹn dữ liệu và độ sẵn sàng đáp ứng.

Nghiên cứu các vấn đề liên quan đến an ninh trong các hệ thống tương tự rồi tạo các testcase để chứng minh rằng các vấn đề này cũng tồn tại trong chương trình cần kiểm thử.

Các ứng dụng mạng và ứng dụng theo công nghệ Web hiện nay cần được kiểm thử tính an ninh ở mức độ cao hơn nhiều so với phần mềm truyền thống trên máy đơn. Điều này đặc biệt đúng cho các website thương mại, ngân hàng...

7. Kiểm thử hiệu xuất làm việc (Performance Test)

Mục đich của kiểm thử hiệu xuất làm việc là chỉ ra rằng phần mềm không đạt được hiệu xuất được đặc tả trong mục tiêu chương trình.

Thí dụ :

- trình chiếu phim full HD không chiếu kịp 20 frame/sec.
- trình nén dữ liệu không nén dữ liệu kịp với tốc độ đề ra.
- trình soạn thảo văn bản không nhận và xử lý kịp các ký tự được nhập bởi người dùng.
- trình ghi DVD không tạo dữ liệu ghi kịp tốc độ mà ổ DVD yêu cầu...

8. Kiểm thử độ sử dụng bộ nhớ (Storage Test)

Mục đich của kiểm thử độ sử dụng bộ nhớ là chỉ ra rằng phần mềm không tuân thủ về dung lượng bộ nhớ tối thiểu/tối đa được đặc tả trong mục tiêu chương trình.

Thí dụ :

- kích thước tối thiểu 128KB không đủ để chạy chương trình.
- chương trình không dùng hết kích thước bộ nhớ tối đa là 4GB.
- chương trình không chạy được khi đĩa còn dung lượng trống tối thiểu là 4MB.
- chương trình không quản lý được dung lượng đĩa là 1TB...

9. Kiểm thử cấu hình làm việc (Configuration Test)

Nhiều chương trình như HĐH, hệ quản trị CSDL, Website,... thường sẽ làm việc được trên nhiều cấu hình phần cứng/phần mềm cấp thấp. Số lượng các cấu hình khác nhau có thể quá lớn, nhưng ta nên chọn 1 số cấu hình phổ dụng nhất để kiểm thử xem chương trình có chạy tốt trên các cầu hình này không.

10. Kiểm thử tính tương thích/chuyển đổi/cấu hình (Compatibility/ Configuration/Conversion Test)

Đời sống của 1 phần mềm thường dài, nhất là phần mềm thương mại của các hãng lớn. Trong cuộc đời của mình, phần mềm được phát triển tăng dần theo từng release, từng version. Về nguyên tắc, version mới sẽ tương thích ngược với version đã có.

Mức độ tương thích, khả năng chuyển đổi định dạng file dữ liệu từ cũ sang mới hay ngược lại, khả năng cấu hình version mới để có thể làm việc như version cũ,... có thể được đặc tả trong mục tiêu của chương trình.

Nếu có thì ta phải kiểm thử các đặc tả này xem version cần kiểm thử có đáp ứng được không.

Thí dụ : Word 2003 có thể cấu hình để chạy y như Word 97 không ?

11. Kiểm thử khả năng cài đặt (Installability Test)

Một số hệ thống phần mềm có thủ tục cài đặt khá phức tạp.

Chương trình cài đặt chạy sai có thể ngăn chận người dùng không dùng được phần mềm được cài đặt.

Nhiệm vụ của kiểm thử khả năng cài đặt là kiểm thử chương trình cài đặt có hoạt động đúng không ?

12. Kiểm thử độ tin cậy (Reliability Test)

Mục tiêu của mọi loại kiểm thử đều hướng đến việc cải tiến độ tin cậy của chương trình.

Nếu mục tiêu của chương trình chứa các phát biểu đặc biệt về độ tin cậy, ta cũng cần phải thực hiện hoạt động kiểm thử độ tin cậy đặc thù.

Việc kiểm thử các mục tiêu về độ tin cậy có thể khó khăn. Thí dụ, 1 hệ thống online hiện đại như WAN hay ISP thường có thời gian làm việc thực tế bằng 99.97% thời gian sống của nó.

Chưa có cách để ta có thể kiểm thử mục tiêu này với thời gian kiểm thử hàng tháng hay hàng năm.

13. Kiểm thử độ phục hồi sau lỗi (Recovery Test)

Các chươnng trình như hệ điều hành, hệ quản trị database, các chương trình xử lý từ xa thường có các mục tiêu về phục hồi sau lỗi để miêu tả cách hệ thống phục hồi sau khi lỗi dữ liệu, lỗi phần mềm hay lỗi phần cứng xảy ra.

Mục tiêu của kiểm thử độ phục hồi sau lỗi:

- chỉ ra rằng các chức năng phục hồi không làm việc đúng.
- chỉ ra rằng hệ thống không thỏa sự thỏa thuận về thời gian trung bình để phục hồi sau lỗi (MTTR).

14. Kiểm thử tài liệu (Documentation Test)

Kiểm thử hệ thống cũng có liên quan đến độ chính xác của tài liệu dành cho người dùng.

Cách chính yếu để thực hiện điều này là dùng tài liệu để xác định các testcase hệ thống có độ ưu tiên cao.

Tài liệu dành cho người dùng nên là chủ đề của 1 hoạt động thanh tra (tương tự như khái niệm thanh tra mã nguồn), hãy kiểm tra nó để biết được độ chính xác và tính trong sáng.

15. Kiểm thử thủ tục (Procedure Test)

Nhiều phần mềm là thành phần của hệ thống lớn hơn nhưng chưa được tự động hóa hoàn toàn liên quan đến nhiều thủ tục mà con người cần thực hiện.

Bất kỳ thủ tục của con người nào được kê ra, như thủ tục dành cho người quản trị hệ thống, quản trị database, người dùng đầu cuối nên được kiểm thử trong suốt hoạt động kiểm thử hệ thống.

9.4 Kiểm thử độ chấp nhận của user (Acceptance)

là qui trình so sánh chương trình thực tế với các yêu cầu ban đầu của nó và với các nhu cầu hiện hành của người dùng đầu cuối.

Thường được thực hiện bởi khách hàng hay người dùng đầu cuối và thường không được coi như là 1 trách nhiệm của tổ chức phát triển phần mềm.

Trong trường hợp chương trình làm theo hợp đồng, bên đặt hàng thực hiện kiểm thử độ chấp nhận bằng cách so sánh hoạt động của chương trình với các điều khoản trong hợp đồng.

Trong trường hợp chương trình thương mại như HĐH, trình biên dịch, hệ quản trị CSDL, khách hàng nhạy cảm sẽ thực hiện kiểm thử độ chấp nhận để xác định xem sản phẩm có thỏa mãn các yêu cầu của họ không ?

9.5 Kiểm thử việc cài đặt (Installation)

Mục đích của kiểm thử việc cài đặt không phải là tìm lỗi của phần mềm mà là tìm lỗi xảy ra trong quá trình cài đặt phần mềm. Hiện nay, hầu hết các chương trình đều có chương trình cài đặt kèm theo.

Có nhiều sự kiện xảy ra trong quá trình cài đặt hệ thống phần mềm :

- Người dùng phải chọn 1 trong nhiều options.
- Các file và thư viện phải được phân phối và tải về.
- Các cấu hình phần cứng hợp lệ phải có sẵn.
- Chương trình có thể cần nối mạng để giao tiếp với các phần mềm trên các máy khác.

Các testcase có thể kiểm tra để đảm bảo rằng :

- 1 tập các option tương thích nhau đã được chọn.
- tất cả các thành phần của hệ thống phần mềm đã có sẵn.
- tất cả các file đã được tạo ra và có nội dung cần thiết.
- Cấu hình phần cứng phù hợp.

Nên được phát triển bởi đơn vị tạo hệ thống phần mềm, được phân phối như là 1 thành phần của hệ thống phần mềm và chạy sau khi hệ thống được cài đặt.

9.6 Kiểm thử hồi qui (Regression)

Kiểm thử hồi qui là chạy lại các testcase đã có trên TPPM đã được hiệu chỉnh nâng cấp để đảm bảo rằng những thay đổi của TPPM không có ảnh hưởng lề nào, rằng TPPM vẫn đáp ứng tốt với những testcase trước đây.

Thường được bắt đầu khi code mới đang viết và code hiện hành đang được cập nhật.

Cũng được thực hiện từ version này sang version khác phần mềm.

Qui trình lý tưởng là tạo 1 tập testcase bao quát và chạy nó sau mỗi lần có thay đổi phần mềm.

Các kỹ thuật chọn kiểm thử hồi qui :

- Hiệu suất không tiên lượng

- Các giả định về process không tương thích
- Các mô hình đánh giá không thích hợp
- Có thể được tự động hóa

9.6 Kết chương

Chương này đã giới thiệu lý do cần nhiều hoạt động kiểm thử khác nhau, trong đó kiểm thử đơn vị mà ta đã giới thiệu trong các chương trước chỉ là 1 hoạt động kiểm thử đầu tiên.

Chúng ta cũng đã giới thiệu các vấn đề cơ bản liên quan đến các hoạt động kiểm thử khác như kiểm thử chức năng của chương trình, kiểm thử các yêu cầu không chức năng, kiểm thử độ chấp thuận của người dùng, kiểm thử việc cài đặt phần mềm, kiểm thử hồi qui.

Chương 10

Phân tích và giải thích kết quả kiểm thử

10.1 Một số thuật ngữ

Lúc bắt đầu kiểm thử, các testcase đều được ghi nhận là chưa được kiểm thử (unattempted).

Nếu kết quả kiểm thử thỏa mãn đầy đủ kết quả kỳ vọng, testcase sẽ chuyển về trạng thái đã kiểm thử và thành công (attempted and successful).

Nếu chỉ 1 phần kết quả kiểm thử phù hợp với kết quả kỳ vọng, testcase sẽ chuyển về trạng thái đã kiểm thử nhưng chưa thành công (attempted but unsuccessful).

1 trong các công việc chính của quản lý kiểm thử là theo dõi trạng thái của từng testcase vì trạng thái của các testcase là 1 chỉ thị rõ ràng về tiến độ kiểm thử : nếu còn 90% testcase chưa được kiểm thử, ta nói quá trình kiểm thử chỉ mới bắt đầu, nếu chỉ còn 10% testcase chưa được kiểm thử, ta nói quá trình kiểm thử sắp kết thúc.

Ước lượng ban đầu về lịch kiểm thử : Số testcase được kiểm thử trong 1 khoảng thời gian và số người kiểm thử sẽ cho người quản lý biết tiến độ kiểm thử tốt hay quá chậm. Thí dụ 15 testcase được kiểm thử trong 2 tuần đầu (10 ngày làm việc), ta tính được tốc độ kiểm thử là 1.5 testcase/ngày. Nếu kế hoạch cần kiểm thử 100 testcase, ta phải tốn 100/1.5 = 67 ngày (tức 14 tuần).

Nhưng khi kiểm thử một số testcase, ta có thể phát hiện lỗi, do đó ta phải tốn thời gian sửa lỗi và kiểm thử lại testcase đó lần 2, 3, ... Do đó thời gian kiểm thử sẽ lâu hơn nhiều so với ước lượng ban đầu về lịch kiểm thử.

Một số testcase sẽ về trạng thái "được kiểm thử nhưng không thành công" vì kết quả thu được không đúng theo kết quả kỳ vọng hay vì máy bị dừng trước khi hoàn thành kiểm thử testcase đó.

Thách thức cho người quản lý là lập thứ tự ưu tiên cho việc sửa lỗi và kiểm thử lại các testcase này :

- Ứng với testcase làm máy bị dừng khi chưa cho kết quả thì nên ưu tiên cho việc sữa lỗi nó và kiểm thử lại ngay.
- Còn các testcase kiểm thử làm TPPM chạy được nhưng cho kết quả không giống với kỳ vọng thì sẽ lập thứ tự ưu tiên cho việc sửa lỗi. Các tester thường dùng 4 mức 1 tới 4 để đánh giá mức độ cần sửa : mức 1 là tầm trọng nhất và mức 4 là nhẹ nhất và có thể bỏ qua.

Lịch kiểm thử và kết quả tuần đầu

Test case execution schedule					
Date	Test case ID	Date atttempted	Outcome	Severity	Date corrected
Week 1	FT-001	5 Jul	Successful		
	FT-002	6 Jul	Successful		
	FT-003	6 Jul	Successful		
	FT-004	6 Jul	Unsuccessful	1	
	FT-005	7 Jul	Successful		
	FT-006	7 Jul	Successful		
	FT-007	7 Jul	Unsuccessful	2	
	FT-008	7 Jul	Unsuccessful	1	
	FT-009	7 Jul	Unsuccessful	1	
	FT-010	8 Jul	Successful		
Week 2	FT-011				
	FT-012				
	FT-013				
	FT-014				
...	...				
Week 14	FT-095				
	FT-096				
	FT-097				
	FT-098				
	FT-099				
	FT-100				

Kết quả phân tích tuần kiểm thử đầu tiên

Test case execution progress—week 1				
100	Total test cases to attempt			
10	Test cases attempted to date			
10%	Percent attempted to date			
6	Test cases attempted—successful			
60%	Percent test cases successful			
4	Test case attempted—unsuccessful			
40%	Percent test case unsuccessful			
	Severity 1s	3	75%	
	Severity 2s	1	25%	
	Severity 3s	0	0%	
	Severity 4s	0	0%	

10.2 Khám phá lỗi dựa vào các lỗi riêng lẻ tìm được

Người kiểm thử phát hiện từng lỗi theo thời gian. Mỗi khi lỗi được phát hiện, nó có thể được sửa và kiểm thử lại. Vùng code cần kiểm thử có thể chạy được cho toàn bộ testcase.

Nhưng thường gặp hơn là khi kiểm thử lại lỗi vừa sửa, ta phát hiện lỗi khác và phải sửa nó. Điều này có thể lặp lại nhiều lần cho đến khi kiểm thử xong testcase tương ứng.

Cũng có thể ta phải kiểm thử nhiều testcase khác nhau phục vụ cho những mục tiêu khác nhau trên cùng vùng code xác định, và mỗi testcase sẽ giúp phát hiện các lỗi khác nhau.

Tóm lại, việc kiểm thử thành công 1 testcase riêng lẻ chưa đảm bảo vùng code được kiểm thử tương ứng không còn lỗi. Việc kiểm thử/ phát hiện lỗi/sửa lỗi/kiểm thử lại theo kiểu tăng tiến là cách chính yếu để người kiểm thử giúp đở người phát triển viết đúng phần mềm thỏa mãn các yêu cầu đặt ra.

Cách thức và mức độ theo dõi các lỗi tìm được, sửa chữa và kiểm thử lại sẽ quyết định độ hiệu quả của việc kiểm thử. Ta có thể xây dựng bảng theo dõi lỗi bằng worksheet Excel hay bằng 1 tiện ích cao cấp hơn.

Lịch kiểm thử & bảng theo dõi việc xử lý lỗi

Test case execution schedule

Date	Test case ID	Date attempted	Outcome	Severity	Date corrected
Week 1	FT 001	5 Jul	Successful		
	FT 002	6 Jul	Successful		
	FT 003	6 Jul	Successful		
	FT 004	6 Jul	Unsuccessful	1	
	FT 005	7 Jul	Successful		
	FT 006	7 Jul	Successful		
	FT 007	7 Jul	Unsuccessful	2	
	FT 008	7 Jul	Unsuccessful	1	
	FT 009	7 Jul	Unsuccessful	1	
	FT 010	8 Jul	Successful		
Week 2	FT 011				
	FT 012				

Defect tracking log

Defect ID	Date discovered	Test case ID	Severity	Date assigned for correction	Date corrected
SD-0001	6 Jul	FT-004	1		
SD-0002	7 Jul	FT-007	2		
SD-0003	7 Jul	FT-007	2		
SD-0004	7 Jul	FT-007	3		
SD-0005	7 Jul	FT-007	2		
SD-0006	7 Jul	FT-008	1		
SD-0007	7 Jul	FT-009	1		

Mỗi lỗi được gán 1 chỉ số đánh giá mức 1 độ tầm trọng. Mức độ tầm trọng của lỗi phụ thuộc vào loại lỗi và thời điểm lỗi xảy ra ở đâu trong chu kỳ phát triển phần mềm.

Thường ta dùng 3 loại lỗi sau đây :

- loại 1 : lỗi ngăn cản việc kiểm thử.
- loại 2 : lỗi ngăn cản việc phát triển phần mềm.
- loại 3 : lỗi ngăn cản việc triển khai/phân phối phần mềm.

Càng về cuối chu kỳ phát triển phần mềm thì loại lỗi 3 nên có mức độ tầm trọng ngày càng cao.

10.3 Khám phá lỗi dựa vào backlog

Chúng ta đã giới thiệu 2 thước đo hoạt động kiểm thử :

- thước đo 1 : số lượng và tỉ lệ testcase đã thử/testcase trong kế hoạch cho ta biết ta đang ở giai đoạn nào của quá trình kiểm thử.

- thước đo 2 : số lượng và tỉ lệ testcase bị lỗi/testcase đã kiểm thử cho ta biết mức độ tìm lỗi của người kiểm thử.

Bây giờ ta sẽ giới thiệu thước đo thứ 3 : danh sách các lỗi chưa được sửa (backlog).

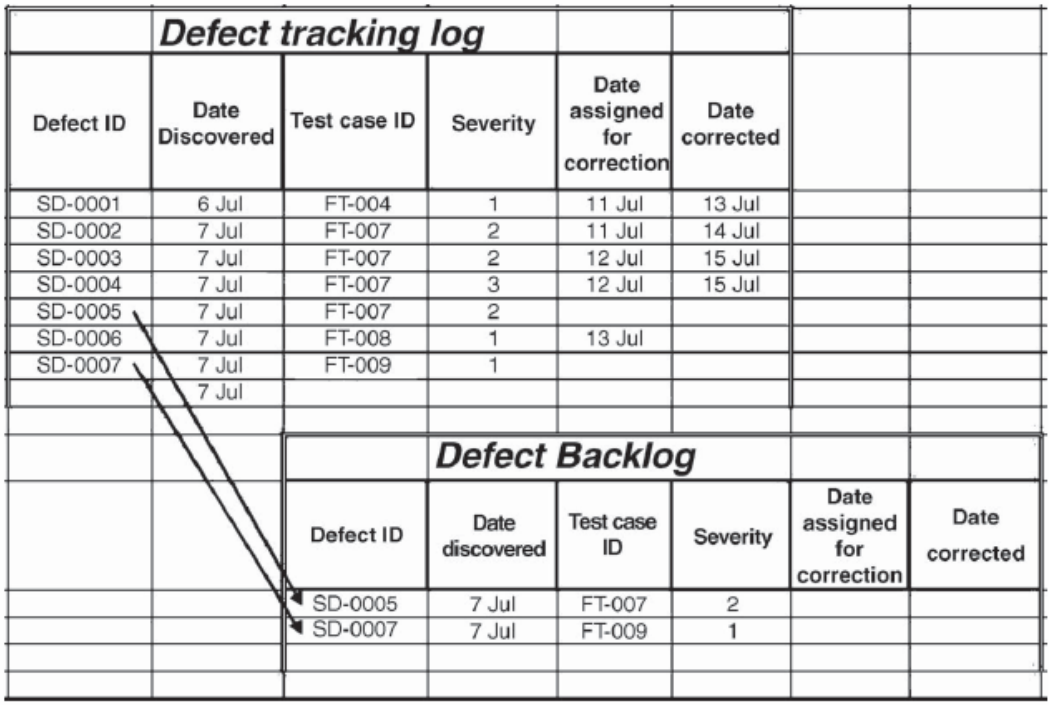
Định nghĩa Danh sách lỗi chưa sửa (Backlog)

Nếu việc kiểm thử đã phát hiện 300 lỗi và người phát triển đã sửa được 100 lỗi thì danh sách các lỗi chưa sửa sẽ chứa 200 lỗi.

Thách thức cho đội phát triển là xác định xem có đủ thời gian, tài nguyên lập trình và kiểm thử để kiểm thử các lỗi còn lại sao cho backlog không còn chứa lỗi nào. Câu trả lời thường là "không thể".

Thách thức kế tiếp là xem lại backlog để lập thứ tự mức độ tầm trọng của lỗi, dựa vào đó các lỗi càng nặng thì nên được sửa đầu tiên. Hy vọng rằng khi thời gian phát triển và kiểm thử đã hết thì backlog sẽ được tối thiểu hóa và chỉ chứa các lỗi không ảnh hưởng đến chức năng nghiệp vụ của chương trình.

Các lỗi trong backlog sẽ được sửa và phân phối trong service pack hay trong version kế tiếp của phần mềm.



Defect tracking log

Defect ID	Date Discovered	Test case ID	Severity	Date assigned for correction	Date corrected
SD-0001	6 Jul	FT-004	1	11 Jul	13 Jul
SD-0002	7 Jul	FT-007	2	11 Jul	14 Jul
SD-0003	7 Jul	FT-007	2	12 Jul	15 Jul
SD-0004	7 Jul	FT-007	3	12 Jul	15 Jul
SD-0005	7 Jul	FT-007	2		
SD-0006	7 Jul	FT-008	1	13 Jul	
SD-0007	7 Jul	FT-009	1		
	7 Jul				

Defect Backlog

Defect ID	Date discovered	Test case ID	Severity	Date assigned for correction	Date corrected
SD-0005	7 Jul	FT-007	2		
SD-0007	7 Jul	FT-009	1		

Một thách thức khác cho đội phân tích backlog là backlog được cập nhật theo chu kỳ (thí dụ hàng tuần), các lỗi mới phát hiện (do việc sửa lỗi trước đây tạo ra) có thể được đánh giá là tầm trọng và được ưu tiên để ở đầu backlog để cần sửa gấp.

Ở điểm nối này có thể xảy ra tranh cải nảy lửa về chất lượng phần mềm vì chưa có chuẩn công nghiệp nào về vấn đề này.

Trong phạm vi của môn học, chúng ta đánh giá chất lượng phần mềm dựa vào những gì phần mềm thực hiện được so với những đặc tả đề ra ban đầu. Cụ thể, nếu đội phát triển phần mềm đã viết được phần mềm thỏa mãn các đặc tả chức năng và đội kiểm thử đã xác thực được điều này thì ta nói phần mềm có chất lượng tốt.

10.4 Khám phá lỗi dựa vào chùm lỗi

Mã earmark của lỗi và công dụng

Bây giờ ta xem xét sự dung hoà của giá/lợi ích mang lại khi thêm 1 field thông tin mới vào từng record theo dõi lỗi trong bảng theo dõi lỗi. Ta gọi field này là mã earmark : mã nhận dạng vùng code chứa lỗi.

Chi phí quản lý field earmark không cao :

- Các thành viên đội phát triển phải thống nhất qui tắc tạo và đọc mã này. Thường trong đặc tả phần mềm, ta dùng 1 mã để nhận dạng 1 module, 1 class, 1 hàm chức năng, mã này cũng có thể dùng làm mã earmark cho lỗi tìm được.
- Các thành viên đội phát triển phải điền giá trị đúng vào field này cho từng lỗi trong bảng theo dõi lỗi mà tester tạo ra (sau khi họ sửa lỗi này).

	Defect tracking log					
Defect ID	**Date discovered**	**Test case ID**	**Severity**	**Date assigned for correction**	**Date corrected**	**code earmark**
SD-0001	6 Jul	FT-004	1	11 Jul	13 Jul	GL106
SD-0002	7 Jul	FT-007	2	11 Jul	14 Jul	AP234
SD-0003	7 Jul	FT-007	2	12 Jul	15 Jul	AP234
SD-0004	7 Jul	FT-007	3	12 Jul	15 Jul	AP236
SD-0005	7 Jul	FT-007	2			
SD-0006	7 Jul	FT-008	1	13 Jul		
SD-0007	7 Jul	FT-009	1			

Phương pháp phân tích nguyên nhân gốc (root cause analysis) là 1 phương pháp toán học trong lĩnh vực thống kê được trình bày khá phổ biến trong các sách về toán thống kê. Người ta đã dùng khá nhiều phương pháp này để phân tích danh sách lỗi phát hiện của dự án phần mềm.

Ý tưởng cơ bản của phương pháp này trong phân tích lỗi phát hiện là giúp tìm kiếm và xác định các chùm lỗi lớn nhất (buggiest) :

- tính số lượng lỗi/tỉ lệ lỗi theo từng mã earcode.
- lập thứ tự từ cao xuống thấp.

Phương pháp này rất hữu dụng trong khoảng thời gian từ 1/3 tới 1/2 thời gian trong kế hoạch kiểm thử vì lúc này phần mềm có độ ổn định rất thấp, nên cần nhiều sửa chữa và cập nhật.

Kết quả phân tích nguyên nhân gốc của 1 dự án phần mềm trong giai đoạn đầu của công đoạn hiện thực, lúc này đội phát triển đã sửa được 2000 lỗi.

Defect tracking log

Root cause analysys of defects

Corrected during preliminary construction

May 1 thru Aug 15

2000 total defects corrected during preliminary construction

Code earmarks	Defects corrected per code earmark	Percent defects corrected per code earmark
AP234	450	22.5%
AP745	310	15.5%
GL106	150	7.5%
AR218	75	3.8%
PY512	10	0.5%
BK459	8	0.4%
DP113	8	0.4%
All others	989	less than 0.4%
Total	2000	

Phân tích số liệu của bảng trong slide trước, ta thấy :

- cần thiết phải xem lại các đoạn code và bảng thiết kế liên quan gây ra 3 nhóm lỗi AP234, AP745 và GL106 (vì chúng chiếm tới 45.5% lỗi) trước khi tiến hành bước hiện thực kế tiếp. Lưu ý thêm là 2 nhóm lỗi AP234, AP745 có mối quan hệ rất khắng khích vì chúng thuộc cùng 1 module chức năng.
- còn các lỗi thuộc nhóm AR218 có tỉ lệ tương đối thấp nên có thể được xem xét hay không tùy thuộc vào đội kiểm thử đã tìm hiểu và nắm vững về vùng code gây lỗi chưa.
- Có thể bỏ qua việc xem lại vùng code chứa các lỗi thuộc các nhóm còn lại vì chúng chiếm tỉ trọng quá nhỏ.

Sở dĩ xuất hiện các nhóm lỗi chứa nhiều lỗi là vì :

- thiết kế không chính xác với chức năng hoặc thiết kế chưa đầy đủ.
- viết code chưa đủ.
- debug code chưa đủ.
- dùng chuẩn lập trình không tốt.
- người lập trình chưa thông thạo và nắm vững khả năng của ngôn ngữ lập trình.
- người lập trình chưa nắm vững giải thuật phức tạp được dùng để giải quyết chức năng.

Việc xem lại vùng code chứa nhiều lỗi cũng tốn thời gian (thí dụ 3 tuần) và làm cho bước kế tiếp bị delay, tuy nhiên cái lợi thì lớn hơn nhiều :

- trong việc xem lại vùng code/bảng thiết kế, ta có thể tìm ra được 1 số nguyên nhân gốc gây ra đồng thời nhiều lỗi, chỉ cần sửa và cập nhật các nguyên nhân gốc này thì rất nhiều lỗi đã được sửa.
- và khi kiểm thử lại, số lỗi thuộc các vùng code liên quan sẽ giảm thiểu rất lớn. Thí dụ ta chỉ còn phát hiện 50 lỗi (so với 1500 lỗi của lần kiểm thử trước). các lỗi tìm được này có xu hướng không phải là lỗi nặng và có thể được đưa vào danh sách lỗi chưa sửa nên không cần sửa gắp chúng, có thể để cho bước phát triển sau đó thực hiện.

Ta có thể lặp lại việc phân tích nguyên nhân gốc của 1 dự án phần mềm trong các giai đoạn sau đó, thí dụ tại giai đoạn cuối của phát triển phần mềm, đội phát triển đã sửa được 1500 lỗi.

Defect tracking log

Root cause analysys of defects

Corrected during Final construction in progress

Sept 15 – Nov 15

1500 total defects corrected during Final construction in progress

Code earmarks	Defects corrected per code earmark	Percent defects corrected per code earmark
AR218	30	2.0%
PY512	25	1.7%
BK459	15	1.0%
DP113	8	0.5%
AP234	5	0.3%
GL106	4	0.3%
All others	1413	less than 0.3%
Total	1500	

Bây giờ ta thấy :

- 3 nhóm lỗi quan trọng nhất chỉ chiếm tỉ lệ rất nhỏ (4.7%), điều này nói lên phần mềm đã khá ổn định.
- các nhóm lỗi quan trọng của bước trước là AP234 và GL106 vẫn còn trong danh sách, nhưng với tỉ lệ rất nhỏ. Điều này cho thấy tính hiệu quả của việc dùng mã earmark và việc xem xét lại vùng code tương ứng.
- do các nhóm lỗi quan trọng nhất chỉ chiếm tỉ lệ rất nhỏ nên việc phân tích nguyên nhân gốc ở bước này là không có tác dụng lớn như lúc trước nữa. Mặc dù vậy, thông tin earmark vẫn hữu dụng cho việc quản lý danh sách lỗi chưa sửa (backlog).

Defect backlog		
Root cause analysys of defects		
Not corrected as of Dec 15		
1 Total defect—severity 1	Included in analysis	
24 Total defects—severity 2	Included in analysis	
475 Total defects—severity 3 and 4	Not included in analysis	
500 Total defects not corrected as of Dec 15		
Code earmark guesses	**Defects awaiting correction per code earmark**	**Percent defects awaiting correction per code earmark**
AR477 (includes severity 1)	13	52.0%
GL105	5	20.0%
PY632	3	12.0%
GL498	2	8.0%
GL551	1	4.0%
GL600	1	4.0%
Total	25	

10.5 Dùng mẫu về phát hiện lỗi của project trước

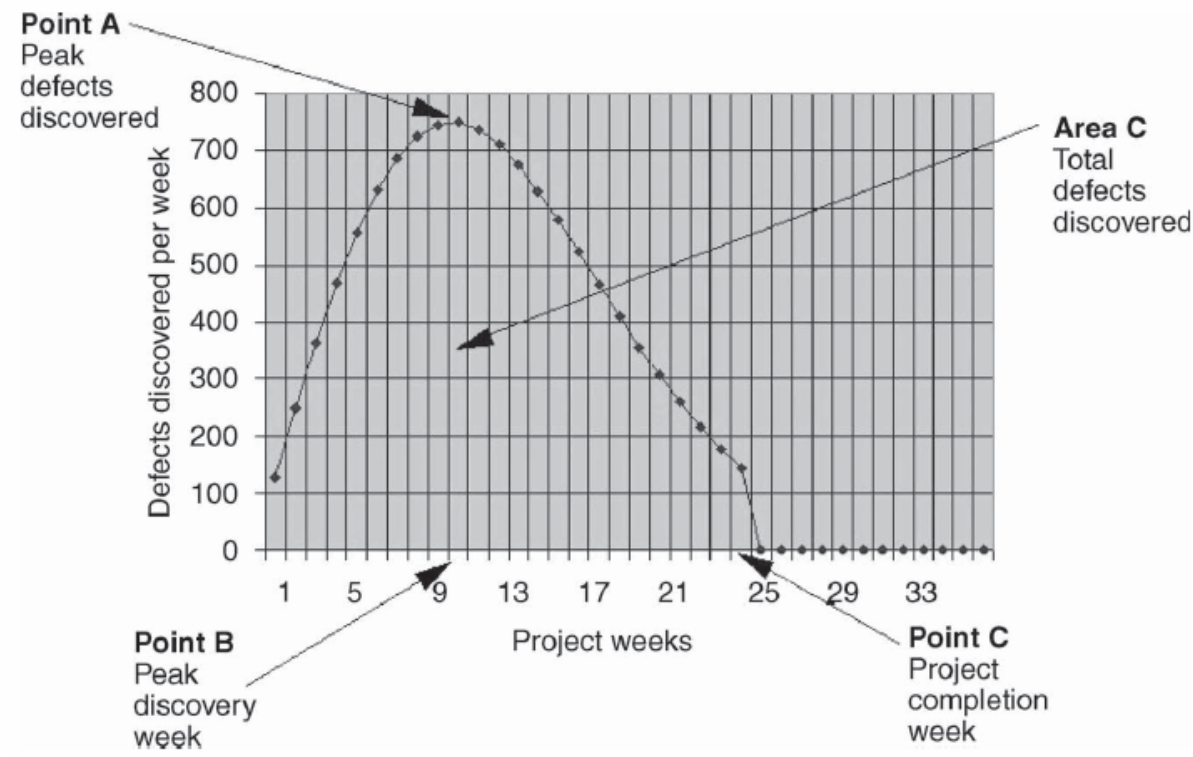
Việc tiên đoán số lỗi mà đội kiểm thử sẽ phát hiện được trong dự án hiện hành là rất quan trọng, nó giúp ta đưa ra kế hoạch, chuẩn bị tài nguyên kiểm thử hiệu quả. Nếu không dựa trên thông tin nào, hiện nay chưa có phương pháp nào giúp ta tiên đoán chính xác được.

Tuy nhiên, có 1 số phương pháp dựa vào yếu tố lịch sử sẽ giúp ta tiên đoán với độ chính xác nằm trong phạm vi lệch 10%.

Phương pháp tiên đoán dựa vào yếu tố lịch sử đặc biệt thích hợp cho các công ty phần mềm lâu đời, họ đã phát triển thành công nhiều phần mềm với nhiều kích cở khác nhau theo thời gian.

Càng có nhiều thông tin trong record miêu tả lỗi trong project trước càng cho ta tiên đoán với độ chính xác cao hơn. Chúng ta hãy bắt đầu bởi 2 thông tin cơ bản trong record lỗi : mã lỗi và ngày phát hiện.

Đường biểu diễn lỗi phát hiện trong project 200KLOC, được kiểm thử trong 24 tuần.



Đường cong miêu tả việc phát hiện lỗi của project trước cho chúng ta những thông tin chung cho hầu hết các project phần mềm :

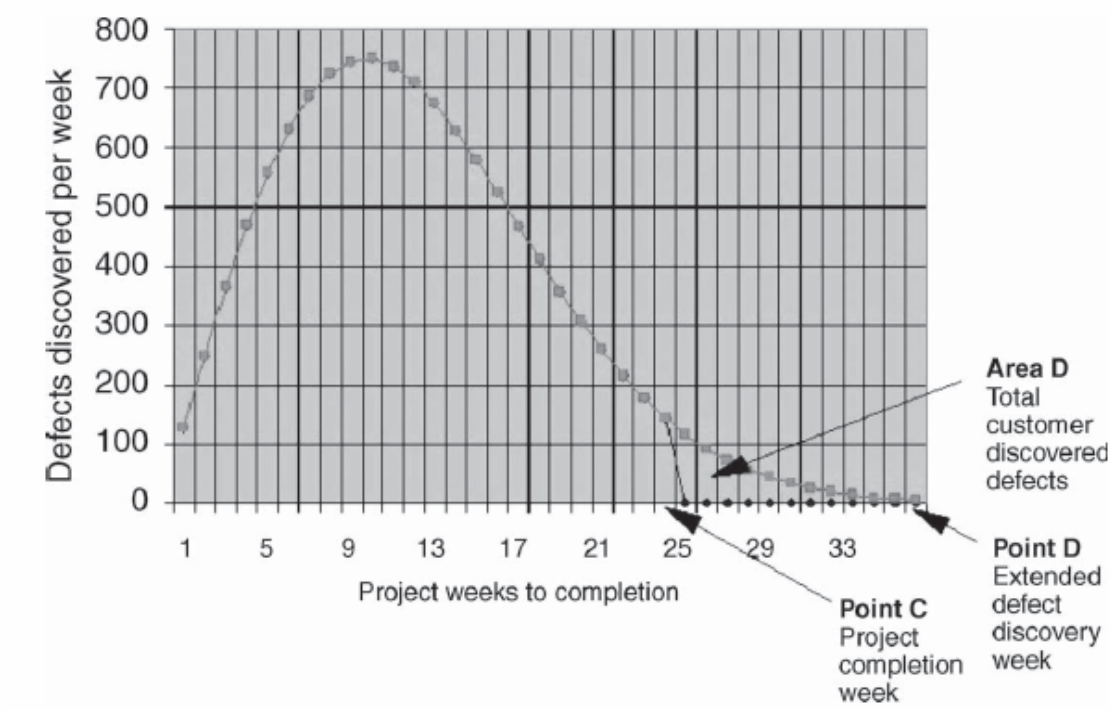
- lúc đầu số lỗi phát hiện được thường rất nhiều và lên nhanh tới đỉnh ở thời điểm khoảng 1/3 thời gian phát triển phần mềm (giai đoạn đầu này độ ổn định của phần mềm còn rất thấp).
- sau đó số lỗi được phát hiện giảm dần khi phần mềm ngày càng hoàn chỉnh và ổn định.
- điểm A miêu tả số lỗi được phát hiện nhiều nhất (là 749 lỗi).
- điểm B miêu tả tuần phát hiện lỗi nhiều nhất (là tuần 10).
- điểm C miêu tả tuần kết thúc việc phát triển phần mềm (tuần 24).

- vùng C dưới đường cong (diện tích của vùng) miêu tả tổng số lỗi tìm được trong dự án phần mềm (11,497).

Có nhiều nghiên cứu nói rằng : nếu ta tìm được lỗi càng nhiều và càng sớm thì lỗi được phát hiện bởi khách hàng sau này càng ít. Lưu ý rằng chi phí sửa lỗi do khách hàng phát hiện được là rất lớn.

Nghiên cứu này còn nói là nếu ta nội suy đường biểu diễn lỗi phát hiện về phía phải trục x đến khi tiếp xúc trục x thì vùng nới rộng này (vùng D) sẽ sắp xỉ bằng số lỗi mà khách hàng sẽ phát hiện được sau đó.

Biểu đồ nới rộng được vẽ ở slide kế cho ta thấy điểm D tiếp xúc với trục x ở tuần 36, số lỗi ở vùng D là 487 lỗi.



10.6 Sử dụng biểu đồ phân loại các nhóm lỗi

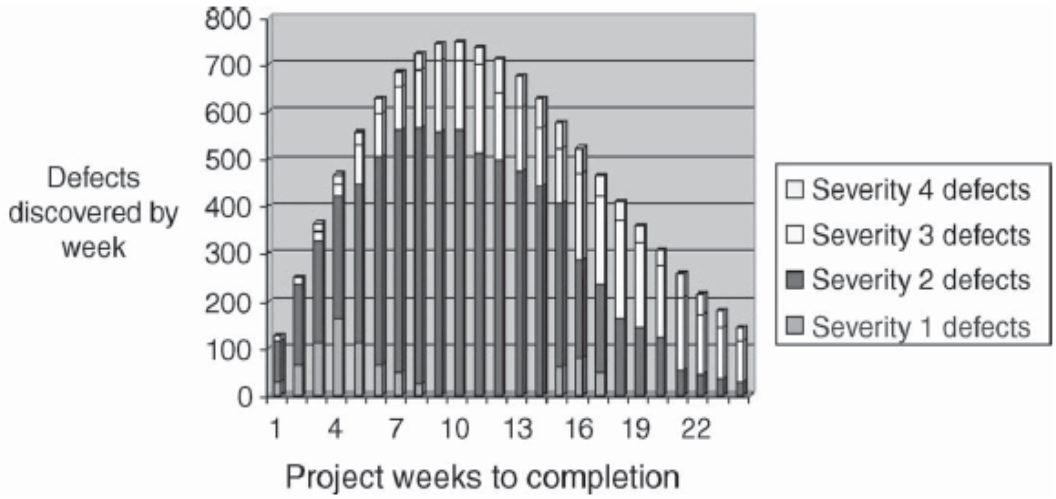
Nếu trong record thông tin về lỗi được phát hiện trong project trước có thêm field miêu tả mức độ tầm trọng của lỗi (1-4), ta có thể xây dựng và sử dụng biểu đồ phân loại các nhóm lỗi như sau :

- Từng chu kỳ (tuần), tính số lượng lỗi thuộc từng mức nặng/nhẹ phát hiện được (ta dùng 4 mức).

- Xếp tầng số lỗi thuộc từng mức nặng/nhẹ trong tuần và vẽ thành 1 cột cho tuần đó.

Dựa vào kết quả của biểu đồ ta rút ra được các kết luận :

- Các lỗi có mức độ càng nặng thường xảy ra và được phát hiện rất sớm vì lúc này project chưa có độ ổn định cao.
- Mức độ lỗi phát hiện giảm dần về sau khi project ngày càng ổn định hơn.



10.7 Độ tương quan về lỗi do khách hàng phát hiện

Trong khoảng thời gian từ lúc project trước đã hoàn thành đến lúc project sau được phát triển, ta có thêm nhiều thông tin khác. Một trong các thông tin quan trọng là số lượng lỗi và tính chất lỗi được phát hiện và phản hồi về bộ phận HelpDesk bởi khách hàng.

Ta dùng độ tương quan về lỗi do khách hàng phát hiện như sau :

- Tỉ lệ lỗi tiên đoán/lỗi được khách hàng phát hiện thực tế
- 487/70 = 6.9

Nhờ độ tương quan này mà ta có thể tiên đoán được số lượng lỗi mà khách hàng phát hiện được khi sử dụng project phần mềm sắp phát triển. Dựa vào số lượng lỗi này mà ta qui hoạch tương đối chính xác số lượng và thời lượng làm việc của các nhân viên trong

2 bộ phận hỗ trợ (Support) và giúp đỡ (HelpDesk) người dùng ⇒ dự toán chính xác chi phí sửa lỗi.

Bảng theo dõi các lỗi được phản ánh bởi khách hàng do bộ phận giúp đở khách hàng ghi nhận và bộ phận phát triển phân loại nhóm lỗi.

HelpDesk tracking log

Root cause analysys of defects

Corrected after prior project completed before next project started

2 Total defect—severity 1	Included in analysis
13 Total defect—severity 2	Included in analysis
55 Total defect—severity 3 & 4	Not included in analysis
-----	
70 Total defects discovered by customers	

Code earmarks	Defects corrected per code earmark	Percent defects corrected per code earmark
AR477 (includes 2 Severity 1s)	7	46.7%
GL431	3	20.0%
DP268	2	13.3%
AP365	1	6.7%
BK663	1	6.7%
PY315	1	6.7%
Total	15	

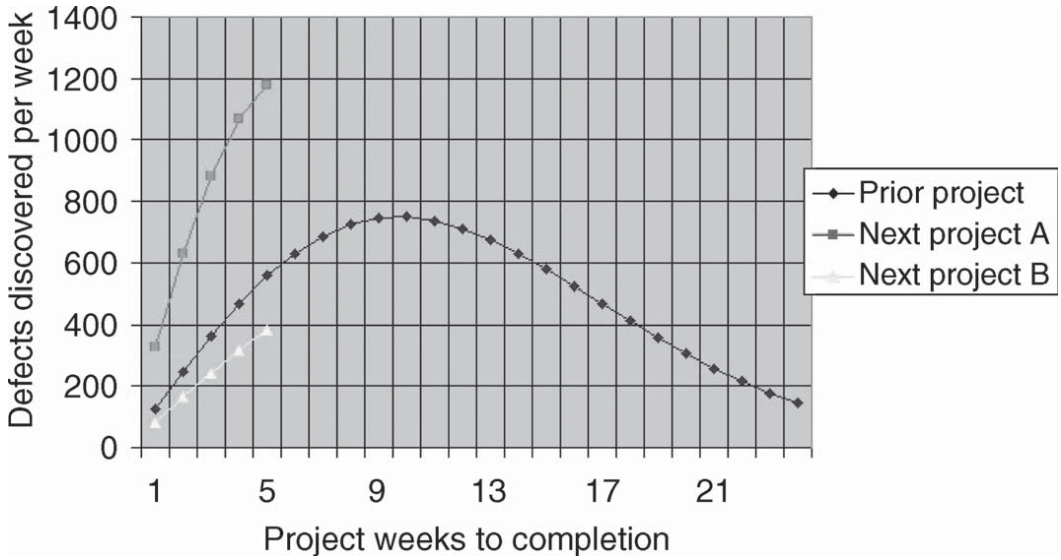
Bắt đầu project mới

Lúc này ta đã có :

- số lượng hàng lệnh và số lỗi của project trước (được thể hiện trong đường cong miêu tả số lỗi phát hiện được).
- mức độ tầm trọng của các lỗi và cách phân phối chúng theo thời gian (được thể hiện trong biểu đồ phân phối mức độ lỗi của project trước).

So sánh với số lượng hàng lệnh của project đang phát triển, ta có thể tính số lượng lỗi cho project mới này, từ đó kế hoạch hóa việc kiểm thử, chuẩn bị tài nguyên phù hợp cho việc kiểm thử.

Trong từng chu kỳ kiểm thử, ta tổng kết và vẽ số lượng lỗi phát hiện trực tiếp lên đường biểu diễn của project trước để dễ dàng so sánh và nội suy thông tin khi cần.



Do các project có qui mô và tính chất khác nhau, số lượng lỗi phát hiện trong tuần đầu sẽ khác nhau.

Thí dụ Project hiện hành là A có số lỗi phát hiện được nhiều hơn project cũ và tốc độ phát hiện lỗi trong các tuần đầu cao hơn nhiều so với project cũ. Đây là xu thế mà ta mong muốn. Điều này có thể đặt ra câu hỏi : tại sao ta kiểm thử hiệu quả hơn trước ? Câu trả lời có thể là do ta dùng nhiều tools kiểm thử tự động hơn trước.

Thí dụ Project hiện hành là B có số lỗi phát hiện được ít hơn project cũ và tốc độ phát hiện lỗi cũng chậm hơn so với project cũ. Đây là xu thế mà ta không mong muốn. Điều này có thể đặt ra câu hỏi : tại sao ta kiểm thử kém hiệu quả hơn trước ? Câu trả lời có thể là do ta bỏ đi hay làm thiếu 1 số hoạt động mà project trước đã làm. Sau khi xác định được nguyên nhân rõ ràng, ta sẽ khắc phục để việc kiểm thử sẽ hiệu quả hơn cho các tuần còn lại.

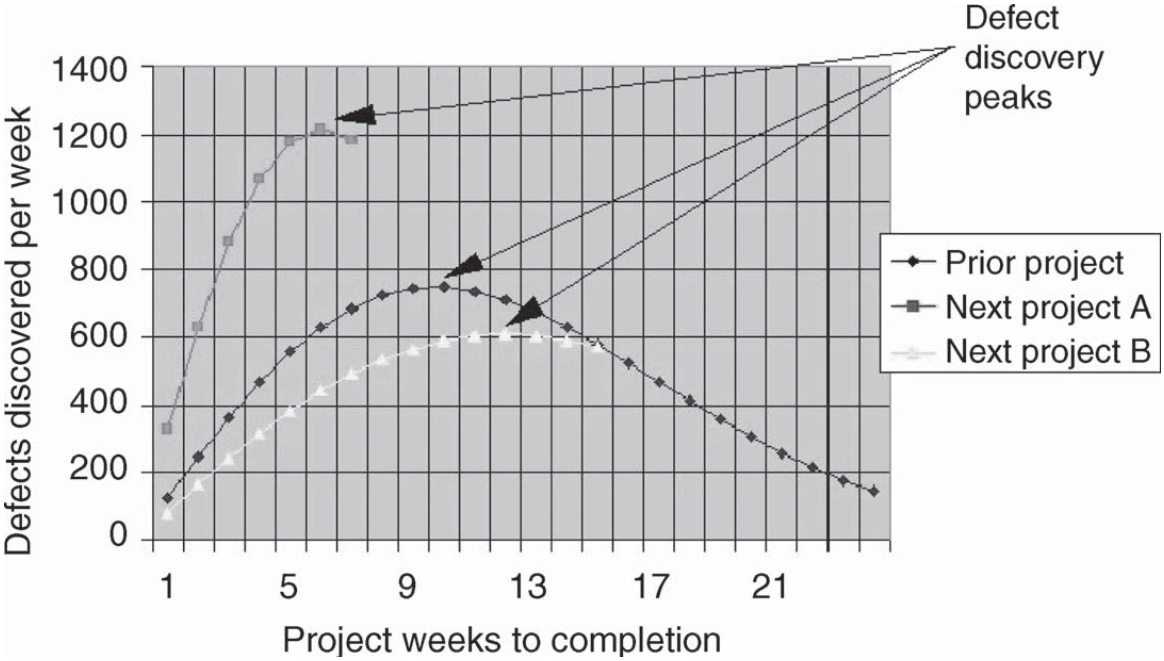
Khi project mới tiếp tục

Tiếp tục kiểm soát và vẽ đường cong biểu diễn lỗi phát hiện cho tới cột mốc kế tiếp trong quá trình phát triển phần mềm. Tới 1 lúc nào đó, thường là khoảng 1/3 thời gian phát triển phần mềm,

tốc độ phát hiện lỗi (số lỗi phát hiện/trên tuần) sẽ đạt ngưỡng và bắt đầu giảm xuống.

Điểm uốn này là điểm neo quan trọng nhất cho hầu hết mọi hoạt động phân tích kết quả về lỗi :

- Project A có điểm ngưởng ở tuần 6 với số lượng 1213 lỗi ⇒ Project A được kiểm thử hiệu quả hơn project cũ.
- Project B có điểm ngưỡng ở tuần 12 với số lượng 607 lỗi ⇒ Project B được kiểm thử kém hiệu quả hơn project cũ và project A.



Khi project mới kết thúc

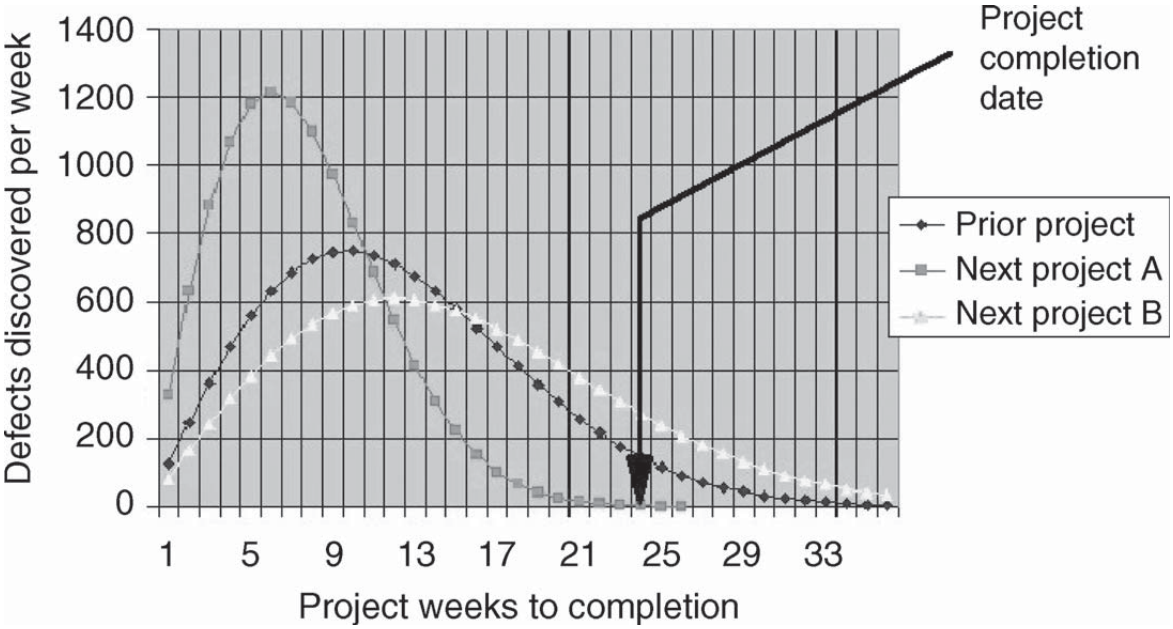
Tiếp tục kiểm thử và vẽ kết quả theo thời gian. Mỗi lần vẽ 1 điểm mới vào đồ thị, ta phân tích điểm này có bất thường gì không ? Nếu có hãy đặt câu hỏi tạo sao và từ đó có biện pháp khắc phục.

Khi project mới kết thúc (giả sử theo kế hoạch là tuần 24 như trong biểu đồ), ta nội suy và nới rộng đồ thị sang phải cho đến khi tiếp xúc trục x. Một số tiên đoán và ước lượng :

- Project A chứa 14 lỗi được tiên đoán là khách hàng sẽ tìm được nên sẽ có 2 lỗi mà khách hàng thực sự tìm ra → chi

phí sửa là 2*14000 = 28000$ (giảm được 952000$ so với project trước).

- Project B chứa 1377 lỗi được tiên đoán là khách hàng sẽ tìm được nên sẽ có 196 lỗi mà khách hàng thực sự tìm ra → chi phí sửa là 196*14000 = 2744000$ (tăng hơn 1746000$ so với project trước).

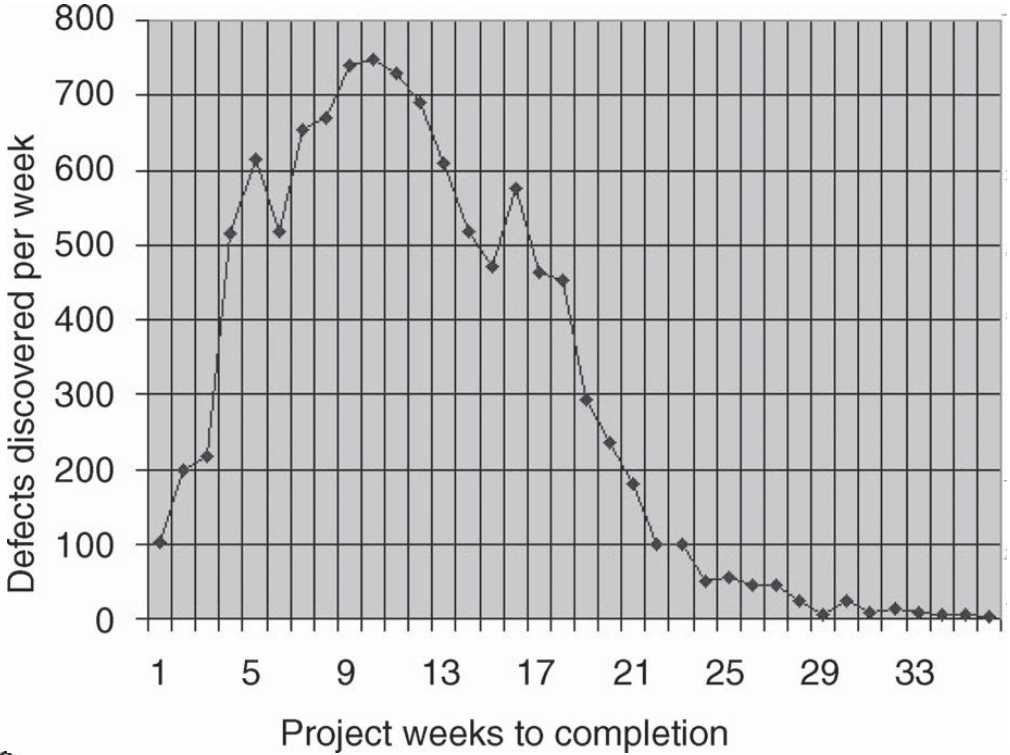


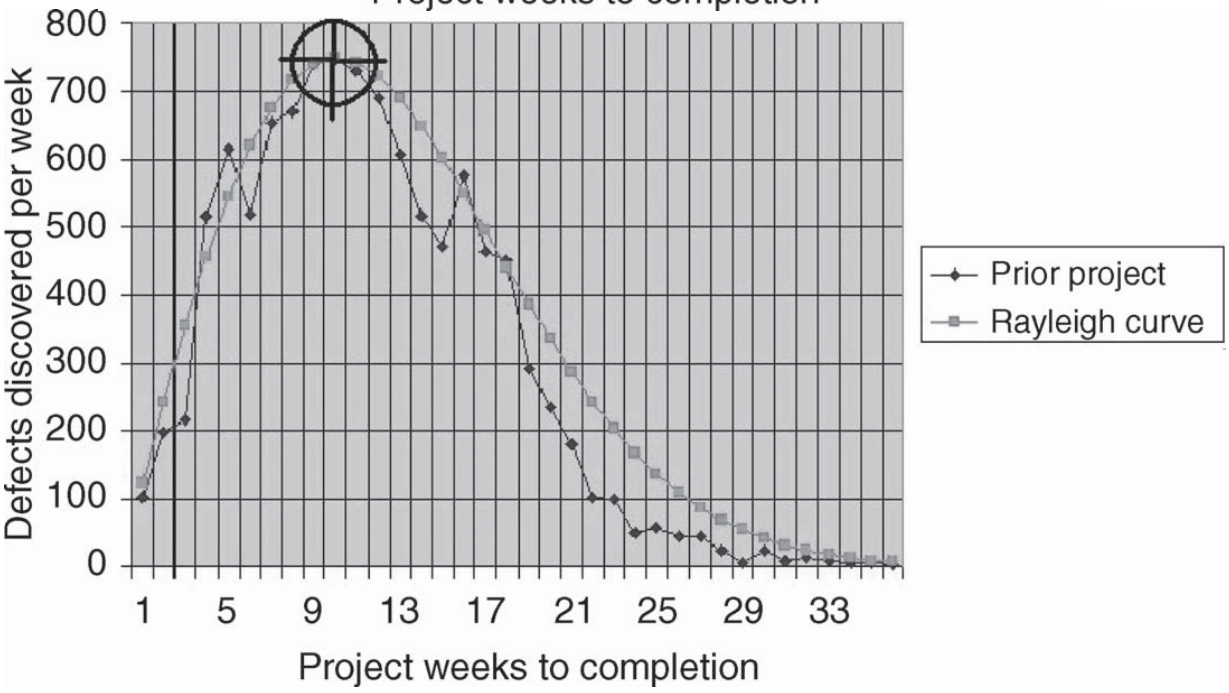
10.8 Đường cong Rayleight - Gunsight cho mẫu phát hiện lỗi

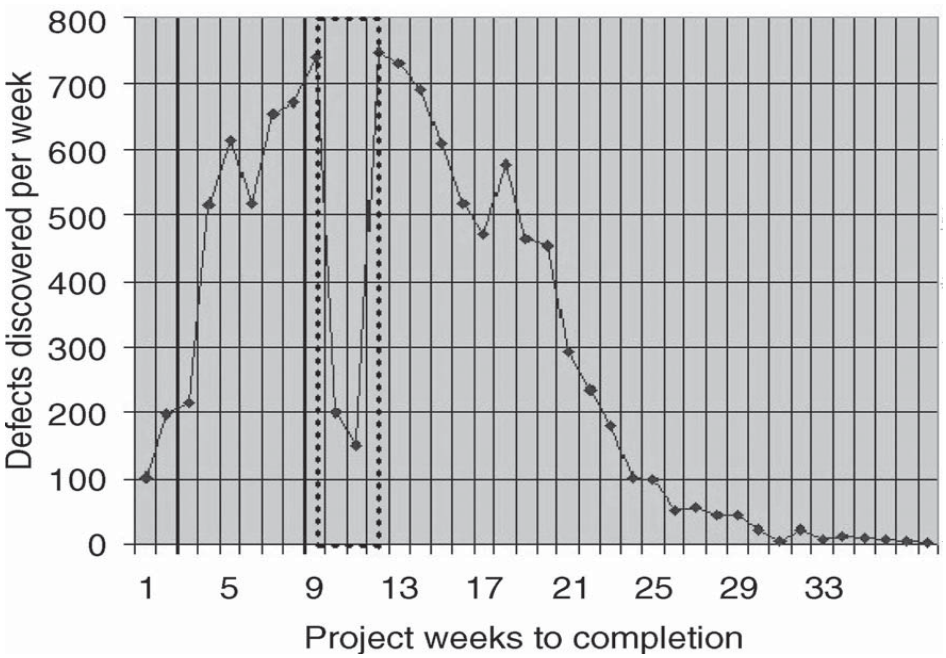
Làm sao biết được tốc độ phát hiện lỗi trong các project công nghiệp của các hãng lớn hiện nay ? Nhờ đó so sánh, đánh giá hoạt động kiểm thử trong project của mình ?

Rất tiếc, ta khó lòng có được các thông tin xác thực này vì các công ty phần mềm có khuynh hướng dấu nhẹm, không công bố (vì nó ảnh hưởng lớn đến hình ảnh công ty).

Hiện nay nguồn cung cấp chủ yếu là ứng dụng đường cong Rayleight. Đây là 1 công thức toán học cho phép ta tái tạo đường cong chuẩn dựa vào điểm ngưỡng (gunsight). Đường cong chuẩn này sai lệch so với giá trị thật của nhiều dự án phần mềm trong 5 năm gần đây dưới 10%.


800
700
600
500
400
300
200
100
0
Defects discovered per week
1
5
9
13
17
21
25
29
33
Project weeks to completion


800
700
600
500
400
300
200
100
0
Defects discovered per week
1
5
9
13
17
21
25
29
33
Project weeks to completion
Prior project
Rayleigh curve



10.9 Các thước đo khác về sự theo dõi lỗi

Các thước đo của hoạt động phát triển phần mềm :

- mã lỗi phát triển duy nhất.
- ngày phát hiện lỗi phát triển.
- mức độ trầm trọng của lỗi phát triển.
- ngày sửa lỗi phát triển.
- mã earcode của lỗi phát triển.

Các thước đo của hoạt động HelpDesk :

- mã lỗi khách hàng duy nhất.
- ngày phát hiện lỗi khách hàng.
- mức độ trầm trọng của lỗi khách hàng.
- ngày sửa lỗi khách hàng.
- mã earcode của lỗi khách hàng.

Thước đo khác : ODC (Orthogonal Defect Classifi cation) của hãng IBM.

10.10 Kết chương

Chương này đã giới thiệu 1 số thuật ngữ được dùng trong hoạt động quản lý quá trình kiểm thử phần mềm.

Chúng ta cũng đã giới thiệu 1 số kỹ thuật phổ dụng để hỗ trợ việc phát hiện lỗi như phát hiện lỗi mới dựa vào lỗi đã phát hiện được, phát hiện lỗi mới dựa vào backlog, phát hiện lỗi mới dựa vào chùm lỗi...

Chúng ta cũng đã giới thiệu 1 số kỹ thuật để quản lý hoạt động kiểm thử phần mềm như dùng đường cong phát hiện lỗi của các project trước, dùng biểu đồ phân loại các nhóm lỗi của các project trước, dùng đường cong Rayleight...