



# **Đảm bảo chất lượng phần mềm**

**Biên tập bởi:**

Khoa CNTT ĐHSP KT Hưng Yên

# **Đảm bảo chất lượng phần mềm**

**Biên tập bởi:**

Khoa CNTT ĐHSP KT Hưng Yên

**Các tác giả:**

Khoa CNTT ĐHSP KT Hưng Yên

Phiên bản trực tuyến:

<http://voer.edu.vn/c/4c771e16>

# MỤC LỤC

1. Bài 1 : Cơ bản về kiểm thử phần mềm(Software Testing)
  - 1.1. Những lỗi (bug) phần mềm nghiêm trọng trong lịch sử
  - 1.2. Lỗi (bug) là gì
  - 1.3. Tại sao lỗi xuất hiện
2. Bài 2 : Quy trình phát triển phần mềm
  - 2.1. Quy trình phát triển phần mềm
  - 2.2. Thực trạng của quá trình kiểm thử phần mềm
    - 2.2.1. Thực trạng của quá trình kiểm thử phần mềm
    - 2.2.2. Các định nghĩa và thuật ngữ kiểm thử phần mềm
  - 2.3. Quá trình nghiên cứu bản đặc tả phần mềm
3. Bài 3 : Các phương pháp kiểm thử
  - 3.1. Phương pháp kiểm thử
  - 3.2. Thiết kế trường hợp kiểm thử
  - 3.3. Chiến lược kiểm thử
4. Bài 4 : Các kỹ thuật kiểm thử
  - 4.1. Test và kiểm tra
  - 4.2. Kiểm tra miền
5. Bài 5 : Các vấn đề cần kiểm thử
  - 5.1. Kiểm thử cấu hình
    - 5.1.1. Kiểm thử cấu hình
    - 5.1.2. Cấu Hình Hardware
  - 5.2. Kiểm thử khả năng tương thích
  - 5.3. Kiểm thử Foreign – Language
  - 5.4. Kiểm thử khả năng tiện dụng (tính khả dụng)
  - 5.5. Kiểm thử tài liệu
  - 5.6. Kiểm thử khả năng bảo mật phần mềm
6. Bài 6 : Các giai đoạn kiểm thử
  - 6.1. Các giai đoạn kiểm thử
7. Bài 7 : Tổng quan quy trình kiểm tra hệ thống phần mềm
  - 7.1. Mô hình kiểm tra phần mềm
    - 7.1.1. Mô hình kiểm tra phần mềm TMM
    - 7.1.2. Quy trình kiểm tra
  - 7.2. Mô tả

- 7.3. Lập kế hoạch
  - 7.4. Các yêu cầu test
  - 7.5. Một ví dụ
  - 7.6. Test
  - 8. Bài 8 : Viết và theo dõi các trường hợp kiểm thử
    - 8.1. Viết và theo dõi các trường hợp kiểm thử
  - 9. Bài 9 : Thực hiện test,viết báo cáo và đánh giá kết quả
    - 9.1. Thực hiện test, viết báo cáo và đánh giá kết quả
  - 10. Bài 10 : Kiểm thử tự động và các công cụ kiểm thử
    - 10.1. Lợi ích của quá trình tự động hóa và các công cụ
    - 10.2. Một số công cụ kiểm thử
      - 10.2.1. Giới thiệu về công cụ KTTĐ
      - 10.2.2. Chương Trình TestTrack Pro Version 6.0.3
      - 10.2.3. Kiểm tra hiệu năng phần mềm với LoadRunner 8.1
      - 10.2.4. Esstimate
    - 10.3. Kiểm thử tự động với phần mềm
  - 11. Bài 11 : Thảo luận về kiểm thử hướng đối tượng
    - 11.1. Thảo luận và kiểm thử hướng đối tượng
  - 12. Bài 12 : Bài tập
    - 12.1. Bài Tập
- Tham gia đóng góp

# Bài 1 : Cơ bản về kiểm thử phần mềm(Software Testing)

## Những lỗi (bug) phần mềm nghiêm trọng trong lịch sử

- Hãy đánh giá thử xem các phần mềm đã thâm nhập vào cuộc sống của chúng ta như thế nào.
  - Sau năm 1947, chiếc máy tính *Mark II* yêu cầu hàng tá những nhà lập trình phải bảo trì liên miên. Những người bình thường không bao giờ tưởng tượng được rằng một ngày nào đó trong căn nhà của họ sẽ có một chiếc máy tính của chính họ.
  - Bây giờ, máy tính tràn ngập khắp nơi, nó không chỉ đến với từng gia đình, mà còn đến với từng cá nhân. Những đĩa CD phần mềm miễn phí với các đoạn video game cho trẻ em, tặng kèm theo các hộp ngũ cốc còn nhiều hơn cả phần mềm trên các tàu con thoi.
- Hãy thử so sánh sự phát triển của các máy nhắn tin và các buồng điện thoại, dịch vụ chuyển phát nhanh... với sự phát triển của máy tính và phần mềm máy tính. Dường như không gì có thể theo kịp sự bùng nổ của ngành công nghiệp đầy chất xám này. Bây giờ, chúng ta có thể không sử dụng không sử dụng các dịch vụ chuyển phát nhanh..., nhưng không thể bắt đầu một ngày mà không vào mạng và kiểm tra thư điện tử.
- Phần mềm ở khắp mọi nơi. Tuy nhiên, nó được viết bởi nhiều người, vì vậy mà nó không hoàn hảo. Chúng ta hãy cùng đi tìm hiểu một số ví dụ dưới đây:

### Disney's Lion King, 1994 – 1995

Vào cuối năm 1994, công ty Disney đã tung ra thị trường trò chơi đa phương tiện đầu tiên cho trẻ em, *The Lion King Animated StoryBook*. Mặc dù rất nhiều công ty khác đã quảng bá các chương trình cho trẻ em trong nhiều năm, đây là lần đầu tiên Disney mạo hiểm lao vào thị trường. Nó đã được xúc tiến và quảng cáo mạnh mẽ. Số lượng bán ra vô cùng đồ sộ. Nó được mệnh danh là “*the game to buy*” cho trẻ em trong kỳ nghỉ. Tuy nhiên, chuyện gì đã xảy đến? Đó là một sự thất bại khủng khiếp. Vào 26/12, ngay sau ngày Giáng Sinh, khách hàng của Disney đã liên tục gọi điện. Ngay lập tức, các kỹ thuật viên trợ giúp bằng điện thoại đã bị sa lầy với các cuộc gọi từ các bậc cha mẹ đang giận dữ và những đứa trẻ đang khóc, vì chúng không thể cho phần mềm làm việc. Nhiều câu chuyện đã xuất hiện trên các mặt báo và trên bản tin của TV.

...Disney đã thất bại khi không kiểm tra phần mềm rộng rãi trên nhiều mô hình máy tính khác nhau có sẵn trên thị trường. Phần mềm đã làm việc trên một vài hệ thống mà

các các lập trình viên của Disney đã dùng để tạo ra trò game này, nhưng nó không phải là các hệ thống phổ biến nhất mà người dùng hay sử dụng.

Lỗi chia dấu phẩy động của bộ vi xử lý Intel Pentium (Intel Pentium Floating – Point Division Bug), 1994

Hãy mở phần mềm Calculator trong máy tính của bạn và thực hiện phép toán sau:

$(4195835 / 3145727) * 3145727 - 4195835$

Nếu kết quả là 0, máy tính của bạn hoạt động tốt. Nếu như bạn nhận được một kết quả khác, thì bạn đang sở hữu một Intel Pentium CPU với lỗi **floating – point division** (chia dấu phẩy động) – một lỗi phần mềm đã làm nóng chip của bạn mà vẫn được tái sản xuất liên tục.

Ngày 30/10/1994, **Thomas R. Nicely** thuộc trường cao đẳng **Lynchburg (Virginia)** đã phát hiện một kết quả không mong muốn trong khi thực hiện phép chia (**division**) trên máy tính của ông. Ông đã công bố kết quả nghiên cứu của mình trên internet và ngay lập tức ông đã làm bùng lên ngọn lửa với một số lượng lớn những người cũng gặp vấn đề như ông. Và họ tìm thêm những tình huống máy tính đưa ra câu trả lời sai. May thay những trường hợp này là hiếm thấy và kết quả đưa ra câu trả lời sai chỉ trong những trường hợp phục vụ cho Toán học chuyên sâu, Khoa học, và các Tính toán kỹ thuật. Hầu hết mọi người sẽ không bao giờ bắt gặp chúng trong khi đang thực hiện các tính toán thông thường hoặc khi đang chạy các ứng dụng thương mại của họ.

Điều gì đã làm cho vấn đề đáng chú ý này không được *Intel* coi là bug, mặt khác cái cách mà *Intel* điều khiển tình hình:

- Họ đã phát hiện ra các vấn đề trong khi thực thi các bài test của chính họ trước khi chip được tung ra thị trường. Các nhà quản lý của Intel đã quyết định rằng vấn đề này không đủ nghiêm trọng và ít khả năng xảy ra để cần thiết phải *fixing* (sửa) nó hoặc thậm chí là *publicizing* (công khai) nó.
- Lỗi đã bị phát hiện, Intel cố gắng để giảm bớt tính chất nghiêm trọng của vấn đề đã bị nhận bằng cách công bố công khai (*press release*).
- Khi bị gây áp lực, Intel đã ngó ý muốn thay thế miễn phí những chip bị lỗi, nhưng chỉ với điều kiện là người sử dụng đó phải chứng minh được rằng anh ta đã bị ảnh hưởng do lỗi (bug).
- Họ đã gặp phải sự phản đối kịch liệt. Các diễn đàn trên Internet đã tạo sức ép với sự giận dữ của những khách hàng khó tính, đòi Intel phải fix vấn đề này. Các bản tin thì vẽ lên hình ảnh về Intel giống như một công ty vô trách nhiệm với khách hàng. Cuối cùng, Intel đã phải xin lỗi bằng cách điều chỉnh bug và đã phải bỏ ra trên **400 triệu dollar** để chi trả cho quá trình thay thế các chip bị lỗi.

Bây giờ, Intel luôn công khai các vấn đề trên Website của họ và cẩn trọng giám sát sự hồi đáp của các khách hàng trên các diễn đàn (*newsgroups*).

**Chú ý:** Vào ngày 28/08/2000, một thời gian ngắn trước khi phiên bản đầu tiên của cuốn sách này được sản xuất, Intel đã thông báo việc thu hồi tất cả các bộ vi xử lý Pentium III 1.13MHz, sau khi các con chip này được tung ra thị trường khoảng 1 tháng. Một vấn đề đã bị phát hiện. Vì vậy họ phải thực thi cho lời khẳng định chắc chắn rằng các ứng dụng sẽ luôn chạy ổn định. Họ phải lập kế hoạch để thu hồi những chiếc máy tính đã tới tay khách hàng và tính toán giá thành để thay thế cho những con chip bị lỗi. Giống như lời của huyền thoại bóng chày *Yogi Berra* đã nói: “*This is like déjà vu all over again*”.

Tàu vũ trụ của NASA đáp xuống địa cực của sao Hỏa (NASA Mars Polar Lander), 1999

Ngày 3/12/1999, Tàu vũ trụ của NASA đáp xuống địa cực của sao Hỏa đã biến mất khỏi vòng kiểm soát trong khi nó đang cố gắng đáp xuống bề mặt của sao Hỏa. Ban Báo Cáo sự cố đã điều tra sự cố và xác định rằng nguyên nhân có thể xảy ra nhất của sự cố này là việc cài đặt một bit dữ liệu đơn lẻ. Điều đáng chú ý nhất là tại sao sự cố này lại chưa từng được xảy ra trong các cuộc thí nghiệm nội bộ.

Theo lý thuyết, kế hoạch đáp tàu như sau: khi tàu đang đáp xuống bề mặt, nó sẽ mở ra chiếc dù nhằm làm giảm tốc độ. Một vài giây sau khi mở dù, 3 chân của máy dò sẽ mở ra và chệt ở vị trí đáp. Khi máy dò ở vị trí cách bề mặt sao hỏa 1.800m nó sẽ nhả dù và đốt nóng (thruster) để giảm khoảng cách còn lại so với bề mặt sao hỏa

Để tiết kiệm, NASA đã đơn giản bộ máy quyết định thời gian ngắt. Thay thế Rada đất tiền trên tàu vũ trụ, họ đã cài đặt công tắc tiếp xúc (*Contact switch*) ở chân của máy dò. Nói một cách đơn giản, khi các chân của máy dò mở ra sẽ bật công tắc để động cơ đốt cháy cho đến khi các chân chạm đất.

Thật không may ban báo cáo sự cố đã phát hiện ra trong quá trình kiểm tra của họ rằng khi các chân được tách ra để chạm tới đất, một rung động của máy đã làm trượt công tắc đốt cháy và việc thiết đặt bit này đã gây tai họa. Đây là một vấn đề rất nghiêm trọng, máy tính đã tắt bộ phận đốt nóng và con tàu đã bị vỡ ra từng mảnh sau khi rơi từ độ cao 1.800m xuống bề mặt sao Hỏa.

Kết quả thật là thâm trầm, nhưng lý do lại rất đơn giản. Con tàu thám hiểm đã được kiểm tra bởi rất nhiều đội. Một đội đã kiểm tra chức năng mở các chân của con tàu và một đội khác thì kiểm tra việc đáp tàu xuống mặt đất. Đội đầu tiên đã không biết rằng: một bit được thiết đặt cho việc mở các chân của con tàu không nằm trong vùng kiểm tra của họ. Đội thứ 2 thì luôn luôn thiết lập lại máy tính, xóa bit dữ liệu trước khi nó bắt đầu được kiểm tra. Cả 2 đội trên đều làm việc độc lập và hoàn thành nhiệm vụ của mình rất hoàn hảo. Nhưng lại không hoàn hảo khi kết hợp các nhiệm vụ với nhau.

Hệ thống phòng thủ tên lửa Patriot, 1991

Hệ thống phòng thủ tên lửa *Patriot* (người yêu nước) của Mỹ là một phiên bản *scaled-back* của chương trình khởi động chiến lược phòng thủ “*Star Wars*” được khởi động bởi tổng thống *Ronald Reagan*. Nó đặt nền móng cho chiến tranh Vùng Vịnh (*Gulf war*) như một hệ thống phòng thủ tên lửa *Iraqi Scud*. Mặc dù đã có rất nhiều câu chuyện quảng bá về sự thành công của hệ thống, tuy nhiên vẫn tồn tại lỗi khi chống lại một vài tên lửa. Một trong số đó đã giết chết 28 lính Mỹ ở *Dhahran, Saudi Arabia*. Quá trình phân tích đã cho thấy rằng, phần mềm đã bị lỗi nghiêm trọng. Một thời gian trễ rất nhỏ trong đồng hồ hệ thống đã được tích lũy lại sau 14h, và hệ thống theo dõi không còn chính xác nữa. Trong cuộc tấn công *Dhahran*, hệ thống đã điều hành trong hơn 100h.

Sự cố Y2K (năm 2000), khoảng 1974

Vào đầu những năm 1970, một lập trình viên, tên anh ta là Dave, đang làm việc cho hệ thống trả tiền của công ty anh ta. Máy tính mà anh ta sử dụng có bộ nhớ lưu trữ rất nhỏ, buộc anh ta phải giữ gìn những byte cuối cùng mà anh ta có. Dave đã rất tự hào rằng anh ta có thể đóng gói các chương trình của mình một cách chặt chẽ (*tightly*) hơn so với một vài đồng nghiệp của anh ta. Một phương thức mà anh ta đã sử dụng là chuyển định dạng ngày tháng từ 4 chữ số, ví dụ 1973 thành định dạng 2 chữ số, ví dụ 73. Bởi vì, hệ thống trả tiền (*Payroll*) của anh ta phụ thuộc rất nặng vào xử lý ngày tháng, nhờ thế Dave có thể giữ lại những không gian nhớ có giá trị. Trong một thời gian ngắn, anh ta đã xem xét những vấn đề có thể xuất hiện khi đến thời điểm năm 2000 và hệ thống của anh ta đã bắt đầu thực hiện các công việc tính toán với các năm được đại diện bằng 00, 01... Anh ta cũng nhận thấy rằng, sẽ có một vài vấn đề xảy đến, nhưng anh ta đã nghĩ rằng chương trình của anh ta sẽ được thay thế hoặc cập nhật trong vòng 25 năm, và nhiệm vụ hiện tại của anh ta là quan trọng hơn là kế hoạch trong tương lai xa như vậy. Và thời hạn đó cũng đã đến. Năm 1995, chương trình của Dave vẫn được sử dụng, Dave đã nghỉ hưu. Và không một ai biết làm thế nào để vào được hệ thống kiểm tra xem nếu đến năm 2000 thì chuyện gì sẽ xảy ra. Chỉ một mình Dave biết cách để fix nó.

Người ta đã ước tính rằng, phải mất đến **vài trăm tỷ dollar** để có thể cập nhật và fix những lỗi tiềm tàng vào năm 2000, cho các chương trình máy tính trên toàn thế giới có sử dụng hệ thống của Dave.

Mối hiểm nguy của Virus, năm 2004

01/04/1994, một thông điệp đã được gửi tới một vài nhóm người sử dụng internet và sau đó nó được truyền bá như một email có chứa một loại virus ẩn trong các bức ảnh có định dạng JPEG trên internet. Người ta đã cảnh báo rằng chỉ cần thao tác mở và xem những bức tranh bị nhiễm sẽ dẫn đến việc cài đặt virus trên PC của bạn. Sự thay đổi của những lời cảnh báo nói rõ rằng virus có thể làm hỏng màn hình máy tính của bạn và rằng những chiếc màn hình Sony Trinitron là “đặc biệt nhạy cảm”.



Nhiều người đã chú ý tới những lời cảnh báo và làm sạch những file ảnh JPEG trong hệ thống của họ. Thậm chí, một số người quản trị hệ thống còn tìm hiểu sâu bên trong những khối ảnh JPEG được nhận từ email trên hệ thống của họ.

Cuối cùng, mọi người cũng đã nhận thấy rằng, thông điệp ban đầu được gửi đi vào ngày cá tháng 4 (“April Fools Day”) và sự thật là không có chuyện gì cả, nhưng câu chuyện đùa này đã đi quá xa. Các chuyên gia đã rung hồi chuông cảnh báo rằng: không có một cách khả thi nào để một bức ảnh JPEG có khả năng làm máy tính của bạn bị nhiễm virus. Sau tất cả, người ta khẳng định rằng một bức tranh cũng chỉ là dữ liệu, nó không thể thực thi mã chương trình.

Mười năm sau, vào mùa thu năm 2004, một virus *proof-of-concept* đã được tạo ra, nó chứng minh rằng một bức ảnh JPEG có thể được tải về cùng với 1 virus. Nó sẽ gây ảnh hưởng tới hệ thống được sử dụng để xem nó. Những mẫu tin (*software patches*) được tạo ra một cách nhanh chóng và được thông báo rộng khắp để ngăn chặn virus lan tràn. Tuy nhiên, chỉ là vấn đề thời gian cho đến khi họ không chế được vấn đề này trên internet bằng cách làm sạch các bức ảnh trên đường truyền.

## Lỗi (bug) là gì

Bạn vừa được tìm hiểu về một số vấn đề có thể xảy ra khi phần mềm bị lỗi. Nó có thể dẫn đến những phiền phức, giống như khi một máy chơi *game* không thể làm việc một cách hợp lý, hoặc nó có thể dẫn đến một thảm họa khủng khiếp nào đó. Số tiền để giải quyết vấn đề và sửa lỗi có thể lên tới hàng triệu *dollar*. Trong các ví dụ ở trên, rõ ràng phần mềm không hoạt động như dự tính ban đầu. Nếu là một tester, bạn sẽ phải tìm thấy hầu hết những lỗi của phần mềm. Hầu hết là những lỗi đơn giản và tinh vi, có khi quá nhỏ đến nỗi không thể phân biệt được cái nào là lỗi và cái nào không phải là lỗi.

### NHỮNG THUẬT NGỮ VỀ CÁC LỖI PHẦN MỀM:

Phụ thuộc vào nơi mà bạn được làm việc (như một tester), bạn sẽ sử dụng những thuật ngữ khác nhau để mô tả: điều gì sẽ xảy đến khi phần mềm bị lỗi. Dưới đây là một số thuật ngữ:

*Defect* nhược điểm

*Fault* khuyết điểm

*Failure* sự thất bại

*Anomaly* sự dị thường

*Variance* biến dị

*Incident* việc rắc rối

*Problem* vấn đề

*Error* lỗi

*Bug* lỗi

*Feature* đặc trưng

*Inconsistency* sự mâu thuẫn

(Chúng ta có một danh sách các thuật ngữ không nên nhắc đến, nhưng hầu hết chúng được sử dụng riêng biệt, độc lập giữa các lập trình viên)

Bạn có thể thấy ngạc nhiên rằng có quá nhiều từ để mô tả một lỗi phần mềm. Tại sao lại như vậy? Có phải tất cả chúng đều thật sự dựa trên văn hóa của công ty và quá trình mà công ty sử dụng để phát triển phần mềm của họ. Nếu bạn tra những

từ này trong từ điển, bạn sẽ thấy rằng tất cả chúng đều có ý nghĩa khác nhau không đáng kể. Chúng cũng có ý nghĩa được suy ra từ cách mà chúng được sử dụng trong các cuộc đàm thoại hàng ngày.

Ví dụ, *fault*, *failure* và *defect* có xu hướng ám chỉ một vấn đề thật sự quan trọng, thậm chí là nguy hiểm. Đường như là không đúng khi gọi một biểu tượng (*icon*) không được tô đúng màu là 1 lỗi (*fault*). Những từ này cũng thường ám chỉ một lời khiển trách: chính là do nó (*fault*) mà phát sinh lỗi phần mềm (*software failure*) (“it’s his fault that the software failure.”)

*Anomaly*, *incident*, *variance* thì không có vẻ là quá tiêu cực và thường được sử dụng để đề cập tới sự vận hành không được dự tính trước thay vì hoàn toàn là lỗi (*all-out failure*). “Tổng thống đã tuyên bố rằng nó là một sự dị thường phần mềm đã làm cho tên lửa đi sai lịch trình của nó” (“The president stated that it was a software anomaly that caused the missile to go off course.”)

Có lẽ, *Problem*, *error* và *bug* là những thuật ngữ chung nhất thường được sử dụng.

JUST CALL IT WHAT IT IS AND GET ON WITH IT (Hãy chỉ gọi nó là cái mà nó là và tiếp tục với nó)

Một điều thú vị khi một số công ty và các đội sản xuất đã tiêu tốn khá nhiều thời gian quý báu của quá trình phát triển phần mềm vào việc thảo luận và tranh cãi về những thuật ngữ được sử dụng. Một công ty máy tính nổi tiếng đã mất hàng tuần để thảo luận với những kỹ sư của họ trước khi quyết định đổi tên *Product Anomaly Report (PARs)* thành *Product Incident Report (PIRs)*. Một số tiền lớn đã được sử dụng cho việc quyết định thuật ngữ nào là tốt hơn. Một khi quyết định đã được đưa ra (Once the decision was made), tất cả các công việc liên quan đến giấy tờ, phần mềm, định dạng... phải được cập nhật để phản ánh thuật ngữ mới. Nó sẽ không được biết tới nếu nó gây ra bất kỳ sự khác biệt nào đối với hiệu quả làm việc của lập trình viên và tester.

Vậy, tại sao phải đưa ra chủ đề này? Thực sự là quan trọng khi một tester hiểu khả năng cá nhân đằng sau nhóm phát triển phần mềm mà bạn đang làm việc cùng. Cách thức họ đề cập tới các vấn đề về phần mềm của họ là dấu hiệu thông

thường về cách họ tiếp cận quá trình phát triển toàn bộ của họ. Họ có đề phòng, cẩn thận, thẳng thắn hay chỉ đơn giản là blunt?

Mặc dù tổ chức của bạn có thể chọn một cái tên khác, nhưng trong cuốn sách này tất cả các vấn đề về phần mềm sẽ được gọi là các *bug*. Không thành vấn đề nếu lỗi là lớn, nhỏ,

trong dự định, hay ngoài dự định, hoặc cảm giác của ai đó sẽ bị tổn thương bởi họ tạo ra chúng. Không có lý do gì để mở xẻ các từ. *A bug's a bug's a bug.*

## ĐỊNH NGHĨA VỀ LỖI PHẦN MỀM:

Các vấn đề về *software problems bugs* nghe có vẻ đơn giản, nhưng chưa hẳn đã giải quyết được nó. Bây giờ, từ *problem* cần được định nghĩa. Để tránh việc định nghĩa vòng quanh (*circular definitions*), điều quan trọng là phải mô tả được lỗi là gì?

Đầu tiên, bạn cần một thuật ngữ trợ giúp (*supporting term*): đặc tả phần mềm (*product specification*). Đặc tả phần mềm có thể gọi một cách đơn giản là *spec* hoặc *product spec*, là luận cứ của các đội phát triển phần mềm. Nó định nghĩa sản phẩm mà họ tạo ra, chi tiết là gì, hành động như thế nào, sẽ làm gì, và sẽ không làm gì? Luận cứ này có thể vạch ra phạm vi về hình thức từ một dạng hiểu biết về ngôn từ đơn giản, một email, hoặc 1 chữ viết nguệch ngoạc trên tờ giấy ăn, tới một tài liệu thành văn được hình thức hóa, chi tiết hơn. Trong bài 2, “Quy trình phát triển phần mềm”, bạn sẽ học về đặc tả phần mềm và quy trình phát triển, nhưng không phải là bây giờ, định nghĩa này là đầy đủ.

Một lỗi phần mềm xuất hiện khi 1 hoặc nhiều hơn trong 5 quy tắc dưới đây là đúng:

1. Phần mềm không thực hiện một số thứ giống như mô tả trong bản đặc tả phần mềm
2. Phần mềm thực hiện một số việc mà bản đặc tả yêu cầu nó không được thực hiện
3. Phần mềm thực hiện một số chức năng mà bản đặc tả không đề cập tới
4. Phần mềm không thực hiện một số việc mà bản đặc tả không đề cập tới, nhưng là những việc nên làm
5. Trong con mắt của người kiểm thử, phần mềm là khó hiểu, khó sử dụng, chậm đối với người sử dụng

Để tìm hiểu kỹ hơn về mỗi quy tắc, hãy cố gắng xem ví dụ dưới đây để áp dụng chúng cho phần mềm *calculator* trong máy tính.

Có lẽ, đặc tả của 1 *calculator* đã nói rõ rằng: nó sẽ thực thi phép cộng, phép trừ, phép nhân, phép chia đúng. Nếu bạn là một tester, nhận việc kiểm thử phần mềm *Calculator*, nhấn phím “+” và không có chuyện gì xảy ra, đó là một lỗi theo quy tắc 1. Nếu bạn nhận được câu trả lời sai, cũng có nghĩa rằng đó là một lỗi, bởi vì theo **quy tắc 1**.

Bản đặc tả phần mềm yêu cầu rằng, *calculator* sẽ không bao giờ bị đột ngột ngưng hoạt động, bị khóa lại hoặc bị đóng băng. Nếu bạn đập thành thịch lên các phím và nhận được thông điệp từ *calculator* là “*not responding*” (dừng quá trình hồi đáp với dữ liệu vào), đây là một lỗi theo **quy tắc 2**.

Mục đích là bạn nhận được phần mềm calculator để kiểm thử và thấy rằng bên cạnh các phép cộng, trừ, nhân, chia; nó còn thực hiện các phép căn bậc 2. Điều này chưa từng được nêu trong bản đặc tả. Một lập trình viên có nhiều tham vọng vừa thêm nó vào bởi vì anh ta cảm thấy nó sẽ là một đặc tính tuyệt vời (*great feature*). Đây không phải là một *feature*, nó thật sự là một *bug* theo **quy tắc 3**. Phần mềm đang thực hiện một số công việc mà bản đặc tả phần mềm không hề đề cập tới. Mặc dù sự điều khiển không định hướng này có thể là tốt, nhưng nó sẽ yêu cầu thêm những lỗi lập trình và kiểm thử (vì có thể sẽ xuất hiện thêm nhiều lỗi). Lúc này chi phí cho việc sản xuất phần mềm cũng lớn hơn, điều này làm giảm hiệu quả kinh tế của quá trình sản xuất phần mềm.

Đọc **quy tắc thứ 4** có thể thấy hơi lạ với sự phủ định kép, nhưng mục đích của nó là tìm thấy những đặc điểm bị lãng quên, không được nhắc tới trong bản đặc tả. Bạn bắt đầu kiểm thử phần mềm *Calculator* và khám phá ra rằng, khi Pin yếu, bạn không nhận được những câu trả lời đúng cho quá trình tính toán của bạn nữa. Chưa ai từng xem xét xem *calculator* phản ứng lại như thế nào trong chế độ này. Một giả định không tốt được tạo ra rằng: pin luôn được nạp đầy. Bạn mong muốn rằng công việc sẽ được duy trì cho đến khi pin hoàn toàn hết, hoặc ít nhất bằng cách

nào đó báo cho bạn biết Pin đã yếu. Những phép tính đúng đã không xảy ra khi pin yếu, và nó cũng không chỉ rõ điều gì sẽ xảy đến. Quy tắc số 4 tạo nên lỗi này.

**Quy tắc số 5** mang tính chất tổng hợp (*catch-all*). *Tester* là người đầu tiên thực sự sử dụng phần mềm như một khách hàng lần đầu sử dụng phần mềm. Nếu bạn phát hiện một vài điều mà bạn thấy không đúng vì bất cứ lý do gì, thì nó là một lỗi. Trong trường hợp của *calculator*, có lẽ bạn đã tìm thấy những nút có kích thước quá nhỏ. Hoặc có thể sự sắp xếp của nút “=” đã làm cho nó khó sử dụng. Hoặc sự bố trí màu sắc làm cho nó rất khó nhìn... Tất cả những điều này đều là lỗi (bug) theo quy tắc 5.

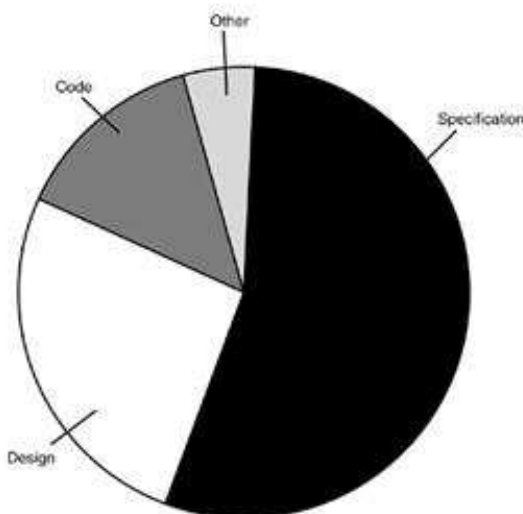
*Chú ý:* Mọi người sử dụng một phần của phần mềm sẽ có những mong đợi khác nhau và những ý kiến phần mềm nên hoạt động như thế nào. Sẽ không thể viết được phần mềm mà mọi người sử dụng đều thấy nó hoàn hảo. *Tester* sẽ luôn phải giữ những ý nghĩ này trong suy nghĩ của họ khi họ áp dụng quy tắc 5 để kiểm thử. Xét một cách thấu đáo, hãy sử dụng khả năng đánh giá tốt nhất của bạn, và **quan trọng nhất là phải hợp lý**. Ý kiến của bạn có giá trị, nhưng, bạn sẽ được tìm hiểu trong các phần sau, không phải tất cả các lỗi bạn tìm được có thể hoặc sẽ được sửa (*fix*).

Để có một số ví dụ đơn giản và điển hình, bạn hãy nghĩ xem các quy tắc được áp dụng như thế nào với phần mềm mà bạn sử dụng hàng ngày. Đây là điều bạn mong đợi, đây là điều không mong đợi? Bạn thấy điều gì đã được chỉ rõ và điều gì bị bỏ quên? Và điều mà bạn hoàn toàn không thích về phần mềm này?

Định nghĩa trên về lỗi của một phần mềm đã bao quát những vấn đề cơ bản, nhưng nếu sử dụng tất cả 5 quy tắc trên sẽ giúp bạn định nghĩa được các loại vấn đề khác nhau trong phần mềm mà bạn đang kiểm thử.

## Tại sao lỗi xuất hiện

Bây giờ, bạn biết lỗi là gì, bạn có thể tự hỏi tại sao lỗi xuất hiện. Bạn sẽ ngạc nhiên khi khám phá ra rằng hầu hết những lỗi này không phải là lỗi do lập trình. Quá trình khảo sát trên vô số dự án từ rất nhỏ tới cực lớn và kết quả luôn luôn giống nhau. Các lý do quan trọng gây ra lỗi phần mềm được mô tả như hình 1.1 dưới đây:



Các lỗi có thể bị phát sinh do nhiều lý do, nhưng trong quá trình phân tích các dự án mẫu thì lý do chính có thể được tìm thấy trong quá trình truy vết theo bản đặc tả.

Những lý do liên quan đến bản đặc tả là nguyên nhân chính làm xuất hiện lỗi *specification*:

- Một số bản đặc tả không viết cụ thể, không đủ kỹ lưỡng
- Hoặc nó liên tục thay đổi, nhưng lại không có sự phối hợp, trao đổi thông tin kịp thời với các đội phát triển dự án.
- Lập kế hoạch cho phần mềm là vô cùng quan trọng. Nếu nó không được thiết kế đúng, lỗi sẽ phát sinh

Lý do quan trọng tiếp theo mà dễ phát sinh lỗi là quá trình thiết kế. Đây là nơi mà các lập trình viên sắp xếp kế hoạch về phần mềm của họ. So sánh nó với kiến trúc được thiết kế cho một ngôi nhà. Các lỗi xuất hiện ở đây với những lý do tương tự như khi chúng xuất hiện trong bản đặc tả. Nó bị dồn lại, thay đổi, hoặc giao tiếp không tốt.

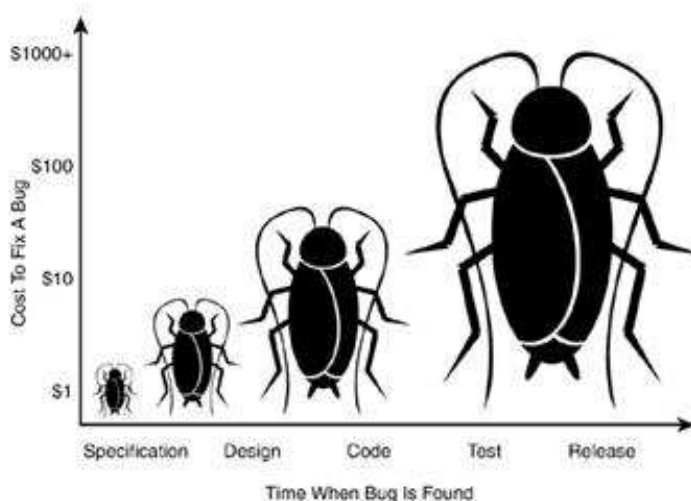
**Chú ý:** Có một câu nói quen thuộc như sau: “Nếu bạn không thể nói, bạn cũng sẽ không thể làm” (“*If you can’t say it, you can’t do it*”). Điều này có thể áp dụng một cách hoàn hảo cho quy trình phát triển và kiểm thử phần mềm.

Những lỗi về code có thể quen thuộc với bạn hơn nếu như bạn là một lập trình viên. Điển hình, như là có thể theo dõi được độ phức tạp của phần mềm, văn bản nghèo nàn (đặc biệt trong các đoạn mã được cập nhật hoặc thay đổi), áp lực của lịch làm việc, hoặc những lỗi ngớ ngẩn. Điều quan trọng là phải chú ý rằng có nhiều lỗi thường xuất hiện bên ngoài là những lỗi lập trình được theo dõi qua bản đặc tả và lỗi thiết kế.

Một số thứ tưởng rằng là lỗi nhưng thực ra lại không phải. Một số lỗi bị nhân bản lên, bắt nguồn từ những nguyên nhân giống nhau. Một số lỗi bắt nguồn từ việc kiểm tra sai. Và cuối cùng, những bug (hay những thứ mà chúng ta tưởng là lỗi) hóa ra lại không phải là lỗi và chúng chiếm tỷ lệ nhỏ trong những bug được báo cáo.

## Chi phí cho việc sửa lỗi

Bạn sẽ tìm hiểu trong bài 2, phần mềm không xuất hiện một các kỳ diệu mà thường là phải có cả 1 quá trình phát triển các phương thức, kế hoạch được sử dụng để tạo ra nó. Từ sự khởi đầu của phần mềm, qua quá trình lập kế hoạch, lập trình, và kiểm thử, đến khi nó được sử dụng bởi khách hàng, những lỗi này có khả năng được tìm thấy. Hình 1.1 biểu diễn một ví dụ về những chi phí cho việc sửa những lỗi có thể phát sinh trong toàn bộ thời gian thực hiện dự án.



Chi phí cho việc sửa lỗi có thể tăng đột ngột trên toàn bộ dự án

Chi phí được tính theo hàm loga, mỗi lần chúng tăng lên gấp 10 lần. Lỗi được tìm thấy và sửa lại trong thời gian gần nhất khi bản đặc tả bắt đầu được viết thì chi phí có thể không là gì cả, hoặc chỉ là 1\$ cho ví dụ của chúng ta. Cũng với lỗi tương tự, nếu nó không được tìm thấy cho đến khi phần mềm được lập trình và kiểm thử thì chi phí có thể lên tới 10\$ đến 100\$. Nếu một để một khách hàng tìm ra nó, thì chi phí có thể lên tới hàng nghìn thậm chí hàng triệu dollar.



Như một ví dụ trong cuốn sách này, hãy xem xét về *The Disney Lion King* đã được thảo luận gần đây. Lý do cơ bản của vấn đề là phần mềm này không làm việc được trên nền máy tính phổ biến lúc đó. Nếu như ngay ở giai đoạn đặc tả đầu tiên, ai đó đã nghiên cứu dạng máy PC phổ biến và chỉ ra rằng phần mềm cần được thiết kế và kiểm thử để làm việc được trên những cấu hình đó, thì chi phí cho những cố gắng trên sẽ là rất nhỏ. Nếu điều này không được thực hiện, một bản backup sẽ được gửi cho tester để thu thập những mẫu máy tính phổ biến và thay đổi phần mềm cho phù hợp với chúng. Họ sẽ tìm thấy lỗi, nhưng quá trình sửa lỗi sẽ tốn kém hơn bởi vì phần mềm sẽ phải gỡ lỗi, sửa lỗi và kiểm thử lại. Đội phát triển dự án có thể cũng phải gửi đi một phiên bản đầu tiên của phần mềm tới một nhóm nhỏ các khách hàng để họ kiểm tra, quá trình này gọi là kiểm thử *beta*. Những khách hàng này, đặc trưng cho một thị trường lớn, sẽ có khả năng khám phá ra nhiều vấn đề. Tuy nhiên, lỗi phần mềm không phải là hoàn toàn cho đến khi có hàng nghìn CD-ROM được sản xuất và bán. Disney đã kiên trì trợ giúp khách hàng qua điện thoại trả lời về sản phẩm, thay thế các ổ CD-ROM, tốt như quá trình gỡ lỗi khác, sửa lỗi và vòng đời kiểm thử. Thật dễ dàng để làm tiêu tan toàn bộ lợi ích của sản phẩm nếu các lỗi là nguy hiểm đối với khách hàng.

## **Người kiểm thử phần mềm (software tester) làm những gì?**

Bây giờ bạn có phần mềm với những lỗi ngớ ngẩn, bạn đã biết định nghĩa của một lỗi là gì, và bạn cũng biết chi phí cho chúng đắt đỏ như thế nào. Vậy mục đích của một tester là gì: *Mục đích của tester là tìm ra lỗi phần mềm (“The goal of a software tester is to find bugs”)*

Bạn có thể liên kết (run across) với các đội phát triển sản phẩm, người luôn muốn các tester xác nhận rằng phần mềm của họ làm việc tốt, không có lỗi. Hãy kiểm tra lại các Case study về con tàu thám hiểm lên địa cực của sao Hỏa, và bạn sẽ thấy tại sao đây là hướng tiếp cận sai. Nếu bạn chỉ kiểm tra những thứ mà sẽ làm việc và cài đặt cho quá trình kiểm tra của bạn, vì vậy, chúng sẽ luôn pass. Và bạn sẽ rất dễ bỏ quên những thứ không làm việc. Cuối cùng, bạn sẽ không phát hiện ra một số lỗi.

Nếu bạn đang bỏ sót một số lỗi, bạn sẽ phải trả giá cho dự án của bạn và tiền của công ty bạn. Là một *tester*, bạn không nên bằng lòng với những lỗi đã được tìm thấy, bạn nên nghĩ xem làm thế nào để tìm thấy chúng sớm nhất trong quy trình phát triển phần mềm, như vậy, chi phí để *fix* lỗi sẽ ít hơn.

*Mục đích của một tester là tìm các lỗi và tìm thấy chúng một cách sớm nhất có thể (“The goal of a software tester is to find bugs and find them as early as possible”).*

Nhưng, tìm kiếm các lỗi, thậm chí phát hiện chúng từ rất sớm vẫn là chưa đủ. Hãy nhớ lại định nghĩa về 1 lỗi. Bạn, 1 tester, là con mắt của khách hàng, trước tiên phải xem xét phần mềm. Bạn nói chuyện với khách hàng và phải tìm kiếm một sự hoàn chỉnh.

*Mục đích của một tester là tìm các lỗi, tìm thấy chúng một cách sớm nhất có thể, và chắc chắn rằng chúng sẽ được sửa (“ The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed. ”).*

Câu nói cuối cùng này rất quan trọng. Bạn hãy ghi nhớ nó và lấy ra xem lại khi bạn tìm hiểu về các kỹ thuật kiểm thử được thảo luận trong toàn bộ những nội dung quan trọng của cuốn sách này.

**Chú ý:** Điều quan trọng là phải chú ý: lỗi phần mềm được sửa không có nghĩa rằng phần mềm đã hoàn hảo. Phần mềm nên được bổ sung thêm những hướng dẫn sử dụng hoặc cung cấp các khóa đào tạo đặc biệt cho khách hàng. Việc này có thể còn yêu cầu thay đổi những số liệu mà nhóm Marketing quảng cáo hoặc thậm chí trì hoãn việc giải phóng (bỏ qua) *buggy feature*. Trong suốt cuốn sách này, bạn sẽ học cách để trở thành người đi tìm kiếm sự hoàn hảo và phải chắc chắn rằng những lỗi được phát hiện sẽ được sửa, và sẽ có những bài thực hành thực tế cho bạn kiểm thử. Đừng có tìm kiếm đường xoắn ốc nguy hiểm của sự hoàn hảo không thể có thật (*Don't get caught in the dangerous spiral of unattainable perfection*).

## **Những tố chất nào tạo nên một tester tốt?**

Trong đoạn phim *Star Trek II: The Wrath of Khan*, Spock nói rằng: “Là một vấn đề của lịch sử vũ trụ, phá hủy bao giờ cũng dễ hơn kiến tạo” (“As a matter of cosmic history, it has always been easier to destroy than to create”). Mới nhìn qua, có thể mọi người sẽ nghĩ công việc của một tester là đơn giản hơn so với công việc của một lập trình viên. Phát hiện ra những lỗi lập trình chắc chắn là dễ hơn so với viết code. Nhưng thật ngạc nhiên, điều đó lại không đúng. Cách tiếp cận rất bài bản và có kỷ luật với phần mềm mà bạn sẽ tìm hiểu trong cuốn sách này yêu cầu bạn phải cống hiến và làm việc một cách chăm chỉ chẳng kém gì một lập trình viên. Hai công việc đều phải sử dụng rất nhiều kỹ năng tương tự nhau, và mặc dù kiểm thử phần mềm không nhất thiết phải cần là một lập trình viên chuyên nghiệp, nhưng họ lại tạo ra những khoản lợi nhuận lớn.

Ngày nay, hầu hết những công ty đã trưởng thành đều coi kiểm thử phần mềm như công việc của một kỹ sư kỹ thuật. Họ thừa nhận rằng phải đào tạo các tester trong các đội dự án của họ và cho phép họ áp dụng các kỹ thuật vào quá trình phát triển phần mềm để xây dựng được một phần mềm có chất lượng tốt hơn. Thật không may, vẫn có một vài công ty không đánh giá đúng nhiệm vụ khó khăn của quá trình kiểm thử và giá trị của những nỗ lực kiểm thử rất bền bỉ. Với sự giao thiệp trong thị trường tự do, có những công ty thường nổi bật hơn hẳn, bởi vì khách hàng thì thường nói rằng: không nên mua những sản phẩm có nhiều khiếm khuyết. Một tổ chức kiểm thử tốt (hoặc thiếu một cái) có thể tạo nên hoặc phá hỏng một công ty.

Hãy nhìn vào danh sách những đặc điểm mà một tester nên có:

- **Họ là những người thám hiểm:** tester không sợ mạo hiểm khi ở trong những hoàn cảnh mà họ chưa làm chủ được. Họ thích những khía cạnh mới của phần mềm, cài đặt nó trên máy của họ, và xem xét chuyện gì sẽ xảy ra.
- **Họ là những người thợ sửa chữa:** các tester làm rất tốt các công việc tính toán xem tại sao một số chức năng của phần mềm lại không làm việc. Họ rất thích những vấn đề khó giải quyết
- **Họ rất nghiêm khắc:** Các tester luôn phải thử nghiệm, họ có thể nhìn thấy một lỗi mà đã nhanh chóng biến mất hoặc là rất khó để tạo lại tình huống có lỗi đó. Đúng hơn là giải tán nó như một sự may mắn, họ sẽ cố gắng bằng mọi cách có thể để tìm ra nó.
- **Họ luôn sáng tạo:** việc kiểm thử những điều hiển nhiên, rõ ràng là không thể đủ với một tester. Công việc của họ cần những ý tưởng sáng tạo và thậm chí là các cách tiếp cận mới mẻ để tìm kiếm lỗi (*bug*).
- **Họ là những người cầu toàn:** Họ cố gắng để đạt đến sự hoàn hảo, nhưng họ cũng biết rằng điều đó là không thể đạt được và họ chấp nhận dừng quá trình kiểm thử khi họ thấy có thể.
- **Họ sử dụng óc phán đoán rất tốt:** tester cần đưa ra những quyết định về những thứ mà họ sẽ phải kiểm tra, và ước lượng quá trình kiểm tra sẽ diễn ra trong thời gian bao lâu, nếu như vấn đề mà họ tìm kiếm thật sự là một lỗi.
- **Họ là những người rất khéo léo và thích ngoại giao:** Tester luôn là người thông báo những tin tức xấu. Họ phải nói với lập trình viên những lỗi mà họ phát hiện. Một tester tốt sẽ biết cách để làm việc khéo léo và rất chuyên nghiệp, và họ cũng biết cách để làm việc với lập trình viên, những người không phải lúc nào cũng khéo léo và lịch thiệp.
- **Họ là người biết cách thuyết phục người khác:** các lỗi mà tester tìm thấy sẽ luôn được xem xét một cách đủ khắt khe để đảm bảo nó sẽ được sửa. Các tester cần chứng minh những luận điểm của họ rằng tại sao những lỗi mà họ phát hiện lại cần được sửa, và những lỗi này có thể gây ra những gì?

**KIỂM THỬ PHẦN MỀM LÀ MỘT CÔNG VIỆC RẤT THÚ VỊ:** Một đặc điểm cơ bản của các tester là họ rất thích thú với những thứ bị hỏng. Họ sống là để tìm kiếm những sai lầm của các hệ thống khó nắm bắt. Họ toại nguyện khi phát hiện lỗi trong các chương trình phức tạp. Họ thường nhảy lên sung sướng vì điều đó.

Trong những nét đặc trưng này, nếu tester có một số kiến thức về lập trình phần mềm là một ưu thế rất lớn. Bài này sẽ hiểu cách thức mà phần mềm được viết, nó đưa cho bạn một cách nhìn khác về nơi mà các lỗi phần mềm được tìm thấy. Vì vậy, bài này sẽ giúp bạn trở thành một tester làm việc có hiệu quả và gây ảnh hưởng nhiều hơn. Có thể nó cũng giúp bạn phát triển các công cụ kiểm thử.

Cuối cùng, nếu bạn là một chuyên gia trong lĩnh vực không phải là về máy tính, thì sự hiểu biết của bạn có thể vô cùng giá trị để đội dự án phần mềm tạo ra được một sản phẩm mới. Hàng ngày, phần mềm đang được viết để làm mọi thứ. Sự hiểu biết của bạn

về dạy học, nấu ăn, máy bay, y học hoặc bất cứ cái gì sẽ là sự trợ giúp đặc lực cho các lỗi được tìm thấy trong phần mềm về lĩnh vực đó.

# Bài 2 : Quy trình phát triển phần mềm

## Quy trình phát triển phần mềm

### Quy trình phát triển phần mềm

Mục đích của phần này là hướng dẫn cho bạn mọi thứ về quy trình phát triển phần mềm sẽ được áp dụng trong môn học này. Mục đích là cho bạn một cái nhìn tổng quan về tất cả các phần bên trong một sản phẩm phần mềm và thấy được một vài hướng tiếp cận chung thường được sử dụng ngày nay. Với những hiểu biết này, bạn sẽ tự có cách tốt nhất để áp dụng các kỹ năng kiểm thử phần mềm mà bạn sẽ được học.

Phần chính của bài này bao gồm:

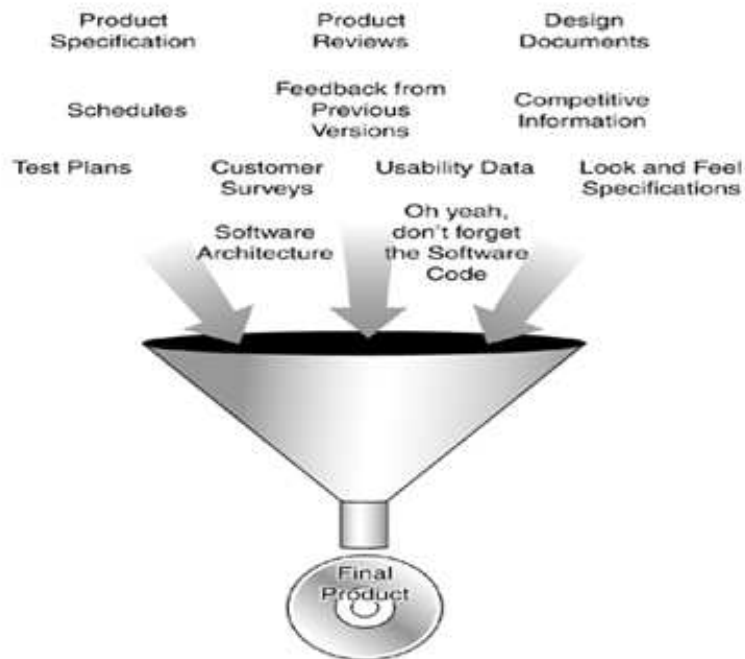
- Các thành phần (component) chính nào bên trong một sản phẩm phần mềm
- Những ai và các kỹ năng nào đóng góp vào một sản phẩm phần mềm
- Xử lý phần mềm như thế nào để từ một ý tưởng xây dựng lên một sản phẩm cuối cùng

### Các thành phần của phần mềm (product components)

Một sản phẩm phần mềm chính xác là cái gì? Nhiều người cho rằng, đơn giản nó là thứ mà người ta down được từ internet hoặc cài đặt được từ DVD để nó chạy được trên máy tính. Đây là một mô tả tốt, nhưng là một mô tả tốt trong một phạm vi nhỏ, nhưng thật sự, nhiều thành phần được ẩn bên trong phần mềm. Có nhiều phần bên trong hộp “*come in the box*”, mà chúng thường được lấy ra để trợ giúp hoặc có thể bị bỏ qua. Mặc dù rất dễ quên tất cả các phần này, nhưng là một tester, bạn cần biết về chúng. Bởi vì chúng là những nội dung cần kiểm tra và chúng có thể chứa lỗi.

#### 1. Lỗi lực đằng sau một sản phẩm phần mềm như thế nào?

Trước tiên, bạn hãy nhìn nhận những lỗi lực phía sau một sản phẩm phần mềm. Hình 2.1 chỉ ra một vài những thành phần trừu tượng mà bạn có thể không hề nghĩ tới.



Rất nhiều nỗ lực (effort) ẩn dấu phía dưới một sản phẩm phần mềm

Vậy, đâu sẽ là tất cả những thứ này, bên cạnh mã nguồn thật sự, liệu đây có phải là một cái phễu (*funnel*) phần mềm? Nhìn lướt qua, có lẽ chúng là những thứ hiển nhiên mà một lập trình viên tạo ra. Và rõ ràng chúng không phải là những thứ mà có thể được xem trực tiếp từ CD – ROM. Nhưng để dùng một dòng diễn tả cho “*món mĩ ổng thương mại*”: “chúng ở đây”. Ít nhất cũng là như vậy.

Những thuật ngữ được dùng trong ngành công nghiệp phần mềm để mô tả thành phần một sản phẩm phần mềm, mà đã được tạo ra và tiếp tục tới một nơi nào khác có thể được làn truyền. Cách dễ nhất để giải thích những thứ mà tất cả sự chuyển giao này là tổ chức chúng thành những loại lớn.

### 1. Yêu cầu của khách hàng

Phần mềm cần được viết một cách đầy đủ các yêu cầu mà một người hoặc một nhóm người đưa ra. Đó là những khách hàng. Để làm hợp lý những yêu cầu này, một nhóm phát triển phần mềm phải tìm ra những cái mà khách hàng muốn. Một nhóm thì phỏng đoán những yêu cầu, nhưng hầu hết các thông tin chi tiết được thu thập trong quá trình khảo sát, hỏi đáp từ những phiên bản trước của phần mềm, cạnh tranh các thông tin về sản phẩm, các nhìn tổng quan, các nhóm trọng tâm, và một số các phương thức khác, một số formal, một số các khác. Tất cả những thông tin này được tìm hiểu, xem xét và làm sáng tỏ để quyết định chính xác những đặc trưng nào mà sản phẩm phần mềm cần có.

HÃY ĐƯA CÁC FEATURES (ĐẶC TRƯNG) NÀY VÀO PERSPECTIVE VỚI CÁC FOCUS GROUP (NHÓM TRỌNG TÂM): một phương tiện phổ biến để nhận những hồi đáp trực tiếp từ các khách hàng tiềm năng là sử dụng các *focus group*. *Focus group* thường được tổ chức bởi các công ty khảo sát độc lập - những người thiết đặt các cơ quan trong các phố mua bán lớn. Các cuộc khảo sát hoặc những chuyến đi bộ điền hình vòng quanh phố mua bán với một bìa kẹp hồ sơ và hỏi những người đi qua nếu họ muốn đóng góp một phần vào quá trình nghiên cứu. Họ sẽ hỏi một vài câu hỏi liên quan đến chất lượng như: “Bạn có một PC ở nhà không? Bạn sử dụng phần mềm X như thế nào? Bạn sử dụng bao nhiêu thời gian để online?” và tiếp tục... Nếu bạn phù hợp với yêu cầu về đối tượng, họ sẽ mời bạn quay trở lại một vài giờ để tham gia cùng với một vài người khác trong *focus group*. Ở đây bạn sẽ được hỏi các câu hỏi chi tiết hơn về phần mềm máy tính. Bạn có thể được biểu diễn những hộp phần mềm khác nhau và hỏi về những sở thích của bạn để bạn lựa chọn. Hoặc bạn có thể thảo luận như một đặc trưng nhóm (*group features*) giống như bạn nhìn thấy một sản phẩm mới. Tốt hơn hết bạn nên bỏ ra chút thời gian của mình. Hầu hết các *focus group* được hướng dẫn nhưng với tư cách là một công ty phần mềm mà yêu cầu thông tin được giữ kín. Nhưng thường thì rất dễ dàng để đoán ra họ là ai.

### 1. Đặc tả

Kết quả của quá trình nghiên cứu các yêu cầu của khách hàng chỉ là những dữ liệu thô. Nó không mô tả được những sản phẩm đề xuất, nó chỉ xác nhận những thứ nên hay không nên tạo ra và các đặc trưng mà khách hàng mong muốn. Các bản đặc tả lưu giữ các thông tin trên với các yêu cầu bắt buộc và đưa ra những định hình xem phần mềm sẽ là gì, sẽ làm gì, và trông nó như thế nào.

Định dạng của các bản đặc tả thay đổi rất nhiều. Đặc biệt, một số công ty mà các sản phẩm của họ dành cho chính phủ, cho vũ trụ, tài chính, và ngành công nghiệp được phẩm sử dụng một quy trình rất nghiêm khắc với nhiều sự kiểm tra ngặt nghèo và cân nhắc kỹ lưỡng. Kết quả thu được là một bản đặc tả vô cùng kỹ lưỡng, chi tiết được chốt lại, có nghĩa là không được phép thay đổi nó dưới mọi điều kiện. Mọi người trong nhóm phát triển biết chính xác chúng đang tạo nên cái gì.

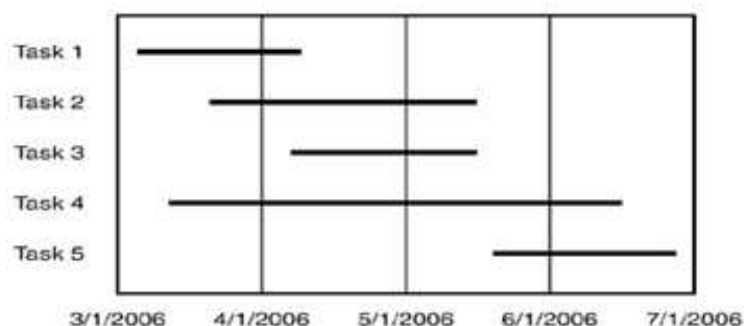
Đây là các nhóm phát triển phần mềm, thường tạo ra những sản phẩm ít bị chê trách, những người đã đưa ra những bản đặc tả trên bàn ăn (*on cocktail napkins*), nếu họ tạo ra tất cả chúng. Đây là những thuận lợi dễ nhận thấy, chúng rất mềm dẻo, nhưng lại chứa đựng đầy rủi ro. Và sản phẩm cuối cùng không được biết đến cho đến khi nó được tung ra thị trường.

### d) Kế hoạch làm việc (schedule)

Một phần rất quan trọng của quá trình sản xuất phần mềm là kế hoạch làm việc của nó. Là một dự án phát triển rất phức tạp với rất nhiều phần và nhiều người cùng tham gia.

Vì vậy, cần một số cơ cấu để theo dõi quá trình xử lý này. Nó có thể là một danh sách các nhiệm vụ đơn giản để hình thành nên biểu đồ *Gantt* (hình 2.2) để theo dõi chi tiết mỗi nhiệm vụ với phần mềm quản lý dự án.

Mục đích của *Schedule* là biết được công việc nào đã được hoàn thành, có bao nhiêu việc bị bỏ quên, và khi nào thì công việc được hoàn thành.



*Một biểu đồ Gantt biểu diễn các nhiệm vụ của một dự án tương phản với các đường ngang biểu diễn thời gian (horizontal timeline)*

#### e) Tài liệu thiết kế phần mềm

Một nhận thức sai lầm rất phổ biến là khi một lập trình viên tạo ra một chương trình, đơn giản là anh ta ngồi xuống và bắt đầu viết code. Điều này có thể xảy ra trong một cửa hàng phần mềm nhỏ và không chuyên nghiệp. Nhưng ở các công ty lớn, dù là với phần mềm nhỏ nhất cũng phải trải qua quá trình thiết kế để lập kế hoạch về cách mà phần mềm sẽ được viết. Trong cuốn sách này, phần mềm cũng yêu cầu được phác thảo và lập kế hoạch trước khi những đoạn mã đầu tiên được gõ, hoặc được xây dựng.

Những tài liệu mà những lập trình viên tạo ra biến đổi rất nhiều phụ thuộc vào công ty, dự án, và nhóm phát triển, những mục đích của chúng đều là lập kế hoạch và tổ chức mã được viết.

Đây là một danh sách một vài tài liệu thiết kế phần mềm rất phổ biến:

- Kiến trúc (*Architecture*): Một tài liệu mô tả cho toàn bộ thiết kế của phần mềm, bao gồm mô tả của tất cả các thành phần lớn và cách mà chúng gây ảnh hưởng tới các bộ phận khác.
- Sơ đồ luồng dữ liệu (*Data flow diagram*): Một sơ đồ chính thức biểu diễn dữ liệu xuyên suốt một chương trình. Thỉnh thoảng nó tham chiếu tới một *bubble chart* bởi vì nó sẽ gây chú ý hơn với các vòng tròn (*circle*) và các dòng (*line*)



- Sơ đồ chuyển trạng thái (*State transition diagram*): Một sơ đồ chính thức khác phá vỡ phần mềm ở trạng thái cơ bản, hoặc các điều kiện, và biểu diễn các phương tiện di chuyển từ trạng thái này đến trạng thái khác.

- Sơ đồ luồng (*Flow chart*): Các phương tiện truyền thống diễn đạt bằng hình ảnh mô tả luồng logic của phần mềm. Ngày nay, flowcharting không còn phổ biến, nhưng khi nó được sử dụng, viết code từ một flowchart chi tiết là một quá trình xử lý đơn giản.

- Mã chú giải (*commented code*): Có một cách nói cũ rằng bạn có thể viết code một lần, nhưng nó sẽ được đọc bởi bất kỳ ai, ít nhất là 10 lần. Bởi vậy, những lời chú giải hợp lý cho các đoạn code là rất quan trọng, vì vậy, các lập trình viên được giao nhiệm vụ bảo trì code coa thể dễ dàng hiểu được đoạn mã đó làm gì và làm như thế nào.

#### f) Tài liệu kiểm thử

Tài liệu kiểm thử được thảo luận chi tiết từ bài 17 đến bài 20. Nhưng nó vẫn được đề cập ở đây bởi vì nó là thành phần không thể thiếu để tạo nên một sản phẩm phần mềm. Với các lý do này, các lập trình viên phải lập kế hoạch và xây dựng tài liệu cho công việc của họ, tester phải hiểu rõ điều này. Không ai nghe thấy rằng một nhóm kiểm thử phần mềm phải tạo ra nhiều khả năng chuyển giao (*deliverables*) hơn các lập trình viên.

Đây là một danh sách *test deliverables* quan trọng:

- Kế hoạch kiểm thử (*test plan*) mô tả toàn bộ các phương thức được sử dụng để thay đổi phần mềm sao cho phù hợp với bản đặc tả và các yêu cầu của khách hàng. Nó bao gồm mục tiêu về chất lượng, các yêu cầu về tài nguyên, kế hoạch làm việc, những nhiệm vụ được giao, phương thức, và những thứ tương tự như thế (*and so forth*)

- Danh sách các trường hợp kiểm thử (*test case list*) Những phần sẽ được kiểm tra và mô tả từng bước chi tiết và sẽ được thực hiện theo để kiểm tra phần mềm.

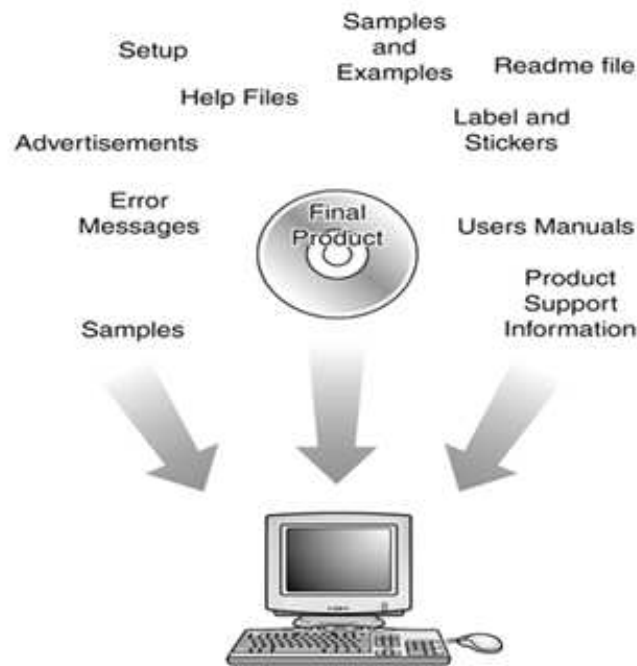
- Báo cáo lỗi (*bug reports*) mô tả các vấn đề được phát hiện nhờ các *test case*. Có thể chúng không được ghi ra giấy nhưng chúng sẽ được theo dõi qua database.

- Các công cụ kiểm thử và kiểm thử tự động (*Test tools and automation*) được mô tả chi tiết trong bài 13. Nếu nhóm của bạn sử dụng các công cụ tự động để kiểm thử phần mềm, thì hoặc là chúng được mua hoặc được tự viết, và chúng đưa ra kết quả bằng tài liệu.

#### g) Thành phần tạo nên một sản phẩm phần mềm

Như vậy, trong bài này bạn đã được tìm hiểu về các lỗi lực để tạo ra một sản phẩm phần mềm. Cũng cần phải nhận thức rằng khi một sản phẩm đã sẵn sàng để được đóng gói và mang đi thì không phải mỗi code được chuyển đi, còn rất nhiều những bộ phận khác đi

cùng với nó (hình 2.3). Bởi vì tất cả những phần này cũng được này cũng được thấy và sử dụng bởi khách hàng, chúng cũng cần được kiểm tra.



CD-ROM phần mềm là một trong rất nhiều phần tạo nên một sản phẩm phần mềm

Nhưng thật không may, các thành phần này thường xuyên bị bỏ qua trong quy trình kiểm tra phần mềm. Chắc hẳn rằng bạn cũng đã thử sử dụng các trợ giúp gắn liền với sản phẩm và thấy nó không được tiện dụng lắm thậm chí rất là tồi tệ. Hoặc có lẽ bạn đã kiểm tra yêu cầu hệ thống trên một sticker ở bên cạnh hộp phần mềm (*software box*) chỉ khám phá ra sau khi bạn mua phần mềm nhưng nó lại không hoạt động trên PC của bạn. Dường như, việc kiểm thử là rất đơn giản, nhưng không hẳn thế, hãy kiểm tra lại chúng một lần nữa trước khi đưa phần mềm ra thị trường. Bạn sẽ làm vậy chứ.

Sau khi đọc cuốn sách này, bạn sẽ được biết về những bộ phận không phải là phần mềm này (*non-software pieces*) và cách để kiểm tra chúng một cách hợp lý. Sau đó, hãy giữ lại danh sách này trong đầu như một ví dụ rằng một sản phẩm phần mềm thì không chỉ là code:

Help files	User's manual
Samples and examples	Labels and stickers
Product support info	Icons and art
Error messages	Ads and marketing material

ĐỪNG QUÊN KIỂM TRA NHỮNG THÔNG ĐIỆP LỖI: thông điệp lỗi (error message) là những phần dễ bị bỏ qua nhất trong một sản phẩm phần mềm. Các lập trình viên, không phải những người thực sự có kinh nghiệm, mà cụ thể là những người viết ra chúng. Hiếm khi họ lập kế hoạch mà thường sửa chữa dần chương trình trong khi kiểm tra các lỗi. Vì vậy, thật khó khăn khi các tester muốn tìm thấy và hiển thị đầy đủ các lỗi. Đừng đưa ra những thông điệp lỗi gây sợ e sợ trong phần mềm của bạn.

Error: Keyboard not found. Press F1 to continue.

Can't instantiate the video thing.

Windows has found an unknown device and is installing a driver

for it.

A Fatal Exception 006 has occurred at 0000:00000007.

# Thực trạng của quá trình kiểm thử phần mềm

## Thực trạng của quá trình kiểm thử phần mềm

Trong phần 1, bạn đã được tìm hiểu những khái niệm cơ bản về kiểm thử phần mềm và quy trình phát triển phần mềm. Những thông tin đã biểu diễn trong các bài này chỉ là ở mức tổng quan, và cho bạn cái nhìn về cách mà các dự án phần mềm có thể hoạt động. Nhưng thật không may, trong thế giới thật, bạn sẽ không bao giờ thấy một phần mềm hoàn hảo theo một bất kỳ một mô hình phát triển phần mềm nào. Bạn sẽ không thể đưa ra được một bản đặc tả chi tiết hoàn toàn đầy đủ mà khách hàng cần và bạn cũng sẽ không đủ thời gian để làm tất cả những bài kiểm tra mà bạn cần phải làm. Không có vấn đề gì cả. Nhưng để trở thành một tester làm việc có hiệu quả, bạn cần phải biết tưởng tượng ra quy trình phần mềm làm việc như thế nào để đạt được mục đích.

Mục đích của bài này là làm dịu đi tác động của chủ nghĩa lý tưởng lên quá trình kiểm tra thực tế trên phần mềm. Nó sẽ giúp bạn thấy rằng, trong thực tế, **sự thỏa hiệp và nhượng bộ phải xuyên suốt vòng đời phát triển phần mềm**. Nhiều điều trong những sự thỏa hiệp này là liên quan trực tiếp đến nỗ lực kiểm thử phần mềm. Những lỗi mà bạn tìm thấy và những vấn đề mà bạn ngăn chặn, tất cả đều có ảnh hưởng đặc biệt tới dự án của bạn. Sau khi đọc bài này, bạn sẽ thu nhận được rất nhiều quy tắc, sự tiếp xúc, và những khả năng hồi đáp mà tester cần và hi vọng rằng bạn sẽ đưa ra những quyết định giúp kiến tạo ra một sản phẩm phần mềm.

Trọng tâm của bài này bao gồm:

- Tại sao phần mềm không bao giờ là hoàn hảo
- Tại sao kiểm thử phần mềm không phải là một vấn đề mang tích chất khuôn mẫu
- Những thuật ngữ phổ biến được sử dụng trong kiểm thử phần mềm

## Phương châm của việc kiểm thử

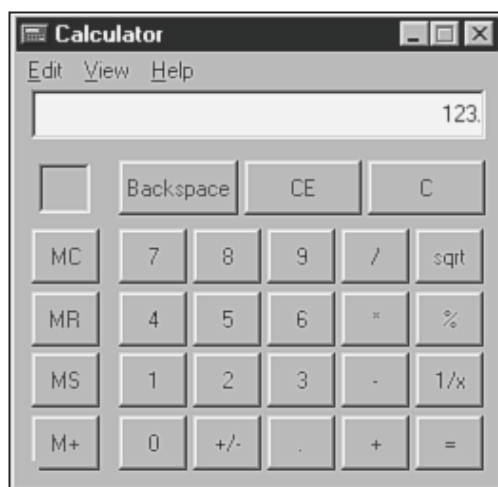
Đoạn đầu của bài này là một danh sách các phương châm hoặc các chân lý. Hãy coi chúng giống như “*quy tắc đi đường*” (“*rules of the road*”) hoặc “*chân lý của cuộc sống*” (“*facts of life*”) dành cho quá trình kiểm thử hoặc phát triển phần mềm. Mỗi một phương châm này là một sự hiểu biết nho nhỏ giúp họ đặt một số khía cạnh của toàn bộ quá trình xử lý vào viễn cảnh tương lai.

a) Tầm quan trọng của việc kiểm thử đầy đủ một chương trình:

Là một tester, bạn có thể tin rằng bạn có khả năng tiếp cận với một khía cạnh của phần mềm, kiểm tra nó, tìm ra tất cả các lỗi, và đảm bảo rằng phần mềm là hoàn hảo. Nhưng thật không may, điều này là không thể được, thậm chí là với một chương trình rất đơn giản, vì 4 lý do sau:

- Số lượng các dữ liệu có thể là đầu vào là rất lớn
- Số lượng các dữ liệu có thể đưa ra cũng vô cùng lớn
- Số lượng các “lỗi đi” trong phần mềm là rất lớn
- Đặc tả phần mềm có tính chất chủ quan. Bạn có thể nói rằng lỗi là những khuyết điểm dưới con mắt của độc giả.

Tất cả các trường hợp trên nếu kết hợp cùng nhau, bạn sẽ thu được một *tập các điều kiện vô cùng lớn đến mức không thể thử hết được*. Nếu bạn không tin điều này thì có thể xem xét trong hình 3.1, phần mềm Microsoft Windows Calculator.



Thậm chí một chương trình đơn giản như Windows Calculator cũng quá phức tạp để kiểm thử đầy đủ

- Khi bạn được phân công kiểm tra phần mềm *Windows Calculator*. Bạn quyết định là sẽ bắt đầu kiểm tra phép cộng. Bạn thử nghiệm xem  $1+0=?$  Bạn nhận được câu trả lời là 1. Phép kiểm tra này cho kết quả đúng. Sau đó, bạn tiếp tục kiểm tra  $1+1=?$  Kết quả nhận được là 2. Bạn đi bao xa? Máy tính chấp nhận một số có 32 chữ số, vì vậy, bạn phải cố gắng thử tất cả các khả năng có thể:

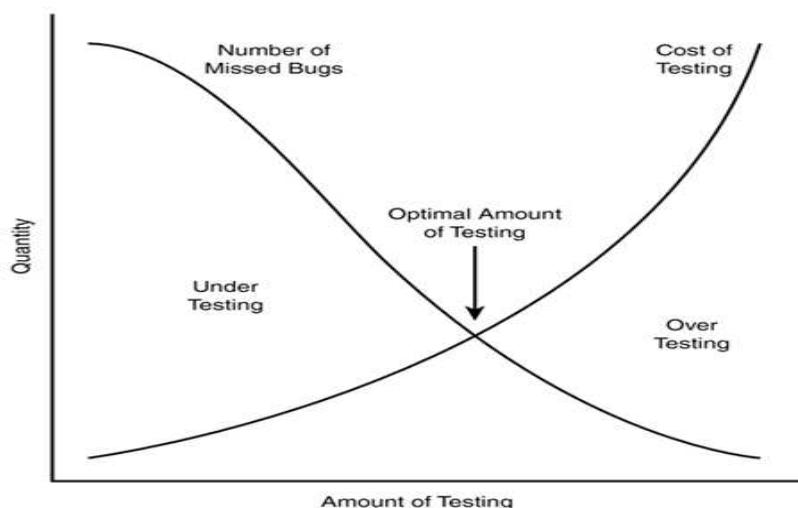
[illegible]

Sau lần đầu tiên, bạn hoàn thành chuỗi số trên, bạn cũng có thể thử trên các phép toán  $2+0=?$ ,  $2+1=?$ ,  $2+2=?$ ... Và cứ tiếp tục như vậy. Cuối cùng, bạn sẽ phải thử nghiệm trên phép tính:



thể nào để sáng suốt lựa chọn những quyết định ít rủi ro nhất. Điều này buộc *tester* phải xác định được đâu là vấn đề quan trọng và đâu là vấn đề không quan trọng.

Hình 3.2 mô tả mối quan hệ giữa số lượng các trường hợp test với số lượng các lỗi được tìm thấy. Nếu bạn cố thử kiểm tra mọi thứ, chi phí có thể tăng lên đột ngột và những lỗi bị bỏ quên sẽ giảm xuống thấp nhất, nhưng cũng sẽ không còn chi phí để tiếp tục dự án. Nếu bạn cắt giảm công việc kiểm thử thì chi phí cho nó sẽ ít, nhưng bạn sẽ bỏ quên rất nhiều lỗi. Mục đích là bạn phải lọc ra số các trường hợp kiểm thử tối ưu, để đảm bảo bạn không phải kiểm thử quá nhiều hay quá ít các trường hợp.



Mọi dự án phần mềm đều có một điểm nỗ lực kiểm thử tối ưu

Bạn sẽ được tìm hiểu làm thế nào để thiết kế và lựa chọn các kịch bản kiểm thử (*test scenarios*) sao cho ít rủi ro nhất và quá trình kiểm tra là tối ưu nhất.

### c) Quá trình kiểm thử không thể biểu diễn những lỗi không tồn tại

Hãy nghĩ về điều này trong chốc lát. Bạn là một “kẻ hủy diệt” (*exterminator*) với bài kiểm tra các lỗi. Bạn xem xét hàng giờ và tìm ra dấu vết của các lỗi, lỗi này có thể vẫn đang tồn tại (*live bug*), đã được sửa (*dead bug*), hoặc còn đang tiềm ẩn (*nest*). Bạn có thể nói một cách an toàn rằng “*the house has bugs*”

Bạn đến thăm một “*house*” khác. Lần này, bạn không tìm thấy dấu vết của lỗi. Bạn hãy nhìn vào tất cả những địa điểm rõ ràng (*obvious place*) và tìm xem không có dấu hiệu nào của sự tàn phá. Có lẽ bạn nên tìm một vài lỗi đã từng được xử lý hoặc tiềm ẩn từ lâu, nhưng bạn hãy coi như không thấy gì cả. Có thể bạn tuyên bố một cách chắc chắn rằng “*the house is bug free*”? Không. Kết luận cuối cùng, có thể bạn không tìm thấy một *live bug* nào cả. Ngược lại, bạn tháo gỡ hoàn toàn *the house* thành *foundation*, bạn không thể chắc chắn rằng: bạn không bỏ quên một số lỗi đơn giản.

Tester làm việc chính xác như một “kẻ hủy diệt”. Nếu có thể biểu diễn những lỗi đang tồn tại, nhưng không thể biểu diễn những lỗi không tồn tại. Bạn có thể thực hiện bài kiểm tra của bạn, tìm và báo cáo các lỗi, nhưng bạn có thể kết luận rằng: lỗi không được tìm thấy nữa. Bạn có thể tiếp tục kiểm tra và khả năng tìm thấy lỗi là lớn hơn.

#### **d) Những lỗi được tìm thấy và những lỗi không thể tìm thấy (The More Bugs You Find, the More Bugs There Are)**

**Thậm chí, có rất nhiều điểm tương đồng giữa real bug và software bug. Cả hai loại này đều cần đưa vào một nhóm. Thường thì một tester sẽ chạy phần mềm như thể nó không có một lỗi nào. Sau đó, anh ta sẽ tìm ra một lỗi rồi những lỗi khác, lỗi khác nữa. Có một vài lý do cho điều này:**

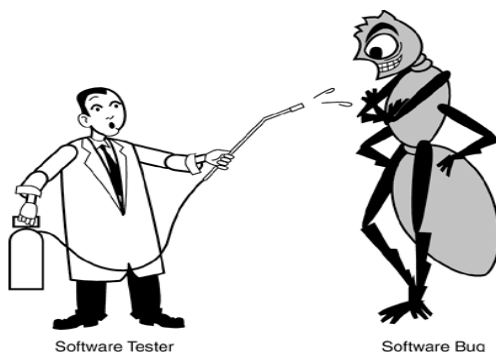
- Các lập trình viên có những ngày thật tội tệ: Giống như tất cả chúng ra, những người lập trình có thể có những lúc không được minh mẫn lắm. Vào một thời điểm này code có thể được viết rất hoàn hảo, nhưng lúc khác anh ta lại viết code rất cẩu thả. Một lỗi có thể là một dấu hiệu *tell – tale* rất quen thuộc.
- Một lập trình viên thường xuyên mắc những lỗi tương tự nhau: Ai cũng có những thói quen. Một lập trình viên thiên về một loại lỗi nào đó mà thường xuyên mắc đi mắc lại.
- Một số lỗi thật sự nguy hiểm như đỉnh của một tảng băng chìm: Thường thì trong các bản thiết kế và kiến trúc của phần mềm đều ẩn chứa một số vấn đề chưa được phát hiện. Tester đã tìm được một số lỗi mà dường như nó không liên quan đến nhau. Nhưng không hẳn thế, những lỗi này lại có những quan hệ mật thiết với nhau và đều xuất phát từ một lý do chính vô cùng quan trọng.

**Vấn đề quan trọng bây giờ là cần chú ý rằng ngược với ý tưởng “bugs follow bugs” này cũng có thể coi là đúng. Nếu bạn không thể tìm ra lỗi của phần mềm thì cũng không có vấn đề gì. Sẽ rất thuận lợi nếu các đặc trưng mà bạn kiểm tra được viết một cách trong sáng và quả thực sẽ có một vài điều nếu như bất kỳ một lỗi nào được tìm ra.**

#### **e) Nghịch lý về thuốc trừ sâu (The Pesticide Paradox)**



Vào năm 1990, Boris Beizer, trong cuốn sách *Software Testing Techniques*, tái bản lần 2, đã xây dựng thuật ngữ **Pesticide Paradox** để mô tả một hiện tượng bạn kiểm thử phần mềm. Những điều tương tự với hiện tượng này đã xảy ra khi bạn dùng pesticides để diệt sâu bọ (mô tả trong hình 3.3). Nếu bạn liên tục dùng một loại pesticide giống nhau, sâu bọ sẽ kháng cự lại thuốc, và pesticide không còn hiệu quả nữa.



Phần mềm đã phải trải qua những phép thử lặp đi lặp lại tương tự nhau để chống lại các lỗi

Hãy nhớ về mô hình xoắn ốc của quy trình phát triển phần mềm được mô tả trong bài 2. Quá trình kiểm thử cũng phải lặp đi lặp lại mỗi lần quanh vòng lặp. Với mỗi lần lặp lại, tester nhận phần mềm để kiểm tra và chạy các trường hợp kiểm thử của họ. Cuối cùng, ngoài một vài trường hợp phần mềm chạy đúng yêu cầu, thì các trường hợp kiểm tra của tester sẽ tìm ra và phơi bày các lỗi. Nhưng ở lần lặp sau, nếu tester vẫn tiếp tục chạy các trường hợp kiểm thử này, chúng sẽ không giúp tìm ra lỗi mới.

Để cách vượt qua pesticide paradox, tester phải viết thêm các trường hợp kiểm thử khác nữa, và tìm những cách tiếp cận mới để kiểm tra chương trình và tìm ra nhiều lỗi hơn.

f) Không phải tất cả các lỗi mà bạn phát hiện sẽ được sửa

Một trong những điều thật sự đáng buồn của kiểm thử phần mềm là sau tất cả những nỗ lực cố gắng làm việc của bạn, không phải tất cả các lỗi bạn phát hiện ra sẽ được sửa. Nhưng điều này cũng không gây thất vọng bởi vì nó không có nghĩa rằng bạn đã làm sai điều gì khi cố gắng thực hiện mục đích của mình, cũng không có nghĩa rằng bạn cùng với cả đội của bạn sẽ phải chấp nhận giao cho khách hàng một sản phẩm kém chất lượng. Tuy nhiên, nó có nghĩa rằng, bạn sẽ cần dựa trên những cặp tiêu chí về nghề tester đã được liệt kê trong bài 1. Bạn và đội của bạn cần mô tả lại những quyết định dựa trên sự rủi ro cho riêng từng lỗi và cho tất cả các lỗi, để đưa ra quyết định cái nào sẽ được sửa và cái nào thì không.

## **Có một số lý do khiến một số lỗi không được fix:**

- Không đủ thời gian: Trong mọi dự án luôn có rất nhiều feature của phần mềm, nhưng bạn lại có quá ít người để viết mã và kiểm thử chúng, và cũng không đủ khả năng để thay đổi kế hoạch làm việc cho đến khi kết thúc. Nếu bạn đang làm việc cho một chương trình đòi hỏi những thử thách lớn, mà thời hạn hoàn thành đã sắp đến thì bạn buộc phải bỏ qua một số lỗi
- Nó không hẳn là một lỗi: Có lẽ bạn cũng đã từng nghe thấy thành ngữ sau: “*it’s not a bug, it’s a feature!*” Không phải là hiếm những trường hợp: hiểu sai, lỗi do quá trình kiểm tra, hoặc đặc tả thay đổi dẫn đến kết quả có thể phát sinh lỗi trong tương lai
- Có quá nhiều rủi ro khi sửa lỗi: Thật không may điều này là quá thường xuyên xảy đến. Phần mềm có thể rất dễ hỏng, các bộ phận gắn kết chặt chẽ với nhau, và đôi khi chúng giống như “*món mì Ý*”. Có thể bạn sửa một lỗi lại khiến một lỗi khác xuất hiện. Dưới áp lực về thời gian hoàn thành sản phẩm, dưới một lịch trình kín đặc, thì có lẽ thay đổi phần mềm là một quyết định chứa quá nhiều rủi ro. Có lẽ tốt hơn hết là bỏ qua những lỗi có thể chấp nhận được để tránh phát sinh những lỗi mới, những rủi ro mới.
- Nó không đáng để phải sửa: điều này nghe có vẻ bất hợp lý, nhưng nó là sự thật. Những lỗi hiếm khi xuất hiện hoặc những lỗi xuất hiện trong những *feature* ít sử dụng thì có thể bỏ qua được. Những lỗi “*work-around*”, tức là có cách để người sử dụng có thể ngăn chặn hoặc tránh được lỗi, thì thường không được sửa. Tất cả các quyết định phải mang tính chất thị trường dựa trên độ rủi ro.

**Quá trình xử lý các quyết định này thường bao gồm các tester, các project manager, các coder. Mỗi người sẽ có những cách nhận định về một viễn cảnh xảy ra khi một số lỗi không được sửa. Nếu lỗi không được fix, tester hiểu khách hàng sẽ phải gánh chịu những hậu quả như thế nào. Project manager có tầm nhìn chiến lược và đoán nhận được những hậu quả có thể xảy ra với dự án khi lỗi không được giải quyết. Và coder hiểu được rằng nếu fix lỗi này thì chi phí cho việc đó sẽ lớn như thế nào. Dựa vào đó, họ sẽ đưa ra những lý do tại sao họ nên sửa hoặc không nên sửa các lỗi đó.**

## **CHUYỆN GÌ SẼ XẢY ĐẾN KHI BẠN ĐƯA RA MỘT QUYẾT ĐỊNH SAI:**

**Hãy nhớ lại về lỗi mà hãng Intel Pentium mô tả trong bài 1. Hãng Intel đã tìm ra lỗi này trước khi sản phẩm được tung ra thị trường, nhưng đội phát triển sản phẩm đã quyết định rằng: nó là một lỗi quá nhỏ và không đáng để phải sửa. Họ có một lịch làm việc quá dày đặc và đã đến hạn hoàn thành sản phẩm. Vì vậy, họ đã quyết định việc sửa lỗi này sẽ được thực hiện trong phiên bản sau của chip.**

**Nhưng thật không may, lỗi này đã bị khách hàng phát hiện ra. Trong một số khía cạnh của phần mềm, có thể có tới hàng trăm lỗi không được sửa bởi vì họ nhận thấy rằng hiệu quả tích cực của nó là không lớn. Vậy rất khó để có thể nói rằng các quyết định này là đúng hay sai.**

**g) Một lỗi có tồn tại nhưng không ai phát hiện thì có phải là lỗi không? (When a Bug's a Bug Is Difficult to Say)**

**Nếu có một vấn đề trong phần mềm, nhưng không một ai phát hiện ra nó, không phải lập trình viên, không phải một tester, và thậm chí là một khách hàng nào đó, thì nó có được gọi là lỗi không?**

**Một nhóm các tester ở trong một phòng và hỏi chúng tôi câu hỏi này. Bạn sẽ phải thảo luận về vấn đề này. Ai cũng có ý kiến riêng của mình và có thể là bạn cũng thế. Vấn đề ở đây là không có câu trả lời xác định. Câu trả lời phụ thuộc vào bạn và đội phát triển của bạn với việc đưa ra những quyết định tốt nhất cho mình.**

**Với mục đích của cuốn sách này, bạn hãy tham khảo những quy tắc để xác định một lỗi trong bài 1:**

1. Phần mềm không thực hiện một số thứ giống như mô tả trong bản đặc tả phần mềm
2. Phần mềm thực hiện một số việc mà bản đặc tả yêu cầu nó không được thực hiện
3. Phần mềm thực hiện một số chức năng mà bản đặc tả không đề cập tới
4. Phần mềm không thực hiện một số việc mà bản đặc tả không đề cập tới, nhưng là những việc nên làm
5. Trong con mắt của người kiểm thử phần mềm là khó hiểu, khó sử dụng, chậm đối với người sử dụng

Quy tắc này sẽ giúp chúng ta lọc ra những tình huống khó xử để đưa ra quyết định. Có một cách khác để suy xét nó. Không phải là hiếm những trường hợp mà 2 người sử dụng có những nhận xét hoàn toàn trái ngược nhau về vấn đề chất lượng phần mềm. Một người thì nói rằng chương trình như vậy là không thể chấp nhận và người còn lại thì quả quyết rằng chương trình này là hoàn hảo. Có thể cả hai đều đúng? Câu trả lời là một người sử dụng sản phẩm theo cách mà rất nhiều lỗi bị bộc lộ. Còn người kia thì không.

**Chú ý:** Lỗi không được khám phá hoặc chưa từng được chú ý thì thường được gọi là lỗi ngầm (*latent bug*)

Nếu điều này quá khó hiểu thì cũng đừng lo lắng. Hãy đem nó ra thảo luận với đồng nghiệp trong đội kiểm thử của bạn và cố gắng hiểu điều mà họ nghĩ. Hãy lắng nghe

những ý kiến khác, kiểm tra ý tưởng của họ và định hình lại suy nghĩ của chính mình. Hãy nhớ lại một câu hỏi quen thuộc: “nếu một cái cây đổ trong rừng và không ai nghe thấy gì cả, vậy nó có tạo ra âm thanh không?” (*"If a tree falls in the forest and there's no one there to hear it, does it make a sound?"*).

#### **h) Xây dựng bản đặc tả phần mềm là công việc không bao giờ kết thúc**

**Phát triển phần mềm có một vấn đề. Ngành công nghiệp này đang phát triển quá nhanh: năm ngoái nó có thể là một sản phẩm sắc bén, nhưng năm nay nó đã trở nên lỗi thời. Tại cùng một thời điểm, phần mềm có quy mô lớn hơn, có nhiều feature hơn và tương đối phức tạp, dẫn đến kế hoạch phát triển sẽ lâu hơn. Có 2 vấn đề đối lập nhau, và kết quả là liên tục phải thay đổi bản đặc tả sản phẩm.**

**Không có cách nào khác để phản ứng lại sự thay đổi mau lẹ của bản đặc tả. Bạn cho rằng, sản phẩm của bạn đã bị khóa và chúng ta tuyệt đối không thể thay đổi bản đặc tả sản phẩm. Bạn đã đi được nửa chặng đường trong kế hoạch phát triển 2 năm của sản phẩm, đối thủ chính của bạn thì đã tung ra thị trường một sản phẩm hoàn toàn tương tự với sản phẩm của bạn. Mà thậm chí một vài feature rất tuyệt vời mà sản phẩm của bạn không có được. Liệu bạn có nên tiếp tục với bản đặc tả của mình và giao cho khách hàng một sản phẩm thua kém hơn không? Hay đội phát triển dự án của bạn nên tập hợp lại, suy ngẫm lại về những feature của sản phẩm, viết lại bản đặc tả và làm việc trên một sản phẩm được sửa lại? Trong hầu hết các trường hợp, những nhà kinh doanh sáng suốt sẽ tuyên bố sau cùng.**

**Là một tester, bạn phải thừa nhận rằng bản đặc tả sẽ thường xuyên thay đổi. Các feature sẽ được thêm vào mà nó không hề nằm trong kế hoạch kiểm thử. Các feature sẽ thay đổi và thậm chí là bị xóa hoàn toàn khi bạn đã kiểm tra và sẵn sàng báo cáo lỗi về nó. Điều này hoàn toàn có thể xảy ra. Bạn sẽ được tìm hiểu các kỹ thuật linh hoạt để lập kế hoạch và thực thi việc kiểm thử trong phần còn lại của cuốn sách.**

#### **k) Tester không phải là thành viên được mọi người chờ đợi trong một dự án**

**Hãy nhớ lại mục đích của quá trình kiểm thử phần mềm là gì? Mục đích của một tester là tìm ra lỗi, tìm thấy chúng sớm nhất có thể, và chắc chắn rằng chúng phải được sửa.**

**Công việc của bạn là xem xét thật kỹ lưỡng và phê bình công việc của các đồng nghiệp của bạn, phát hiện những vấn đề của công việc, và phải thực hiện công khai những gì bạn tìm thấy. Và bạn sẽ phải cố gắng chiến thắng trong các cuộc tranh luận với các đồng nghiệp.**

## **Hãy giữ thái độ hòa bình với những đồng nghiệp trong đội của bạn:**

- Phát hiện lỗi thật sớm: Dĩ nhiên, đây là công việc của bạn, và bạn phải kiên trì làm công việc này. Và dĩ nhiên, nếu bạn phát hiện một lỗi nguy hiểm trước 3 tháng thì tốt hơn là 1 ngày trước khi đến thời điểm tung sản phẩm ra thị trường.
- Giữ thái độ hăng hái, nhiệt tình: Tốt thôi, bạn thật sự yêu công việc của mình. Bạn sẽ cảm thấy phấn khích khi bạn tìm được một lỗi khủng khiếp. Nhưng nếu bạn huênh hoang dồn ép lập trình viên và nói với anh ta rằng bạn vừa mới tìm được một lỗi kinh khủng nhất (*nastiest bug*) trong suốt quá trình làm việc của bạn, thì chắc hẳn rằng anh ta sẽ cảm thấy khó chịu.
- Đừng chỉ báo cáo những thông tin xấu: Nếu bạn đã phát hiện ra một đoạn mã chứa đầy lỗi, hãy nói cho mọi người biết, dù bạn sẽ bị phản đối. Bởi vì nếu bạn chưa từng phát hiện ra lỗi của các lập trình viên, mọi người cũng sẽ tránh xa bạn.

### **i) Kiểm thử phần mềm là một công việc đòi hỏi tính kỷ luật**

Kiểm thử phần mềm là công việc được thực hiện sau khi có sản phẩm. Các sản phẩm phần mềm và không phức tạp. Số lượng người với máy tính sử dụng phần mềm là bị giới hạn. Và, một số ít lập trình viên trong đội dự án của bạn có khả năng gỡ lỗi cho mỗi đoạn mã của người khác. Các lỗi là một vấn đề không tốt. Chúng xuất hiện và sớm được sửa chữa thì chi phí cho chúng sẽ không nhiều. Thường thì các tester không được huấn luyện và họ vẫn phát huy khả năng của họ trong các dự án sau để làm thay đổi nhiều thứ.

Hãy nhìn xem sản phẩm phần mềm cần được giúp đỡ và bạn sẽ nhìn thấy một danh sách các tester. Ngành công nghiệp phần mềm đang phát triển vượt bậc với mũi nhọn là đội ngũ tester chuyên nghiệp. Bởi vì hiện tại, người ta đã mất quá nhiều chi phí để xây dựng lên những phần mềm kém chất lượng.

Thật là tội tề, không phải mọi công ty đều thống nhất quan điểm đó. Nhiều *computer game* và những công ty phần mềm nhỏ vẫn thường xuyên sử dụng những mô hình phát triển lỏng lẻo như *big-bang* hoặc *code-and-fix*. Nhưng bây giờ, nhiều phần mềm được phát triển và luôn tuân thủ kỷ luật. Các tester trở thành lực lượng lòng cốt, những thành viên sống còn trong nhiệm vụ của họ.

Đây sẽ là một vấn đề lớn, nếu bạn là thấy hứng thú với kiểm thử phần mềm. Nó đã trở thành một nghề nghiệp được nhiều người lựa chọn và cần phải được đào tạo, làm việc có kỷ luật và thúc đẩy sự tiến bộ.

## Các định nghĩa và thuật ngữ kiểm thử phần mềm

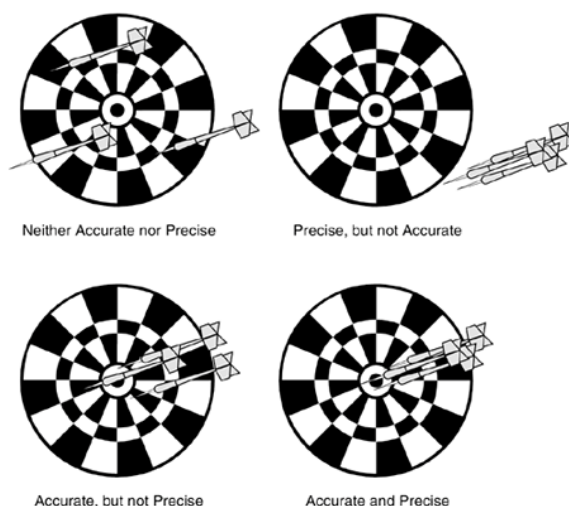
Bài này bao gồm một danh sách các thuật ngữ và các định nghĩa. Các thuật ngữ này mô tả những khái niệm nền tảng về quá trình phát triển phần mềm và kiểm thử phần mềm. Bởi vì chúng thường rất lộn xộn và được sử dụng không hợp lý, chúng được định nghĩa ở đây như một cặp để giúp bạn hiểu ý nghĩa thật sự chúng và sự khác nhau giữa chúng. Hãy ý thức rằng nhiều người không bằng lòng về ngành công nghiệp phần mềm với những khái niệm của nhiều công ty, được phổ biến rộng rãi, (đó là các thuật ngữ). Là một tester, bạn nên thường xuyên làm rõ ràng ý nghĩa của các thuật ngữ mà đội của bạn sử dụng. Thường thì đây là cách tốt nhất để một khái niệm được đồng tình hơn là bạn phải cố gắng để mọi người chấp nhận rằng thuật ngữ đó là đúng.

### a) Precision (tập chung) và accuracy (chính xác)

Là một tester, điều quan trọng là bạn phải biết sự khác nhau giữa *precision* và *accuracy*. Hãy coi như bạn đang kiểm thử phần mềm *Calculator*. Bạn có nên kiểm tra rằng các câu trả lời phần mềm trả về cho bạn là *precise* hay *accurate*? Hay cả hai? Nếu lịch làm việc của dự án buộc bạn phải đưa ra những quyết định dựa trên sự rủi ro và bạn chỉ được chọn một trong những từ này, khi đó, bạn sẽ chọn từ nào?

Nếu phần mềm mà bạn kiểm tra là mô phỏng một chò chơi như bóng chày hoặc mô phỏng một chuyến bay. Trước hết, bạn có nên kiểm tra khả năng *precision* của nó hoặc khả năng *accuracy* của nó?

Hình dưới mô tả một cách hình ảnh về 2 thuật ngữ này. Mục đích của trò phóng phi tiêu này (*dart game*) là ném trúng vào tâm của tấm bảng. Phi tiêu trên tấm bảng phía trên bên trái là không *precise* mà cũng không *accurate*. Chúng không được tập chung lại cùng nhau mà cũng không trúng tâm của tấm bảng.



*Những cây phi tiêu trên tấm bảng giải thích về sự khác nhau giữa 2 thuật ngữ Precision và Accuracy*

Tấm bảng phía trên, bên phải biểu diễn những cây phi tiêu *precise* nhưng không *accurate*. Chúng tập chung lại thành một nhóm, vì vậy người ném đã thực hiện *precision*, nhưng anh ta không *accurate* bởi vì các cây phi tiêu không trúng đích.

Tấm bảng ở phía dưới, bên trái là một ví dụ về sự *accuracy* nhưng lại thiếu sự *precision*. Những cây phi tiêu này đều trúng đích. Người ném đã ném trúng mục tiêu mà anh ta nhắm đến, nhưng những cây phi tiêu không nằm tập chung tại một vị trí.

Tấm bảng ở phía dưới, bên phải là sự kết hợp hoàn hảo của *precision* và *accuracy*. Các cây phi tiêu tập chung tại một chỗ và đều trúng đích.

Phần mềm bạn kiểm tra cần có đạt *precise* hay *accurate* hay không phụ thuộc rất nhiều vào cái đích mà sản phẩm cuối cùng của bạn hướng tới. Phần mềm *Calculator* giống là phần mềm đòi hỏi cả hai yêu cầu đều phải đạt được. Nhưng, cũng có thể quyết định rằng *Calculator* sẽ chỉ đạt yêu cầu về sự chính xác (*accurate*) và sự tập chung (*precise*) đạt tới chữ số thập phân thứ 5. Sau tất cả, sự tập chung (*precision*) có thể thay đổi. Tùy thuộc vào việc *tester* nhận thức về bản đặc tả như thế nào. Họ có thể tự thiết kế những bài kiểm tra của họ để chứng minh những nhận định của họ.

## **b) Verification (sự kiểm tra) và Validation (sự xác nhận)**

*Verification* và *Validation* thường được sử dụng thay thế cho nhau nhưng thực chất chúng là các khái niệm khác nhau. Sự khác nhau này rất quan trọng trong kiểm thử phần mềm.

*Verification* là quy trình xác nhận rằng một số khía cạnh của phần mềm là phù hợp với bản đặc tả của nó. *Validation* là quy trình xác nhận rằng phần mềm phù hợp với yêu cầu của người sử dụng. Chúng có vẻ như rất giống nhau, nhưng một lời giải thích cho các vấn đề kính thiên văn không gian *Hubble* (*Hubble space telescope*) sẽ giúp biểu diễn sự khác nhau này.

Vào tháng 4 năm 1990, Kính thiên văn không gian *Hubble* (*Hubble space telescope*) được đưa vào quỹ đạo quanh trái đất. Là một thiết bị phản chiếu, *Hubble* sử dụng một tấm gương lớn như một phương tiện chính để khuếch đại đối tượng mà nó nhằm tới. Quá trình chế tạo tấm gương này là đòi sự *chính xác* và *tập chung* tuyệt đối. Kiểm tra tấm gương rất khó, từ khi chiếc kính thiên văn này được thiết kế để sử dụng trong không gian và người ta không thể xác định được vị trí, thậm chí là nó có tầm nhìn xuyên suốt ngay cả khi nó vẫn ở trên trái đất. Với những lý do này thì chỉ có một cách kiểm tra tốt nhất là đo đạc cẩn thận tất cả các thuộc tính của nó và so sánh với những tiêu chuẩn đã được chỉ ra. Quá trình kiểm tra này đã được thực thi và *Hubble* được tuyên bố là đã sẵn sàng.

Nhưng thật không may, ngay sau khi nó được đưa vào quỹ đạo hoạt động, các bức ảnh nó gửi về không hề có trung tâm. Tổ chức điều tra đã khám phá ra rằng tấm gương đã được chế tạo không hợp lý. Khi ở trên mặt đất, tấm gương này đã được sản xuất theo đúng bản đặc tả, nhưng bản đặc tả này lại sai. Tấm gương vô cùng *precise* nhưng nó không *accurate*. Quá trình kiểm tra đã xác nhận rằng tấm gương được sản xuất đã đáp ứng được sự kiểm tra của bản đặc tả (*spec verification*), nhưng nó không xác nhận được rằng nó đáp ứng được yêu cầu cơ bản (*original requirement validation*).

Năm 1993, một phái đoàn trên tàu con thoi đã sửa kính thiên văn *Hubble* bằng cách cài đặt một “*corrective len*” để lấy lại trung tâm của những bức ảnh được chụp bởi *Hubble*.

Mặc dù không có một ví dụ về phần mềm, *verification* và *validation* áp dụng tốt như nhau với quá trình kiểm thử. Chưa từng có bản đặc tả nào là đúng. Nếu bạn thay đổi bản đặc tả và thông qua sản phẩm cuối cùng, thì bạn sẽ tránh được những vấn đề như với chiếc kính thiên văn *Hubble*.

### c) Quality (chất lượng) và reliability (sự đáng tin cậy)

Trong cuốn từ điển của trường cao đẳng *Merriam-Webster* đã định nghĩa rằng *quality* là “độ đo sự hoàn hảo” hoặc “sự vượt trội về thứ hạng”. Nếu sản phẩm phần mềm có chất lượng cao, nó sẽ đáp ứng được nhu cầu của khách hàng. Khách hàng sẽ cảm thấy sản phẩm hoàn hảo và nó sẽ được sắp thứ hạng cao hơn trong danh sách lựa chọn của khách hàng.

*Tester* có thể cảm thấy 2 khái niệm *quality* và *reliability* là gần như nhau. Họ cảm thấy rằng nếu như họ có thể kiểm tra một chương trình cho đến khi nó chạy ổn định và có thể tin tưởng được (*reliability*). Khi đó, họ có thể quả quyết rằng sản phẩm đã đạt chất lượng tốt. Nhưng thật không may, điều này không hẳn đã đúng. *Reliability* chỉ là một khía cạnh của *quality*.

Quan niệm về *quality* của người sử dụng phần mềm có thể bao gồm cả sự thoải mái của các *feature*, sản phẩm có khả năng chạy trên cả những PC cũ, dịch vụ hậu mãi của các công ty phần mềm, và thường bao gồm cả giá cả của sản phẩm. Sự tin tưởng hoặc cách thức mà phần mềm thâm nhập vào dữ liệu của khách hàng, có thể là rất quan trọng, nhưng không phải lúc nào cũng thế.

Chắc rằng, với một chương trình có chất lượng cao và đáng tin cậy, thì *tester* phải kiểm tra và thông qua trong suốt quá trình phát triển sản phẩm.

### d) Testing (Kiểm thử) và quality assurance (đảm bảo chất lượng) (QA)

Cặp khái niệm cuối cùng là *testing* và *quality assurance* (có thể viết tắt là QA). Hai thuật ngữ này, một cái thường được sử dụng để mô tả nhóm hoặc quá trình kiểm tra và xác



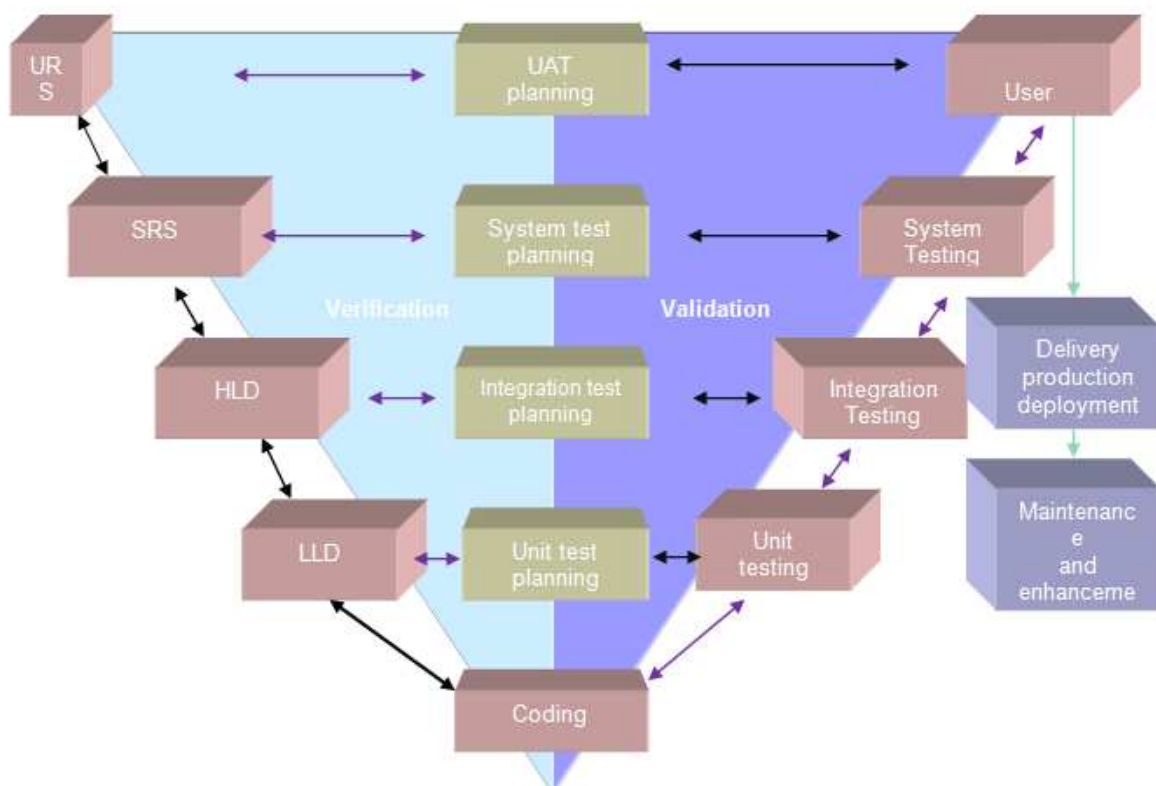
nhận chất lượng phần mềm. Bạn sẽ được tìm hiểu nhiều hơn về thước đo chất lượng phần mềm, nhưng trước tiên hãy xem xét những khái niệm sau:

- Mục đích của *testing* là tìm ra lỗi, tìm thấy chúng sớm nhất có thể, và đảm bảo rằng chúng đã được sửa.
- Trách nhiệm chính của người *QA* là tạo và bắt phần mềm phải tuân theo các chuẩn để cải tiến quy trình phát triển phần mềm và ngăn chặn các lỗi xuất hiện bất cứ lúc nào

Dĩ nhiên, 2 khái niệm này vẫn có sự chồng chéo nhau. Một số *tester* sẽ làm nhiệm vụ *QA*, một số thì thực thi việc kiểm tra. Hai công việc này cùng các nhiệm vụ của nó có quan hệ chặt chẽ với nhau. Tuy nhiên khó mà tránh khỏi sự lộn xộn giữa các thành viên làm nhiệm vụ kiểm thử (*testing*) và các thành viên đảm bảo chất lượng phần mềm (*QA*).

## Mô hình chữ V

Mô hình chữ V sẽ giúp chúng ta hình dung về quy trình test trong toàn bộ kế hoạch thực hiện dự án.



# Quá trình nghiên cứu bản đặc tả phần mềm

Trong phần này chúng ta sẽ tìm hiểu:

- Khởi động
- Thực hiện duyệt ở mức cao của bản đặc tả
- Kỹ thuật kiểm thử bản đặc tả mức thấp

Bài này sẽ giúp bạn bắt tay kiểm thử một phần mềm thật sự đầu tiên nhưng đó không phải là điều mà chúng ta mong chờ. Bạn sẽ không cài đặt và chạy phần mềm và bạn cũng không đập thành thịch vào bàn phím để hi vọng sẽ thấy phần mềm chạy sai. Trong bài này, bạn sẽ học cách làm thế nào để kiểm thử bản đặc tả của sản phẩm để tìm lỗi trước khi chúng được chuyển thành một phần mềm.

Kiểm tra bản đặc tả không phải kiểm tra những thứ mà tất cả các tester cho rằng đó là công việc xa xỉ, không cần thiết. Đôi khi bạn đang ở giữa quá trình phát triển một dự án, sau khi bản đặc tả đã được viết và việc viết mã cũng đã bắt đầu, người ta phát hiện bản đặc tả không ổn. Nếu gặp hoàn cảnh này thì cũng đừng lo lắng, bạn vẫn có thể sử dụng các kỹ thuật đã được mô tả ở đây để kiểm tra bản đặc tả cuối cùng.

Nếu bạn không có đủ may mắn và lâm vào cảnh rắc rối với dự án ngay từ đầu và phải xem xét tới bản đặc tả sơ bộ, bài này sẽ dành cho bạn. Các lỗi được tìm kiếm tại giai đoạn này là tiềm năng giúp bạn tiết kiệm được một số lượng lớn tiền của và thời gian cho dự án của bạn.

Trọng tâm của bài này bao gồm:

- Thế nào kiểm thử *black-box* và *white-box*
- Kiểm thử *static* và *dynamic* khác nhau như thế nào
- Kỹ thuật mức cao nào có thể được sử dụng để duyệt lại một bản đặc tả phần mềm
- Vấn đề đặc biệt nào bạn nên tìm kiếm khi duyệt lại bản đặc tả chi tiết

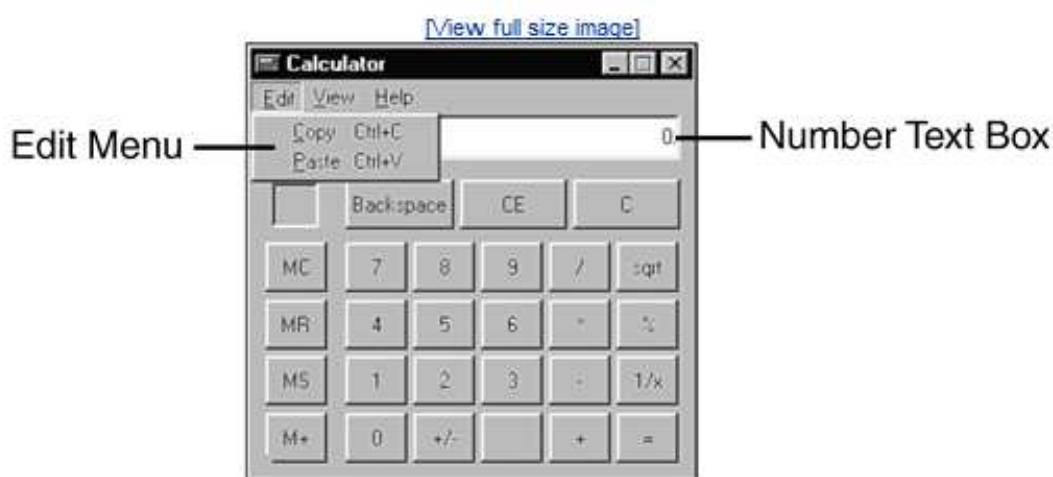
## Khởi đầu

Hãy nghĩ về 4 mô hình phát triển phần mềm được mô tả trong bài 2 “Quy trình phát triển phần mềm”: *big-bang*, *code-and-fix*, *waterfall* và *spiral*. Trong mỗi mô hình, ngoại trừ *big-bang*, đội phát triển phần mềm tạo ra một bản đặc tả (*product specification*) từ tài liệu yêu cầu của khách hàng (*requirement document*) để định nghĩa những cái mà phần mềm sẽ làm.

Diễn hình, bản đặc tả sản phẩm là tài liệu được viết bằng *word* và có các bức tranh để mô tả sản phẩm dự định. Một trích dẫn từ bản đặc tả phần mềm *Windows Calculator* (trong hình 4.1) có thể đọc một số thứ giống như dưới đây:

Menu Edit sẽ có 2 lựa chọn: Copy và paste. Có thể chọn bằng một trong 3 cách: trỏ chuột và click vào các item của menu, sử dụng phím truy cập (alt+E và sau đó là C để Copy và P để Paste), hoặc sử dụng các phím tắt chuẩn của windows như Ctrl+C để Copy và Ctrl+V để Paste.

Chức năng Copy sẽ sao chép toàn bộ dữ liệu hiện tại được hiển thị trên ô textbox và đưa vào Windows Clipboard. Chức năng Paste sẽ dán những dữ liệu được lưu trữ trong Windows Clipboard thành dữ liệu trong ô textbox.



Phần mềm Windows Calculator chuẩn hiển thị menu drop-down Edit

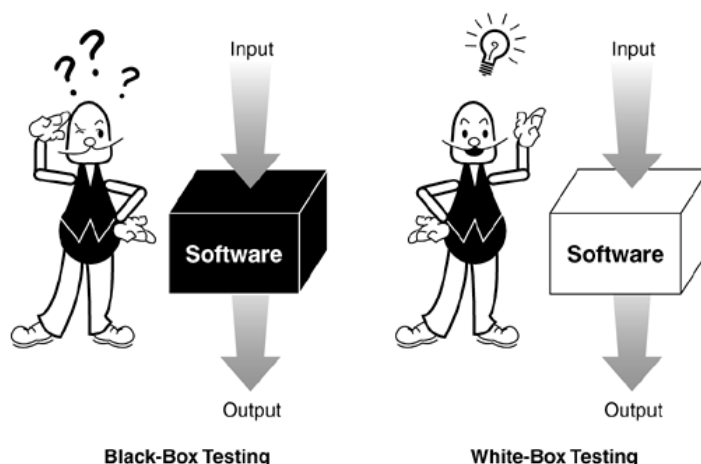
Bạn có thể nhìn thấy rằng, phải mất một đoạn văn bản ngắn để mô tả về việc điều khiển 2 *item menu* trong một chương trình *Calculator* đơn giản. Một bản đặc tả chi tiết và triệt để cho toàn bộ ứng dụng có thể phải dài hàng trăm trang.

Dường như với cách viết như vậy, chúng ta đã tạo ra một văn bản quá tỉ mỉ cho một phần mềm đơn giản. Tại sao lại không để cho lập trình viên viết một phần mềm *Calculator* theo ý của anh ta? Vấn đề là bạn sẽ không thể có ý tưởng nào là cuối cùng. Ý tưởng của các lập trình viên sẽ hình thành nên phần mềm, chức năng nào nên có, và người sử dụng sẽ sử dụng nó như thế nào. Chỉ có cách đảm bảo rằng sản phẩm cuối cùng là cái mà khách hàng yêu cầu với kế hoạch hợp lý về những lỗi kiểm thử để mô tả thấu đáo sản phẩm trong một bản đặc tả.

Những thuận lợi khác có trong một bản đặc tả, và là nội dung cơ bản của bài này, một *tester* sẽ có một văn bản về các *item* để thực hiện kiểm tra. Bạn có thể tìm các lỗi trong chính văn bản đó trước khi dòng mã đầu tiên được viết.

### a) Kiểm thử black-box và white-box

Hai thuật ngữ mà các *tester* sử dụng để mô tả cách mà họ tiếp cận để kiểm thử phần mềm là kiểm thử *black-box* và kiểm thử *white-box*. Hình 4.2 biểu diễn sự khác nhau giữa 2 hướng tiếp cận này. Trong kiểm thử *black-box*, *tester* chỉ biết cái mà phần mềm giả định là thực hiện được và anh ta cũng không hề biết cách thức phần mềm hoạt động như thế nào. Nếu anh ta đưa vào một dữ liệu chính xác, anh ta cần nhận được dữ liệu chính xác. Anh ta không hề biết làm cách nào và tại sao điều đó lại xảy ra.



Với kiểm thử black-box, tester không biết chi tiết về cách thức mà phần mềm làm việc

**Lời khuyên:** Đôi khi, kiểm thử *black-box* được quy về kiểm thử chức năng (*function testing*) hoặc kiểm thử về cách hoạt động (*behavioral testing*). Đừng có cố chạy theo những thuật ngữ được dùng trong thực tế. Đôi của bạn có thể sử dụng những thuật ngữ khác đi. Công việc của bạn là phải hiểu những thuật ngữ này và cách mà chúng được sử dụng trong đội của bạn.

Hãy suy nghĩ về phần mềm *Calculator* được mô tả trong hình 4.1. Nếu bạn đưa dữ liệu vào là 3.14159 và nhấn nút Sqrt, bạn sẽ nhận được 1.772453102341. Với kiểm thử *black-box*, không có vấn đề gì khi phần mềm thực hiện tính toán. Nó vừa thực hiện điều này. Là một tester, bạn có thể xác minh kết quả trên một *Calculator* đã được chứng thực khác và xác nhận rằng *Windows calculator* thực hiện chức năng này đúng.

Trong kiểm thử *white-box* (đôi khi còn được gọi là kiểm thử *clear-box*), *tester* có khả năng truy cập vào mã nguồn và có thể kiểm tra nó với các manh mối để giúp ích quá trình kiểm thử. Dựa trên những gì mà *tester* nhìn thấy, anh ta có thể xác định rằng những con số chính xác nào có thể gây lỗi và từ đó anh ta có thể thiết kế bài kiểm tra của mình dựa trên những thông tin này.

**Chú ý:** Kiểm thử *white-box* có nhiều rủi ro. Rất dễ dàng để phát sinh lỗi khi kiểm thử phần mềm 1 cách khách quan bởi vì bạn có thể thiết kế bài kiểm tra kết nối với điều khiển của mã.

## b) Kiểm thử static và dynamic

Có 2 thuật ngữ khác cũng được sử dụng để mô tả cách phần mềm được kiểm tra là kiểm thử *static* và kiểm thử *dynamic*. Kiểm thử *static* quy về việc kiểm tra một số thứ mà nó không phải đang được chạy, đang kiểm tra và đang duyệt lại nó. Kiểm thử *dynamic* là cái mà thông thường bạn nghĩ về cách để kiểm tra, cách chạy và sử dụng phần mềm.

Sự giống nhau nhiều nhất của các thuật ngữ này là quy trình bạn đi xuyên qua khi kiểm tra một chiếc xe car đã được sử dụng. Đá vào vỏ xe, kiểm tra lớp sơn và nhìn xuống dưới mui xe là các kỹ thuật kiểm tra *static*. Nào hãy bắt đầu, lắng nghe tiếng động cơ, và đánh xe trên đường là các kỹ thuật kiểm thử *dynamic*.

## c) Kiểm thử static black-box: Kiểm tra bản đặc tả

Kiểm tra bản đặc tả là kiểm thử *static black-box*. Bản đặc tả là một tài liệu, không phải là một chương trình đang chạy, vì vậy mà nó được xem xét tĩnh (*static*). Nó cũng có thể là một số thứ được tạo ra sử dụng dữ liệu từ nhiều đề tài nghiên cứu tiện dụng, các nhóm trọng tâm, đầu vào mang tính thị trường... Bạn không cần phải biết làm cách nào và tại sao các thông tin này lại được sử dụng hoặc các chi tiết của quá trình xử lý đã sử dụng nó. Chúng được tóm tắt lại trong một bản đặc tả sản phẩm. Sau đó, bạn có thể nắm bắt được tài liệu này, thực thi việc kiểm thử *static black-box*, và nghiên cứu cẩn thận về các lỗi.

Ở ngay phần đầu, bạn đã quan sát một ví dụ về bản đặc tả sản phẩm cho phần mềm *Windows Calculator*. Ví dụ này đã sử dụng một tài liệu được viết rất chuẩn với một bức tranh mô tả về cách thức hoạt động của phần mềm. Mặc dù đây là cách thức rất phổ biến cho việc viết một bản đặc tả, có nhiều sự thay đổi. Đội phát triển dự án của bạn có thể làm nổi bật biểu đồ dựa trên các từ hoặc nó có thể sử dụng một ngôn ngữ máy *self-document* ví dụ như Ada. Với bất kể lựa chọn nào của họ, bạn vẫn có thể áp dụng tất cả các kỹ thuật biểu diễn trong bài này. Bạn sẽ phải thiết kế chúng dựa trên định dạng của bản đặc tả mà bạn có, nhưng ý tưởng vẫn giống như vậy.

Bạn phải làm gì nếu dự án của bạn không có một bản đặc tả? Có thể đội của bạn đang sử dụng mô hình *big-bang* hoặc một mô hình *code-and-fix* không chặt chẽ lắm. Là một tester, đây là một vị trí khác. Mục đích của bạn là tìm ra các lỗi sớm nhất có thể, trước khi phần mềm được viết code. Nhưng nếu như sản phẩm của bạn không có một bản đặc tả, dường như điều này là không thể được. Mặc dù, bản đặc tả có thể không được viết ra, một ai đó, hoặc một vài người, biết cái thứ mà họ đang cố gắng để xây dựng. Có thể đó là người phát triển phần mềm, là một quản trị dự án, hoặc là một thương gia. Sử dụng

chúng như khi đi dạo, nói chuyện, đặc tả sản phẩm và áp dụng các kỹ thuật giống như đánh giá đánh giá bản đặc tả tinh thần này (*mental specification*), mặc dù nó được viết trên giấy. Thậm chí, bạn có thể từng bước ghi lại các thông tin và tụ họp lại để tính toán lại nó.

Hãy nói với đội dự án của bạn, “Đây là cái mà tôi lập kế hoạch để kiểm tra và đệ trình lại các lỗi”. Bạn sẽ hết sức ngạc nhiên rằng có bao nhiêu chi tiết, họ sẽ ngay lập tức điền đầy nó.

**Lời khuyên:** Bạn có thể kiểm tra một bản đặc tả với các kỹ thuật *static black-box* mà không có vấn đề gì về định dạng của bản đặc tả. Nó có thể là văn bản viết hoặc tài liệu mô tả hoặc cả hai. Thậm chí, bạn có thể kiểm tra một bản đặc tả không được viết, mà chỉ gồm các câu hỏi của những người đang thiết kế và viết phần mềm

# Bai 3 : Các phương pháp kiểm thử

## Phương pháp kiểm thử

### Kiểm thử tĩnh (Static)

#### 1. Khái niệm

- Phương pháp thử phần mềm thông qua việc sử dụng giấy, bút trên bàn để kiểm tra logic, lần từng chi tiết ngay sau khi lập trình xong
- Chủ yếu kiểm tra mã, các tài liệu đặc tả

#### Các phương pháp thử tĩnh

- Thanh tra:
- Khái niệm: Phương pháp kiểm tra ngang hàng sản phẩm phần mềm thực hiện bởi những người nghiên cứu riêng lẻ để tìm ra những lỗi có thể bằng một tiến trình chuẩn cho trước
- Một cuộc thanh tra bao gồm:
  - Đặc tả phần mềm
  - Kế hoạch thanh tra
  - Sản phẩm phần mềm
  - Điều phối viên
  - Thanh tra viên
  - Tác giả phần mềm
- Tiến trình thanh tra:

#### 1. Lên kế hoạch

#### 2. Gặp gỡ trước

#### 3. Chuẩn bị

#### 4. Gặp gỡ thanh tra

#### 5. Gia công lại

#### 6. Bám sát

(Chú ý: các khâu 3,4,5 có thể thực hiện lặp lại)

Duyệt:

- Khái niệm: Là một phương pháp kiểm tra ngang hàng với một người thiết kế hướng nhóm phát triển đến các hoạt động chú ý của quá trình sản xuất phần mềm, tham gia đặt câu hỏi và chú thích cho các lỗi có thể có
- Khác biệt với thanh tra:
- Cấu trúc mở
- Khả năng gợi ý định hướng thay đổi phần mềm
- Tiến trình duyệt:

1. Đánh giá đầu vào

2. Chuẩn bị quản lí

3. Lập kế hoạch

4. Gặp gỡ trước

5. Chuẩn bị riêng

6. Duyệt

7. Gia công/ bám sát

8. Kết thúc, đánh giá

## **Kiểm thử động (Dynamic)**

1. Khái niệm

Khái niệm: Phương pháp thử phần mềm thông qua việc dùng máy chạy chương trình để điều tra trạng thái từng động tác của chương trình

Tiến trình thử động:

1. Thiết kế trường hợp thử theo thử tĩnh
2. Trường hợp thử có kết quả kì vọng
3. Dịch chương trình nguồn và tạo modul tải để thử
4. Xác định miền vào ra của tệp nếu cần thiết
5. Nhập dữ liệu cho trường hợp thử
6. Điều chỉnh môi trường thực hiện modul tải
7. Thực hiện modul tải và ghi nhận kết quả
8. Xác nhận kết quả

(Lặp lại thao tác từ 5-8)



1. Các phương pháp thử động:

- Thử nghiệm khuyết tật
- Khái niệm: Phương pháp thử để tìm ra khuyết tật của phần mềm chủ yếu là lỗi lập trình
- Các loại thử nghiệm khuyết tật
- Thử nghiệm chức năng
- Thử nghiệm cấu trúc
- Thử nghiệm thống kê

# Thiết kế trường hợp kiểm thử

## Khái niệm

Quá trình thiết kế trường hợp thử là quá trình xây dựng các phương pháp kiểm thử có thể phát hiện lỗi ,sai sót khiếm khuyết của phần mềm để xây dựng một phần mềm đạt tiêu chuẩn

## Vai trò

Tạo ra các trường hợp kiểm thử tốt nhất có khả năng phát hiện ra lỗi, sai sót của phần mềm một cách nhiều nhất.

Tạo ra các trường hợp kiểm thử có chi phí rẻ nhất đồng thời tốn ít thời gian và công sức nhất

## Quy trình thiết kế

### a) Kiểm thử hộp trắng (White box test)

Khái niệm : Là phương pháp tập trung kiểm tra về mặt cấu trúc và logic phần mềm theo mục tiêu(Trong trường hợp này yêu cầu người kiểm thử phải biết ngôn ngữ lập trình). Là Kiểm tra trạng thái của chương trình tại nhiều điểm của chương trình.

Tiến trình:

Kiểm thử đường diễn tiến của chương trình

- Khái niệm: Là việc thiết kế các trường hợp kiểm thử trên từng câu lệnh trong chương trình được sẽ được thực hiện ít nhất 1 lần không quan tâm đến ảnh hưởng lên các đường quyết định.
- Các bước tiến hành xây dựng một tập hợp kiểm thử như sau:
- Dùng tài liệu thiết kế hay mã nguồn để vẽ thuật toán của chương trình hay hàm
- Xác định đồ thị V(G)
- Từ đồ thị xác định tập đường độc lập tuyến tính lẫn nhau
- Xây dựng trường hợp kiểm thử dựa trên tập đường xác định ở trên
- Ví dụ: xét thủ tục **average**

Int average( )

```
{ int value[100];
```

```

    int average,totalinput, totalvalid,
    minimum, maximum, sum;

    int i;

    i=0;

    totalinput = totalvalid =0;

    sum = 0;

    while( value[i]<>-999 and totalinput <100)

    {
        totalinput ++;

        if( value[i] >= minimum AND value[i] < maximum)

        {
            totalvalid ++;

            sum = sum + value[i]

        }

        i++;

    }

    if( totalvalid > 0 )

    {
        average = sum/totalvalid

    }

    Else

    {

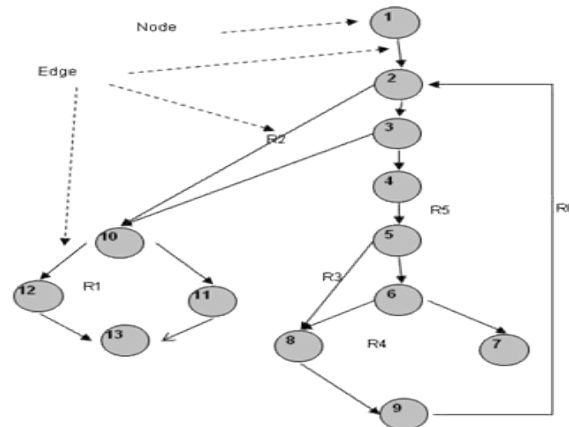
```

average = -999

}

}

- Mô tả:



Mô tả thủ tục Average

Có 6 đường kiểm thử:

Đường 1 : 1-2-10-11-13

Đường 2 : 1-2-10-12-13

Đường 3 : 1-2-3-10-11-13

Đường 4 : 1-2-3-4-5-8-9-2 ...

Đường 5 : 1-2-3-4-5-6-8-9-2 ...

Đường 6 : 1-2-3-4-5-6-7-8-9-2 ...

- Kiểm thử cấu trúc điều khiển
- Kiểm thử các biểu thức điều kiện: là phương pháp kiểm thử dựa trên những điều kiện logic của hàm hay module. Những lỗi phát hiện được:
  - Lỗi do giá trị của biến
  - Lỗi do dấu ngoặc
  - Lỗi do phép toán quan hệ
  - Lỗi trong biểu thức toán học
- Kiểm thử luồng dữ liệu: Phương pháp kiểm thử luồng dữ liệu là chọn lựa một số đường diễn tiến của chương trình dựa vào việc cấp phát, định nghĩa, và sử

dụng những biến trong chương trình. Phương pháp này thích hợp cho kiểm thử một chương trình mà có nhiều lệnh if và vòng lặp nhiều cấp.

- Kiểm thử vòng lặp: Chúng ta cần kiểm thử 4 loại vòng lặp sau:
- Vòng lặp đơn
- Vòng lặp tạo tổ
- Vòng lặp móc nối
- Vòng lặp không có cấu trúc

#### b) Phương pháp kiểm thử hộp đen (Black box test)

- Khái niệm: Là phương pháp tập trung về mặt yêu cầu chức năng của sản phẩm. Có thể tạo ra một bộ các điều kiện đầu vào để kiểm thử tất cả.
- Mục đích của Kiểm thử hộp đen:
- Bổ sung cho phương pháp kiểm thử hộp trắng để phát hiện ra tất cả các lỗi khác nhau mà kiểm thử hộp trắng không phát hiện ra được
- Mục tiêu phát hiện ra các lỗi sau:
- Chức năng thiếu hoặc không đúng đắn
- Sai về giao diện
- Sai thực thi chức năng
- Không đúng hay mất một số hàm/module
- Quy trình của Kiểm thử hộp đen:
- Phân vùng tương đương
- Phân tích giá trị biên
- Kỹ thuật Cause-Effect Graphing

# Chiến lược kiểm thử

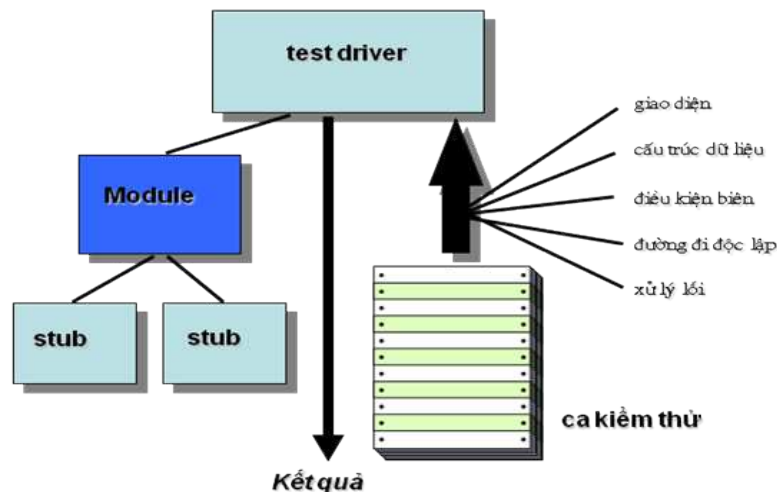
## Các công đoạn kiểm thử

Quá trình kiểm thử có thể chia làm các giai đoạn :

- Kiểm thử mô đun
- Kiểm thử tích hợp
- Kiểm thử hệ con
- Kiểm thử hệ thống
- Kiểm thử big bang
- Kiểm thử nghiệm thu
- Kiểm thử Alpha
- Kiểm thử Beta

## Kiểm thử module

- Kiểm tra một đơn vị thiết kế nhỏ nhất – một **mô đun** – của phần mềm.
- Người tiến hành kiểm thử thông thường là người lập trình mô đun đó hoặc lập trình viên cùng nhóm.
- Các mô đun thứ cấp của mô đun được kiểm thử nếu chưa được phát triển sẽ được thay bằng các chương trình tạm thời gọi là các *stub*.
- Mô đun thượng cấp được thay bằng một trình điều khiển kiểm thử gọi là *test driver*.
- Ví dụ:



Ví dụ về kiểm thử module

- Ví dụ:

```
String calc_day(Date d)
```

```
{
```

```
    return "Sunday";
```

```
}
```

hoặc:

```
String calc_day(Date d)
```

```
{
```

```
    String s;
```

```
    cout << "Enter day_of_week of "<< d;
```

```
    cin >> s;
```

```
    return s;
```

```
}
```

- VD: Dưới đây là một ví dụ đơn giản về test driver của nó:

```
void calc_day_test_drive()
```

```
{
```

```
    Date d;
```

```
    String s;
```

```
    while (1) {
```

```
        cout << "Enter date: ");
```

```
        cin >> d;
```

```
        s = calc_day(d);
```

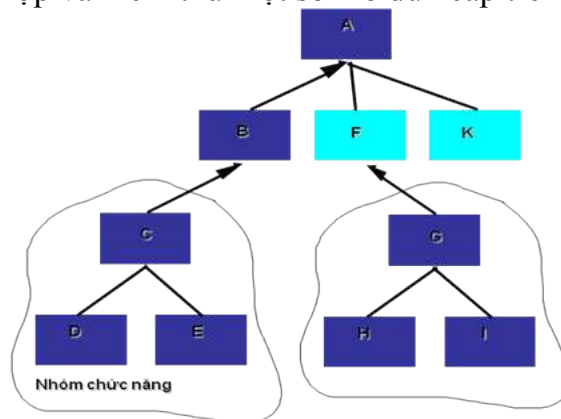
```
        cout << s << endl;
```

}

}

## Kiểm thử tích hợp

- Tích hợp các mô đun và kiểm thử chúng dưới một thể thống nhất.
- Các đơn vị phần mềm (unit) được tích hợp dần thành các mô đun, hệ con, và cuối cùng là thành hệ thống hoàn chỉnh.
- Một số lỗi giao diện (mô đun) điển hình:
  - Sử dụng sai giao diện
  - Hiểu nhầm về giao diện
  - Xung đột
- Các chiến lược kiểm thử tích hợp:
  - **Kiểm thử dưới lên** (*bottom-up testing*)
  - Là quá trình tích hợp và kiểm thử với các mô đun ở mức độ thấp trước.
  - Thông thường người ta không tuân tủy kiểm thử tất cả các mô đun ở tầng dưới cùng mà nhóm các mô đun này thành các nhóm chức năng, tích hợp và kiểm thử chúng theo từng nhóm.
  - Tiến hành tích hợp và kiểm thử một số mô đun cấp trên trước

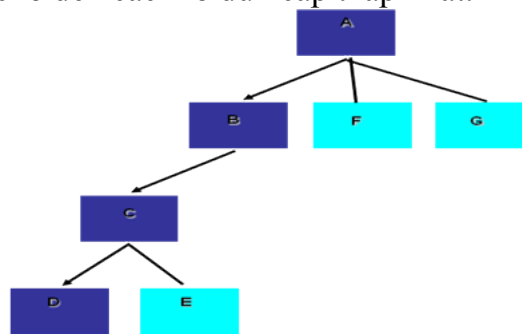


### Kiểm thử từ dưới lên

- Kiểm thử dưới lên có một số ưu điểm:
  - Tránh phải tạo các stub phức tạp hay tạo các kết quả nhân tạo
  - Thuận tiện cho phát triển các mô đun thứ cấp dùng lại được
- Nhược điểm của phương pháp bottom-up:
  - Phát hiện chậm các lỗi thiết kế
  - Chậm có phiên bản thực hiện được của hệ thống
- **Kiểm thử trên xuống** (*top-down testing*)
- Kiểm thử trên xuống tiến hành kiểm thử với các mô đun ở mức cao trước, các mô đun mức thấp được tạm thời phát triển với các chức năng hạn chế.



- Thông thường, để sớm có một phiên bản thực hiện người ta thường tích hợp theo một nhánh cho đến các mô đun cấp thấp nhất.



### Ví dụ Kiểm thử trên xuống

- Ưu điểm của kiểm thử trên xuống
- Phát hiện sớm các lỗi thiết kế
- Có phiên bản hoạt động sớm
- Nhược điểm của kiểm thử trên xuống
- Khó có thể mô phỏng được các chức năng của mô đun cấp thấp phức tạp
- Không kiểm thử đầy đủ các chức năng
- Trên thực tế người ta thường tìm cách phối hợp hai chiến lược này, gọi là *sandwich testing*
- **Kiểm thử hồi qui** (*regression testing*)
- Là tiến hành lại các phép thử đã thành công mỗi khi tích hợp thêm mô đun hoặc khi cập nhật mã nguồn chương trình
- Khi chúng ta tích hợp thêm mô đun vào hệ thống hoặc khi tiến hành nâng cấp chương trình thì sẽ tạo ra một số tổ hợp trạng thái mới dẫn đến:
- Xuất hiện lỗi ở mô đun trước đây chưa gây lỗi
- Khắc phục một lỗi mới có thể sẽ làm ảnh hưởng tới một lỗi chúng ta đã sửa
- Sinh ra lỗi mới mà trước đây chưa có

### Kiểm thử hệ thống

- Kiểm thử khả năng hoạt động của hệ thống
- Kiểm tra các vấn đề về hiệu năng của hệ thống, khả năng phục hồi khi gặp sự cố,...
- Một số các dạng kiểm thử hệ thống chính
- **Kiểm thử phục hồi** (*recovery testing*)
- Là các kiểm thử được tiến hành nhằm làm hệ thống ngừng hoạt động và đánh giá khả năng phục hồi sau đó
- Với các hệ thống có khả năng phục hồi tự động, chúng ta cần đánh giá các công đoạn tái thiết lập thông số, khả năng khôi phục dữ liệu và tái khởi động

- Với các trường hợp đòi hỏi khởi động lại thủ công, chúng ta cần đánh giá thời gian ngừng để sửa chữa (*MTTR – Mean Time To Repair*) và trong một số trường hợp đánh giá cả chi phí cho việc khôi phục.
- **Kiểm thử gây áp lực** (*stress testing*)
- Đây là loại (bước) kiểm thử được tiến hành khi đã có phiên bản làm việc, nhằm tìm hiểu hoạt động của hệ thống trong các trường hợp tải trọng lớn (dữ liệu lớn, số người sử dụng lớn, tài nguyên hạn chế...)
- Mục đích của kiểm thử áp lực là:
- Tìm hiểu giới hạn chịu tải của hệ thống
- Tìm hiểu về đặc trưng của hệ thống khi đạt và vượt giới hạn chịu tải (khi bị sụp đổ)
- Ngoài ra kiểm thử áp lực còn nhằm xác định các trạng thái đặc biệt như tổ hợp một số điều kiện dẫn đến sự sụp đổ của hệ thống; tính an toàn của dữ liệu, của dịch vụ khi hệ thống sụp đổ
- **Kiểm thử hiệu suất** (*performance testing*)
- Kiểm thử hiệu suất (*performance testing*) được thiết kế để đánh giá hiệu suất hoạt động của phần mềm trong một ngữ cảnh cho trước, thông thường là trong một môi trường tích hợp các phần mềm và phần cứng cụ thể
- Được tiến hành ở tất cả các công đoạn kiểm thử
- Kiểm thử hiệu suất liên quan chặt chẽ đến ngữ cảnh sử dụng bao gồm cả các phần mềm khác (hệ điều hành, CSDL,...) và môi trường phần cứng (CPU, bộ nhớ, mạng)
- Kiểm thử hiệu suất thường được tiến hành cùng với kiểm thử áp lực

## **Kiểm thử big – bang**

- Kiểm thử big bang (*big bang testing*) là một chiến lược kiểm thử hệ thống tiến hành một lần duy nhất khi đã phát triển toàn bộ các mô đun và tích hợp thành một phần mềm hoàn chỉnh
- Phương pháp này vẫn thường được tiến hành khi phát triển các phần mềm có kích thước nhỏ

# Bài 4 : Các kỹ thuật kiểm thử

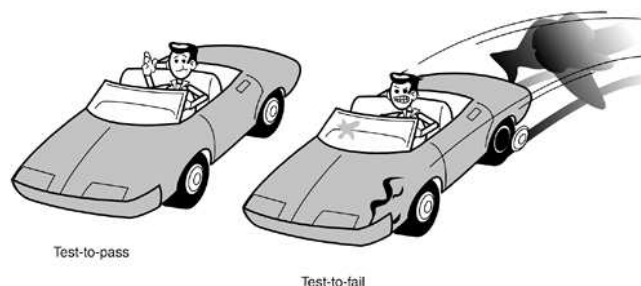
## Test và kiểm tra

### Test – to – pass và Test – to – fail

Có 2 hướng tiếp cận cơ bản khi kiểm thử phần mềm là: *test – to – pass* và *test – to – fail*. Khi bạn *test – to – pass*, bạn thực sự chỉ đảm bảo được rằng phần mềm thực hiện được các chức năng tối thiểu. Bạn đừng cố thúc đẩy những khả năng của nó. Bạn không biết rằng bạn có thể làm hỏng nó. Bạn xem xét nó với một *kid gloves*, áp dụng những *testcase* đơn giản nhất và rõ ràng nhất.

Bạn có thể đang nghĩ rằng, nếu như mục đích của bạn là tìm ra lỗi, tại sao bạn lại lựa chọn *test – to – pass*? Bạn có nghĩ rằng bạn muốn tìm lỗi bằng một vài phương tiện có khả năng thực hiện được không? Câu trả lời là không.

Hãy nghĩ về sự tương đồng với một bản thiết kế xe car mới (hình 4.1). Bạn được giao nhiệm vụ kiểm tra phiên bản đầu tiên mà bạn thì chưa từng lái xe bao giờ. Có lẽ bạn sẽ lái nó với tốc độ lớn nhất mà nó đạt được. Và có thể bạn sẽ bị đâm và chết. Với một chiếc xe car mới, có thể có tất cả các loại lỗi mà nó bộc lộ ra khi lái với tốc độ dưới mức bình thường. Có lẽ là những cái lốp xe có kích cỡ không đúng, hoặc những cái phanh không tương xứng, hoặc là bộ máy quá lớn. Bạn có thể phát hiện ra những vấn đề này và sửa chúng trước khi tung ra thị trường.



Sử dụng *test – to – pass* để khám phá lỗi trước khi *test – to – fail*

**Chú ý:** Khi thiết kế và chạy các *test case*, luôn luôn phải chạy trường hợp *test – to – pass* trước. Đây là điều quan trọng để kiểm tra những chức năng cơ bản của phần mềm trước khi bạn lún sâu vào nó. Bạn có thể sẽ phải ngạc nhiên về cái cách mà bạn tìm thấy rất nhiều lỗi khi sử dụng phần mềm một cách thông thường.

CÁC THÔNG ĐIẾP VỀ LỖI: TEST – TO – PASS HOẶC TEST – TO – FAIL:

Đừng lo lắng về sự tương phản này. Cái quan trọng là cố gắng để bắt được lỗi và xây dựng các test case về các vấn đề chưa từng được xem xét tới.

Lựa chọn các *test case* là một nhiệm vụ vô cùng quan trọng mà một *tester* phải thực hiện và *equivalence partitioning*, đôi khi được gọi là *equivalence classing*, có nghĩa là cái mà chúng ta phải làm. *Equivalence partitioning* là quá trình làm giảm số lượng các *test case* nhưng vẫn đảm bảo đạt được hiệu quả tương đương như khi kiểm thử với số lượng các *test case* cũ.

Ví dụ, Nếu bạn không biết gì hơn về *Equivalence partitioning*, bạn sẽ nghĩ rằng nếu kiểm tra trường hợp 1+1, 1+2, 1+3 và 1+4 thì bạn cũng cần kiểm tra trường hợp 1+5 và 1+6? Bạn có thừa nhận về cách kiểm an toàn như vậy không?

Bạn hãy nhìn và sẵn sàng suy nghĩ giống như một tester!

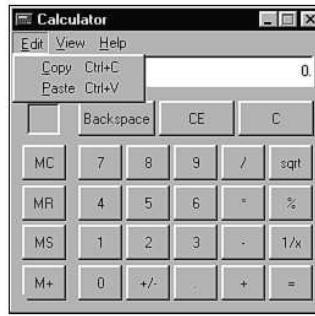
**Chú ý:** Một lớp tương đương (*equivalence class*) hoặc một phân vùng tương đương (*equivalence partition*) là một tập các *test case* kiểm tra những trường hợp tương tự nhau hoặc để khám phá ra những lỗi tương tự nhau.

Sự khác nhau giữa  $1+99999999999999999999999999999$  và  $1+13$  là gì? Trường hợp  $1+13$  là một phép cộng chuẩn, đơn giản, cũng giống như  $1+5$  hay  $1+329$ . Tuy nhiên,  $1+99999999999999999999999999999$  là trường hợp sắc bén hơn. Nếu bạn nhập vào một số lớn nhất có thể, và sau đó cộng thêm 1 vào nó. Điều này có thể dẫn đến một lỗi. Những trường hợp dữ liệu đạt ngưỡng như vậy được đưa vào một phân vùng duy nhất (*unique partition*), khác với các phân vùng thông thường chứa các số theo quy tắc nào đó.

**Chú ý:** Khi tìm kiếm những *equivalence partition*, hãy nghĩ cách để nhóm những dữ liệu đầu vào tương tự nhau, dữ liệu đầu ra tương tự nhau, và những điều khiển tương tự của phần mềm. Những nhóm này sẽ là một *equivalence partition* của bạn

Hãy xem một vài ví dụ dưới đây:

- [illegible]



Có nhiều cách để thực hiện chức năng copy và tất cả đều đưa ra kết quả như nhau.

- Ví dụ thứ 3, hãy xem xét khả năng đưa một file name vào hộp thoại Save As chuẩn (hình 4.4).



Ô Textbox File Name trong hộp thoại Save As minh họa cho một vài khả năng của equivalence partition

- Một filename của Windows của thể chứa các ký tự, ngoại trừ \ : \* ? " < > và |. Các filename có thể có từ 1 đến 25 ký tự. Nếu bạn tạo ra các test case để kiểm thử trường hợp này, bạn sẽ có những equivalence partition với những ký tự hợp lý, ký tự không hợp lý, những cái tên với độ dài hợp lý, những cái tên quá ngắn hoặc những cái tên quá dài.

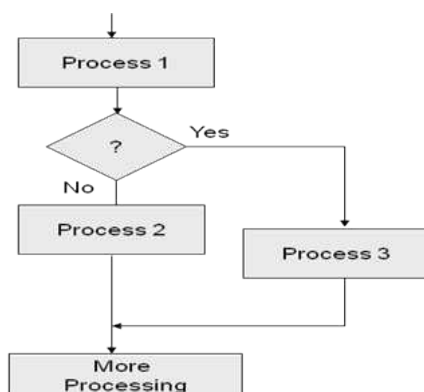
Hãy nhớ lại, mục đích của equivalence partitioning là làm giảm bớt số lượng các test case mà vẫn đảm bảo nó thỏa đáng yêu cầu kiểm thử. Bạn đang phải có trách nhiệm về sự rủi ro bởi vì bạn sẽ không kiểm thử tất cả mọi thứ, vì vậy mà bạn cần phải cẩn thận để lựa chọn cách phân chia lớp.

**Chú ý:** Nếu việc phân chia lớp vượt quá những cố gắng của bạn, những trường hợp bị loại trừ khỏi quá trình kiểm thử cũng có thể chứa lỗi. Nếu bạn là một người mới trong lĩnh vực kiểm thử, bạn hãy nhờ một người có nhiều kinh nghiệm hơn để họ duyệt lại những lớp tương đương mà bạn đề xuất.

Điểm quan trọng cuối cùng về *equivalence partitioning* là sự phân chia này mang tính chủ quan. Nó vừa mang tính khoa học vừa mang tính nghệ thuật. Hai tester kiểm thử một chương trình phức tạp thì chúng ta sẽ có 2 tập các *partition* khác nhau. Sau khi các phân vùng được phân chia đã được duyệt và mọi người đồng ý chấp nhận toàn bộ phần mềm đã được kiểm thử.

## Kiểm tra luồng điều khiển (Control flow testing )

Luồng điều khiển là một kỹ thuật testing căn bản, sử dụng sơ đồ luồng xử lý (control flow graph). Đó là sơ đồ mô hình hoá hành vi của hệ thống, chứ không phải là sơ đồ mô tả các câu lệnh trong code. Áp dụng được cho hầu hết các phần mềm, và có hiệu quả. Và áp dụng được trong mọi giai đoạn (testing stages).



Sơ đồ luồng xử lý

## Kiểm tra luồng dữ liệu (Data flow testing)

- Áp dụng cho các hệ thống “data-intensive”
- Ví dụ các hệ thống sản sinh báo cáo, thống kê.
- Ví dụ các hệ thống có tính toán thay đổi số liệu.
- Phương pháp xây dựng test case:
- Lập sơ đồ luồng dữ liệu (Data flow diagram)
- Lăn theo từng đường dẫn trong sơ đồ
- Bắt đầu từ node output
- Lăn ngược lại tới khi gặp node input
- Ta đã được một test case

## Kiểm tra sự giao dịch (Transaction testing)

- Áp dụng cho các hệ thống xử lý giao dịch (như đặt vé máy bay, đặt phòng khách sạn, ...)

- Sử dụng mô hình xử lý của hệ thống, chú trọng đến điểm bắt đầu, điểm kết thúc của từng xử lý, chú trọng tới hàng đợi (queue)



# Kiểm tra miền

## Kiểm tra miền (Domain testing)

- Áp dụng cho các xử lý mà có xác định phạm vi (range) giá trị dữ liệu
- Chú trọng test các giá trị biên on và off

## Kiểm tra lặp (Loop testing)

- Áp dụng trong whitebox testing: quan tâm đến vòng lặp trong code
- Áp dụng trong blackbox testing: quan tâm đến vòng lặp trong hành vi của hệ thống
- Ví dụ khi hệ thống phải tìm ra tất cả các bản ghi thoả mãn một tiêu chí tìm kiếm nào đó
- Giả sử khả năng hệ thống có thể hỗ trợ tối đa Max vòng lặp, chỉ cần chọn thực hiện những test case sau là đủ:
  - 0 lần, 1 lần, 2 lần qua vòng lặp
  - X lần (X: số ngẫu nhiên, giữ khoảng 0 và Max)
  - Max -1, Max, Max+1 lần

## Kiểm tra cú pháp (Syntax testing)

- Là kỹ thuật dùng để:
  - Test các câu lệnh (commands)
  - Test việc xử lý các trường phải tuân theo một định dạng xác định
- Ví dụ về syntax:
  - Ngày tháng
  - Địa chỉ email
  - Công thức toán học do NSD định nghĩa
  - ...
- Phân tích, nắm rõ qui tắc syntax
- Thiết kế các positive test cases, sử dụng kỹ thuật Equivalence class partitioning
- Thiết kế các negative test
  - Mỗi lần làm sai một phần tử trong syntax
- Thực hiện test

## Kiểm tra trạng thái (State machine testing)

- Áp dụng khi:
  - Test các “menu driven application”
  - Test các hệ thống thiết kế bằng phương pháp hướng đối tượng

- Test bất cứ hệ thống nào có sơ đồ chuyển đổi trạng thái (state)
- Ví dụ về trạng thái:
  - Account sử dụng Portal chưa có hiệu lực (inactive account)
  - Account sử dụng Portal có hiệu lực (active account)
- Đặc trưng của trạng thái:

Ở mỗi trạng thái, có một số hành động được phép thực hiện và một số khác thì không

### Kiểm tra khả năng chịu tải và vận hành của hệ thống (Load, stress và performance testing)

- Load testing: bắt hệ thống phải chịu tải lớn (thực hiện nhiều xử lý), ví dụ:
  - Có nhiều client cùng lúc truy cập
  - Có nhiều giao dịch cùng một lúc
  - Xử lý file rất lớn
  - Xử lý cùng lúc nhiều file
- Stress testing: bắt hệ thống vận hành trong điều kiện bất thường, ví dụ:
  - Thiếu bộ nhớ
  - Kết nối mạng bị ngắt khi đang vận hành
  - Database server down
- Performance testing: xác định điều kiện để hệ thống hoạt động tốt nhất

Bảng 2								
			Performance Test (PT)		Load Test (LT)		Stress Test (ST)	
Khái niệm			- Được thực hiện nhằm xác định tốc độ, khả năng phân tải và mức độ tin tưởng của PM trong môi trường nhiều người dùng, có nhiều hoạt động khác nhau.- Dùng công cụ KTTĐ để kiểm tra hiệu năng PM ở điều kiện có sự điều		- Là một phần trong qui trình thực hiện PT. Load Test đôi khi còn gọi là <i>Volume Test</i> .- Dùng công cụ KTTĐ để kiểm tra PM ở điều kiện liên tục tăng mức độ chịu tải. Tuy nhiên mức độ chịu tải cao nhất vẫn ở mức chức năng PM hoạt động đúng chức năng.		- Đây là cách thực hiện nhằm làm cho PM không còn chạy được nữa.- Phương pháp này rất hay áp dụng để kiểm tra PM và phần cứng.	

			chỉnh về mức độ tải.					
	Mục tiêu		<p>- Tìm ra điểm “thắt cổ chai” của PM và từ đó có những cải tiến để tăng khả năng hoạt động của PM.- Đề ra các thông số, tiêu chuẩn về hiệu năng thực thi của PM. Một số thông số đó là: số phiên làm việc, thời gian xử lý của phiên làm việc, và các tài nguyên khác bị chiếm giữ.</p>		<p>Khi PM không còn khả năng cải tiến. Ở mức chịu tải cao nhất:- Giám sát việc PM quản lý tài nguyên khi chạy trong thời gian dài.- Giám sát thông số đề ra trong PT như thời gian xử lý,... khi chạy trong thời gian dài.</p>		<p>- Kiểm tra khả năng phục hồi của PM sau khi có sự cố.- Kiểm tra khả năng chịu tải cho một máy khác khi máy đó gặp sự cố do không có khả năng chịu tải được nữa.</p>	
	Ví dụ		<p><i>Ví dụ 1:</i> Có ứng dụng web, yêu cầu cần tìm thông số về hiệu năng thực thi của ứng dụng. Dùng LoadRunner tạo tình huống khởi đầu có 10 người dùng, cứ 2 phút tăng thêm 10 người, tăng tối đa là 2000 người. Quan sát: Biểu đồ thời gian đáp ứng với kết quả xử lý đúng và kết quả sai, có bao nhiêu yêu cầu không được xử lý, tài nguyên</p>		<p><i>Ví dụ 2:</i> Ứng dụng web chỉ hoạt động tốt với tối đa là 1000 người dùng. Yêu cầu kiểm tra khả năng của web khi hoạt động ở ngưỡng đáp ứng với thời gian dài 2 ngày. Dùng LoadRunner tạo tình huống khởi đầu có 800 người dùng, cứ 1 phút tăng 2 người, tăng tối đa 1000 người, và giữ ở mức đó tiếp tục chạy trong vòng 2 ngày. Quan sát: Tài nguyên sử dụng như RAM,</p>		<p><i>Ví dụ 3:</i> Ứng dụng web chỉ có thể quản lý, đáp ứng tối đa 1000 yêu cầu. Yêu cầu cần kiểm tra ứng xử của web khi gặp sự cố quá ngưỡng số người sử dụng. Dùng LoadRunner tạo tình huống có 1100 người truy cập, khi đạt đến ngưỡng đó LoadRunner ngừng tải. Quan sát: Kết quả xử lý 1000 yêu cầu đầu, 100 yêu cầu sau đó bị từ chối ra sao, webserver có khả</p>	

			<p>sử dụng như RAM, CPU,... Thông qua đó giúp xác định <i>ứng dụng hoạt động tốt nhất trong điều kiện nào.</i></p>			<p>CPU, thời gian đáp ứng với kết quả đúng và sai khi ứng dụng chịu tải ở mức cao nhất và <i>hoạt động trong thời gian dài.</i></p>			<p>năng bị khởi động lại hay không,... Từ đó giúp đưa ra kết luận ứng dụng sẽ <i>ứng xử như thế nào khi đạt ngưỡng chịu tải tối đa.</i></p>	
--	--	--	--	--	--	---	--	--	---	--

*So sánh giữa Load, stress và performance testing*

# Bài 5 : Các vấn đề cần kiểm thử

## Kiểm thử cấu hình

### Kiểm thử cấu hình

Trong thực tế có nhiều cửa hàng có thể rộng tới 50.000 foot<sup>2</sup> đang chào bán các máy tính cá nhân, các máy in, màn hình, card vào mạng, modem, máy quét, máy quay phim kỹ thuật số, thiết bị ngoại vi, net-ca và hàng nghìn các đại lý máy tính của hàng trăm công ty – tất cả đều có thể kết nối với máy tính của bạn.

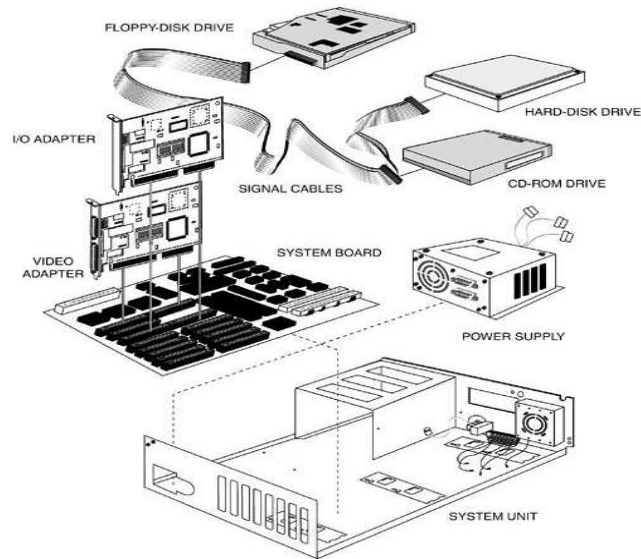
Nếu bạn chỉ vừa mới bắt đầu công việc kiểm thử phần mềm, một trong những nhiệm vụ đầu tiên của bạn có thể là kiểm thử cấu hình. Bạn phải đảm bảo rằng phần mềm của bạn làm việc với càng nhiều tổ hợp phần cứng khác nhau càng tốt. Nếu bạn không kiểm thử phần mềm với một platform phần cứng phổ biến hay nếu bạn đang kiểm thử một số hệ thuộc quyền sở hữu chuyên biệt bạn sẽ vẫn cần xem xét những gì bạn học trong bài này vào tình huống của mình.

### Tổng quan về kiểm thử cấu hình

Khi bạn vào một trong các cửa hàng máy tính, hãy để ý tới một vài hộp phần mềm và đọc kỹ các yêu cầu về tính hệ thống. Bạn sẽ thấy những thứ này chẳng hạn như một PC với bộ xử lý là Pentium 4, màn hình màu 1024×768, card hình 32 bit, game port, ..... Kiểm thử cấu hình là kiểm tra hoạt động của phần mềm bạn đang kiểm thử với tất cả các cấu hình. Hãy xem xét khả năng cấu hình khác nhau đối với một PC hoạt động dựa trên chuẩn Windows được sử dụng trong các gia đình và các cơ quan.

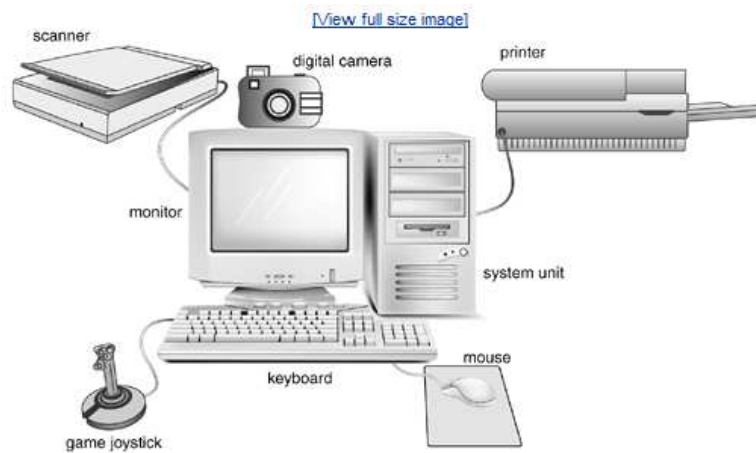
- The PC (máy tính cá nhân), có nhiều nhà sản xuất máy tính nổi tiếng như là Dell, Gateway, Packard. Một nhà sản xuất tạo ra các PC bằng cách sử dụng các linh kiện họ tự thiết kế hay thu được từ các nhà sản xuất khác. Nhiều người yêu thích thậm chí họ còn tạo ra những PC của riêng mình bằng cách sử dụng những linh kiện cũ có sẵn tại các cửa hàng máy tính.
- Components (các linh kiện). hầu hết các PC đều có tính khuôn mẫu và được xây dựng từ rất nhiều bảng hệ thống, các linh kiện và các thiết bị nội vi như đĩa cứng, đĩa CD-ROM, đèn DVD, bộ biến hoàn hình ảnh, âm thanh, máy fax và card vào mạng (xem hình 5.1). Có các card dò sóng tivi, và các chuyên biệt cho việc thu hình và kỹ thuật tự động hóa tại nhà. Thậm chí còn có các card đầu ra đầu vào có thể cung cấp cho PC khả năng kiểm soát của cả một nhà máy nhỏ. Những thiết bị nội vi này được sản xuất bởi hàng trăm các nhà sản xuất khác nhau.

- **Peripherals (các thiết bị ngoại biên):** (các thiết bị ngoại biên như trong hình 5.2), là các máy in, máy quét, chuột, bàn phím, màn hình, camera, cần điều khiển, và các thiết bị khác cắm bên ngoài máy của bạn và điều khiển ngoại vi đối với máy tính.
- **Interface (giao diện):** các linh kiện và các thiết bị ngoại biên được cắm vào máy tính của bạn thông qua nhiều loại thiết bị nối Interface (xem hình 5.3). Những Interface này có thể là ngoại vi hoặc nội vi đối với PC, Những tên điển hình của các Interface là ISM, PCI, USB, PS/2, RS/232, RJ-11, RJ-45 và Firewire. Có nhiều khả năng những nhà sản xuất phần cứng sẽ tạo ra cùng một thiết bị ngoại biên nhưng với các Interface khác nhau. Có thể sẽ mua được chính xác cùng một loại chuột ở ba cấu hình khác nhau



## Vô số các linh kiện ngoại vi tạo nên cấu hình một PC

- Các tùy chọn và bộ nhớ (Option and memory). Nhiều linh kiện với các thiết bị ngoại biên có thể được giao bán với các tùy chọn phần cứng và kích cỡ bộ nhớ khác nhau. Các máy in có thể được nâng cấp để hỗ trợ các font phụ hay chấp nhận bộ nhớ lớn hơn để đẩy nhanh tốc độ in. Các thẻ đồ họa (graphics cards) có bộ nhớ lớn hơn có thể hỗ trợ màu sắc bổ sung và độ phân giải cao hơn. Bảng hệ thống có thể có các phiên bản BIOS khác nhau (BIOS-hệ đầu ra đầu vào cơ bản) và dĩ nhiên, có thể có cả bộ nhớ dung lượng lớn.



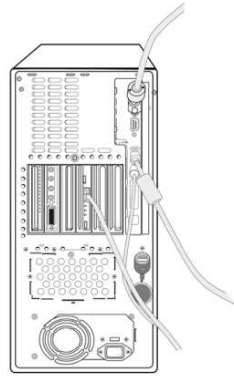
Một PC có thể kết nối với nhiều thiết bị ngoại biên

- Các điều khiển thiết bị (Device Drivers): Tất cả các linh kiện và thiết bị ngoại biên liên hệ với hệ điều hành và các ứng dụng phần mềm được gọi là các đĩa Device. Những đĩa này thường được cung cấp bởi các nhà phân phối đĩa cứng và được lắp đặt khi bạn tạo ra phần cứng. Dù trong kĩ thuật chúng là các phần mềm. Cho mục đích kiểm thử, chúng được coi là bộ phận của cấu hình phần cứng.

Nếu bạn là một người kiểm thử chuẩn bị bắt đầu công việc kiểm thử đối với một loại phần mềm, bạn xem xét xem vùng cấu hình nào trong các vùng cấu hình này liên hệ chặt chẽ nhất với chương trình. Một chương trình trò chơi vì tính mang tính đồ họa cao sẽ đòi hỏi nhiều sự quan tâm về các khía cạnh âm thanh và hình ảnh. Một chương trình thiếp mờ đặc biệt sẽ dễ bị tổn thương đối với các vấn đề in ấn. Một chương trình fax hay giao tiếp sẽ cần được kiểm thử với vô số modem và cấu hình mạng.

Bạn có thể sẽ băn khoăn tại sao điều này là cần thiết. Sau cùng, có các chuẩn cần biết để tạo phần cứng cho dù nó là cho máy tính cũ hay là một máy tính chuyên dụng trong bệnh viện. Bạn có thể sẽ mong muốn rằng nếu mỗi người thiết kế phần cứng của chính mình thành một tập hợp chuẩn phần mềm có thể sẽ chỉ làm việc với nó mà không gặp bất kì một vấn đề nào. Trong một thế giới lí tưởng, điều đó có thể sẽ xảy ra nhưng thật không may các chuẩn mực không phải lúc nào cũng tuân theo. Đôi khi các chuẩn khá lỏng lẻo (gọi là những guideline). Các nhà sản xuất card và thiết bị ngoại vi luôn cạnh tranh mạnh mẽ với nhau và thường phá luật nhằm gây áp lực đối với một đặc tính phụ hoặc nhằm thành công trong công việc gạt hái chút ít thành tích cuối cùng.

Thông thường các đĩa device được gói trong hộp khi phần cứng được đem ra ngoài, Kết quả là phần mềm không làm việc chính xác với những cấu hình phần cứng nhất định.



Mặt sau của một PC chỉ ra vô số các đầu nối giao diện cho việc gắn các thiết bị ngoại biên,

### 1. Phân loại (tách biệt, cô lập) lỗi cấu hình

Những lỗi cấu hình có thể rất khó. Bạn có nhớ lại lỗi của phim vua sư tử ở bài 1. Đó là một vấn đề về cấu hình. Nếu bạn từng chơi một trò chơi hoặc sử dụng một chương trình đồ họa và màu sắc của nó đột nhiên rối loạn hay các mảnh cửa sổ lui về sau khi bạn rê chúng. Có lẽ bạn đã phát hiện ra lỗi cấu hình của thiết bị tiếp hợp hiển thị. Nếu bạn từng dành nhiều giờ (hay nhiều ngày) để cố gắng làm cho một chương trình cũ làm việc với chiếc máy in mới của mình, đó cũng có thể là một lỗi cấu hình.

**Chú ý:** Cách chắc chắn nhất để kết luận một lỗi chỉ là vấn đề cấu hình nào là thực hiện lại hoạt động đã gây ra vấn đề từng bước một trên một máy khác có cài đặt phần cứng hoàn toàn khác. Nếu lỗi không xuất hiện đó dường như chỉ là vấn đề cấu hình chỉ xuất hiện do phần cứng được sử dụng trong việc kiểm thử.

Giả sử rằng bạn kiểm thử phần mềm của mình trên một cấu hình duy nhất và phát hiện ra một vấn đề. Ai nên sửa lỗi này- đội của bạn hay nhà sản xuất phần cứng? Nó có thể sẽ trở thành câu hỏi đáng giá hàng triệu đô.

Trước hết bạn cần kiểm tra xem vấn đề nằm ở đâu. Đây thường là việc kiểm thử *Dynamic white-box* và nỗ lực diễn lại lỗi người lập trình. Một vấn đề cấu hình có thể xuất hiện với nhiều lý do, tất cả đều đòi hỏi ai đó phải kiểm tra mã kỹ lưỡng trong khi chạy phần mềm dưới các cấu hình khác nhau để tìm lỗi.

- Phần mềm của bạn có thể có lỗi xuất hiện dưới Broadclass của cấu hình. Ví dụ, chương trình thiệp mời của bạn làm việc tốt với các máy in laser nhưng không làm việc được với các máy in inkjet.
- Phần mềm của bạn có thể có một lỗi cụ thể với một cấu hình xác định. Nó không làm việc với máy in Okee Dokee Model BR549 Inkjet Deluxe.
- Ổ cứng hay các đĩa khởi động của nó có một lỗi mà chỉ xuất hiện có phần mềm của bạn. Có thể phần mềm của bạn là một nơi duy nhất dùng chỉ một cài đặt



card hiển thị. Khi phần mềm của bạn được chạy bởi một card hình cụ thể, máy tính sẽ crash (tắt hổng)

- Ổ cứng hay các đĩa khởi động có thể xuất hiện lỗi mà có thể được thấy bởi nhiều phần mềm khác nhau – dù có thể đó chỉ là trường hợp cá biệt xảy ra với máy của bạn, Ví dụ, nếu một đĩa in chuyên biệt luôn mặc định chế độ nháp và phần mềm in ảnh của bạn phải cài đặt chế độ chất lượng cao mỗi lần in.

Trong hai trường hợp đầu tiên, dường như khá dễ dàng để thấy rằng đội dự án của bạn có trách nhiệm phải sửa lỗi. Đó là vấn đề của bạn. Bạn nên sửa nó.

Trong hai trường hợp sau, mọi thứ có vẻ mơ hồ hơn. Hãy cho rằng lỗi là ở máy in và máy in đó thì lại là loại phổ biến nhất trên thế giới với hàng triệu người sử dụng.

Phần mềm của bạn thực sự phải làm việc với máy in đó. Có thể máy in cần tới vài tháng để giải quyết vấn đề (nếu nó thực sự làm). Vì vậy, đội của bạn cần phải thực hiện các thay đổi đối với phần mềm của mình, thậm chí khi phần mềm thực hiện tốt mà mọi thứ khi làm việc với lỗi.

Cuối cùng trách nhiệm của đội bạn là phải chỉ ra vấn đề dù nó nằm ở đâu. Khách hàng của bạn không quan tâm tại sao có lỗi và lỗi đó xảy ra thế nào, họ chỉ cần phần mềm mới để làm việc trên cấu hình thuộc về hệ thống của họ.

## OF PURPLE FUZZ VÀ CÁC ÂM THANH

- Năm 1977 Microsoft cho ra sản phẩm Acti Mates và hỗ trợ phần mềm học CD-ROM cho trẻ em. Những búp bê hoạt cảnh này tương tác với phần mềm qua một radio hai chiều trong búp bê và một radio khác gắn với một PC
- Radio của PC kết nối với một giao diện ít được sử dụng trên hầu hết các tiếng được gọi là một bộ nối. Giao diện này được sử dụng cho các phím chơi nhạc và các nhạc cụ khác. Microsoft cho rằng bộ nối có thể là một lựa chọn tốt vì hầu hết mọi người chưa có các thiết bị nhạc. Điều này giống như một vật gì đó được cắm vào nó và sẵn sàng cho việc sử dụng với Acti mates radio.
- Suốt quá trình kiểm thử cấu hình, lượng lỗi điển hình được chỉ ra. Một số các vấn đề về card tiếng một số nằm ở phần mềm Acti Mates. Tuy nhiên có một lỗi mà có thể không bao giờ dừng lại kể cả khi đã bị phát hiện. Nhưng ít khi xuất hiện trong trường hợp PC chạy phần mềm sẽ chỉ khóa và đòi hỏi phải khởi động lại. Tất nhiên vấn đề chỉ xuất hiện với các card tiếng nổi tiếng nhất trên thị trường.
- Chỉ với vài tuần còn lại trong lịch trình, nỗ lực phối hợp sẽ được đưa ra để giải quyết vấn đề. Sau rất nhiều thời gian kiểm thử và xóa lỗi, lỗi đó được cô lập đối với phần cứng của các tiếng. Dường như bộ nối MIDI luôn có lỗi này nhưng do ít khi được sử dụng không ai có thể thấy được nó. Phần mềm Acti Mates đã chỉ ra nó là lần đầu tiên.

- Có rất nhiều tranh cãi và chỉ trích. Cuối cùng, nhà sản xuất card tiếng phải thừa nhận rằng có một vấn đề và hứa sẽ giải quyết nó trong phiên bản cập nhật của Driver thiết bị của nó. Microsoft đã đưa vào một số đĩa sửa lỗi trên CD-ROM Acti Mates và thực hiện những sự thay đổi với phần mềm trong nỗ lực hạn chế lỗi xuất hiện thường xuyên bất chấp tất cả những nỗ lực đó, những vấn đề có tính tương thích của card tiếng vẫn là lí do chính để người ta kêu gọi sự trợ giúp của sản phẩm.

## b) Đánh giá công việc

Công việc về kiểm thử cấu hình có thể là một nhiệm vụ lớn. Giả sử rằng bạn đang kiểm thử một phần mềm game chạy dưới *Microsoft Windows*. Game này mang tính đồ họa cao, có nhiều hiệu ứng âm thanh cho phép nhiều người chơi cạnh tranh với nhau qua đường điện thoại và có thể in được các chi tiết game cho việc vạch kế hoạch chiến lược.

Ít nhất bạn cần đánh giá việc kiểm thử cấu hình với các card hình, card tiếng, modem và máy in khác nhau. Window Add New Hardware Wizard (xem hình 5.4) cho phép bạn chọn phần cứng từ một trong số các danh mục này và 25 danh mục khác.

Dưới mỗi danh mục phần cứng là tên các nhà sản xuất và các mẫu khác nhau (xem hình 5.5), Hãy giữ trong đầu suy nghĩ, chỉ có các mẫu có hỗ trợ được mới gắn vào Windows. Nhiều mẫu khác cung cấp chính cái đĩa cài đặt với phần cứng của chúng.

Nếu bạn quyết định trình bày một cuộc kiểm thử cấu hình toàn diện và đầy đủ, hãy kiểm tra mỗi thứ khả thi rồi kết hợp mẫu, bạn sẽ có một nhiệm vụ lớn đang chờ đợi ở phía trước.



Hộp thoại microsoft Add New Hardware cho phép bạn bổ sung một phần cứng vào cấu hình hiện tại trong PC của mình.



Mỗi loại phần cứng có vô số nhà sản xuất và mẫu

Như vậy có tới xấp xỉ 336 card hiển thị, 210 card tiếng, 1500 modem và 1200 máy in. Lượng phối hợp là  $336 \times 210 \times 1500 \times 1200$  cho tổng số hàng triệu cách. (Quá nhiều để xem xét)

Nếu bạn giới hạn việc kiểm thử của mình để loại ra các phương pháp kết hợp chỉ kiểm thử từng card trong khoảng 30 phút cho mỗi cấu hình, bạn sẽ mất tới cả năm. Hãy nhớ rằng chỉ có một card phù hợp với các cấu hình. Thường thì không phổ biến lắm khi lỗi chạy được qua 2 hoặc 3 cuộc kiểm tra cấu hình khi sản phẩm được tung ra.

Câu trả lời đối với sự rắc rối này là “phân chia lớp tương đương”. Bạn cần chỉ ra một cách để hạn chế các tập hợp cấu hình khả thi lớn đến chỉ những cấu hình gây vấn đề nhất. Bạn sẽ giảm thiểu sự rủi ro khi không kiểm thử tất cả nhưng đó chính là kiểm thử phần mềm.

## Cấu Hình Hardware

### Tiếp cận nhiệm vụ

Quá trình quyết định đi tới việc quyết định xem thiết bị nào dùng để kiểm thử và việc chạy chúng nên được kiểm thử như thế nào là một dự án ‘phân chia lớp tương đương’ khá rành mạch.

Điều gì là quan trọng và điều làm cho nỗ lực có thành công hay không là những thông tin bạn cần sử dụng để lập nên các quyết định. Nếu bạn không có kinh nghiệm về phần cứng và phần mềm của bạn chạy trên, bạn nên học càng nhiều càng tốt và hãy nhớ sự giúp đỡ của những kiểm thử viên hay lập trình viên có kinh nghiệm. hãy hỏi thật nhiều và đảm bảo rằng kế hoạch của bạn được cải thiện.

Những phần sau sẽ chỉ ra quá trình chung mà bạn nên tiến hành khi vạch ra kế hoạch kiểm thử cấu hình.

**a)** Hãy quyết định những kiểu phần cứng bạn sẽ cần.

Ứng dụng của bạn có cần in không? Nếu có bạn cần kiểm thử máy in. Nếu nó có tiếng bạn cần kiểm tra thử card tiếng. Nếu nó là một chương trình đồ họa hay ảnh, bạn cần phải có máy quét và camera kỹ thuật số. Hãy xem xét kỹ lưỡng bộ đặc tính –phần mềm để đảm bảo rằng bạn không bỏ sót thứ gì. Hãy đặt đĩa phần mềm của bạn lên bàn và tự hỏi những phần cứng nào bạn đang sử dụng với nhau để làm cho nó hoạt động.

**ĐĂNG KÍ TRỰC TUYẾN:** Ví dụ về đặc tính bạn dễ bỏ qua khi chọn phần mềm để sử dụng là đăng kí trực tuyến. Ngày nay nhiều chương trình cho phép người sử dụng đăng kí phần mềm của mình trong quá trình cài đặt thông qua sự kết hợp broadband (dải rộng, băng rộng ) và modem. Người sử dụng gõ nhập tên địa chỉ và các dữ liệu cá nhân khác của mình, nhấp vào một nút và modem kết nối với một máy tính tại công ty phần mềm nơi nó tải thông tin và hoàn thành việc đăng kí. Phần mềm có thể không thực hiện bất cứ điều gì với các liên lạc trực tuyến. Nhưng nếu nó đăng kí trực tuyến, bạn cần coi các modem và việc liên lạc qua card mạng là một phần trong công việc kiểm thử cấu hình của mình.

**b)** Quyết định phần cứng, mẫu và device driver nào là có sẵn?

Nếu bạn đang làm lệch đi một chương trình đồ họa cutting-edge, có lẽ bạn không cần kiểm thử xem nó có in tốt với máy in dot-matrix đen trắng 1987. Hãy làm việc với những người bán hàng nhằm đưa ra danh sách thiết bị để kiểm thử. Nếu họ không thể hoặc không giúp bạn, hãy kiểm tra thông tin về các thiết bị gần đây trên các tạp chí PC hoặc Mac Word để biết phần mềm nào là có sẵn và thiết bị nào đang phổ biến. Cả hai tạp chí

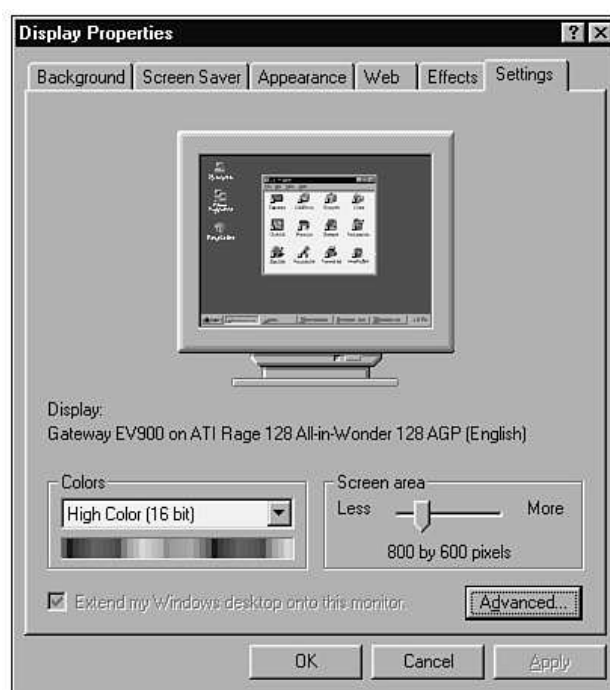
này cũng như các tạp chí khác đều có những tổng kết hàng năm về các máy in, các tiếng, thiết bị tiếp hợp hiển thị và những thiết bị ngoại biên khác.

Hãy tiến hành nghiên cứu xem một vài thiết bị có tách dòng với các thiết bị khác hay không và vì thế có sự lắp ghép tương đương hay không. Ví dụ, một hãng sản xuất máy in có thể cấp phép để một công ty khác đặt phần vỏ máy và dán nhãn hiệu trên máy đó. Từ đây có thể thấy, đó là chiếc máy in giống như vậy.

Hãy quyết định xem *driver* thiết bị (*device driver*) nào bạn chuẩn bị kiểm thử. Những tùy chọn của bạn thường là các driver được với hệ điều hành các driver được gắn với các thiết bị hay các driver mới nhất có sẵn trên website của công ty sản xuất phần cứng hay hệ điều hành. Thông thường 3 loại driver là khác nhau. Bạn hãy hỏi chính mình xem khách hàng có gì hay họ có thể nhận được những gì.

c) Quyết định xem đặc tính phần cứng, chế độ và tùy chọn nào là có thể

Máy in màu có thể in đen trắng hoặc in màu, chúng có thể in ở các chế độ chất lượng khác nhau và có thể có những cài đặt cho việc in ảnh hoặc text. Các card hiển thị, như được chỉ ra trong hình 5.6 có thể có các cài đặt màu và độ phân giải màu hình khác nhau.



các thuộc tính hiển thị về số lượng màu và vùng màn hình là các cấu hình khả thi đối với một card hiển thị.

Mỗi thiết bị đều có các tùy chọn và phần mềm của bạn có thể không cần hỗ trợ tất cả các tùy chọn này. Một ví dụ rất hay là về các trò chơi trên máy vi tính. Nhiều trò chơi đòi

hội ít màu hiển thị và độ phân giải thấp. Nếu cấu hình có ít hơn lượng yêu cầu, chúng sẽ không chạy.

**d) Làm giảm cấu hình phần cứng xác định xuống tới một tập hợp có thể quản lý**

Giả sử rằng bạn không có đủ thời gian và ngân sách để kiểm thử tất cả, bạn cần làm giảm hàng ngàn cấu hình tiềm năng xuống các cấu hình mà bạn cần kiểm thử.

Một cách để thực hiện là đặt tất cả thông tin về cấu hình vào một bảng tính có các cột đề tên hãng sản xuất các phiên bản mẫu và các phiên bản driver cùng các tùy chọn. hình 5.7 là một bảng chỉ ra ví dụ về một bảng xác định rất nhiều các cấu hình máy in. Bạn và đội của bạn có thể xem lại biểu đồ và quyết định xem bạn nên kiểm thử cấu hình nào.

Popularity (1=most, 10=least)	Type (Laser / InkJet)	Age (years)	Manufacturer	Model	Device Driver Version	Options	Options
1	Laser	3	HAL Printers	LDIY2000	1.0	B/W	Draft Quality
5	InkJet	1	HAL Printers	IJDIY2000	1.0a	Color B/W	Draft Quality Draft Quality
5	InkJet	1	HAL Printers	IJDIY2000	2.0	Color  B/W	Art Photo Draft Quality
10	Laser	5	OkeeDohKee	LJ100	1.5	B/W	100dpi 200dpi 300dpi
2	InkJet	2	OkeeDohKee	EasyPrint	1.0	Auto	600dpi

tổ chức thông tin cấu hình của bạn vào một trang máy tính

Hãy chú ý rằng hình 5.7 cũng có các cột thông tin về sự phổ biến các thiết bị cũng như kiểu và tuổi thọ. Khi bạn tạo ra những tách phần tương ứng, bạn sẽ có thể quyết định rằng bạn chỉ muốn kiểm thử những máy in có tuổi thọ dưới 5 năm. Với thông tin về kiểu trong trường hợp này là máy in laser hay mực, bạn có thể quyết định kiểm thử 75% máy laser và 25% máy in mực.

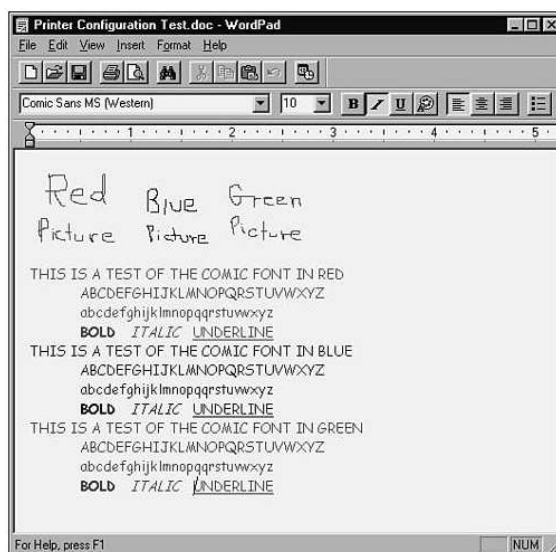
**Chú ý:** Cuối cùng quá trình quyết định để bạn tách phần tương đương các cấu hình thành tập hợp nhỏ hơn phụ thuộc vào bạn và cả đội. Không có một công thức đúng nào. Mọi dự án phần mềm đều khác nhau và sẽ có các tiêu chuẩn lựa chọn khác nhau. Phải đảm bảo rằng mọi người trong đội dự án, đặc biệt là người trong đội quản lý, dự án phải

có ý thức về cấu hình nào đang được kiểm định và những biến số nào dẫn đến việc lựa chọn chúng.

e) Xác định các đặc tính duy nhất của phần mềm của bạn mà có thể làm việc với cấu hình phần cứng

Từ khóa ở đây chính là từ “duy nhất”. Bạn không muốn cũng không cần kiểm thử hoàn toàn phần mềm của mình đối với mỗi cấu hình. Bạn chỉ cần kiểm thử những đặc tính khác nhau (các sự tách phần tương ứng khác nhau) mà tương tác với phần cứng.

Ví dụ, nếu bạn đang kiểm thử một bộ xử lý từ chẳng hạn Word Pad (xem hình 5.8) bạn không cần kiểm thử việc lưu giữ file và đặc tính tải (load) trong mỗi cấu hình. Việc lưu và tải file không có gì để thực hiện đối với việc in ấn. Một bài kiểm thử tốt có thể là bài kiểm thử phải tạo được một tài liệu mà chứa các fonts khác nhau (tất nhiên được lựa chọn bởi việc tách phần tương đương), các size điểm màu sắc và hình ảnh khác nhau...sau đó có thể sẽ cố gắng in tài liệu này trên mỗi cấu hình máy in đã được lựa chọn



bạn có thể sử dụng một document mẫu được tạo bởi nhiều font và style khác nhau để kiểm thử cấu hình một máy in.

Việc lựa chọn các đặc tính duy nhất để thử không phải là dễ. trước hết bạn nên tạo một pass hộp đen bằng cách xem xét sản phẩm của mình và kéo ra những đặc tính rõ ràng. Sau đó hãy nói chuyện với những người khác trong đội, đặc biệt các lập trình viên để nhận được một cái nhìn tổng quan về hộp trắng. Có thể bạn sẽ ngạc nhiên bởi các đặc tính của hộp trắng thậm chí cả những đặc tính chỉ hơi gần với cấu hình.

f) Thiết kế các trường hợp chạy thử trên mỗi cấu hình

Hãy xem xét những điều bạn cần khi thực hiện các bước để kiểm thử mỗi cấu hình. Điều này rất đơn giản

1. Chọn và cài đặt cấu hình kiểm thử kế tiếp từ danh sách.
2. Khởi động phần mềm
3. Tải về các file-configtest.doc
4. Xác nhận rằng file được hiển thị chính xác
5. In tài liệu
6. Xác nhận rằng không có message báo lỗi nào và tài liệu đã in phù hợp với tài liệu chuẩn
7. Ghi bất kì sự sai khác nào là một lỗi

Trong thực tế các bước liên quan khác nhau nhiều hơn rất nhiều bao gồm các chi tiết cụ thể hơn về chính xác cái để thực hiện và cái cần tìm kiếm. Mục đích là để tạo ra các bước mà bất kì ai cũng có thể thực hiện được.

#### **g) Thực hiện các trường hợp kiểm thử trên mỗi cấu hình**

ạn cần chạy các trường hợp kiểm thử và báo cáo cẩn thận các kết quả cho đội của bạn và cho nhà sản xuất nếu cần. Như đã được nói ở phần trước của bài này, xác định các nguồn gốc đặc biệt tới các vấn đề về cấu hình thường khó và tốn nhiều thời. bạn cần kết hợp chặt chẽ với các lập trình viên và những người kiểm thử White-box để tách các nguyên nhân và quyết định xem các lỗi bạn tìm là do phần mềm hay do phần cứng.

Nếu lỗi là rõ ràng do phần cứng, hãy tham khảo trang web của nhà sản xuất để có các thông tin về các lỗi của họ. Hãy chắc chắn xác định được rằng bạn là một tester phần mềm và bạn đang làm việc cho công ty nào. Nhiều công ty có đội ngũ riêng được thành lập để hỗ trợ cho công ty phần mềm viết phần mềm làm việc với phần cứng của họ. Họ có thể yêu cầu bạn gửi các bản copy về phần mềm kiểm thử của bạn các trường hợp kiểm thử của bạn và các chi tiết hỗ trợ nhằm giúp họ cô lập vấn đề.

#### **h) Chạy lại các bài kiểm thử cho đến khi kết quả làm hài lòng cả đội bạn.**

Có một điều không phổ biến trong kiểm thử cấu hình là chạy toàn bộ dự án. Ban đầu một vài cấu hình được thử, sau đó, tất cả các test đều đạt, sau đó, các tập hợp nhỏ hơn, nhỏ hơn nữa các lỗi được sửa. Cuối cùng, bạn sẽ tập chung tại nơi mà không thấy lỗi hoặc nơi mà lỗi vẫn tồn tại trên các cấu hình ít được sử dụng. Tại đây, bạn có thể gọi quá trình kiểm thử cấu hình của bạn là đầy đủ.

### **Obtaining (thu được) the Hardware**

Một thứ mà đã không được đề cập xa như là nơi bạn thu được tất cả những phần cứng này. Thậm chí nếu bạn bỏ rất nhiều công sức và chịu sự rủi ro, để phân chia lớp tương



đương cho các cấu hình để đạt số lượng tối thiểu, bạn vẫn có thể yêu cầu hàng tá các cài đặt phần cứng khác nhau. Nó sẽ là một lời xác nhận quá tốn kém khi phải đi ra ngoài và mua riêng lẻ mọi thứ, đặc biệt, nếu bạn sử dụng phần cứng đó chỉ một lần cho một *test pass*. Dưới đây là một vài ý kiến để khắc phục những vấn đề trên:

- Chỉ mua những cấu hình mà bạn có thể hoặc sẽ sử dụng thường xuyên nhất. Một kế hoạch lớn là được thiết lập cho mọi tester trong nhóm để có phần cứng khác nhau. Nhưng nó là một phương tiện rất hiệu quả khi thường xuyên phải thay đổi các cấu hình khác nhau để kiểm tra trên chúng. Thậm chí nếu nhóm kiểm tra của bạn là rất nhỏ, chỉ gồm 3 hoặc 4 người có một vài cấu hình sẽ trợ giúp rất lớn.
- Sự tiếp xúc với nhà sản xuất phần cứng và yêu cầu họ cho mượn hoặc thậm chí là cho bạn phần cứng. Nếu bạn giải thích rằng bạn đang kiểm tra sự hoạt động của phần mềm trên phần cứng mới và bạn muốn đảm bảo rằng nó làm việc tốt trên phần cứng của họ, chắc hẳn rằng họ sẽ sẵn sàng giúp bạn. Họ cũng có lợi từ kết quả của quá trình kiểm tra, vì vậy hãy nói với họ rằng bạn sẽ cũng cấp cho họ kết quả của quá trình kiểm tra, và nếu có thể hãy copy bản phần mềm cuối cùng cho họ. Đó cũng là một ý kiến hay để xây dựng các mối quan hệ này, đặc biệt nếu bạn kiểm tra được thấy lỗi và cần tiếp xúc với công ty phần cứng để báo cáo về nó.
- Gửi một thông báo hoặc email tới tất cả mọi người trong công ty của bạn hỏi về cấu hình phần cứng nào mà họ đã có trong cơ quan của họ hoặc thậm chí là ở nhà họ và nếu họ cho phép bạn hãy thực hiện một vài bài test trên các phần cứng này. Để thực hiện kiểm tra cấu hình, bạn có thể phải lái xe quanh thành phố, nhưng toàn bộ chi phí cho việc này vẫn ít hơn so với việc mua toàn bộ phần cứng mới.
- Nếu công ty của bạn có ngân sách, thì hãy làm việc với người quản lý dự án để cam kết rằng quá trình kiểm tra của bạn giúp phần mềm làm việc được trên những cấu hình chuyên nghiệp và đảm bảo sự tương thích.

## **Xác định các chuẩn Hardware**

Nếu bạn thấy thích thú khi thực hiện phân tích một chút static black-box, xem xét lại các bản đặc tả mà các công ty phần cứng sử dụng để tạo các sản phẩm của họ, bạn có thể quan sát những những cặp địa chỉ sau. Tìm hiểu một số chi tiết về các đặc tả phần cứng có thể giúp bạn am hiểu hơn những quyết định phân chia tương đương.

Với phần cứng của Apple, bạn hãy tới thăm website về phần cứng của Apple: [developer.apple.com/hardware](http://developer.apple.com/hardware). Ở đây, bạn sẽ tìm thấy thông tin và các link tới quá trình phát triển, quá trình kiểm tra phần cứng và cung cấp các thiết bị driver cho máy tính Apple. Một link Apple khác, [developer.apple.com/testing](http://developer.apple.com/testing), chỉ cho bạn những thông tin kiểm tra đặc biệt, giới thiệu các link tới thư viện các test giúp thực thi việc kiểm tra cấu hình.

Với PCs, link hay nhất là [www.microsoft.com/whdc/system/platform](http://www.microsoft.com/whdc/system/platform). Đây là trang cung cấp kỹ thuật thực thi các guideline, các mẹo, các công cụ để phát triển và kiểm tra phần cứng mở rộng cho việc sử dụng windows.

Microsoft cũng xuất bản một tập các chuẩn cho phần mềm và phần cứng để có thể nhận logo của windows. Thông tin này tại [msdn.microsoft.com/certification](http://msdn.microsoft.com/certification) và [www.microsoft.com/whdc/whql](http://www.microsoft.com/whdc/whql).

## Kiểm tra cấu hình với các Hardware khác

Vì vậy, nếu bạn không kiểm tra phần mềm khi chạy trên một PC hoặc một Mac nào cả? bài này có phải là không cần thiết với bạn không? Không hẳn thế! Mọi thứ bạn học được có thể áp dụng để kiểm tra các hệ thống chung chung hoặc có giữ độc quyền. Đây không phải là vấn đề gặp phải với phần cứng hoặc với việc kết nối tới nó, nếu nó có thể biến đổi liên tục như cơ của bộ nhớ, tốc độ CPU,... hoặc nếu nó kết nối tới phần khác của phần cứng, đưa ra cấu hình phần mềm cần thiết để kiểm tra.

Nếu bạn đang kiểm tra một điều khiển công nghiệp, một mạng, các thiết bị y tế, hoặc một hệ thống điện thoại. Hãy tự hỏi mình những câu hỏi giống nhau khi bạn đang kiểm tra một máy tính để bàn:

- Phần cứng mở rộng nào sẽ hoạt động được với phần mềm này?
- Các mô hình và phiên bản nào của phần cứng là có sẵn?
- Các đặc trưng nào hoặc các tùy chọn nào là hỗ trợ cho phần cứng này?

Tạo ra các phân vùng tương đương của phần cứng dựa trên input từ những người làm việc với thiết bị, quản lý dự án của bạn, hoặc người bán hàng của bạn. Phát triển các test case, thu thập phần cứng lựa chọn, và chạy các test. Kiểm tra cấu hình cho phép áp dụng các kỹ thuật kiểm tra tương tự như bạn đã được học.

## Tổng kết

Bài này cho phép bạn nghĩ về những hướng kiểm tra cấu hình. Nó là một công việc mà một tester mới thường xuyên được phân công bởi vì nó một khái niệm rõ ràng nhất. Vì thế, đây sẽ là một lời giới thiệu tốt để có các kỹ năng tổ chức cơ bản và phân chia lớp tương đương, là một nhiệm vụ mà bạn sẽ phải làm việc với rất nhiều thành viên của đội dự án khác, và là một thứ mà người quản lý của bạn có thể dễ dàng kiểm định lại được kết quả.

Nếu bạn được giao cho việc kiểm tra cấu hình của một dự án, hãy hít thở thật sâu, đọc lại bài này, lập kế hoạch cẩn thận cho bài này, và lập kế hoạch về thời gian. Khi bạn đã

hoàn thành, ông chủ của bạn sẽ giao cho bạn những công việc rất tuyệt vời: ví dụ như kiểm tra khả năng tương thích

## Kiểm thử khả năng tương thích

Trong phần trước, bạn đã được tìm hiểu về kiểm thử cấu hình phần cứng và cách đảm bảo rằng phần mềm sẽ làm việc hoàn toàn với phần cứng mà nó được thiết kế để chạy trên và kết nối với. Phần này đề cập tới lĩnh vực kiểm thử tính tương tương thích giữa các phần mềm.

Việc kiểm thử xem một chương trình có hoạt động tốt với các chương trình khác hay không đã trở nên ngày càng quan trọng vì các khách hàng luôn yêu cầu khả năng chia sẻ dữ liệu giữa các chương trình có kiểu khác nhau, và có tác dụng là có khả năng chạy nhiều chương trình cùng một lúc.

Một chương trình có thể đã từng được giới thiệu (phát triển) như chỉ một ứng dụng. Nó có thể chạy trong môi trường quen biết, thông hiểu, lành tính, tách khỏi bất kì điều kiện nào dẫn tới việc làm nó bị tổn thương (hỏng hóc). Ngày nay, chương trình như vậy dường như vậy cần nhập và xuất dữ liệu tới các chương trình khác, được chạy với các hệ điều hành khác nhau và với các trình duyệt web cũng khác nhau, đồng thời gắn liền với các phần mềm khác mà đang được cùng được chạy trên cùng một phần cứng. Việc kiểm thử khả năng tương thích nhằm đảm bảo rằng sự tương tác này đạt hiệu quả tốt như mong muốn của người sử dụng.

Trọng tâm của bài này là:

- Phần mềm tương thích có nghĩa là gì?
- Các tiêu chuẩn để kết luận khả năng tương thích.
- Các cơ sở nền tảng là gì và chúng có ý nghĩa gì với khả năng tương thích.
- Tại sao khả năng chuyển tải dữ liệu giữa các ứng dụng phần mềm là chìa khóa (mấu chốt) dẫn tới khả năng tương thích.

### Tổng quan về kiểm thử khả năng tương thích

Kiểm thử khả năng tương thích của phần mềm nghĩa là kiểm tra xem phần mềm của bạn có tương tác và chia sẻ thông tin chính xác với các phần mềm khác nhau không. Sự tương tác này có thể xảy ra giữa 2 chương trình chạy đồng thời trên cùng một máy tính, hoặc thậm chí trên các máy tính khác nhau được kết nối Internet ở cách nhau tới hàng nghìn dặm. Việc tương tác cũng có thể đơn giản như việc lưu dữ liệu ở một đĩa mềm và chuyển nó bằng tay tới một máy khác qua những căn phòng.

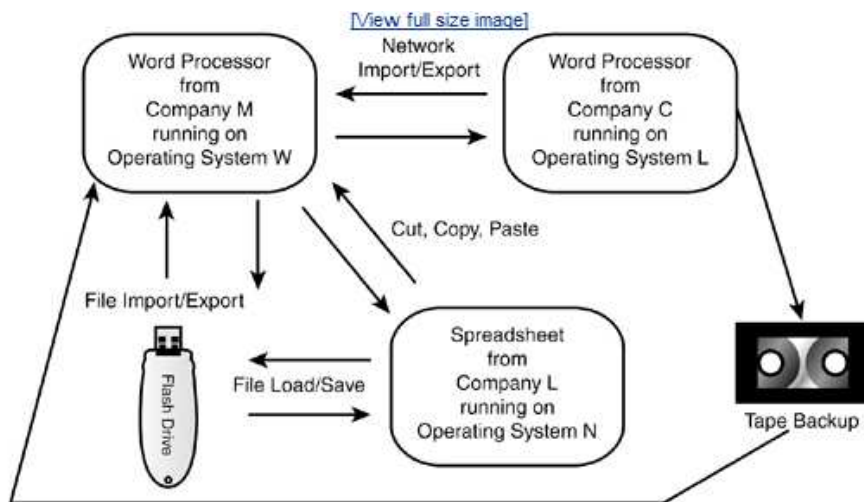
Các ví dụ về phần mềm tương thích:

- Cắt text từ một trang web và dán nó vào một document đã được mở trong bộ xử lý từ của bạn.
- Lưu dữ liệu về các phép tính từ một chương trình có trang bảng tính và sau đó load nó bởi một chương trình bảng tính khác hoàn toàn
- Có một phần mềm sửa ảnh làm việc tốt trên các phiên bản khác nhau của cùng hệ điều hành
- Có bộ xử lý từ load về tên, địa chỉ từ chương trình quản lý những địa chỉ liên lạc của bạn và in ra những thiệp mời và phòng bì riêng cho từng cá nhân
- Nâng cấp một chương trình cơ sở dữ liệu mới và có tất cả các cơ sở dữ liệu đang tồn tại đã load về và làm việc như chúng đã làm với chương trình cũ
- Khả năng tương thích có ý nghĩa gì với phần mềm của bạn phụ thuộc vào việc đội kiểm thử của bạn quyết định tập chung vào cái gì và cấp độ kiểm thử nào được hệ thống (hệ điều hành) phần mềm của bạn sẽ chạy trên yêu cầu. Phần mềm cho thiết bị y tế chạy trên hệ điều hành của riêng nó lưu giữ dữ liệu của nó trên các hộp chứa và không kết nối với bất kỳ thiết bị nào khác sẽ không có sự quan tâm tới khả năng tương thích.

Tuy nhiên, phiên bản thứ 5 của bộ xử lý từ (xem hình 5.9) vốn đọc viết các file từ các bộ xử lý khác cho phép nhiều người sử dụng chỉnh sửa trên Internet và hỗ trợ việc thêm vào các hình ảnh và trang bảng tính từ các ứng dụng khác có vô số điều cần quan tâm về khả năng tương thích.

Nếu bạn được giao nhiệm vụ thực hiện việc kiểm thử khả năng tương thích của phần mềm đối với một loại phần mềm khác, bạn cần có câu trả lời cho các câu hỏi:

- Các nền khác nào (hệ điều hành trình duyệt web, hay một trường điều hành khác) và các phần mềm ứng dụng khác nào mà phần mềm của bạn được thiết kế để tương thích với? Nếu phần mềm bạn đang kiểm thử là một nền (platform), những ứng dụng nào được thiết kế để chạy dưới chúng?
- Các tiêu chuẩn hay hướng dẫn về khả năng tương thích nào nên được làm theo mà xác định cách thức phần mềm của bạn tương tác với các phần mềm khác?
- Kiểu dữ liệu nào phần mềm của bạn sẽ dùng để tương tác và chia sẻ thông tin với các nền (platform) và phần mềm khác?



Khả năng tương thích với các ứng dụng phần mềm khác nhanh chóng trở nên phức tạp.

Việc có được các câu trả lời cho các câu hỏi trên là việc kiểm thử thống kê cơ bản- cả hộp trắng lẫn hộp đen. Nó bao gồm toàn bộ sự phân tích những đặc tả về sản phẩm và bất kì chi tiết hỗ trợ nào. Nó cũng có thể đòi hỏi phải thảo luận với các lập trình viên và cả việc xem xét lại mã (code) để đảm bảo rằng tất cả các đường dẫn đến và đi từ phần mềm của bạn là xác định. Phần còn lại của chương trình này sẽ đề cập chi tiết hơn về các câu hỏi này.

## Platform và các phiên bản ứng dụng

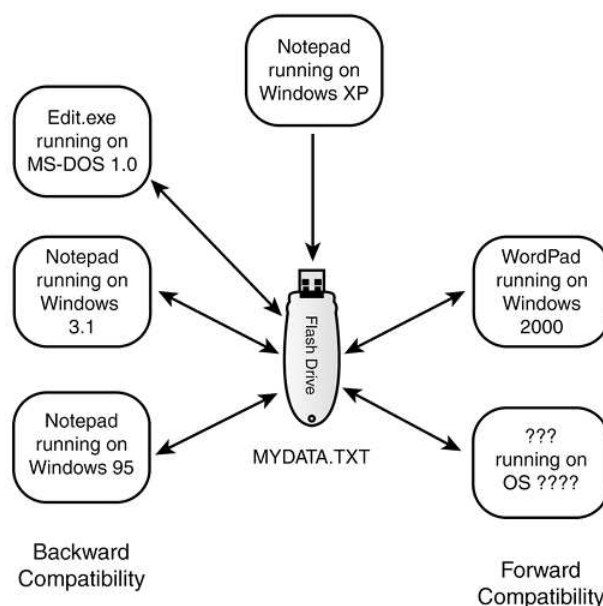
Việc lựa chọn các nền mục tiêu hay các ứng dụng tương thích thực sự là nhiệm vụ quản lý chương trình hoặc nhiệm marketing. Người nào đó rất quen thuộc với khách hàng sẽ quyết định xem phần mềm của bạn được thiết kế cho một hệ điều hành chuyên biệt. Một trình duyệt web hay các platform khác, chúng sẽ xác định phiên bản hay những phiên bản nào phần mềm cần tương thích trên các gói phần hoặc các màn hình start up.

Thông tin này là một phần trong bản đặc tả và nói cho đội phát triển và kiểm thử biết họ nên hướng mục tiêu vào cái gì. Mỗi platform đều có tiêu chuẩn phát triển của riêng mình và nó rất quan trọng trên quan điểm quản lý dự án, để làm cho sanh sách platform này càng nhỏ càng tốt nhưng vẫn đáp ứng được nhu cầu của khách hàng.

### a) Backward and Forward Compatibility (khả năng tương thích trước và sau)

Hai thuật ngữ bạn sẽ nghe liên quan đến kiểm thử khả năng tương thích là khả năng tương thích trước và sau (Backward and Forward Compatibility). Nếu cài gì đó có khả năng tương thích trước, nó sẽ làm việc được với các phiên bản phần mềm trước. Nếu cài gì đó có khả năng tương thích sau, nó sẽ làm việc được với các phiên bản phần mềm tương lai.

Minh họa đơn giản nhất về tương thích trước và sau là với 1 file .txt hoặc một file text (như chỉ ra trong hình 5.10), một file text đã được tạo ra bởi cách dùng notepad 98 chạy dưới Windows 98 là tương thích trước với các đường dẫn về MS – DOS 1.0. Nó cũng tương thích sau với dịch vụ windows XP gói 2 và dường như còn hơn nữa.

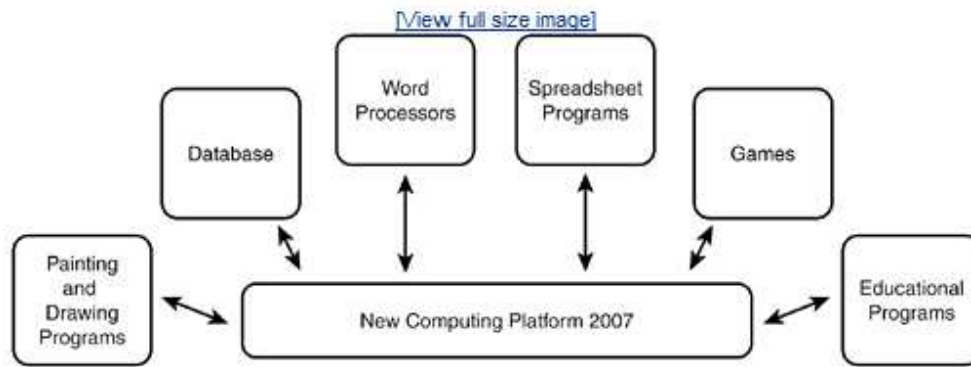


khả năng tương thích trước và sau cho biết những phiên bản nào sẽ làm việc được với phần mềm hoặc các file dữ liệu của bạn

**Chú ý:** Không nhất thiết tất cả các phần mềm hay các file đều phải tương thích trước và sau. Đó là quyết định trong tương lai về sản phẩm do những nhà thiết kế phần mềm cần tạo ra. Dù vậy, bạn nên cung cấp input về việc cần kiểm thử nhiều thế nào để kiểm tra khả năng tương thích trước và sau đối với phần mềm.

## b) The Impact of Testing Multiple Versions (Khó khăn của việc kiểm thử nhiều phiên bản)

Kiểm tra xem nhiều phiên bản của Platform và các ứng dụng phần mềm có làm việc tốt với nhau hay không có thể là một nhiệm vụ khó khăn. Hãy cân nhắc tình huống bạn phải kiểm thử khả năng tương thích một phiên bản mới của một hệ điều hành đã nổi tiếng. Các lập trình viên đã tạo ra vô số lỗi và nâng cấp sự hoạt động, đồng thời thêm vào code rất nhiều đặc tính. Có thể có hàng chục hoặc hàng trăm nghìn các phiên bản của OS. Mục tiêu của dự án là 100% phải tương thích với chúng.



nếu bạn kiểm thử tính tương thích, bạn cần kiểm tra xem các gói ứng dụng có làm việc tốt với nó không

Đây là một việc lớn nhưng nó chỉ là một ví dụ về cách phân chia tương đương có thể được áp dụng để giảm thiểu số lượng công việc.

Chú ý: Để bắt đầu nhiệm vụ kiểm thử khả năng tương thích của phần mềm, bạn cần phân chia tương đương tất cả các sự kết hợp phần mềm có thể thành các tập hợp nhỏ nhất có hiệu quả làm cho phần mềm của bạn tương tác tốt được với các phần mềm khác.

Tóm lại, bạn không thể kiểm thử tất cả hàng nghìn chương trình trên hệ điều hành của mình. Vậy bạn cần quyết định xem phần mềm nào quan trọng nhất để kiểm thử. Từ mẫu chốt ở đây là “tính quan trọng”. Tiêu chuẩn dẫn tới việc quyết định chọn chương trình nào là:

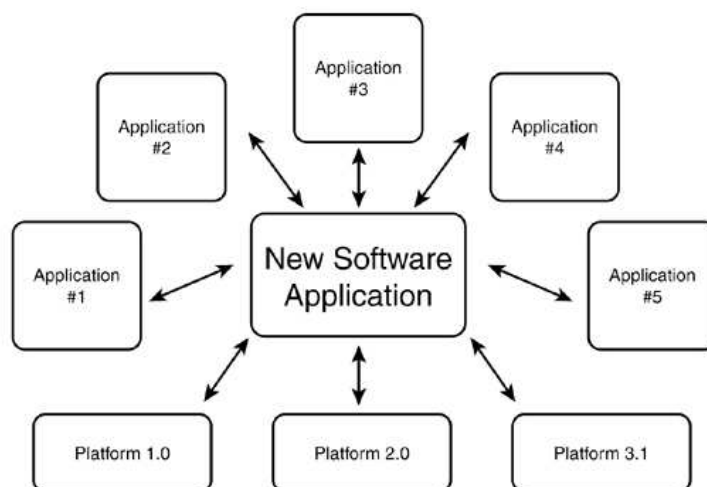
- Sự phổ biến (popularity): Hãy sử dụng dữ liệu về kinh doanh để chọn ra top 100 hay 1000 chương trình phổ biến nhất
- Tuổi thọ (age): Có thể bạn muốn chọn các chương trình hay phiên bản có tuổi đời dưới 3 tuổi.
- Loại (type): Phân chia thế giới phần mềm thành các loại như CSDL, painting (vẽ), writing (viết), về tính toán (accounting), liên lạc,... Hãy chọn các phần mềm từ mỗi danh mục để kiểm thử.
- Hãng sản xuất: Một tiêu chuẩn khác để chọn phần mềm là dựa vào công ty đã tạo ra nó.

Cũng giống trong kiểm thử cấu hình phần cứng không có câu trả lời “text box” đúng, bạn và đội của bạn sẽ cần quyết định cái gì là vấn đề mẫu chốt rồi sau đó sử dụng tiêu chuẩn đó để tạo ra những sự tách phần tương đương đối với phần mềm bạn cần kiểm thử.

Ví dụ trước đó đã đề cập tới việc kiểm thử khả năng thích một platform hệ điều hành mới. Những vấn đề tương thích được áp dụng vào việc kiểm thử một ứng dụng mới (xem hình 5.12). Bạn cần quyết định bạn nên kiểm thử phần mềm của mình trên phiên



bản platform nào và bạn nên kiểm thử phần mềm của bạn với các ứng dụng phần mềm nào khác



Kiểm thử khả năng tương thích của một ứng dụng mới có thể yêu cầu bạn phải kiểm thử nó trên nhiều platform và với nhiều ứng dụng

## Standards and Guidelines

Từ bài này, bạn đã học về việc lựa chọn phần mềm mà bạn sẽ kiểm thử khả năng tương thích với chương trình của bạn. Bây giờ đã đến lúc xem xét cách thức bạn cập nhật kiểm thử trên thực tế. Bước dừng lại đầu tiên của bạn là nghiên cứu các tiêu chuẩn và hướng dẫn hiện có mà có thể áp dụng đối với phần mềm của bạn hay platform.

Thực ra có 2 cấp độ yêu cầu: cấp độ cao và cấp độ thấp. Đó có thể là cách dùng sai thuật ngữ khi đề cập chúng là cao và thấp, nhưng xét ở phương diện nào đó, đó thực sự là như vậy. Các tiêu chuẩn cấp độ cao là các tiêu chuẩn và cảm giác của nó, các đặc tính được hỗ trợ của nó... Các tiêu chuẩn cấp độ thấp là các chi tiết cơ bản, chẳng hạn các định dạng file, các giao thức truyền thông qua mạng. Cả hai loại tiêu chuẩn đều quan trọng và đều cần được kiểm thử để đảm bảo khả năng tương thích.

### a) Các tiêu chuẩn và hướng dẫn cấp độ cao (High-Level Standards and Guidelines)

Liệu phần mềm của bạn có chạy dưới các hệ điều hành Windows, Mac, Linux? Liệu nó có phải là một ứng dụng Web? Nếu vậy, nó sẽ chạy trên các trình duyệt nào? Mỗi điều sau đây coi như một platform và hầu hết đều có những tập hợp tiêu chuẩn và hướng dẫn phải được làm theo nếu một ứng dụng được cho là tương thích với platform.

Ví dụ về việc cấp chứng chỉ logo của Microsoft (*the Certified for Microsoft Windows logo*) (hình 5.13). Để được cấp logo này phần mềm của bạn phải chạy và qua trực sử

kiểm tra về khả năng kiểm thử tương thích bởi một phòng kiểm thử độc lập. Mục đích là để đảm bảo rằng phần mềm của bạn có thể chạy tốt và đáng tin cậy trên hệ điều hành.



The Certified for Microsoft Windows logo có nghĩa là phần mềm đáp ứng tất cả các tiêu chuẩn được đưa ra theo tiêu chuẩn

Một số ví dụ về các yêu cầu về logo về các yêu cầu về logo là các phần mềm phải:

- Hỗ trợ được cho chuột với hơn 3 nút (button)
- Hỗ trợ cài đặt trên ổ đĩa thay cho ổ C: và D:
- Hỗ trợ filename (tên file) dài hơn định dạng DOS 8.3
- Không đọc, viết, mặt khác phải sử dụng được các file hệ thống cũ như win.ini, system.ini, autoexec.bat, hoặc config.sys

Những điều này có vẻ đơn giản và là những yêu cầu đương nhiên, nhưng chúng chỉ là 4 hạng mục trong tài liệu hơn 100 trang. Việc đảm bảo rằng phần mềm của bạn tuân theo tất cả các yêu cầu về logo là điều khó khăn, nhưng nó có giá trị nhiều hơn rất nhiều các phần mềm tương thích

Chú ý: Các chi tiết về logo của windows có thể được xem tại địa chỉ [msdn.microsoft.com/certification](http://msdn.microsoft.com/certification). Các chi tiết cho việc sử dụng logo Apple Mac có thể được xem tại địa chỉ [developer.apple.com/testing](http://developer.apple.com/testing)

## **b) Các tiêu chuẩn hướng dẫn cấp độ thấp**

Thực tế, các tiêu chuẩn hướng dẫn cấp độ thấp quan trọng hơn các tiêu chuẩn cấp độ cao, bạn có thể tạo ra một chương trình chạy trên windows mà không có hình dạng và cảm giác của các phần mềm windows khác. Nó có thể sẽ không được cấp chứng chỉ về *logo certified for Microsoft windows*. Những người sử dụng có thể được cảnh báo về sự khác biệt so với các ứng dụng khác, nhưng họ vẫn có thể sử dụng sản phẩm. Tuy nhiên, nếu phần mềm của bạn là một chương trình đồ họa mà lưu giữ các file của nó trên đĩa ví dụ như các file \*.pict (định dạng file đồ họa Macintosh chuẩn) nhưng chương trình lại không tuân theo các tiêu chuẩn dành cho file \*.pict, thì những người sử dụng sản phẩm

của bạn sẽ không thể xem các file này ở bất kỳ chương trình nào khác. Phần mềm của bạn sẽ không tương thích với chuẩn và sẽ là một sản phẩm có tuổi thọ ngắn.

Tương tự như vậy, các giao thức truyền thôn, những cú pháp lập trình, và bất kỳ phương tiện nào mà các chương trình sử dụng để chia sẻ thông tin phải bám chặt các tiêu chuẩn và hướng dẫn đã được đưa ra.

Các tiêu chuẩn cấp độ thấp thường được cho là hiển nhiên, nhưng từ khía cạnh của một kiểm thử viên, chúng phải được kiểm thử. Bạn nên xử lý các tiêu chuẩn tương thích như một sự mở rộng đối với các đặc tả của phần mềm. Nếu bản đặc tả phần mềm cho biết rằng “phần mềm sẽ lưu và tải các file đồ họa như \*.bmp, \*.jpg, và định dạng \*.gif”, bạn cần tìm kiếm các tiêu chuẩn cho những định dạng này và thiết kế các bài kiểm tra để xác nhận rằng phần mềm thực sự tuân theo các tiêu chuẩn đó.

**Chú ý:** Đừng tin tưởng một cách cần thiết đối với bản dịch về các tiêu chuẩn và hướng dẫn của đội của bạn. Hãy tự tra chúng và phát triển các bài kiểm tra trực tiếp của mình từ nguồn gốc của nó. Hãy nhớ rằng sự khác biệt giữa sự chính xác và tập trung (cô đọng). Bạn không muốn mã tương thích của sản phẩm của mình tập chung hoàn toàn nhưng lại không cô đọng.

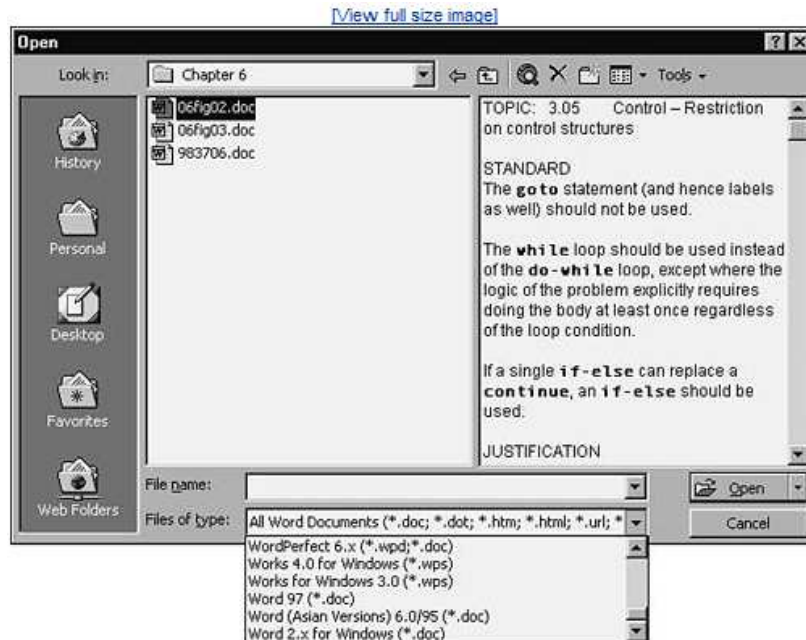
## **Khả năng tương thích khi chia sẻ dữ liệu**

Việc chia sẻ dữ liệu giữa các ứng dụng thực sự là thứ cung cấp sức mạnh cho phần mềm. Một chương trình thứ cung cấp sức mạnh cho phần mềm. Một chương trình được viết tốt mà hỗ trợ và gắn với các tiêu chuẩn cho trước đồng thời cho phép người sử dụng dễ dàng chuyển dữ liệu đến và đi từ phần mềm khác là sản phẩm có khả năng tương thích lớn.

Các phương tiện chuyển tải dữ liệu từ chương trình này tới chương trình khác quen thuộc nhất thì lưu và tải các file đĩa (disk files). Như đã được bàn luận trong các phần trước, việc gắn với các tiêu chuẩn cấp thấp đối với các định dạng file và đĩa là thứ làm cho việc chia sẻ này trở nên khả thi. Dù đôi khi các phương tiện khác được cho là đương nhiên, vẫn cần được kiểm thử khả năng tương thích. Sau đây là một số ví dụ:

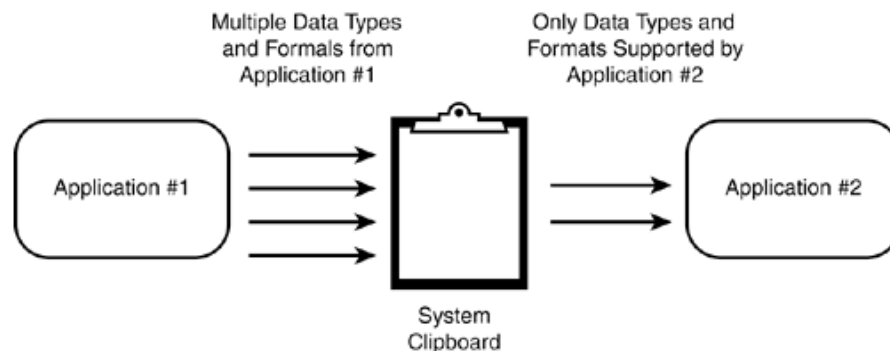
- Lưu file (file save) và tải file (file load) là các phương pháp chia sẻ dữ liệu mà mọi người đều quan tâm. Bạn lưu dữ liệu của mình vào một đĩa mềm (hoặc các phương tiện lưu trữ từ và quang khác) và sau đó di chuyển nó bằng tay tới máy tính khác đang chạy phần mềm khác và tải nó. Định dạng dữ liệu của các file cần đáp ứng các yêu cầu để nó tương thích trên cả 2 máy.
- Nhập file (file import) và xuất file (file export) là các phương tiện mà nhiều chương trình sử dụng để tương thích với các phiên bản cũ hơn của chính chúng và với cả các chương trình khác. Hình 5.14 chỉ ra rằng hộp thoại Microsoft

word file open và một số trong 23 định dạng file khác nhau có thể nhập vào bộ xử lý từ



Microsoft Word có thể nhập tới 23 định dạng file khác nhau

- Để kiểm thử đặc tính nhập file, bạn cần tạo ra các tài liệu kiểm thử trong mỗi định dạng file tương thích – có lẽ bằng cách xử dụng phần mềm gốc mà đã viết nên định dạng đó. Những tài liệu đó có thể cần có các mẫu text khả thi được phân tách tương đương và định dạng nhằm kiểm tra, rằng mã nhập đã chuyển đổi nó hoàn toàn sang một định dạng mới.
- Cut, copy, paste, là các phương pháp chia sẻ dữ liệu giữa các chương trình mà không phải chuyển dữ liệu qua một đĩa. Trong trường hợp này việc chuyển diễn ra một bộ nhớ qua một chương trình sơ cấp gọi là Clipboard. Hình 5.15 chỉ ra cách việc chuyển này diễn ra.



Hệ thống Clipboard tạm thời lưu giữ một loại dữ liệu khác được copy từ ứng dụng này sang ứng dụng khác

- - Clipboard được thiết kế để tổ chức nhiều kiểu dữ liệu khác nhau. Những kiểu thông thường trên Window là text, hình ảnh và âm thanh. Những kiểu dữ liệu này cũng có thể là các định dạng khác nhau. Ví dụ, text có thể là text cũ thuần túy, HTML hay Rich text. Hình ảnh có thể là bitmap, hay metafile hoặc tiff. Bất cứ khi nào một người sử dụng thực hiện thành công cut hay copy, dữ liệu được chọn sẽ được đặt ở Clipboard. Khi anh ta paste nó được copy từ clipboard sang phần mềm đích. Một số ứng dụng có thể chỉ chấp nhận những kiểu dữ liệu hoặc định dạng đang được paste trong chúng. Ví dụ một chương trình painting có thể chấp nhận các hình ảnh nhưng không chấp nhận các text.
  - Hình 5.15 clipboard hệ thống là nơi lưu trữ tạm thời cho các kiểu dữ liệu khác mà đang được copy từ một ứng dụng sang ứng dụng khác.
  - Nếu bạn đang kiểm thử khả năng tương thích của một chương trình, bạn cần đảm bảo rằng dữ liệu của nó có thể được copy hoàn toàn vào hoặc ra từ clipboard đến các chương trình khác. Đặc tính này được sử dụng thông qua việc sử dụng thẻ nhúng lằm và quá thường xuyên, người ta quên có nhiều mã phía sau việc đảm bảo rằng nó làm việc và tương thích qua rất nhiều phần mềm khác nhau.
- DDE (phát âm là D-D-E), COM (mô hình đối tượng chung), OLE (phát âm là oh-lay) là các phương pháp trong window để chuyển dữ liệu qua hai ứng dụng. DDE là viết tắt của dynamic data exchange (trao đổi dữ liệu động) và OLE là viết tắt của object Linking and Embedding (liên kết và nhúng đối tượng). Các platform khác hỗ trợ các phương pháp tương tự.
  - Không cần phải xem xét quá chi tiết các công nghệ trong cuốn sách này, nhưng sự khác biệt đầu tiên giữa hai phương pháp lay và clipboard là với DDE và OLE, dữ liệu có thể đi từ ứng dụng này sang ứng dụng khác trong khoảng thời gian thực. Cut và copy là hoạt động bằng tay đơn giản thủ công. Với DDE và OLE việc chuyển có thể xảy ra một cách tự động. Một ví dụ về cách những phương pháp trên được sử dụng có thể là một báo cáo thành văn được thực hiện trong bộ xử lý từ mà có một biểu đồ hình bánh (pie-chart) được tạo ra bởi một chương trình bảng tính. Nếu tác giả của bài báo cáo copy và paste biểu đồ vào báo cáo đó có thể là một snap-shot về phương diện thời gian của dữ liệu. Tuy nhiên nếu tác giả nối biểu đồ đó vào báo cáo như một đối tượng, khi các số ở dưới cho sự thay đổi của biểu đồ, đồ họa mới sẽ tự động xuất hiện trong bản báo cáo.
  - Đây là tất cả những điều khá thú vị, nhưng đó cũng có thể là một thử thách khi kiểm thử để chắc chắn rằng tất cả các đối tượng đang liên kết, nhúng và việc kiểm tra dữ liệu diễn ra chính xác.
  - Bài này đã giới thiệu cho bạn những điều cơ bản về kiểm thử khả năng tương thích. Trong thực tế, một cuốn sách hoàn chỉnh có thể được viết về vấn đề này, và chỉ một chương thì không thể nói hết được về chủ đề.

Mỗi platform và mỗi ứng dụng đều là duy nhất và các vấn đề tương thích có thể hoàn toàn khác trên một hệ thống khác.

- Khi là một kiểm thử viên mới, bạn có thể được phân công nhiệm vụ kiểm thử phần mềm tương thích của mình. Điều đó có thể hơi là vì nó là một nhiệm vụ quá lớn và phức tạp, nhưng bạn sẽ được giao cho chỉ một phần của toàn bộ công việc. Nếu dự án của bạn là một hệ điều hành mới bạn có thể được yêu cầu chỉ kiểm thử tương thích bộ xử lý từ hoặc các chương trình đồ họa. Nếu dự án của bạn là chương trình về các ứng dụng, bạn có thể được yêu cầu kiểm thử tương thích nó trên nhiều platform khác nhau.
- Mỗi kiểu đều là nhiệm vụ có thể quản lý được mà bạn có thể được xoay sở dễ dàng nếu bạn tiếp cận kiểm thử với ba điều đầu tiên trong đầu.

- Tách phần tương đương tất cả các lựa chọn có thể đối với phần mềm tương thích thành các tập hợp có thể quản lý được. Dĩ nhiên người quản lý của bạn nên đồng ý với danh sách của bạn và hiểu được rủi ro khi không kiểm thử tất cả.

- Nghiên cứu các chuẩn và hướng dẫn cấp độ cao và thấp mà có thể áp dụng vào phần mềm của bạn. Hãy sử dụng phần mềm này như sự mở rộng bản đặc tả về sản phẩm của bạn.

- Hãy kiểm thử tất cả các cách mà dữ liệu có thể đi giữa các chương trình bạn đang kiểm thử. Việc trao đổi dữ liệu này là thử nghiệm để đảm bảo một chương trình tương thích với một chương trình khác.

## Kiểm thử Foreign – Language

Ngày nay hầu hết các phần mềm được tung ra trên toàn thế giới không phải một quốc gia nào đó hay một dạng ngôn ngữ nào đó. Microsoft đã tung ra Windows XP với sự hỗ trợ của 106 ngôn ngữ và thổ âm từ Afirikaans tới Hungari hay Zulu. Hầu hết các công ty phần mềm đều làm như vậy, và thấy rằng thị trường Anh - Mỹ có lượng khách hàng tiềm năng chưa bằng một nửa. Nó làm cho việc kinh doanh có ý nghĩa hơn khi thiết kế và kiểm thử phần mềm có tính toàn cầu.

Phần này đề cập đến những vấn đề liên quan tới kiểm thử phần mềm được viết cho các quốc gia và ngôn ngữ khác nhau. Đây có vẻ là một quá trình đơn giản, nhưng thực tế lại không phải vậy bạn sẽ biết được lý do tại sao.

### Ý nghĩa của từ ngữ và hình ảnh

Bạn đã bao giờ đọc một cuốn số về một thiết bị hay một thứ đồ chơi mà chỉ được chuyển đổi đơn giản sang một ngôn ngữ khác chưa? Có thể là dễ dàng khi chuyển đổi từng từ nhưng để tạo thành một khối tổng thể đúng nghĩa và dễ hiểu đòi hỏi đầu tư nhiều về thời gian và trí tuệ.

Những người phiên dịch tốt có thể làm được điều đó. Nếu họ thành thạo cả hai thứ tiếng, họ có thể biến bản text từ tiếng nước ngoài thành bản dịch có thể đọc được như bản gốc. Thật không may, điều bạn tìm thấy trong ngành công nghiệp phần mềm là thậm chí một bản dịch tốt cũng không đầy đủ.

Tiếng Tây Ban Nha là một ví dụ, liệu có phải là một vấn đề đơn giản khi chuyển một text tiếng Anh sang tiếng Tây Ban Nha? Những loại tiếng Tây Ban Nha nào bạn đang đề cập tới? Tiếng Tây Ban Nha xuất phát từ Tây Ban Nha? Nhưng còn tiếng Tây Ban Nha từ Costa Rica, Peru hay cộng hòa Dominica? Tất cả đều là tiếng Tây Ban Nha nhưng chúng đủ khác nhau để phần mềm được viết cho mỗi ngôn ngữ trong số chúng có thể không đầy đủ từ các ngôn ngữ khác. Thậm chí tiếng Anh cũng có những vấn đề này. Không chỉ có tiếng Anh-Mỹ mà còn có cả tiếng Anh-Canada, tiếng Anh-Australia và tiếng Anh-Anh.

### Các vấn đề về dịch thuật

Quan trọng là bạn hay ai đó trong đội kiểm thử của bạn phải quen thuộc chút ít với ngôn ngữ các bạn đang kiểm thử. Dĩ nhiên, nếu bạn chuyển chương trình của mình sang 32 thứ tiếng khác nhau, điều này có thể rất khó khăn. Giải pháp là ký hợp đồng với một công ty kiểm thử địa phương hóa. Vô số các công ty như vậy trên thế giới có thể thực hiện kiểm thử gần như trên với bất kỳ ngoại ngữ nào. Để biết thêm thông tin, hãy tìm “localization testing” trên Internet.

Không yêu cầu rằng mỗi người trong đội kiểm thử phải nói được thứ ngôn ngữ mà phần mềm đang được địa phương hóa thành, các bạn chỉ cần một người. Nhiều thứ có thể được kiểm tra mà không cần biết các từ ngữ nói gì. Chắc chắn sẽ rất có ích khi biết một chút ít ngoại ngữ, tuy nhiên bạn sẽ thấy rằng, bạn có thể vẫn thực hiện được công việc kiểm thử mà không cần phải thành thạo.

#### a) Sự mở rộng text (text expansion)

Ví dụ dễ thấy nhất về vấn đề dịch thuật có thể được xảy ra là do cái mà được gọi là sự mở rộng text. Dù tiếng Anh có vẻ dài dòng, nhưng thực sự khi được dịch ra các ngôn ngữ khác, chúng ta thường phải dùng nhiều ký tự hơn để nói về cùng một vấn đề.

Hình 5.16 cho thấy kích cỡ của một nút (button) cần mở rộng tới thế nào để lưu giữ text được dịch khi dịch hai từ rất phổ biến của ngôn ngữ máy tính.

Một quy tắc rất hay về dạng thumb là tăng 100% về kích cỡ đối với các từ. Ví dụ, trên một nút, hãy tăng 50% về kích cỡ đối với các câu và đoạn văn ngắn có các cấu trúc điển hình mà bạn có thể thấy ngay trong các hội thoại hoặc các message báo lỗi.



Khi được dịch sang ngôn ngữ khác từ Minimize hoặc Maximize có thể mở rộng về kích cỡ và buộc phải thiết kế lại để có thể lưu giữ được chúng

Bởi vì sự mở rộng này, bạn cần kiểm tra các vùng cẩn thận của phần mềm các vùng của phần mềm mà có thể bị ảnh hưởng bởi các text dài hơn. Hãy tìm kiếm text không được bao hoàn toàn, bị xén bớt hoặc bị gạch nối không đúng. Điều này có thể xảy ra bất kỳ ở đâu- trên màn hình, trong các cửa sổ, trong các boxes, các button,... Đồng thời hãy tìm kiếm các trường hợp nơi text có đủ chỗ để mở rộng nhưng vẫn bị như trên bởi đã đẩy cái gì đó ra ngoài.

Để giải quyết vấn đề trên, một white – box tester có thể nắm bắt được vấn đề đó mà không cần hiểu rõ về ngôn ngữ đó.



## 1. ASCII, DBCS, và Unicode

Bảng mã ASCII sử dụng một byte gồm có 256 ký tự không đủ cho việc biểu diễn các ngôn ngữ khác nhau.

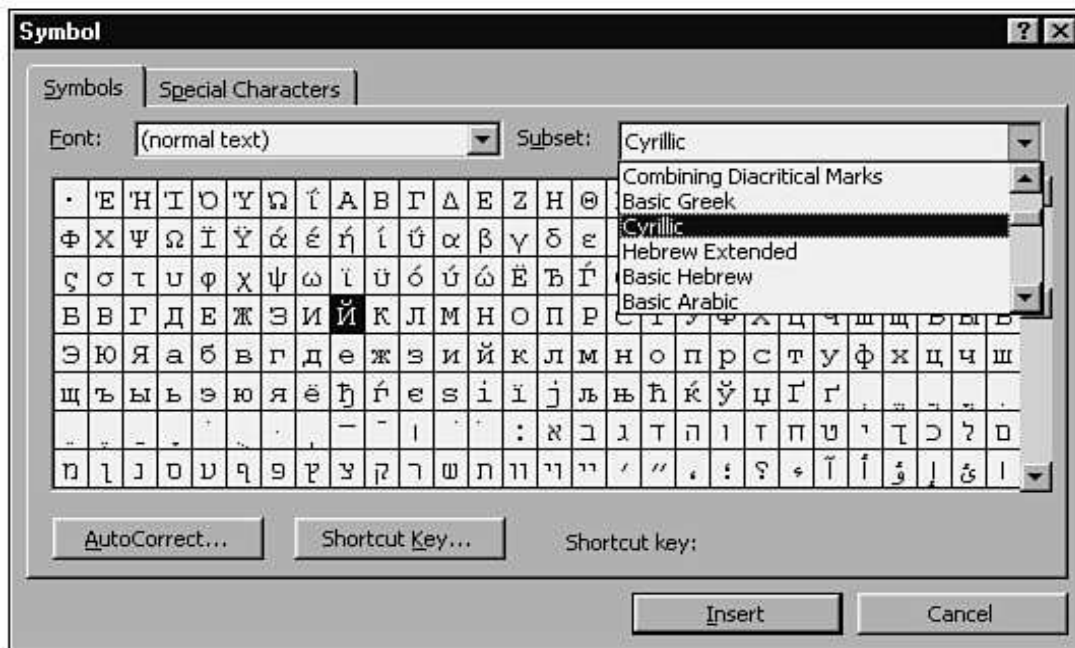
Ví dụ: như tiếng Nhật và tiếng Trung Quốc có hàng nghìn các ký tự đặc biệt.

Vì vậy hệ thống DBCS(Double Byte Character Set) sử dụng hai byte gồm 65.536 ký tự khác nhau để giải quyết cho vấn đề này. Nhưng những trang mã DBCS lại gặp một số vấn đề trở ngại trong khả năng tương thích.

Ví dụ: tài liệu của người Do Thái được tải trên máy tính của người Đức đang chạy một bộ xử lý văn bản tiếng Anh, kết quả là có thể gây ra sai về ngữ pháp.

Giải pháp chung cho những vấn đề này là chuẩn Unicode. Tìm hiểu “ Unicode là gì” từ Unicode Consortium website, [www.unicode.org](http://www.unicode.org)

Bởi vì Unicode là một chuẩn mở rộng hỗ trợ chính bởi những công ty phần mềm, những nhà sản xuất phần cứng, và những nhóm tiêu chuẩn khác, nó đang trở nên thông dụng hơn. Đa số những ứng dụng phần mềm hỗ trợ nó



Hộp thoại này cho thấy sự hỗ trợ của chuẩn Unicode

### 1. Hot Keys và Shortcuts (Phím nóng và phím tắt)

Trong tiếng Anh, nó là “Search”. Trong tiếng Pháp, thì nó là “Rechercher”. Nếu phím nóng để lựa chọn cho sự tìm kiếm “Search” trong phiên bản tiếng Anh phần mềm của các bạn là Alt+S, mà nó cần sự thay đổi trong phiên bản tiếng Pháp.

Trong các phiên bản địa phương hoá phần mềm, bạn cần kiểm tra tất cả những phím nóng và công việc của những phím tắt sao cho dễ sử dụng và mang ý nghĩa gợi nhớ.

### 1. Extended Characters (Mở rộng bảng ký tự)

Bảng mã ASCII, ngoài 26 chữ cái hoa từ A..Z và 26 chữ cái thường từ a..z còn có những ký tự đặc biệt không có trên bàn phím như



, nhưng với sự hỗ trợ của bảng mã Unicode hoặc DBCS thì sự biểu diễn các ký tự đặc biệt trên không là vấn đề lớn.

Để kiểm tra bảng ký tự mở rộng đó bạn có thể kiểm tra việc nhập xuất của các ký tự đó có diễn ra như các ký tự thông thường không. Và kiểm tra xem chuyện gì sẽ xảy ra khi bạn thực hiện thao tác cut, copy và paste giữa các chương trình khác nhau?

### 1. Computations on Characters (Sự tính toán trên các ký tự)

Liên quan đến mở rộng ký tự là cách thức phần mềm tính toán trên các ký tự đó. VD: sắp xếp từ và đổi chữ hoa và chữ thường.

Reading Left to Right and Right to Left (Đọc từ trái qua phải và từ phải qua trái)

Một vấn đề lớn cho bản dịch là một vài ngôn ngữ, như tiếng Do Thái và tiếng Ả rập đọc từ phải sang trái không đọc từ trái sang phải.

Đa số hệ điều hành hỗ trợ xử lý những ngôn ngữ này. Nhưng nó là một nhiệm vụ gập ghềnh như không thể đạt được. Vì thế nó vẫn chưa là một vấn đề đơn giản của việc dịch văn bản.

### 1. Text in Graphics (Text dưới dạng đồ họa)

Một vấn đề phiên dịch khác xuất hiện khi văn bản được sử dụng trong đồ họa.



Word 2000 có những ví dụ của văn bản ở dạng bitmaps thì khó có thể dịch

Khi biểu diễn văn bản dưới dạng đồ hoạ, nếu nó không phù hợp cho việc địa phương hoá phần mềm thì bạn cần thiết kế lại sao cho phù hợp.

## Localization Issues (vấn đề địa phương hóa)

Việc dịch chỉ là một vấn đề nhỏ trong quá trình kiểm thử ngôn ngữ. Với bất kỳ một văn bản dù có độ dài hay bao gồm nhiều kí tự khác nhau như thế nào đi chăng nữa thì đều có thể dịch một cách dễ dàng. Nhưng vấn đề ở đây là làm thế nào để các phiên bản dịch có thể phù hợp với tất cả mọi nơi.

Khi một phần mềm đã được dịch và kiểm tra cẩn thận thì thường được coi là chính xác và tin cậy. Nhưng nếu người lập trình viên không xem xét đến việc có thể xảy ra vấn đề địa phương hoá thì có lẽ phần mềm không còn chính xác, không còn được coi là chất lượng cao nữa.

### 1. Một số ví dụ về sự địa phương hoá

Ví dụ ở nước Mỹ:



Football



Always drive  
on the left

Ví dụ về sự địa phương hoá

Nếu bạn đang kiểm tra một sản phẩm mà sẽ được địa phương hoá, bạn cần xem xét cẩn thận nội dung để chắc chắn rằng nó sẽ thích hợp cho những nơi mà nó được sử dụng.

Danh sách các sản phẩm với nội dung mà chúng ta cần xem xét cho sự địa phương hoá:

Sample documents	Icons
Pictures	Sounds
Video	Help files
Maps with disputed boundaries	Marketing material
Packaging	Web links

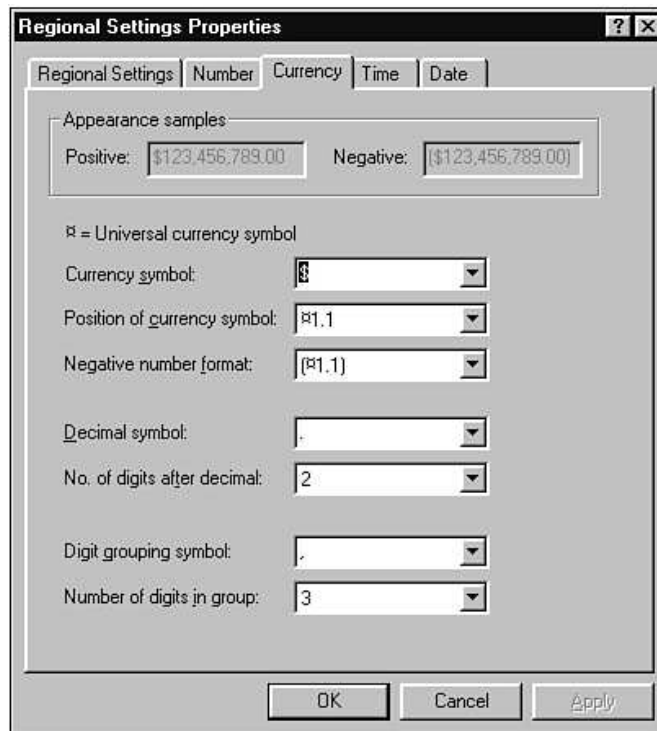
Ví dụ về sự địa phương hoá trong sự kiện: *A Nose too long*:

- Vào năm 1993, Microsoft cho ra mắt hai sản phẩm mới cho trẻ em được gọi là phần mềm sáng tạo ra cách viết và cách vẽ đẹp. Sản phẩm này có đặc điểm là sử dụng người trợ giúp với tên gọi là McZee để hướng dẫn cho trẻ em. Nhiều nghiên cứu đã đi vào thiết kế McZee để lựa chọn ra giao diện, màu sắc, kiểu cách, tính cách ..v..v.
- Không may là sau khi nhiều công việc đã được thiết kế những hình ảnh mà xuất hiện trên màn hình như vậy, một số người trong bộ phận ngoại giao của Microsoft, sau khi nhận được phiên bản sơ bộ của phần mềm họ xem xét họ và cho rằng nó không thể chấp nhận. Lý do: McZee có cái mũi quá dài. Trong nhận thức (văn hoá) của họ, con người với những cái mũi quá lớn là không bình thường. Họ nói rằng sản phẩm sẽ không bán được nếu nó không được địa phương hoá trong địa phương của họ.
- Thế nhưng để tạo ra 2 phiên bản Mczee khác nhau cho mỗi địa phương thì sẽ quá tốn, vì vậy để hoàn thành được công việc thiết kế, thì công việc thiết kế mỹ thuật cho cái mũi đó sẽ là công việc đầu tiên phải làm
- Định dạng dữ liệu

Những địa phương khác nhau sử dụng cách định dạng khác nhau cho những đơn vị dữ liệu như tiền tệ, thời gian, kích thước. Và làm thế nào để có thể định dạng phù hợp với từng địa phương đó chính là địa phương hoá chứ không phải là bản dịch.

Một chương trình Anh Mỹ đang được đưa ra, chương trình này làm việc với đơn vị Inch, đã cho thấy không phải đơn giản là từ đơn vị inch thông qua một bản dịch văn bản để chuyển về centimet. Điều đó yêu cầu là phải thay đổi mã code như thế nào để có thể chuyển đổi được những công thức đó...

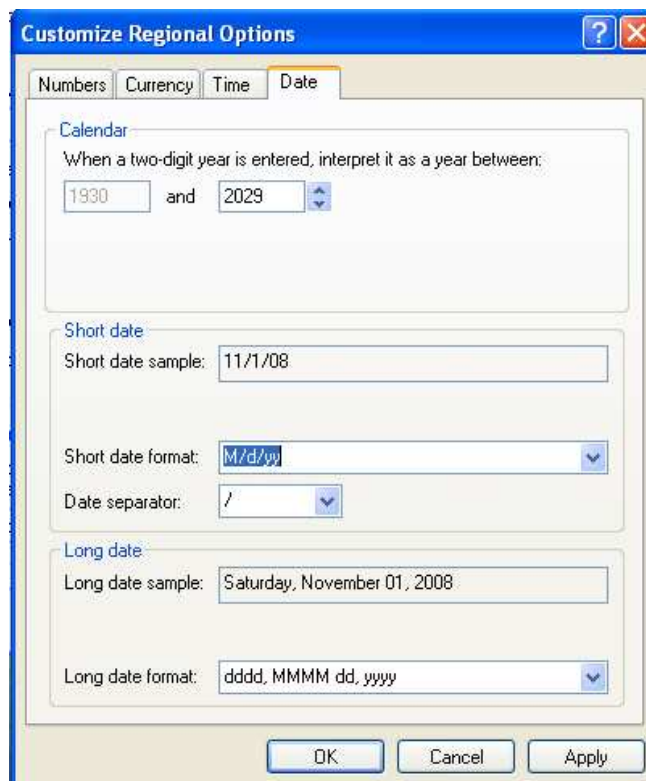
Ví dụ: Những tùy chọn và những thiết đặt trong windows cho phép một người sử dụng lựa chọn số, tiền tệ, thời gian, và ngày tháng sẽ được hiển thị:



Thiết đặt các thuộc tính địa phương hoá trong windows

*Chú ý :*

- Làm thế nào để việc lựa chọn ngày tháng không ảnh hưởng gì đến việc xử lý nội tại trong hệ điều hành và trong các phần mềm khác.
- Ví dụ: Bảng ngày tháng trên “*Customize regional options*” là kiểu ngày tháng ngắn.
- Nếu bạn là tester phần mềm được địa phương hoá bạn sẽ phải quen thuộc sử dụng với tất cả các đơn vị đo lường để có thể đạt đến đích cuối cùng. Để kiểm tra phần mềm một cách đúng đắn, bạn cần tạo các lớp tương đương của việc phân chia dữ liệu.

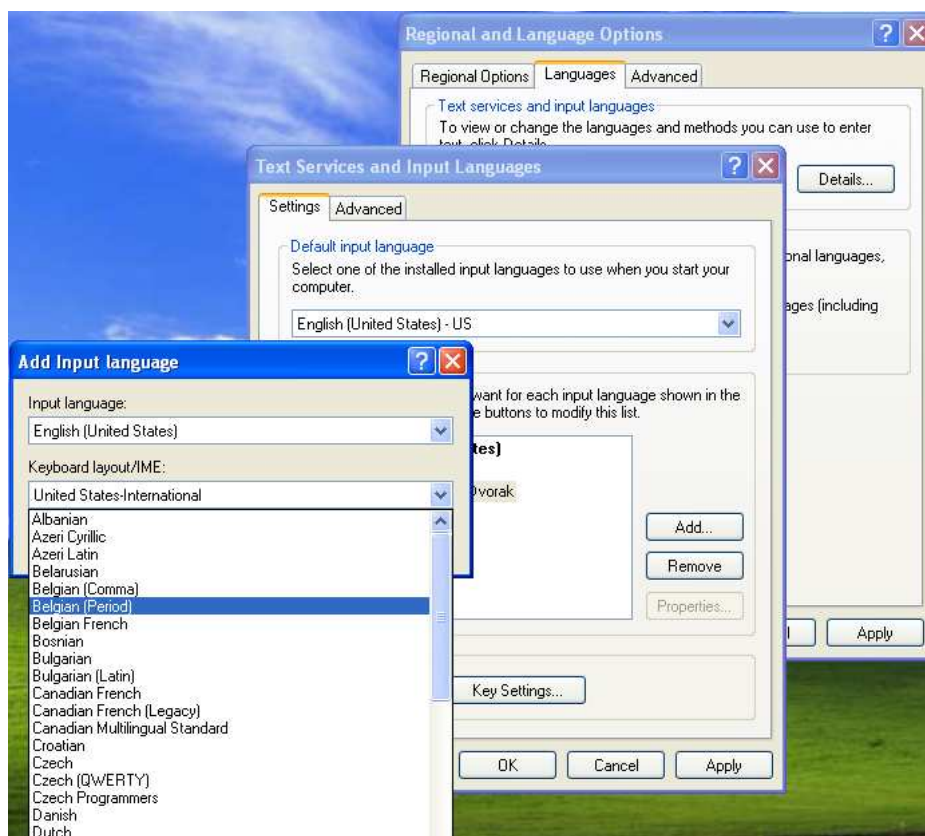


Tuỳ chọn địa phương hoá ngày, tháng, số, tiền tệ

### **Configuration and Compatibility Issues (Những vấn đề về cấu hình và sự tương thích)**

Việc kiểm thử về cấu hình và kiểm thử về tính tương thích là rất quan trọng khi kiểm thử địa phương hoá những phiên bản của phần mềm. Những vấn đề này có thể nảy sinh khi phần mềm tương tác với phần cứng và với các phần mềm khác.

#### **1. Những vấn đề về cấu hình**



Windows XP hỗ trợ 106 ngôn ngữ và 66 cách trình bày bàn phím khác nhau

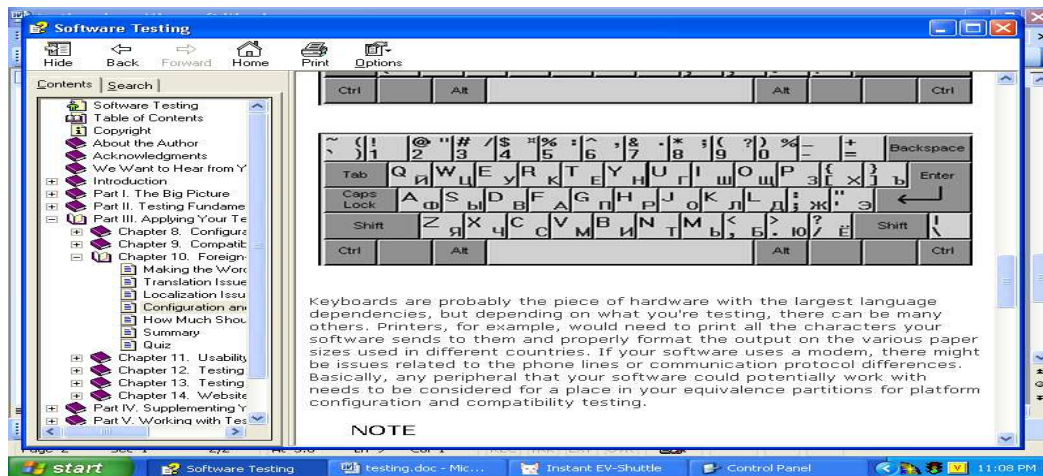
Ví dụ về ba cách trình bày bàn phím khác nhau thiết kế cho những nước khác nhau:



Bàn phím nước Ả Rập



Bàn phím nước pháp



Bàn phím nước Nga

Bạn sẽ thấy mỗi bàn phím biểu diễn bằng những từ khoá đặc biệt đối với ngôn ngữ của nước mình, nhưng vẫn lấy tiếng anh làm đặc tính chung. Tiếng anh được coi như là một ngôn ngữ thứ hai trong nhiều nước, và cho phép bàn phím sử dụng cả phần mềm bằng tiếng bản xứ của mình lẫn tiếng anh.

- Các thiết bị khác:
  - Máy in: Giải sử bạn cần in tất cả các đặc tính phần mềm của bạn gửi tới cho ai đó và đúng định dạng đầu ra trên nhiều kích thước giấy được dùng trong những nước khác nhau.
  - Nếu phần mềm của bạn sử dụng Modem thì nó sẽ liên quan đến đường điện thoại hay những giao thức truyền thông khác nhau ...
  - Như vậy có thể thấy, bất kỳ một thiết bị ngoại vi nào mà phần mềm của bạn có khả năng làm việc với thì cần được xem xét một vùng tương đương cho sự thử về tương thích cấu hình.



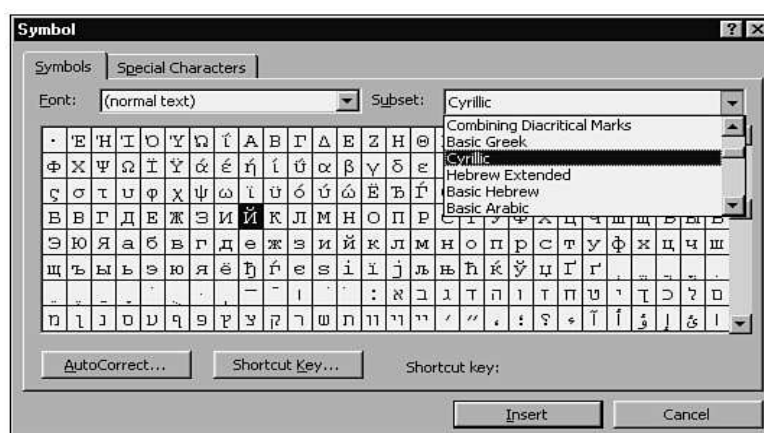
- Chú ý:

- Khi thiết kế những phân vùng tương đương, bạn đừng quên xem xét tất lại cả những phần cứng và phần mềm có liên quan đến. Bao gồm phần cứng, ổ cứng, và hệ điều hành.

#### b) Những vấn đề về sự tương thích

- Cũng như sự thử cấu hình, sự thử tương thích dữ liệu đảm nhiệm một ý nghĩa hoàn toàn mới khi bạn thêm sự địa phương hoá vào chương trình.

- Hình ảnh sau cho thấy sự thử tương thích dữ liệu của phần mềm được địa phương hoá có thể trở lên khá phức tạp thế nào.



Sự phức tạp về tính tương thích của phần mềm được địa phương hoá

Trong thời gian dữ liệu dịch chuyển vòng tròn, với tất cả sự chuyển biến và xử lý những đơn vị đo lường và kí tự mở rộng như vậy thì đã xuất hiện nhiều chỗ lỗi. Một số lỗi này có thể đúng như trong thiết kế.

- Chẳng hạn như:
- Chuyện gì sẽ xảy ra khi dữ liệu được di chuyển từ ứng dụng này sang ứng dụng khác nếu nó cần phải thay đổi những định dạng?
- Nó cần phải được tự động chuyển đổi, hay người sử dụng chuyển đổi ?
- Chú ý:
- Để tránh tốn nhiều thời gian, chi phí cho địa phương hoá phần mềm cho phù hợp với nhiều nơi thì chúng ta nên địa phương hoá ngay từ khi dự án bắt đầu.
- Số lượng kiểm thử sự địa phương hoá là một việc khó xác định vì tùy thuộc vào vấn đề địa phương hoá và kinh nghiệm của tester về vấn đề địa phương hoá đó.
- Trong việc kiểm thử địa phương hoá, Những người kiểm thử hộp trắng sẽ kiểm tra mã code, sử dụng các đơn vị đo lường, và các đặc tính mở rộng ... Những người kiểm thử hộp đen cần thận xem xét spec và sản phẩm để địa phương hóa những vấn đề như văn bản trong đồ họa và những vấn đề cấu hình

## Kiểm thử khả năng tiện dụng (tính khả dụng)

Giao diện người dùng (hoặc UI) là những phương tiện mà bạn thường tương tác với một chương trình phần mềm. Sự thật không có một quy ước UI.

Những máy tính đầu tiên có những cái công tắc bật và hệ thống bóng đèn. Trong những năm 60 và những năm 70, tập giấy, thẻ đục và những máy điện báo đánh chữ là giao diện người dùng. Tiếp theo là hệ thống video, màn hình giám sát và trình soạn thảo như MS-DOS. Ngày nay chúng ta đang sử dụng những máy tính cá nhân với những hệ giao diện đồ họa phức tạp (GUIs).

Mặc dù những UI này rất khác nhau, về mặt kỹ thuật nó được cung cấp cùng sự tương tác với những phương tiện máy tính để trao đổi đầu vào và nhận được đầu ra.

### Kiểm thử giao diện người dùng

#### 1. Cho phép các tiêu chuẩn và hướng dẫn

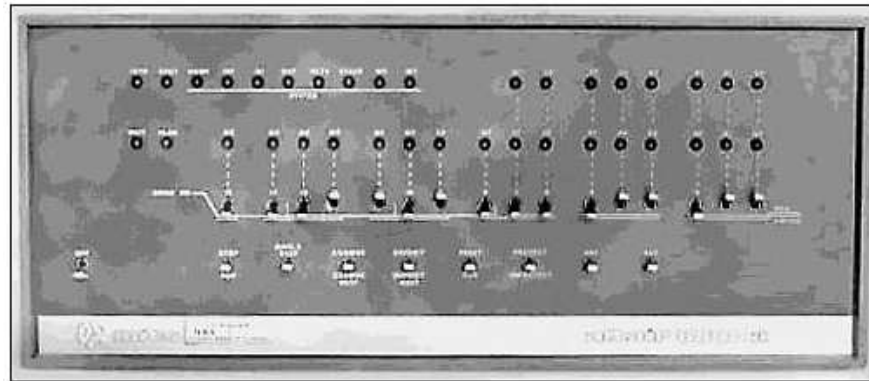
Một đặc tính đơn quan trọng giao diện người dùng là phần mềm của bạn cho phép tồn tại chuẩn và hướng dẫn. Nếu phần mềm của bạn chạy trên nền tảng xác định như: Mac hoặc Window là những chuẩn được thiết lập. Apple thì định nghĩa trong cuốn sách *Maccintosh Human Interface Guidelines*, sách của Microsoft là *Microsoft Windows User Experience*. Mỗi cuốn sách đều đưa ra những chi tiết phần mềm chạy trên mỗi nền tảng thì người dùng nhìn và cảm nhận như thế nào. Mọi thứ đều được định nghĩa khi sử dụng các ô check thay cho các nút chọn cho tới các thuộc tính sử dụng như thông tin, cảnh báo, thông báo, phê bình.



Các tiêu chuẩn và hướng dẫn

## 1. Trực giác

Vào năm 1975 MITS Altair 8800 đã đưa ra bán máy tính cá nhân đầu tiên. Giao diện người sử dụng của nó không có cái gì ngoài những các công tắc và bóng đèn không trực giác để sử dụng:



Hình ảnh chiếc máy tính cá nhân đầu tiên

Ngày nay người sử dụng muốn nhiều hơn cho phần mềm của họ cái Altair cung cấp. Mọi người từ ông già đến những đứa trẻ đều sử dụng máy tính trong cuộc sống hàng ngày. Máy tính cùng với nhiều giao diện là cái mà người sử dụng không hiểu rõ khi sử dụng.

Vì vậy người kiểm thử giao diện có thể áp dụng những cái sau đây để xem giao diện trực quan như thế nào:

## 1. Nhất quán

Nhất quán với phần mềm của bạn và những phần mềm khác là một chìa khoá của thuộc tính. Người sử dụng sẽ phát triển những thói quen và mong chờ nếu họ làm mọi cái một cách xác định trong một chương trình, mặt khác sẽ cùng cách đó trong hệ thống tương tự.

Sự trái ngược khi người sử dụng từ chương trình này đến chương trình khác. Nó sẽ là vấn đề nếu sự mâu thuẫn này có trong cùng một chương trình, nếu có một cái chuẩn cho phần mềm của bạn hoặc nền tảng của bạn cho phép nó. Nếu không, chú ý đặc biệt những đặc tính phần mềm của bạn để chắc chắn rằng những thao tác tương tự được thực hiện tương tự.

## 1. Linh hoạt

Trạng thái nhậy: Phần mềm linh hoạt cung cấp cho ta nhiều lựa chọn và nhiều cách hoàn thành cùng một nhiệm vụ. Kết quả đều thêm vào các đường trong trạng thái khác nhau

của phần mềm. Những sơ đồ chuyển tiếp trạng thái sẽ trở lên phức tạp hơn bạn sẽ cần tiêu chí thời gian nhiều hơn khi ra quyết định đường dẫn nào nối với nhau để kiểm tra.

Điểm cuối trạng thái và bỏ qua: Đây là hiển nhiên khi phần mềm có sức mạnh- những người sử dụng rất quen thuộc với phần mềm trực tiếp và có thể bỏ qua nhiều sự nhắc nhở sau những cửa sổ và đi đến đâu mà họ muốn đi.

Dữ liệu vào và ra: Người sử dụng có rất nhiều cách khác nhau để nhận và xem kết quả của họ. Nhập một đoạn vào trong WordPad bạn cần đánh nó, load nó từ sáu cái định dạng khác nhau, chèn một đối tượng hoặc kéo nó bằng chuột từ một chương trình khác.

## 1. Thoải mái

*Sự thích hợp:* Phần mềm có thể nhìn và cảm nhận thích hợp cho những cái nó làm và người sử dụng nó. Một ứng dụng tài chính sẽ diễn rõ với những màu sắc sắc sỡ và hiệu ứng âm thanh. Một không gian game trên một tay nào đó sẽ là mất thời gian với những quy tắc. Không nên quá loè loẹt hoặc quá rõ ràng cho một nhiệm vụ khi thực hiện.

*Dùng lỗi:* Một chương trình cần phải cảnh báo người sử dụng và cho phép người sử dụng và cho phép họ khôi phục dữ liệu khi bị mất bởi lỗi.

*Sự thực hiện:* Việc nhanh chóng không phải thường xuyên là tốt. Nhiều chương trình đã loé sáng lỗi những thông báo quá nhanh để đọc. Nếu hệ thống chậm, nó sẽ gửi ít nhất những phản hồi đủ dài để nó chỉ ra nó vẫn làm việc hay không. Thanh Status bar là cách phổ biến trong trường hợp này.

## 1. Đúng đắn

Tính tiện dụng phải thừa nhận là hơi mới và thường có thể bỏ đi được với giải thích. Tuy nhiên tính chính xác thì không thể. Khi bạn kiểm thử tính chính xác, bạn sẽ phải kiểm thử giao diện đó có giả định để làm.



là một ví dụ cho một giao diện không chính xác

Hình chỉ ra một hộp thông báo từ quét trong Windows. Hộp xuất hiện khi bắt đầu quét và nó cung cấp cho người dùng khi sử lý. Thật không may là nó không hoạt động. Quét

tiếp tục cho đến khi hoàn thành. Nếu ta nhấn vào nút Abort với đồng hồ cái thì nó sẽ dừng quét sẽ gây lỗi. Bạn sẽ nhận được điều tồi tệ. Bạn sẽ chú ý những phần riêng này.

## 1. Hữu ích

Đặc tính cuối cùng của giao diện người dùng là tính hữu dụng, Nhớ là bạn sẽ không quan tâm đến phần mềm nếu nó không hữu ích. Khi bạn xem sản phẩm xác định, chuẩn bị test hoặc thực sự thử bạn hãy hỏi bản thân mình nếu tính năng bạn nhìn thấy thực sự đóng góp vào giá trị phần mềm. Điều đó có giúp chúng ta sử dụng những phần mềm có ý nghĩa. Nếu như bạn nghĩ nó không cần thiết, hãy tìm một vài người xem tại sao chúng lại ở trong phần mềm. Những đặc tính thừa thãi đó sẽ ẩn chương trình hoặc màn hình sẽ xấu đi cho người dùng và có ý nghĩa thêm phần thử của bạn.

## **Phần mềm dành cho những người khuyết tật (Kiểm tra tính dễ tiếp cận)**

Một đề tài quan trọng đã được đưa ra nhằm kiểm tra sự ảnh hưởng của tính dễ tiếp cận tác động đến những người tàn tật.

Dân số ngày càng được già hoá và sự thâm nhập của công nghệ gần như vào mọi khía cạnh của cuộc sống chúng ta, chính vì vậy tính tiện dụng của phần mềm trở nên quan trọng hơn trong cuộc sống hàng ngày.

Mặc dù có rất nhiều kiểu tàn tật, và dưới đây là một trong những cách rất khó khi sử dụng máy tính và các phần mềm máy tính:

### 1. Sự suy giảm thị giác

Chứng mù màu, cận thị và viễn thị, nhìn trong đường hầm là những ví dụ điển hình của sự giới hạn trực quan. Không chỉ ít người mà rất nhiều người gặp khó khăn khi sử dụng phần mềm. Thử nghĩ về việc khi ta nhìn con trỏ chuột được định vị ở trong văn bản hay nó xuất hiện rất nhỏ ngay trên màn hình. Nếu bạn không thể nhìn thấy màn hình máy tính rồi thì sẽ ra sao?

### 1. Sự suy giảm thính giác

Có một số người bị điếc một bên tai hoặc toàn phần, chỉ cho phép nghe ở những tần số nhất định hay các tạp âm đặc biệt. Một người như vậy có thể không nghe thấy những âm thanh trên video, phần trợ giúp người dùng hay những cảnh báo của hệ thống.

### 1. Khó khăn khi di chuyển

Bệnh tật hay thương tích đều có thể là nguyên nhân làm cho con người ta suy yếu về sức khoẻ, sút cân hoặc mất đi hoàn toàn sự điều khiển bằng tay, chân. Nó có thể sẽ là điều rất khó khăn khi sử dụng hoặc không thể sử dụng được bàn phím và chuột. Ví dụ,

họ không thể ấn nhiều lần một phím tại một thời điểm hay không thể nhìn thấy phím đó. Chính xác hơn là rất khó hoặc không thể di chuyển chuột được.

### 1. Khuyết tật về ngôn ngữ và nhận thứ

Nếu ai đó gặp vấn đề khi khó đọc hay trí nhớ kém có thể rất khó khi sử dụng giao diện phức tạp. Những mục được phác thảo trước đó trong bài này và có thể đều ảnh hưởng đến những người sử dụng ngôn ngữ một cách khó khăn và khả năng nhận thức kém.

#### 1. Pháp luật yêu cầu

Ở Mỹ, ba luật đã được ban hành và các nước khác đang xem xét và chấp nhận những luật tương tự và có đến hơn 15 mục hợp pháp đã được điều chỉnh.

Khu vực 255 của Telecommunications Act yêu cầu tất cả phần mềm và phần cứng khi truyền thông tin qua Internet, mạng máy tính hay qua đường điện thoại phải được thích hợp sử dụng cho cả người tàn tật. Nếu nó không trực tiếp sử dụng được thì nó phải tương thích.

#### 1. Những đặc tính dễ tiếp cận trong phần mềm

Phần mềm được viết ra có thể được tiếp cận một trong số hai cách. Dễ nhất là tận dụng sự hỗ trợ để xây dựng trên nền tảng hệ điều hành có sẵn. Phần mềm của bạn chỉ cần tuân theo các chuẩn theo hệ điều hành để giao tiếp được với bàn phím, chuột, card âm thanh và màn hình là có khả năng truy cập. Hình 5.30 cho thấy một ví dụ của Window về khả năng truy cập những bảng điều khiển.

Nếu bạn kiểm tra phần mềm không chạy trên nền hệ điều hành hiện tại mà chỉ chạy trên nền hệ điều hành của riêng phần mềm đó thì nó sẽ cần phải được tính đến tính dễ tiếp cận và những tính năng trên lý thuyết cần phải được chỉ rõ, phải được lập trình và cần được kiểm tra.

Hiển nhiên sau cùng khi sự chạy thử sẽ tốt hơn so với ban đầu. Nhưng chưa chắc phần mềm đã chạy tốt hơn lúc đầu, cứ giả sử như vậy. Chính vì vậy nên ta cần phải kiểm tra những đặc tính dễ tiếp cận trong cả hai trường hợp để chắc chắn hơn cho phần mềm rằng chúng tuân theo các chuẩn.



Những đặc tính dễ truy cập của Windows được tập hợp từ bảng điều khiển

### 1. Chú ý

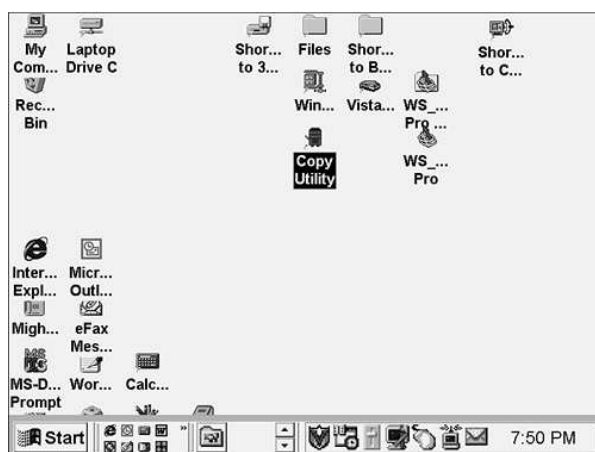
Nếu bạn kiểm thử cho tính khả dụng của sản phẩm, chắc chắn để tạo ra những trường hợp đặc biệt điển hình cho tính dễ tiếp cận. Bạn sẽ cảm thấy tốt nếu phần nào được kiểm tra kỹ.

Mỗi hệ điều hành thì hơi khác nhau trong những đặc điểm phần mềm đưa ra, nhưng họ đều cố gắng làm nó dễ dàng hơn cho những ứng dụng dễ tiếp cận được cho phép. Window cung cấp một số khả năng sau đây.

Bộ âm thanh tạo ra một thông báo trực quan bất cứ khi nào mà hệ thống yêu cầu.

Trình phát âm thông tin với những chương trình để hiển thị những đoạn chú thích cho bất kỳ âm thanh hay tiếng nói nào. Những đoạn chú thích này cần được lập trình sẵn vào trong phần mềm của bạn.

Độ tương phản cao cần được thiết lập vào màn hình với những màu và phông chữ thiết kế để cho những người bị suy giảm về thị giác có thể đọc được.



Máy để bàn Windows giao diện có thể được chuyển sang chế độ phân giải cao này để cho những người bị tàn tật dễ sử dụng hơn

## Tóm tắt

Ghi nhớ định nghĩa về lỗi ở bài 1. Phần mềm mà khó hiểu và khó sử dụng thì người kiểm thử dùng mắt của mình để kiểm thử sẽ rất chậm và rất khó.

Một người kiểm thử phần mềm sẽ kiểm tra tính tiện dụng của phần mềm. Bạn là người đầu tiên sử dụng phần mềm, có nghĩa là người đầu tiên và cũng là người cuối cùng đưa ra những dạng của lỗi giao diện. Giao diện có thể được dùng hay không là do nó ảnh hưởng đến giác quan của bạn như thế nào, khách hàng sẽ phát hiện ra những lỗi giao diện đó.

Nếu bạn đang kiểm thử giao diện của một sản phẩm mới, thì nên chiếu tới danh sách những tiêu chuẩn trong bài này để biết cái gì là có lợi nhất. Nếu nó không đúng tiêu chuẩn này, nó được coi là một lỗi, và nếu nó là một lỗi có khả năng sửa thì nên sửa vì nó có lẽ đã là quy định.



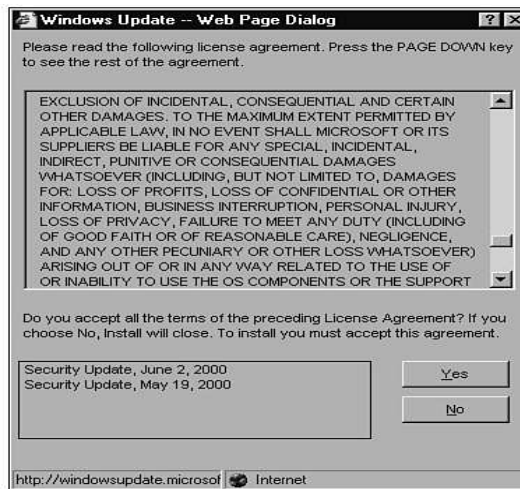
# Kiểm thử tài liệu

Nhưng ngày nay tài liệu phần mềm không chỉ đơn giản là tệp *Readme* nữa nó bao gồm nhiều loại tài liệu khác nữa. Vì vậy việc kiểm thử tài liệu cũng phức tạp hơn.

## Các kiểu tài liệu phần mềm

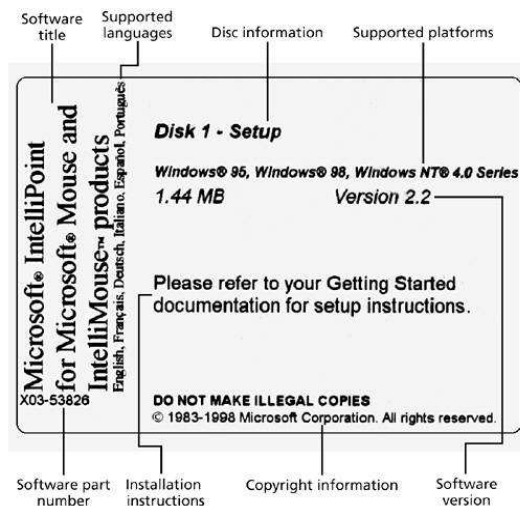
Chúng ta sẽ cùng đi tìm hiểu xem những tài liệu trong phần mềm bao gồm những gì:

- **Đóng gói văn bản và đồ họa** (*Packaging text and graphics*)
- Chúng ta hình dung việc đóng gói như là việc một cái hộp bìa cứng được bọc lại bên ngoài một lớp giấy
- Tài liệu chứa những hình ảnh được chụp từ phần mềm, danh sách những đặc tính, những yêu cầu của hệ thống, và những thông tin về bản quyền.
- *Đưa ra những tài liệu quảng cáo và tiếp thị* (*Marketing material, ads, and other inserts*)
- Khi sản phẩm phần mềm hoàn thành chúng ta thường quan tâm tới nó nhiều hơn là các tài liệu liên quan tới sản phẩm đó. Nhưng chúng lại là một trong những công cụ quan trọng trong việc đẩy mạnh bán sản phẩm phần mềm
- Thông tin mà chúng ta đưa cho khách hàng phải là những thông tin đúng để khách hàng có cảm giác là chúng ta luôn coi trọng họ.
- *Đăng ký mua sản phẩm* (*Warranty/registration*)
- Khách hàng sẽ được dùng thử phần mềm đó
- Nếu sản phẩm được chấp nhận, Khách hàng điền vào phiếu đăng ký mua sản phẩm và gửi đến để nhà cung cấp.
- *EULA* (*End User License Agreement*)
- Đây là hồ sơ pháp lý mà khách hàng đồng ý khi mua sản phẩm với điều kiện:
  - Không được sao chép phần mềm đó
  - Nếu có lỗi xảy ra khách hàng không được kiện nhà sản xuất
- EULA có thể được xuất hiện trong quá trình cài đặt
- Có khi được ghi trong đĩa CD cùng với phần mềm
- Hoặc trong những phong bì gửi tới khách hàng



*EULA là một phần của tài liệu phần mềm, giải thích thời hạn pháp lý để sử dụng phần mềm .*

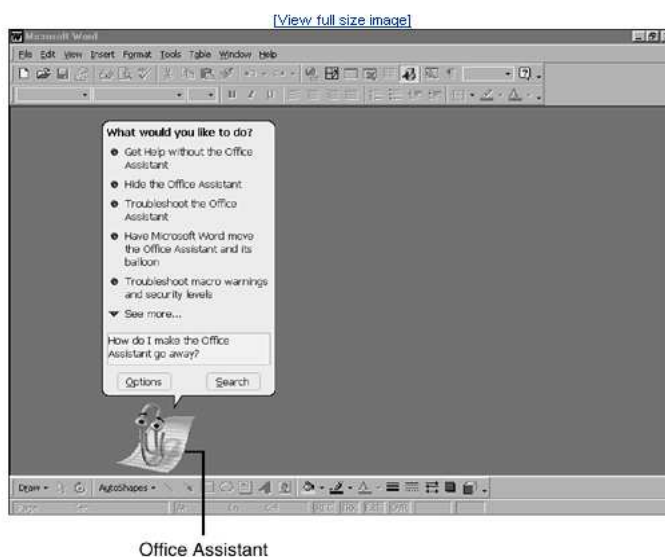
- **Nhãn và nhãn hiệu ( Labels and stickers )**
- Labels and stickers có thể xuất hiện trên:
  - Phương tiện truyền thông
  - Trên hộp
  - Hoặc trên tài liệu
- Labels and stickers cùng với số serial được đặt trong phong bì cùng với EULA



*Một ví dụ về một nhãn đĩa và tất cả những thông tin mà chúng ta cần kiểm tra.*

- **Những chỉ dẫn cài đặt và cài đặt (Installation and setup instructions )**
- Thông tin này được tin trực tiếp trên đĩa CD đôi khi được ghi trong đĩa CD cùng với phần mềm.
- Nếu phần mềm đó phức tạp thì nó có thể là toàn bộ tài liệu cài đặt.
- **Tài liệu của người sử dụng (User's manual)**

- Sự xuất hiện của tài liệu trực tuyến với sự linh hoạt và hữu ích làm cho tài liệu trên giấy không còn được thông dụng như trước nữa.
- Ngày nay hầu hết các phần mềm đều hướng tới sự nhỏ gọn ngay từ những chi tiết đầu tiên.
- Những tài liệu trực tuyến có thể được phân bố trên các phương tiện truyền thông hoặc trên những website.
- **Trợ giúp trực tuyến (Online help)**
- *Online help* được lồng vào với tài liệu của người sử dụng, đôi khi *Online help* còn được thay thế tài liệu người sử dụng.
- *Online help* giúp cho việc tìm kiếm trở nên dễ dàng hơn
- *Online help* giúp cho người sử dụng đặt ra những câu hỏi giống như một ngôn ngữ tự nhiên.
- Ví dụ: *Tell me how to copy text from one program to another!* (Hãy nói với tôi làm thế nào để sao chép một đoạn văn bản giữa hai chương trình!)
- **Các tài liệu hướng dẫn (Tutorials, wizards, and CBT -Computer Based Training)**
- Người sử dụng đặt ra câu hỏi sau đó phần mềm sẽ hướng dẫn người sử dụng thông qua từng bước để hoàn thành nhiệm vụ.
- Một trong những vấn đề đó chính là trợ lý của Microsoft's Office "paper clip guy"



*Trợ lý của Microsoft Office là một ví dụ rất tinh tế về sự trợ giúp của hệ thống*

- **Samples, examples và templates**
- Đây là một loại văn bản với những mẫu mà người sử dụng có thể điền vào và cho một kết quả nhanh chóng
- Người biên tập sẽ tổng hợp những văn bản đó và trình bày theo một ngôn ngữ nhất định
- **Thông báo lỗi (Error messages)**

- Vấn đề về thông báo lỗi đã được đề cập đến trong cuốn sách này một vài lần nhưng sau đó nó lại bị lãng quên
- Cuối cùng những vấn đề này chỉ nằm trong tài liệu

## **Tầm quan trọng của việc kiểm thử tài liệu**

Người sử dụng họ không quan tâm tới phần mềm được tạo ra như thế nào bởi ai. Cái mà người sử dụng quan tâm chính là chất lượng của sản phẩm sau khi đã được đóng gói.

- *Chú ý:*
- *Nếu phần hướng dẫn cài đặt bị sai hoặc nếu những thông báo lỗi không đúng khi đó sẽ làm cho người sử dụng sang hiểu sai vấn đề.*
- Người sử dụng sẽ gửi những lỗi đó cho nhà cung cấp sản phẩm, khi đó một vài lỗi phần mềm đó sẽ được tìm thấy bởi người kiểm thử phần mềm.

Một tài liệu phần mềm tốt nó sẽ làm cho chất lượng của toàn bộ phần mềm được tốt hơn. Điều đó được thể hiện ở ba cách sau:

- *Tận dụng được tính khả dụng (It improves usability)*
- *Trong phần trước chúng ta đã được học “Tính khả dụng” -Usability Testing*
- *Tất cả những vấn đề liên quan đến tính khả dụng là gì?*
- *Liệu những tính khả dụng kia có liên quan đến tài liệu phần mềm không?*
- *Cải thiện sự tin cậy (It improves reliability)*
- *Sự tin cậy và ổn định của phần mềm là bao nhiêu ?*
- *Nó làm cho người sử dụng hy vọng hay không khi họ đón nhận nó.*
- *Nếu người sử dụng đọc tài liệu, sử dụng phần mềm và không tin cậy vào phần mềm đó.*
- Trong phần này chúng ta sẽ thấy rằng kiểm thử phần mềm và tài liệu là một trong những cách tốt nhất để loại bỏ những lỗi đó.
- *Nó giúp cho việc giảm chi phí (It lowers support costs)*
- Trong phần trước bạn đã được học về những vấn đề tìm thấy bởi những khách hàng
- Chi phí bỏ ra ít hay nhiều tùy thuộc vào quá trình phát hiện sớm hay muộn của khách hàng
- Nếu khách hàng dùng sản phẩm và phát hiện ra lỗi sớm thì chi phí bỏ ra sửa lại sẽ không cao
- Nếu khách hàng dùng sản phẩm và phát hiện ra lỗi muộn thì chi phí bỏ ra sửa lại là rất đắt.
- *Chú ý:*
- Là một Tester bạn nên quan tâm tới tài liệu phần mềm giống như Coder đang nỗ lực hoàn thành sản phẩm.

- Nếu bạn không coi trọng tài liệu thì toàn bộ phần mềm mà bạn kiểm thử sẽ không đạt kết quả cao.

## Bạn tìm thấy gì khi xem xét tài liệu

Kiểm thử tài liệu có thể xét ở hai trường hợp:

- Nếu tài liệu không phải là mã, thì kiểm thử thường là một quá trình tĩnh và được mô tả trong bài 3
- Nếu tài liệu và mã có ràng buộc gần với nhau, kiểm thử trở thành quá trình động và được mô tả trong bài 3
- *Chú ý:*
- Nếu có đoạn mã đơn giản, bạn hãy gõ nó vào máy và thử như mô tả, với cách tiếp cận thực tế đơn giản này, bạn sẽ tìm ra những lỗi trong cả phần mềm và tài liệu.
- *Dù cho tài liệu có mã hay không, thì hãy đọc tài liệu cẩn thận, thực hiện từng bước, khảo sát tất cả các hình, thử mọi ví dụ.*

Danh sách các trường hợp kiểm thử (là cơ sở để xây dựng các trường hợp kiểm thử tài liệu):

Danh mục	Công việc thực hiện
Thuật ngữ	<ul style="list-style-type: none"> <li>• Thuật ngữ có thích hợp với người sử dụng không?</li> <li>• Thuật ngữ có được sử dụng thường xuyên và chính xác không?</li> <li>• Một thuật ngữ được viết tắt có phải là thuật ngữ chuẩn hay không?</li> <li>• Nếu viết tắt tên công ty thì điều gì sẽ xảy ra?</li> </ul>
Nội dung và đề tài	<ul style="list-style-type: none"> <li>• Liệu đề tài bất kỳ nào cũng có thể sinh lỗi?</li> <li>• Những loại đề tài nào không thể khái quát nếu một đặc tính được lấy ra từ sản phẩm và người viết tài liệu không hề biết?</li> </ul>
Thực tế	<ul style="list-style-type: none"> <li>• Liệu mọi thông tin về mặt thực tế và kỹ thuật đều đúng?</li> <li>• Hãy kiểm tra nội dung, chỉ số, và các chương khác nhau của tài liệu.</li> <li>• Hãy thử với các website URL. Liệu các sản phẩm này có hỗ trợ số phone đúng hay không?</li> </ul>
Thực hiện từng bước	<ul style="list-style-type: none"> <li>• Đọc tất cả văn bản cẩn thận và chậm rãi. Theo sau là những chỉ dẫn chính xác. Không đặt ra giả thiết gì! Và sửa những lỗi sai.</li> </ul>

Bắt hình và màn ảnh	<ul style="list-style-type: none"> <li>Kiểm tra các hình với độ chính xác cao. Bạn không được sao chép hình từ phần mềm đã có sự thay đổi. Và kiểm tra cùng với tên của hình đó.</li> </ul>
Mẫu và ví dụ	<ul style="list-style-type: none"> <li>Nhập vào và sử dụng từng mẫu như một khách hàng. Nếu nó là mã, có thể sao chép hoặc gõ vào máy và chạy nó.</li> </ul>
Đánh vần và ngữ pháp	<ul style="list-style-type: none"> <li>Đây không phải là một lỗi gây rắc rối lớn. Bạn đừng quên kiểm tra hoặc bỏ qua những thuật ngữ hay kỹ thuật chuyên ngành. Đó cũng có thể là những kỹ thuật kiểm tra bằng tay, như trong việc sao chép hình và vẽ hình.</li> </ul>

Cuối cùng, nếu bạn phải xét nhiều trường hợp kiểm thử, hãy sử dụng kỹ thuật phân lớp tương đương để chọn ra các trường hợp điển hình để kiểm thử.

## Thực tế của việc kiểm thử tài liệu

Một Tester sẽ gặp nhiều khó khăn trong quá trình kiểm thử bởi đôi khi người viết tài liệu cho phần mềm (writer) không phải là chuyên gia. Cách tốt nhất là nên làm việc gần gũi với writer để họ hiểu những khó khăn mà bạn gặp phải và giải thích rõ ràng hơn trong tài liệu.

Một tài liệu của sản phẩm phần mềm cần phải được cố định trước khi phần mềm hoàn thành. Nếu một chức năng nào đó của phần mềm bị thay đổi hoặc gây lỗi, tài liệu sẽ không kịp thời thay đổi theo. Đó là lý do tệp *Readme* ra đời.

Tệp *Readme* phản ánh những thay đổi cuối cùng của phần mềm cho người sử dụng biết mà chúng không được thay đổi trong tài liệu.

## Tổng kết

Để việc kiểm thử tài liệu có kết quả, hãy tiến hành trên các tệp đi kèm với phần mềm: *Readme*, hướng dẫn cài đặt...; các tài liệu quảng cáo; các bản đăng ký, thậm chí là các tiêu chuẩn của phần mềm; cùng với các hình, mẫu, và các ví dụ...

Là một Tester, nếu bạn kiểm thử các tài liệu đúng cách, bạn sẽ tìm thấy những lỗi trước khi khách hàng của bạn phát hiện ra chúng.

# Kiểm thử khả năng bảo mật phần mềm

## WarGames the Movie

War Game được ra mắt công chúng vào năm 1983, David Lightman, người đã sử dụng máy tính IMSAI 8080 300 đơn vị âm thanh của modem để tấn công vào hệ thống máy tính NORAD của chính phủ. Làm thế nào để truy cập được? Ông ta lập trình trên máy tính của mình để quay một chuỗi số liên tục từ 5550000 đến 5559999, và lắng nghe những modem máy tính khác để trả lời.

Hơn 20 năm sau, công nghệ đã thay đổi, nhưng các quy trình và kỹ thuật vẫn giữ nguyên. Phần mềm và hệ thống máy tính hiện nay đã có nhiều mối quan hệ nhưng lại có rất nhiều chi tiết tặc. War Game đã đưa ra một ý nghĩa mới thông qua những hacker đang chơi trò chơi và các ngành công nghiệp phần mềm là một cuộc chiến tranh để chống lại họ. Việc ra đời phần mềm bảo mật là một yếu tố quan trọng.

## [WAR DRIVING](#)

Với sự phát triển của “WiFi”, những hacker đã phát hiện ra rằng họ có thể tìm kiếm để mở nhiều mạng máy tính giống như nhân vật của David Lightman trong WarGame. Các kỹ thuật sau khi có tên trong war game được biết đến như là “war driving”. War Driving là kỹ thuật sử dụng modem để tự động quét một danh sách số điện thoại. Hacker đã sử dụng danh sách kết quả tìm được cho nhiều mục đích khác nhau và cho việc đoán mật khẩu

## Tìm hiểu về Motivation

Trong war Game, các hacker có được động lực là do sự tò mò và mong muốn sử dụng máy tính với mục đích riêng của mình. Hiểu được mục đích đó sẽ giúp bạn trong suy nghĩ về nơi mà các lỗ hổng bảo mật có thể có trong phần mềm mà bạn thử nghiệm.

Một sản phẩm an toàn là một sản phẩm được bảo vệ bởi:

- Confidentiality (giữ bí mật)
- Integrity (tình trạng còn nguyên vẹn)
- Availability (tính sẵn sàng)
- Tính khả dụng của thông tin khách hàng
- Sự tích hợp và chế biến sẵn có của các nguồn tài nguyên

*Hacker:* Là người có thông tin từ một chương trình hoặc bằng cách sử dụng máy vi tính. Là người sử dụng các kỹ thuật lập trình để đạt được sự truy cập bất hợp pháp vào mạng lưới máy tính hay tập tin

*5 lý do hacker có thể có để đạt được quyền truy cập:*

- Challenge/Prestige (Thách thức / Uy tín)
- Curiosity (sự tò mò)
- Use/Leverage
- War Game là một sự tấn công Use/Leverage
- Vandalize (Phá hoại)
- Vandalize (Phá hoại) quan tâm đến 3 yếu tố :
  - Defacing (Làm mất vẻ đẹp)
  - Destruction(sự phá hủy)
  - Denial of Service (từ chối của dịch vụ)

**Steal:** nặng nhất của hình thức hacking là trộm cắp ngay, mục đích là một số thứ có giá trị có thể sử dụng được hoặc để bán, ví dụ: số thẻ tín dụng, thông tin cá nhân, tài sản, dịch vụ, thậm chí là cả quyền đăng nhập.

## **Threat Modeling**

Chúng mô tả quá trình về một đe dọa mẫu cho việc đánh giá một hệ thống phần mềm về các vấn đề bảo mật. Nó được xem xét để thiết lập lỗ hổng bảo mật để giảm rủi ro. Với thông tin đó nhóm có thể được chọn để thay đổi sản phẩm, dành nhiều hơn cho việc nỗ lực thiết kế các tính năng, hoặc tập trung thử nghiệm trên những rắc rối tiềm năng. Cuối cùng với sự hiểu biết như vậy sẽ là kết quả trong một sản phẩm an toàn hơn. Thực hiện threat modeling (mối đe dọa mẫu) không phải là trách nhiệm của tester phần mềm. Trách nhiệm này nên thuộc về nhiệm vụ của người quản lý dự án và tất cả thành viên của nhóm dự án.

- **Assemble the threat modeling team** (Hình thành nhóm về mối đe dọa mẫu): Ngoài các tiêu chuẩn về thành viên trong nhóm, phải có 1 thành viên có nền tảng sâu rộng về các phần mềm an ninh. Đối với nhóm nhỏ thì hướng dẫn trong việc thiết kế giai đoạn. Trong một công ty lớn thì người này là một chuyên gia về vấn đề đó, có thể đến từ dự án khác. Đề xuất kiến thức và chuyên môn, đây là điều quan trọng cho đội ngũ nhân viên để hiểu rằng mục đích ban đầu của họ không phải là giải quyết vấn đề an ninh mà là nhận dạng
- **Identify the Assets** (Xác định tài sản): Xác định xem hệ thống có thông tin gì cho một kẻ đột nhập ?. Có thông tin cá nhân của khách hàng ? số thẻ tín dụng? Làm thế nào về các nguồn tài nguyên máy tính?
- **Create an Architecture Overview** (Tạo một kiến trúc tổng quát): Trong giai đoạn này sẽ xác định công nghệ kỹ thuật, kế hoạch để sử dụng phần mềm và mối quan hệ của chúng như thế nào. Nhóm sẽ tạo một sơ đồ kiến trúc cho thấy công nghệ và việc chúng giao tiếp với nhau như thế nào.
- **Decompose the Application** (Phân tích ứng dụng): Đây là quá trình xác định dữ liệu chảy tràn qua hệ thống như thế nào và ở đâu.



- **Identify the Threats** (Nhận biết Threats): Sau khi đã hiểu kỹ các thành phần(tài sản,kiến trúc , dữ liệu), nhóm sẽ di chuyển đến việc xác định mối đe dọa, mỗi phần nên được quan tâm tới mục tiêu của mỗi đe dọa, nên giả định rằng chúng sẽ bị tấn công.
- **Rank the threats** (Đánh giá về các mối đe dọa): Cuối cùng để hiểu rằng tất cả các mối đe dọa không phải được tạo ra như nhau

Dữ liệu trong hệ thống bị đe dọa ngay cả với mức độ bảo mật cao,nhóm bạn cần nhìn vào mỗi mối đe dọa và xác định về việc xếp hạng các mối đe dọa đó.

- **Damage Potential** (Khả năng hỏng): việc hư hỏng sẽ là bao nhiêu nếu vùng này bị hacked.
- **Reproducibility** (Sự sản sinh): nó là dễ dàng như thế nào để một hacker khai thác được vùng dễ bị tấn công một cách hợp lý, mọi sự cố gắng sẽ thành công, một trong một trăm sự cố gắng, một trong một triệu.
- **Affected Users** (sự ảnh hưởng của người sử dụng): nếu một hacker thành công có bao người sử dụng sẽ bị ảnh hưởng
- **Discoverability** (có thể phát hiện): khả năng gì mà hacker sẽ tìm ra vùng dễ bị tấn công, một sự đăng nhập “backdoor” mà không gây ra âm thanh để dễ phát hiện

### **Tính năng bảo mật của phần mềm có phải là đặc trưng (feature)? Lỗ hổng bảo mật có phải là một lỗi?**

Có lẽ hầu hết mọi người đều không muốn dữ liệu tài chính cá nhân của họ bị xâm nhập bởi một hacker nào đó. Họ luôn cho rằng phần mềm bảng biểu của họ có khả năng giữ thông tin riêng tư (như) một đặc tính cần thiết. Nếu phần mềm bị lỗi và cho phép hacker nhìn thấy những thông tin cá nhân và thông tin thẻ tín dụng, hầu hết người dùng sẽ xem đó là lỗi phần mềm.

Tổng kết định nghĩa của 1 lỗi từ “Kiểm thử bối cảnh”:

1. Phần mềm không làm những gì như trong bản đặc tả yêu cầu.
2. Phần mềm làm những gì như trong bản đặc tả yêu cầu.
3. Phần mềm làm những gì mà bản đặc tả không nói đến.
4. Phần mềm không làm những gì như bản đặc tả không nói đến nhưng nên làm.
5. Phần mềm khó hiểu, khó sử dụng.

Khi bạn kiểm thử “test to pass”, bạn đảm bảo đã hoàn thành tối thiểu các yêu cầu công việc đặt ra. Bạn không loại bỏ khả năng bạn không nhìn thấy những gì bạn có thể làm sai lệch. Với những ứng dụng đơn giản nhất và thường thì các “test case” không mấy phức tạp. Sau khi bảo đảm rằng phần mềm không có gì xảy ra trong trường hợp bình thường, đây là lúc để đặt trên giả định của bạn, làm ngơ trước các lỗi, xoay vòng và cố gắng tìm

thấy lỗi do sự cố nên ép chúng “outtest-to-fail”. Một Tester, bạn phải chịu trách nhiệm kiểm tra về sự an toàn của phần hoặc chịu trách nhiệm kiểm tra một vài tính năng được giao. Bạn không nhất thiết phải đề ra cho một sản phẩm đặc tả rõ ràng: định nghĩa như thế nào là phần mềm an toàn được đề cập trong phần mềm của bạn. Bạn sẽ cần phải đặt “test- to –fail” đối với các trường hợp phần mềm bị một hacker tấn công, và tất cả các tính năng bảo mật có khả năng bị tổn thương.

**KẾT LUẬN:** Kiểm thử tính năng bảo mật là kế hoạch kiểm tra trường hợp có thể không hoạt động và sẽ thường bao gồm các phần của các sản phẩm mà chưa được hiểu hoàn toàn hay chỉ định.

### **Tìm hiểu về tràn bộ đệm (Buffer Overrun)**

- Một bảng tính chia sẻ trên trang chủ của bạn trên mạng không đây là rất khác nhau, từ một hay nhiều trò chơi video phát trên các trang web hoặc một phân phối trên hệ thống máy tính của Bộ Truyền Thông.
- Các hệ điều hành và các công nghệ quan trọng đang là duy nhất và có sẽ nguy cơ gặp rủi ro an ninh khác nhau.
- Một trong những vấn đề thường gặp, đó là một vấn đề bảo mật do bị tràn bộ đệm trong bất kỳ sản phẩm phần mềm nào.
- Bạn tìm hiểu về các dữ liệu tham khảo Errorsbugs gây ra bằng cách sử dụng một biến cố định, mảng, chuỗi, hoặc tài liệu mà không được đúng tuyên bố hoặc khởi động như thế nào cho điều đó đang được sử dụng và tham chiếu. Tràn bộ đệm là một lỗi.
- Nó là kết quả của ngôn ngữ nghèo nàn, như ngôn ngữ như C và C ++, các hàm xử lý chuỗi thiếu an toàn.

Danh mục 13-1. *Ví dụ đơn giản về tràn bộ đệm.*

```
1: void myBufferCopy(char * pSourceStr) {  
2: char pDestStr[100];  
3: int nLocalVar1 = 123;  
4: int nLocalVar2 = 456;  
5: strcpy(pDestStr, pSourceStr);...  
6: }  
7: void myValidate()  
8: {
```

9: /\*

10: Assume this function's code validates a user password

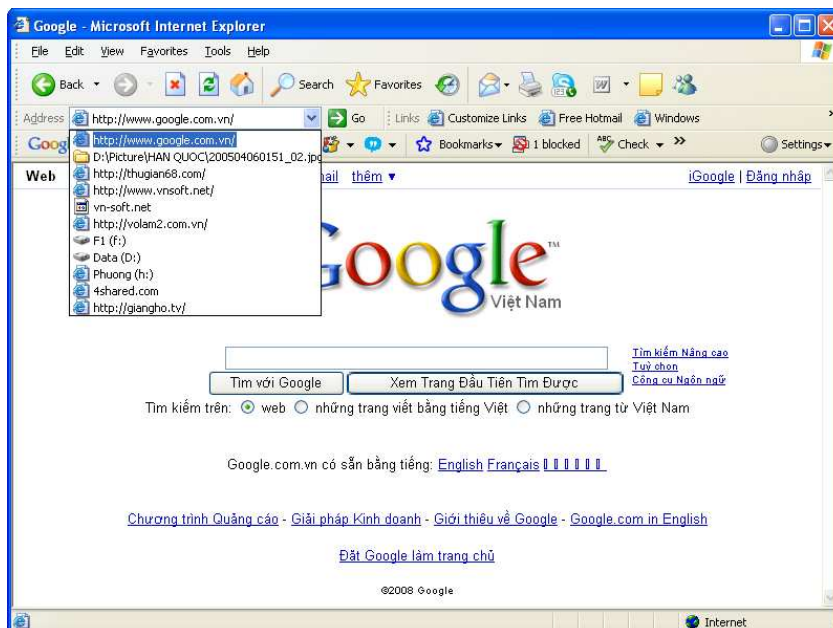
11: and grants access to millions of private customer records

12: /\*

- Nếu lớn hơn 100 bytes, nó sẽ điền vào chuỗi các điểm đến và sau đó tiếp tục Overwriting các giá trị được lưu trữ trong các biến địa phương.
- Tồi tệ hơn, nếu là nguồn chuỗi dài là đủ, nó cũng có thể trở lại đề lên các địa chỉ của chức năng myBufferCopy và các nội dung của mã thi hành trong chức năng myValidate.
- Trong ví dụ này, một cơ hacker có thể nhập một mật khẩu siêu dài, thay vì lắp ráp bằng văn bản của alphanumeric ký tự ASCII, và ghi đè lên nội dung mật khẩu xác nhận thực hiện trong myValidatepossibly được truy cập vào hệ thống.

## Computer forensics

Một hacker tấn công phần mềm của bạn bằng cách tìm những điểm yếu bảo mật dễ bị tấn công của phần mềm và khai thác chúng để truy cập vào dữ liệu hoặc để kiểm soát hệ thống. Ví dụ đầu tiên của vấn đề này là WebBrowser:



Lỗ hổng bảo mật trên webbrowser

**AutoFill Settings**

The Google Toolbar can automatically fill out web forms with the information below. This information will be stored on your computer and will not be sent to Google.

You can fill in as many (or as few) options as you like.

**Name**

Full name: Nguyễn Thị Phương

Email address: phuongnt23@gmail.com

Phone number: 097-422-9765

**Primary Address**

Line 1: 09090909

Line 2: 0909097766

City: Hải Dương

State/Province: 084

Postal code: 084

Country/Region: Vietnam

When a page asks for a shipping address, use

☒ my primary address

☐ an alternate address

Add/Edit Alternate Address...

**Credit Card (optional)**

Securely store credit card information

Add/Edit Credit Card...

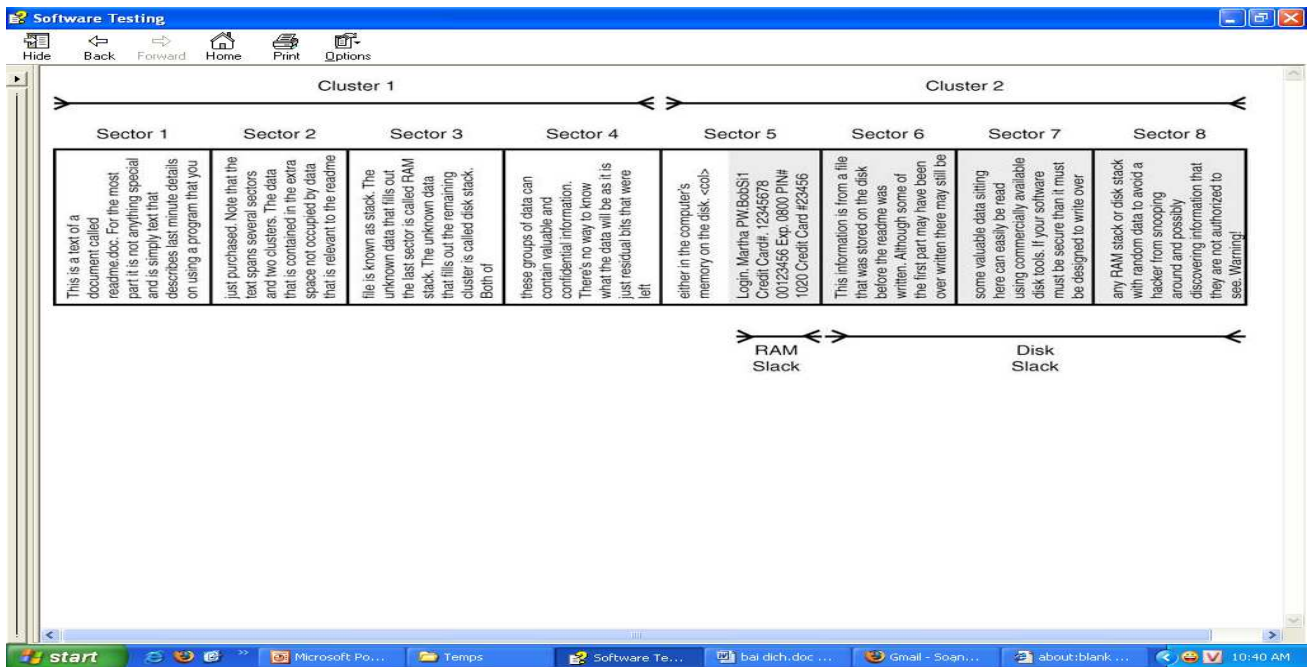
**Highlight**

☒ Highlight fields on Web pages that AutoFill can update in yellow

OK Cancel

Một ví dụ khác của dữ liệu tiềm năng là Thanh công cụ của Google có tính năng Autofill

Khi dữ liệu được ghi vào một đĩa, nó được ghi trong khối. Kích thước của các khối, gọi là các sector, các sector này khác nhau tùy thuộc vào hệ điều hành đang sử dụng. MSDOS / Windows sử dụng khối 512 byte. Tùy thuộc vào hệ thống tập tin được sử dụng, các sector được viết ra trong các nhóm gọi là clusters Windows FAT tập tin sử dụng các cluster của 2048 byte, gồm bốn cluster 512 byte. Tập có chiều dài 2.200 byte và được hiển thị bởi vùng trắng trải rộng từ Sector 1 đến giữa của sector 5



### clusters Windows FAT

Nếu file là có độ dài 2.200 byte, nó sẽ mất 4,3 khối mà mỗi khối là 512 byte ( $2200/512 = 4.3$ ). Dữ liệu ở phần cuối của sector 5 được gọi là RAM “slack” vì những gì mà được chứa ở đó là thông tin xảy ra khi truy cập vào RAM của hệ thống khi file được tạo. Nó có thể là không có gì, hoặc nó có thể được quản lý mật khẩu hay số thẻ tín dụng.

Không có cách nào để biết, nhưng những dữ liệu đó khác với những dữ liệu được ghi ở các file trong ổ đĩa máy tính của bạn. Mặc dù ví dụ này sử dụng một ổ đĩa để minh họa các khái niệm về dữ liệu tiềm tàng, các vấn đề bảo mật của RAM và đĩa slack cũng áp dụng để ghi đĩa CD, DVD, thẻ nhớ, và hầu như bất kỳ loại phương tiện lưu trữ.

Khi bạn đang kiểm thử một sản phẩm phần mềm, bạn sẽ cần phải làm việc với nhóm của bạn để quyết định nếu dữ liệu tiềm năng là một lỗ hổng bảo mật. Bạn và nhóm của bạn sẽ cần phải nghĩ ra cách để ngăn cản điều đó xảy ra.

### Tổng kết

Không hệ thống máy tính nào là an toàn tuyệt đối. Trong việc thiết kế một hệ thống an toàn và bảo mật, bạn phải chú ý quan tâm đến vấn đề bảo mật ngay từ khi bắt đầu của quá trình thiết kế sản phẩm. Bạn có thể không chỉ "test in" bảo mật phần mềm. Việc này phải được quy hoạch, xem xét, thiết kế, và sau đó thử nghiệm.

Áp dụng cho một quá trình lặp của phát triển phần mềm như Spiral Model chi tiết trong Chương 2, "Quy trình phát triển phần mềm," sẽ bảo đảm rằng những chủ đề này sẽ được

thăm lại trong suốt quá trình phát triển. Một trang web có rất nhiều thông tin sẽ giúp cho bạn cập nhật hằng ngày trên máy tính các vấn đề về an ninh là [www.securityfocus.com](http://www.securityfocus.com).

# Bài 6 : Các giai đoạn kiểm thử

## Các giai đoạn kiểm thử

### Unit Test – Kiểm tra mức đơn vị

Để có thể hiểu rõ về Unit Test, khái niệm trước tiên ta cần làm rõ: thế nào là một đơn vị PM (Unit)?

Một Unit là một thành phần PM nhỏ nhất mà ta có thể kiểm tra được. Theo định nghĩa này, các hàm (Function), thủ tục (Procedure), lớp (Class), hoặc các phương thức (Method) đều có thể được xem là Unit.

Vì Unit được chọn để kiểm tra thường có kích thước nhỏ và chức năng hoạt động đơn giản, chúng ta không khó khăn gì trong việc tổ chức, kiểm tra, ghi nhận và phân tích kết quả kiểm tra. Nếu phát hiện lỗi, việc xác định nguyên nhân và khắc phục cũng tương đối dễ dàng vì chỉ khoanh vùng trong một đơn thể Unit đang kiểm tra. Một nguyên lý đúc kết từ thực tiễn: thời gian tốn cho Unit Test sẽ được đền bù bằng việc tiết kiệm rất nhiều thời gian và chi phí cho việc kiểm tra và sửa lỗi ở các mức kiểm tra sau đó.

Unit Test thường do lập trình viên thực hiện. Công đoạn này cần được thực hiện càng sớm càng tốt trong giai đoạn viết code và xuyên suốt chu kỳ PTPM. Thông thường, Unit Test đòi hỏi kiểm tra viên có kiến thức về thiết kế và code của chương trình. Mục đích của Unit Test là bảo đảm thông tin được xử lý và xuất (khỏi Unit) là chính xác, trong mối tương quan với dữ liệu nhập và chức năng của Unit. Điều này thường đòi hỏi tất cả các nhánh bên trong Unit đều phải được kiểm tra để phát hiện nhánh phát sinh lỗi. Một nhánh thường là một chuỗi các lệnh được thực thi trong một Unit, ví dụ: chuỗi các lệnh sau điều kiện If và nằm giữa then ... else là một nhánh. Thực tế việc chọn lựa các nhánh để đơn giản hóa việc kiểm tra và quét hết Unit đòi hỏi phải có kỹ thuật, đôi khi phải dùng thuật toán để chọn lựa.

Cũng như các mức kiểm tra khác, Unit Test cũng đòi hỏi phải chuẩn bị trước các tình huống (test case) hoặc kịch bản (script), trong đó chỉ định rõ dữ liệu vào, các bước thực hiện và dữ liệu mong chờ sẽ xuất ra. Các test case và script này nên được giữ lại để tái sử dụng.

### Integration Test – Kiểm tra tích hợp

Integration test kết hợp các thành phần của một ứng dụng và kiểm tra như một ứng dụng đã hoàn thành. Trong khi Unit Test kiểm tra các thành phần và Unit riêng lẻ thì Integration Test kết hợp chúng lại với nhau và kiểm tra sự giao tiếp giữa chúng.

Integration Test có 2 mục tiêu chính:

- Phát hiện lỗi giao tiếp xảy ra giữa các Unit.
- Tích hợp các Unit đơn lẻ thành các hệ thống nhỏ (subsystem) và cuối cùng là nguyên hệ thống hoàn chỉnh (system) chuẩn bị cho kiểm tra ở mức hệ thống (System Test).

Trong Unit Test, lập trình viên cố gắng phát hiện lỗi liên quan đến chức năng và cấu trúc nội tại của Unit. Có một số phép kiểm tra đơn giản trên giao tiếp giữa Unit với các thành phần liên quan khác, tuy nhiên mọi giao tiếp liên quan đến Unit thật sự được kiểm tra đầy đủ khi các Unit tích hợp với nhau trong khi thực hiện Integration Test.

Trừ một số ít ngoại lệ, Integration Test chỉ nên thực hiện trên những Unit đã được kiểm tra cẩn thận trước đó bằng Unit Test, và tất cả các lỗi mức Unit đã được sửa chữa. Một số người hiểu sai rằng Unit một khi đã qua giai đoạn Unit Test với các giao tiếp giả lập thì không cần phải thực hiện Integration Test nữa. Thực tế việc tích hợp giữa các Unit dẫn đến những tình huống hoàn toàn khác.

Một chiến lược cần quan tâm trong Integration Test là nên tích hợp dần từng Unit. Một Unit tại một thời điểm được tích hợp vào một nhóm các Unit khác đã tích hợp trước đó và đã hoàn tất (passed) các đợt Integration Test trước đó. Lúc này, ta chỉ cần kiểm tra giao tiếp của Unit mới thêm vào với hệ thống các Unit đã tích hợp trước đó, điều này làm cho số lượng kiểm tra sẽ giảm đi rất nhiều, sai sót sẽ giảm đáng kể.

Có 4 loại kiểm tra trong Integration Test:

- Kiểm tra cấu trúc (structure): Tương tự White Box Test (kiểm tra nhằm bảo đảm các thành phần bên trong của một chương trình chạy đúng), chú trọng đến hoạt động của các thành phần cấu trúc nội tại của chương trình chẳng hạn các lệnh và nhánh bên trong.
- Kiểm tra chức năng (functional): Tương tự Black Box Test (kiểm tra chỉ chú trọng đến chức năng của chương trình, không quan tâm đến cấu trúc bên trong), chỉ khảo sát chức năng của chương trình theo yêu cầu kỹ thuật.
- Kiểm tra hiệu năng (performance): Kiểm tra việc vận hành của hệ thống.
- Kiểm tra khả năng chịu tải (stress): Kiểm tra các giới hạn của hệ thống.

## **System Test - Kiểm tra mức hệ thống**

Mục đích System Test là kiểm tra thiết kế và toàn bộ hệ thống (sau khi tích hợp) có thỏa mãn yêu cầu đặt ra hay không.

System Test bắt đầu khi tất cả các bộ phận của PM đã được tích hợp thành công. Thông thường loại kiểm tra này tốn rất nhiều công sức và thời gian. Trong nhiều trường hợp,



việc kiểm tra đòi hỏi một số thiết bị phụ trợ, phần mềm hoặc phần cứng đặc thù, đặc biệt là các ứng dụng thời gian thực, hệ thống phân bố, hoặc hệ thống nhúng. Ở mức độ hệ thống, người kiểm tra cũng tìm kiếm các lỗi, nhưng trọng tâm là đánh giá về hoạt động, thao tác, sự tin cậy và các yêu cầu khác liên quan đến chất lượng của toàn hệ thống.

Điểm khác nhau then chốt giữa Integration Test và System Test là System Test chú trọng các hành vi và lỗi trên toàn hệ thống, còn Integration Test chú trọng sự giao tiếp giữa các đơn thể hoặc đối tượng khi chúng làm việc cùng nhau. Thông thường ta phải thực hiện Unit Test và Integration Test để bảo đảm mọi Unit và sự tương tác giữa chúng hoạt động chính xác trước khi thực hiện System Test.

Sau khi hoàn thành Integration Test, một hệ thống PM đã được hình thành cùng với các thành phần đã được kiểm tra đầy đủ. Tại thời điểm này, lập trình viên hoặc kiểm tra viên (tester) bắt đầu kiểm tra PM như một hệ thống hoàn chỉnh. Việc lập kế hoạch cho System Test nên bắt đầu từ giai đoạn hình thành và phân tích các yêu cầu. Phần sau ta sẽ nói rõ hơn về một quy trình System Test cơ bản và điển hình.

System Test kiểm tra cả các hành vi chức năng của phần mềm lẫn các yêu cầu về chất lượng như độ tin cậy, tính tiện lợi khi sử dụng, hiệu năng và bảo mật. Mức kiểm tra này đặc biệt thích hợp cho việc phát hiện lỗi giao tiếp với PM hoặc phần cứng bên ngoài, chẳng hạn các lỗi "tắc nghẽn" (deadlock) hoặc chiếm dụng bộ nhớ. Sau giai đoạn System Test, PM thường đã sẵn sàng cho khách hàng hoặc người dùng cuối cùng kiểm tra để chấp nhận (Acceptance Test) hoặc dùng thử (Alpha/Beta Test).

Đòi hỏi nhiều công sức, thời gian và tính chính xác, khách quan, System Test thường được thực hiện bởi một nhóm kiểm tra viên hoàn toàn độc lập với nhóm phát triển dự án.

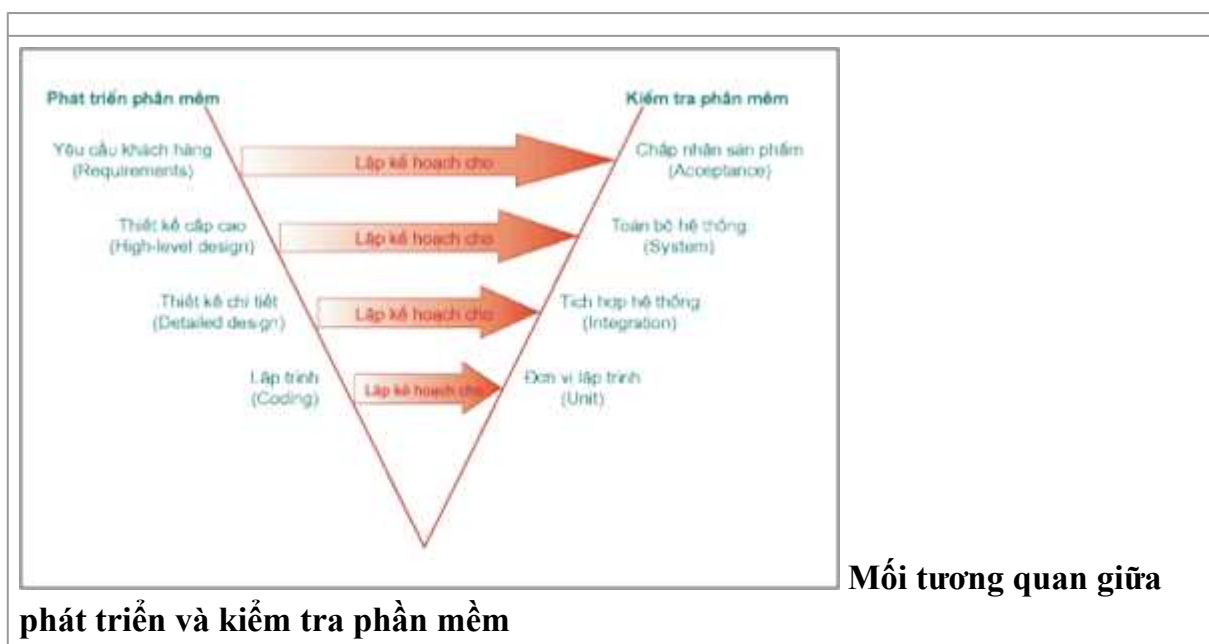
Bản thân System Test lại gồm nhiều loại kiểm tra khác nhau (xem hình 6.2), phổ biến nhất gồm:

- Kiểm tra chức năng (Functional Test): bảo đảm các hành vi của hệ thống thỏa mãn đúng yêu cầu thiết kế.
- Kiểm tra khả năng vận hành (Performance Test): bảo đảm tối ưu việc phân bổ tài nguyên hệ thống (ví dụ bộ nhớ) nhằm đạt các chỉ tiêu như thời gian xử lý hay đáp ứng câu truy vấn...
- Kiểm tra khả năng chịu tải (Stress Test hay Load Test): bảo đảm hệ thống vận hành đúng dưới áp lực cao (ví dụ nhiều người truy xuất cùng lúc). Stress Test tập trung vào các trạng thái tới hạn, các "điểm chết", các tình huống bất thường...
- Kiểm tra cấu hình (Configuration Test)
- Kiểm tra khả năng bảo mật (Security Test): bảo đảm tính toàn vẹn, bảo mật của dữ liệu và của hệ thống.

- Kiểm tra khả năng phục hồi (Recovery Test): bảo đảm hệ thống có khả năng khôi phục trạng thái ổn định trước đó trong tình huống mất tài nguyên hoặc dữ liệu; đặc biệt quan trọng đối với các hệ thống giao dịch như ngân hàng trực tuyến.

Nhìn từ quan điểm người dùng, các kiểm tra trên rất quan trọng: bảo đảm hệ thống đủ khả năng làm việc trong môi trường thực.

Lưu ý không nhất thiết phải thực hiện tất cả các loại kiểm tra nêu trên. Tùy yêu cầu và đặc trưng của từng hệ thống, tùy khả năng và thời gian cho phép của dự án, khi lập kế hoạch, trưởng dự án sẽ quyết định áp dụng những loại kiểm tra nào.



## Acceptance Test - Kiểm tra chấp nhận sản phẩm

Thông thường, sau giai đoạn System Test là Acceptance Test, được khách hàng thực hiện (hoặc ủy quyền cho một nhóm thứ ba thực hiện). Mục đích của Acceptance Test là để chứng minh PM thỏa mãn tất cả yêu cầu của khách hàng và khách hàng chấp nhận sản phẩm (và trả tiền thanh toán hợp đồng).

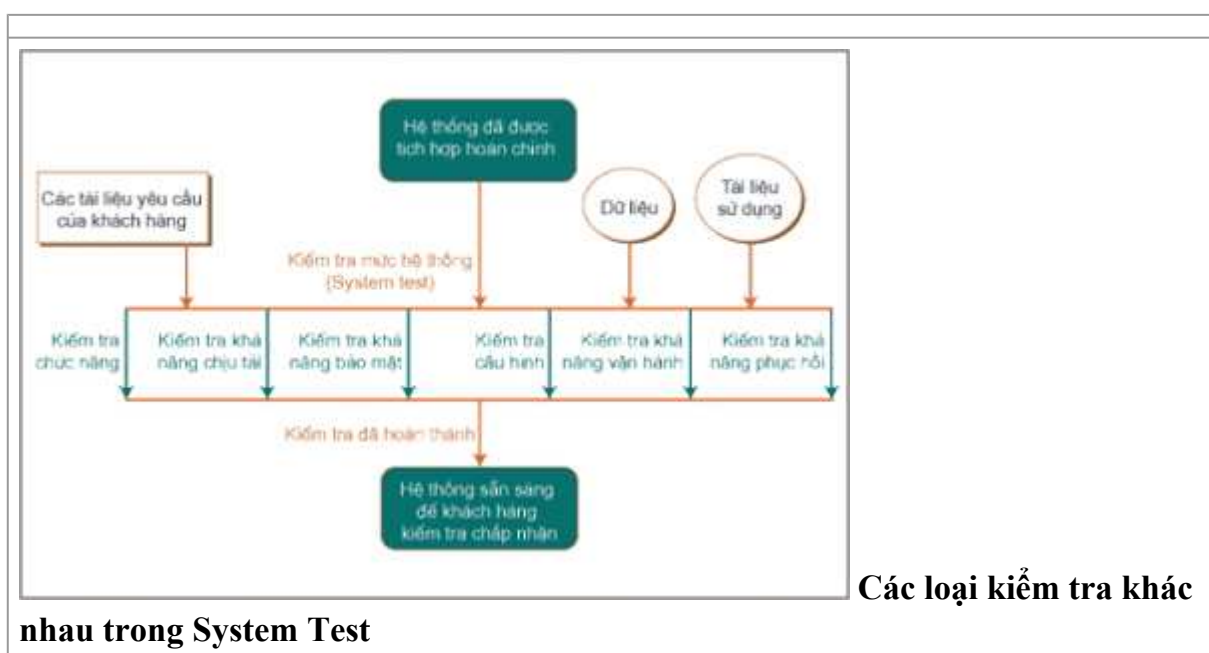
Acceptance Test có ý nghĩa hết sức quan trọng, mặc dù trong hầu hết mọi trường hợp, các phép kiểm tra của System Test và Acceptance Test gần như tương tự, nhưng bản chất và cách thức thực hiện lại rất khác biệt.

Đối với những sản phẩm dành bán rộng rãi trên thị trường cho nhiều người sử dụng, thông thường sẽ thông qua hai loại kiểm tra gọi là Alpha Test và Beta Test. Với Alpha Test, người sử dụng (tiềm năng) kiểm tra PM ngay tại nơi PTPM, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi, và lên kế hoạch sửa chữa. Với Beta Test, PM sẽ được gửi tới

cho người sử dụng (tiềm năng) để kiểm tra ngay trong môi trường thực, lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho lập trình viên để sửa chữa.

Thực tế cho thấy, nếu khách hàng không quan tâm và không tham gia vào quá trình PTPM thì kết quả Acceptance Test sẽ sai lệch rất lớn, mặc dù PM đã trải qua tất cả các kiểm tra trước đó. Sự sai lệch này liên quan đến việc hiểu sai yêu cầu cũng như sự mong chờ của khách hàng. Ví dụ đôi khi một PM xuất sắc vượt qua các phép kiểm tra về chức năng thực hiện bởi nhóm thực hiện dự án, nhưng khách hàng khi kiểm tra sau cùng vẫn thất vọng vì bố cục màn hình nghèo nàn, thao tác không tự nhiên, không theo tập quán sử dụng của khách hàng v.v...

Gắn liền với giai đoạn Acceptance Test thường là một nhóm những dịch vụ và tài liệu đi kèm, phổ biến như hướng dẫn cài đặt, sử dụng v.v... Tất cả tài liệu đi kèm phải được cập nhật và kiểm tra chặt chẽ.



## Regression Test - Kiểm tra hồi quy

Trước tiên cần khẳng định Regression Test không phải là một mức kiểm tra, như các mức khác đã nói ở trên. Nó đơn thuần kiểm tra lại PM sau khi có một sự thay đổi xảy ra, để bảo đảm phiên bản PM mới thực hiện tốt các chức năng như phiên bản cũ và sự thay đổi không gây ra lỗi mới trên những chức năng vốn đã làm việc tốt. Regression test có thể thực hiện tại mọi mức kiểm tra.

Ví dụ: một PM đang phát triển khi kiểm tra cho thấy nó chạy tốt các chức năng A, B và C. Khi có thay đổi code của chức năng C, nếu chỉ kiểm tra chức năng C thì chưa đủ, cần phải kiểm tra lại tất cả các chức năng khác liên quan đến chức năng C, trong ví dụ này

là A và B. Lý do là khi C thay đổi, nó có thể sẽ làm A và B không còn làm việc đúng nữa.

Mặc dù không là một mức kiểm tra, thực tế lại cho thấy Regression Test là một trong những loại kiểm tra tốn nhiều thời gian và công sức nhất. Tuy thế, việc bỏ qua Regression Test là "không được phép" vì có thể dẫn đến tình trạng phát sinh hoặc tái xuất hiện những lỗi nghiêm trọng, mặc dù ta "tưởng rằng" những lỗi đó hoặc không có hoặc đã được kiểm tra và sửa chữa rồi!

## TÓM TẮT

Trên đây là tổng quan về các mức và loại kiểm tra PM cơ bản. Thực tế nếu đi sâu vào từng mức và loại kiểm tra, còn có rất nhiều kiểm tra đặc thù khác nữa, mang tính chuyên biệt cho từng vấn đề hoặc từng loại ứng dụng. Tuy nhiên, những mức độ và loại kiểm tra nêu trên là cơ bản nhất và có thể áp dụng trong hầu hết các loại ứng dụng PM khác nhau.

Trong số báo tới chúng tôi sẽ giới thiệu những bước cơ bản của một quy trình KTPM, làm thế nào để đánh giá và cải tiến năng lực KTPM của một tổ chức thông qua mô hình TMM (Testing Maturity Model), một mô hình được các chuyên gia đánh giá khá tốt dành cho hoạt động KTPM.

# Bài 7 : Tổng quan quy trình kiểm tra hệ thống phần mềm

## Mô hình kiểm tra phần mềm

### Mô hình kiểm tra phần mềm TMM

MÔ HÌNH KIỂM TRA PHẦN MỀM TMM (TESTING MATURITY MODEL) Mặc dù không ít người trong cũng như ngoài ngành biết hoặc đã từng nghe về mô hình CMM/CMMi (Capability Maturity Model/Intergration) của SEI (Software Engineering Institute – Viện công nghệ phần mềm của Mỹ) dùng để đánh giá và nâng cao năng lực PTPM, song có lẽ ít người biết về TMM - mô hình được các chuyên gia đánh giá là khá tốt – được dùng để đánh giá và nâng cao năng lực KTPM của một tổ chức.

TMM thực ra không mới, phần lớn nội dung của mô hình này đã được phát triển từ năm 1996, tuy nhiên chúng không được chấp nhận rộng rãi. Một trong những lý do chính đó là tài liệu về TMM rất ít. Các bài báo, sách về nó thường được viết dưới dạng nặng về lý thuyết. Một lý do nữa là phần lớn các tổ chức đều “say mê” mô hình CMM/CMMi và nghĩ rằng quá đủ cho qui trình PTPM của mình.

Thực tế cho thấy không hoàn toàn như vậy.

KTPM là một bộ phận sống còn của quy trình PTPM, sự hỗ trợ quan trọng để đảm bảo chất lượng của PM. Nhiều tổ chức PM trong thực tế vẫn chưa nhận thấy tính non nớt yếu kém trong quy trình cũng như năng lực KTPM của họ. Các mô hình hàng đầu hiện nay như CMM/CMMi/ISO9000 thực tế vẫn không chú tâm đầy đủ vào các vấn đề của KTPM.

TMM được phát triển tại IIT (Illinois Institute of Technology – Viện công nghệ Illinois) vào giữa thập niên 90 trong bối cảnh hầu như chưa có quy trình PM nào đề cập một cách toàn diện vấn đề kiểm tra trong PTPM. Tương tự SW-CMM, nó có một cấu trúc cơ bản bao gồm 5 mức trưởng thành. Vì TMM là mô hình chuyên biệt cho lĩnh vực KTPM, các mức trưởng thành này trực tiếp mô tả các mục tiêu trưởng thành của một quy trình KTPM. Trong một tổ chức PM, TMM không mâu thuẫn mà có thể dùng độc lập hoặc phối hợp với CMM/CMMi.

Mục đích của TMM là hỗ trợ tổ chức PM đánh giá và cải tiến các quy trình và năng lực PM của mình, mục tiêu cuối cùng là giúp tổ chức có thể:

- Hoàn thành sản phẩm đúng hạn và trong phạm vi ngân sách đã định.

- Tạo ra sản phẩm phần mềm có chất lượng cao hơn.
- Xây dựng nền tảng cho việc cải tiến quy trình ở phạm vi rộng trong một tổ chức.

TMM bao gồm hai thành phần chính: 1. Tập hợp 5 mức độ trưởng thành, định nghĩa năng lực KTPM của một tổ chức. Mỗi mức độ bao gồm:

- Mục tiêu
  - Hoạt động để hiện thực các mục tiêu
  - Công việc và phân công trách nhiệm
2. Mô hình đánh giá năng lực KTPM của một tổ chức, bao gồm:
- Bảng câu hỏi đánh giá
  - Thủ tục tiến hành đánh giá
  - Hướng dẫn để chọn lựa và huấn luyện nhóm đánh giá.

Phần sau ta sẽ khảo sát rõ hơn về các mức độ trưởng thành của TMM

Cấu trúc của một mức trưởng thành Các mức trưởng thành cấu thành TMM, vậy bản thân một mức trưởng thành là gì và cấu trúc của nó ra sao? (Hình 07).

Mỗi mức độ, ngoại trừ mức độ thấp nhất là 1, có cấu trúc bao gồm các thành phần sau:

- Mục tiêu trưởng thành: Xác định các mục tiêu cần phải đạt trong việc cải tiến quy trình KTPM. Để đạt một mức trưởng thành, tổ chức phải đạt tất cả các mục tiêu của mức trưởng thành đó.
- Mục tiêu con: Các mục tiêu trưởng thành đã nói ở trên có tầm bao quát rộng. Do vậy để làm rõ hơn phạm vi cũng như những công việc cần làm để đạt được một mục tiêu, mỗi mục tiêu lại được mô tả rõ hơn thông qua những mục tiêu con, dễ hiểu và cụ thể hơn. Nếu ta đạt được tất cả mục tiêu con của một mục tiêu nghĩa là ta đã đạt được mục tiêu đó.
- Công việc và trách nhiệm: Mô tả rõ hơn các công việc cần làm, cũng như ai trong dự án (trưởng dự án, lập trình viên, kiểm tra viên...) sẽ thực hiện các công việc đó. Nghĩa là, để đạt được một mục tiêu con, ta cần thực hiện tất cả các công việc được đề nghị cho mục tiêu con đó.

- Sự tham gia của các nhóm khác nhau: TMM cho rằng có 3 nhóm người quan trọng với cách nhìn và quan điểm khác nhau ảnh hưởng đến công việc KTPM, đó là người quản lý/quản lý dự án, lập trình viên/kiểm tra viên, và khách hàng/người sử dụng. Do vậy mô hình TMM yêu cầu các công việc phải được phân trách nhiệm cho 3 nhóm người này. Ý nghĩa và tổ chức của các mức trưởng thành 5 mức độ trưởng thành do TMM quy định được xác định như hình 08.

Mức trưởng thành 1: Khởi đầu Mức khởi đầu của đa số tổ chức PM, không có mục tiêu nào đặt ra cho mức này. Quy trình KTPM hoàn toàn hỗn độn. KTPM được thực hiện một cách không dự tính và phi thể thức sau khi code được viết xong; không có kế hoạch, không có quy trình. Nói chung ở mức này KTPM đồng nghĩa với tìm lỗi (debugging). Một lập trình viên viết code và sau đó tìm lỗi, sửa chữa, dò lỗi... cho đến khi tin rằng mọi thứ đạt yêu cầu. Kiểm tra viên không được huấn luyện, tài nguyên cần thiết cũng không đầy đủ.

Do hầu như chỉ có lập trình viên làm mọi thứ, chi phí kiểm tra hầu như không biết trước hoặc được bao gồm trong chi phí PTPM.

Mức trưởng thành 2: Định nghĩa KTPM là một quy trình riêng biệt, là một chặng của toàn bộ chu trình PTPM và hoàn toàn phân biệt với công việc dò tìm lỗi (debug). Mục tiêu của kiểm tra nhằm chứng minh PM hoặc hệ thống đáp ứng được các yêu cầu.

KTPM được lập kế hoạch chi tiết và được theo dõi chặt chẽ. Quy trình kiểm tra có thể được sử dụng lặp lại trong các dự án khác nhau. Kế hoạch kiểm tra thường được hoàn thành sau khi đã xong giai đoạn viết code. Kỹ thuật và phương pháp kiểm tra cơ bản được thiết lập và đưa vào sử dụng. Các mục tiêu của mức 2 bao gồm:

- Phát triển các mục tiêu dò lỗi và kiểm tra phần mềm
- Quy trình lập kế hoạch kiểm tra
- Thể chế hóa các kỹ thuật và phương pháp kiểm tra cơ bản

So sánh mức 2 giữa TMM và CMM:

TMM Mức 2: Định nghĩa	CMM Mức 2: Có thể lập lại
<ul style="list-style-type: none"> <li>• Kỹ thuật và phương pháp kiểm tra cơ bản</li> <li>• Quy trình lập kế hoạch kiểm tra</li> <li>• Mục tiêu dò lỗi và kiểm tra phần mềm</li> </ul>	<ul style="list-style-type: none"> <li>• Quản lý yêu cầu</li> <li>• Lập kế hoạch dự án</li> <li>• Giám sát và theo dõi dự án</li> <li>• Quản lý thầu phụ</li> <li>• Đảm bảo chất lượng</li> <li>• Quản lý cấu hình</li> </ul>

Mức trưởng thành 3: Tích hợp Một nhóm kiểm tra viên được thành lập như một bộ phận trong công ty. Kiểm tra viên được huấn luyện kỹ và đặc biệt. KTPM không còn là một chặng, mà được thực hiện xuyên suốt toàn bộ chu kỳ PTPM. Việc sử dụng công cụ kiểm tra tự động bắt đầu được tính đến. Kế hoạch kiểm tra được thực hiện sớm hơn nhiều so với mức trưởng thành 2. Quy trình kiểm tra được giám sát, tiến độ và hiệu quả kiểm tra được kiểm soát chặt chẽ. Mục tiêu của mức 3 bao gồm:

- Thiết lập bộ phận KTPM
- Thiết lập chương trình huấn luyện kỹ thuật
- Tích hợp KTPM vào chu kỳ PTPM
- Kiểm soát và giám sát quy trình kiểm tra

So sánh mức 3 giữa TMM và CMM:



TMM Mức 3: Tích hợp	CMM Mức 3: Được định nghĩa
<ul style="list-style-type: none"> <li>• Kiểm soát và giám sát quy trình kiểm tra</li> <li>• Tích hợp kiểm tra phần mềm</li> <li>• Thiết lập chương trình huấn luyện kỹ thuật</li> <li>• Thiết lập tổ chức kiểm tra phần mềm</li> </ul>	<ul style="list-style-type: none"> <li>• Tập trung quy trình cấp tổ chức</li> <li>• Định nghĩa quy trình cấp tổ chức</li> <li>• Chương trình huấn luyện</li> <li>• Tích hợp quản lý phần mềm</li> <li>• Kỹ thuật phát triển sản phẩm</li> <li>• Điều phối liên nhóm</li> <li>• Xem xét ngang hàng</li> </ul>

Mức trưởng thành 4: Quản lý và đo lường Một chương trình xem xét cấp công ty được thành lập với mục tiêu loại bỏ sai sót trong sản phẩm kể cả sản phẩm trung gian bằng kỹ thuật xem xét ngang hàng (peer review – kỹ thuật phổ biến để phát hiện lỗi sớm trên các sản phẩm và sản phẩm trung gian không thi hành được như yêu cầu khách hàng, bản thiết kế, mã nguồn, kế hoạch kiểm tra... được thực hiện bởi một nhóm người cùng làm việc).

Quy trình kiểm tra là một quy trình định lượng. Các chỉ số liên quan đến KTPM được định nghĩa và thu thập nhằm phân tích, khảo sát chất lượng và hiệu quả của quy trình kiểm tra. Một số ví dụ về các chỉ số này như: tỷ lệ lỗi trên một đơn vị kích thước PM, số lượng lỗi do kiểm tra viên tìm thấy trên tổng số lỗi của PM (bao gồm lỗi do khách hàng phát hiện), thời gian trung bình để sửa chữa một lỗi... Mục tiêu của mức 4 bao gồm:

- Thiết lập chương trình xem xét xuyên suốt các dự án trong công ty
- Thiết lập chương trình đo lường việc KTPM
- Đánh giá chất lượng PM

So sánh mức 4 giữa TMM và CMM

TMM Mức 4: Quản lý và đo lường	CMM Mức 4: Được quản lý
<ul style="list-style-type: none"> <li>• Đánh giá chất lượng phần mềm</li> <li>• Đo lường việc kiểm tra phần mềm</li> <li>• Chương trình xem xét xuyên dự án</li> </ul>	<ul style="list-style-type: none"> <li>• Quản lý quy trình theo lượng hóa</li> <li>• Quản lý chất lượng phần mềm</li> </ul>

Mức trưởng thành 5: Tối ưu hóa, phòng ngừa lỗi và kiểm soát chất lượng. Dữ liệu liên quan đến các sai sót đã thu thập (ở mức 4) được phân tích để tìm ra nguyên nhân gốc phát sinh các sai sót đó. Căn cứ vào các nguyên nhân này, hành động phòng ngừa được thiết lập và thi hành. Các phép thống kê được dùng để ước lượng tính tin cậy của phần mềm, cũng như làm cơ sở cho các quyết định liên quan đến xác định các mục tiêu về độ tin cậy của phần mềm. Chi phí và tính hiệu quả của KTPM được giám sát chặt chẽ, công cụ kiểm tra tự động được sử dụng rộng rãi.

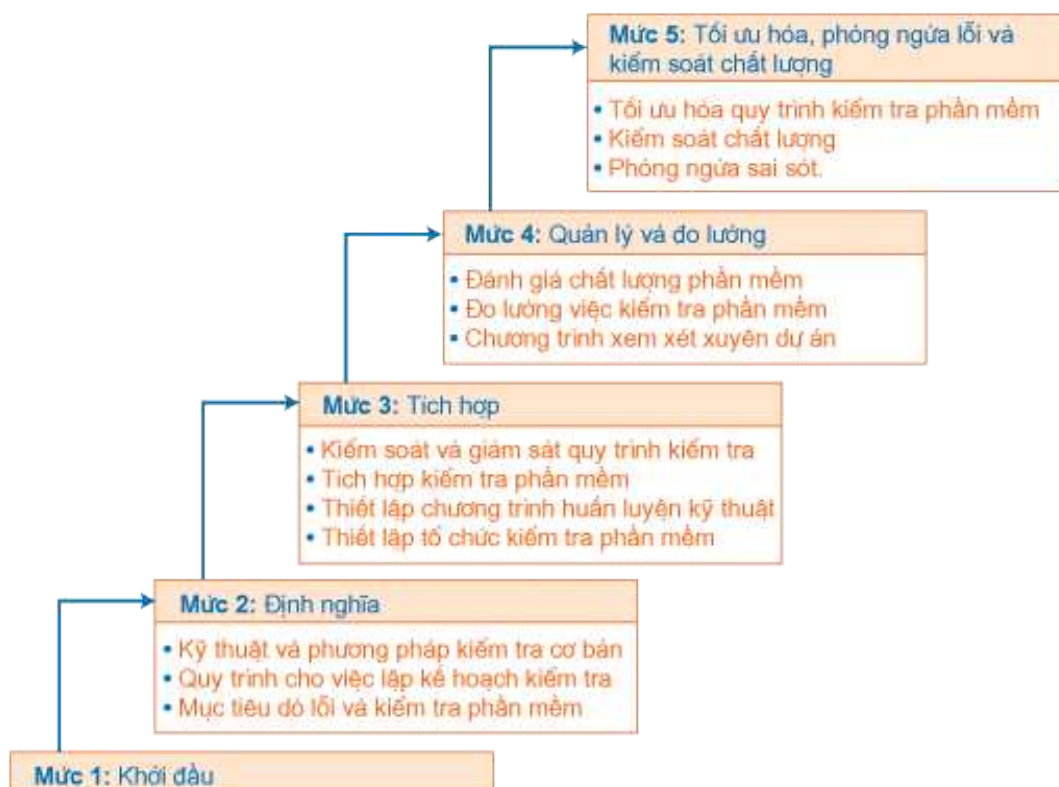
Mặt khác, ở mức 5, quy trình KTPM phải được cải tiến một cách liên tục, nhằm khắc phục những yếu kém của quy trình, cũng như hướng đến những mục tiêu xa hơn. Mục tiêu của mức 5 bao gồm:

- Sử dụng dữ liệu thu thập để phòng ngừa sai sót.
- Kiểm soát chất lượng
- Tối ưu hóa quy trình KTPM

So sánh mức 5 giữa TMM và CMM:

TMM Mức 5: Tối ưu hóa, phòng ngừa lỗi và kiểm soát chất lượng	CMM Mức 5: Tối ưu hóa
<ul style="list-style-type: none"> <li>• Phòng ngừa sai sót.</li> <li>• Kiểm soát chất lượng</li> <li>• Tối ưu hóa quy trình kiểm tra phần mềm</li> </ul>	<ul style="list-style-type: none"> <li>• Phòng ngừa sai sót.</li> <li>• Quản lý thay đổi kỹ thuật/công nghệ</li> <li>• Quản lý thay đổi quy trình</li> </ul>

KẾT LUẬN KTPM là một lĩnh vực rất quan trọng trong hoạt động sản xuất cũng như gia công PM. Các mức kiểm tra và loại kiểm tra rất phong phú, phục vụ mục tiêu đảm bảo chất lượng toàn diện cho một PM hoặc một hệ thống. Trong thực tế, để triển khai tất cả các mức và loại kiểm tra đã liệt kê cho một dự án PM đòi hỏi sự đầu tư rất lớn cả về thời gian lẫn công sức. Các tổ chức “còn non” trong quy trình kiểm tra thường cố gắng tiết kiệm tối đa đầu tư vào KTPM, thường lờ việc lập kế hoạch kiểm tra đến khi hoàn thành việc viết code, bỏ qua một vài hoặc hầu hết các chặng kiểm tra. PM giao cho khách hàng trong điều kiện như thế thường nghèo nàn về chất lượng. Kết quả thường là sự đột biến về chi phí bỏ ra cho việc sửa chữa lỗi, hoặc bảo trì PM, tuy nhiên sự mất mát lớn nhất là sự thất vọng của khách hàng hoặc những người dùng cuối.



## Năm mức độ trưởng thành trong TMM

Mẫu quy trình kiểm thử phần mềm:

### 1.1 Mục đích

Tài liệu này mô tả các bước công việc, trình tự thực hiện việc kiểm tra hệ thống phần mềm từ khi có yêu cầu kiểm tra cho đến khi sản phẩm được thông báo phát hành, nhằm mục đích:

- Đảm bảo chất lượng phần mềm đúng theo yêu cầu và tiêu chuẩn thiết kế, chuyển giao cho khách hàng những sản phẩm đạt chất lượng.
- Chuẩn hóa và chuyên nghiệp hóa quá trình đảm bảo chất lượng phần mềm.
- Giảm rủi ro và chi phí trong quá trình phát triển phần mềm.

### 1.2 Phạm vi áp dụng

Áp dụng cho nhóm đảm bảo chất lượng phần mềm trong quá trình thực hiện việc kiểm tra hệ

thống phần mềm.

### 1.3 Tài liệu liên quan

- MTQT: Quy trình phát triển phần mềm ESDM.
- HDCV: Thuật ngữ ESDM.
- MTSP: Mô tả tài liệu sản phẩm phần mềm.
- MTCV: Các mô tả công việc của nhóm đảm bảo chất lượng (PQA).
- Biểu mẫu: Kế hoạch kiểm tra phần mềm.
- Biểu mẫu: Tình huống kiểm tra phần mềm.
- Biểu mẫu: Kết quả kiểm tra hệ thống phần mềm.
- Biểu mẫu: Thông báo phát hành phần mềm.
- Biểu mẫu: Tài liệu hướng dẫn sử dụng
- Biểu mẫu : Biên bản ghi nhận môi trường kiểm tra
- Biểu mẫu – Kế hoạch công việc, báo cáo công việc, biên bản họp...

#### **1.4 Định nghĩa, thuật ngữ**

*DMS (Defect Management System):* Hệ thống/công cụ quản lý lỗi

*PQA (Product Quality Assurance):* Đảm bảo chất lượng sản phẩm

### **3 Thông tin chung**

#### **3.1 Điều kiện bắt đầu/kết thúc**

##### **3.1.1 Điều kiện bắt đầu**

Khi có yêu cầu kiểm tra hệ thống phần mềm

##### **3.1.2 Điều kiện kết thúc**

Thông báo phát hành sản phẩm phần mềm

#### **3.2 Thông tin đầu vào/kết quả**

##### **3.2.1 Thông tin đầu vào**

- Kế hoạch dự án
- Tài liệu SRS
- Tài liệu HLD
- Prototype
- Tài liệu mô tả nghiệp vụ
- Các tài liệu khác liên quan

### **3.2.2 Kết quả chính**

- Kế hoạch kiểm tra
- Kịch bản kiểm tra
- Tình huống kiểm tra
- Thông báo phát hành phần mềm
- Tài liệu hướng dẫn sử dụng
- Các báo cáo công việc

### **3.3 Chỉ tiêu chất lượng**

- Tỷ lệ thời gian thực hiện/thời gian kế hoạch
- Tỷ lệ nguồn lực thực tế/nguồn lực kế hoạch
- Tỷ lệ lỗi/tổng số chức năng kiểm tra
- Vòng đời trung bình của lỗi (thời gian từ khi phát hiện lỗi cho đến khi lỗi được sửa)

## Quy trình kiểm tra

### Tổng quan

Trong phần trước chúng tôi đã giới thiệu tổng quan về các mức và loại kiểm tra phần mềm (KTPM) cơ bản. Thực tế đi sâu vào từng mức và loại kiểm tra, còn có rất nhiều kiểm tra đặc thù khác nữa, mang tính chuyên biệt cho từng vấn đề hoặc từng loại ứng dụng. Trong phần này, chúng tôi sẽ giới thiệu chi tiết về những bước cơ bản của một quy trình KTPM, làm thế nào để đánh giá và cải tiến năng lực KTPM của một tổ chức thông qua mô hình TMM (Testing Maturity Model), được các chuyên gia đánh giá khá tốt, dành riêng cho hoạt động KTPM.

### Quy trình kiểm tra phần mềm cơ bản

Trước khi tìm hiểu một quy trình kiểm tra phần mềm cơ bản, ta cần hiểu hai khái niệm sau: Test Case và Test Script.

#### Test Case

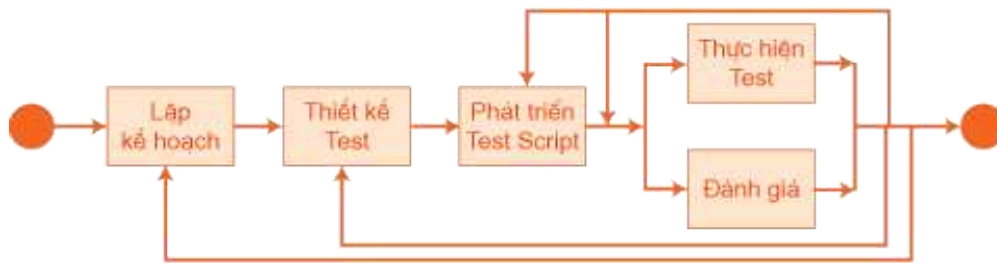
Một Test Case có thể coi nôm na là một tình huống kiểm tra, được thiết kế để kiểm tra một đối tượng có thỏa mãn yêu cầu đặt ra hay không. Một Test Case thường bao gồm 3 phần cơ bản:

- Mô tả: đặc tả các điều kiện cần có để tiến hành kiểm tra.
- Nhập: đặc tả đối tượng hay dữ liệu cần thiết, được sử dụng làm đầu vào để thực hiện việc kiểm tra.
- Kết quả mong chờ: kết quả trả về từ đối tượng kiểm tra, chứng tỏ đối tượng đạt yêu cầu.

#### Test Script

Một Test Script là một nhóm mã lệnh dạng đặc tả kịch bản dùng để tự động hóa một trình tự kiểm tra, giúp cho việc kiểm tra nhanh hơn, hoặc cho những trường hợp mà kiểm tra bằng tay sẽ rất khó khăn hoặc không khả thi. Các Test Script có thể tạo thủ công hoặc tạo tự động dùng công cụ kiểm tra tự động. (Hình 04)

Phần sau sẽ giải thích rõ hơn các bước cơ bản của một quy trình kiểm tra.



*Một quy trình kiểm tra cơ bản có thể áp dụng rộng rãi cho nhiều hệ thống PM với những đặc trưng khác nhau.*

### Lập kế hoạch kiểm tra

Mục đích: Nhằm chỉ định và mô tả các loại kiểm tra sẽ được triển khai và thực hiện. Kết quả của bước lập kế hoạch là bản tài liệu kế hoạch KTPM, bao gồm nhiều chi tiết từ các loại kiểm tra, chiến lược kiểm tra, cho đến thời gian và phân định lực lượng kiểm tra viên.

Bản kế hoạch kiểm tra đầu tiên được phát triển rất sớm trong chu trình phát triển phần mềm (PTPM), ngay từ khi các yêu cầu đã tương đối đầy đủ, các chức năng và luồng dữ liệu chính đã được mô tả. Bản kế hoạch này có thể được coi là bản kế hoạch chính (master test plan), trong đó tất cả các kế hoạch chi tiết cho các mức kiểm tra và loại kiểm tra khác nhau đều được đề cập (hình 05).

Lưu ý, tùy theo đặc trưng và độ phức tạp của mỗi dự án, các kế hoạch kiểm tra chi tiết có thể được gom chung vào bản kế hoạch chính hoặc được phát triển riêng.

Sau khi bản kế hoạch chính được phát triển, các bản kế hoạch chi tiết lần lượt được thiết kế theo trình tự thời gian phát triển của dự án. (Hình 06 minh họa thời điểm phù hợp để thiết lập các kế hoạch kiểm tra, gắn liền với quá trình phát triển của dự án. Quá trình phát triển các kế hoạch kiểm tra không dừng lại tại một thời điểm, mà liên tục được cập nhật chỉnh sửa cho phù hợp đến tận cuối dự án.).



*Bản kế hoạch chính và các bản kế hoạch chi tiết*

Các bước lập kế hoạch:



Xác định yêu cầu kiểm tra: chỉ định bộ phận, thành phần của PM sẽ được kiểm tra, phạm vi hoặc giới hạn của việc kiểm tra. Yêu cầu kiểm tra cũng được dùng để xác định nhu cầu nhân lực.

Khảo sát rủi ro: Các rủi ro có khả năng xảy ra làm chậm hoặc cản trở quá trình cũng như chất lượng kiểm tra. Ví dụ: kỹ năng và kinh nghiệm của kiểm tra viên quá yếu, không hiểu rõ yêu cầu.

Xác định chiến lược kiểm tra: chỉ định phương pháp tiếp cận để thực hiện việc kiểm tra trên PM, chỉ định các kỹ thuật và công cụ hỗ trợ kiểm tra, chỉ định các phương pháp dùng để đánh giá chất lượng kiểm tra cũng như điều kiện để xác định thời gian kiểm tra.

Xác định nhân lực, vật lực: kỹ năng, kinh nghiệm của kiểm tra viên; phần cứng, phần mềm, công cụ, thiết bị giả lập... cần thiết cho việc kiểm tra.

Lập kế hoạch chi tiết: ước lượng thời gian, khối lượng công việc, xác định chi tiết các phần công việc, người thực hiện, thời gian tất cả các điểm mốc của quá trình kiểm tra.

Tổng hợp và tạo các bản kế hoạch kiểm tra: kế hoạch chung và kế hoạch chi tiết.

Xem xét các kế hoạch kiểm tra: phải có sự tham gia của tất cả những người có liên quan, kể cả trưởng dự án và có thể cả khách hàng. Việc xem xét nhằm bảo đảm các kế hoạch là khả thi, cũng như để phát hiện (và sửa chữa sau đó) các sai sót trong các bản kế hoạch.

## Thiết kế Test

Mục đích: Nhằm chỉ định các Test Case và các bước kiểm tra chi tiết cho mỗi phiên bản PM. Giai đoạn thiết kế test là hết sức quan trọng, nó bảo đảm tất cả các tình huống kiểm tra “quét” hết tất cả yêu cầu cần kiểm tra.

Hình dưới cho thấy việc thiết kế test không phải chỉ làm một lần, nó sẽ được sửa chữa, cập nhật, thêm hoặc bớt xuyên suốt chu kỳ PTPM, vào bất cứ lúc nào có sự thay đổi yêu cầu, hoặc sau khi phân tích thấy cần được sửa chữa hoặc bổ sung.



## *Thời điểm phù hợp để thiết lập các kế hoạch kiểm tra*

Các bước thiết kế test bao gồm:

- Xác định và mô tả Test Case: xác định các điều kiện cần thiết lập trước và trong lúc kiểm tra. Mô tả đối tượng hoặc dữ liệu đầu vào, mô tả các kết quả mong chờ sau khi kiểm tra.
- Mô tả các bước chi tiết để kiểm tra: các bước này mô tả chi tiết để hoàn thành một Test Case khi thực hiện kiểm tra. Các Test Case như đã nói ở trên thường chỉ mô tả đầu vào, đầu ra, còn cách thức tiến hành như thế nào thì không được định nghĩa. Thao tác này nhằm chi tiết hóa các bước của một Test Case, cũng như chỉ định các loại dữ liệu nào cần có để thực thi các Test Case, chúng bao gồm các loại dữ liệu trực tiếp, gián tiếp, trung gian, hệ thống...
- Xem xét và khảo sát độ bao phủ của việc kiểm tra: mô tả các chỉ số và cách thức xác định việc kiểm tra đã hoàn thành hay chưa? bao nhiêu phần trăm PM đã được kiểm tra? Để xác định điều này có hai phương pháp: căn cứ trên yêu cầu của phần mềm hoặc căn cứ trên số lượng code đã viết.
- Xem xét Test Case và các bước kiểm tra: Việc xem xét cần có sự tham gia của tất cả những người có liên quan, kể cả trưởng dự án nhằm bảo đảm các Test Case và dữ liệu yêu cầu là đủ và phản ánh đúng các yêu cầu cần kiểm tra, độ bao phủ đạt yêu cầu, cũng như để phát hiện (và sửa chữa) các sai sót.

**Phát triển Test Script**  
Mục đích: Bước này thường không bắt buộc trong các loại và mức kiểm tra, chỉ yêu cầu trong những trường hợp đặc thù cần thiết kế, tạo ra các Test Script có khả năng chạy trên máy tính giúp tự động hóa việc thực thi các bước kiểm tra đã định nghĩa ở bước thiết kế test.

Các bước phát triển Test Script bao gồm:

- Tạo Test Script: thủ công hoặc dùng công cụ hỗ trợ để phát sinh script một cách tự động (tuy nhiên trong hầu hết mọi trường hợp, ta vẫn phải chỉnh sửa ít hoặc nhiều trên các script được sinh tự động). Thông thường, mỗi bước kiểm tra được thiết kế trong phần thiết kế test, đòi hỏi ít nhất một Test Script. Các Test Script có khả năng tái sử dụng càng nhiều càng tốt để tối ưu hóa công việc.

- Kiểm tra Test script: xem có “chạy” tốt không nhằm bảo đảm các Test Script hoạt động đúng yêu cầu, thể hiện đúng ý đồ của các bước kiểm tra.

- Thành lập các bộ dữ liệu ngoài dành cho các Test Script: bộ dữ liệu này sẽ được các Test Script sử dụng khi thực hiện kiểm tra tự động. Gọi là “ngoài” vì chúng được lưu độc lập với các Test Script, tránh trường hợp vì dễ dãi, một số kiểm tra viên “tích hợp” luôn phần dữ liệu vào bên trong code của các script (thuật ngữ chuyên môn gọi là “hard-

code”). Việc tách riêng dữ liệu cho phép dễ dàng thay đổi dữ liệu khi kiểm tra, cũng như giúp việc chỉnh sửa hoặc tái sử dụng các script sau này.

- Xem xét và khảo sát độ bao phủ của việc kiểm tra: bảo đảm các Test Script được tạo ra bao phủ toàn bộ các bước kiểm tra theo yêu cầu.

**Thực hiện kiểm tra**  
Mục đích: Thực hiện các bước kiểm tra đã thiết kế (hoặc thi hành các Test Script nếu tiến hành kiểm tra tự động) và ghi nhận kết quả.

Hình 06 cho ta thấy, việc thực hiện kiểm tra cũng được làm rất nhiều lần trong suốt chu trình kiểm tra, cho đến khi kết quả kiểm tra cho thấy đủ điều kiện để dừng hoặc tạm dừng việc thực hiện.

Quá trình thực hiện kiểm tra thường thông qua các bước sau:

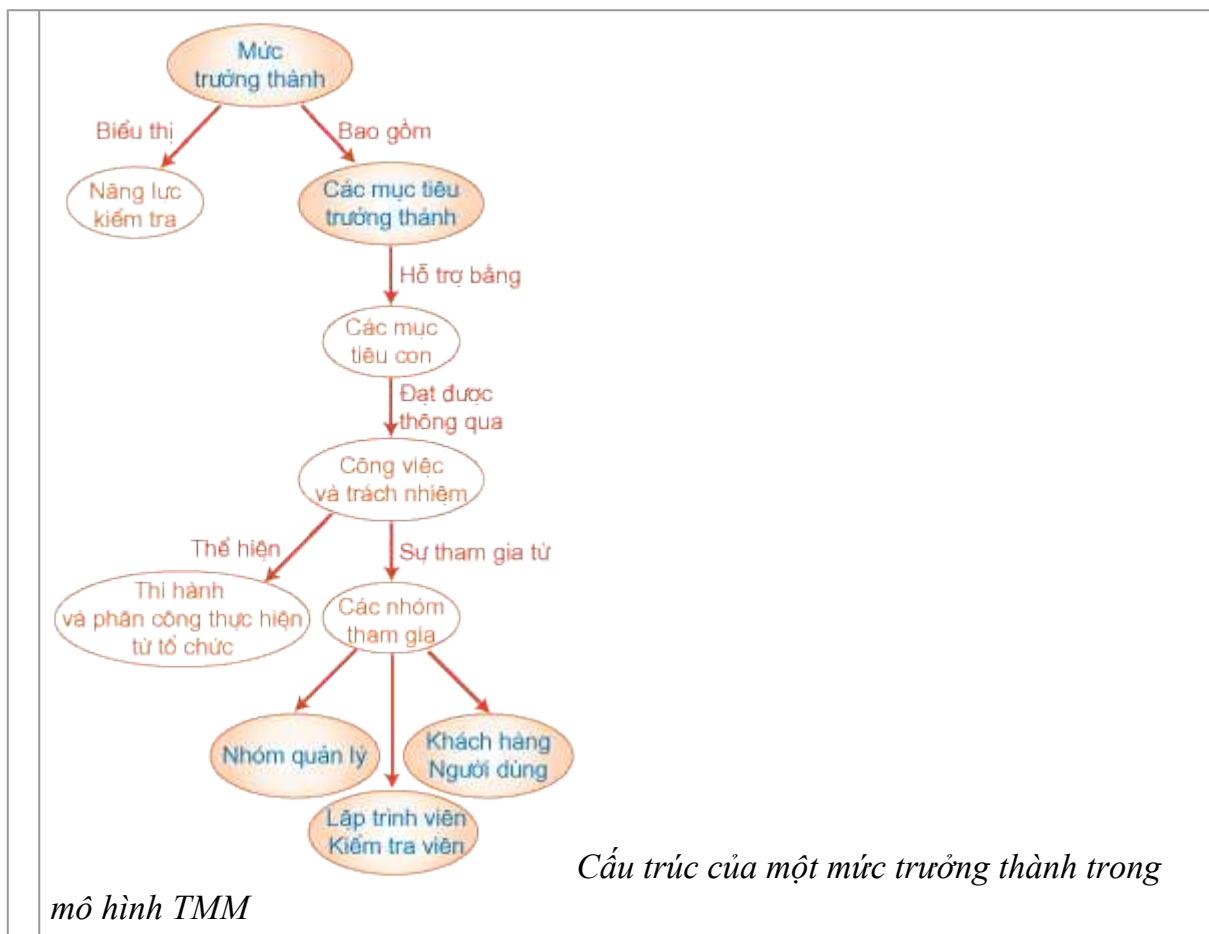
- Thực hiện các bước kiểm tra: thủ công hoặc thi hành các Test Script nếu là quy trình kiểm tra tự động. Để thực hiện kiểm tra, thao tác đầu tiên cần làm là xác lập và khởi động môi trường và điều kiện kiểm tra. Việc này nhằm bảo đảm tất cả các bộ phận liên quan (như phần cứng, phần mềm, máy chủ, mạng, dữ liệu...) đã được cài đặt và sẵn sàng, trước khi chính thức bắt đầu thực hiện kiểm tra.
- Đánh giá quá trình kiểm tra: giám sát quá trình kiểm tra suốt từ khi hoàn thành hay bị treo và dừng giữa chừng, có cần bổ sung hay sửa chữa gì không để quá trình kiểm tra được tốt hơn.
- Nếu quá trình diễn ra trơn tru, kiểm tra viên hoàn thành chu kỳ kiểm tra và chuyển qua bước “Thẩm định kết quả kiểm tra”
- Nếu quá trình bị treo hoặc dừng giữa chừng, kiểm tra viên cần phân tích để xác định nguyên nhân lỗi, khắc phục lỗi và lập lại quá trình kiểm tra.
- Thẩm định kết quả kiểm tra: sau khi kết thúc, kết quả kiểm tra cần được xem xét để bảo đảm kết quả nhận được là đáng tin cậy, cũng như nhận biết được những lỗi xảy ra không phải do PM mà do dữ liệu dùng để kiểm tra, môi trường kiểm tra hoặc các bước kiểm tra (hoặc Test Script) gây ra. Nếu thực sự lỗi xảy ra do quá trình kiểm tra, cần phải sửa chữa và kiểm tra lại từ đầu.

**Đánh giá quá trình kiểm tra**  
Mục đích: Đánh giá toàn bộ quá trình kiểm tra, bao gồm xem xét và đánh giá kết quả kiểm tra, liệt kê lỗi, chỉ định các yêu cầu thay đổi, và tính toán các số liệu liên quan đến quá trình kiểm tra (chẳng hạn số giờ, thời gian kiểm tra, số lượng lỗi, phân loại lỗi...).

Lưu ý, mục đích của việc đánh giá kết quả kiểm tra ở bước này hoàn toàn khác với bước thẩm định kết quả kiểm tra sau khi hoàn tất một vòng kiểm tra. Đánh giá kết quả kiểm

tra ở giai đoạn này mang tính toàn cục và nhằm vào bản thân giá trị của các kết quả kiểm tra.

Hình 06 cho thấy, việc đánh giá quá trình và kết quả kiểm tra được thực hiện song song với bất kỳ lần kiểm tra nào và chỉ chấm dứt khi quá trình kiểm tra đã hoàn tất.



Đánh giá quá trình kiểm tra thường thông qua các bước sau:

- Phân tích kết quả kiểm tra và đề xuất yêu cầu sửa chữa: Chỉ định và đánh giá sự khác biệt giữa kết quả mong chờ và kết quả kiểm tra thực tế, tổng hợp và gửi thông tin yêu cầu sửa chữa đến những người có trách nhiệm trong dự án, lưu trữ để kiểm tra sau đó.

- Đánh giá độ bao phủ: Xác định quá trình kiểm tra có đạt được độ bao phủ yêu cầu hay không, tỷ lệ yêu cầu đã được kiểm tra (tính trên các yêu cầu của PM và số lượng code đã viết).

- Phân tích lỗi: Đưa ra số liệu phục vụ cho việc cải tiến các qui trình phát triển, giảm sai sót cho các chu kỳ phát triển và kiểm tra sau đó. Ví dụ, tính toán tỷ lệ phát sinh lỗi, xu hướng gây ra lỗi, những lỗi “ngoan cố” hoặc thường xuyên tái xuất hiện.

- Xác định quá trình kiểm tra có đạt yêu cầu hay không: Phân tích đánh giá để xem các Test Case và chiến lược kiểm tra đã thiết kế có bao phủ hết những điểm cần kiểm tra hay không? Kiểm tra có đạt yêu cầu dự án không? Từ những kết quả này, kiểm tra viên có thể sẽ phải thay đổi chiến lược hoặc cách thức kiểm tra.

- Báo cáo tổng hợp: Tổng hợp kết quả các bước ở trên và phải được gửi cho tất cả những người có liên quan.

Tóm lược: Trên đây là tóm tắt các bước cơ bản của một quy trình KTPM. Tùy theo đặc thù của dự án, loại kiểm tra và mức độ kiểm tra, quy trình kiểm tra trong thực tế có thể chi tiết hơn nhiều, tuy nhiên các bước trên là xương sống của bất kỳ quy trình kiểm tra nào.

Sau đây, chúng tôi sẽ giới thiệu một mô hình giúp các tổ chức đánh giá và nâng cao năng lực KTPM của mình, đó là mô hình TMM (Testing Maturity Model).

# Mô tả

## 4 Mô tả các công việc chi tiết

### 4.1 Lập kế hoạch kiểm tra

#### 4.1.1 Đầu vào

- Kế hoạch dự án
- Tài liệu SRS, SRD, HLD
- Các tài liệu mô tả nghiệp vụ

#### 4.1.2 Mô tả

Sau khi tiếp nhận yêu cầu kiểm tra của dự án phần mềm, nhóm PQA tiến hành lập kế hoạch kiểm tra. Việc lập kế hoạch kiểm tra hệ thống phần mềm nhằm mục đích xác định các yêu cầu, mục đích, phương thức kiểm tra cũng như xác định nguồn lực và thời gian, công cụ kiểm tra cần thiết cho toàn bộ quá trình kiểm tra, trước khi tiến hành kiểm tra hệ thống phần mềm. Giai đoạn này được tiến hành song song với thời gian phát triển dự án phần mềm. Khi đã có kế hoạch dự án, tài liệu SRS và HLD thì có thể tiến hành xây dựng kế hoạch kiểm tra. Đồng thời, nhóm PQA thống nhất phương thức giao tiếp với nhóm phát triển.

- Nhóm dự án cung cấp các tài liệu liên quan của phần mềm cho nhóm PQA
- Nhóm PQA nghiên cứu các yêu cầu, thời gian và nội dung các yêu cầu nhận được để quyết định các bước tiếp theo.
- Nhóm PQA chuẩn bị nhân lực và đào tạo nghiệp vụ cần thiết phục vụ cho nhu cầu dự án

Cán bộ PQA lập kế hoạch kiểm tra và trình Giám Đốc Dự án phê duyệt. Bản kế hoạch kiểm tra đã phê duyệt sẽ được phổ biến đến từng thành viên tham gia quá trình kiểm tra và là cơ sở cho việc đánh giá mức độ hoàn thành công việc và quản lý nhóm làm việc, tiến độ công việc.

#### 4.1.3 Công việc

- T4101: Tiếp nhận yêu cầu kiểm tra

- T4102: Tham gia quá trình phân tích và xác định yêu cầu
- T4103: Tiếp nhận tài liệu
- T4104: Lập kế hoạch kiểm tra
- T4105: Trình phê duyệt kế hoạch kiểm tra
- T4106: Phổ biến kế hoạch kiểm tra đã được phê duyệt

#### **4.1.4 Kết quả**

- Kế hoạch kiểm tra được phê duyệt.

#### **4.1.5 Người thực hiện**

- Cán bộ PQA
- Trưởng nhóm PQA
- Giám đốc dự án

### **4.2 Chuẩn bị môi trường kiểm tra**

#### **4.2.1 Đầu vào**

- Kế hoạch kiểm tra
- Tài liệu SRS, SRD, HLD
- Tài liệu nghiệp vụ

#### **4.2.2 Mô tả**

Chuẩn bị dữ liệu và môi trường kiểm tra là tiền đề để đảm bảo cho việc kiểm tra có chất lượng và năng suất cao. Tiến hành xây dựng dữ liệu, dựng môi trường, kiểm tra thiết bị cần thiết, công cụ kiểm tra đã đáp ứng kế hoạch kiểm tra đề ra. Mục đích của giai đoạn này là chuẩn bị môi trường kiểm tra cho hệ thống phần mềm.

Cán bộ kiểm tra có trách nhiệm phối hợp cùng với bên hỗ trợ kỹ thuật để chuẩn bị đầy đủ môi trường thực tế hay giả lập cho chương trình, dự án phần mềm trước khi tiến hành kiểm tra.

#### **4.2.3 Công việc**

- T4201: Xây dựng các công cụ tạo dữ liệu, kiểm tra cơ sở dữ liệu.
- T4202: Tạo dữ liệu kiểm tra hệ thống cho chương trình/công cụ kiểm tra.
- T4203: Xây dựng môi trường phân cứng
- T4204: Xây dựng môi trường phân mềm

#### **4.2.4 Kết quả**

- Môi trường kiểm tra đã sẵn sàng

#### **4.2.5 Người thực hiện**

- Cán bộ PQA
- Trưởng nhóm PQA

### **4.3 Thiết kế kiểm tra**

#### **4.3.1 Đầu vào**

- Tài liệu SRS, SRD, HLD
- Prototype (nếu có)

#### **4.3.2 Mô tả**

Giai đoạn thiết kế kiểm tra nhằm đưa ra các tình huống kiểm tra. Đây là tài liệu cụ thể hoá các bước kiểm tra sẽ phải tiến hành trong quá trình thực hiện tình huống kiểm tra. Các tình huống kiểm tra cần chỉ rõ phần mềm có thể đáp ứng được các yêu cầu nào, có những khía cạnh nào. Các khía cạnh cần tập trung là:

- Các yêu cầu chức năng bắt buộc, các yêu cầu chức năng tùy chọn hay phi chức năng
- Các tính năng bảo mật mà hệ thống cần đáp ứng
- Tính ổn định của chương trình
- Khả năng phục hồi của hệ thống
- Các giai đoạn tiến hành kiểm tra



Việc lập các tình huống kiểm tra sẽ do cán bộ kiểm tra đảm nhiệm, quá trình lập tình huống sẽ được Trưởng nhóm kiểm tra thường xuyên xem xét và sửa đổi, cập nhật nếu cần.

#### **4.3.3 Công v i ệ c**

- T4301: Phân tích yêu cầu
- T4302: Lập tình huống kiểm tra

#### **4.3.4 Kết q u ả**

- Tình huống kiểm tra được phê duyệt

#### **4.3.5 N g u ời i thực hiện**

- Cán bộ kiểm tra
- Trưởng nhóm PQA
- Quản trị dự án

### **4.4 Th ự c hiện kiểm tra**

#### **4.4.1 Đầ u v ào**

- Kế hoạch kiểm tra
- Tình huống kiểm tra
- Môi trường kiểm tra đã sẵn sàng

#### **4.4.2 Mô t ả**

Giai đoạn thực hiện tình huống kiểm tra là giai đoạn quan trọng nhất trong toàn bộ quy trình kiểm tra hệ thống phần mềm, vì nó là quá trình thực hiện toàn bộ kế hoạch, tình huống kiểm tra và các hoạt động khác; nhằm bảo đảm phần mềm được phát hành và chuyên giao đáp ứng các yêu cầu hoặc các tiêu chuẩn được xác lập. Các công việc kiểm tra cần chỉ rõ phần mềm đã đáp ứng hoặc chưa đáp ứng các yêu cầu nào, có những lỗi nào. Các khía cạnh cần tập trung là:

- Khẳng định các module riêng biệt và các giao dịch không có lỗi
- Hệ thống đáp ứng các yêu cầu về giao diện

- Hệ thống thực hiện đúng quy trình, thao tác nghiệp vụ.
- Sự ổn định, tốc độ yêu cầu với khối lượng giao dịch/dữ liệu trong phạm vi cho phép
- Sự ổn định, tốc độ yêu cầu với môi trường tối thiểu
- Hệ thống đáp ứng các yêu cầu về tính thống nhất (ngôn ngữ, phím nóng, thuật ngữ,...)

Các lỗi được tìm thấy trong quá trình kiểm tra được ghi nhận và theo dõi trạng thái lỗi, tình trạng lỗi ...đảm bảo rằng lỗi không được ở trạng thái Open quá lâu, các lỗi nghiêm trọng phải được ưu tiên sửa trước...Kiểm tra lại những lỗi đã được sửa và các chức năng liên quan đảm bảo lỗi được Đóng triệt để và không ảnh hưởng đến các lỗi khác, các chức năng khác...

Việc thực hiện tình huống kiểm tra sẽ do cán bộ kiểm tra đảm nhiệm.Trong quá trình thực hiện tình huống kiểm tra, Trưởng nhóm kiểm tra thường xuyên xem xét (nếu cần).

#### **4.4.3 Công v i ệc**

- T4401: Tạo dữ liệu mô phỏng
- T4402: Thực hiện kiểm tra theo tình huống kiểm tra
- T4403: Ghi nhận lỗi

#### **4.4.4 Kết q uả**

- Báo cáo kết quả kiểm tra
- Tình huống kiểm tra được cập nhật (nếu có)

#### **4.4.5 N gư ời i thực hiện**

- Cán bộ kiểm tra
- Trưởng nhóm PQA
- Quản trị dự án

### **4.5 Theo dõi xử lý lỗi**

#### **4.5.1 Đầu v ào**

- Báo cáo kết quả kiểm tra

- Tình huống kiểm tra

#### **4.5.2 Mô t ả**

Theo dõi xử lý lỗi nhằm phân tích, tổng hợp các lỗi mới nhất để gửi tới nhóm phát triển tiến hành sửa đổi cũng như cập nhật các tình huống kiểm tra mới vào tài liệu tình huống kiểm tra khi có các lỗi phát sinh mới, nhằm đảm bảo:

- Lỗi được sửa đúng tiến độ
- Thực hiện kiểm tra lại các lỗi đã được sửa

Việc xử lý lỗi cũng phải được tập hợp vào công cụ kiểm tra, dữ liệu kiểm tra. Các công việc theo dõi và xử lý sẽ do Trưởng nhóm kiểm tra đảm nhiệm phối hợp cùng với bên phát triển

#### **4.5.3 Công v i ệc**

- T4501: Thông báo tình trạng lỗi và yêu cầu nhóm phát triển khắc phục
- T4502: Theo dõi tiến độ xử lý lỗi
- T4503: Báo cáo kết quả kiểm tra lỗi

#### **4.5.4 Kết q uả**

- Có được phiên bản mới nhất sau khi được sửa lỗi
- Cập nhật tình huống kiểm tra nếu có lỗi mới phát sinh
- Cập nhật dữ liệu mới vào công cụ kiểm tra
- Kết quả xử lý lỗi phát hiện

#### **4.5.5 N gư ờ i thực hiện**

- Cán bộ kiểm tra
- Trưởng nhóm PQA
- Nhóm phát triển
- Quản trị dự án

## **4.6 Thông kê và báo cáo kiểm tra**

### **4.6.1 Đầu vào**

- Kế hoạch kiểm tra
- Tình huống kiểm tra
- Báo cáo ghi nhận lỗi và xử lý lỗi.

### **4.6.2 Mô tả**

Đây là giai đoạn hoàn thành công việc kiểm tra hệ thống phần mềm, nhằm mục đích tổng hợp kết quả kiểm tra và đánh giá quá trình thực hiện kiểm tra theo kế hoạch. Nhóm kiểm tra thống kê lại số lượng lỗi, các loại lỗi gặp phải và làm báo cáo kết quả kiểm tra hệ thống phần mềm. Đồng thời nhóm Trưởng nhận xét, đánh giá công việc hoàn thành của các cán bộ kiểm

tra, rút kinh nghiệm trong quá trình kiểm tra. Nhóm PQA có trách nhiệm chuyển kết quả kiểm tra và các yêu cầu phát sinh cho Quản trị dự án xác nhận để giảm rủi ro phát sinh. Mặt khác, chuẩn bị thủ tục chuẩn bị thông báo phát hành sản phẩm phần mềm.

### **4.6.3 Công việc**

- T4601: Thống kê lỗi, lỗi phát sinh
- T4602: Báo cáo kết quả kiểm tra hệ thống phần mềm

### **4.6.4 Kết quả**

- Báo cáo kết quả kiểm tra hệ thống phần mềm

### **4.6.5 Người thực hiện**

- Cán bộ PQA
- Trưởng nhóm PQA
- Quản trị dự án

# Lập kế hoạch

## 4.7 Thông báo phát hành

### 4.7.1 Đầu vào

- Tài liệu SRS, SRD
- Kế hoạch kiểm tra
- Báo cáo kết quả kiểm tra hệ thống phần mềm
- Tình huống kiểm đã cập nhật

### 4.7.2 Mô tả

Đây là bước cuối cùng của quá trình kiểm tra nhằm mục đích thông báo sản phẩm phần mềm mới đã đạt yêu cầu và sẵn sàng chuyển giao cho khách hàng.

### 4.7.3 Công việc

- T4701: Làm thủ tục thông báo phát hành

### 4.7.4 Kết quả

- Biên bản thông báo phát hành sản phẩm

### 4.7.5 Người thực hiện

- Trưởng nhóm PQA
- Quản trị dự án

## 5 Hồ sơ / Kết quả

- Tài liệu SRS, SRD, HLD
- Kế hoạch kiểm tra
- Tình huống kiểm tra
- Báo cáo kết quả kiểm tra hệ thống



<Ngày>

<Chức vụ>

**Người phê duyệt:** <Ngày>

<Chức vụ>

## GIỚI THIỆU

### Mục đích

<Mô tả ngắn gọn về mục đích và tổ chức của tài liệu, có mấy phần, mỗi phần nói về cái gì?>

### Thông tin chung

<Mô tả ngắn gọn về mục đích test (các thành phần, ứng dụng, hệ thống, ...) và mục đích của chúng. Mô tả các thông tin về các chức năng và tính năng chính, kiến trúc của nó và lịch sử dự án một cách vắn tắt>

### Tài liệu liên quan

STT	Tên tài liệu	Nguồn	Ghi chú
1	<Kế hoạch dự án>		
2			

### Phạm vi test

<Mô tả các giai đoạn test – ví dụ Unit, Intergration, System.

Mô tả các kiểu test có trong kế hoạch, ví dụ Function hoặc Performance.

Liệt kê các tính năng và chức năng sẽ được hoặc không được test. Đặt độ ưu tiên cho chức năng được test ( nếu cần ).

Liệt kê các giả thiết trong quá trình lập kế hoạch có thể ảnh hưởng đến việc thiết kế, phát triển hoặc thực hiện test.

Định nghĩa các điều kiện để test hồi qui (đặc biệt áp dụng cho các dự án nâng cấp), chu kỳ và phạm vi test hồi qui.

Số lỗi dự kiến>

Ràng buộc

<Liệt kê các ràng buộc trong quá trình test, có thể là:

- Môi trường test khác hoặc thiếu một số hệ thống ngoài cần để giao tiếp với hệ thống cần test (có thể thêm phần tham khảo tài liệu SRS nếu các ràng buộc được mô tả trong SRS)
- Ràng buộc về nguồn lực, lịch trình hoặc thiếu công cụ test, ...>

Liệt kê các mạo hiểm

<Liệt kê các mạo hiểm/rủi ro và phương án khắc phục, phòng ngừa có thể ảnh hưởng đến việc thiết kế, phát triển và thực hiện test. Khi lập tài liệu thì cần xoá dòng hướng dẫn trên đi>

Stt	Mạo hiểm	Phương án khắc phục & phòng ngừa	Mức độ ảnh hưởng (MD)
1	<Nếu không có thì ghi NA>		
2			



# Các yêu cầu test

## CÁC YÊU CẦU CHO TEST

Danh sách dưới đây xác định các thành phần (tình huống test, các yêu cầu chức năng và phi chức năng) được xác định như mục tiêu test. Các thành phần liệt kê trong danh sách này sẽ được test.

<Liệt kê danh sách các yêu cầu chính cho test>

## CHIẾN LƯỢC TEST

<Chiến lược test giới thiệu phương án tiếp cận để test các mục tiêu test.

Những vấn đề chính trong chiến lược test là các kỹ thuật được áp dụng và điều kiện **để biết khi nào việc test được hoàn thành**.

Mô tả các kiểu test dùng trong dự án.

Có thể liệt kê với mỗi kiểu test tương ứng test cho chức năng nào

.

Việc test có thể dừng khi nào.

Ví dụ:

- Nó không còn hữu ích
- Nó đòi hỏi một phạm vi nhất định
- Nó đòi hỏi một số lỗi nhất định phải tìm được
- Hết thời gian
- Tester trả lại gói phần mềm cho LTV khi chưa sửa lỗi

>

## CÁC KIỂU TEST

<Đối với mỗi kiểu test phải giải thích **kỹ thuật, điều kiện hoàn thành và các vấn đề đặc biệt liên quan**.

**Kỹ thuật:** Kỹ thuật phải mô tả việc test được thực hiện như thế nào, bao gồm cả những gì sẽ được test, các hoạt động chính sẽ được thực hiện trong quá trình test và các phương pháp dùng để đánh giá kết quả.

**Điều kiện hoàn thành:** Điều kiện hoàn thành được phát biểu nhằm hai mục đích:

- Xác định chất lượng sản phẩm được chấp nhận
- Xác định thời điểm mà các nỗ lực test được thực hiện thành công

Một điều kiện hoàn thành được phát biểu rõ ràng phải bao gồm:

- Chức năng, hoạt động hoặc các điều kiện được tính toán
- Phương pháp tính toán

Điều kiện hoặc mức độ thích ứng với phép đo

**Các vấn đề đặc biệt:** Phần này phải chỉ ra các ảnh hưởng hoặc phụ thuộc có thể tác động hoặc ảnh hưởng đến nguồn lực test mô tả trong chiến lược. Các ảnh hưởng có thể bao gồm: Nhân công (ví dụ sự sẵn sàng hoặc cần thiết của các nguồn lực khác test để hỗ trợ/tham gia trong test); các ràng buộc (ví dụ hạn chế về thiết bị hoặc sự sẵn sàng hoặc cần thiết/thiếu các thiết bị đặc biệt); các yêu cầu đặc biệt (ví dụ lịch test hoặc truy cập vào hệ thống)

## Một ví dụ

Một ví dụ về mô tả kiểu test:

*Kỹ thuật:*

- Functional Test

Đối với chu trình sự kiện của mỗi UC, sẽ xác định một tập các giao dịch đại diện cho mỗi hành động của tác nhân khi thực hiện UC.

Tối thiểu phải có 2 TC cho mỗi giao dịch, một TC để phản ánh điều kiện tích cực và một phản ánh điều kiện tiêu cực (không được chấp nhận)

Trong giai đoạn đầu tiên, các UC 1-4 và 12 sẽ được test, theo hình thức sau:

UC 1 bắt đầu với tác nhân đã truy cập thành công vào ứng dụng và tại cửa sổ chính, và kết thúc khi người dùng xác định SAVE.

Mỗi TC sẽ được tiến hành và thực hiện bằng cách sử dụng Rational Robot.

Việc kiểm tra và đánh giá việc thực hiện mỗi TC sẽ được thực hiện theo phương pháp sau:

*Thực hiện Test script (Mỗi test script có được thực hiện thành công như mong muốn không?)*

Tình trạng Window hoặc phương pháp kiểm tra Object Data (tiến hành trong các test script) sẽ được dùng để kiểm tra sự hiển thị của các màn hình chính và dữ liệu được xác định được nắm bắt/hiển thị bởi mục tiêu test trong khi thực hiện test.

Cơ sở dữ liệu của các mục tiêu test (sử dụng Microsoft Access) sẽ được kiểm tra trước khi test và kiểm tra lại sau khi test để kiểm chứng rằng các thay đổi thực hiện trong quá trình test đã được phản ánh chính xác trong dữ liệu.

- Performance Test:

Với mỗi UC, xác định một tập các giao dịch, như định nghĩa trong tài liệu phân tích workload, sẽ được tiến hành và thực hiện bằng Rational Suite PerformanceStudio và Rational Robot (GUI scripts)

Ít nhất 3 workloaf được phản ánh trong test script và lịch trình thực hiện test, bao gồm:

Stressed workload: 750 người dùng (15 % quản lý, 50 % bán hàng, 35 % marketing)

Peak workload: 350 người dùng (10 % quản lý, 60 % bán hàng, 30 % marketing)

Nominal workload: 150 người dùng (2 % quản lý, 75% bán hàng, 23 % marketing)

Test script dùng để thực hiện mỗi giao dịch sẽ bao gồm bộ đếm thời gian tương tự để đo thời gian phản hồi, ví dụ tổng thời gian giao dịch (như định nghĩa trong tài liệu phân tích workload), và các hoạt động giao dịch chính hoặc thời gian xử lý.

Test script sẽ thực hiện các workload trong 1 giờ (trừ phi được ghi chú khác trong tài liệu phân tích workload).

Kiểm tra và đánh giá việc thực hiện mỗi thực hiện test (của một workload) bao gồm:

Thực hiện test được theo dõi bằng biểu đồ trạng thái (để xác định rằng việc test và workload được thực hiện như mong muốn)

Thực hiện test script (mỗi test script có được thực hiện thành công như mong đợi không?)

Ghi nhận và đánh giá thời gian phản hồi đã định nghĩa bằng các báo cáo sau:

- Performance Percentile
- Response Time

*Điều kiện hoàn thành :*

Tất cả các TC có trong kế hoạch đều đã được thực hiện

Tất cả các lỗi được xác định phải được ghi nhận vào một giải pháp đã thỏa thuận (All identified defects have been addressed to an agreed upon resolution)

Tất cả các TC có trong kế hoạch đã được thực hiện lại và toàn bộ các lỗi mở đã được ghi nhận như đã thỏa thuận và không có lỗi mới nào được phát hiện

Hoặc

Toàn bộ các TC đặt mức ưu tiên cao đều đã được thực hiện

Toàn bộ các lỗi tìm thấy đều được ghi nhận vào một giải pháp đã thỏa thuận

Toàn bộ các lỗi có trọng số 1 và 2 đều được giải quyết

Tất cả các TC có mức ưu tiên cao đều đã được thực hiện lại và toàn bộ các lỗi mở đã được ghi nhận như đã thỏa thuận và không có lỗi mới nào được phát hiện

Các vấn đề đặc biệt

- Cơ sở dữ liệu test yêu cầu người thiết kế hoặc quản trị CSDL hỗ trợ để tạo mới, cập nhật và làm tươi dữ liệu test
- Việc test hiệu suất hệ thống sử dụng máy chủ trong mạng hiện tại (có hỗ trợ cả các giao dịch khác không thuộc việc test). Việc test sẽ phải được lập lịch vào những giờ không còn các giao dịch khác trên mạng.
- Mục tiêu test phải đồng nhất với hệ thống hợp lệ (hoặc giả lập đồng bộ) để việc test chức năng có thể được tiến hành và thực hiện
- Việc test có thể bị dừng khi <số lỗi vượt quá norm, ...>
- Cán bộ test có thể dừng test khi lập trình viên không thực hiện unit test, ...

>

## Test chức năng (Function Testing)

<Mục đích của test chức năng là tập trung vào các yêu cầu test có thể được lưu vết trực tiếp trong các UC hoặc các chức năng và qui tắc nghiệp vụ. Mục tiêu của kiểu test này là kiểm tra tính đúng đắn của các dữ liệu, qui trình và báo cáo cũng như việc thực hiện đúng những qui tắc nghiệp vụ. Kiểu test này dựa vào kỹ thuật black box, tức là kiểm tra ứng dụng và các xử lý nội tại bằng cách tương tác với ứng dụng thông qua giao diện người sử dụng và phân tích các kết quả hoặc đầu ra. Bảng sau liệt kê một số gợi ý đối với mỗi ứng dụng:

Mục đích test:	<Đảm bảo mục tiêu test đúng đắn của chức năng, bao gồm định hướng, dữ liệu đầu vào, xử lý và dữ liệu nhận được>
Cách thực hiện:	<Thực hiện mỗi UC, chu trình UC hoặc chức năng, sử dụng dữ liệu hợp lệ và không hợp lệ để kiểm tra:- Kết quả mong đợi với dữ liệu hợp lệ.- Lỗi thích hợp hoặc thông báo hiển thị khi dữ liệu không hợp lệ.- Mỗi qui tắc nghiệp vụ đều được áp dụng đúng>
Điều kiện hoàn thành:	- <Toàn bộ kế hoạch test đã được thực hiện.- Toàn bộ các lỗi phát hiện ra đã được ghi nhận.>
Các vấn đề	<Xác định hoặc mô tả các vấn đề (nội bộ hoặc bên ngoài) ảnh hưởng đến việc test chức năng>

đặc biệt:	
-----------	--

>

## Test giao diện người sử dụng (User Interface Testing)

<Test giao diện người dùng (UI) kiểm tra các tương tác của người dùng với phần mềm. Mục tiêu của test UI là để đảm bảo rằng giao diện người dùng cung cấp cho người sử dụng cách truy cập và sử dụng thích hợp thông qua các chức năng trong mục tiêu test. Ngoài ra, test UI còn để đảm bảo rằng các đối tượng trong phạm vi chức năng UI giống như mong đợi và phù hợp với tổ chức hoặc chuẩn ngành.>

Mục đích test:	<Kiểm tra:? Việc sử dụng thông qua mục tiêu test phản ánh đúng các chức năng và yêu cầu nghiệp vụ, bao gồm màn hình đến màn hình, trường đến trường và sử dụng các phương pháp truy cập (phím tabs, di chuột, tổ hợp phím)? Các đối tượng và thuộc tính màn hình như menus, size, position, state, và tập trung vào việc tương thích với chuẩn>
Cách thực hiện:	<Tạo ra và chỉnh sửa test cho mỗi màn hình để kiểm tra việc sử dụng đúng cách và tình trạng các đối tượng cho mỗi màn hình và đối tượng của ứng dụng>
Điều kiện hoàn thành:	<Mỗi màn hình được kiểm tra thành công đúng với phiên bản kiểm tra hoặc phạm vi chấp nhận được>
Các vấn đề đặc biệt:	<Không phải toàn bộ các thuộc tính của các đối tượng đều truy cập được>

## Test dữ liệu và tích hợp dữ liệu (Data and Database Integrity Testing)

<Cơ sở dữ liệu và xử lý cơ sở dữ liệu phải được test như một hệ thống con trong dự án. hệ thống con này phải được test không cần thông qua giao diện người dùng để giao tiếp với dữ liệu. Nghiên cứu thêm về DBMS để xác định các công cụ và kỹ thuật có thể có giúp hỗ trợ cho việc test:

Mục đích test:	<Đảm bảo rằng các phương pháp truy cập và chức năng xử lý là đúng và không có sai lệch dữ liệu>
Cách thực hiện:	? <Thực hiện từng phương pháp truy cập và xử lý, thử từng trường hợp với dữ liệu hợp lệ và không hợp lệ hoặc các yêu cầu dữ liệu.? Kiểm tra cơ sở dữ liệu để đảm bảo rằng dữ liệu được lưu trữ như mong đợi, toàn bộ các sự kiện với cơ sở dữ liệu xảy ra đều đúng, học xem xét các dữ liệu trả về để đảm bảo rằng đã nhận được dữ liệu đúng cho các lý do đúng>
Điều kiện hoàn thành:	<Tất cả các phương pháp truy cập và chức năng xử lý đều giống như thiết kế và không có sai lệch dữ liệu>
Các vấn đề đặc biệt:	? <Việc test có thể đòi hỏi phải môi trường phát triển DBMS hoặc drivers để truy cập hoặc sửa dữ liệu trực tiếp trong cơ sở dữ liệu.? Các xử lý phải được thực hiện bằng tay.? Cơ sở dữ liệu có kích thước nhỏ hoặc tối thiểu (giới hạn số bản ghi) phải được dùng để làm rõ thêm các sự kiện không được phép chấp nhận>

>

### Test chu trình nghiệp vụ (Business Cycle Testing)

<Test chu trình nghiệp vụ phải thực hiện các hoạt động trong dự án qua thời gian. Phải xác định một chu kỳ, ví dụ một năm, và các giao dịch và hoạt động có thể xảy ra trong chu kỳ của năm đó phải được thực hiện. Việc này bao gồm cả các chu kỳ hàng ngày, hàng tuần hoặc hàng tháng và các sự kiện là ảnh hưởng bởi ngày tháng, ví dụ như ứng dụng ngân hàng>

Mục đích test:	<Đảm bảo mục đích của test là đúng đắn và các tiến trình chạy ngầm thực hiện đúng yêu cầu về mô hình nghiệp vụ và lịch trình>
Cách thực hiện:	<Việc test sẽ giả lập vài chu trình nghiệp vụ bằng cách thực hiện các công việc sau:? Các test dùng cho việc test chức năng sẽ được sửa lại hoặc nâng cấp để tăng số lần mỗi chức năng được thực hiện để giả lập một số người dùng khác nhau trong chu kỳ đã định.? Toàn bộ các chức năng theo ngày tháng sẽ được thực hiện với dữ liệu hợp lệ và không hợp lệ hoặc chu kỳ thời gian? Toàn bộ các chức năng xảy ra trong lịch trình chu kỳ sẽ được thực hiện vào thời gian thích hợp? Việc test sẽ bao gồm cả dữ liệu hợp lệ và không hợp lệ để kiểm tra: - Kết quả xảy ra khi dữ liệu hợp lệ. - Lỗi tương

	tự hoặc cảnh báo hiển thị khi dữ liệu không hợp lệ.? Mỗi qui tắc nghiệp vụ đều được áp dụng.
Điều kiện hoàn thành:	? <Toàn bộ kế hoạch test đã được thực hiện.? Toàn bộ các lỗi phát hiện ra đều được ghi nhận>
Các vấn đề đặc biệt:	? <Ngày và các sự kiện của hệ thống có thể đòi hỏi các hoạt động hỗ trợ đặc biệt? Mô hình nghiệp vụ đòi hỏi xác định các yêu cầu và thủ tục test thích hợp>

## Test hiệu suất (Performance testing)

### Performance Profiling

<Performance profiling là một dạng test hiệu suất trong đó thời gian phản hồi, tỷ lệ giao dịch và các yêu cầu phụ thuộc thời gian khác được đo đạc và đánh giá. Mục đích của Performance Profiling là kiểm tra các yêu cầu về hiệu suất có đạt được hay không. Performance profiling là tiến hành và thực hiện để mô tả sơ lược và điều chỉnh các hành vi hiệu suất của mục tiêu test như một hàm của các điều kiện ví dụ workload hoặc cấu hình phần cứng.

Chú ý: Các giao dịch dưới đây tham chiếu đến “các giao dịch nghiệp vụ logic”. Các giao dịch này được định nghĩa như xác định các UC mà tác nhân của hệ thống hy vọng được thực hiện bằng cách sử dụng mục tiêu test, như thêm mới hoặc sửa một hợp đồng>

Mục đích test:	<Kiểm tra các biểu hiện về hiệu suất cho các giao dịch hoặc chức năng nghiệp vụ đã thiết kế theo những điều kiện sau:? workload bình thường đã biết trước (normal anticipated workload)? workload xấu đã biết trước (anticipated worst case workload)>
Cách thực hiện:	? <Sử dụng các thủ tục test cho test chức năng và chu trình nghiệp vụ? Chỉnh sửa file dữ liệu để tăng số lượng các giao dịch hoặc scripts để tăng số tương tác xảy ra trong mỗi giao dịch? Scripts phải được chạy trên một máy (trường hợp tốt nhất để đánh giá người dùng đơn lẻ, giao dịch đơn lẻ) và phải lặp lại trên nhiều máy trạm (ảo hoặc thực, xem các vấn đề đặc biệt dưới đây)>
Điều kiện	? <Giao dịch đơn lẻ hoặc người dùng đơn lẻ: Thực hiện thành công test script không có lỗi và trong phạm vi mong đợi hoặc thời gian phản hồi cho



hoàn thành:	mỗi giao dịch>? <Nhiều giao dịch hoặc nhiều người dùng: Thực hiện thành công test script không có lỗi và trong thời gian chấp nhận được>
Các vấn đề đặc biệt:	<Việc test hiệu suất toàn diện bao gồm phải có một workload nền trên máy chủ. Có một số phương pháp để thực hiện, bao gồm:? “Drive transactions” trực tiếp đến máy chủ, thường trong các form gọi SQL.? Tạo các người dùng ảo để giả lập nhiều máy trạm, thường là vài trăm. Sử dụng công cụ Remote Terminal Emulation để thực hiện việc load này, kỹ thuật này còn được dùng để load giao dịch trên mạng? Sử dụng nhiều người dùng, mỗi người chạy một test script để load lên hệ thống? Test hiệu suất phải được thực hiện trên máy chuyên dụng hoặc thời gian chuyên dùng. Điều đó cho phép việc tính toán được đầy đủ và chính xác. Cơ sở dữ liệu sử dụng để test hiệu suất phải có kích thước thực tế hoặc đo bằng nhau>

## Load Testing

<Load testing là một kiểu test hiệu suất mà mục tiêu là kiểm tra workload để tính toán và đánh giá hiệu suất và khả năng của mục đích test để tiếp tục thực hiện các chức năng thích hợp với các workload khác. Mục đích của load testing là xác định và đảm bảo các chức năng hệ thống thích hợp với nhiều nhất các workload. Ngoài ra load testing còn đánh giá các tính năng hiệu suất như thời gian phản hồi, tỉ lệ giao dịch và các vấn đề liên quan đến thời gian khác.>

<Chú ý: Các giao dịch dưới đây tham chiếu đến “các giao dịch nghiệp vụ logic”. Các giao dịch này được định nghĩa như các chức năng xác định mà người dùng cuối của hệ thống mong muốn thực hiện thông qua ứng dụng như thêm hoặc sửa các thông tin hợp đồng>

Mục tiêu test:	<Kiểm tra hiệu suất về thời gian cho các giao dịch hoặc tình huống nghiệp vụ đã thiết kế với nhiều điều kiện workload>
Cách thực hiện:	? <Sử dụng các test đã xây dựng cho test chức năng và chu trình nghiệp vụ.? Sửa lại file dữ liệu để tăng số lượng giao dịch hoặc test nhằm tăng thêm số lần thực hiện mỗi giao dịch>
Điều kiện hoàn thành:	<Nhiều giao dịch hoặc nhiều người dùng: Thực hiện thành công việc test không có lỗi và trong thời gian chấp nhận được>

Các vấn đề đặc biệt:	? <Load testing phải được thực hiện trên máy chuyên dụng hoặc vào những giờ chuyên biệt. Nó cho phép đo đạc đầy đủ và chính xác.? Cơ sở dữ liệu dùng cho load testing phải có kích thước thực tế hoặc đo bằng nhau>
----------------------	---

## Stress Testing

<Stress testing là một kiểu test hiệu suất được thực hiện để tìm ra các lỗi trong trường hợp thiếu tài nguyên hoặc cạnh tranh về tài nguyên. Bộ nhớ hoặc dung lượng đĩa ít có thể làm xuất hiện lỗi trong mục đích test mà nó không xuất hiện dưới điều kiện bình thường. Các lỗi khác có thể là kết quả của việc cạnh tranh hoặc chia sẻ tài nguyên như khóa cơ sở dữ liệu hoặc băng thông mạng. Stress testing cũng được dùng để xác định wordload tối đa mà mục đích test có thể điều khiển được.>

<Chú ý: Tham khảo các giao dịch dưới đây tham chiếu đến các giao dịch nghiệp vụ logic>

Mục đích test:	<Kiểm tra các chức năng của mục đích test là đúng đắn và không có lỗi với những điều kiện sau:? Có ít hoặc không có bộ nhớ phù hợp trên máy chủ (RAM và DASD)? Số lượng máy trạm tối đa trong thực tế hoặc giả lập kết nối vào máy chủ? Nhiều người dùng thực hiện cùng một giao dịch với cùng dữ liệu hoặc account? Độ lớn các giao dịch xấu hoặc hỗn hợp (xem phần Performance Testing ở trên).Chú ý: Mục đích của Stress Testing có thể được phát biểu rõ và ghi ra các điều kiện mà hệ thống có thể lỗi, không thể tiếp tục thực hiện các chức năng một cách thích hợp>
Cách thực hiện:	? <Sử dụng các test đã xây dựng để thực hiện Performance Profiling hoặc Load Testing.? Để test việc hạn chế tài nguyên, test phải chạy trên máy đơn lẻ và RAM và DASD trên máy chủ phải giảm đi hoặc hạn chế? Để thực hiện các stress tests khác phải sử dụng nhiều người dùng cùng chạy một TC hoặc bổ sung các test để thực hiện độ lớn giao dịch xấu hoặc hỗn hợp.
Điều kiện hoàn thành:	<Toàn bộ kế hoạch test được thực hiện và các hạn chế của hệ thống được xác định thỏa mãn các điều kiện tối thiểu đã đặt ra hoặc chỉ sai trong trong hợp các điều kiện không nằm trong điều kiện đã xác định>
Các vấn đề	? <Việc test Stressing mạng có thể đòi hỏi những công cụ để load mạng với nhiều thông báo hoặc gói dữ liệu.? DASD dùng cho hệ thống phải tạm thời giảm xuống để hạn chế khả năng chỗi trỗng cho tăng trưởng cơ sở dữ liệu.?

đặc biệt:	Đồng bộ hóa các máy trạm đồng thời truy cập vào cùng một bản ghi hoặc các account dữ liệu>
-----------	--

## Volume Testing

<Mục tiêu của Volume Testing là để kiểm tra giới hạn của độ lớn của dữ liệu có thể làm phần mềm bị sai. Volume Testing cũng xác định load lớn nhất liên tục hoặc độ lớn mà mục đích test có thể điều khiển được trong chu kỳ đã cho. Ví dụ, nếu mục đích test là xử lý một tập các bản ghi để tạo báo cáo, Volume Test có thể dùng một cơ sở dữ liệu test lớn và kiểm tra xem phần mềm có chạy bình thường và cho ra báo cáo đúng không>

Mục đích test:	<Kiểm tra xem mục tiêu test có thực hiện thành công các chức năng theo những điều kiện sau không?: Số máy trạm lớn nhất kết nối (thực tế hoặc vật lý – có thể), hoặc giả lập, tất cả đều thực hiện cùng một chức năng nghiệp vụ trong một chu kỳ mở rộng.? Kích thước cơ sở dữ liệu lớn nhất có thể (thực tế hoặc đo được) và nhiều query hoặc giao dịch báo cáo được thực hiện đồng thời.>
Cách thực hiện:	? <Sử dụng các test đã xây dựng cho Performance Profiling hoặc Load Testing.? Có thể dùng nhiều người dùng, chạy cùng một test hoặc bổ sung các test để thực hiện trường hợp giao dịch volume hoặc hỗn hợp xấu nhất (xem Stress Testing ở trên) trong một chu kỳ mở rộng.? Tạo ra cơ sở dữ liệu lớn nhất (thực tế, qui đổi, hoặc lọc các dữ liệu đại diện) và nhiều người dùng chạy các query và giao dịch báo cáo đồng thời trong một chu kỳ mở rộng>
Điều kiện hoàn thành:	? <Toàn bộ kế hoạch test được thực hiện và các giới hạn của hệ thống được xác định là đạt tới hoặc xử lý mà không có lỗi>
Các vấn đề đặc biệt:	<Chu kỳ thời gian như thế nào là chấp nhận được cho điều kiện cơ sở dữ liệu lớn, như đã nói ở trên?>

Test Bảo mật và Kiểm soát truy cập (Security and Access Control Testing)

<Test bảo mật và kiểm soát truy cập tập trung vào hai lĩnh vực bảo mật chính:

- Bảo mật ở mức ứng dụng, bao gồm truy cập dữ liệu và các chức năng nghiệp vụ
- Bảo mật ở mức hệ thống, bao gồm truy cập vào hệ thống hoặc truy cập từ xa

Bảo mật mức ứng dụng đảm bảo rằng, dựa trên bảo mật đã yêu cầu, người dùng bị hạn chế sử dụng một số chức năng hoặc tình huống sử dụng, hoặc bị hạn chế trong giới hạn dữ liệu phù hợp với họ. Ví dụ, mọi người có thể được phép nhập dữ liệu để tạo account nhưng chỉ có người quản lý có thể xóa chúng. Nếu là bảo mật ở mức dữ liệu, việc test đảm bảo rằng “người dùng nhóm 1” có thể nhìn thấy các thông tin khách hàng, bao gồm dữ liệu tài chính, tuy nhiên “người dùng nhóm 2” chỉ nhìn thấy các thông tin chung chung cho cùng một khách hàng.

Bảo mật mức hệ thống đảm bảo rằng chỉ những người dùng được cho quyền truy cập vào hệ thống mới có khả năng truy cập vào ứng dụng và chỉ bằng các cổng thích hợp

>

Mục đích test:	? Bảo mật mức ứng dụng: Đảm bảo rằng một người dùng chỉ có thể truy cập vào những chức năng hoặc dữ liệu mà nhóm người dùng đó được phép? Bảo mật mức hệ thống: Đảm bảo rằng chỉ những người được phép truy cập hệ thống và ứng dụng được phép truy cập chúng
Cách thực hiện:	? Bảo mật ứng dụng: Xác định và liệt kê từng nhóm người dùng và các chức năng hoặc dữ liệu mà họ được phép truy cập? Tạo test case cho mỗi nhóm người dùng và kiểm tra từng quyền bằng cách tạo các giao dịch xác định cho mỗi nhóm? Sửa lại nhóm người dùng và chạy lại tình huống test cho cùng những người dùng. Với mỗi trường hợp, kiểm tra các chức năng thêm vào hoặc dữ liệu có đúng không hay bị từ chối.? Truy cập mức hệ thống: tham khảo các điều kiện đặc biệt dưới đây
Điều kiện hoàn thành:	<Với mỗi nhóm người dùng đều có các chức năng hoặc dữ liệu thích hợp, và toàn bộ các chức năng giao dịch đều như dự kiến và chạy trong các test chức năng ứng dụng trước đó>
Các vấn đề đặc biệt:	<Truy cập vào hệ thống phải được xem xét hoặc thảo luận với quản trị hệ thống hoặc quản trị mạng, có thể không cần nếu nó là chức năng của quản trị mạng hoặc quản trị hệ thống>

Test hồi qui (Regression Testing)

<Test hồi qui là một hoạt động cần thiết để chỉ ra rằng việc thay đổi code không gây ra những ảnh hưởng bất lợi>

Mục đích test:	Test hồi qui dùng để kiểm tra các phần được sửa chữa trong phần mềm, để đảm bảo rằng những sự thay đổi đó không gây ra lỗi trong những phần khác
Cách thực hiện:	? <Tái sử dụng các TC từ những phần test trước để test các module đã được sửa chữa>.? <Sử dụng công cụ Rational Robot: Tạo một số test script về chức năng. Định nghĩa lịch thực hiện tự động cho chúng>? <80% các TC được chọn ngẫu nhiên>? <Xây dựng một chương trình phân tích sơ sở hạ tầng. Chúng ta dựng một cơ sở hạ tầng có thể mở rộng được để thực hiện và đánh giá chương trình phân tích. Dựa vào kết quả phân tích chúng ta xác định phạm vi cần test hồi qui.>
Điều kiện hoàn thành:	? <Toàn bộ các TC được thực hiện và đạt yêu cầu>? <Toàn bộ các TC được chọn được thực hiện và đạt yêu cầu>
Các vấn đề đặc biệt:	

# Test

## 3.1. Giai đoạn test

<Làm rõ trạng thái của giai đoạn sẽ thực hiện test. Bảng sau liệt kê các giai đoạn mà việc test thường được thực hiện>

Kiểu test				
Giai đoạn test				
Unit	Integration	System	Acceptance	
<Functional Tests (Function, User Interface)>	X	X	X	X
<Performance Tests (Performance profiles of individual components)>	X	X		
<Performance Tests (Load, Stress, Contention)>			X	X
<Reliability (Integrity, Structure)>	X	X		

## 3.2. Các công cụ test

<Liệt kê các công cụ sẽ áp dụng cho dự án>

Mục đích	Công cụ	Nhà cung cấp/Tự xây dựng	Phiên bản

## 3.3. Môi trường test

<Chỉ rõ môi trường sẽ xây dựng để thực hiện test qua các giai đoạn Unit test, Intergration test, System test, Acceptance test.... Với mỗi giai đoạn, hãy xác định các yếu tố để xây dựng môi trường test như thế nào, sử dụng như môi trường mà chương trình sẽ chạy thật hay tạo môi trường giả lập gần giống với môi trường chạy thật của chương trình. Các yếu tố về môi trường như:

- Khi test chạy chương trình bằng bản dịch hay chạy trên code. Thông thường, các giai đoạn System test, Acceptance test phải chạy trên bản dịch.
- Các database sẽ sử dụng độc lập hay dùng chung với database phát triển. Thông thường, từ Intergration test, nhóm test phải thiết lập database riêng và thiết lập các thông số cho database gần giống hoặc giống hệt như khi chương trình sẽ chạy thật.

- Điều kiện về mạng: sẽ sử dụng mạng LAN hay Dial up... Thông thường, khi Unit test, có thể sử dụng mạng LAN nhưng khi System test trở đi thì nên sử dụng hệ thống đường truyền giống như hoặc gần giống như môi trường chạy thật.

- Mô hình sẽ cài đặt chương trình test: số lượng máy chủ, máy trạm; việc chia tách các server, các máy trạm, việc cài đặt các domain ... Thông thường, trong Unit test có thể sử dụng việc thiết lập như khi lập trình, nhưng khi System test trở đi, phải chú ý thiết lập sao cho gần giống mô hình sẽ chạy trong thực tế nhất >.

## TÀI NGUYÊN

### 3.1. Nhân lực

Bảng sau mô tả nguồn lực test cho dự án.

Họ tên	Trách nhiệm/Ghi chú

### 3.2. Hệ thống

<Liệt kê các yêu cầu về phần cứng và phần mềm>

## CÁC MỐC KIỂM SOÁT CỦA GIAI ĐOẠN TEST (TEST MILESTONES)

Test v1.0 phải phối hợp các hoạt động test cho nguồn lực test được xác định trong phần trước. Độc lập với milestone của dự án, phải xác định để thông tin về tình trạng hoàn thành của dự án

Milestone Task	Nguồn lực	Ngày bắt đầu	Ngày kết thúc

## CÁC SẢN PHẨM

STT	Sản phẩm	Ngày bàn giao	Người bàn giao	Người nhận bàn giao
	<Test cases>			
	<Test procedures>			
	<Defect log>			
	<Defect reports>			

# **Bài 8 : Viết và theo dõi các trường hợp kiểm thử**

## **Viết và theo dõi các trường hợp kiểm thử**

### **Mục đích của việc lập các test case**

Lập kế hoạch test sẽ giúp quá trình kiểm tra phần mềm chặt chẽ hơn và tránh được rất nhiều rủi ro.

### **Tổng quan về lập các test case**

#### **Mô tả requirement bằng use case**

Khái niệm use case:

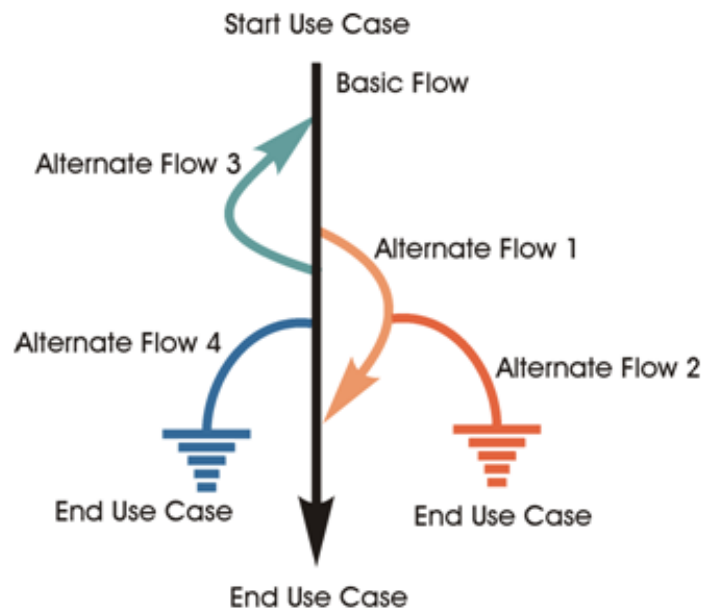
- Một Use case là tập hợp của một loạt các cảnh kịch về việc sử dụng hệ thống
- Mỗi cảnh kịch mô tả một chuỗi các sự kiện
- Mỗi một chuỗi này sẽ được kích hoạt bởi một người nào đó, một hệ thống khác hay là một phần trang thiết bị nào đó
- Những thực thể kích hoạt nên các chuỗi sự kiện như thế được gọi là các Tác Nhân (Actor)

Sự cần thiết phải có use case:

- Use Case là một công cụ xuất sắc để khuyến khích những người dùng tiềm năng nói về hệ thống từ hướng nhìn của họ
- Use case tạo nên một lời mô tả rõ ràng và nhất quán về việc hệ thống cần phải làm gì
- Có thể đơn giản hóa việc thay đổi và mở rộng hệ thống qua việc thay đổi và mở rộng mô hình Use Case, sau đó chỉ theo dõi riêng những Use Case đã bị thay đổi cùng những hiệu ứng của chúng trong thiết kế hệ thống và xây dựng hệ thống



## Mô hình luồng các sự kiện trong một use case



### Lập test cases từ use case

Các bước lập test case từ use case

- Tìm ra các “path” của các luồng sự kiện trong use case. Mỗi path như này được gọi là một scenario
- Chỉ ra điều kiện xảy ra mỗi scenario
- Mô tả điều kiện, trình tự diễn ra trong mỗi scenario và kết quả mong muốn.

### Theo dõi và quản lý các test case

Các bước chuẩn bị test

- Phân tích requirements
- Lập các kế hoạch test
- Lập testing checklist
- Mô tả các test case phức tạp
- Theo dõi các thay đổi

Phân tích dữ liệu giả định

- Lập test procedure cho nhiều test case
- Sử dụng Excel

# Bài 9 : Thực hiện test,viết báo cáo và đánh giá kết quả

## Thực hiện test, viết báo cáo và đánh giá kết quả

### Chuẩn bị môi trường

Giai đoạn chuẩn bị môi trường trên thực tế sẽ tiến hành song song với giai đoạn thiết kế các test case và các thiết kế test. Người trưởng nhóm quản lý chất lượng sẽ giao công việc chuẩn bị môi trường cho một cán bộ chuyên trách thực hiện khảo sát môi trường phần cứng cũng như phần mềm trước khi mà thực thi test. Cán bộ chuyên trách dưới sự phân công của trưởng nhóm test sẽ tiến hành khảo sát môi trường dựa vào: bản đặc tả SRS, prototype( nếu có) ... Và cuối cùng là điền kết quả khảo sát vào mẫu sau:

#### BIÊN BẢN GHI NHẬN KIỂM TRA MÔI TRƯỜNG

Máy số: .....Ngày: ...../...../.....

Người sử dụng: .....

Phòng/Đơn vị: .....

Người khảo sát: .....

MÔI TRƯỜNG PHẦN CỨNG	
1. Cấu hình máy CPU: .....RAM: ..... .....HDD: ..... .....CD- ROM: .....CD- RW: .....	2.Thiết bị khác[ ] LAN card: ..... .....
MÔI TRƯỜNG PHẦN MỀM	
1. Hệ điều hành[ ] Windows NT[ ] Windows 2000[ ] Windows 2003Loại khác: ..... .....	1. Hệ quản trị [ ] Foxpro[ ] Oracle[ ] SQL Server[ ] Acce

4. Phần mềm văn phòng/công cụ	
[ ] MS Word[ ] MS Excel[ ] MS Access[ ] MS PowerPoint[ ] MS Visio	[ ] MS Outlook[ ] Outlook Express[ ] IE[ ]
5. Môi trường lập trình[ ] JDK[ ] Jbuidar[ ] SQL Navigator[ ].....	6. Dữ liệu cần lưu trữ/thư mục gốc[ ] DOC.....
7. Các ứng dụng cài trên máy chủ PQA[ ] Công cụ quản lý lỗi[ ] .....[ ] .....[ ].....[ ].....	Ghi chú, thông tin bổ sung: .....

## Thực thi Test

### Khái niệm Test chạy

- Test chạy là gì?

- Test không có tài liệu đầu vào
- Test nhằm phát hiện và sửa lỗi được chút nào hay chút ấy

- Khi nào chấp nhận test chạy?

- Dự án quá gấp, không đủ thời gian cho nhóm phát triển lập tài liệu
- Requirements không thể được xác lập rõ ràng

- Làm gì để test chạy có hiệu quả?

- Tích cực, mau chóng trao đổi với nhóm phát triển để nắm bắt được bài toán
- Tham khảo các sản phẩm tương tự, nếu có thể.
- Thống nhất với nhóm phát triển danh sách những vùng trọng tâm cần test
- Tranh thủ tiếp xúc với khách hàng, làm sáng tỏ những nghi vấn về yêu cầu, nghiệp vụ
- Lập checklist cho việc test, nếu kịp
- Tập trung test với cường độ cao
- Thường xuyên review Mức độ khẩn cấp của các lỗi.
- Làm các báo cáo ngắn và linh hoạt về tình trạng lỗi để thúc đẩy tiến độ fix lỗi.
- Tranh thủ xây dựng và tận dụng các testing checklist sẵn có

### Test thế nào cho đủ

- Càng test càng tìm ra thêm lỗi, nhất là với các hệ thống lớn

- Vấn đề không phải là liệu tất cả các lỗi đã được tìm ra chưa, mà là liệu phần mềm đã đủ “tốt” để ngừng test hay chưa?
- Đó là một quyết định có tính “kinh tế”.
- Và là một trong những vấn đề khó nhất của testing.

#### Tìm câu trả lời:

- Khả năng tìm được thêm lỗi nếu tiếp tục test?
- Chi phí cận biên của việc đó?
- Khả năng NSD đụng phải các bug còn sót?
- Hậu quả những bug đó với NSD?

#### Kết luận:

- Không thể có câu trả lời chính xác mang tính công thức cho vấn đề trên
- Kinh nghiệm và cảm nhận cụ thể trong từng dự án, từng phần mềm vẫn là điều then chốt
- Tuy nhiên cần biết cần dành thời gian và nguồn lực cho những test nào trước, theo các cách sau

#### Ưu tiên phân bổ nguồn lực cho:

##### - Danh sách “where to focus testing”:

- Những vùng quan trọng nhất của phần mềm
- Những lỗi dễ xảy ra nhất
- Những lỗi (người dùng) dễ nhìn thấy nhất
- Những vùng phần mềm hay được dùng nhất
- Những vùng có đặc trưng riêng, khác biệt hẳn với các vùng khác của phần mềm.
- Những vùng phần mềm dễ bị ảnh hưởng nhất của các change vừa có (khi regression test).
- Những loại lỗi khó fix nhất
- Những loại lỗi mà tester biết rõ nhất
- Những loại lỗi mà tester biết mờ mờ nhất
- Positive test trước, negative test sau.
- Ưu tiên sắp xếp test theo quality dimension:
  - Số 1: thường là Function testing, và phải bao quát được business cycle của hệ thống.
  - Số 2: Usability testing, chú ý test GUI, đảm bảo đúng syntax, theo standards và user friendly.
  - Số 3: security testing, installation testing, ...
- Chọn lọc các test case hiệu quả:

- Dùng kỹ thuật Equivalence class partitioning, chỉ lấy một vài test case đại diện cho từng class là đủ.
  - Dùng kỹ thuật Domain testing, chỉ lấy một số giá trị on và off là đủ.
  - Dùng kỹ thuật Loop testing, thì chỉ một số test case cho các trường hợp đi qua vòng lặp 0 lần, 1 lần, 2 lần, x lần và *số lần tối đa  $\pm 1$*  là đủ.
  - ...- Chọn lọc các test case hiệu quả (tiếp):
  - Tính mức ưu tiên cho mỗi test case bằng cách xây dựng “Test Coverage matrix”
  - “Test Coverage matrix” map từng test case vào từng software feature
  - Cần ưu tiên những test case có liên quan đến nhiều feature nhất.
  - Thống kê hiệu quả của mỗi test case qua thời gian
  - Thu thập số liệu về số lỗi mà mỗi test case đem lại theo từng loại phần mềm, từng khoảng thời gian
  - Đào thải dần những test case đã “hao mòn hiệu quả”, tìm các test case mới thay thế.
  - Đánh giá xác suất lỗi còn tiềm ẩn:- Dựng đồ thị biểu diễn số lượng lỗi đã tìm thấy trong mỗi đơn vị thời gian
  - có thể tạo cho mỗi loại bug một đồ thị dạng này, ví dụ tạo một đồ thị dạng này cho loại lỗi về Security – High Priority)
  - Đồ thị đi xuống một cách tương đối ổn định, đạt một tần suất lỗi/đơn vị thời gian tương đối thấp là dấu hiệu tốt.
- Dựng đồ thị thống kê số lỗi tìm thấy trong mỗi phiên bản đã release của phần mềm
- Đồ thị đi xuống một cách tương đối ổn định là dấu hiệu tốt
- So sánh với mức xác suất còn lỗi sau khi release có thể chấp nhận
- Mức này được xác định một cách định tính, từ nhiều yếu tố mang tính business

## **Lưu kết quả Test trong Test Result và đưa ngay lên Bugtracker**

- Test log:

Là tài liệu ghi lại theo trật tự thời gian những sự kiện test đã được thực hiện và kết quả.

- Test Result Report:

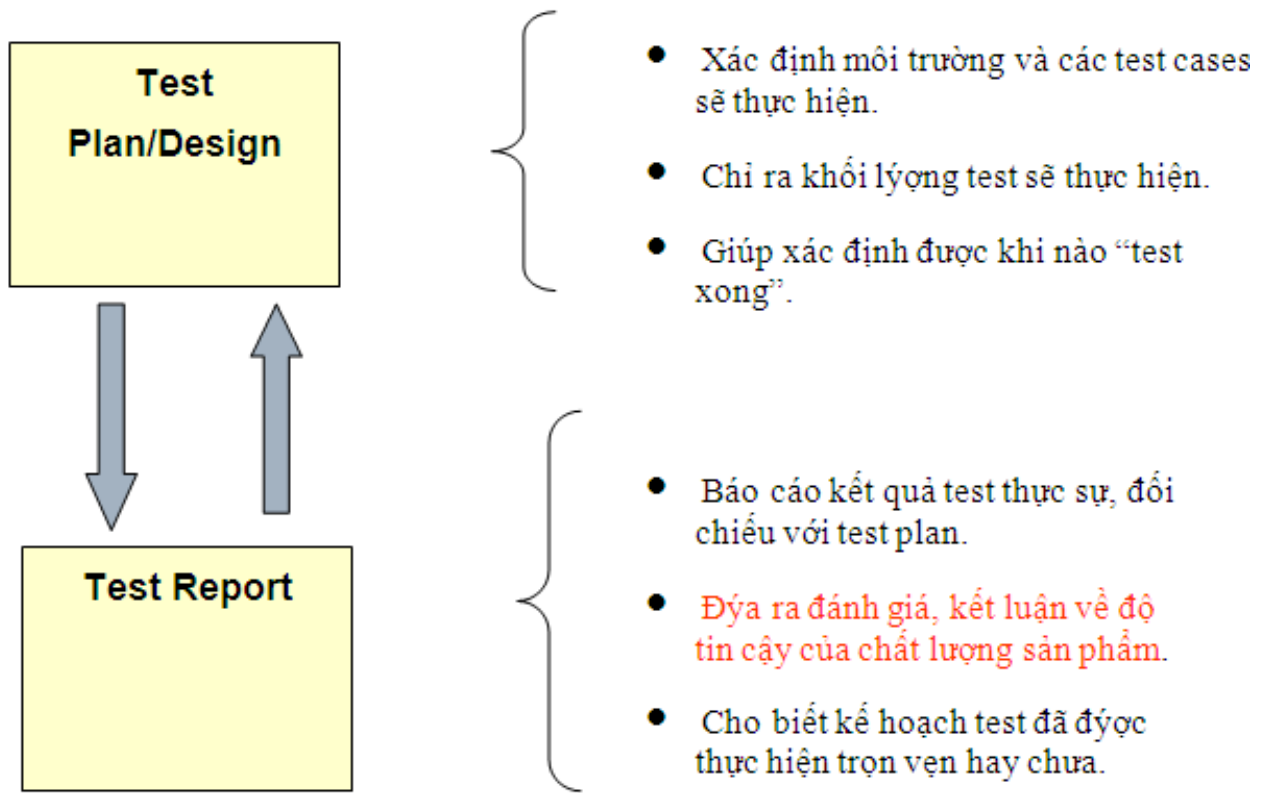
Tester

- Test Release Report:

Trong khi thực hiện test.

- Test result report là gì?
  - Là tài liệu báo cáo kết quả thực hiện test.
- Ai làm Test result report?
  - Tester
- Khi nào làm Test result report?
  - Sau khi thực hiện test xong một test item.
  - Sau khi test được một khoảng thời gian nào đó.
  - Sau khi test xong một lượt của toàn hệ thống hay một tiểu hệ thống.
  - Khi cần cung cấp thông tin cho người quản lý dự án
  - Khi Test leader yêu cầu
- Nội dung căn bản của Test result report
  - Giới thiệu tài liệu
  - Giới thiệu các test item mục tiêu
  - Các test case dự kiến thực hiện
  - Các test case đã thực hiện
  - Kết quả chi tiết của việc thực hiện từng test case
  - Thống kê tỉ lệ test case được thực hiện và tỉ lệ pass của các test case.
  - Đưa ra các change request
  - Ghi chú về các hiện tượng cần theo dõi thêm
- Test release report là gì?
  - Là tài liệu báo cáo kết quả test của sản phẩm, đánh giá chất lượng sản phẩm đã đủ tốt để release hay chưa?
- Ai làm Test release report?
  - Test leader
- Khi nào làm Test release report?
  - Khi ngừng test một bản sản phẩm
  - Khi cần kết luận sản phẩm có thể được release hay chưa?
- Nội dung căn bản của Test Release Report
  - Giới thiệu tài liệu
  - Giới thiệu hệ thống được test
  - Khái quát về kết quả test:
    - Đánh giá chung về hệ thống được test, kết luận release hoặc không.
    - Đánh giá ảnh hưởng của môi trường kiểm tra
    - Các nhận xét khuyến nghị
  - Kết quả test chi tiết
  - Các phụ lục: thống kê số lỗi, số test case được thực hiện,...

## Xử lý các vấn đề phát sinh trong quá trình Test





# Bài 10 : Kiểm thử tự động và các công cụ kiểm thử

## Lợi ích của quá trình tự động hóa và các công cụ

Ngày nay tự động hóa được ứng dụng ở rất nhiều lĩnh vực, mục đích thường rất đa dạng và tùy theo nhu cầu đặc thù của từng lĩnh vực, tuy nhiên điểm chung nhất vẫn là giảm nhân lực, thời gian và sai sót. Ngành CNTT mà cụ thể là phát triển phần mềm (PTPM) cũng không ngoại lệ. Như chúng ta biết, để tạo ra sản phẩm CNTT hay PM có chất lượng thì hoạt động kiểm tra phần mềm (KTPM) đóng vai trò rất quan trọng, trong khi đó hoạt động này lại tiêu tốn và chiếm tỷ trọng khá lớn công sức và thời gian trong một dự án. Do vậy, nhu cầu tự động hoá qui trình KTPM cũng được đặt ra. Qua thực tế cho thấy việc áp dụng kiểm tra tự động (KTTĐ) hợp lý sẽ mang lại thành công cho hoạt động KTPM. KTTĐ giúp giảm bớt công sức thực hiện, tăng độ tin cậy, giảm sự nhầm lẫn và rèn luyện kỹ năng lập trình cho kiểm tra viên (KTV). Bài viết này sẽ giới thiệu các khái niệm cơ bản của KTTĐ, đồng thời giới thiệu một công cụ KTTĐ khá mạnh hiện nay là QuickTest Professional 8.2 (QTP) của Mercury. TẠI SAO PHẢI DÙNG TEST TOOL Test Tool (TT) trong lĩnh vực PTPM là công cụ giúp thực hiện việc kiểm tra PM một cách tự động. Tuy nhiên không phải mọi việc kiểm tra đều có thể tự động hóa, câu hỏi đặt ra là trong điều kiện hoặc tình huống nào dùng TT là thích hợp? Việc dùng TT thường được xem xét trong một số tình huống sau: 1. Không đủ tài nguyên Khi số lượng tình huống kiểm tra (test case) quá nhiều mà các KTV không thể hoàn tất bằng tay trong thời gian cụ thể nào đó. Có thể lấy một dẫn chứng là khi thực hiện kiểm tra chức năng của một website. Website này sẽ được kiểm tra với 6 môi trường gồm 3 trình duyệt và 2 hệ điều hành Tình huống này đòi hỏi số lần kiểm tra tăng lên và lặp lại 6 lần so với việc kiểm tra cho một môi trường cụ thể. 2. Kiểm tra hồi qui Trong quá trình PTPM, nhóm lập trình thường đưa ra nhiều phiên bản PM liên tiếp để kiểm tra. Thực tế cho thấy việc đưa ra các phiên bản PM có thể là hàng ngày, mỗi phiên bản bao gồm những tính năng mới, hoặc tính năng cũ được sửa lỗi hay nâng cấp. Việc bổ sung hoặc sửa lỗi code cho những tính năng ở phiên bản mới có thể làm cho những tính năng khác đã kiểm tra tốt chạy sai mặc dù phần code của nó không hề chỉnh sửa. Để khắc phục điều này, đối với từng phiên bản, KTV không chỉ kiểm tra chức năng mới hoặc được sửa, mà phải kiểm tra lại tất cả những tính năng đã kiểm tra tốt trước đó. Điều này khó khả thi về mặt thời gian nếu kiểm tra thủ công.

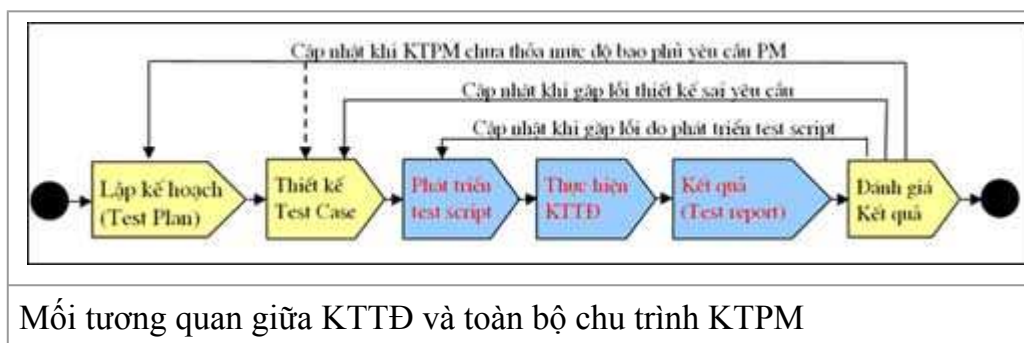
	Trình duyệt: IE, Netscape, Opera		
Hệ điều hành: WinXP, Linux	WinXP, IE	WinXP, Netscape	WinXP, Opera
	Linux, IE	Linux, Netscape	Linux, Opera

3. Kiểm tra khả năng vận hành PM trong môi trường đặt biệt Đây là kiểm tra nhằm đánh giá xem vận hành của PM có thỏa mãn yêu cầu đặt ra hay không. Thông qua đó KTV có thể xác định được các yếu tố về phần cứng, phần mềm ảnh hưởng đến khả năng vận hành của PM. Có thể liệt kê một số tình huống kiểm tra tiêu biểu thuộc loại này như sau:

- Đo tốc độ trung bình xử lý một yêu cầu của web server.
- Thiết lập 1000 yêu cầu, đồng thời gửi đến web server, kiểm tra tình huống 1000 người dùng truy xuất web cùng lúc.
- Xác định số yêu cầu tối đa được xử lý bởi web server hoặc xác định cấu hình máy thấp nhất mà tốc độ xử lý của PM vẫn có thể hoạt động ở mức cho phép.

Việc kiểm tra thủ công cho những tình huống trên là cực khó, thậm chí "vô phương". Cần lưu ý là hoạt động KTTĐ nhằm mục đích kiểm tra, phát hiện những lỗi của PM trong những trường hợp đoán trước. Điều này cũng có nghĩa là nó thường được thực hiện sau khi đã thiết kế xong các tình huống (test case). Tuy nhiên, như đã nói, không phải mọi trường hợp kiểm tra đều có thể hoặc cần thiết phải tự động hóa, trong tất cả test case thì KTV phải đánh giá và chọn ra những test case nào phù hợp hoặc cần thiết để áp dụng KTTĐ dựa trên những tiêu chí đã đề cập bên trên.

**KHÁI QUÁT VỀ KTTĐ** Việc phát triển KTTĐ cũng tuân theo các bước PTPM, chúng ta phải xem việc phát triển KTTĐ giống như phát triển một dự án. Bạn đọc có thể tham khảo bài viết về kiểm tra phần mềm trên TGV.T A tháng 12/2005 (ID: A0512\_110). Hình 1 cho chúng ta thấy mối tương quan giữa KTTĐ và toàn bộ chu trình KTPM.



Giống như PTPM, để thành công trong KTTĐ chúng ta nên thực hiện các bước cơ bản sau:

- Thu thập các đặc tả yêu cầu hoặc test case; lựa chọn những phần cần thực hiện KTTĐ.
- Phân tích và thiết kế mô hình phát triển KTTĐ.
- Phát triển lệnh đặc tả (script) cho KTTĐ.
- Kiểm tra và theo dõi lỗi trong script của KTTĐ.

Bảng sau mô tả rõ hơn các bước thực hiện KTTĐ:

STT	Bước thực hiện	Mô tả
1	Tạo test script	Giai đoạn này chúng ta sẽ dùng test tool để ghi lại các thao tác lên PM cần kiểm tra và tự động sinh ra test script.
2	Chỉnh sửa test script	Chỉnh sửa để test script thực hiện kiểm tra theo đúng yêu cầu đặt ra, cụ thể là làm theo test case cần thực hiện.
3	Chạy test script để KTTĐ	Giám sát hoạt động kiểm tra PM của test script.
4	Đánh giá kết quả	Kiểm tra kết quả thông báo sau khi thực hiện KTTĐ. Sau đó bổ sung, chỉnh sửa những sai sót.

KTTĐ có một số thuận lợi và khó khăn cơ bản khi áp dụng:

Thuận lợi	Khó khăn
<ul style="list-style-type: none"> <li>• KTPM không cần can thiệp của KTV.</li> <li>• Giảm chi phí khi thực hiện kiểm tra số lượng lớn test case hoặc test case lặp lại nhiều lần.</li> <li>• Giả lập tình huống khó có thể thực hiện bằng tay.</li> </ul>	<ul style="list-style-type: none"> <li>• Mất chi phí tạo các script để thực hiện KTTĐ.</li> <li>• Tốn chi phí dành cho bảo trì các script.</li> <li>• Đòi hỏi KTV phải có kỹ năng tạo script KTTĐ.</li> <li>• Không áp dụng được trong việc tìm lỗi mới của PM.</li> </ul>

# Một số công cụ kiểm thử

## Giới thiệu về công cụ KTTĐ

Trong lĩnh vực KTTĐ hiện có khá nhiều TT thương mại nổi tiếng, phổ biến như QuickTest Professional, WinRunner, Rational Robot, SilkTest, JTest,... Trong số đó, QuickTest Professional (QTP) phiên bản 8.2 của hãng Mercury khá tốt và mạnh, bao gồm nhiều chức năng điển hình của một công cụ kiểm tra tự động. Lưu ý là QTP 8.2 đã có một cái tên mới hơn là Mercury Functional Testing 8.2. QTP là TT dùng để kiểm tra chức năng (functional test) và cho phép thực hiện kiểm tra hồi qui (regression test) một cách tự động. Đây cũng là công cụ áp dụng phương pháp Keyword-Driven, một kỹ thuật scripting (lập trình trong KTTĐ) hiện đại, cho phép KTV bổ sung test case bằng cách tạo file mô tả cho nó mà không cần phải chỉnh sửa hay bổ sung bất cứ script nào cả. Nó cũng phù hợp trong tình huống chuyển giao công việc mà người mới tiếp nhận chưa có thời gian hoặc không hiểu script vẫn có thể thực hiện kiểm tra PM theo đúng yêu cầu.

1. Loại phần mềm hỗ trợ QTP giúp chúng ta KTPM theo hướng chức năng trên rất nhiều loại chương trình phần mềm khác nhau. Tuy nhiên Mercury chỉ hỗ trợ sẵn một số loại chương trình thông dụng như:

- Ứng dụng Windows chuẩn/Win32.
- Ứng dụng web theo chuẩn HTML, XML chạy trong trình duyệt Internet Explorer, Netscape hoặc AOL.
- Visual Basic.
- ActiveX.
- QTP hỗ trợ Unicode (UTF-8, UTF-16).

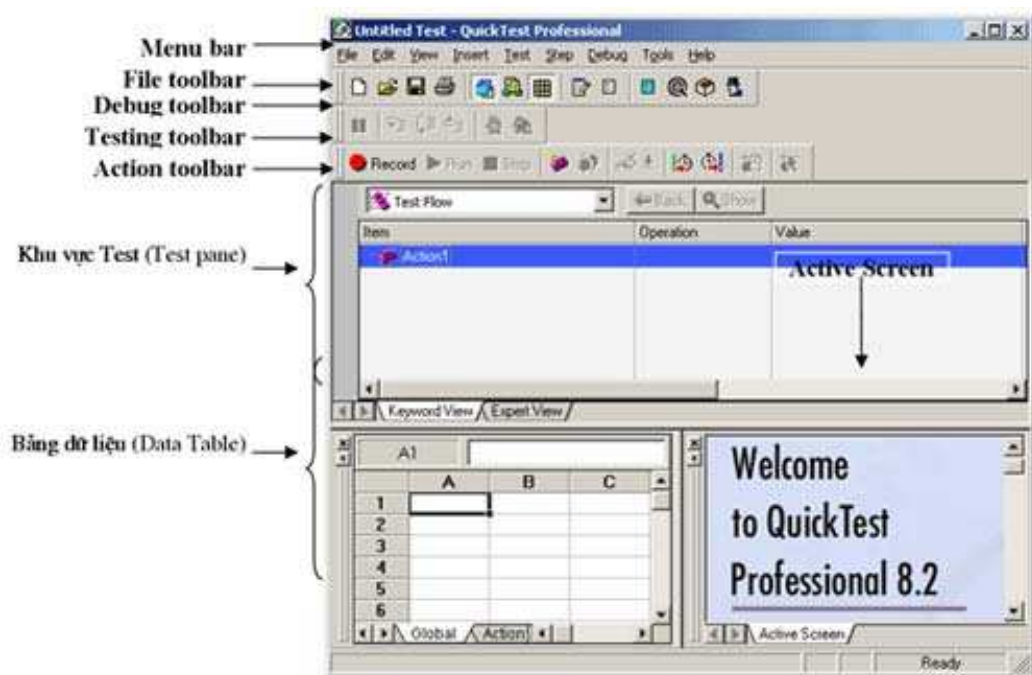
Một số loại chương trình khác đòi hỏi chúng ta phải cài đặt thêm thành phần bổ sung của QTP thì mới thực hiện kiểm tra được. Các loại chương trình đó là:

<b>.NET</b>	<ul style="list-style-type: none"> <li>• NET Framework 1.0, 1.1, 2.0 beta</li> <li>• Các đối tượng chuẩn của .NET và các đối tượng khác thừa kế từ các đối tượng chuẩn.</li> </ul>
<b>Java</b>	<ul style="list-style-type: none"> <li>• Sun JDK 1.1 – 1.4.2</li> <li>• IBM JDK 1.2 – 1.4</li> </ul>
<b>Oracle</b>	<ul style="list-style-type: none"> <li>• Oracle Applications 11.5.7, 11.5.8, 11.5.9</li> </ul>
<b>People Soft</b>	<ul style="list-style-type: none"> <li>• PeopleSoft Enterprise 8.0 – 8.8</li> </ul>
<b>SAP</b>	<ul style="list-style-type: none"> <li>• SAP GUI HMTL 4.6D, 6.10, 6.20</li> <li>• SAP Workplace 2.11</li> <li>• SAP Enterprise Portal 5.0</li> </ul>
<b>Siebel</b>	<ul style="list-style-type: none"> <li>• Siebel 7.0, 7.5, 7.7</li> </ul>
<b>Terminal Emulators</b>	<ul style="list-style-type: none"> <li>• Attachmate EXTRA! 6.7, 7.1</li> <li>• Attachmate EXTRA! Terminal Viewer 3.1 Java sessions</li> <li>• IBM Personal Communications</li> <li>• ...</li> </ul>

2. Đặc điểm

- Dễ sử dụng, bảo trì, tạo test script nhanh. Cung cấp dữ liệu kiểm tra rõ ràng và dễ hiểu.
- Kiểm tra phiên bản mới của ứng dụng với rất ít sự thay đổi. Ví dụ khi ứng dụng thay đổi nút tên "Login" thành "Đăng nhập", thì chỉ cần cập nhật lại Object Repository (OR - được giải thích ở phần sau) để QTP nhận ra sự thay đổi đó mà không cần thay đổi bất cứ test script nào.
- Hỗ trợ làm việc theo nhóm thông qua sự chia sẻ thư viện, thống nhất quản lý Object Repository.
- Thực tế cho thấy, QTP thực hiện KTTĐ trên nhiều trình duyệt cùng lúc tốt hơn những TT khác.
- Với chức năng Recovery Scenarios, QTP cho phép xử lý những sự kiện hoặc lỗi không thể đoán trước có thể làm script bị dừng trong khi đang chạy.
- QTP có khả năng hiểu test script của Mercury Winrunner (một công cụ kiểm tra khác của Mercury). Đặc biệt phiên bản v.8.2 có một số tính năng mới nổi bật:

<b>Quản trị Object Repository</b>	<ul style="list-style-type: none"> <li>• Phối hợp giữa các KTV qua việc đồng bộ hóa dữ liệu, khả năng trộn, nhập/xuất ra file XML</li> </ul>
<b>Thư viện hàm mới</b>	<ul style="list-style-type: none"> <li>• Chia sẻ các thư viện hàm giữa các nhóm KTV</li> </ul>
<b>Kiểm tra tài nguyên</b>	<ul style="list-style-type: none"> <li>• Kiểm tra tài nguyên cần thiết trước khi thực thi lệnh kiểm tra tự động.</li> </ul>
<b>Nâng cấp khả năng kéo thả</b>	<ul style="list-style-type: none"> <li>• Kéo thả các bước kiểm tra trong môi trường ngôn ngữ tự nhiên.</li> </ul>
<b>Hỗ trợ XML cho báo cáo</b>	<ul style="list-style-type: none"> <li>• Lưu trữ kết quả kiểm tra dưới dạng XML, HTML, từ đó cho phép tùy biến báo cáo.</li> </ul>
<b>Trình phát triển mới (IDE)</b>	<ul style="list-style-type: none"> <li>• Môi trường soạn thảo mới, mềm dẻo cho tùy biến và sử dụng.</li> </ul>
<b>Trình dò lỗi mới</b>	<ul style="list-style-type: none"> <li>• Cho phép KTV kiểm soát lỗi khi viết test case.</li> </ul>



Khu vực	Chức năng
Menu bar	Cấu hình thao tác với QTP và script
File toolbar	Hỗ trợ quản lý script
Debug toolbar	Hỗ trợ kiểm tra lỗi trong test script (debug)
Testing toolbar	Hỗ trợ quá trình tạo test script hoặc thực hiện KTTĐ
Action toolbar	Xem một Action (thủ tục, hàm) hoặc toàn bộ chu trình của test script
Test pane	Soạn thảo script ở một trong 2 chế độ Keyword View hoặc Expert View
Data Table	Nơi lưu trữ dữ liệu cho test script
Active Screen	Xem lại giao diện PM được kiểm tra

3. Các thành phần quan trọng trong QTP

a. Action: Giống như thủ tục hay hàm trong các ngôn ngữ lập trình khác, Action ghi lại các bước thực hiện KTTĐ và nó có thể được sử dụng lại nhiều lần. Trong một test script có thể có nhiều Action.

b. DataTable: Nơi lưu trữ dữ liệu phục vụ cho KTTĐ. Một test script sẽ có một DataTable được dùng chung cho tất cả các Action. Bên cạnh đó mỗi Action cũng có một DataTable cho riêng mình.

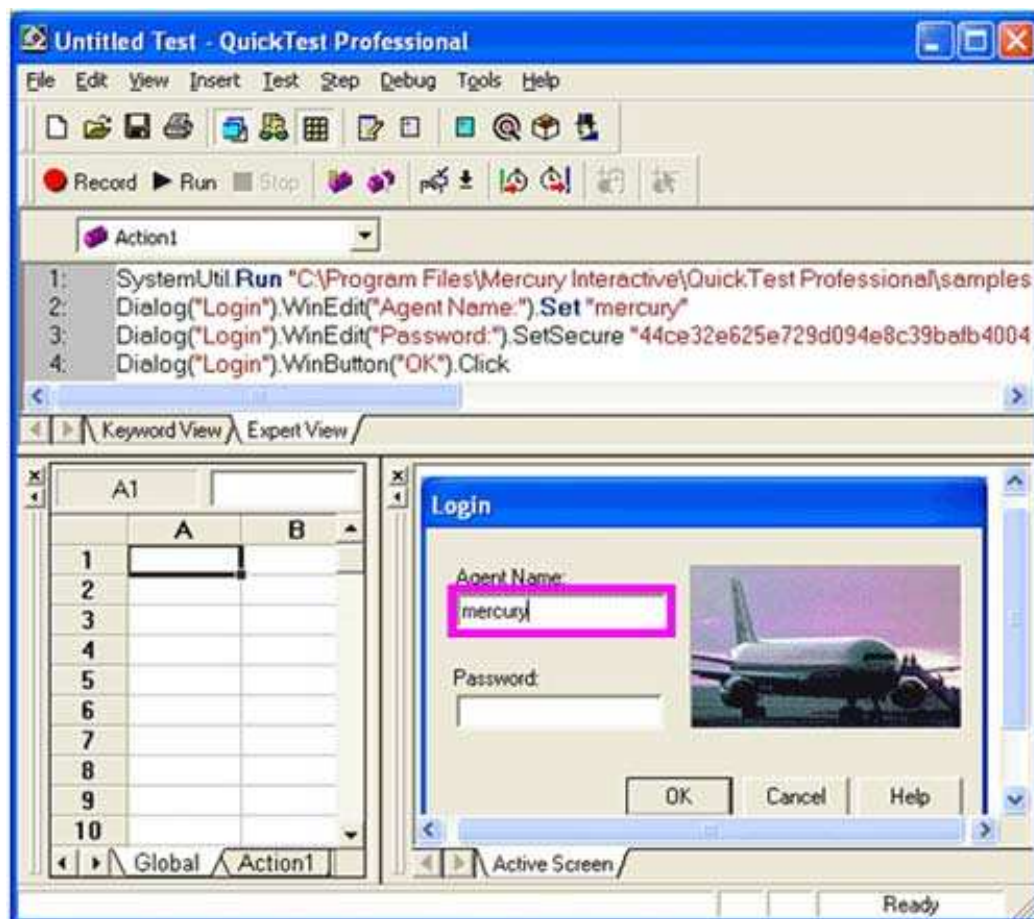
c. Object Repository (OR): Cấu trúc theo dạng cây, mô tả các đối tượng trong PM được kiểm tra. Đây được xem là cầu nối để test script tương tác với PM được kiểm tra. Khi ra lệnh cho QTP ghi lại thao tác người dùng lên PM thì trong OR sẽ tự động phát sinh thành phần đại diện cho những đối tượng trên PM vừa được thao tác. OR có thể tổ chức thành 2 loại, một loại dùng chung trong nhiều test script, loại khác dùng theo từng Action. Để xem OR, chọn menu Tools > Object Repository.

d. Checkpoint: Có thể hiểu là nơi kiểm tra trong test script, khi chạy nó sẽ thực hiện so sánh kết quả thực tế khi kiểm tra PM với kết quả mong đợi. Sau khi tiến hành so sánh QTP sẽ tự động ghi lại kết quả vào Test Results (nơi lưu kết quả khi chạy test script).

4. Ngôn ngữ sử dụng viết script QTP sử dụng ngôn ngữ VBScript để viết test script. Đây là ngôn ngữ dễ học; rất giống ngôn ngữ VBA. Chế độ Expert View của QTP là chế độ soạn thảo dành cho VBScript. Ngoài việc



dùng VBScript để tương tác với PM được kiểm tra, QTP còn có khả năng cấu hình hệ thống bằng ngôn ngữ Windows Script. Chi tiết về ngôn ngữ VBScript, người đọc có thể dễ dàng tìm trong các sách hiện có trên thị trường, thậm chí ngay chính trong phần help của QTP.



1. Sử Dụng QTP a. Yêu cầu cấu hình hệ thống: b. Bản quyền sử dụng:



CPU	PIII trở lên
Hệ điều hành	Windows 2000 SP3, SP4; Windows XP SP1, SP2 hoặc Windows 2003 Server
RAM	256 MB trở lên
Dung lượng đĩa	Tối thiểu 250MB cho ứng dụng, 120MB trên ổ đĩa hệ điều hành. Sau khi cài QTP, dung lượng cần thiết thêm trên ổ đĩa cài hệ điều hành là 150 MB
Trình duyệt	IE 5.5 SP 2 trở lên

• Bạn có thể vào <http://www.mercury.com> để đăng ký và tải về bản dùng thử trong 14 ngày. Các bước cài đặt theo sự hướng dẫn của chương trình. Sau thời gian dùng thử, để có thể tiếp tục sử dụng QTP chúng ta cần phải mua bản quyền, giá tham khảo từ nhà cung cấp như sau: cho một máy 9.000 USD; cho nhiều máy dùng cùng lúc 12.000 USD. Tại Việt Nam, nếu có nhu cầu, bạn có thể liên hệ để mua bản quyền của QTP tại công ty Tân Đức (TD&T, 103 Pasteur, Q.1, TP.HCM).

6. Lời kết Với nhiều chức năng ưu việt như đã đề cập bên trên, QTP là một Test Tool mạnh mẽ có khả năng hỗ trợ đắc lực cho KTV. Việc ứng dụng nó hợp lý chắc chắn sẽ giúp giảm công sức của KTV đồng thời làm tăng chất lượng PM. Trong điều kiện Việt Nam hiện nay, với tỷ trọng gia công phần mềm ngày càng lớn tại các công ty phần mềm, người viết cho rằng kiểm tra phần mềm tự động với những công cụ như QTP rất đáng để các doanh nghiệp phần mềm quan tâm nghiên cứu, đầu tư và ứng dụng.

## Chương Trình TestTrack Pro Version 6.0.3

### 1. Những bước cơ bản để sử dụng

-Để khởi động TestTrackPro Client: **Start→Program files→Seapine software→TestTrackPro→TestTrackPro Client.**

-Chọn server cần connect, sau đó ấn vào Connect.

-Khi đã connect được với server, NSD cần chọn Database cho dự án, nhập tên (User Name) và Mật khẩu (password) ấn OK.

ĐỐI VỚI ADMIN:

### I. Tạo và thiết lập thông số cho Database

-Ứng với một dự án cần phải có một database, vào **TestTrackPro Server Admin → Databases**, chọn Create Database để tạo mới từ đầu, hoặc chọn Create database from template để tạo dựa trên một database đã tồn tại.

-Cài đặt cấu hình cho data: **Edit → Option → Database option.**

+**General**: Các lựa chọn chung, gồm: Project Name (Tên dự án), Description (mô tả)..

+**Defect**: Các lựa chọn cho những thiết sót (việc đánh số thứ tự, ghi vào history..)

+**Mail**: Các lựa chọn cho việc gửi mail qua lại các thành viên của dự án.

+**Relationships**: Các quan hệ.

Và một số lựa chọn khác.

### 1. Quản lý User và phân quyền cho các nhóm

+**View → User**: Dùng các chức năng Add, Edit,.. để thêm, sửa thông tin các User.

+**View → User Groups**: Dùng các chức năng Add, Delete: Để thêm, xóa một nhóm.

Chọn nhóm cần thay đổi, ấn vào **Edit**:

-Users: Quản lý các user thuộc nhóm.

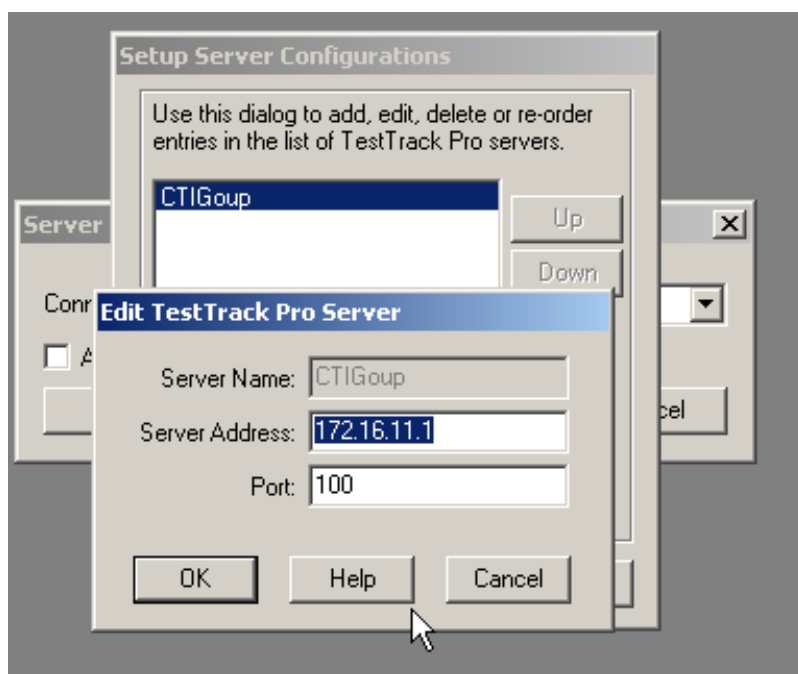
-Command/Defect/Fields security: Dùng để bảo mật và phân quyền cho các nhóm. Mặc định tất cả các User thuộc nhóm sẽ có quyền ngang bằng nhóm

a. Đối với các USER

Kết nối với Server

1 Tạo các thông số kết nối:

-File→New Server Connection: Ấn vào setup, nếu muốn thêm server chọn Add, muốn sửa thì chọn tên server → Edit. Sau đó nhập Tên server, địa chỉ IP máy server, Port (mặc định là 99). Tất cả các tham số này phải trùng với giá trị do Admin đặt cho TestTrackPro server.



2 Thực hiện kết nối:

-File → New Server connection: Chọn tên server cần kết nối, sau đó ấn vào **Connect**. Nếu connect thành công sẽ có bảng chọn Database: Welcome to TestTrackPro.

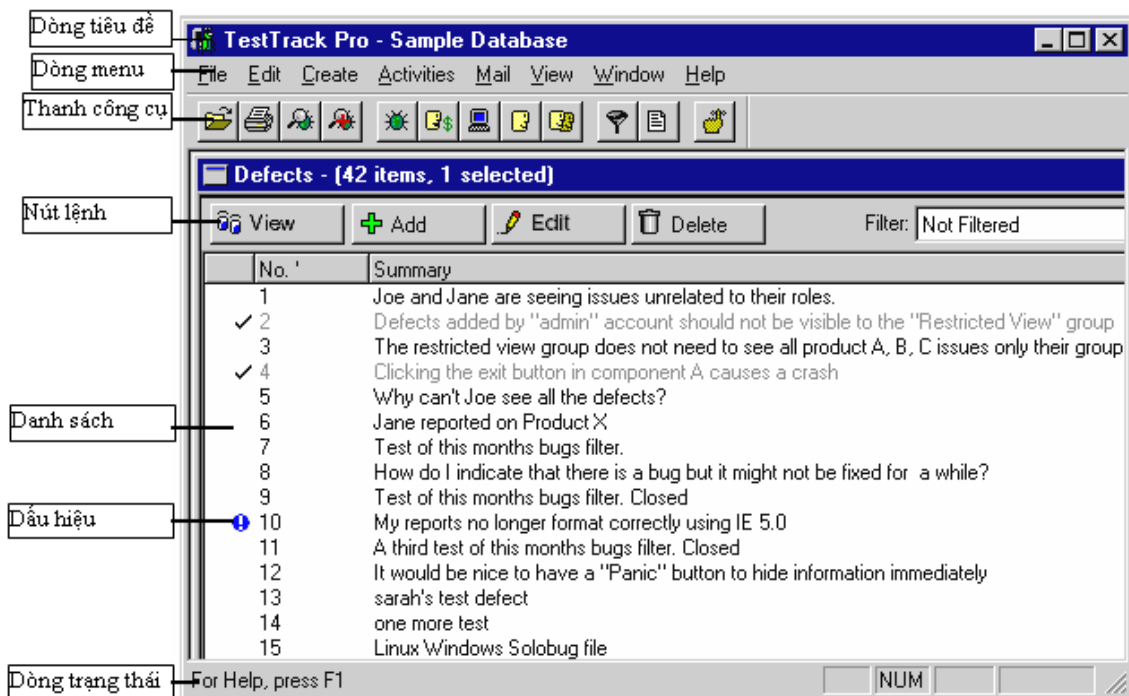


### 3 Mở database để làm việc:

-Từ bảng Welcome, hoặc chọn File → Open Database: Chọn Database cần mở, nhập tên và Password của User, ấn OK.

-Trong quá trình làm việc, khi nào ta muốn thay đổi Database( tương ứng với thay đổi dự án) thì lại thực hiện như trên.

*Sau khi đã mở được database, giao diện của TestTrackPro Client như sau:*



## II. Làm việc với DEFECT (Thiếu sót, nhược điểm)

**Defect** dùng để gỡ rối, nó bao gồm các thiếu sót, nhược điểm, các yêu cầu, các câu hỏi hoặc bất kỳ một vấn đề gì đó mà ta muốn lưu lại vết và giải quyết.

1. Mở cửa sổ danh sách các defect:

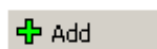
Vào menu **View** → **Defect**. hoặc chọn nút defect



trên thanh công cụ.

2. Thêm defect mới:

Chọn nút Add



trên cửa sổ danh sách các defect.

+**Summary**: Tên vấn tắt của defect.

+**Type**: Kiểu/tình trạng defect (ví dụ như: Lỗi nặng, lỗi nhẹ, đề nghị, yêu cầu...).

+**Product**: Xuất hiện trên sản phẩm/module nào?

+**Reference**: Tham khảo.

+**Entered by**: Tên User đã thêm defect vào.

+**Disposition**: (Sắp xếp theo nhu cầu)

+**Priority**: (Theo phiên bản)

+**Component**: (= module)

+**Severity**: (Lỗi hiểm)

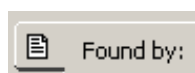
+**Date**:

## 2.a Tab Detail :

Trong tab Detail có chứa một số **mini tab**, là các tab nằm dọc theo màn hình, như hình dưới:



*\*Mini tab Found by:*



-**Found by**: Tên người đã tìm thấy defect.

-**Date**: Ngày tìm thấy

-**Version**: Phiên bản của project có defect này

-**Description**: Mô tả chi tiết defect.

Nếu thêm tiếp, hoặc những lần sau User muốn thêm vào thì chọn nút **New**.

*\*Mini tab reprocedure:*



Mô tả các bước khắc phục. Thông tin ở đây giúp cho các nhà lập trình sửa lỗi và người test kiểm tra kỹ lưỡng thiếu sót này.

Tương tự như trên ta có thể chọn nút **New**.

*\*Mini tab Computer config:*



-Lưu lại cấu hình máy, và trạng thái hệ thống lúc phát hiện ra các thiếu sót này.

*\*Mini tab Attach:*



-Cho phép đính kèm các file dữ liệu để cung cấp thêm thông tin. Để thêm ấn **Attach**.

2.b Tab Workflow: Các giải pháp được đưa ra.

2.c Tab WorkAround: Mô tả cách tiến hành

2.d Tab SourceCode: Tab này chỉ dùng với SCC (nói sau).

2.e Tab Notify: Dùng để thông báo tự động khi có sự chỉnh sửa defect, sử dụng Add/ Remove để thêm/ bớt số user sẽ nhận mail tự động khi có sự thay đổi trên defect này.

2.f Tab History: Bỏ qua tab này.

Sau khi nhập xong số liệu, chọn nút **OK** để cập nhật defect này vào danh sách.

Cấu hình các Column trong Testrac Pro

- Để xem các các cột tùy theo thông tin yêu cầu

Number	Summary '	Cc
6	Chưa load kênh từ database trong phần thiết lập cây dịch vụ	Qu
9	Kiểm tra việc chương trình bị lock	Bit
1	Lỗi font giao diện	Ba
2	Lỗi Xóa(delete) một nhánh trên cây	Ba
8	Phản Back phím âm thanh #	Bit
5	Sửa lại lỗi font load từ Database	DE
3	Thêm bot số Trung, Khoa, Khoi trong môi trường chưa hoàn chỉnh	Ba
✓ 4	Thiếu file Wave	Qu
7	Thông báo chi tiết trong cấu hình hoạt động.	Qu

- Click phải chuột

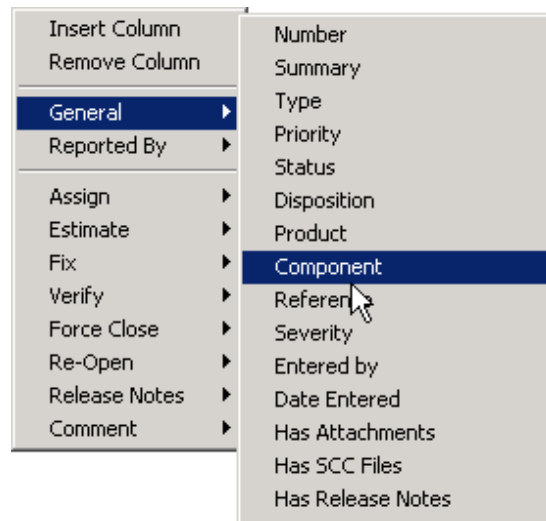
Number	Summary '	Component
6	Insert Column	tabase trong phần thiết lập cây dịch vụ
9	Remove Column	trình bị lock
1		Bao điểm thi
2	General	Bao điểm thi
8	Reported By	Bình chọn từ xa (Tele
5		Database
3	Assign	Khoa, Khoi trong môi trường chưa hoàn chỉnh
✓ 4	Estimate	Qua tang am nhac
7	Fix	Qua tang am nhac
	Verify	
	Force Close	
	Re-Open	
	Release Notes	
	Comment	

- Sẽ xuất hiện một cột trống

<not set>	Number	Summary '	Component
	6	Chưa load kênh từ database trong phần thiết lập cây dịch vụ	Qua tang am nhac
	9	Kiểm tra việc chương trình bị lock	Bình chọn từ xa (TeleVoting)
	1	Lỗi font giao diện	Bao điểm thi
	2	Lỗi Xóa(delete) một nhánh trên cây	Bao điểm thi
	8	Phản Back phím âm thanh #	Bình chọn từ xa (TeleVoting)
	5	Sửa lại lỗi font load từ Database	DBCommunication
	3	Thêm bot số Trung, Khoa, Khoi trong môi trường chưa hoàn chỉnh	Bao điểm thi
✓	4	Thiếu file Wave	Qua tang am nhac
	7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhac

- Chọn thông tin muốn hiển thị





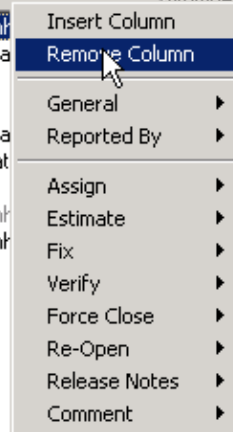
- Kết quả, sẽ hiện ra thông tin cột muốn xem:

Component	Number	Summary
Qua tang am nhac	6	Chua load kênh tu database trong phan thiet lap cay dich vu
Binh chon tu xa (TeleVoting)	9	Kiem tra viec chuong trinh bi lock
Bao diem thi	1	Loi font giao dien
Bao diem thi	2	Loi Xoa(delete) mot nhanh tren cay
Binh chon tu xa (TeleVoting)	8	Phan Back phi am phim #
DBCommunication	5	Sua lai loi font load tu Database
Bao diem thi	3	Them bot so Trung, Khoa, Khoi trong moi truong chua hoan chinh
✓ Qua tang am nhac	4	Thieu file Wave
Qua tang am nhac	7	Thong bao chi tiet trong cau hinh hoat dong.

- Tương tự như vậy ta có thể lọc được rất nhiều các thông tin muốn xem.

- Xóa cột, click chuột phải chọn Remove Column

Component	Number	Summary
Qua tang am n		Chua load kênh tu database trong phân thi
Bình chọn tu xa		Kiem tra viec chuong trinh bi lock
Bao diem thi		Loi font giao dien
Bao diem thi		Loi Xoa(delete) mot nhanh tren cay
Bình chọn tu xa		Phan Back phi am phim #
DBCommunicat		Sua lai loi font load tu Database
Bao diem thi		Them bot so Truong, Khoa, Khoi trong moi tr
✓ Qua tang am n		Thieu file Wave
Qua tang am n		Thong bao chi tiet trong cau hinh hoat dong

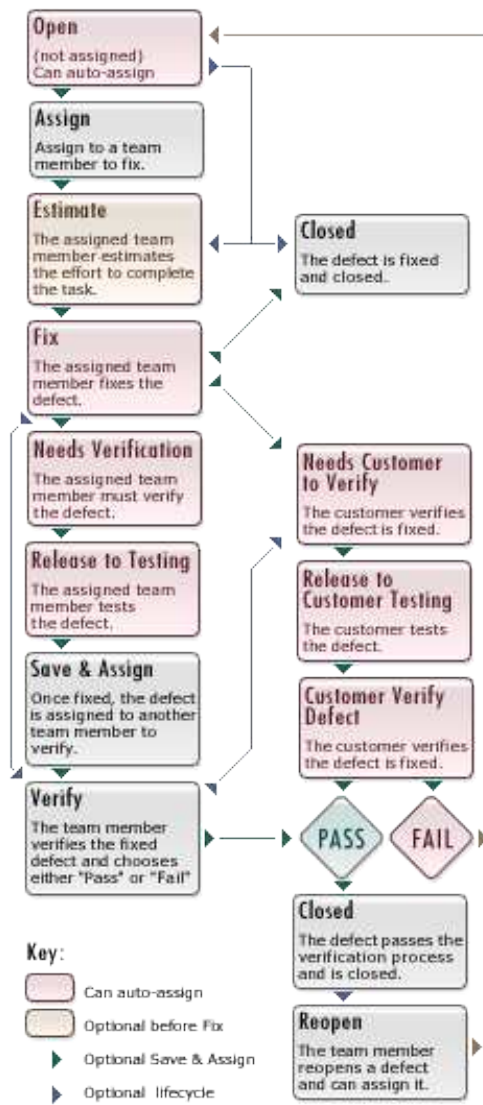


Làm việc với các sự kiện defects

- Activity ->... hoặc click phải chuột

Nó hoạt động dựa trên mô hình sau:

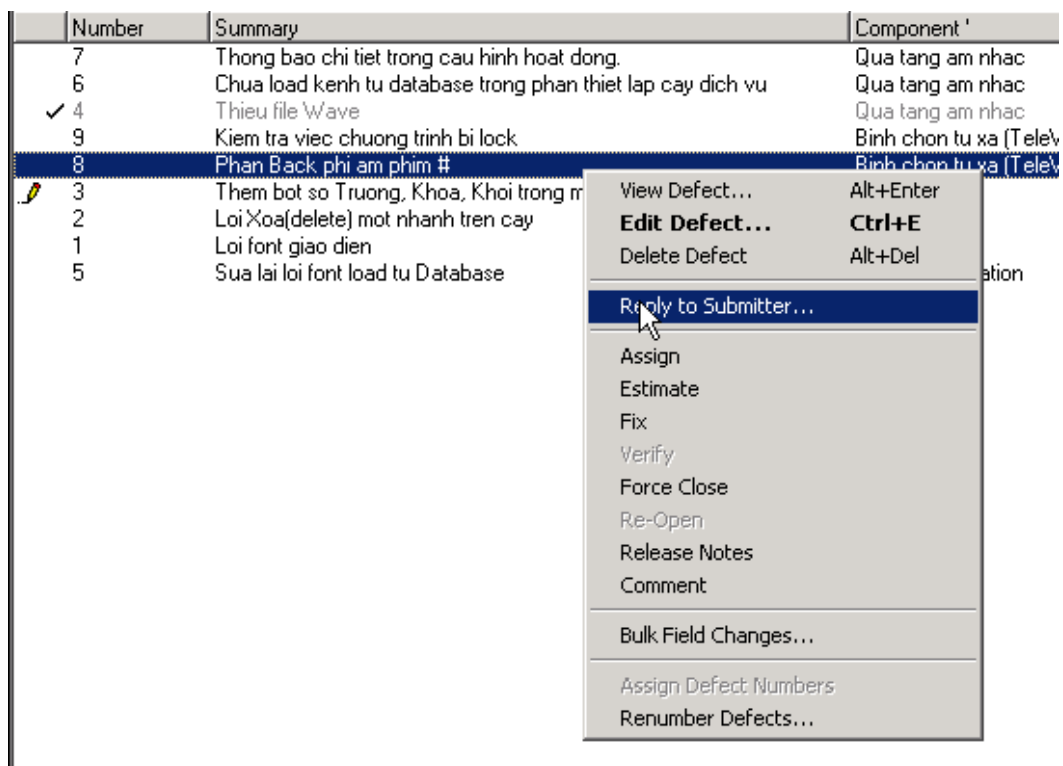
## Extended Defect Workflow



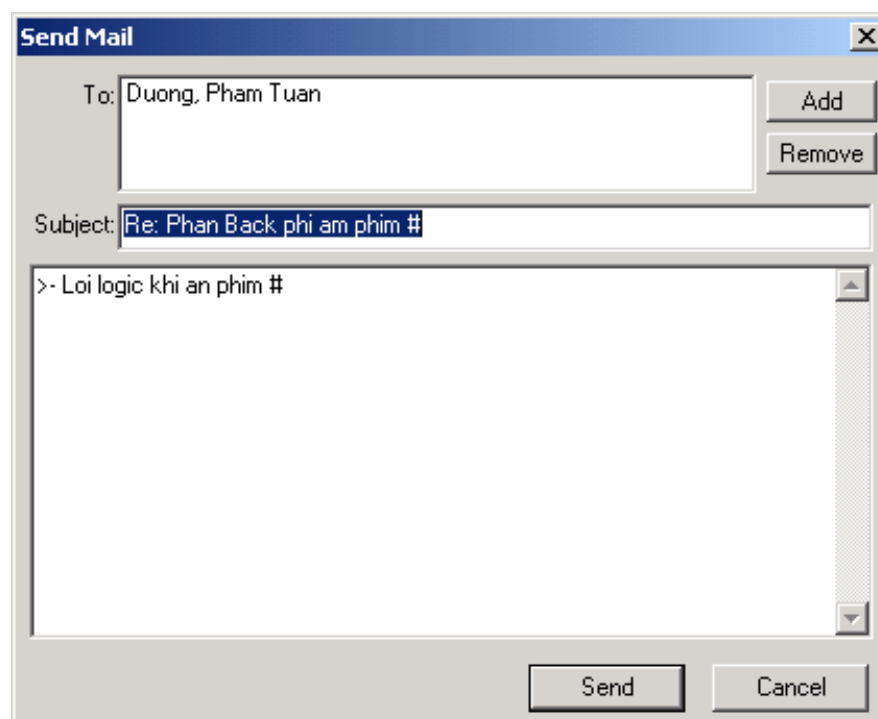
- Reply to Submitter

Nhằm mục đích gửi mail đến tác giả đã add defect hoặc một thành viên hay một nhóm thành viên trong nhóm phát triển.

- Click chuột phải chọn Reply to Submitter



- Chọn người muốn gửi và gõ nội dung thông tin vào



- Xem tiến trình các công việc đã được thực hiện kể từ khi lỗi đó xuất hiện. Chọn View Defects → Chọn Workflow

Action	Date	Who	Other Information
Assign	4/8/2005	Duong, Pham Tuan	Assign To: Cuong, Duong Xuan
Force Cl...	4/9/2005	Duong, Pham Tuan	Resolution: Not a Bug

Notes:

Phai sua loi nay ngay, neu khong khong chon duoc bai hat.

- Click đúp chuột để xem nội dung

Summary: Thieu file Wave

Status: Closed

Type: Requirement Test

Product: InteBox

Reference:

Entered by: Duong,

Disposition: Xem chương trình

Priority: Module

**Assign**

Assign By: Duong, Pham Tuan

Assign To: Cuong, Duong Xuan

Date: 4/ 8/2005

Notes:

Phai sua loi nay ngay, neu khong khong chon duoc bai hat.

OK Cancel

En

OK Cancel

## Kiểm tra hiệu năng phần mềm với LoadRunner 8.1



Mục tiêu của bài viết là giới thiệu những tính năng cơ bản của một công cụ kiểm tra tự động (KTTĐ) áp dụng trong kiểm tra hiệu năng của phần mềm hay còn gọi là Performance Test (PT). Thực tế, có rất nhiều công cụ hỗ trợ việc kiểm tra hiệu năng phần mềm, tuy nhiên qua thực tiễn sử dụng và kinh nghiệm, chúng tôi cho rằng công cụ LoadRunner 8.1 của hãng Mercury là một trong những công cụ khá tốt (và được nhiều chuyên gia kiểm tra phần mềm sử dụng).

### 1. KHÁI QUÁT VỀ PERFORMANCE TEST

PT là một dạng KTTĐ, để tìm ra điểm “thất cổ chai” của phần mềm cần kiểm tra, qua đó giúp cho người làm phần mềm có thay đổi thích hợp để tăng khả năng thực thi của phần mềm (PM). Bên cạnh đó cũng giúp người kiểm tra biết được những thông số ngưỡng của phần mềm, đề ra tiêu chuẩn cho những lần kiểm tra sau. Khi thực hiện PT, người kiểm tra phải đề ra kết quả mong đợi một cách rõ ràng. Ví dụ: đối với ứng dụng web, chúng ta cần biết thông số quan trọng là: số kết nối (session) đồng thời mà server có thể phục vụ, thời gian (bao nhiêu phút/giây) mà trình duyệt nhận được kết quả từ server... Khi thực hiện PT người ta thường chọn thời điểm mà chương trình tương đối ổn định. Thông thường chức năng nằm trong tình huống cần kiểm tra hiệu năng đã được kiểm tra là chạy đúng. Điều này sẽ giúp cho việc phân tích đánh giá kết quả của PT dễ dàng và đúng đắn. Đồng nhất với quy trình kiểm tra phần mềm (xem chi tiết trên TGVN A tháng 12/2005), mô hình sau mô tả các bước phát triển PT một cách tổng quát. Công việc trong từng bước trên như bảng sau.

**Bảng 1**

STT	Bước thực hiện	Mô tả
1	Thiết kế test case	Thiết kế test case với các bước thực hiện rõ ràng, và đề ra các thông số cần đo lường cụ thể.
2	Phát triển test script	Dùng công cụ PT chúng ta có thể lưu lại các thao tác người dùng tương tác với hệ thống dưới dạng script. Script này cũng cho phép chỉnh sửa để đạt được mục tiêu của tình huống kiểm tra đề ra.
3	Thiết kế tình huống	Hay còn gọi là thiết kế scenario, nhằm giả lập môi trường mà phần mềm hoạt động với hiệu năng giống trong thực tế.
4	Thực thi tình huống	Chạy, quản lý và giám sát việc thực hiện PT
5	Phân tích kết quả	Phân tích kết quả dựa trên thống kê mà công cụ KTTĐ cung cấp. Nếu kết quả thực tế chưa đáp ứng được yêu cầu thì PM được kiểm tra cần được điều chỉnh.

Trong lĩnh vực kiểm tra hiệu năng có một vài thuật ngữ rất dễ gây nhầm lẫn. Bảng 2 so sánh giữa 3 thuật ngữ thường hay được đề cập nhất với một số cách nhìn khác nhau.

**2. TẠI SAO PHẢI ỨNG DỤNG PTĐ** Để đảm bảo PM có chất lượng thì người kiểm tra viên (KTV) phải có những cách kiểm tra giả lập gần giống môi trường thực tế nhất. Trong thực tế có rất nhiều PM theo mô hình client-server đáp ứng nhiều người dùng cùng một lúc. Một số yêu cầu thực tế rất hay đặt ra là: • Xác định thời gian đáp ứng khi có nhiều người dùng như: số yêu cầu trên giây, số giao dịch thành công trên giây, số thông điệp chuyển giao trên giây, số gói tin trên giây, ... • Xác định biểu đồ tải nguyên chiếm giữ của PM khi có nhiều người dùng trong thời gian dài như: CPU, bộ nhớ, I/O của đĩa cứng, I/O của mạng. • Xác định khả năng phân tải, khả năng phục hồi dữ liệu khi có sự cố vì quá nhiều người dùng, ... • Đề ra cấu hình phần cứng tối thiểu để PM có thể hoạt động. • Kiểm tra việc thực hiện giao dịch có bị sai lệch khi có nhiều người cùng làm giống thao tác. Với những bài toán trên việc dùng công cụ PT là không thể tránh khỏi. Đây đồng thời cũng là đặc điểm để KTV xác định xem những trường hợp nào thì phải áp dụng KTTĐ, và tiêu biểu là dùng công cụ để thực hiện PT. PT giúp chúng ta đoán trước được những lỗi có thể xảy ra khi triển khai PM vào thực tế trong môi trường có nhiều người dùng. Bên cạnh đó còn giúp tìm ra hiệu năng thực thi tối đa của PM và tìm ra nơi cần cải tiến cho PM. PT mang các đặc tính ưu việt của KTTĐ như giảm thời gian

kiểm tra hồi qui, thực hiện đo lường các thông số chính xác, giúp giảm thiểu chi phí cho dự án...

**3. GIỚI THIỆU CÔNG CỤ LOADRUNNER** Mercury LoadRunner là công cụ KTTĐ thực hiện việc kiểm tra hiệu năng của PM. Nó cho phép chúng ta tìm ra những lỗi về khả năng thực thi bằng việc phát hiện nguyên nhân, chỗ làm cho PM chạy chậm hoặc không đúng yêu cầu. Đây là công cụ mạnh với giải pháp kiểm tra tải, phát hiện và đưa ra giải pháp cải tiến. Ứng dụng LoadRunner sẽ giúp giảm thời gian viết test script đến 80%, đó là nhờ nó cung cấp chức năng tự động phát sinh script mô tả lại các tình huống muốn kiểm tra.

**3.1 Đặc điểm** Theo một số quan niệm thì một công cụ PT phải có khả năng thực hiện kiểm tra chức năng. Điều này mang nghĩa tương đối vì thực tế công cụ PT không thể thay thế công cụ kiểm tra chức năng và ngược lại. Ví dụ: trong môi trường web, công cụ kiểm tra chức năng kiểm tra hoạt động của phần mềm ở cả phía client và server. Còn công cụ PT chỉ kiểm tra ở phía server mà thôi. LoadRunner có khả năng tạo ra hàng ngàn người dùng ảo thực hiện các giao dịch cùng một lúc. Sau đó LoadRunner giám sát các thông số xử lý của PM được kiểm tra. Kết quả thống kê sẽ được lưu lại và cho phép KTV thực hiện phân tích.

- Các thành phần của LoadRunner (bảng 3)

Bảng 3	
Thành phần	Chức năng
Virtual User Generator	Tự động tạo ra VuGen script để lưu lại các thao tác người dùng tương tác lên PM. VuGen script này còn được xem là hoạt động của một người ảo mà LoadRunner giả lập.
Controller	Tổ chức, điều chỉnh, quản lý và giám sát hoạt động kiểm tra tải. Thành phần này có chức năng tạo ra những tình huống (scenario) kiểm tra.
Load Generator	Cho phép giả lập hàng ngàn người dùng, hoạt động của từng người sẽ được thực hiện theo VuGen script. Kết quả thực hiện sẽ được thông báo cho Controller.
Analysis	Cung cấp việc xem, phân tích và so sánh các kết quả PT.
Launcher	Nơi tập trung tất cả các thành phần của LoadRunner cho người dùng.

- Ngôn ngữ viết script Script của LoadRunner được tự động phát sinh dưới dạng ngôn ngữ C chuẩn. Tuy nhiên LoadRunner có khả năng hỗ trợ thêm việc viết script theo dạng cú pháp của ngôn ngữ Java và Visual Basic. Ngoài các hàm cơ bản của từng ngôn ngữ, LoadRunner còn tự cung cấp thêm những mã lệnh (API) riêng dành cho từng môi trường hỗ trợ khác nhau như web, Oracle, SAP, Java,... Sau đây chúng



ta xem qua script tự động phát sinh sau khi thao tác với ứng dụng Mercury Web Tours tại <http://127.0.0.1:1080/mercuryWebTours>. Lưu ý mở web server chạy trước khi thực hiện tại menu Start>Programs>Mercury LoadRunner >Samples>Web>Start Web Server. Trên đây là hành động của một người dùng ảo sẽ thực hiện:- Hàm web\_url(...): cho phép tải (mở) trang web <http://127.0.0.1:1080/mercuryWebTours>.- Hàm web\_submit\_form(...): người dùng ảo đăng nhập website bằng tài khoản jojo/bean.- Hàm web\_image(...): người dùng ảo chọn nút SignOff để thoát.

### 3.2 Các bước thực hiện

Nhằm minh họa cho qui trình phát triển PT, sau đây chúng ta đi sơ lược qua các bước triển khai PT trong LoadRunner. Sau khi cài đặt, LoadRunner sẽ cung cấp sẵn một số ứng dụng cho chúng ta thử nghiệm. Trong bài viết này chúng ta thực hiện kiểm tra ứng dụng Mercury Web Tours.

- Mở web server tại \$Mercury LoadRunner >Samples>Web>Start Web Server.
- Mở LoadRunner \$menu Start>Programs>Mercury LoadRunner>LoadRunner. Chọn Create/Edit Scripts để mở Virtual User Generator tạo script cho người dùng ảo.
- Chọn giao thức để kiểm tra ứng dụng web: chọn New Vuser script, sau đó chọn Web (HTTP/HTML).
- LoadRunner tổ chức các bước thực hiện một cách rõ ràng bên trái màn hình, rất dễ sử dụng. Sau đây là công việc phải làm trong các bước đó.

#### 1. Recording (Ghi nhận):

Cho phép tự động phát sinh script ghi lại thao tác người dùng lên ứng dụng cần kiểm tra. Cách tổ chức script của LoadRunner được chia ra thành 3 thành phần chính.

- vuser\_init: mỗi người dùng ảo sẽ thực hiện một lần trước khi chạy PT.
- Run: bao gồm một hoặc nhiều hàm (action), và cho phép người dùng ảo chạy lặp lại nhiều lần. Dựa trên action chúng ta có thể tổ chức các nhóm chứa các action khác nhau và theo thứ tự tùy ý.
- vuser\_end: mỗi người dùng ảo thực hiện một lần cuối cùng khi chạy PT.

#### 2. Replay (Phát lại):

Đây là bước cho phép chạy thử để kiểm tra script đã chạy đúng yêu cầu của một người dùng ảo hay chưa, qua đó có sự chỉnh sửa nếu cần. Bên cạnh đó LoadRunner còn cho phép tổ chức thứ tự, số lần lặp lại các action hiện đang có bằng cách chọn Open Run-Time Settings, và thiết lập thời gian tương tác giữa người dùng ảo và web server.

#### 3. Enhancements (Nâng cao):

- Tạo transaction: transaction là một số hành động do chúng ta chọn từ quá trình tự động phát sinh script. Mục tiêu là giám sát thông số hoạt động của một số hành động trong transaction đó. Thông số giám sát sẽ được thể hiện sau khi chúng ta thực hiện kiểm tra PT.
- Tham biến hóa: thay thế các giá trị cố định trong script bằng các biến.
- Kiểm tra nội dung: cho phép thêm các điểm kiểm tra nội dung, trong LoadRunner gọi là content check, có thể hiểu giống như một thuật ngữ khác đã được đề cập trong bài viết trước là checkpoint.

#### 4. Prepare For Load (Chuẩn bị thực thi):

- Thiết lập sự lặp lại của các action, ở giai đoạn Replay chúng ta cũng có thể làm điều này.
- Thiết kế tình huống: thiết lập số người dùng ảo tối đa thực hiện cùng một lúc, thời gian chạy bao lâu, số lượng người dùng ảo tăng như thế nào (Ramp Up) hoặc giảm như thế nào (Ramp Down). Ví dụ: Giả lập tổng số người dùng tối đa là 100, bắt đầu là 10 người và cứ 10 phút thì tăng 1 người. Chạy trong thời gian 5 tiếng, sau đó cứ 10 phút thì giảm 1 người dùng.
- Thực hiện tình huống: thực thi các tình huống kiểm tra, phân tích kết quả dựa trên các thông số của môi trường kiểm tra, ví dụ: số yêu cầu gửi tới web server xử lý trong 1 giây, số hồi đáp từ server trong 1 giây, số trang mà người dùng có thể mở trong 1 giây, ...

#### 5. Finish (Kết thúc):

Upload script lên Performance Center server, đây là một phần trong việc thực hiện giải pháp chia sẻ

tài nguyên PT qua Internet.- Quản lý script thông qua việc kết hợp với giải pháp mà Mercury cung cấp là Quality Center.

<b>Bảng 4</b>	
• Triển khai ứng dụng	Citrix ICA
• Client/Server	DB2 CLI, DNS, Infomix, MS SQL Server, ODBC, Oracle® (2-tier), Sybase CTLib, Sybase DBLib, Windows Sockets.
• Hệ thống phân tán	COM/DCOM, CORBA-Java™, Rmi-Java.
• E-Business	FTP, LDAP, Microsoft .NET, Palm, Web (HTTP/HTML), Webservices, Web/Winsocket Dual Protocol.
• ERP/CRM	Oracle NCA, Oracle Web Applications 11i, PeopleSoft Enterprise, PeopleSoft Tuxedo, SAP-Web, SAPGUI/SAP-Web Dual Protocol, Siebel-DB2 CLI, Siebel-Oracle, Siebel-MSSQL, Siebel-Web.
• Enterprise Java Bean	EJB, Rmi-Java.
• Legacy	Terminal Emulation (RTE).
• Mailing Services	IMAP, MAPI, POP3, SMTP.
• Middle ware	Jacada, Tuxedo 6, Tuxedo 7.
• Streaming	Media Player (MMS), Real.
• Wireless	i-mode, VoiceXML, WAP.
• Khác	C Vuser, Java Vuser, Javascript Vuser, VB/VB Script/VBNet Vuser,...

**3.3 Môi trường hỗ trợ** LoadRunner có khả năng thực hiện PT trên nhiều môi trường khác nhau, cụ thể những giao thức và công nghệ phổ biến như ERP/CRM, Web, J2EE/.NET, XML, .NET, Wireless, Streaming Media. LoadRunner hỗ trợ hơn 60 giao thức và được xem là công cụ có khả năng hỗ trợ tối đa những công nghệ mới hiện nay. Bảng 4 sau đây là một số môi trường hỗ trợ tiêu biểu.

Bên cạnh đó còn có những môi trường yêu cầu phải mua và cài đặt thêm thành phần hỗ trợ như: VMWare, Web Forms, Java (SWT, AWT), ActiveX, Delphi 8 .NET WinForms, WPF from .NET 3.0, JBDC, SIP...**4. SỬ DỤNG**  
**4.1 Hướng dẫn cài**

**đặt**Mercury cung cấp bản LoadRunner chạy thử trong vòng 10 ngày tại <http://downloads.mercury.com>. Sau khi tải về chúng ta thực hiện theo các bước hướng dẫn để cài đặt. Toàn bộ các thành phần của LoadRunner đều có thể cài trên Windows, còn trên hệ thống Unix thì chúng ta chỉ có thể cài đặt Load Generator.**4.2 Yêu cầu hệ thống (bảng 5)**

<b>Bảng 5</b>				
<b>Yêu cầu</b>	<b>Thành phần</b>			
	<b>Controller</b>	<b>Vuser Generator</b>	<b>Load Generator</b>	<b>LoadRunner</b>
<b>CPU</b>	Pentium III trở lên (đề nghị Pentium IV) 1 Ghz trở lên (đề nghị 2.4 Ghz)			
<b>Hệ điều hành</b>	Windows 2000 SP4 Windows 2003 PS3 (Standard and Enterprise editions) Windows XP SP 2 (đã tắt firewall)			
			HĐH dựa trên Unix	
<b>RAM</b>	1 GB	512 MB trở lên (đề nghị 1 GB)	Phụ thuộc vào giao thức và ứng dụng phải kiểm tra.	512 MB trở lên (đề nghị 1 GB)
<b>Đĩa cứng</b>	2 GB	1 GB	1 GB	1 GB
<b>Trình duyệt</b>	Internet Explorer 6.0 service pack 1 trở lên			

**4.3 Giá cả**Sau đây là bảng giá tham khảo, giá này được bán trên thị trường vào năm 2006 (chúng tôi chưa có điều kiện xác nhận mức giá tại thời điểm hiện tại, bạn đọc có thể kiểm tra giá chính xác tại một số đại lý tại Việt Nam như công ty TD&T Tân Đức, 103 Pasteur, Q.1, HCM). Bản quyền LoadRunner gồm hai thành phần sau:  
 • Virtual User Generator: 22.000 USD  
 • Load Generator: giá được bán theo số lượng người dùng được giả lập\* 25 người dùng ảo: 16.000 USD\* 50 người dùng ảo: 22.000 USD\* 100 người dùng ảo: 33.000 USD\* 200 người dùng ảo: 44.000 USD\* 500 người dùng ảo: 55.000 USD  
**4.4 Giới thiệu nguồn tài liệu hay**Do khuôn khổ bài báo, chúng tôi chỉ giới thiệu những phần cốt lõi và chung nhất về kiểm tra hiệu năng và tính năng sử dụng của LoadRunner, chi tiết sâu hơn, bạn đọc có thể tham khảo các nguồn tài liệu sau:  
 • <http://www.wilsonmar.com>: Giới thiệu PT, mô hình KTTĐ. Trình bày chi tiết kiến trúc, chức năng của LoadRunner, kỹ thuật phát triển VuGen script và nhiều chủ

đề liên quan. • <http://www.aptest.com/resources.html>: Giới thiệu rất nhiều công cụ PT • <http://www.sqaforums.com>: Diễn đàn trao đổi về chất lượng PM, có rất nhiều chủ đề về công cụ PT.

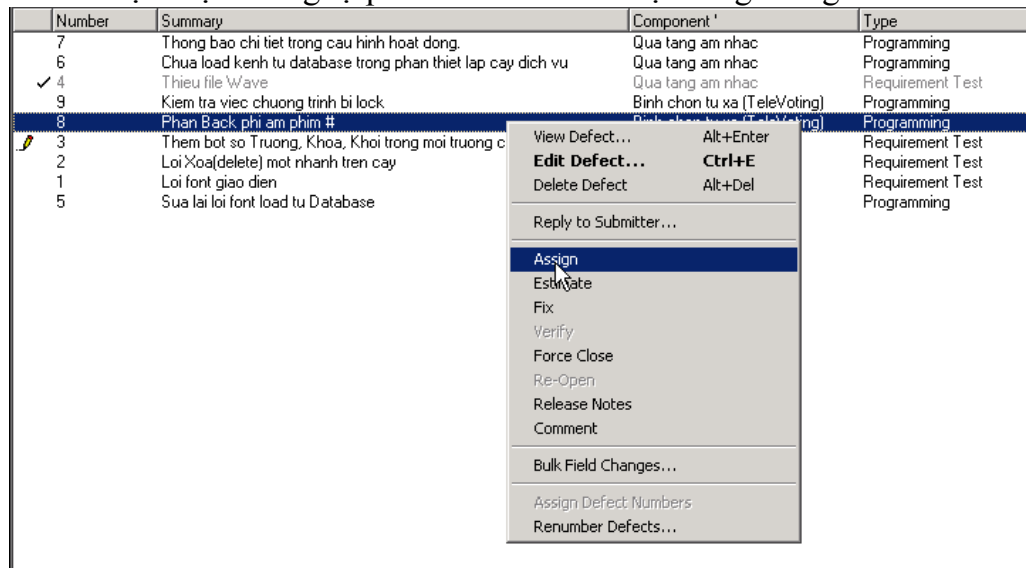
**5. LỜI KẾT** Thực tế cho thấy một phần mềm gọi là “chạy tốt” phải đồng thời đáp ứng 2 tiêu chí vận hành quan trọng là chức năng và hiệu năng. Việc tránh né không áp dụng PT để kiểm tra hiệu năng hoạt động có thể dẫn đến tình trạng PM bị lỗi khi hoạt động thực tế, trong nhiều trường hợp phần mềm trở nên “vô dụng” vì không đáp ứng yêu cầu về hiệu năng, mặc dù chức năng vẫn “chạy tốt”. Ngược lại nếu áp dụng PT một cách hợp lý sẽ giúp tìm ra nơi mà PM cần được cải tiến ngay trong quá trình phát triển phần mềm. LoadRunner là công cụ PT mạnh mẽ với nhiều tính năng phong phú, đặc tính dễ sử dụng của nó giúp các KTV tiết kiệm rất nhiều thời gian trong việc thực hiện PT, đặc biệt trong kiểm tra hồi qui. Đây cũng là công cụ KTTĐ tiêu biểu giúp KTV hiểu được nguyên tắc hoạt động của một công cụ hỗ trợ và thực thi PT, từ đó giúp KTV có khả năng đề ra các mục tiêu cho việc thực hiện PT và phân tích kết quả báo cáo để tìm ra sự cải tiến cần thiết cho PM.

## Esstimate

### b.Assign

Giao nhiệm vụ này cho ai giải quyết, nó sẽ được gửi đến mail của người được giao và sẽ được ghi cụ thể trong defect đó ở mục **workflow**.

- Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin



**Assign**

Assign By:  Date: ☒ 4/11/2005

Assign To:

Notes:

OK Cancel

Action	Date	Who	Other Information
Assign	4/8/2005	Duong, Pham Tuan	Assign To: Cuong, Duong Xuan
Force Cl...	4/9/2005	Duong, Pham Tuan	Resolution: Not a Bug

Notes:

Phai sua loi nay ngay, neu khong khong chon duoc bai hat.

Summary: Thieu file Wave

Status: Closed

Type: Requirement Test

Product: InteBox

Reference:

Entered by: Duong,

Disposition: Xem chương trình

Priority: Module

**Assign**

Assign By: Duong, Pham Tuan

Assign To: Cuong, Duong Xuan

Date: 4/ 8/2005

Notes:

Phai sua loi nay ngay, neu khong khong chon duoc bai hat.

OK Cancel

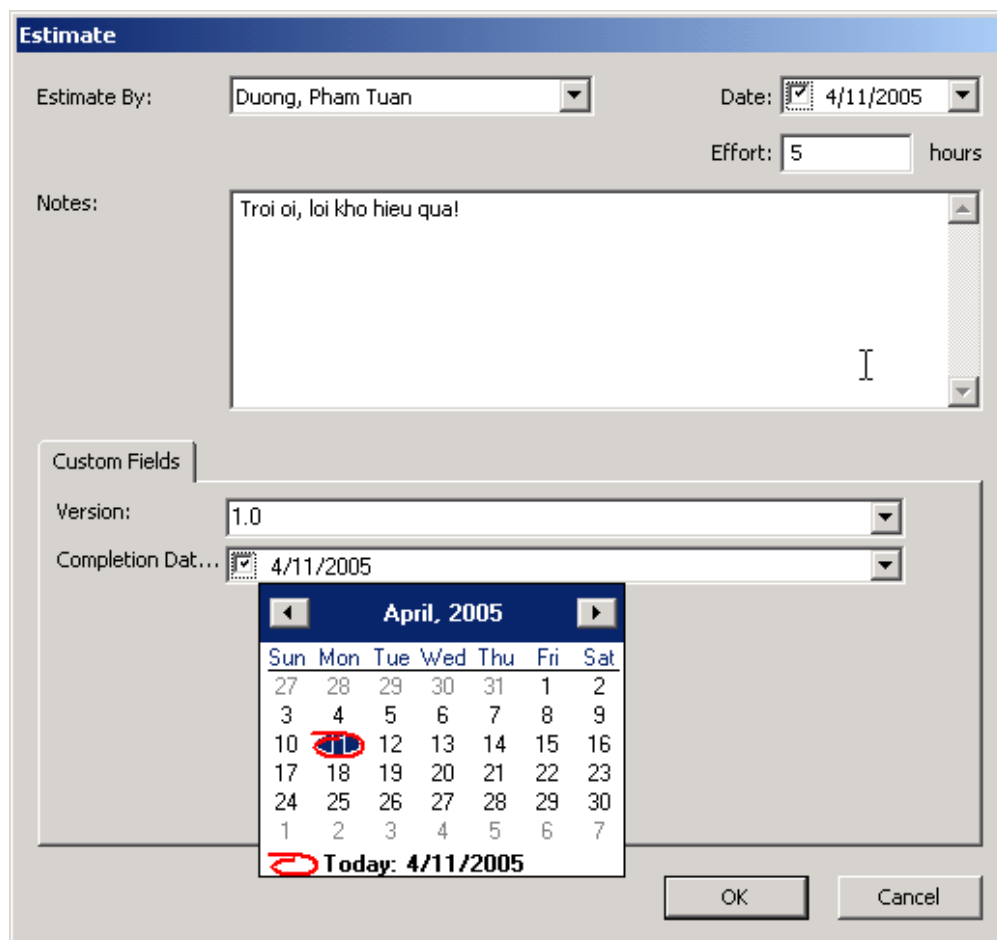
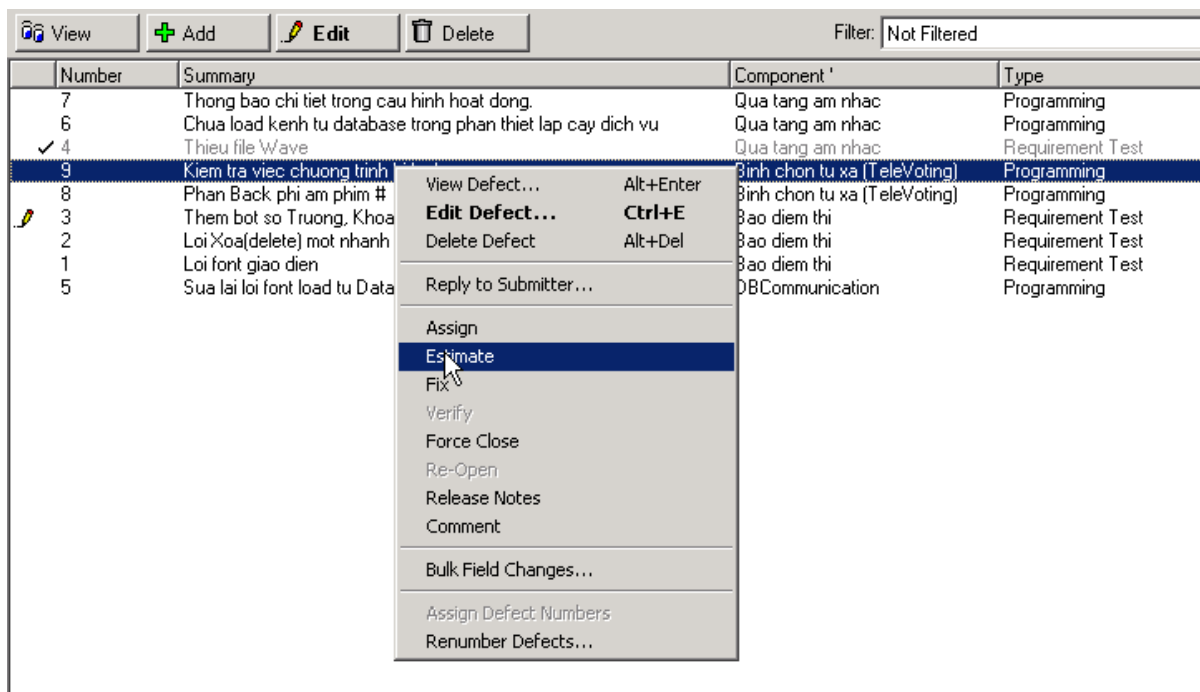
En

OK Cancel

- Esstimate

Nó là luồng việc tiếp theo sau khi assign như dự kiến thời gian, ngày hoàn thành và sẽ được ghi cụ thể trong defect đó ở mục **workflow**.

Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin



- Fix

Cụ thể lỗi là gì, đưa ra hướng giải pháp ra sao và đã sửa trong bao lâu sẽ được ghi cụ thể trong defect đó ở mục **workflow**.

Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin

Number	Summary	Component '	Type
7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhạc	Programming
6	Chưa load kênh tu database trong phần thiết lập cây dịch vụ	Qua tang am nhạc	Programming
✓ 4	Thiếu file Wave	Qua tang am nhạc	Requirement Test
9	Kiểm tra việc chương trình	Bình chọn tu xa (TeleVoting)	Programming
8	Phản Back phi âm phim	Bình chọn tu xa (TeleVoting)	Programming
3	Thêm bot so Trương, Kh	Bao diem thi	Requirement Test
2	Lỗi Xóa(delete) một nhar	Bao diem thi	Requirement Test
1	Lỗi font giao diện	Bao diem thi	Requirement Test
5	Sửa lại lỗi font load tu D	DBCcommunication	Programming

Fix

Fixed By:

Duong, Pham Tuan

Date:

4/11/2005

Resulting State:

Fixed

Effort:

hours

Notes:

Custom Fields

Affects Documentation

Affects Test Plan

Resolution...

Code Change

Version:

OK

Cancel



- Verify

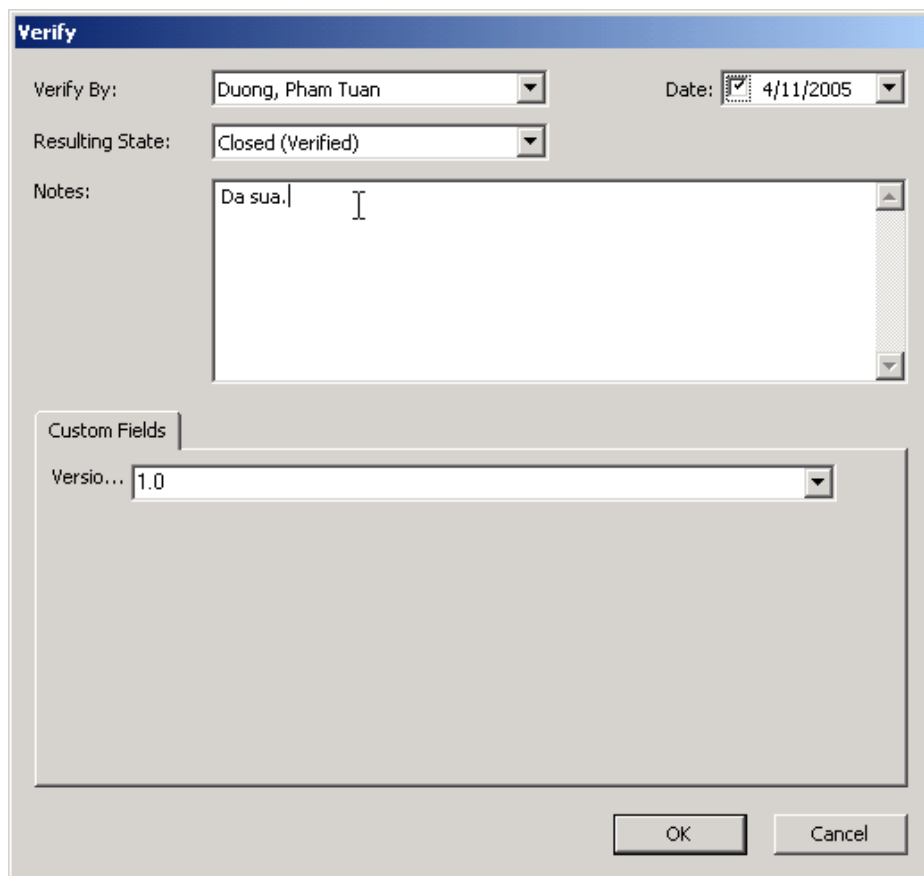
Thành viên trong nhóm kiểm tra và xác nhận đây liệu lỗi đã được sửa chưa sẽ được ghi cụ thể trong defect đó ở mục **workflow**.

Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin

Number	Summary	Component '	Type
7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhạc	Prograr
6	Chưa load kênh tu database trong phần thiết lập cây dịch vụ	Qua tang am nhạc	Prograr
✓ 4	Thieu file Wave	Qua tang am nhạc	Require
9	Kiểm tra việc chương trình bị lock	Bình chọn tu xa (TeleVoting)	Prograr
8	Phan Back phi âm phim #	Bình chọn tu xa (TeleVoting)	Prograr
3	Thêm bot số Trường, ...	Bao diem thi	Require
2	Lỗi Xóa(delete) một nh	Bao diem thi	Require
1	Lỗi font giao diện	Bao diem thi	Require
5	Sửa lại lỗi font load tu	DBCommunication	Prograr

View Defect...	Alt+Enter
<b>Edit Defect...</b>	<b>Ctrl+E</b>
Delete Defect	Alt+Del
Reply to Submitter...	
Assign	
Estimate	
Fix	
<b>Verify</b>	
Force Close	
Re-Open	
Release Notes	
Comment	
Bulk Field Changes...	
Assign Defect Numbers	
Renumder Defects...	

A screenshot of a 'Verify' dialog box. The title bar is blue with the word 'Verify' in white. The dialog has a light gray background. At the top, there are two dropdown menus: 'Verify By:' with the value 'Duong, Pham Tuan' and 'Date:' with the value '4/11/2005'. Below these is another dropdown menu 'Resulting State:' with the value 'Closed (Verified)'. Underneath is a text area labeled 'Notes:' containing the text 'Da sua.' with a cursor. Below the text area is a section titled 'Custom Fields' with a tabbed interface. The first tab is selected and shows a dropdown menu labeled 'Versio...' with the value '1.0'. At the bottom right are 'OK' and 'Cancel' buttons.

Verify

Verify By: Duong, Pham Tuan Date: 4/11/2005

Resulting State: Closed (Verified)

Notes: Da sua.

Custom Fields

Versio... 1.0

OK Cancel

- Force Close

Sau khi sửa lỗi hoàn thành đóng lại và defect sẽ được ẩn đi và sẽ được ghi cụ thể trong defect đó ở mục **workflow**.

Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin

Number	Summary	Component '	Type
7	Thong bao chi tiet trong cau hinh hoat dong.	Qua tang am nhac	Programming
6	Chua load kenh tu database trong phan thiet lap cay dich vu	Qua tang am nhac	Programming
✓ 4	Thieu file Wave	Qua tang am nhac	Requirement Test
9	Kiem tra viec chuong trinh bi lock	Binh chon tu xa (TeleVoting)	Programming
8	Phan Back phi am phim #	Binh chon tu xa (TeleVoting)	Programming
3	Them bot so Truong, Khoa, Kh	diem thi	Requirement Test
2	Loi Xoa(delete) mot nhanh tren	diem thi	Requirement Test
1	Loi font giao dien	diem thi	Requirement Test
5	Sua lai loi font load tu Databas	Communication	Programming

View Defect...	Alt+Enter
<b>Edit Defect...</b>	<b>Ctrl+E</b>
Delete Defect	Alt+Del
Reply to Submitter...	
Assign	
Estimate	
Fix	
Verify	
<b>Force Close</b>	
Re-Open	
Release Notes	
Comment	
Bulk Field Changes...	
Assign Defect Numbers	
Renumbr Defects...	

Force Close

Force Close By:

Duong, Pham Tuan

Date:

4/11/2005

Notes:

Custom Fields

Resolutio...

Not Our Bug

OK

Cancel

- Re-Open

Nhưng sau một thời gian lại phát hiện lỗi thì có thể mở lại lỗi đó thì defect đó sẽ lại enable sẽ được ghi cụ thể trong defect đó ở mục **workflow**.

Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin

Number	Summary	Component '	Type	Dispos
7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhac	Programming	Xem cl
6	Chưa load kênh tu database trong phân thiết lập cây dịch vụ	Qua tang am nhac	Programming	Xem cl
✓ 4	Thieu file Wave	Qua tang am nhac	Requirement Test	Xem cl
9	Kiểm tra việc chương trình bị lock	Bình chọn tu xa (TeleVoting)	Programming	Khi tick
8	Phản Back phi âm phẩm #	Bình chọn tu xa (TeleVoting)	Programming	Xem cl
3	Thêm bot số Trường, Khoa, Khoi trong môi trường chưa hoàn chỉnh	Bao diem thi	Requirement Test	Khi cap
2	Lỗi Xóa(delete) một nhánh trên cây	Bao diem thi	Requirement Test	Khi xoa
1	Lỗi font giao diện	Bao diem thi	Requirement Test	Xem cl
5	Sửa lại lỗi font load tu Database	DBCommunication	Programming	Khi tick

Number	Summary	Component '	Type
7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhac	Programming
6	Chưa load kênh tu database trong phân thiết lập cây dịch vụ	Qua tang am nhac	Programming
✓ 4	Thieu file Wave	Qua tang am nhac	Requirement Test
9	Kiểm tra việc chương trình bị lock	Bình chọn tu xa (TeleVoting)	Programming
8	Phản Back phi âm phẩm #	Bình chọn tu xa (TeleVoting)	Programming
3	Thêm bot số Trường, Khoa, Khoi	Bao diem thi	Requirement Test
2	Lỗi Xóa(delete) một nhánh trên cây	Bao diem thi	Requirement Test
1	Lỗi font giao diện	Bao diem thi	Requirement Test
5	Sửa lại lỗi font load tu Database	DBCommunication	Programming

View Defect... Alt+Enter  
**Edit Defect...** Ctrl+E  
Delete Defect Alt+Del  
  
Reply to Submitter...  
  
Assign  
Estimate  
Fix  
Verify  
Force Close  
**Re-Open**  
Release Notes  
Comment  
  
Bulk Field Changes...  
  
Assign Defect Numbers  
Renumber Defects...

**Re-Open**

Re-Open By: Duong, Pham Tuan
Date: 4/11/2005

Notes:

Ly

OK Cancel

- Release Notes

Là những thông báo ghi cụ thể cho mỗi lần sửa lỗi cho một phiên bản cụ thể

Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin

Number	Summary	Component '	Type	D
7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhạc	Programming	Xi
6	Chưa load kênh tu database trong phần thiết lập cây dịch vụ	Qua tang am nhạc	Programming	Xi
✓ 4	Thiếu file Wave	Qua tang am nhạc	Requirement Test	Xi
9	Kiểm tra việc chương trình bị lock	Bình chọn tu xa (TeleVoting)	Programming	Ki
8	Phản Back phi âm phim #	Bình chọn tu xa (TeleVoting)	Programming	Xi
3	Thêm bot số Trung, Khoa	Bao diem thi	Requirement Test	KI
2	Lỗi Xóa(delete) một nhánh	Bao diem thi	Requirement Test	KI
1	Lỗi font giao diện	Bao diem thi	Requirement Test	Xi
5	Sửa lại lỗi font load tu Data	DBCommunication	Programming	KI

View Defect...  
Alt+Enter

Edit Defect...  
Ctrl+E

Delete Defect  
Alt+Del

Reply to Submitter...

Assign

Estimate

Fix

Verify

Force Close

Re-Open

Release Notes

Comments

Bulk Field Changes...

Assign Defect Numbers

Renumber Defects...

Release Notes

Release Notes By: Duong, Pham Tuan

Date: 4/11/2005

Release Version: 1.0

Notes: Can chu y.

OK

Cancel

- Comment

Là những chú ý cần nhớ cho mỗi defects khi được sửa sẽ được ghi cụ thể trong defect đó ở mục **workflow**.

Cách thực hiện tương tự phần trên chỉ khác nội dung thông tin

Number	Summary	Component
7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhạc
6	Chưa load kênh từ database trong phần thiết lập cây dịch vụ	Qua tang am nhạc
✓ 4	Thiếu file Wave	Qua tang am nhạc
9	Kiểm tra việc chương trình bị lock	Bình chọn từ xa (TeleVoting)
8	Phản Back phím âm phím #	Chọn từ xa (TeleVoting)
3	Thêm bot số Trường, Khoa, Khoa, Khoa	thi
2	Lỗi Xóa(delete) một nhánh trên cây	thi
1	Lỗi font giao diện	thi
5	Sửa lại lỗi font load từ Database	Communication

View Defect...Alt+Enter

Edit Defect...Ctrl+E

Delete DefectAlt+Del

Reply to Submitter...

Assign

Estimate

Fix

Verify

Force Close

Re-Open

Release Notes

Comment

Bulk Field Changes...

Assign Defect Numbers

Renumber Defects...

Comment

Comment By:Duong, Pham Tuan

Date:4/11/2005

Notes:

Can chu y

OKCancel

- Bulk Field Changes

Lọc defect theo điều kiện

**Bulk Field Changes**

General | Reported By | Steps to Reproduce | Computer Config | Defect Actions | Custom Fields

General Defect Fields

Summary:  ☒ Prepend ☐ Append

Summary text:

Replace:  With:

Type:  Priority:

Product:  Component:

Entered by:  Severity:

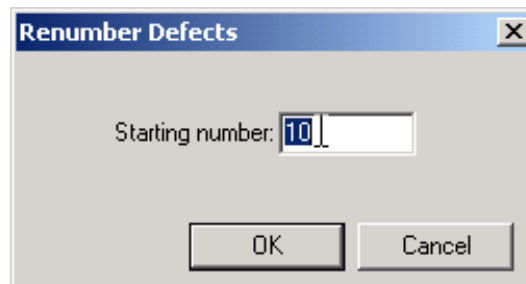
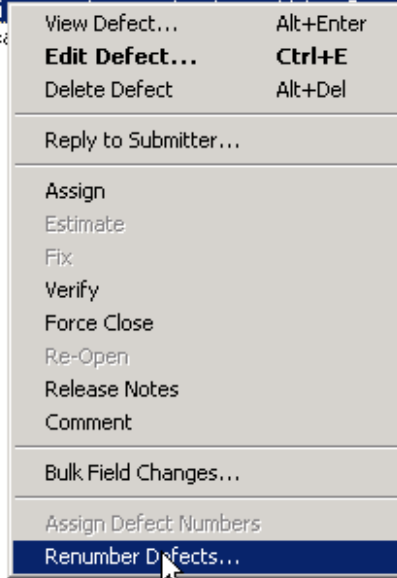
Disposition:  Date Entered: ☒ 4/11/2005

OK Cancel Help

- Renumber Defects

Đánh lại số thứ tự defect cho theo logic de hiểu tùy theo nhu cầu người Test

Number	Summary	Component	Type
7	Thông báo chi tiết trong cấu hình hoạt động.	Qua tang am nhạc	Pro
6	Chưa load kênh từ database trong phần thiết lập cây dịch vụ	Qua tang am nhạc	Pro
✓ 4	Thiếu file Wave	Qua tang am nhạc	Rec
9	Kiểm tra việc chương trình bị lock	Bình chọn từ xa (TeleVoting)	Pro
8	Phản Back phím am phím #	Bình chọn từ xa (TeleVoting)	Pro
3	Thêm bot số Trường, Khoa, Khoa	em thi	Rec
2	Lỗi Xóa(delete) một nhánh trên cây	em thi	Rec
1	Lỗi font giao diện	em thi	Rec
5	Sửa lại lỗi font load từ Database	communication	Pro



## 1. Sử dụng Workbook

Nhằm ghi lại hay chuẩn bị các công việc, nhiệm vụ sắp tới, chọn biểu tượng



Tạo mới: Create->To Do.. hoặc



Chỉnh sửa: Edit -> Edit Task..



Xoá: Edit ->Delete Task





Sau khi hoàn thành công việc đã dự tính bấm Edit -> Done



To Do:

Priority:  Date: ☒ 03/12/2004 ☐ Done

Need by date: ☒ 04/12/2004 or by version:

Description:

1. Tài liệu hướng dẫn sử dụng

- TestTrackPro Client (đi kèm với bộ cài)
- TestTrackPro Admin Server (đi kèm với bộ cài)
- TestTrackPro Server.doc (Duongpt)

## Kiểm thử tự động với phần mềm

**VẤN ĐỀ1. Test Script** Khi tự động hóa quá trình kiểm tra phần mềm, hầu hết kiểm tra viên (KTV - tester) đều thực hiện các bước sau:

- Ghi lại các thao tác kiểm tra
- Chỉnh sửa các đoạn script phát sinh
- Chạy lại đoạn script đã được chỉnh sửa
- Phân tích kết quả

Nhìn chung các bước trên dễ thực hiện với mọi KTV. Tuy nhiên, những đoạn script phát sinh thường khó đọc và khó bảo trì sau này. Trong khi đó, QTP đã hỗ trợ tối đa về việc viết Test script, cho phép tạo thư viện hàm cần thiết để dùng lại sau này cho những dự án khác.

**2. Test Report** Sau khi chạy hoàn tất một đoạn test, QTP sinh ra một report "rất đẹp" với những loại kết quả như: passed, failed, done và warning. Kết quả này rất có ích cho KTV trong việc xác định những lỗi (bug) của test script hay của ứng dụng. Nhưng liệu trường hợp dự án (Project Manager) hay khách hàng có thể hiểu được bản báo cáo này?

**3. Cấu trúc action** Đối với những ai có kinh nghiệm viết script, cấu trúc action trong QTP quả là phiền phức. Vấn đề được giải quyết bằng các hàm hay thủ tục. Tuy nhiên, nếu kiểm tra tự động cho một dự án lớn mà có quá nhiều hàm và thủ tục, không phân nhóm thì rất dễ xảy ra những vấn đề về quản lý vùng (scope management) sau này.

**4. Lưu trữ và lưu vết** Đây là vấn đề mà hầu hết người dùng QTP đều gặp phải. QTP lưu các thành phần test của mình với rất nhiều định dạng tập tin và thư mục khác nhau. Việc lưu trữ và lưu vết (tracking) những sửa đổi trên cho cả tập hợp này không phải đơn giản.

**GIẢI PHÁP1. Lưu trữ độc lập** Lưu trữ các thành phần test thành từng tập tin độc lập. Với số lượng tập tin không nhiều, chúng ta có thể hoàn toàn lưu vết và lưu trữ một cách dễ dàng. Nhìn chung, ta có 4 loại tập tin cần phải lưu:

- Test script: lưu trữ những đoạn test script dưới dạng \*.vbs
- Repository: lưu trữ những đối tượng test của ứng dụng dưới dạng \*.tsr
- Parameter: Lưu trữ những thông số test dưới dạng \*.xls
- Settings: Lưu trữ cấu hình cho QTP hoặc những cấu hình đặc thù dưới dạng \*.vbs

Tùy vào những dự án cụ thể, ta có thể tạo những cấu trúc thư mục test khác nhau. Cơ bản, thư mục test nên có những thành phần như sau:

**2. Chia sẻ Repository** Sẽ có nhiều KTV cùng làm việc cho một dự án. Vì vậy, để đảm bảo các đối tượng test không bị thừa/thiếu, chúng ta nên chia sẻ repository file cho nhau. Các tên của đối tượng cũng nên đặt theo một quy chuẩn nhất định.

**3. Viết script theo cấu trúc class** Từ phiên bản QTP 8.2 đã hỗ trợ VBScript 5.0, cho phép sử dụng cấu trúc class.

- Mỗi class là một test suite, bao hàm nhiều test case.
- Dùng hàm Class\_Initialize và Class\_Terminate của cấu trúc class tương ứng như là hàm setup và teardown của một lớp test suite.
- Nên có một hàm Run (thuộc tính truy cập là public) để gọi các test case trong lớp test suite.
- Tạo ra các hàm cho các test case tương ứng.

```
Class CTest =====Test case: footestDescription: footestPublic Sub TC01_footest()Do something hereEnd Sub=====Description: Run test cases=====Public Sub Run()TC01_fooEnd Sub===== CONSTRUCTORPrivate Sub Class_InitializeEnd Sub=====DESTRUCTORPrivate Sub Class_Terminate End SubEnd Class
```

**4. Thay đổi hàm thao tác của đối tượng test** Để tạo ra một report dễ hiểu, chúng ta phải ghi lại các thao tác

của người dùng trong quá trình test. Do đó, chúng ta cần thay đổi lại các hàm thao tác cho các lớp đối tượng test. Sau đó đăng ký với QTP bằng hàm RegisterUserFunc. Ví dụ: Function Click\_Webbutton(obj) If not obj.exists Then Report: "Button does not exist" Exit function End If bVisible=obj.GetROProperty ("visible") If bVisible=False Then Report: "Button is not visible" Exit function End If Report: " Click on WebButton" Click\_Webbutton = obj.Click End Function Register this function RegisterUserFunc "WebButton", "Click", "Click\_Webbutton"

**5. Thay đổi hàm check** Ý nghĩa của check point là để xét xem những kết quả hiện hành ở ứng dụng có đúng như mong đợi hay không và xuất kết quả ra report. Để tránh phụ thuộc về cấu trúc tập tin và thư mục của QTP, chúng ta cũng nên chỉnh sửa lại hàm check. Function Check\_Object(obj, arr) bLastCheck=true iSize=ubound(arr) for i=0 to iSize-1 sExpected=obj.GetTOProperty(sProp) sReal=obj.GetROProperty(sProp) if sExpected <> sReal then Report : "Error:..." bLastCheck=false exit for end if next i End Function Register this function RegisterUserFunc "WebList", "Check", "Check\_Object"

**6. Viết thư viện hàm cho report** Sau cùng, chúng ta nên viết bộ thư viện hàm để xuất report ở dạng file text. Để thêm phần sinh động, ta có thể xuất report dưới dạng RTF (Rich Text Format). Dưới đây là một report mẫu:

**QUY TRÌNH ỨNG DỤNG**

**1. Xây dựng thư viện hàm** Xây dựng thư viện hàm chung cho tất cả các dự án. Bao gồm: 1. Các hàm ứng dụng. 2. Các hàm đăng ký (đăng ký sự thay đổi cho các hàm check, hàm thao tác) 3. Các hàm cho report

**2. Khởi đầu một dự án** Để bắt đầu kiểm tra tự động cho một dự án, ta nên thực hiện các bước sau: 1. Tạo một cấu trúc thư mục 2. Chèn các tập tin thư viện vào QTP. 3. Cấu hình cho QTP, cho dự án.

**3. Viết Test script** Sau khi bước 1 và 2 đã sẵn sàng, ta bắt đầu viết test script với các bước sau:

1. Thêm các đối tượng test từ ứng dụng và object repository
2. Thêm parameter vào tập tin parameter.xls.
3. Viết một test suite class và lưu dạng \*.vbs trong thư mục test scripts
4. Viết các hàm tương ứng cho các test case
- 4. Thực thi test script** Sau khi hoàn tất các test script, ta có thể chạy hàng loạt file test script hoặc chạy từng test script đơn lẻ.
- 5. Phân tích report** Phân tích các lỗi có trong report nếu có.

# Bài 11 : Thảo luận về kiểm thử hướng đối tượng

## Thảo luận và kiểm thử hướng đối tượng

### Kiểm thử phần mềm truyền thống

Hướng tiếp cận kiểm thử phần mềm truyền thống thường bắt đầu kiểm thử từ các Module chương trình độc lập sau đó mới tiến hành kiểm thử tích hợp dần các Module chương trình đó lại. Các chiến lược kiểm thử trong phần này gồm:

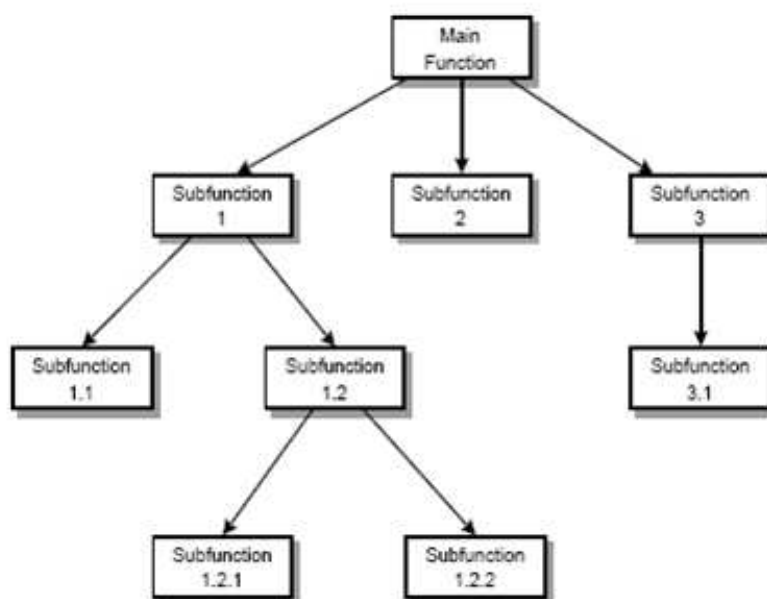
Kiểm thử đơn vị (Unit Testing)

Kiểm thử tích hợp (Integration Testing)

Kiểm thử hệ thống (System Testing)

Kiểm thử thẩm định (Validation Testing)

Nội dung chi tiết những kiểm thử trên đã có trong tài liệu [1]



Hướng tiếp cận theo Module chương trình

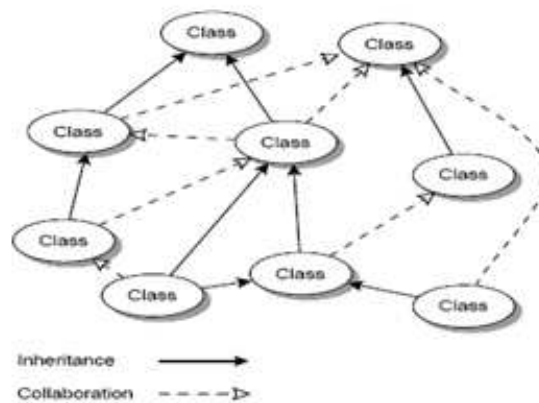
## Kiểm thử phần mềm hướng đối tượng

Ở phần mềm hướng đối tượng, chương trình được thực hiện dựa trên sự thực thi phương thức của đối tượng nào đó là thể hiện cụ thể của một lớp mỗi khi có một sự kiện được kích hoạt. Hướng tiếp cận tập trung vào kiểm thử các đối tượng thuộc các lớp và sự tương tác giữa các đối tượng thuộc các lớp khác nhau. Sự phức tạp của lớp đối tượng đó chính là: tính bao gói về mặt giữ liệu, sự thừa kế các lớp với nhau, tính đa hình... điều đó dẫn đến hoạt động kiểm thử phức tạp hơn. Có thể đưa ra 3 cấp độ khác nhau cho kiểm thử phần mềm hướng đối tượng như sau:

Kiểm thử lớp (Class Testing)

Kiểm thử tích hợp đối tượng (Object Integration Testing)

Kiểm thử toàn bộ hệ thống (System Testing)



Hướng tiếp cận theo Class

## Kiểm thử tích hợp hướng đối tượng (Object Oriented Integration Testing)

### Định nghĩa

Kiểm thử tích hợp hướng đối tượng (OOTT) là kiểm thử tập hợp các thành phần hướng đối tượng (set of object oriented components).

Như vậy, có thể hiểu OOTT là kiểm thử sự tương tác giữa các thành phần hướng đối tượng. Hiểu các thành phần hướng đối tượng theo nghĩa hẹp là các đối tượng sinh ra bởi mã lệnh chương trình, theo nghĩa rộng là các đối tượng của chương trình phần mềm và các thiết bị phần cứng khác hỗ trợ việc thực thi một phiên làm việc trong hệ thống. Ví dụ, hệ thống thanh toán thẻ rút tiền tự động ATM (Automated Teller Machine), các

thành phần hướng đối tượng được hiểu là các đối tượng chương trình phần mềm và thiết bị của máy ATM.

Theo sự phân cấp độ trong mục 1.2, kiểm thử tích hợp được thực hiện sau hoạt động kiểm thử lớp (kiểm thử đối tượng), các thành phần lỗi độc lập được tích hợp lại. Mục đích kiểm thử tích hợp là tìm ra các khiếm khuyết phát sinh khi những thành phần lỗi ảnh hưởng lẫn nhau trong một trường hợp cụ thể.

OOTT kiểm thử sự tương tác giữa hai đối tượng nảy sinh khi một đối tượng gọi phương thức của các đối tượng khác bằng cơ chế gửi thông báo. Những đối tượng này cần phải là thể hiện của các lớp khác nhau. Nếu các đối tượng đều là thể hiện của cùng một lớp, khi đó kiểm thử được đề nghị là kiểm thử lớp (kiểm thử đối tượng).

Phần mềm hướng đối tượng được phát triển một cách gia tăng với vòng đời của việc lập kế hoạch, phân tích, thiết kế, thực hiện và kiểm thử. Kiểm thử tích hợp có vai trò đặc biệt ở đây khi nó được thực hiện sau mỗi bước phát triển.

### **Một số điểm khác biệt trong phần mềm hướng đối tượng**

Phần mềm hướng đối tượng có cấu trúc và hành vi khác so với phần mềm thực hiện bằng ngôn ngữ hướng thủ tục. Phần mềm hướng đối tượng không phân rã chức năng trong các thủ tục riêng biệt, nó có các đối tượng và các lớp, sự tương tác với nhau thông qua gửi thông báo và dùng chung các định nghĩa thông qua sự thừa kế. Một đối tượng có thể ở trong một vài trạng thái khác nhau và có một số quan hệ với các đối tượng khác. Đối tượng phản hồi với một thông báo như thế nào còn phụ thuộc vào trạng thái hiện tại của nó.

Phần mềm hướng đối tượng không được phát triển với vòng đời phát triển theo mô hình thác nước truyền thống. Nó được phát triển tăng trưởng với chu kỳ có tính hồi quy của việc phân tích, thiết kế, thực hiện và kiểm thử. Những sự khác nhau căn bản này có ảnh hưởng lớn đến hoạt động kiểm thử.

Các ngôn ngữ lập trình hướng đối tượng đều trả lại điều khiển tới lời gọi đối tượng khi một thông báo được kết thúc. Không có vết thực thi đơn (single execution trace) trong phần mềm hướng đối tượng, một vết thực thi (execution trace) có thể được chia thành nhiều vết thực thi nhỏ hơn, quá trình cứ như vậy...

### **Những khó khăn cho hoạt động kiểm thử**

Không có sự định nghĩa rõ ràng về cấu trúc tích hợp, không có cây quyết định thứ tự việc kiểm thử tích hợp các đối tượng bởi có thể không có một thành phần ở mức cao nhất hoặc mức thấp nhất trong phần mềm hướng đối tượng.

Kiểm thử phức tạp hơn nhiều so với kiểm thử phần mềm truyền thống. Các đối tượng giao tiếp và ảnh hưởng lẫn nhau thông qua các thông báo. Một thông báo thay đổi trạng thái của một đối tượng. Một đối tượng phản hồi với một thông báo như thế nào là phụ thuộc vào trạng thái của nó. Các đối tượng luôn được tạo ra và bị bỏ qua, những quan hệ qua lại giữa chúng luôn được tạo dần lên trong suốt thời gian thực thi chương trình. Những tương tác khả thi giữa các đối tượng (thành phần hướng đối tượng) tăng theo cấp số 2 khi đem so sánh với phần mềm truyền thống như thể hiện minh họa trong hình 3.

a) Sự tương tác giữa các đối tượng b) Quan hệ giữa các Module

So sánh mức độ phức tạp giữa phần mềm hướng đối tượng

và truyền thống

Sự phức tạp đó dẫn đến không có khả năng kiểm thử hết được tất cả các tương tác giữa các đối tượng, chi phí dành cho kiểm thử là quá lớn (cả về thời gian và công sức). Để giảm tối thiểu chi phí, người kiểm thử (Tester) cần phải chọn phương pháp “*thông minh*”. Ví dụ xét sự thừa kế lớp, giả sử tester đã kiểm thử sự tương tác giữa hai đối tượng A<sub>1</sub> và B<sub>1</sub>, nếu có đối tượng thứ 3 là A<sub>2</sub> được thừa kế từ A<sub>1</sub>. Khi kiểm thử đối tượng A<sub>2</sub>, tester không phải kiểm thử đối tượng A<sub>1</sub> nữa mà chỉ phải kiểm thử những phương thức mới và đã được định nghĩa lại.

OOTT sau mỗi lần tăng trưởng có thể làm tăng thêm một chi phí rất lớn. Có nhiều chiến lược khác nhau mà có thể làm tối thiểu chi phí này, một số câu hỏi đặt ra là:

Việc kiểm thử bắt đầu từ đâu ?

Việc kiểm thử kết thúc ở đâu ?

Hoạt động kiểm thử như thế nào ?

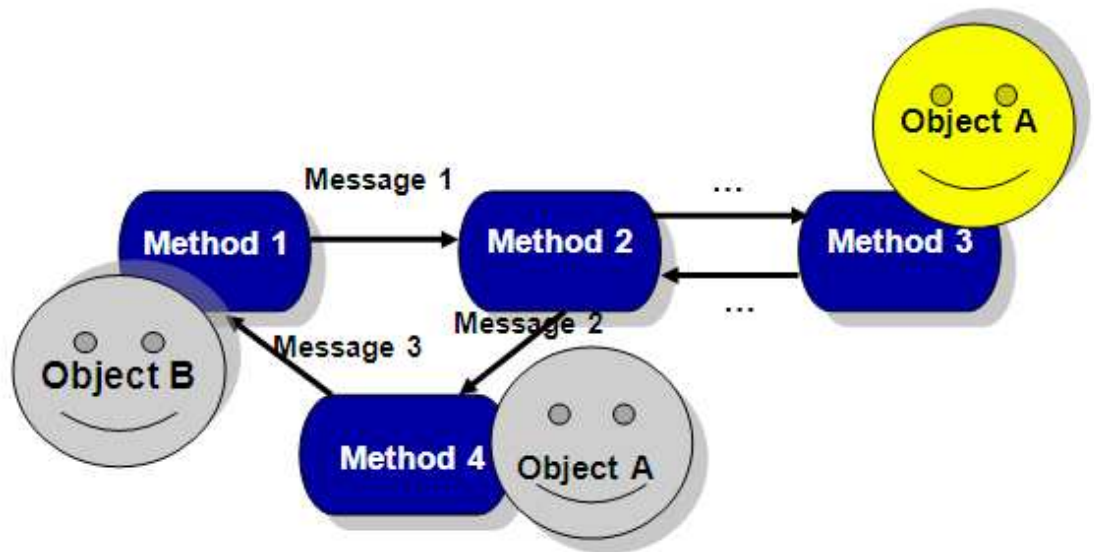
Kiểm thử cái gì ?,vv

### **Lựa chọn ca kiểm thử**

Ca kiểm thử có thể rất nhiều khi có nhiều tương tác phức tạp giữa các đối tượng, câu hỏi đặt ra là tester phải chọn được ca kiểm thử tốt. Một ca kiểm thử được coi là “*tốt*” cần thỏa mãn các yêu cầu sau: Có hiệu quả cao trong việc phát hiện lỗi, không rườm rà, cần phải là “*tốt nhất*” trong số các kỹ thuật cùng loại áp dụng, không đơn giản và cũng không phức tạp,v.v

Giải pháp (Solution) là lựa chọn ca kiểm thử thỏa mãn được các yêu cầu về sự bao phủ (Covering) các đường cơ bản trong biểu đồ thông báo lớp (Class message diagram). Một

lược đồ thông báo lớp bao gồm các nút và các cung, các nút (Node) thể hiện các phương thức và các cung thể hiện thông báo giữa hai phương thức như chỉ ra trong hình 4.



### Biểu đồ thông báo lớp

Có nhiều tiêu chuẩn về sự bao phủ trong phần mềm hướng đối tượng, bài viết này đề xuất đến 2 tiêu chuẩn sau:

1. Bao phủ phương thức- Tất cả các nút phương thức của một lược đồ thông báo lớp phải được viếng thăm ít nhất một lần, nghĩa là mỗi phương thức trong hệ thống phải được thực hiện ít nhất một lần.
2. Bao phủ thông báo – Các thông báo có thể được chia thành các phần theo nguồn của thông báo. Ít nhất một thông báo từ mỗi nguồn cần phải được thực hiện.

Có thể sẽ không khả thi để bao phủ và thực hiện tất cả các ca kiểm thử. Tuy nhiên, các ca kiểm thử quan trọng và then chốt nhất cần phải được thực hiện. Một phân tích những ca kiểm thử cần phải được thực hiện để phát hiện ra các ca kiểm thử đó. Sự lựa chọn các ca kiểm thử có thể dựa vào việc trả lời các câu hỏi sau:

1. Ca kiểm thử quan trọng như thế nào đối với khách hàng ?
2. Độ sâu vết của ca kiểm thử trong phần mềm như thế nào ?,v.v



## Một số kỹ thuật kiểm thử tích hợp

Nhiều kỹ thuật kiểm thử có thể được sử dụng trong suốt quá trình kiểm thử tích hợp. Các kỹ thuật kiểm thử khác nhau tìm ra sự khác biệt khác nhau của các lỗi, một số kỹ thuật phổ biến là:

1. Kiểm thử sự phân lớp (Classification Testing) - Kiểm thử mối quan hệ giữa các thể hiện của các lớp.
2. Kiểm thử kịch bản (Scenario Testing) - Kiểm thử sự tương tác của các đối tượng khác nhau mà đáp ứng được sự mong đợi của một ca kiểm thử đã được đặc tả.
3. Kiểm thử chấp nhận (Acceptance Testing) - Tìm ra các lỗi và các mâu thuẫn trong sự tích hợp của các thành phần.
4. Kiểm thử dựa trên sự kiện (Event – Based Testing) - Tìm ra các luồng điều khiển mà có thể không được thực hiện.
5. Kiểm thử dựa trên UML (UML – Based Testing) - Kết hợp đặc tả phân tích thiết kế phần mềm bằng ngôn ngữ mô hình hoá thống nhất UML (Unified Modeling Language) bởi các biểu đồ: biểu đồ ca sử dụng (use case diagram), biểu đồ lớp (class diagram), biểu đồ tuần tự (sequence diagram), v.v. với công cụ kiểm thử Rational Rose để sinh mã xuôi và ngược.

## Kiểm thử kịch bản

Phần này bài viết sẽ thảo luận chi tiết hơn về 2 kỹ thuật kiểm thử kịch bản.

### Method-Message Path (MM-Path)

MM-Path là một chuỗi các sự kiện thực hiện phương thức được liên kết bởi các thông báo như trong hình 5.

**Object 3 Method 1 Method 2 Method 3 Method 1 Method 2 Object 2 3 Object 1 Method 3 Method 2 Method 1 1 Input port event B Output port event B Output port event A Input port event A 2**

### MM-PathMessagesChú thích

Lược đồ các đường Method-Message

Một MM-Path bắt đầu với một phương thức và kết thúc khi nó gặp một phương thức mà không gọi một phương thức khác. Sự thực hiện trong phần mềm hướng đối tượng được bắt đầu bởi một sự kiện, sự kiện này có thể được hiểu như một cổng vào cho phần mềm (Input port event), ví dụ sự kiện cổng vào B trong hình vẽ ở trên. Sự kiện cổng vào khởi sự (gây ra) một chuỗi thông báo phương thức của một MM-Path. Một MM-Path kết thúc

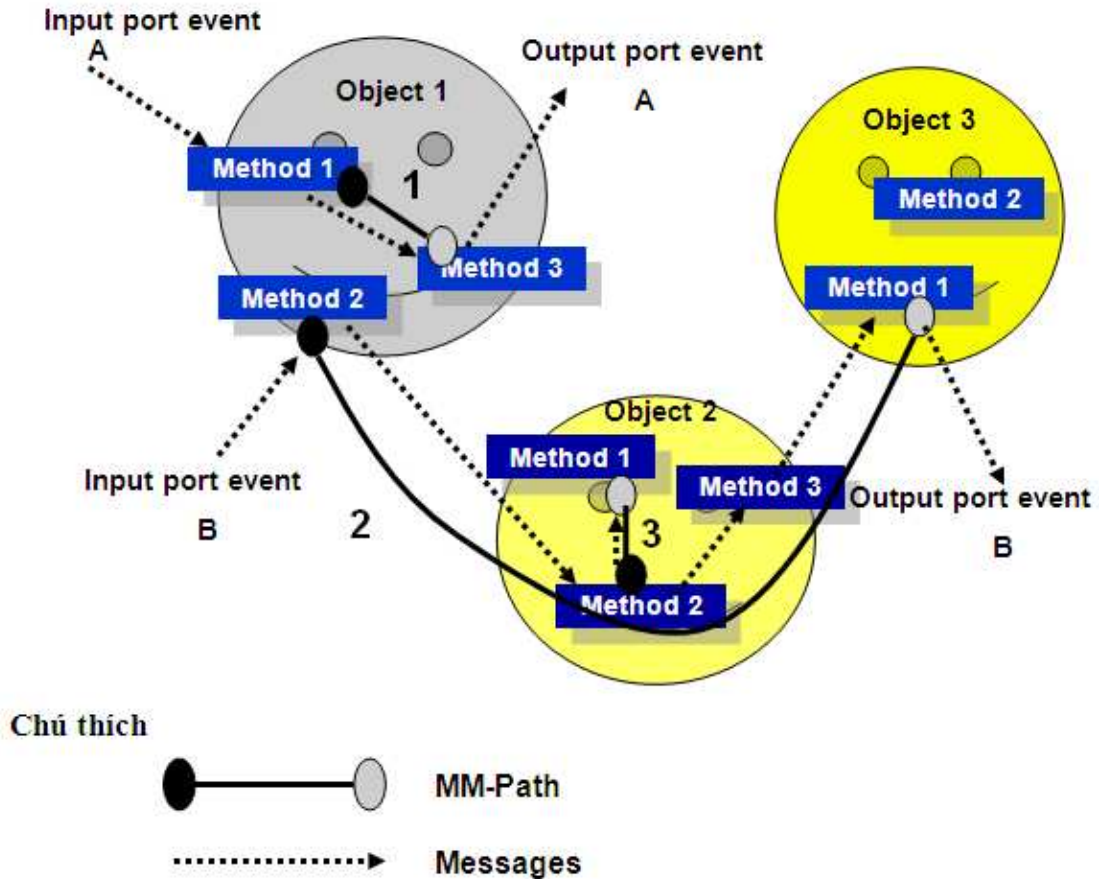
với một sự kiện công ra (Output port event), sự kiện này có thể khởi sự một cửa sổ để mở hoặc thay đổi trạng thái của hệ thống.

Một MM-Path kiểm thử sự tích hợp và sự tương tác của nhiều đối tượng khác nhau, các phương thức khác nhau truyền thông báo lẫn nhau. MM-Path là một khối nhỏ được xây dựng trong một kịch bản, bởi vậy kiểm thử MM-Path chỉ dừng lại ở kiểm thử tích hợp chứ không thể là kiểm thử hệ thống.

### **Atomic System Function (ASF)**

Một ASF là một sự kiện công vào được theo bởi một tập các MM-Path và kết thúc với nhiều sự kiện công ra. Có thể xem một ASF như một MM-Path bao gồm nhiều MM-Path khác.

Sự khác nhau giữa một ASF và một MM-Path là một ASF là một chức năng căn bản nhìn thấy được ở cấp độ hệ thống. Một ASF là một điểm gặp nhau của kiểm thử tích hợp và kiểm thử hệ thống. Hình 6 dưới đây là ví dụ về một ASF của một phiên giao dịch kiểm tra số PIN (Personal Identification Number) trên một thẻ (Card) rút tiền của khách hàng (Customer) tại một máy ATM.



## 6. ATM hiển thị một thông báo hỏi khách hàng số tiền muốn rút

Một phiên giao dịch kiểm tra PIN đơn giản dùng cho ASF trên đây là một phần nhỏ của cả một phiên giao dịch ATM lớn hơn. Nó kiểm thử sự tương tác của một phần nhỏ trong hệ thống với một số đối tượng có liên quan. Đó là lý do đề nghị của việc kiểm thử tích hợp. Khi Tester kiểm thử toàn bộ phiên giao dịch ATM, lúc đó Tester sẽ kiểm thử hệ thống.

### **Ca kiểm thử**

Việc thực hiện ca kiểm thử được thực hiện bình thường trong khi kiểm thử toàn bộ hệ thống. Với ASF, ca kiểm thử là một phần của một ca kiểm thử lớn hơn được đề nghị là kiểm thử tích hợp. Ví dụ, ca kiểm thử giao thức truyền thông xuyên suốt tất cả các tầng trong mô hình OSI được đề nghị là kiểm thử hệ thống. Một ca sử dụng mà kiểm thử sự truyền thông giữa giao thức TCP và IP được đề nghị là kiểm thử tích hợp.

Ca kiểm thử là một cuộc đối thoại giữa hệ thống và tác nhân bên trong. Tác nhân có thể là một hệ thống khác hoặc con người. Một kịch bản là một thể hiện rõ ràng của một ca kiểm thử. Một ca kiểm thử xác định rõ các bước bên trong đó, nhưng kịch bản xác định rõ sự việc xảy ra ở mỗi bước, các thông tin đầu vào và các thông tin đầu ra.

Kiểm thử MM-Path và ASF thuộc kiểm thử hộp đen của phần mềm hướng đối tượng bởi Tester sẽ tiến hành việc kiểm thử từ khung nhìn của người dùng và chỉ quan tâm đến thông tin đầu vào và đầu ra của hệ thống. Một ca kiểm thử cho số PIN dùng cho ASF có thể được xác định như sau:

#### Ca kiểm thử 1

Input: Khách hàng đút thẻ rút tiền của họ vào máy ATM

Output: Hệ thống hiển thị thông báo trên màn hình máy ATM hỏi khách hàng số PIN

#### Ca kiểm thử 2

Input: Khách hàng nhập số PIN của họ

Output: Hệ thống hiển thị thông báo trên màn hình máy ATM hỏi khách hàng về số tiền mà khách hàng muốn rút.

Như đã đề cập ở trên, kiểm thử phần mềm hướng đối tượng là rất phức tạp. Tester không thể thực hiện được tất cả các ca kiểm thử. Một cách để giải quyết vấn đề này là thực hiện kiểm thử sử dụng sắc xuất thống kê.

## Kết luận

Kiểm thử tích hợp hướng đối tượng có vai trò quan trọng trong việc phát triển phần mềm hướng đối tượng. Phần mềm hướng đối tượng được phát triển gia tăng trong các chu kỳ có tính hồi quy. Sau mỗi chu kỳ, các thành phần hướng đối tượng của phần mềm được sắp đặt cùng nhau và kiểm thử tích hợp được thực hiện.

Không có một cấu trúc rõ ràng trong phần mềm hướng đối tượng. Có thể không có thành phần ở mức cao nhất hoặc mức thấp nhất. Các thành phần có thể gửi thông báo tới các thành phần khác. Mỗi thông báo làm thay đổi trạng thái của thành phần, điều này gần như dẫn đến một số lượng vô tận các ca kiểm thử. Có quá nhiều trạng thái có thể xảy ra và sự tương tác lẫn nhau trong hệ thống. Sự phức tạp này dẫn đến một chi phí rất lớn trong việc kiểm thử.

Kiểm thử kịch bản là một kỹ thuật tốt cho việc kiểm thử phần mềm hướng đối tượng. Bài viết đã trình bày được các vấn đề liên quan đến kiểm thử phần mềm hướng đối tượng và đề xuất kỹ thuật kịch bản như một phương pháp luận cho kiểm thử tích hợp phần mềm hướng đối tượng.

# **Bài 12 : Bài tập**

## **Bài Tập**

### **THỰC HÀNH LẬP KẾ HOẠCH TEST**

Bài 1: Lập kế hoạch để kiểm thử phần mềm TestPro (trộn đề thi) với nguồn nhân lực là các thành viên trong lớp (coi mỗi thành viên là một tester)

Bài 2: Lập kế hoạch test phần mềm TestOnline (thi trắc nghiệm trực tuyến) với nguồn nhân lực là các thành viên trong lớp (coi mỗi thành viên là một tester)

Bài 3: Lập kế hoạch để kiểm thử website của khoa CNTT – ĐHSPKT Hưng Yên với nguồn nhân lực là các thành viên trong lớp (coi mỗi thành viên là một tester)

### **THỰC HÀNH XÂY DỰNG CÁC TEST CASE**

Bài 1: Xây dựng test case cho form đăng nhập (cung cấp kèm theo)

Bài 2: Xây dựng test case cho phần mềm TestPro (theo phân công trong pha lập kế hoạch)

Bài 3: Xây dựng test case cho website của khoa CNTT – ĐHSPKT Hưng Yên

### **THỰC HÀNH THỰC THI TEST VÀ VIẾT BÁO CÁO**

**Bài 1:** Thực hiện kiểm thử với những test case đã xây dựng cho form đăng nhập. Từ đó viết báo cáo và đánh giá kết quả pass và fail

**Bài 2:** Thực hiện kiểm thử với những test case đã xây dựng cho phần mềm TestPro. Từ đó viết báo cáo và đánh giá kết quả pass và fail

**Bài 3:** Thực hiện kiểm thử với những test case đã xây dựng cho website của khoa CNTT – ĐH SPKT Hưng Yên. Từ đó viết báo cáo và đánh giá kết quả pass và fail

### **THỰC HÀNH TEST HỒI QUY TRÊN QUICK TEST PROFESSIONAL**

**Bài 1:** Thực hiện kiểm thử form đăng nhập Form\_Login\_V1 (cung cấp kèm theo) theo bản đặc tả. Đưa ra báo cáo và đoạn script trên Quick Test Pro.

Thực hiện kiểm tra hồi quy trên Form\_Login\_V2. Đưa ra báo cáo về những lỗi đã được giải quyết và những lỗi còn tồn đọng (dựa trên script đã của Form\_Login\_V1)

**Bài 2:** Thực hiện kiểm thử (viết test case và lựa chọn test case kiểm thử tự động) cho phần mềm Quick test professional

**Bài 3:** Thực hiện kiểm tra hồi quy trên 2 phiên bản 1 và 2 của phần mềm TestPro (Phần mềm trộn đề thi trắc nghiệm)

## **THỰC HÀNH NGÔN NGỮ VB SCRIPT TRÊN QUICK TEST PROFESSIONAL**

**Bài 1:** Thực hiện kiểm thử 5 test case tùy chọn (2 test case fail và 3 test case pass) của bài 2 – chương 14. Hiệu chỉnh đoạn mã script phát sinh để với phiên bản sau có thể thực hiện test hồi quy tự động cho test case này.

**Bài 2:** Thực hiện kiểm thử khả năng chịu tải của phần mềm NetofSchool nhằm kiểm tra khả năng truy xuất đồng thời từ nhiều máy tới phần mềm. (thực hiện bằng cách hiệu chỉnh mã Script để lặp lại quá trình truy xuất)

## **THỰC HÀNH NGÔN NGỮ VB SCRIPT TRÊN QUICK TEST PROFESSIONAL**

**Bài 1:** Lựa chọn những test case nên được kiểm thử tự động với bài 2 – chương 14. Sau đó, thực hiện hiệu chỉnh mã script phát sinh nhằm giảm thiểu khả năng phải can thiệp của người kiểm thử viên với những test case này cho trường hợp kiểm thử hồi quy với phiên bản sau.

**Bài 2:** Lựa chọn một site của một website tùy ý, xây dựng và thực thi các test case cho site này, hiệu chỉnh script nhằm đạt hiệu quả cao nhất cho những lần test hồi quy với những phiên bản sau.

## **THỰC HÀNH TEST HIỆU NĂNG TRÊN LOAD RUNNER**

**Bài 1:** Thực hiện viết test case để kiểm tra hiệu năng cho phần mềm thi trắc nghiệm TestOnline của khoa CNTT – ĐHSPKT Hưng Yên

## **THỰC HÀNH TEST HIỆU NĂNG TRÊN LOAD RUNNER**

**Bài 1:** Thực hiện các test case kiểm tra hiệu năng cho phần mềm TestOnline trên Load Runner, xử lý các tham số, các đoạn script phát sinh để thuận lợi cho quá trình kiểm thử.

## THỰC HÀNH HIỆU NĂNG TRÊN LOAD RUNNER

**Bài 1:** Thực hiện viết test case và kiểm thử hiệu năng cho website của khoa công nghệ thông tin trường ĐHSPKT Hưng Yên.



## Tham gia đóng góp

Tài liệu: Đảm bảo chất lượng phần mềm

Biên tập bởi: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/c/4c771e16>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Những lỗi (bug) phần mềm nghiêm trọng trong lịch sử

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/5e6c6530>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Lỗi (bug) là gì

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/32455ae1>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Tại sao lỗi xuất hiện

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/bcc429a3>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Quy trình phát triển phần mềm

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/77933154>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Thực trạng của quá trình kiểm thử phần mềm

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/f4398e86>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Các định nghĩa và thuật ngữ kiểm thử phần mềm

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/7d808525>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Quá trình nghiên cứu bản đặc tả phần mềm

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/c0a15853>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Phương pháp kiểm thử

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/cedd3f91>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Thiết kế trường hợp kiểm thử

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/1b05b4be>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Chiến lược kiểm thử

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/125ed1fd>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Test và kiểm tra

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/896649ef>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm tra miền

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/36f4036e>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm thử cấu hình

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/26cc3fc3>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Cấu Hình Hardware

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/569e16fd>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm thử khả năng tương thích

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/c68f9b93>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm thử Foreign – Language

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/86b9bc00>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm thử khả năng tiện dụng (tính khả dụng)

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/d1c751eb>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm thử tài liệu

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/6051fe94>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm thử khả năng bảo mật phần mềm

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/1ef8db18>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Các giai đoạn kiểm thử

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/b2b4cf00>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Mô hình kiểm tra phần mềm TMM

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/e6122a63>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Quy trình kiểm tra

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/6b2b0707>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Mô tả

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/f36ff4b5>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Lập kế hoạch

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/fd3a82da>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Các yêu cầu test

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/888ee726>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Một ví dụ

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/e58fbd7b>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Test

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/563d07d4>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Viết và theo dõi các trường hợp kiểm thử

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/b59134a2>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Thực hiện test, viết báo cáo và đánh giá kết quả

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/2b7788af>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Lợi ích của quá trình tự động hóa và các công cụ

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/d7e0d02e>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Giới thiệu về công cụ KTTĐ

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/b20e533a>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Chương Trình TestTrack Pro Version 6.0.3

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/c88ae851>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm tra hiệu năng phần mềm với LoadRunner 8.1

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/4ccecd38>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Esstimate

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/b86af18e>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểm thử tự động với phần mềm

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/c81c6426>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Thảo luận và kiểm thử hướng đối tượng

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/b684d445>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Bài Tập

Các tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://www.voer.edu.vn/m/0a42e0f6>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

## **Chương trình Thư viện Học liệu Mở Việt Nam**

Chương trình Thư viện Học liệu Mở Việt Nam (Vietnam Open Educational Resources – VOER) được hỗ trợ bởi Quỹ Việt Nam. Mục tiêu của chương trình là xây dựng kho Tài nguyên giáo dục Mở miễn phí của người Việt và cho người Việt, có nội dung phong phú. Các nội dung đều tuân thủ Giấy phép Creative Commons Attribution (CC-by) 4.0 do đó các nội dung đều có thể được sử dụng, tái sử dụng và truy nhập miễn phí trước hết trong môi trường giảng dạy, học tập và nghiên cứu sau đó cho toàn xã hội.

Với sự hỗ trợ của Quỹ Việt Nam, Thư viện Học liệu Mở Việt Nam (VOER) đã trở thành một cổng thông tin chính cho các sinh viên và giảng viên trong và ngoài Việt Nam. Mỗi ngày có hàng chục nghìn lượt truy cập VOER ([www.voer.edu.vn](http://www.voer.edu.vn)) để nghiên cứu, học tập và tải tài liệu giảng dạy về. Với hàng chục nghìn module kiến thức từ hàng nghìn tác giả khác nhau đóng góp, Thư Viện Học liệu Mở Việt Nam là một kho tàng tài liệu khổng lồ, nội dung phong phú phục vụ cho tất cả các nhu cầu học tập, nghiên cứu của độc giả.

Nguồn tài liệu mở phong phú có trên VOER có được là do sự chia sẻ tự nguyện của các tác giả trong và ngoài nước. Quá trình chia sẻ tài liệu trên VOER trở lên dễ dàng như đếm 1, 2, 3 nhờ vào sức mạnh của nền tảng Hanoi Spring.

Hanoi Spring là một nền tảng công nghệ tiên tiến được thiết kế cho phép công chúng dễ dàng chia sẻ tài liệu giảng dạy, học tập cũng như chủ động phát triển chương trình giảng dạy dựa trên khái niệm về học liệu mở (OCW) và tài nguyên giáo dục mở (OER). Khái niệm chia sẻ tri thức có tính cách mạng đã được khởi xướng và phát triển tiên phong bởi Đại học MIT và Đại học Rice Hoa Kỳ trong vòng một thập kỷ qua. Kể từ đó, phong trào Tài nguyên Giáo dục Mở đã phát triển nhanh chóng, được UNESCO hỗ trợ và được chấp nhận như một chương trình chính thức ở nhiều nước trên thế giới.