

## Lời nói đầu

Để có được thành quả học tập như ngày hôm nay, ngoài sự nỗ lực phấn đấu không ngừng của bản thân thì một phần không nhỏ đóng góp nên thành công ấy là nhờ sự hướng dẫn, dạy dỗ của các thầy cô trong viện CNTT&TT nói riêng và trong trường ĐHBK Hà Nội nói chung trong suốt gần năm năm em học tập và nghiên cứu tại DHBK Hà Nội.

Em xin gửi lời cảm ơn chân thành đến thầy Ths. Thạc Bình Cường – giảng viên bộ môn Công nghệ phần mềm, Viện Công nghệ thông tin và Truyền thông, trường Đại Học Bách Khoa Hà Nội đã định hướng, hướng dẫn và chi bảo tận tình trong quá trình em làm báo cáo thực tập tốt nghiệp này. Em xin chúc thầy và gia đình luôn mạnh khỏe và hạnh phúc.

Cuối cùng, em xin được cảm ơn đến gia đình, ta bè đã động viên, chăm sóc, đóng góp ý kiến và giúp đỡ trong quá trình học tập, nghiên cứu và hoàn thành báo cáo thực tập tốt nghiệp này.

Tuy nhiên, do thời gian và trình độ có hạn nên báo cáo không thể tránh khỏi những thiếu sót. Chính vì vậy, em rất mong có được sự góp ý từ các thầy cô giáo và toàn thể các ta.

Sinh viên thực hiện

Nguyễn Vương Quyền

## Lí do lựa chọn đề tài

Ngày nay công nghệ thông tin đang ngày càng phát triển nhanh chóng, kéo theo đó là hệ thống mạng và các phần mềm cũng già tăng cả về số lượng theo quy mô rộng và cả về chất lượng phần mềm theo chiều sâu. Nhưng cũng từ đó đã nảy sinh ra nhiều vấn đề về lỗi hỏng hóc phần mềm không đáng có gây ra các ảnh hưởng nghiêm trọng đến xã hội, kinh tế,...Những lỗi này có thể do tự bản thân phần mềm bị hỏng do không được kiểm duyệt kỹ lưỡng trước khi đưa cho người dùng cuối hay cũng có thể do có người cố tình phá hoại nhằm đánh cắp thông tin cá nhân như mã số tài khoản ngân hàng, số điện thoại, danh bạ, tin nhắn,...Những vấn đề nan giải và cấp thiết này càng có xu hướng mở rộng trong các năm gần đây, điển hình như sự cố máy tính Y2K năm 2000 làm tê liệt nhiều hệ thống máy tính lớn hay như càng có nhiều loại virus phá hoại mới xuất hiện, tấn công vào các lỗ hỏng bảo mật phần mềm làm tê liệt nhiều hệ thống phần mềm và phần cứng. Từ đây ta dễ dàng nhận ra là mặc dù phần mềm phát triển ngày càng phức tạp nhưng vẫn đề chất lượng vẫn là một dấu hỏi lớn cần xem xét cẩn thận.

Do đó yêu cầu đặt ra là cần có công tác kiểm thử phần mềm thật kỹ lưỡng nhằm ngăn chặn các lỗi hay hỏng hóc còn tiềm tàng bên trong phần mềm mà ta chưa kịp nhận ra. Tuy nhiên vì phần mềm ngày càng lớn, hàng nghìn module, có thể do cả một công ty hàng nghìn người phát triển vì vậy để kiểm thử được một phần mềm lớn như vậy sẽ tốn rất nhiều công sức và thời gian nếu làm thủ công, chưa kể đến chất lượng kiểm thử sẽ không cao và thật chính xác phù hợp cho yêu cầu. Theo nhiều tính toán thì công việc kiểm thử đóng vai trò hết sức quan trọng trong quy trình phát triển phần mềm, nó đóng góp tới 40% tổng toàn bộ chi phí cho việc sản xuất phần mềm. Vì vậy cần có các hệ thống kiểm thử phần mềm một cách tự động cho phép ta thực hiện được các công việc một cách nhanh chóng và độ an toàn, chính xác cao nhất có thể. Và đó chính là lí do em lựa chọn đề tài này để nghiên cứu, tìm hiểu và đề ra các giải pháp mới để cải tiến các quy trình kiểm thử như hiện nay sao cho có năng suất cao nhất.

# Chương I Khái quát về phần mềm và kiểm thử phần mềm

## 1.Tổng quan về phần mềm

### 1.1. Định nghĩa

Có nhiều định nghĩa về phần mềm, sau đây là một ví dụ

Phần mềm máy tính (tiếng Anh: Computer Software) hay gọi tắt là Phần mềm (Software) là một tập hợp những câu lệnh hoặc chỉ thị (Instruction) được viết bằng một hoặc nhiều ngôn ngữ lập trình theo một trật tự xác định, và các dữ liệu hay tài liệu liên quan nhằm tự động thực hiện một số nhiệm vụ hay chức năng hoặc giải quyết một vấn đề cụ thể nào đó.( Theo Wikipedia)

### 1.2.Phân loại phần mềm

Có nhiều cách thức phân loại phần mềm, song có thể chia thành hai loại chính sau:

#### 1.2.1.Theo phương thức hoạt động

- Phần mềm hệ thống dùng để vận hành máy tính và các phần cứng máy tính. Đây là các loại phần mềm mà hệ điều hành liên lạc với chúng để điều khiển và quản lý các thiết bị phần cứng.
- Phần mềm ứng dụng: để người sử dụng có thể hoàn thành một hay nhiều công việc nào đó.
- Các phần mềm chuyên dịch bao gồm trình biên dịch và trình thông dịch.
- Các nền tảng công nghệ như .NET, 1C:DOANH NGHIỆP...

#### 1.2.2.Theo khả năng ứng dụng

- Phần mềm thời gian thực (các PM anti-virus, PM chat,...)
- PM giải trí (Game,...)
- PM nhúng: chạy trên các thiết bị đặc thù như điện thoại di động, TV, máy lạnh,...
- PM phân tán: chạy trên nhiều thiết bị, phối hợp hoạt động đồng thời với nhau

### 1.3. Quy trình phát triển phần mềm

#### 1.3.1.Tổng quan

Cũng như mọi ngành sản xuất khác, qui trình là một trong những yếu tố cực kỳ quan trọng đem lại sự thành công cho các nhà sản xuất phần mềm, nó giúp cho mọi thành viên trong dự án từ người cũ đến người mới, trong hay ngoài công ty đều có thể xử lý đồng bộ công việc tương ứng vị trí của mình thông qua cách thức chung của công ty, hay ít nhất ở cấp độ dự án.Có thể nói qui trình phát triển/xây dựng phần mềm (Software Development/Engineering Process - SEP) có tính chất quyết định để tạo ra sản phẩm chất lượng tốt với chi phí thấp và năng suất cao.

#### Vậy qui trình là gì?

Qui trình có thể hiểu là phương pháp thực hiện hoặc sản xuất ra sản phẩm. Tương tự như vậy, SEP chính là phương pháp phát triển hay sản xuất ra sản phẩm phần mềm. Thông thường một qui trình bao gồm những giai đoạn cơ bản sau:

- Đặc tả yêu cầu (Requirements Specification): chỉ ra những “đòi hỏi” cho cả các yêu cầu chức năng và phi chức năng.
- Phát triển phần mềm (Development): tạo ra phần mềm thỏa mãn các yêu cầu được chỉ ra trong

"Đặc tả yêu cầu".

- Kiểm thử phần mềm (Validation/Testing): để bảo đảm phần mềm sản xuất ra đáp ứng những "đòi hỏi" được chỉ ra trong "Đặc tả yêu cầu".
- Thay đổi phần mềm (Evolution): đáp ứng nhu cầu thay đổi của khách hàng.

Tùy theo mô hình phát triển phần mềm, các nhóm công việc được triển khai theo những cách khác nhau. Để sản xuất cùng một sản phẩm phần mềm người ta có thể dùng các mô hình khác nhau. Tuy nhiên không phải tất cả các mô hình đều thích hợp cho mọi ứng dụng.

### 1.3.2. Các mô hình SEP

Có khá nhiều mô hình SLC khác nhau, trong đó một số được ứng dụng khá phổ biến trên thế giới:

- Mô hình Waterfall (Waterfall model)
- Mô hình chữ V (V-model)
- Các mô hình nhiều phiên bản (Multi-version models)
- Mô hình mẫu (Prototype)
- Mô hình tiến hóa (Evolutionary)
- Mô hình lặp và tăng dần (Iterative and Incremental)
- Mô hình phát triển ứng dụng nhanh (RAD)
- Mô hình xoắn ốc (Spiral)
- Mô hình phát triển dựa trên kiểm thử (Test Driven Development-TDD)

### 1.3.3. Mô hình phát triển dựa trên kiểm thử (TDD)

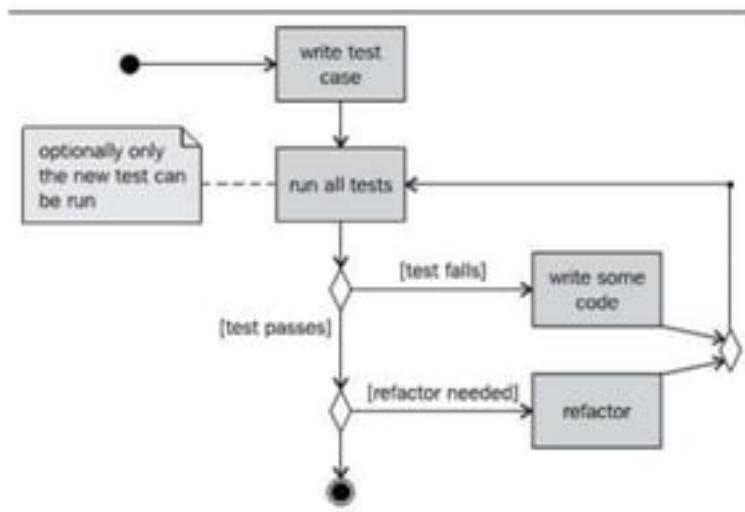
a) Định nghĩa:

TDD là một phương pháp tiếp cận mới nhằm cải tiến quy trình phát triển phần mềm trong đó kết hợp phương pháp Phát triển kiểm thử trước (Test First Development) và phương pháp Điều chỉnh lại mã nguồn (Refactoring). Mục tiêu quan trọng nhất của TDD là viết mã nguồn sáng sủa, rõ ràng và có thể chạy được.

b) Các cải tiến của TDD

-TDD hoàn toàn thay đổi cách phát triển truyền thống. Khi ta bắt đầu thực hiện một tính năng mới, câu hỏi đầu tiên đặt ra là liệu thiết kế hiện tại có phải là thiết kế tốt nhất cho phép ta thực hiện các chức năng hay không. Nếu có, ta tiến hành thông qua một phương pháp Phát triển kiểm thử trước TFD. Nếu không, ta điều chỉnh lại nó một cách cẩn thận để thay đổi riêng phần thiết kế bị ảnh hưởng bởi tính năng mới, cho phép ta dễ dàng bổ sung các tính năng có thể. Kết quả là chất lượng thiết kế của ta sẽ luôn luôn được nâng cao, do đó sẽ thuận lợi hơn khi làm việc với nó trong tương lai.

-Một giả định cơ bản của TDD là ta có sẵn một nền tảng (framework) cho kiểm thử mức đơn vị (unit-test). Những lập trình viên phần mềm theo phương pháp Agile thường sử dụng các công cụ mã nguồn mở thuộc họ xUnit, như JUnit hay VBUnit, mặc dù các công cụ thương mại cũng là những lựa chọn khả dĩ. Nếu không có những công cụ như vậy thì TDD hầu như không thể thực hiện được.



Hình 1.2. Các bước trong TDD

Hai nguyên tắc đơn giản cho TDD: Trước tiên, ta nên viết mã xử lý nghiệp vụ mới chỉ khi mẫu kiểm thử tự động thực hiện không thành công. Thứ hai, ta nên loại bỏ bất kỳ sự trùng lặp mà ta tìm thấy. Những quy tắc đơn giản:

- ✓ Thiết kế với mã nguồn mà chúng chạy được và tạo ra kết quả phản hồi giữa các quyết định.
- ✓ Tự viết các mẫu kiểm thử của riêng mình, không chờ người khác
- ✓ Môi trường phát triển phải cung cấp được kết quả nhanh với những thay đổi nhỏ (ví dụ như ta cần một trình biên dịch nhanh và chuỗi kiểm thử hồi quy (regression test))
- ✓ Thiết kế phải bao gồm những thành phần gắn kết, sự phụ thuộc lẫn nhau nhỏ (loosely coupled) để thực hiện các mẫu kiểm thử dễ dàng hơn (điều này cũng làm cho quá trình nâng cấp và bảo trì các hệ thống của ta dễ dàng hơn).

#### c) TDD và cách kiểm thử truyền thống

TDD là một kỹ thuật thiết kế với một hiệu ứng phụ là việc đảm bảo toàn bộ mã nguồn được thực hiện kiểm thử mức đơn vị. Tuy nhiên, có những điều còn quan trọng hơn cả việc thực hiện kiểm thử. Ta sẽ vẫn cần xem xét các kỹ thuật kiểm thử khác như kiểm thử chấp nhận (acceptance test) hay kiểm thử dò hỏi (investigative test) theo kiểu Agile. Ta có thể thực hiện nhiều những kiểu kiểm thử này trong dự án nếu như ta chọn làm điều đó (và ta nên làm).

Với kiểu kiểm thử truyền thống, một mẫu kiểm thử thành công sẽ tìm ra một hoặc nhiều lỗi. Tương tự với TDD, khi một mẫu kiểm thử thất bại thì ta cũng có sự tiến triển bởi vì bây giờ ta biết rằng ta cần phải giải quyết một số vấn đề. Quan trọng hơn, ta có một cách đo rõ ràng về sự thành công khi mẫu kiểm thử không thất bại nữa. Từ đó TDD làm tăng niềm tin về hệ thống đáp ứng được các yêu cầu được định nghĩa cho nó.

Như với thử nghiệm truyền thống, hệ thống càng có nhiều rủi ro lớn càng cần phải có nhiều mẫu kiểm thử được thực hiện. Với cả hai kiểu kiểm thử truyền thống và TDD ta không phản đầu cho sự hoàn hảo, thay vào đó ta kiểm thử tầm quan trọng của hệ thống. Một hiệu ứng phụ thú vị của TDD là ta đạt được 100% khi kiểm thử độ phủ mã nguồn (coverage test) - mọi dòng mã đều được kiểm thử - điều mà kiểm thử truyền thống không bảo đảm dù cho nó khuyến khích điều đó. Không có gì ngạc nhiên khi nói rằng TDD là một kỹ thuật đặc tả (specification technique), với một tác dụng phụ có giá trị là nó đem lại kết quả trong việc kiểm thử mã nguồn tốt hơn đáng kể so với các kỹ thuật truyền thống.

#### d) Tại sao nên dùng TDD ?

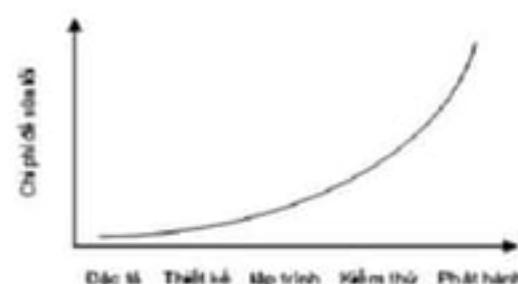
Một lợi thế đáng kể của TDD là nó cho phép ta thực hiện các bước nhỏ khi viết phần mềm. Đây là một thực tế mà người ta đã phát huy trong nhiều năm qua bởi vì nó mang lại hiệu quả nhiều hơn so với cố gắng viết mã trong những bước lớn. Ví dụ, giả sử ta thêm một số mã nguồn cho chức năng mới, biên dịch, và kiểm thử nó. Khả năng rất lớn là các kiểm thử của ta sẽ thất bại bởi những lỗi có trong mã nguồn mới. Sẽ dễ dàng hơn nhiều trong việc tìm kiếm, và sau đó sửa chữa những lỗi đó nếu ta đã viết thêm hai dòng mã mới thay vì hai nghìn dòng.

Nhiều người cho rằng các kỹ thuật Agile hoạt động rất ổn với những dự án nhỏ, cần một số ít người trong một vài tháng, nhưng chúng không hoạt động đối với những dự án thực sự lớn hơn. Tuy nhiên, điều đó không hoàn toàn đúng. Người ta đã đưa ra một bản báo cáo rằng làm việc với một hệ thống Smalltalk sử dụng hoàn toàn phương pháp hướng kiểm thử (test-driven) hết 4 năm với chi phí nhân công 40 man-year, ra kết quả gồm 250,000 dòng mã nguồn chức năng và 250,000 dòng mã kiểm thử. Có 4000 mẫu kiểm thử chạy dưới 20 phút, còn với bộ mẫu kiểm thử đầy đủ thì cần chạy vài ngày. Điều đó chứng tỏ rằng TDD vẫn hoạt động tốt với những dự án có kích thước lớn.

c) Công cụ: Các công cụ phục vụ cho TDD, thường là các nền tảng cho kiểm thử mã nguồn mức đơn vị (unit test): xUnit (NUnit, JUnit,...)

#### 1.4. Mối quan hệ giữa quy trình phát triển phần mềm và kiểm thử phần mềm

Phát triển phần mềm và kiểm thử phần mềm có mối quan hệ khăng khít với nhau. Phát triển phần mềm ngay từ những pha đầu tiên như phân tích yêu cầu, phân tích thiết kế hệ thống,... phải được tiến hành kiểm thử một cách độc lập bởi một đội ngũ có kinh nghiệm để nếu có phát hiện ra sai sót thì phải tiến hành sửa chữa kịp thời, nếu càng để về sau mới phát hiện ra lỗi thì chi phí để sửa chữa là vô cùng lớn. Đồ thị dưới đây minh họa cho điều này



Hình 1.4.

Nhìn vào mô hình ta cũng có thể dễ dàng nhận thấy là ở những pha đầu tiên của quy trình phát triển phần mềm chi phí là nhỏ không đáng kể nhưng càng để về sau thì chi phí tăng lên rất nhiều lần. Vì vậy ta không thể chủ quan mà lơ là việc kiểm thử ngay từ giai đoạn đầu của phát triển phần mềm. Điều này là cần thiết nhất là đối với các dự án phần mềm lớn của các doanh nghiệp ngày nay.

## 2. Tổng quan về kiểm thử phần mềm

### 2.1. Khái niệm

Kiểm thử phần mềm là hoạt động khảo sát thực tiễn sản phẩm hay dịch vụ phần mềm trong đúng môi trường chúng dự định sẽ được triển khai nhằm cung cấp cho những người có lợi ích liên quan những thông tin về chất lượng của sản phẩm hay dịch vụ phần mềm ấy. Mục đích của kiểm thử phần mềm là tìm ra các lỗi hay khiếm khuyết phần mềm nhằm đảm bảo hiệu quả hoạt động tối ưu

của phần mềm trong nhiều ngành khác nhau. (Wikipedia)

Software testing là một trong những kỹ thuật phần mềm “ xác minh và xác nhận ” (verification and validation hay gọi tắt là V&V). Verification (chữ V đầu tiên) là quá trình đánh giá một hệ thống hoặc thành phần để xác định xem các sản phẩm của một giai đoạn phát triển nhất định đáp ứng các điều kiện áp đặt vào lúc bắt đầu của giai đoạn đó hay không. Các hoạt động Verification bao gồm việc kiểm thử và đánh giá. Ví dụ trong phần mềm chơi game Monopoly, chúng ta có thể xác minh rằng hai người chơi không thể sở hữu cùng một nhà. Validation là quá trình đánh giá một hệ thống hoặc một thành phần trong hoặc ở cuối của quá trình phát triển để xác định xem nó đáp ứng yêu cầu quy định

Verification	Validation
<p>Qua Verification chúng ta muốn chắc chắn rằng phần mềm có các hành vi đúng như chúng ta mong đợi. Ví dụ:</p> 	<p>Qua Validation chúng ta sẽ xác nhận rằng những lỗi không đúng với yêu cầu của khách hàng sẽ không được thực hiện tiếp theo trong suốt quá trình xây dựng phần mềm. Validation luôn luôn liên quan tới việc so sánh với các yêu cầu của khách hàng. Ví dụ khách hàng yêu cầu là làm cho họ game Monopoly</p> 

Trong game Monopoly, người chơi có thể được cộng 200 điểm nếu họ hạ cánh trên sân hoặc đi qua Go nhưng người lập trình lại cài đặt là người chơi phải đi qua Go(?!).

nhưng đội ngũ phát triển lại làm dựa cho game Life vì họ nghĩ là game Life hay hơn game Monopoly như yêu cầu ban đầu (?!).

## 2.2. Vai trò của kiểm thử phần mềm

Việc tạo ra một sản phẩm phần mềm phải trải qua nhiều giai đoạn, người ta gọi là quá trình phát triển phần mềm, bắt đầu từ khi bắt đầu có ý tưởng cho đến khi đưa ra sản phẩm phần mềm thực thi. Khối lượng công việc trong từng giai đoạn của quá trình sản xuất phần mềm cũng thay đổi theo thời gian. Bảng 1.1 minh họa cụ thể hơn về điều này.

Giai đoạn	Phân tích yêu cầu	Thiết kế sơ bộ	Thiết kế chi tiết	Lập trình và kiểm thử đơn vị	Tích hợp và kiểm thử tích hợp	Kiểm thử hệ thống
Hai thập kỷ 1960 - 1970		10%		80%		10%
Thập kỷ 1980		20%		60%		20%
Thập kỷ 1990	40%		30%		30%	

Bảng 1.1 - Tỷ lệ công việc của các giai đoạn phát triển phần mềm

Như vậy, một sản phẩm phần mềm không chỉ đơn giản là các đoạn mã chương trình mà còn rất nhiều phần ẩn đằng sau nó (Hình 1.1). Vì vậy, việc mắc lỗi không chỉ xảy ra trong khi lập trình mà còn xảy ra cao hơn trong các công đoạn khác của quá trình phát triển một sản phẩm phần mềm. Việc kiểm thử cũng vì thế phải được tiến hành trong tất cả các phần tạo nên một sản phẩm phần mềm.

## 2.3. Các kỹ thuật kiểm thử phần mềm

- Kiểm thử hộp đen (Black Box testing): dùng để kiểm tra chức năng mà không xem xét mã nguồn cũng như cấu trúc chương trình bên trong. Thường kiểm thử hộp đen quan tâm nhiều đến các bộ dữ liệu kiểm thử đầu vào.
- Kiểm thử hộp trắng (White Box testing): khác với kiểm thử hộp đen, kiểm thử hộp trắng xem xét mọi module trong chương trình, các luồng thực hiện công việc để từ đó đưa ra các chiến lược kế hoạch cụ thể cho việc kiểm thử.
- Kiểm thử hộp xám (Grey Box Testing): đây là một kỹ thuật kiểm thử mới dựa trên những đặc tính của cả kiểm thử hộp đen và hộp trắng. Mục tiêu chính của kiểm thử hộp xám là kiểm thử các ứng dụng trên nền web (web based).

#### **2.4. Các giai đoạn hay cấp độ kiểm thử phần mềm**

- ✓ Kiểm thử đơn vị (Unit test): kiểm thử từng module nhỏ trong chương trình để tìm ra các lỗi và khắc phục.
- ✓ Kiểm thử tích hợp: sau khi đã thực hiện thành công kiểm thử đơn vị, ta sẽ tiến hành tích hợp các module này với nhau và kiểm thử trên toàn bộ khối mã lệnh đã tích hợp này.
- ✓ Kiểm thử hệ thống (System test): kiểm thử trên toàn bộ ứng dụng.
- ✓ Kiểm thử chấp nhận (Acceptance Test): khâu này do khách hàng trực tiếp đảm nhận trước khi bàn giao sản phẩm chính thức
- ✓ Kiểm thử hồi quy là hoạt động trợ giúp để đảm bảo rằng các thay đổi không đưa ra những hành vi hoặc những lỗi bổ sung không mong đợi.

#### **2.5. Một số loại hình kiểm thử phổ biến**

Hiện nay, do sự phát triển mạnh mẽ của công nghệ phần mềm nên có một số loại hình kiểm thử tiêu biểu như:

- ✓ Kiểm thử các phần mềm trên Desktop: đây là các ứng dụng được cài đặt trực tiếp trên máy tính cá nhân nhằm thực hiện các chức năng theo yêu cầu người dùng. Đây vẫn đang là những ứng dụng phổ biến nhất.
- ✓ Kiểm thử Web hay kiểm thử trên đám mây: với sự lớn mạnh của Internet thì các ứng dụng web cũng ngày càng phát triển và đang dần thay thế các ứng dụng trên Desktop truyền thống như Google Document, Microsoft web apps,...
- ✓ Kiểm thử trên Mobile: ngày nay xã hội với sự phát triển nhanh chóng, các thiết bị di động (điện thoại thông minh, máy tính bảng,...) có số lượng người dùng cũng đang tăng lên chóng mặt, cùng với đó là số lượng phần mềm phục vụ cho nhu cầu cũng tăng cao vì vậy đây là một lĩnh vực đầy tiềm năng và thách thức trong công nghệ phần mềm.

# cụ kiểm thử phần mềm tự động

## 1.Các kỹ thuật cơ bản của kiểm thử phần mềm

### 1.1.Kiểm thử hộp đen (Black Box Testing - BBT)

#### 1.1.1. Định nghĩa

Định nghĩa: Kiểm thử hộp đen hay còn gọi là kiểm tra chức năng và thử nghiệm hành vi. Xem chương trình như là một “hộp đen”, hoàn toàn không quan tâm về cách cài đặt và cấu trúc bên trong của chương trình. Thay vào đó, Tập trung vào tìm các trường hợp mà chương trình không thực hiện theo các đặc tả của nó.

#### 1.1.2. Các phương pháp kiểm thử hộp đen

- ◆ Phân lớp tương đương – Equivalence partitioning.
- ◆ Phân tích giá trị biên – Boundary value analysis.
- ◆ Kiểm thử mọi cặp – All-pairs testing.
- ◆ Kiểm thử fuzz – Fuzz testing.
- ◆ Kiểm thử dựa trên mô hình – Model-based testing.
- ◆ Ma trận dấu vết – Traceability matrix.
- ◆ Kiểm thử thám dò – Exploratory testing.
- ◆ Kiểm thử dựa trên đặc tả – Specification-base testing

#### 1.1.3. Đặc điểm BBT

Kiểm thử dựa trên đặc tả tập trung vào kiểm tra tính thiết thực của phần mềm theo những yêu cầu thích hợp. Do đó, kiểm thử viên nhập dữ liệu vào, và chỉ thấy dữ liệu ra từ đối tượng kiểm thử. Mức kiểm thử này thường yêu cầu các ca kiểm thử triệt để được cung cấp cho kiểm thử viên mà khi đó có thể xác minh là đối với dữ liệu đầu vào đã cho, giá trị đầu ra (hay cách thức hoạt động) có giống với giá trị mong muốn đã được xác định trong ca kiểm thử đó hay không. Kiểm thử dựa trên đặc tả là cẩn thận, nhưng không đủ để ngăn chặn những rủi ro chắc chắn.

#### 1.1.4. Nguyên lý thực hiện của BBT

Kiểm thử hộp đen không có mối liên quan nào tới mã lệnh, và kiểm thử viên chỉ rất đơn giản cảm nhận là: một mã lệnh phải có lỗi. Sử dụng nguyên tắc “Hãy đòi hỏi và ta sẽ được nhận”, những kiểm thử viên hộp đen tìm ra lỗi mà những lập trình viên đã không tìm ra. Nhưng, mặt khác, người ta cũng nói kiểm thử hộp đen “giống như là đi trong bóng tối mà không có đèn vậy”, bởi vì kiểm thử viên không biết các phần mềm được kiểm tra thực sự được xây dựng như thế nào. Đó là lý do mà có nhiều trường hợp mà một kiểm thử viên hộp đen viết rất nhiều ca kiểm thử để kiểm tra một thứ gì đó mà đáng lẽ có thể chỉ cần kiểm tra bằng 1 ca kiểm thử duy nhất, và/hoặc một số phần của chương trình không được kiểm tra chút nào.

#### 1.1.5. Các trường hợp ứng dụng

##### ◆ Phân lớp tương đương - Equivalence partitioning.

Ý tưởng : phân hoạch miền dữ liệu vào thành các dữ liệu có liên hệ với nhau

Mỗi lớp dùng để kiểm thử 1 chức năng , gọi là lớp tương đương.

Các bước :

- ✓ Đối với dữ liệu đầu vào , xác định các lớp tương đương từ miền dữ liệu.
- ✓ Chọn dữ liệu đại diện cho mỗi lớp tương đương
- ✓ Kết hợp các dữ liệu thử bởi cách để các để tạo ra bộ dữ liệu kiểm thử.

Nguyên tắc phân hoạch các lớp tương đương

- Nếu dữ liệu vào phụ thuộc một khoảng, xây dựng
  - 1 lớp các giá trị lớn hơn
  - 1 lớp các giá trị nhỏ hơn
  - N các giá trị hợp lệ
- Nếu dữ liệu là tập hợp các giá trị, Xây dựng
  - 1 lớp tập rỗng
  - 1 lớp quá nhiều các giá trị
  - N lớp hợp lệ
- Nếu dữ liệu đầu vào là điều kiện ràng buộc, xây dựng
  - 1 lớp các ràng buộc được thỏa mãn.
  - 1 lớp với ràng buộc không được thỏa mãn.

❖ **Phân tích giá trị biên - Boundary value analysis.:**

Cơ sở : lỗi thường xuất hiện gần các giá trị biên của miền dữ liệu.

Tập trung phân tích các giá trị biên của miền dữ liệu để xây dựng dữ liệu kiểm thử.

Nguyên tắc kiểm thử các dữ liệu bao gồm:

- Giá trị nhỏ nhất.
- Giá trị gần kề lớn hơn giá trị nhỏ nhất.
- Giá trị bình thường.
- Giá trị gần kề nhỏ hơn giá trị lớn nhất
- Giá trị lớn nhất

Nguyên tắc chọn dữ liệu thử

- Nếu dữ liệu vào thuộc một khoảng, chọn
  - 2 Giá trị biên.
  - 4 giá trị = Giá trị biên  $\pm$  sai số nhỏ nhất
- Nếu giá trị vào phụ thuộc danh sách các giá trị, chọn phần tử lớn thứ nhất, phần tử thứ hai, phần tử kề cuối, phần tử cuối.
- Nếu dữ liệu đầu vào là điều kiện ràng buộc số giá trị, chọn số giá trị tối thiểu và số giá trị tối đa và một số giá trị không hợp lệ.

❖ **Bảng quyết định- Decision Table Bases testing**

-Làm giảm số lượng test case không cần thiết so với 2 kỹ thuật trên vì nó loại trừ các phép kết hợp không cần thiết giữa các giá trị biến đầu vào.

-Liệt kê nguyên nhân (cause) – kết quả (result) trong một ma trận. Mỗi cột ma trận đại diện cho 1 phép kết hợp giữa các cause trong việc tạo ra 1 result

-Các bước để tạo bảng quyết định

- Liệt kê các nguyên nhân trong bảng quyết định
- Tính tổng số lượng kết hợp giữa các cause
- Diện vào các cột với tất cả các kết hợp có thể có
- Rút bớt số lượng các phép kết hợp dư thừa
- Kiểm tra các phép kết hợp có bao phủ hết mọi trường hợp hay không
- Bổ sung kết quả vào bảng quyết định

❖ **Đồ thị nguyên nhân kết quả.**

-Là kỹ thuật thiết kế test case dựa tên đồ thị

-Tập trung vào việc xác định các mối kết hợp giữa các điều kiện và kết quả mà các mối kết hợp mang lại.

-Các bước xây dựng đồ thị:

- ✓ B1: Phân chia hệ thống thành vùng hoạt động
- ✓ B2: Xác định nguyên nhân kết quả
- ✓ B3: Chuyển nội dung ngữ nghĩa trong đặc tả thành đồ thị liên kết các cause và result
- ✓ B4: Chuyển đổi đồ thị thành bảng quyết định
- ✓ B5: Thiết lập danh sách test case từ bảng quyết định. Mỗi test case tương ứng với 1 cột trong bảng quyết định

❖ *Kiểm thử dựa trên yêu cầu - Specification-based testing.*

Phương pháp kiểm thử dựa vào chức năng - Specification-based testing: Việc kiểm thử được tiến hành dựa vào việc kiểm thử chức năng của phần mềm xem nó có phù hợp với yêu cầu của người dùng hay không. Vì vậy, các tester nhập data vào phần mềm và chỉ cần xem kết quả của phần mềm và các mục tiêu test. Mức test này thường yêu cầu các tester phải viết test case đầy đủ trước khi test, khi test, đơn giản chỉ cần thực hiện theo các bước mô tả trong test case thao tác và nhập data vào, sau đó xem kết quả trả về hoặc hành vi của phần mềm, rồi so sánh với kết quả mong đợi đã được viết trong test case, điều kết quả test vào test case là OK (OK = is – chương trình làm đúng theo mong đợi) hay NG (not good = is not – chương trình không làm đúng theo mong đợi). Specification-based testing là cần thiết, nhưng nó không đủ để bảo đảm chắc chắn các rủi ro xảy ra (nó chỉ là điều kiện cần chứ không phải là điều kiện đủ).

#### 1.1.6. Ưu, nhược điểm của BBT

a) *Ưu điểm:*

- ✓ Tester không cần kiến thức thực hiện, bao gồm cả ngôn ngữ lập trình cụ thể.
- ✓ Tester thực hiện trên quan điểm của người sử dụng.
- ✓ Giúp phát hiện bất kỳ sự mờ hồ hoặc không nhất quán trong các đặc tả chức năng.

b) *Nhược điểm:*

- ✓ Chỉ có một số nhỏ các yếu tố đầu vào có thể thực sự có thể được thử nghiệm, để kiểm tra tất cả các dòng đầu vào có thể sẽ mất gần như mãi mãi.
- ✓ Có thể để lại nhiều phần chương trình chưa được kiểm tra.

### 1.2. Kiểm thử hộp trắng (White Box Testing - WBT)

#### 1.2.1. Định nghĩa

Là phương pháp kiểm nghiệm dựa vào cấu trúc/mã lệnh chương trình. WBT kiểm nghiệm một chương trình (một phần chương trình, hay một hệ thống, một phần của hệ thống) đáp ứng tốt tất cả các giá trị input bao gồm cả các giá trị không đúng hay không theo dự định của chương trình. Chiến lược này xuất phát từ dữ liệu kiểm thử bằng sự kiểm thử tinh logic của chương trình. Kiểm thử viên sẽ truy cập vào cấu trúc dữ liệu và giải thuật bên trong chương trình (và cả mã lệnh thực hiện chúng).

#### 1.2.2. Đặc điểm của WBT

- ❑ Dựa vào thuật giải cụ thể, vào cấu trúc dữ liệu bên trong của đơn vị phần mềm cần kiểm thử để xác định đơn vị phần mềm đó có thực hiện đúng không.
- ❑ Người kiểm thử phải có kỹ năng, kiến thức nhất định để có thể thông hiểu chi tiết về đoạn code cần kiểm thử.
- ❑ Thường tốn rất nhiều thời gian và công sức nếu mức độ kiểm thử được nâng lên ở cấp kiểm thử tích hợp hay kiểm thử hệ thống.
- ❑ Do đó kỹ thuật này chủ yếu được dùng để kiểm thử đơn vị. Trong lập trình hướng đối

tượng, kiểm thử đơn vị là kiểm thử từng tác vụ của 1 class chức năng nào đó.

- Có 2 hoạt động kiểm thử hộp trắng :
  - Kiểm thử luồng điều khiển.
  - Kiểm thử dòng dữ liệu.

Phụ thuộc vào các cài đặt hiện tại của hệ thống và của phần mềm, nếu có sự thay đổi thì các bài test cũng thay đổi theo Được ứng dụng trong các kiểm tra ở cấp độ mô đun, tích hợp và hệ thống của quá trình test phần mềm.

### 1.2.3.Các kĩ thuật kiểm thử của WBT

WBT dựa trên:

- **Các câu lệnh (statement) :**

Thiết kế quá trình kiểm tra sao cho mỗi câu lệnh của chương trình được thực hiện ít nhất một lần. Phương pháp kiểm tra này xuất phát từ ý tưởng: Từ phi một câu lệnh được thực hiện, nếu không ta không thể biết được có lỗi xảy ra trong câu lệnh đó hay không Nhưng việc kiểm tra với một giá trị đầu vào không đảm bảo là sẽ đúng cho mọi trường hợp

- **Dường dẫn (path)**

Là phương pháp kiểm tra bao trùm mọi đường dẫn của chương trình và cần kết hợp với lược đồ tiến trình.

**Tư tưởng:** Viết đủ các ca kiểm thử mà mỗi quyết định có kết luận đúng hay sai ít nhất 1 lần. Nói cách khác, mỗi hướng phân nhánh phải được xem xét kỹ lưỡng ít nhất 1 lần.

**Nhận xét:**

Phương pháp kiểm tra theo đường dẫn phụ thuộc nhiều vào các biểu thức điều kiện. Tuy nhiên, có những trường hợp số lượng đường dẫn quá lớn (trường hợp vòng lặp). Vì vậy thường không phải là lựa chọn thực tế để tiến hành việc kiểm tra tính đúng đắn của chương trình.

- **Các điều kiện (condition)**

Là phương pháp kiểm tra các biểu thức điều kiện trên 2 giá trị true và false.

- Viết đủ các ca kiểm thử để đảm bảo rằng mỗi điều kiện trong một quyết định đảm nhận tất cả các kết quả có thể ít nhất một lần.

**Nhận xét:** Khi kiểm tra bằng phương pháp kiểm tra theo điều kiện cần xem xét kết hợp các điều kiện với nhau.

- **Vòng lặp (loop)**

Là phương pháp tập trung vào tính hợp lệ của các cấu trúc vòng lặp.

- Các bước cần kiểm tra cho vòng lặp đơn:

- + Bỏ qua vòng lặp.
- + Lặp một lần.
- + Lặp hai lần.
- + Lặp m lần ( $m < n$ ).
- + Lặp ( $n-1$ ),  $n$ , ( $n+1$ ) lần.

Trong đó  $n$  là số lần lặp tối đa của vòng lặp.

- Các bước cần kiểm tra cho vòng lặp dạng lồng nhau:

- + Khởi đầu với vòng lặp nằm bên trong nhất. Thiết lập các tham số lặp cho các vòng lặp bên ngoài về giá trị nhỏ nhất.
- + Kiểm tra với tham số min+1, 1 giá trị tiêu biểu, max-1 và max cho vòng lặp bên trong nhất trong khi các tham số lặp của các vòng lặp bên ngoài là nhỏ nhất.
- + Tiếp tục tương tự với các vòng lặp liền ngoài tiếp theo cho đến khi tắt cả vòng

lặp bên ngoài được kiểm tra.

- Các bước cần kiểm tra cho vòng lặp nối tiếp: Nếu các vòng lặp là độc lập với nhau thì kiểm tra như trường hợp các vòng lặp dạng đơn, nếu không thi kiểm tra như trường hợp các vòng lặp lồng nhau.

#### 1.2.4. Ưu, nhược điểm của WBT

##### a) Ưu điểm

- ❖ Kiểm tra được toàn bộ chương trình nguồn
- ❖ Phát hiện lỗi tại chỗ
- ❖ Tự động hóa kiểm thử

##### b) Nhược điểm

- ❖ Yêu cầu người kiểm thử phải am hiểu cấu trúc mã lệnh chương trình. Do đó đòi hỏi tài nguyên nhân lực và máy tính kém.
- ❖ Có khả năng tồn tại các tổ hợp lệnh khác nhau gây lỗi
- ❖ Không kiểm thử hết đường đi với các vòng lặp lớn, phức tạp
- ❖ Khó thực hiện và chi phí thực hiện cao

### 1.3. Kiểm thử hộp xám (Gray Box Testing - GBT)

#### 1.3.1. Định nghĩa

GBT là một phương pháp kiểm thử phần mềm được kết hợp giữa BBT và WBT. Trong BBT, Tester kiểm thử các hạng mục mà không cần biết cấu trúc bên trong của nó, còn trong WBT thì Tester biết được cấu trúc bên trong của chương trình. Trong GBT, cấu trúc bên trong sản phẩm chỉ được biết một phần, Tester có thể truy cập vào cấu trúc dữ liệu bên trong và thuật toán của chương trình với mục đích là để thiết kế test case, nhưng khi test thì test như là người dùng cuối hoặc là ở mức hộp đen. Được gọi là GBT vì trong chương trình phần mềm, mắt của Tester giống như hộp xám/bán trong suốt - nhìn qua hộp này ta chỉ có thể thấy được một phần.

#### 1.3.2. Ứng dụng

Mặc dù phương pháp GBT có thể ứng dụng vào nhiều mức test khác nhau nhưng chủ yếu nó hữu dụng trong mức Integration Testing - kiểm thử tích hợp.

#### 1.3.3. Ưu điểm và Nhược điểm

Ưu điểm và nhược điểm của Kiểm thử Hộp xám được quyết định dựa vào sự kết hợp các ưu điểm của BBT và WBT.

## 2. Các cấp độ hay giai đoạn kiểm thử phần mềm

### 2.1. Kiểm thử đơn vị (Unit Testing, UT)

Trước hết ta sẽ định nghĩa khái niệm đơn vị. Một đơn vị là một thành phần phần mềm nhỏ nhất mà ta có thể kiểm tra được → UT là kiểm tra sự thực thi của các đơn vị chương trình riêng rẽ.

Do các Unit được kiểm tra thường có kích thước nhỏ và chức năng hoạt động trong đối đơn giản nên chúng ta sẽ không gặp mấy khó khăn 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 vị Unit đang kiểm tra. Trong thực tiễn thường thì thời gian chi phí cho UT 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. Do đó nếu trong bước này chúng ta làm cẩn thận thì các bước test sau sẽ ít gặp lỗi hơn nhiều.

UT 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 phát triển phần mềm. Thông thường, UT đòi hỏi

kiểm tra viên có kiến thức về thiết kế và code của chương trình. Mục đích của UT là bảo đảm thông tin được xử lý và xuất ra khỏi Unit là chính xác trong mối tương quan giữa dữ liệu nhập và chức năng của unit.

Cũng như các mức kiểm tra khác, UT 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.

## 2.2. Kiểm thử tích hợp (Integration Test, IT)

IT kết hợp các thành phần của một ứng dụng và kiểm tra như một ứng dụng đã hoàn thành. Trong khi UT kiểm tra các thành phần và Unit riêng lẻ thì IT kết hợp chúng lại với nhau và kiểm tra sự giao tiếp và truyền thông điệp giữa chúng.

Các mục tiêu chính của IT bao gồm:

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

Một chiến lược cẩn quan tâm trong IT là nên tích hợp dần từng Unit. Một Unit tại một thời điểm được tích hợp vào một nhóm các Unit khác mà nhóm các Unit đó đã được tích hợp trước đó và đã hoàn tất các đợt IT 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 và sai sót sẽ giảm đáng kể.

## 2.3. Kiểm thử hồi quy

Kiểm thử hồi quy là hoạt động trợ giúp để đảm bảo rằng các thay đổi không đưa ra những hành vi hoặc những lỗi bổ sung không mong đợi.

Kiểm thử hồi quy có thể được thực hiện thủ công, bằng cách thực hiện lại tập con tất cả các trường hợp kiểm thử hoặc sử dụng các công cụ tự động. Bộ kiểm thử hồi quy gồm ba lớp các trường hợp kiểm thử khác nhau:

- Một mẫu đại diện của các kiểm thử sẽ được thực hiện tất cả các chức năng của phần mềm.
- Các trường hợp kiểm thử bổ sung tập trung vào các chức năng phần mềm có khả năng bị tác động khi có sự thay đổi.
- Các kiểm thử tập trung vào các thành phần phần mềm vừa bị thay đổi.

Kiểm thử hồi quy là một trong những loại kiểm thử tồn tại lâu nhất và công sức nhất. Tuy nhiên, việc bỏ qua kiểm thử hồi quy là điều không thể 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 kiểm thử và sửa chữa rồi.

## 2.4. Kiểm thử hệ thống (System Testing, ST)

ST bao gồm một loạt những kiểm nghiệm nhằm xác minh toàn bộ các thành phần của hệ thống được tích hợp một cách đúng đắn. Mục đích của ST là đảm bảo toàn bộ hệ thống hoạt động như khách hàng mong muốn.

ST bắt đầu khi tất cả các bộ phận của phần mềm 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 yêu cầu phải có môi trường kiểm thử thích hợp. Ví dụ như 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 bổ, hoặc hệ thống nhúng. Ở mức độ của ST thì tester cũng tìm kiếm các lỗi, nhưng trọng tâm của loại kiểm thử này là đánh giá về chức năng, 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 IT và ST là ST chú trọng các hành vi và lỗi trên toàn hệ thống, còn IT chú trọng sự giao tiếp giữa các thực thể hoặc đối tượng khi chúng làm việc cùng nhau. Trong quy trình kiểm thử thì thông thường ta phải thực hiện UT và IT để đảm bảo mọi đơn vị và giao tiếp, tương tác giữa chúng hoạt động chính xác trước khi thực hiện ST.

## 2.5. Kiểm thử chấp nhận (Acceptance Testing, AT)

Sau giai đoạn ST là AT, đây là giai đoạn kiểm tra được khách hàng thực hiện. Mục đích của AT là để chứng minh phần mềm thỏa mãn tất cả yêu cầu của khách hàng và khách hàng chấp nhận sản phẩm và trả tiền thanh toán hợp đồng.

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

Trên thực tế, nếu khách hàng không quan tâm và không tham gia vào quá trình phát triển phần mềm thì thường kết quả AT 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 bộ cục màn hình nghèo nàn, thao tác không tự nhiên, không theo tập quán sử dụng của khách hàng...

## 3. Các công cụ kiểm thử tự động và ứng dụng

### 3.1. Tổng quan về kiểm thử tự động

Ngày nay, tự động hóa được ứng dụng trong rất nhiều lĩnh vực với mục đích là rất đa dạng tùy vào nhu cầu của mỗi lĩnh vực và điểm chung nhất là giảm chi phí, nhân lực, thời gian cũng như sai sót. Tự động hóa trong kiểm thử phần mềm cũng không nằm ngoài mục đích đó. Thực tế đã cho thấy, áp dụng tự động hóa kiểm thử mang lại những hiệu quả, thành công nhất định cho khâu kiểm thử phần mềm. Kiểm thử tự động giúp giảm chi phí kiểm thử bằng cách hỗ trợ quá trình kiểm thử thông qua các công cụ phần mềm.

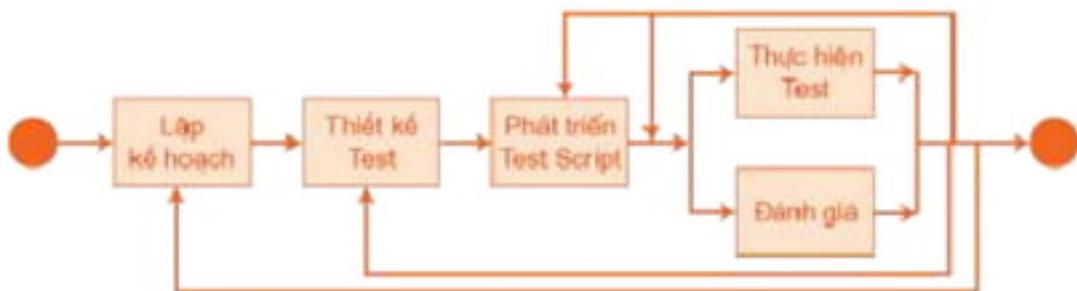
#### 3.1.1. Khái niệm

Kiểm thử tự động là phương pháp sử dụng phần mềm hay các công cụ để xử lý tự động các bước thực hiện test case mà không cần sự can thiệp của con người. Trong kiểm thử tự động, phần mềm phải được biên dịch và chạy thực sự.

Mục tiêu:

- Giảm bớt công sức và thời gian thực hiện quá trình kiểm thử cho cả một kế hoạch kiểm thử.
- Tăng độ tin cậy.
- Rèn luyện kỹ năng lập trình cho tester.
- Giảm chi phí cho tổng quá trình kiểm thử.

#### 3.1.2. Quy trình kiểm thử tự động



Hình 5.1. Quy trình kiểm thử tự động

Việc phát triển kiểm thử tự động cũng tuân theo các bước phát triển phần mềm, ta phải xem việc phát triển kiểm thử phần mềm giống như phát triển một dự án. Giống như phát triển phần mềm, chúng ta thực hiện các bước cơ bản sau:

- *Xây dựng yêu cầu:* thu thập các đặc tả yêu cầu, lựa chọn những phần cần thực hiện kiểm thử tự động, lập kế hoạch kiểm thử.
- *Phân tích thiết kế mô hình kiểm thử tự động:* xây dựng mô hình phát triển kiểm thử tự động, thiết kế và xây dựng các test case để thực thi.
- *Phát triển testscript:*
  - ❖ Tạo testscript: giai đoạn này chúng ta sẽ sử dụng test tool để ghi lại các thao tác lên phần mềm cần kiểm tra và tự động sinh ra test script.
  - ❖ Chỉnh sửa testscript: chỉnh sửa để testscript thực hiện kiểm tra theo đúng yêu cầu đặt ra, cụ thể là làm theo test case cần thực hiện.
- *Chạy testscript:* giám sát các hoạt động kiểm thử phần mềm của test script.
- *Kiểm tra kết quả:* kiểm tra kết quả thông báo ngay sau khi thực hiện kiểm thử tự động.
- *Danh giá kết quả kiểm thử:* thông qua báo cáo kết quả kiểm thử, bổ sung, chỉnh sửa những sai sót.

### 3.1.3. Ưu và nhược điểm của kiểm thử tự động

Ưu điểm	Nhược điểm
<ul style="list-style-type: none"> <li>- Không cần đến sự can thiệp của kiểm thử viên.</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 kiểm thử tự động.</li> <li>- Tốn chi phí cho bảo trì các script.</li> <li>- Đòi hỏi kiểm thử viên phải có kỹ năng tạo các script kiểm thử tự động.</li> <li>- Không áp dụng được trong việc tìm lỗi mới của phần mềm.</li> </ul>

Bảng 5.2. Ưu và nhược điểm của kiểm thử tự động

## 3.2.Các công cụ hỗ trợ kiểm thử tự động

### 3.2.1.Các phần mềm thương mại

Hiện nay có rất nhiều phần mềm của nhiều công ty phần mềm nổi tiếng như Quick Test Professional (QTP) của HP, Rational Rose của IBM,... hỗ trợ kiểm thử phần mềm tự động nhưng chỉ có QTP là phần mềm được ưa chuộng bởi tính dễ sử dụng và kiểm thử với hiệu năng cao

#### a) Giới thiệu QTP

QTP là phần mềm kiểm soát việc test tự động những chức năng của một sản phẩm phần mềm khác, là một bộ phận (module) của hệ thống Mercury Quality Center bao gồm nhiều module phần mềm

phối hợp với nhau để quản lý toàn bộ quy trình đảm bảo chất lượng sản phẩm phần mềm.

Nếu ta có một sản phẩm phần mềm Quản lý Nhân sự. Ví dụ, khi mở phần mềm Quản lý Nhân sự lên, thì người dùng sẽ gặp form Đăng nhập (login) để nhập vào "Tên tài khoản" và "Mật khẩu", rồi nhấn nút "OK" hoặc "Cancel" để vào. Ta lập trình ra lệnh cho QTP tự động điền thông tin vào 2 ô "Tên tài khoản" và "Mật khẩu", và rồi cũng tự động "nhấn" nút "OK" hoặc "Cancel" dùm ta luôn. Công việc này gọi là viết script cho QuickTest Pro. Viết script để thực hiện nhiều trường hợp nhập dữ liệu khác nhau, nhiều thao tác khác nhau, để thử xem chức năng của form "Đăng nhập" có hoạt động đúng hay không. QTP sau khi chạy script xong, sẽ thực hiện ghi nhận kết quả việc test tự động, và có thể xuất report. Nếu có đủ một hệ thống Mercury Quality Center thì ít ra phải có thêm phần mềm Mercury Test Director đóng vai trò là phần mềm chủ (serving software) đảm nhận việc tổng hợp các kết quả test, các báo cáo, các phát sinh... của QTP, từ đó phục vụ cho công việc quản trị chất lượng sản phẩm phần mềm của ta (Software Quality Assurance). Đây là chương trình 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 hiện đại, cho phép kĩ thuật viên 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 chuyên giao công việc mà người mới tiếp nhận chưa có thời gian hoặc không hiểu script vẫn có thể thực hiện kiểm tra phần mềm theo đúng yêu cầu

b) *Loại phần mềm hỗ trợ*

- ✓ QTP giúp chúng ta kiểm tra phần mềm 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: .NET Framework 1.0, 1.1, 2.0, các đối tượng chuẩn của .NET và các đối tượng khác thừa kế từ các đối tượng chuẩn, Java Sun JDK 1.1 – 1.6, IBM JDK 1.2 – 1.4,...

c) *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 kiểm tra tự động trên nhiều trình duyệt cùng lúc tốt hơn những công cụ kiểm tra 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).

- ✓ Quản trị Object Repository
- ✓ Phối hợp giữa các KTV qua việc đồng bộ hóa dữ liệu, khả năng trộn, nhập/xuất ra file XML
- ✓ Kiểm tra tài nguyên: Kiểm tra tài nguyên cần thiết trước khi thực thi lệnh kiểm tra tự động.
- ✓ Hỗ trợ XML cho báo cáo: Lưu trữ kết quả kiểm tra dưới dạng XML, HTML, tự động cho phép tùy biến báo cáo.
- ✓ Quản trị từ khóa trong quá trình sử dụng
- ✓ Hỗ trợ đa giao tiếp: Cho phép người dùng mở và soạn thảo đồng thời nhiều hàm thư viện và Object Repository.
- ✓ Giao diện sử dụng đẹp, dễ bắt nhập
- ✓ Hỗ trợ quản lý script: Debug toolbar, hỗ trợ kiểm tra lỗi trong test script (debug)
- ✓ Testing: Hỗ trợ quá trình tạo test script hoặc thực hiện kiểm tra tự động

d) Các thành phần quan trọng trong QTP

- Action

Giống như thủ tục hay hàm trong các ngôn ngữ lập trình khác, Action ghi lại các bước thực hiện kiểm tra tự động và nó có thể được sử dụng lại nhiều lần. Trong một test script có thể có nhiều Action.

- DataTable

Nơi lưu trữ dữ liệu phục vụ cho kiểm tra tự động. 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.

- Object Repository (OR)

- Cấu trúc theo dạng cây, mô tả các đối tượng trong phần mềm được kiểm tra. Đây được xem là cấu nối để test script tương tác với phần mềm được kiểm tra.
- Khi ra lệnh cho QTP ghi lại thao tác người dùng lên phần mềm thì trong OR sẽ tự động phát sinh thành phần đại diện cho những đối tượng trên PHẦN MỀM 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.

- Checkpoint:

Là nơi kiểm tra trong test script, khi chạy nó sẽ thực hiện so sánh kết quả thực tế khi kiểm tra phần mềm 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).

e) Ngôn ngữ 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 phần mềm 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

### 3.2.2. Các công cụ mã nguồn mở

a) Junit

- Giới thiệu: JUnit là một framework đơn giản dùng cho việc tạo các unit testing tự động, và chạy các test có thể lặp đi lặp lại. Nó chỉ là một phần của họ kiến trúc xUnit cho việc tạo các unit testing. JUnit là một chuẩn trên thực tế cho unit testing trong Java. JUnit về gốc được viết

bởi 2 tác giả Erich Gamma và Kent Beck.

JUnit có những đặc điểm đáng lưu tâm như sau:

- Xác nhận (assert) việc kiểm tra kết quả được mong đợi
- Các Test Suite cho phép chúng ta dễ dàng tổ chức và chạy các test
- Hỗ trợ giao diện đồ họa và giao diện dòng lệnh
- Các test case của JUnit là các lớp của Java, các lớp này bao gồm một hay nhiều các phương thức unit testing, và những test này lại được nhóm thành các Test Suite.
- Mỗi phương thức test trong JUnit phải được thực thi nhanh chóng. Tốc độ là điều tối quan trọng vì càng nhiều test được viết và tích hợp vào bên trong quá trình xây dựng phần mềm, cần phải tốn nhiều thời gian hơn cho việc chạy toàn bộ Test Suite. Các lập trình viên không muốn bị ngắt quãng trong một khoảng thời gian dài trong khi các test chạy, vì thế các test mà chạy càng lâu thì sẽ có nhiều khả năng là các lập trình viên sẽ bỏ qua bước cũng không kém phần quan trọng này.
- Các test trong JUnit có thể là các test được chấp nhận hay thất bại, các test này được thiết kế để khi chạy mà không cần có sự can thiệp của con người. Từ những thiết kế như thế, ta có thể thêm các bộ test vào quá trình tích hợp và xây dựng phần mềm một cách liên tục và để cho các test chạy một cách tự động

- Các thành phần quan trọng của JUnit

+ *Phương thức assertXXX()*

- i. assertEquals(): So sánh 2 giá trị để kiểm tra bằng nhau. Test sẽ được chấp nhận nếu các giá trị bằng nhau
- ii. assertFalse(): Đánh giá biểu thức luận lý. Test sẽ được chấp nhận nếu biểu thức sai
- iii. assertNotNull(): So sánh tham chiếu của một đối tượng với null. Test sẽ được chấp nhận nếu tham chiếu đối tượng khác null
- iv. assertNotSame(): So sánh địa chỉ vùng nhớ của 2 tham chiếu đối tượng bằng cách sử dụng toán tử “==”. Test sẽ được chấp nhận nếu cả 2 đều tham chiếu đến các đối tượng khác nhau
- v. assertNull(): So sánh tham chiếu của một đối tượng với giá trị null. Test sẽ được chấp nhận nếu tham chiếu là null
- vi. assertSame(): So sánh địa chỉ vùng nhớ của 2 tham chiếu đối tượng bằng cách sử dụng toán tử “==”. Test sẽ được chấp nhận nếu cả 2 đều tham chiếu đến cùng một đối tượng
- vii. assertTrue(): Đánh giá một biểu thức luận lý. Test sẽ được chấp nhận nếu biểu thức đúng
- viii. fail(): Phương thức này làm cho test hiện hành thất bại, phương thức này thường được sử dụng khi xử lý các ngoại lệ

Tất cả các phương thức của bảng trên đều nhận vào một String không bắt buộc làm tham số đầu tiên. Khi được xác định, tham số này cung cấp một thông điệp mô tả test thất bại.

+ *Phương thức set up và tear down*

- i. Hai phương thức setUp() và tearDown() là một phần của lớp junit.framework.TestCase. Bằng cách sử dụng các phương thức setUp và tearDown. Khi sử dụng 2 phương thức setUp() và tearDown() sẽ giúp chúng ta tránh được việc trùng lặp mã khi nhiều test cùng chia sẻ nhau ở phần khởi tạo và dọn dẹp các biến.
- ii. JUnit tuân thủ theo một dây có thứ tự các sự kiện khi chạy các test. Đầu tiên, nó tạo ra một thể hiện mới của test case ứng với mỗi phương thức test. Từ đó, nếu ta có 5 phương thức test thì JUnit sẽ tạo ra 5 thể hiện của test case. Vì lý do đó, các biến thể hiện không

- thể được sử dụng để chia sẻ trạng thái giữa các phương thức test. Sau khi tạo xong tất cả các đối tượng test case, JUnit tuân theo các bước sau cho mỗi phương thức test:
- iii. Gọi phương thức `setUp()` của test case / Gọi phương thức test / Gọi phương thức `tearDown()` của test case
  - iv. Quá trình này được lặp lại đối với mỗi phương thức test trong test case.
  - v. Thông thường ta có thể bỏ qua phương thức `tearDown()` vì mỗi unit test riêng không phải là những tiến trình chạy tồn tại nhiều thời gian, và các đối tượng được thu dọn khi JVM thoát. `tearDown()` có thể được sử dụng khi test của ta thực hiện những thao tác như mở kết nối đến cơ sở dữ liệu hay sử dụng các loại tài nguyên khác của hệ thống và ta cần phải dọn dẹp ngay lập tức. Nếu ta chạy một bộ bao gồm một số lượng lớn các unit test, thì khi ta trả tham chiếu của các đối tượng đến null bên trong thân phương thức `tearDown()` sẽ giúp cho bộ dọn rác lấy lại bộ nhớ khi các test khác chạy
  - vi. Đôi khi ta muốn chạy vài đoạn mã khởi tạo chỉ một lần, sau đó chạy các phương thức test, và ta chỉ muốn chạy các đoạn mã dọn dẹp chỉ sau khi tắt cả test kết thúc. Ở phần trên, JUnit gọi phương thức `setUp()` trước mỗi test và gọi `tearDown()` sau khi mỗi test kết thúc, vì thế để làm được điều như trên, chúng ta sẽ sử dụng lớp `junit.extension.TestSetup` để đạt được yêu cầu trên.

+ *Chạy các test lặp đi lặp lại*

Trong một vài trường hợp, chúng ta muốn chạy một test nào đó lặp đi lặp lại nhiều lần để đo hiệu suất hay phân tích các vấn đề trực tiếp. JUnit cung cấp cho chúng ta lớp `junit.extension.RepeatedTest` để làm được điều này. Vì `TestSuite` cài đặt interface `Test` nên chúng ta có thể lặp lại toàn bộ test như trên.

- *Cách tổ chức chương trình chạy với JUnit*

+ *Tổ chức các test vào các test suite*

- ❖ Thông thường JUnit tự động tạo ra các Test Suite ứng với mỗi Test Case. Tuy nhiên ta muốn tự tạo các Test Suite của riêng mình bằng cách tổ chức các Test vào Test Suite. JUnit cung cấp lớp `junit.framework.TestSuite` hỗ trợ việc tạo các Test Suite. Khi sử dụng giao diện text hay graphic, JUnit sẽ tìm phương thức sau trong test case của ta: `public static Test suite() { }`
- ❖ Nếu không thấy phương thức trên, JUnit sẽ sử dụng kỹ thuật reflection để tự động xác định tất cả các phương thức `testXXX()` trong test case của ta, rồi thêm chúng vào một test suite. Sau đó nó sẽ chạy tất cả các test trong suite này.

+ *Test các exception*

Chúng ta sử dụng cặp từ khóa `try/catch` để bắt các exception như mong đợi, chúng ta sẽ gọi phương thức `fail()` khi exception chúng ta mong đợi không xảy ra. Nói chung ta chỉ nên sử dụng kỹ thuật này khi ta mong đợi một exception xảy ra. Đối với các điều kiện lỗi khác ta nên để exception chuyển sang cho JUnit. Khi đó JUnit sẽ bắt lấy và tường trình 1 lỗi test.

b) *Nunit:*

-Giới thiệu: N-unit là một trong số nhiều công cụ kiểm thử tự động, với nhiều version khác nhau. N-unit có hai cách khác nhau để chạy chương trình kiểm nghiệm:

+N-unit-console.exe: là khởi chạy nhanh nhất nhưng không phải là tương tác, là một văn bản dựa trên runner và có thể được sử dụng khi ta muốn chạy tất cả các bài test của ta và không cần phải có màu đỏ/màu vàng/ màu xanh chỉ ra thành công hay thất bại. Nó rất hữu ích cho tự động hóa của bài thi và tích hợp vào các hệ thống khác. Nó tự động lưu kết quả của nó trong định dạng

\*.xml, cho phép ta để sản xuất các báo cáo hay xử lý các kết quả. Dưới đây là một ảnh chụp màn hình của chương trình.

+N-unit-gui.exe: là một hình thức cho phép ta lựa chọn làm việc với các bài test của ta và cung cấp các thông tin phản hồi đồ họa. Đặc biệt hơn, nó sẽ tự động reload lại khi có sự chỉnh sửa và build lại mã nguồn.

-Cách thức hoạt động

Vì Nunit cũng là một phần trong họ các công cụ kiểm thử xUnit nên về cơ bản cách thức hoạt động cũng giống Junit nhưng Nunit thay vì tích hợp vào các IDE của Java như Eclipse hay Netbean thì Nunit lại được tích hợp vào bộ công cụ lập trình của Microsoft Visual Studio (các phiên bản 2003, 2005, 2008, 2010 và 2012).

## Chương III Một số loại hình kiểm thử hiện nay

### 1.Tổng quan về kiểm thử Website

#### 1.1.Giới thiệu

Đã từ lâu việc kiểm thử website đặt ra nhiều thách thức với các nhà phát triển web. Ngày nay hầu hết các ứng dụng đều phải dựa trên nền tảng, dựa trên sức mạnh ngày càng mạnh mẽ của Internet, việc một trang web sau khi được xây dựng thành công xong và đưa lên Server thì dù tại bất kì đâu trên thế giới đều có thể truy cập vào được. Một điểm mạnh nữa của các ứng dụng web đó là nó có thể được truy cập từ bất kì thiết bị đầu cuối nào mà có cài đặt một trình duyệt web và hoạt động trên đa nền tảng, từ các hệ điều hành phổ biến chạy trên PC như Microsoft Windows, Mac OSX, Linux cho đến các hệ điều hành nhúng đơn giản nhỏ gọn chạy trên các thiết bị di động như Google Android, Apple Iphone, RIM Blackberry,...Tính năng cơ động này càng trở nên phổ biến khi các thiết bị Smart Phone có số lượng người dùng tăng lên đột biến trong các năm trở lại đây. Sự phát triển của xã hội cộng với trào lưu dùng mạng xã hội để chia sẻ và giao lưu giữa con người như Facebook, Twitter,... làm cho công nghệ web ngày càng phát triển mạnh, cùng với đó là một nền tảng Điện toán đám mây (cloud computing) với các ứng dụng cần thiết đều có sẵn trên máy chủ web Server, các máy tính Client chỉ đơn giản là có một kết nối tới Internet là có thể sử dụng các dịch vụ có sẵn của Server mà không hề cần cài đặt, điều này giảm đáng kể chi phí mua bán quyền phần mềm và nhân công để quản trị từng máy tính đơn lẻ vì tất cả các công việc này đều được xử lý tập trung tại Server.

Tóm lại, với sự phát triển như vũ bão của Internet trong hơn chục năm trở lại đây đã kéo theo hàng loạt các công nghệ và dịch phát triển điện hình có thể kể đến các nền tảng mới như xuất hiện càng nhiều các thiết bị di động chạy hệ điều hành nhúng (Smart phone, Iphone) mà trước đây có rất ít, các mạng xã hội cũng thu hút số lượng người dùng tăng lên đột biến (như Facebook được giới thiệu sau chỉ vài năm mà số lượng người dùng đã tăng lên hơn một tỷ người, cùng với đó cũng có rất nhiều mạng xã hội được mở ra như Google Plus,...trong Việt Nam cũng có khá nhiều mạng xã hội mới có thể kể tới như Zing Me, Yume,...) và cuối cùng một nền tảng nữa cũng không thể không nhắc tới và hứa hẹn sẽ phát triển mạnh mẽ vào tương lai là công nghệ điện toán đám mây, có thể kể đến các ứng dụng nổi bật như Google Document, Microsoft Sky live Office là các ứng dụng văn phòng trực tuyến khá phổ biến cho phép ta tạo và lưu trữ ngay trên Server mà không cần

cài đặt trên Client các phần mềm văn phòng này, hay như các dịch vụ lưu trữ trực tuyến cho phép người dùng lưu trữ, chia sẻ, đồng bộ dữ liệu giữa các thiết bị di động hay máy tính cá nhân giúp ta không bao giờ bị mất dữ liệu nếu chẳng may máy tính có bị hỏng thì toàn bộ dữ liệu của ta vẫn còn an toàn trên Server mà không hề bị mất mát, các dịch vụ nổi tiếng hiện nay như Google Drive, Microsoft Sky drive, Mediafire, Box,... Sự phát triển mạnh mẽ này đặt ra hàng loạt các vấn đề này sinh bên cạnh sự tiện dụng của nó như vấn đề dữ liệu riêng tư cá nhân hay các tệp liệu quan trọng mang tính an ninh quốc gia có thể bị một người không có thẩm quyền xem trộm hay đánh cắp dữ liệu phục vụ cho mục đích cá nhân hay một âm mưu nào đó. Từ đó đặt ra vấn đề cấp thiết phải có một nghiên cứu kĩ càng công việc kiểm thử các ứng dụng web này trước khi đưa đến tay cho người dùng.

### 1.2.Khái niệm

Kiểm thử website là một thành phần trong kiểm thử phần mềm tập trung vào các ứng dụng web, là một trong những thành phần đang phát triển nhanh nhất của kiểm thử phần mềm

Hoàn tất kiểm tra trang web của một hệ thống trước khi đi vào hoạt động là bước đầu để có được sự yên tâm về khả năng toàn bộ ứng dụng trên trang web hoạt động đúng. Nó có thể giúp giải quyết các vấn đề chẳng hạn như sự sẵn sàng của máy chủ web của ta cho con đường mà ta đang mong chờ và cho số lượng ngày càng cao người sử dụng, khả năng sống sót trong lưu lượng truy cập của người dùng. Việc bỏ qua các vấn đề kiểm thử website trước khi đi vào hoạt động có thể dẫn đến số người truy cập website thấp và sự tồn tại của website khó được đảm bảo. Sau khi thực hiện kiểm thử web, ta sẽ có thể tìm thấy tắc nghẽn trong hệ thống của ta trước khi chúng xảy ra trong môi trường sản xuất.

### 1.3.Mục đích của kiểm thử Website

Kiểm thử web nhằm trả lời cho câu hỏi :

- 1.Các thiết bị phần cứng và phần mềm ảnh hưởng như thế nào tới việc kiểm thử ?
- 2.Các thành phần của ứng dụng web có ảnh hưởng gì tới chiến lược kiểm thử
- 3.Vai trò của một CSDL phụ trợ là gì và làm thế nào để kiểm tra cho một database ko gấp lỗi.
- 4.Làm thế nào để kiểm thử một phần mềm trên server ? Thế nào là sự hiệu suất , sự cảng thẳng , và thử tải, và làm thế nào để lên kế hoạch thực thi chúng?
- 5.Cần biết được cái gì về kiểm tra bảo mật và trách nhiệm của việc kiểm tra chúng?

→ Kiểm thử website nhằm đảm bảo rằng khả năng toàn bộ ứng dụng trên trang web hoạt động đúng đắn và hiệu quả, đáp ứng nhu cầu khách hàng.

### 1.4.So sánh kiểm thử web với kiểm thử phần mềm truyền thống (trên Desktop)

a.Sự khác nhau giữa phần cứng và phần mềm

- Một hệ thống máy tính để bàn duy nhất bao gồm hỗn hợp phần cứng và phần mềm - nhiều thành phần phần cứng được xây dựng và hỗ trợ bởi các nhà sản xuất khác nhau, nhiều hệ điều hành, và kết hợp gần như vô hạn của phần mềm ứng dụng. Cấu hình và các vấn đề tương thích trở nên khó khăn hoặc gần như không thể quản lý trong môi trường này.

- Một hệ thống web bao gồm rất nhiều client cũng như server. Phía server của hệ thống web có thể cũng hỗ trợ hỗn hợp của phần mềm và phần cứng và, do đó, có nhiều phức tạp hơn hệ thống máy tính lớn (mainframe), từ quan điểm cấu hình và khả năng tương thích.

b. Sự khác nhau giữa mô hình web và hệ thống client-server truyền thông

- Hầu hết các hệ thống client-server là hệ thống truy xuất dữ liệu bị động (data-access-driven).

Một client tiêu biểu cho phép người dùng thông qua giao diện người dùng, gửi các dữ liệu đầu vào, nhận các dữ liệu xuất ra. Client trong hệ thống client-server truyền thông là một nền tảng cụ thể,

nó hỗ trợ bởi hệ điều hành, một ứng dụng client được phát triển và kiểm tra trên nền tảng hệ điều hành đó.

- Hầu hết các hệ thống web cũng là hệ thống truy xuất dữ liệu bị động (data-access-driven). Các trình duyệt được thiết kế để xử lý các hoạt động tương tự như client truyền thống. Điểm khác nhau chính đó là web-based client hoạt động trên nền trình duyệt web. Trình duyệt web là một phần mềm chạy trên máy client, được thiết kế riêng biệt cho từng hệ điều hành. Nó hiển thị HTML, cũng như các nội dung khác như Javascript, Vbscript, các điều khiển ActiveX, XML, CSS, DHTML, các tính năng bảo mật ...

- Việc phát triển một ứng dụng web không cần quan tâm đến sự tương thích hệ điều hành vì trình duyệt web đã làm điều đó cho chúng ta rồi. Trên lý thuyết, nếu nội dung HTML được thiết kế theo chuẩn HTML 4, thì ứng dụng web có thể chạy trên bất cứ trình duyệt nào hỗ trợ chuẩn HTML 4.

- Việc tạo ra một website không chỉ kết thúc bằng việc đưa các thành phần như âm thanh, hình ảnh và mã code của website cùng với nhau. Thực chất, sự hoạt động của website không bao giờ kết thúc. Khi đã hoàn thành toàn bộ việc thiết kế website, chúng ta phải kiểm tra nó trước khi đưa lên World Wide Web để mọi người có thể truy cập. Có những phần mềm quản lý website (site management software) có thể giúp chúng ta làm việc này. Những phần mềm này có thể giúp kết nối lại những hình ảnh bị tinh cờ di chuyển, thay đổi tên file và re-link chúng lại, và rất nhiều việc khác nữa.

- Ngoài ra, từ những phần mềm này, chúng ta cũng kiểm tra được chất lượng website của mình. Website phải được kiểm tra, sửa lỗi, test lại và lên danh mục đầy đủ. Nếu bắt ki phần mềm nào được sử dụng trong website, nó cũng phải được kiểm thử. Những điều cần phải được kiểm tra nhằm đảm bảo chất lượng website là tính tương thích giữa các trình duyệt, thời gian tải về các thành phần của trang web (như hình ảnh, âm thanh, các file flash), yêu cầu phản ứng, dung lượng bộ nhớ cần thiết, tốc độ kết nối của người sử dụng và khả năng chịu tải (số lượng người dùng đồng thời mà website có thể đáp ứng). Rất nhiều công ty hiện nay đã phát triển những phần mềm riêng biệt cho việc đảm bảo chất lượng. Nhưng những phần mềm này thường rất đắt.

- Một vài kiểu kiểm tra khác đó là : kiểm tra chức năng - functional test (đảm bảo các chức năng hoạt động đúng), stress test (site được kiểm tra trên những máy tính với cấu hình khác nhau), regression test (định nghĩa cách thức mà site sẽ được kiểm tra trong pha tiếp theo), boundary analysis (kiểm tra sự giới hạn của site như thông tin đầu vào trong các form). Đôi khi cách thức tốt nhất để kiểm thử website là bằng một người duyệt từ đầu đến cuối site của ta và đề anh ta nói cho ta những vấn đề anh ta gặp phải.

Việc kiểm tra website quan trọng như thế nào trước khi cho nó hoạt động ? Kiểm thử bản chất là đảm bảo cho mọi bộ phận của website như các chức năng, đặc biệt là các phần mềm hoạt động tốt. Kiểm thử website đảm bảo rằng không có link lỗi (broken link), không có từ sai chính tả, không có lỗi trong các phần mềm được sử dụng, và thời gian tải theo đúng lý thuyết.

## 1.5. Các phương pháp testing

### 1.5.1. Kiểm tra các chức năng, luồng nghiệp vụ (Functional Test)

Mục đích của việc test chức năng (Functional Test) là đảm bảo các yêu cầu của khách hàng. Tester sẽ ưu tiên kiểm tra các dữ liệu hợp lệ (Valid data), luồng nghiệp vụ (Flow) trước rồi sau đó chuyển lỗi cho phía phát triển. Sau khi kiểm tra phần dữ liệu hợp lệ, luồng nghiệp vụ ổn định thì tester mới tiến hành kiểm tra các trường hợp dữ liệu không hợp lệ (Invalid). Test chức năng đảm bảo các yêu cầu sau:

- ✓ Nhập dữ liệu hợp lệ thì chương trình phải cho nhập

- ✓ Luồng nghiệp vụ phải đúng
- ✓ Quá trình xử lý dữ liệu và kết quả đầu ra phải đúng.
- ✓ Phục hồi được dữ liệu.
- ✓ Kiểm tra với các điều kiện gây lỗi, nhập sai dữ liệu, các giá trị ngoại phạm vi hệ thống phải đưa ra thông báo lỗi.

#### **1.5.2. Kiểm tra giao diện người dùng (User Interface Test)**

Test giao diện người dùng trên các chức năng phải đảm bảo:

- Giao diện người dùng phải phản ánh được đầy đủ các chức năng mà người dùng yêu cầu bao gồm: các cửa sổ lệnh, các trường dữ liệu và phương thức tiếp cận các chức năng đó ( phím tab, dùng chuột và các phím tắt).
- Các đối tượng cửa sổ và các đặc tính như menu, kích thước, vị trí và focus theo yêu cầu và theo tiêu chuẩn.
- Các tiêu đề, nội dung, các Control trên form phải đúng chính tả.
- Hài hòa bố cục màu sắc và quan bố vị trí các control sao cho hợp lý.

#### **1.5.3. Kiểm tra hiệu năng (Performance Test)**

Mục đích của việc test hiệu năng là việc kiểm tra xem thời gian khi thực hiện một thao tác nào đó của phần mềm có đáp ứng được yêu cầu của khách hàng hay không?

Test thông qua các tiêu chuẩn đã được thông qua về hiệu năng ví dụ 3-5 giây / 1 trang và sử dụng các tooltest.

#### **1.5.4. Kiểm tra tính bảo mật và điều khiển truy cập (Security and access control testing)**

Mục đích của việc kiểm tra này là kiểm tra việc tiếp cận của một người dùng sau khi được phân quyền thì chỉ được thao tác với chức năng được phép.

Kỹ thuật kiểm tra:

- o Liệt kê danh sách các nhóm người dùng và các chức năng mà họ có quyền truy cập:
- o Kiểm tra việc tiếp cận của nhóm người dùng được thao tác trên chức năng đã được gán
- o Kiểm tra việc tiếp cận của nhóm người dùng không được thao tác trên một số chức năng.

#### **1.5.5. Kiểm thử tính dùng được**

Là quá trình qua đó các đặc trưng tương tác người-máy của hệ thống được đo đạc. Ta có thể bắt đầu bằng kiểm thử tính dễ dùng, mọi người thấy học dùng ứng dụng web dễ hay khó thế nào, cách họ di từ trang này sang trang khác, thông tin trên website được mô tả tốt thế nào, website trông như thế nào?

#### **1.6. Test Driven Development với Asp.net MVC**

Như đã trình bày trên phần 1.3.3, quy trình phát triển phần mềm dựa trên kiểm thử đang ngày càng được sử dụng rộng rãi bởi tính ưu việt của nó so với mô hình phát triển phần mềm truyền thống, giúp đẩy nhanh quá trình triển khai phần mềm và sửa các lỗi kịp thời trước khi các lỗi này lan rộng ra các thành phần khác trong hệ thống phần mềm. Theo như Microsoft giới thiệu, MVC framework được thiết kế để cho phép kiểm thử mà không cần triển khai trên một Web Server (IIS), trên một cơ sở dữ liệu hay trên các class mở rộng khác (điều này hoàn toàn trái ngược với mô hình Web form truyền thống, luôn luôn yêu cầu cần có một Web server). Với sự hỗ trợ của công cụ mã nguồn mở là Nunit và Unit Test tích hợp sẵn trong Visual Studio, việc kiểm thử trên các ứng dụng Asp.net MVC đã trở nên đơn giản hơn và thuận tiện cho các nhà phát triển phần mềm. Sau đây em xin trình bày một demo nhỏ về kiểm thử trên Asp.net MVC.

Đầu tiên ta sẽ tạo mới một Project với tên là MyTestMVC trên Visual Studio. Sau khi nhấn OK ta

sẽ chọn tiếp vào mục Create a unit test project với tùy chọn Test framework là Visual Studio Unit Test. Nhấn OK, Visual Studio sẽ tạo ra một Project mới để ta bắt đầu tiến hành xây dựng ứng dụng. Ta có thể nhìn tổng quan cấu trúc của một ứng dụng Asp.net MVC:

- ❖ Thư mục Content chứa các tài nguyên tĩnh của chương trình như các file ảnh, CSS,...
- ❖ Thư mục chứa các class controller để điều khiển, định tuyến cho toàn bộ chương trình khi có yêu cầu từ client gửi tới.
- ❖ Thư mục Model chứa các class thao tác với cơ sở dữ liệu nếu có, và tại đây ta cũng sẽ tiến hành xử lý nghiệp vụ (Business logic) cho toàn bộ ứng dụng.
- ❖ Thư mục View chứa các file dạng \*.cshtml, một dạng mã kết hợp của C# và HTML để hiển thị nội dung trang Web cho người dùng, dạng view này được gọi là Razor View engine (sẽ được trình bày thêm ở phần sau).
- ❖ Thư mục Script chứa toàn bộ các file JavaScript, Jquery, Ajax phục vụ trong ứng dụng như xác thực dữ liệu nhập, thêm các hiệu ứng phụ,...

Khi chạy chương trình ta sẽ thấy kết quả:



Và cả project test ta đã tạo ra ban đầu cùng với ứng dụng cũng đã được biên dịch và chạy thành công. Tiếp theo ta sẽ thêm một class Controller là MapsController để hiển thị các bản đồ các thành phố có trong một danh sách cho trước nào đó: Trong cửa sổ Solution Explorer, nhấn phải chuột vào thư mục Controller và chọn Add new Controller, nhập vào tên controller là MapsController. Để áp dụng TDD vào Project này ta cần viết Unit test cho một Action Method trước khi ta cài đặt Action Method đó. Tuy nhiên nếu ta muốn Unit test được biên dịch thì ta cần thực thi các method này trước. Tiếp theo ta sẽ tạo một Action Method với tên là ViewMaps trong MapsController ta vừa tạo

```
public class MapsController : Controller
{
    //
    // GET: /Maps/

    public ActionResult ViewMaps()
    {
        // Add action logic here
        throw new NotImplementedException();
    }
}
```

Sau đó ta sẽ tạo một Unit test cho action method ViewMaps, khi chạy unit test này thì kết quả là fail vì ViewMaps chưa được cài đặt

```
79 // whether you are testing a page, web service, or a WCF service.
80 [TestMethod()]
81 [Url("http://localhost:2040")]
82 [Request("GET", "http://localhost:2040/")]
83 [Response("HTTP/1.1 200 OK")]
84 public void ViewPageTest()
85 {
86     // Arrange
87     MyController target = new MyController(); // This will initialize to an appropriate value
88     ActionResult expected = null; // This will initialize to an appropriate value
89     ActionResult actual;
90 
91     // Act
92     actual = target.ViewPage();
93     // Assert
94     Assert.AreEqual(expected, actual);
95     Assert.Inconclusive("verify the correctness of this test method.");
96 }
```

Để tiếp tục sẽ chính sửa lại phương thức ViewMaps() như sau:

```
public ActionResult ViewMaps()
{
    // Add action logic here
    ViewBag.Title = "Welcome to My Maps!";
    return this.View("ViewMaps");
}
```

Như ta thấy phương thức trên chỉ đơn giản là trả về một View để hiển thị cho người dùng nội dung kèm theo một thông báo do ta tùy chọn.

Tiếp theo ta sẽ cần tạo ra một View (một trang dạng cshtml) để hiển thị danh sách các bản đồ các thành phố trên thế giới. Click phải chuột vào vị trí của phímme thức ViewMaps và chọn Add View.

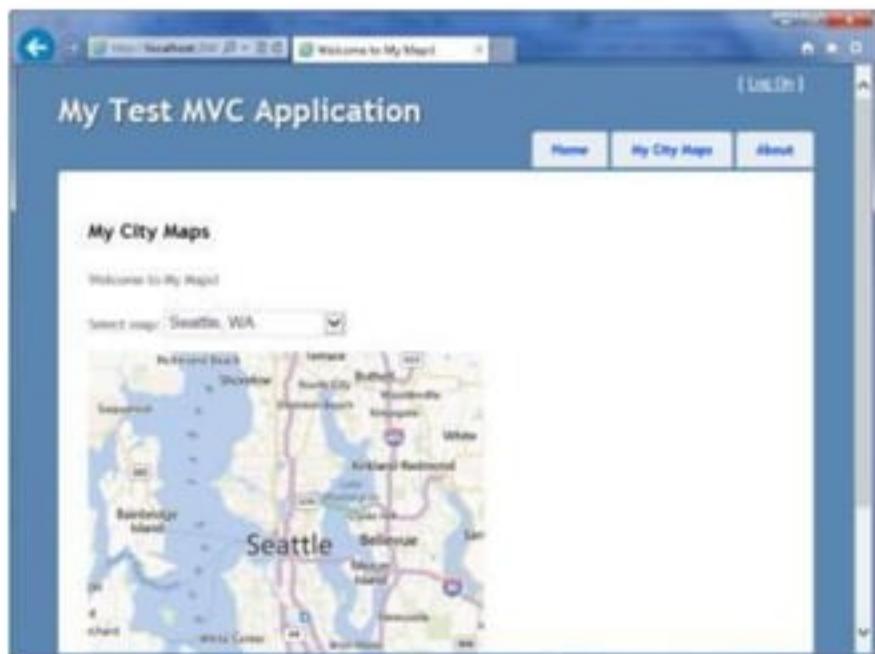
Sau đó thêm vào trang view maps.cshtml đoạn mã sau để hiển thị bản đồ các thành phố

www.citypages.com

```
<p>ViewMap.Title</p>
Select map:
<select onclick="GetMap(value);">
    <option value="Seattle">Seattle, USA</option>
    <option value="LasVegas">Las Vegas, NV</option>
    <option value="SaltLake">Salt Lake City, UT</option>
    <option value="Dallas">Dallas, TX</option>
    <option value="Chicago">Chicago, IL</option>
    <option value="NewYork">New York, NY</option>
    <option value="Rio">Rio de Janeiro, Brasile</option>
    <option value="Paris">Paris, France</option>
    <option value="Rome">Rome, Italy</option>
    <option value="Keta">Keta, Ghana</option>
    <option value="Beijing">Beijing, China</option>
    <option value="Sydney">Sydney, Australia</option>
</select>
<br />
<br />
<div id="earthmap" style="position: relative; width: 400px; height: 400px;">
</div>
<script charset="UTF-8" type="text/javascript"
    src="http://dev.virtualearth.net/mapcontrol/mapcontrol_enhanced.js?ct=1">
</script>
<script type="text/javascript">
    var map = null;
    var mapID = '';
    function GetMap(mapID) {
        switch (mapID) {
            case 'Seattle':
                map = new VtMap('earthmap');
                map.LoadMap(new VtLocation(47.6, -122.33), 10, 'i', true);
                break;
        }
    }
</script>
```

Đoạn mã trên đã tham chiếu đến một thư viện JavaScript Online hỗ trợ bởi BingMap để hiển thị bản đồ đồng thời ta cũng xuất ra thông báo mà ta đã truyền từ controller sang.

Kết quả khi chạy chương trình



Sau khi đã chạy thành công ứng dụng ta sẽ tiến hành kiểm thử phương thức ViewMaps này. Chính sửa lại phương thức ViewMapsTest như sau

```
[TestMethod()]
public void ViewMapsTest()
{
    MapsController controller = new MapsController();

    // Act
    ViewResult result = controller.ViewMaps() as ViewResult;

    // Assert
    Assert.IsNotNull(result);
}
```

Sau đó tiến hành run test và kết quả thu được là

Test Results			
	Run	Debug	Stop
<input checked="" type="checkbox"/> <a href="#">Test run completed</a>	Results: 1/1 passed; Item(s) checked: 0		
<hr/>			
Result	Test Name	Project	Error Message
<input checked="" type="checkbox"/> <span style="color: green;">Passed</span>	ViewMapsTest	MyTestMVC.Tests	

Tiếp tục ta sẽ thực hiện test một phương thức nữa và chạy toàn bộ test trong ứng dụng

```

75
76 [TestMethod]
77 public void ViewMapsTest()
78 {
79     MapsController controller = new MapsController();
80
81     // Act
82     ViewResult result = controller.ViewMaps() as ViewResult;
83
84     // Assert
85     Assert.IsNotNull(result);
86 }
87
88 [TestMethod]
89 public void ViewMapsTestTitle()
90 {
91     MapsController controller = new MapsController();
92
93     // Act
94     ViewResult result = controller.ViewMaps() as ViewResult;
95
96     string title = (String)result.ViewBag.Title;
97
98     Assert.AreEqual("Welcome to My Maps!", title);
99 }

```

**Test Results**

Result	Test Name	Project	Error Message
Passed	ViewMapsTestTitle	MyTestMVC.Tests	
Passed	ViewMapsTest	MyTestMVC.Tests	

## 2. Khái quát về kiểm thử trên SmartPhone

### 2.1. Giới thiệu

Như em đã trình bày ở trên, với sự phát triển nhanh chóng của Internet cộng với trào lưu mạng xã hội bùng nổ điện thoại thông minh đang ngày càng được sử dụng nhiều nhằm đáp ứng nhu cầu giải trí đa dạng của người dùng. Từ một chiếc điện thoại thông thường chỉ được cài đặt sẵn vài ba ứng dụng của nhà sản xuất thì nay với các thiết bị chạy các hệ điều hành nhúng (Android, iOS,...) ta có thể dễ dàng đáp ứng được các nhu cầu của người dùng bằng cách cài thêm các phần mềm bên thứ ba mà không gây ra trở ngại nào. Từ đây lại đặt ra một vấn đề hiển nhiên là kiểm thử các phần mềm chạy trên di động này để xem chúng có đáp ứng được các yêu cầu đề ra ban đầu hay không trước khi phân phát sản phẩm tới tay người tiêu dùng. Thực hiện đúng các điều kiện cơ bản sau sẽ làm giảm khả năng kiểm thử phần mềm trên mobile bị sai. Ví dụ chọn đúng thiết bị cần kiểm thử, vì mỗi thiết bị đều có những tính năng đặc thù riêng ví dụ như iOS thì chỉ có năm mẫu có màn hình kích thước khác nhau là iPad (9.7 inches), iPad Mini (7.9 inches), iPhone 4S (3.5 inches) và iPhone 5 (4.0 inches) trong khi Android thì có tới hơn 10 nhà sản xuất phần cứng với hàng trăm mẫu màn hình kích thước khác nhau. Năm bắt được các kiến thức cơ bản của môi trường lập trình SDK để từ đó ta có thể tạo được các Emulator phù hợp để kiểm thử.

### 2.2. Các yếu tố ảnh hưởng đến hoạt động của phần mềm trên SmartPhone

-Tuổi thọ của Pin: Bình thường một chiếc điện thoại có thời lượng Pin đủ dùng trong nhiều ngày nhưng với những chiếc smartphone do sử dụng rất nhiều dịch vụ giải trí như kết nối mạng, nghe nhạc, xem phim,...nên thời lượng pin bị rút ngắn đi rất nhiều nên cần phải nạp điện thường xuyên hơn. Và trong số các lý do làm thất thoát điện năng thì ứng dụng của ta cũng chịu một phần trách nhiệm như thường xuyên kết nối tới Server hay chạy quá chậm chạp, chiếm giữ tài nguyên quá lâu. Và đó là lý do mà người dùng sẽ gỡ bỏ nó đi.

-Kết nối mạng: các ứng dụng luôn luôn tiêu thụ tài nguyên khi chúng kết nối mạng. Bản chất của di động là vị trí luôn luôn thay đổi vì vậy người dùng có thể tắt kết nối mạng của thiết bị đi trong một vài giờ. Vì vậy ta phải thiết một ứng dụng có khả năng hoạt động ngay cả khi không có mạng (offline) chẳng hạn như gửi email hay viết tin nhắn và ngay khi mạng được kết nối tới thì ứng dụng có khả năng gửi hàng loạt email và tin nhắn mà người dùng đã soạn thảo trước đó một cách tự động.

-Sự khác nhau giữa các thiết bị và phần mềm cài trên từng thiết bị này, bao gồm kích thước màn hình, chipset, bộ nhớ,...Lí tưởng nhất nếu phần mềm có thể hoạt động trên mọi thiết bị phần cứng và nền tảng khác nhau.

-Giới hạn về tài nguyên: hầu hết các thiết bị di động đều có tài nguyên hạn chế như tốc độ xử lý của CPU, không gian lưu trữ,...vì vậy vẫn đề tiết kiệm tài nguyên hệ thống của các ứng dụng cũng rất cần được xem trọng.

### 2.3. Lựa chọn thiết bị SmartPhone để test

Vì các đội kiểm thử không ai có đầy đủ mọi mẫu điện thoại cần thiết nên trên mỗi nền tảng ta có thể chọn ra các thiết bị di động tiêu biểu để tiến hành kiểm thử như

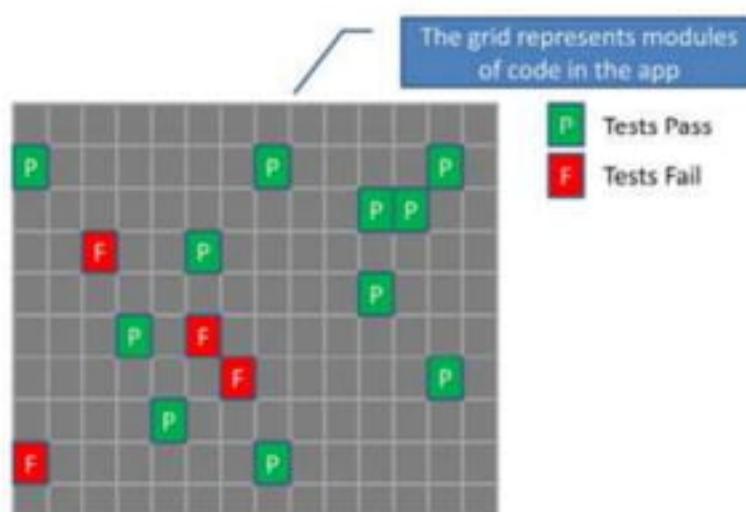
-Các thiết bị phổ biến bao gồm iPhone 4S của Apple, Nokia N73.

-Với Android thi có thể lựa chọn Samsung Galaxy Nexus chạy android 4.0

-với các thiết bị ít phổ biến hơn ta có thể chọn Sony Ericsson LiveView.

### 2.4. Kiểm thử tự động

a) Unit test: được khuyến nghị nên sử dụng cho các đơn vị mã nhỏ. Một đơn vị mã có thể có một vài phương thức riêng lẻ hay phương thức quan hệ trong một file chương trình. Unit test chỉ test một phần nhỏ của chương trình để xem chúng hoạt động có đúng không



Hình 4.3. Kiểm thử tích hợp

Với các ứng dụng mobile, một vài nền tảng chính có và mở rộng các unit testing framework được tạo và chạy như một phần của quy trình phát triển phần mềm.

b) Kiểm thử tích hợp: khi các Unit test đã được kiểm thử thành công thi tích hợp test giúp ta kiểm thử được cả ứng dụng khi kết hợp các module vào với nhau. Như đã đề cập ở bên trên thi kiểm thử đơn vị đã đủ linh hoạt để thay thế các loại hình kiểm thử khác, bao gồm cả kiểm thử tích hợp. Bởi vì kiểm thử tích hợp sẽ cần nhiều mã hơn nên sẽ mất nhiều thời gian, nhất là đối với các thiết bị di động tài nguyên luôn hạn hẹp. Để khắc phục ta có thể chạy các công việc tồn thời gian một cách bắt đồng bộ và chỉ thực hiện kiểm thử tích hợp khi kiểm thử đơn vị đã thành công.

c) Kiểm thử Activity: Activity là khái niệm đóng nhất và cũng là thành phần quan trọng nhất trong ứng dụng Android. Một Activity là một thành phần rời rạc và liên kết chia sẻ dữ liệu với các thành phần khác trong Android thông qua giao diện form và các luồng chạy ngầm. Android SDK cũng bao gồm các framework cho phép ta kiểm thử tự động các Activity.

d) Kiểm thử hệ thống, ứng dụng

Kiểm thử hệ thống là kiểm thử toàn bộ ứng dụng. Một vài nền tảng platform có thể bao gồm cả việc test cả chương trình nhỏ chạy ngầm bên dưới. Có một số phần mềm test tự động mà có thể sinh ra các tác tử (Agent) chạy ngầm trên mobile để tạo ra các test script kiểm thử một cách tự động. Các tác tử này có một vài dạng như chạy trên thiết bị và cho phép chúng tương tác với ứng dụng hay chạy trên các ứng dụng riêng lẻ.

e) Kiểm thử thông qua giao diện chỉ được hỗ trợ trong các SDK của android và iOS. Kiểm thử giao diện được cung cấp sẵn trên iOS và Monkeyrunner là công cụ kiểm thử thông qua giao diện mới nhất trên Android.

### 3.Kiểm thử trên Android

#### 3.1.Giới thiệu android

Android là một hệ điều hành mã nguồn mở chạy trên mọi thiết bị có cấu hình phần cứng phù hợp. Được phát triển bởi một công ty có tên là Android Inc, sau đó năm 2005 công ty này đã được Google mua lại và sau đó dự án về Android tiếp tục được Google phát triển. Năm 2008, Google chính thức công bố chiếc điện thoại đầu tiên là G1 chạy trên nền tảng Android và bộ công cụ Android SDK phiên bản 1.0 cho các nhà phát triển. Từ đó đến nay Android đã không ngừng phát triển và hiện là hệ điều hành được sử dụng nhiều nhất trên rất nhiều thiết bị, đặc biệt là SmartPhone, Tablet. Theo thống kê không đầy đủ, mỗi ngày có hàng triệu thiết bị chạy Android mới được kích hoạt và số lượng thiết bị chạy Android đã lên tới hơn 1 tỷ. Và Google cũng đã cung cấp bộ công cụ phát triển SDK mới với API level 17 (Android 4.2.2). Tuy nhiên do đây là một hệ điều hành mã nguồn mở, Google mặc dù quản lý nhưng một khi mã nguồn đưa tới các nhà sản xuất phần cứng như Samsung, Sony, LG, HTC,... thì họ có toàn quyền chỉnh sửa mã nguồn cho phù hợp với phần cứng do họ sản xuất. Và cũng chính từ đây này sinh ra nhiều vấn đề về an ninh và bảo mật cho thiết bị như các tội phạm có thể dễ dàng cài vào Android các phần mềm độc hại, theo dõi đánh cắp thông tin người dùng. Vì vậy vẫn đề đặt ra là cần phải kiểm thử kĩ các ứng dụng trước khi chúng được cài đặt vào thiết bị. Để kiểm thử được tốt các ứng dụng trên các thiết bị chạy android ta sẽ cần phải tìm hiểu kiến trúc cũng như các thành phần trong ứng dụng android

##### 3.1.1. Kiến trúc Android

Nhìn một cách tổng thể thi hệ điều hành Android được chia thành 4 phần chính như hình sau

-Tầng Linux Kernel: đây có thể nói là nhân của Android, hệ điều hành này ban đầu được xây dựng dựa trên kernel của phiên bản Linux OS 2.6.2. Về sau khi Android được phát triển lên các phiên bản cao hơn (Android 4.0 Ice Cream Sandwich) thì Google cũng đã không còn dùng Linux 2.6 nữa mà đã nâng lên sử dụng Linux kernel 3.0.4. Tầng này chủ yếu là chứa các drive điều khiển phần cứng của thiết bị như màn hình (display), bàn phím (keypad), wifi, audio, bluetooth, quản lý năng lượng (Pin nếu sử dụng),...

-Ngay bên trên tầng Kernel là tầng Libraries chứa các thư viện đồ họa graphics, database, bảo mật phục vụ cho nhiều mục đích khác nhau,...tầng này chủ yếu viết bằng C/C++ những ngôn ngữ mạnh có khả năng can thiệp sâu vào hệ thống để giao tiếp trực tiếp với phần cứng thông qua sự hỗ trợ của tầng Linux Kernel nằm ngay bên dưới

- Bên cạnh tầng Libraries là Android Runtime, chứa các thư viện lõi của Android API, hỗ trợ cho các nhà phát triển phần mềm trên các thiết bị Android và một phần không thể thiếu được đó là máy ảo Dalvik Virtual Machine (DVM), do Google tạo ra dành riêng cho android OS, máy ảo này có cơ chế hoạt động tương tự như máy ảo Java Virtual Machine chạy trên các ứng dụng Desktop nhưng đã được tinh gọn lại cho phù hợp với các thiết bị phần cứng khác nhau. Mọi ứng dụng chạy trên android đều phải hoạt động thông qua DVM.

- Trên hai tầng này các Application Framework cung cấp trực tiếp các API, framework cho phép ta sử dụng ngay mà không cần phải chuyển đổi nhiều như ContentProvider, NotificationManager, Activity Manager, SharePreference framework,...

- Cuối cùng là tầng Application, tầng này hiển thị giao diện của Android OS cùng các ứng dụng cho người dùng cuối. Cũng như bao hệ điều hành chạy trên các thiết bị di động như iOS, Windows Phone thì người dùng cuối chỉ quan tâm tới tầng này. Vì vậy mặc dù tầng này không hỗ trợ nhiều cho lập trình viên như các tầng bên dưới nhưng lại đóng vai trò to lớn trong việc thu hút người dùng sử dụng android OS. Chính vì lẽ đó mà Google song song với việc phát triển các API mới nhằm hỗ trợ lập trình viên và tối ưu, tăng tốc cho android thì giao diện hệ điều hành này cũng liên tục thay đổi từ phiên bản đầu tiên là android 1.0 đến nay đã là android 4.2.2. Giao diện đã được tinh gọn lại và thân thiện với người sử dụng hơn.

### 3.1.2. Các thành phần chính trong một ứng dụng Android

Các ứng dụng trong android sau khi biên dịch sẽ được đóng gói thành một file dạng duy nhất là file \*.apk, từ đây ta có thể dễ dàng cài đặt vào bất cứ thiết bị nào chạy android và tương thích với ứng dụng. Ở trên ta đã hiểu được cơ bản cấu trúc của android OS như thế nào, nhưng đây là xét xong toàn bộ cả hệ điều hành. Đến đây ta sẽ đi sâu vào tìm hiểu các thành phần chính trong một ứng dụng Android:

- Activity: có thể nói đây là thành phần hầu như không thể thiếu trong các ứng dụng Android, nhiệm vụ của nó là hiển thị giao diện dạng xml cho người dùng và trực tiếp nhận các thao tác, sự kiện do người dùng gửi tới thiết bị (chạm, kéo, nhấn giữ,...), chỉ trừ những ứng dụng chạy ẩn bên dưới thì mới không cần, thường thì những ứng dụng ẩn này sẽ cung cấp đầu vào cho một ứng dụng khác như các service hệ thống thông báo khi Pin sắp hết, khi có tin nhắn hay cuộc gọi tới,...

- Service: đây là một thành phần chạy ngầm trong ứng dụng android, nó không cần giao diện để hiển thị, nhiệm vụ của nó là thực hiện các tác vụ mà người dùng không quan tâm tới giao diện lắm như download, Play music hay các thao tác trên database,...

- ContentProvider: đây là thành phần có tác dụng chia sẻ giữ liệu dùng chung cho các ứng dụng khác nhau. Thông thường mỗi ứng dụng sẽ chỉ truy cập được vào dữ liệu của ứng dụng đó thôi nhưng đôi khi ta cần truy cập vào database của một ứng dụng khác để đỡ mất công tạo lại dữ liệu và đảm bảo tính đồng bộ dữ liệu trong toàn bộ thiết bị. Do đó cách tốt nhất là android cho phép truy cập vào dữ liệu của một ứng dụng mà ta muốn và ContentProvider sẽ giúp ta làm điều này mà không chút khó khăn nào. Ví dụ như khi ta muốn nhắn tin thì phải truy cập vào cơ sở dữ liệu của danh bạ để lấy thông tin về một người nào đó,...

- BroadcastReceiver: đây có thể ví như người đưa thư trong ứng dụng android, nhiệm vụ của nó là gửi các thông điệp hay kết quả xử lý tới các thành phần yêu cầu để tiếp tục xử lý tiếp, tuy nhiên chúng ta có thể dùng Intent để thay thế nhưng trong một số trường hợp như đối với các service hệ thống thì cách tốt nhất là dùng BroadcastReceiver.

### 3.1.3. Vòng đời ứng dụng Android

Vòng đời mỗi ứng dụng android gắn liền với các trạng thái của thành phần Activity của nó vì từ

khi một Activity được khởi tạo sẽ chính thức khởi tạo các tài nguyên của ứng dụng. Các giai đoạn chính trong vòng đời này là khi một ứng dụng bắt đầu được khởi tạo (onCreate, onStart, onResume), hoạt động (running), bị dừng (onPause, onStop) và nếu không được gọi lại thì sẽ bị hủy (onDestroy) và giải phóng hoàn toàn bộ nhớ (shutdown):

- Khởi tạo: khai báo các biến và gán các giá trị tương ứng cần thiết cho ứng dụng hoạt động.
- Hoạt động: ứng dụng sẽ chiếm giữ màn hình và cho phép người dùng tương tác với nó.
- Bị dừng: khi có một ứng dụng khác đến chiếm lấy màn hình hiển thị (ví dụ đang lướt web thì có một cuộc gọi đến thì ứng dụng duyệt web sẽ tạm thời bị dừng).
- Bị hủy và shutdown: nếu bộ nhớ của thiết bị hết thì các ứng dụng đang bị dừng sẽ bị giải phóng bộ nhớ và các tài nguyên khác, lúc này muốn gọi lại ứng dụng thì phải gọi lại từ khởi tạo.

### 3.2. Android Testing Framework

Nền tảng Android cung cấp một framework rất tiện dụng mở rộng từ JUnit framework chuẩn với nhiều tính năng phù hợp với các chiến lược kiểm thử. Những tính năng này bao gồm:

- Bổ sung các class Android mở rộng từ JUnit framework cho phép truy cập vào các đối tượng hệ thống trong Android.
- Instrumentation framework cho phép kiểm soát và kiểm tra ứng dụng.
- Các đối tượng giả lập (Mock) được sử dụng phổ biến trong hệ thống Android.
- Các công cụ cho phép thực hiện các test riêng lẻ hay chạy cả một dãy các lệnh test mà có thể không cần đến Instrumentation framework.
- Hỗ trợ quản lý test và các project test trong ADT plugin của Eclipse và cả chế độ dòng lệnh command line của hệ điều hành.

#### 3.2.1. Instrumentation framework (viết tắt là IF)

- Instrumentation framework là một phần cơ bản của testing framework. IF điều khiển ứng dụng kiểm thử và cho phép gắn các đối tượng thay thế giả lập (Mock object) vào ứng dụng để chạy. Ví dụ ta có thể tạo một đối tượng giả lập Context trước khi ứng dụng bắt đầu và cho phép ứng dụng sử dụng nó. Tất cả mọi tương tác giữa ứng dụng và môi trường xung quanh có thể sử dụng phương pháp tiếp cận này. Ta cũng có thể tách riêng ứng dụng của chúng ta trong một môi trường giới hạn để lấy về kết quả, hoặc các dữ liệu được lưu trữ và không thay đổi như ContentProvider, cơ sở dữ liệu hay thậm chí là hệ thống file.

Một project Android thường có một project Test tương ứng với tên kết thúc bằng Test. Bên trong một Test project file AndroidManifest.xml khai báo thẻ cho biết IF là <instrumentation></instrumentation>. Ví dụ: giả sử project Android có file manifest có dạng sau:

```

2  <?xml version="1.0" encoding="utf-8"?>
3  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
4      package="and.example.addresscontacts"
5      android:versionCode="1"
6      android:versionName="1.0" >
7
8      <uses-sdk
9          android:minSdkVersion="10"
10         android:targetSdkVersion="16" />
11
12     <application
13         android:allowBackup="true"
14         android:icon="@drawable/contact"
15         android:label="@string/app_name"
16         android:theme="@android:style/Theme.Black.NoTitleBar" >
17         <activity
18             android:name=".MainActivity"
19             android:label="@string/app_name" >
20             <intent-filter>
21                 <action android:name="android.intent.action.MAIN" />
22
23                 <category android:name="android.intent.category.LAUNCHER" />
24             </intent-filter>
25         </activity>
26         <provider
27             android:name="and.example.data.MyProvider"
28             android:authorities="and.example.provider" >
29         </provider>
30
31         <service android:name="and.example.service.MyService" >
32     </service>
33
34     </application>
35
36 </manifest>

```

Thì trong test project file này sẽ có dạng tương ứng là

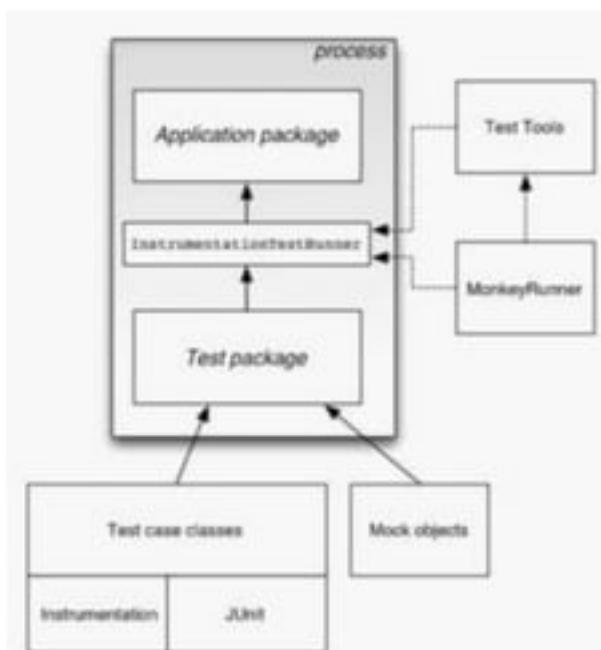
```

2  <?xml version="1.0" encoding="utf-8"?>
3  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
4      package="and.example.addresscontacts.test"
5      android:versionCode="1"
6      android:versionName="1.0" >
7
8      <uses-sdk android:minSdkVersion="14" />
9
10     <instrumentation
11         android:name="android.test.InstrumentationTestRunner"
12         android:targetPackage="and.example.addresscontacts" />
13
14     <application
15         android:icon="@drawable/ic_launcher"
16         android:label="@string/app_name" >
17         <uses-library android:name="android.test.runner" />
18     </application>
19
20 </manifest>
21

```

Nhìn ví dụ này ta có thể thấy ngay rằng trong thẻ khai báo IF là <instrumentation> với thuộc tính name là trình chạy kiểm thử test runner, class mặc định của Android testing API (*android.test.runner*), ta cũng có thể tùy biến class này bằng cách kế thừa từ class InstrumentationTestRunner, thuộc tính targetPackage chỉ ra package của ứng dụng mà ta muốn kiểm thử (trong ví dụ là AddressContacts).

### 3.2.2. Kiến trúc Testing framework



Trong kiến trúc này InstrumentationTestRunner làm nhiệm vụ trung gian trong việc chạy các testcase. Các thành phần chính trong kiến trúc này:

- Application package: chứa toàn bộ ứng dụng để test
- Test projects: chứa các mã nguồn, file manifest và những file khác dùng để kiểm thử ứng dụng. Ta có thể sử dụng Eclipse để tạo ra test project này.
- Testing API
  - + Junit: ta có thể sử dụng các class Junit testcase để kiểm thử đơn vị trên các class.
  - + Instrumentation: Android Instrumentation là một tập các phương thức điều khiển trong hệ thống Andoird. Các điều khiển này độc lập với vòng đời của ứng dụng và chúng cũng kiểm soát cách Android tái ứng dụng để chạy. Thông thường Android framework không cung cấp cách để gọi trực tiếp các hàm callback trong vòng đời của một ứng dụng như onCreate(), onResume(),... nhưng với instrumentation ta có thể gọi các hàm này thông qua các phương thức như getActivity(), activity.finish(),...
- + Test case classes: Android cung cấp một vài class kế thừa từ class TestCase và Assert của Junit framework như ApplicationTestCase, InstrumentationTestCase,...
- Mock Object: để chống sự phụ thuộc (dependency injection) trong kiểm thử, Adroid cung cấp các class để tạo các đối tượng hệ thống giả lập như MockContext, MockContentProvider,...
- MonkeyRunner: một API để thực thi trong môi trường với ngôn ngữ là Python.
- Monkey: một công cụ dòng lệnh để kiểm thử khả năng chịu tải của ứng dụng thông qua công cụ adb của Android.

### 3.2.3.Các mục tiêu kiểm thử

Trong suốt quá trình phát triển phần mềm, các testcase sẽ hướng đến các thiết bị khác nhau. Từ đơn giản, phức tạp và tốc độ kiểm thử trên máy ảo đến trên một thiết bị thật cụ thể nào đó. Ngoài ra có một vài trường hợp trung gian như chạy các test trên một máy ảo cục bộ JVM hay DVM, phụ thuộc vào từng trường hợp. Mỗi trường hợp đều có ưu và nhược điểm riêng. Emulator có lẽ là thiết bị phù hợp nhất mà ta có thể thay đổi gần như tất cả các tham số cấu hình để mô phỏng các điều kiện khác nhau cho testcase.

Thiết bị thật dùng để test hiệu năng vì trên emulator sẽ không thể tính ra được các thông số trên thiết bị thật sê như thế nào. Rendering, filling, và các trường hợp khác nên được kiểm tra trước khi

ứng dụng được chuyển giao cho người dùng cuối.

### 3.2.4. Một vài dạng kiểm thử trên Android

Vì các Android testing API đều là sự mở rộng trên Junit framework nên testing trên Android cũng có một số tính năng tương tự như trên Junit nhưng có một số phần mở rộng để phù hợp với đặc thù của nền tảng Android như sau:

- a) Unit test: Junit framework là một nền tảng cơ bản cho kiểm thử đơn vị trên Android, đơn giản Junit là một framework mã nguồn mở được viết bởi Erich Gamma và Kent Beck. Android API 17 (android 4.2) hiện vẫn chưa hỗ trợ cho Junit 4, vì vậy để kiểm thử đơn vị ta vẫn sử dụng Junit 3.0.
- b) Kiểm thử giao diện người dùng (UI Test): như ta đã biết, chỉ có các luồng chính mới được phép thay đổi giao diện người dùng. Để có thể kiểm thử trên UI, ta phải sử dụng annotation `@UiThreadTest` hay nếu chỉ chạy một phần test trên UI, thì sử dụng lệnh `Activity.runOnUiThread(Runnable r)` với `r` là thread chứa các lệnh kiểm thử. Class TouchUtils cung cấp các sự kiện cho phép ta thực thi các tương tác với UI như: click, drag, long click, scroll, tap, touch.
- c) Kiểm thử tích hợp: được dùng để kiểm thử các thành phần riêng lẻ khi kết hợp với nhau. Các modules khi được kiểm thử đơn vị độc lập sẽ được tích hợp lại trong kiểm thử tích hợp. Thông thường Android Activities cần tích hợp với hệ thống để thực thi được. Các Activities cần ActivityManager cung cấp vòng đời và truy cập vào các tài nguyên, hệ thống file và cơ sở dữ liệu. Tương tự với Service và ContentProvider. Tất cả các thành phần này đều được Android testing framework hỗ trợ cho việc kiểm thử dễ dàng.

### d) Kiểm thử chức năng hay kiểm thử chấp nhận

Trong phát triển phần mềm, kiểm thử chấp nhận thường do những người quản lý chất lượng thực hiện trên một ngôn ngữ đặc thù nào đó. Tuy nhiên khách hàng thường là những người chính thức thực hiện các kiểm thử này. Có một vài công cụ và framework hỗ trợ việc này như FitNesse.

Gần đây có một khuynh hướng phát triển phần mềm mới là Behavior Driven Development (BDD) đang trở nên phổ biến và được biết như là sự tiến hóa của TDD

### e) Kiểm thử hệ thống bao gồm

- Kiểm thử hiệu năng: do đặc tính hiệu năng của các thành phần lặp lại nhiều lần để có thể tối ưu hóa phần mềm.

- Kiểm thử giao diện: như đã trình bày trên

- Kiểm thử Smokes: Trong lĩnh vực sản xuất phần mềm, smoke testing thường được dùng cho lần tích hợp các modules, thành phần hoặc sau khi phần mềm được sửa chữa, bảo trì nhằm mục đích cung cấp cho các bên liên quan những báo đảm phần mềm không có những lỗi nghiêm trọng. Nó chứng minh được phần mềm không bị thất bại ngay lần đầu tiên để chuẩn bị cho bước test tiếp theo là stress test.

- Kiểm thử cài đặt: sau khi đóng gói phần mềm cần kiểm thử việc cài đặt có thành công không trước khi chuyển giao cho khách hàng.

Trong phần trình bày tiếp theo, em xin trình bày chi tiết các class mà Android testing framework hỗ trợ trong việc kiểm thử phần mềm và các bước tiến hành để xây dựng phần mềm theo quy trình phát triển TDD

### 3.2.5. Các bước tiến hành:

- B1: tạo một project Android chính: Từ Menu → File → New Project → Android Application Project, đặt tên cho project là MyFirstProject

- B2: tạo một Project Test: Làm tương tự như bước 1, thay tên Project bằng tên MyFirstProjectTest .

Trong phần chọn Select Test Target, chọn ứng dụng mà vừa tạo xong ở trên

- B3: tạo Test case: Từ TemperatureConverterTest project chọn File → New → JUnit Test case

Sau khi thực hiện xong các bước trên ta sẽ thấy màn hình code như sau

```
1 import junit.framework.TestCase;
2
3 /**
4  * @author quav96
5  *
6  */
7 public class MyFirstTest extends TestCase {
8
9     /**
10      * @param name
11      */
12     public MyFirstTest(String name) {
13         super(name);
14     }
15
16     /* (non-Javadoc)
17      * @see junit.framework.TestCase#setUp()
18      */
19     protected void setUp() throws Exception {
20         super.setUp();
21     }
22
23     /* (non-Javadoc)
24      * @see junit.framework.TestCase#tearDown()
25      */
26     protected void tearDown() throws Exception {
27         super.tearDown();
28     }
29
30 }
31 }
```

- Một vài phương thức đặc biệt

Method	Description
setUp	Thiết lập tài nguyên cho testcase như khởi tạo kết nối database, thiết lập biến toàn cục,... Phương thức này được gọi trước khi test thực thi
tearDown	Hủy bỏ, giải phóng các tài nguyên đã cấp phát. Ví dụ như đóng kết nối tới database, hủy các đối tượng... Phương thức này được gọi sau khi Test đã được thực thi xong.
testSomething	Một test đơn giản sử dụng reflection để test

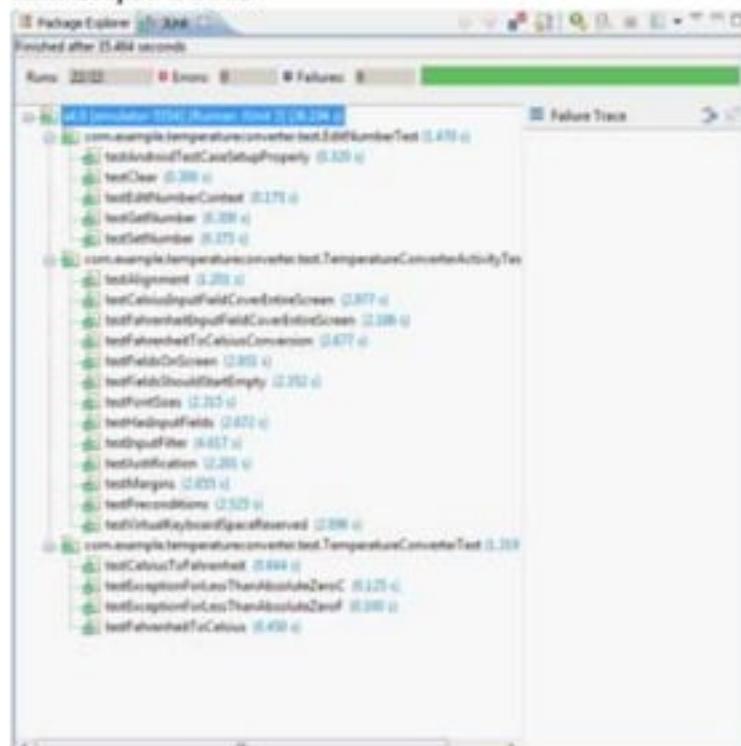
- Test Annotation

Các Annotation thường dùng trong test là

Annotation	Mô tả
@SmallTest	Tạo một test sẽ chạy như một phần của test nhỏ (small test)
@MediumTest	Tạo một test sẽ chạy như một phần của test trung bình (medium test)
@LargeTest	Tạo một test sẽ chạy như một phần của test lớn (large test)
@Smoke	Dành dấu một test sẽ chạy như là một phần của test smoke. Class android.test.suitebuilder.SmokeTestSuiteBuilder sẽ chạy tất cả các phương thức mà có annotation này
@FlakyTest	Sử dụng kí hiệu này cho các phwpwng thức test của class InstrumentationTestCase. Khi test là đang thực hiện, phương thức test này sẽ được thực hiện trước. Điều này là cần thiết nếu các test đều thất bại vì một điều kiện bên ngoài nào đó mà thay đổi theo thời gian.
@UiThreadTest	Sử dụng kí hiệu này trong phương thức test của class

t	InstrumentationTestCase. Khi thực hiện test, phương thức này sẽ được thực thi trên luồng chính của ứng dụng
@Suppress	Sử dụng kí hiệu này trên các class hay method mà không bao gồm một bộ test. Kí hiệu này cũng có thể được sử dụng ở cấp class nếu class đó không bao gồm một method nào cả hay ở một method mà không bao gồm một câu lệnh nào cả hay gọi tới một method khác

- Chạy các test: từ menu chọn Project → Run as Android Unit Testing , kết quả sẽ hiện ra ở màn hình Eclipse DDMS



Để xem chi tiết hơn nữa về các testcase ta có thể xem tại màn hình Logcat trong Eclipse

### 3.2.6. Gõ lỗi cho test

Các test cũng có thể chứa lỗi vì vậy trong trường hợp đó ta sẽ phải áp dụng kĩ thuật debug, ví dụ như gửi các thông báo qua LogCat. Nếu mà việc debug phức tạp hơn cần thiết ta sẽ cần phải định kèm một chương trình gõ lỗi để kiểm thử chính chương trình thực thi. Để làm điều này thì có hai lựa chọn chính sau đây

- Cách thứ 1: rất đơn giản, sử dụng chính công cụ Eclipse với plug in là Android JUnit Test, vì vậy ta có thể đặt breakpoint vào test và thực thi chúng. Để thay đổi breakpoint ta có thể đặt vào một dòng mã nào đó trong trình soạn thảo và chọn từ Menu Option Run | Toggle Line Breakpoint. Hoặc ta cũng có thể thay đổi mã của test để chờ chương trình debug kết nối tới. Ví dụ như ta có thể thêm vào hàm khởi tạo các phương thức cần thiết sau:

```
52 public class MyFirstProjectTests extends TestCase {
53     private static final boolean DEBUG = true;
54
55     public MyFirstProjectTests(String name) {
56         super(name);
57
58         if ( DEBUG ) {
59             Debug.waitForDebugger();
60         }
61     }
62 }
```

Khi việc này được hoàn thành thì ta sẽ có một trình gỡ lỗi tiêu chuẩn và thực hiện test.

Ngoài ra nếu không muốn thay đổi code thì có thể đặt breakpoint vào các dòng lệnh và đưa vào các tùy chọn trong lệnh *am instrumentation*

-e debug true Attach to debugger.

Mỗi lần test thực thi, chương trình kiểm thử sẽ chờ cho chương trình debug được định kèm vào. Thực thi dòng lệnh sau để gỡ lỗi cho test

adb shell am instrument -w -e debug true

com.example.myfirstproject.test/android.test.InstrumentationTestRunner

### 3.3. Xây dựng các khối kiểm thử (Block testing)

#### 3.3.1. Các phương thức Assertion

JUnit API bao gồm các class assert kế thừa từ class testcase chứa các phương thức assertion hữu ích cho việc kiểm thử phần mềm. Những phương thức này hữu ích cho rất nhiều trường hợp khác nhau và nạp chồng nhiều kiểu tham số. Chúng có thể nhóm lại với nhau thành từng nhóm tùy thuộc vào điều kiện kiểm tra, ví dụ: assertEquals, assertFalse, assertNotNull, assertNotSame, assertTrue, fail.

Thỉnh thoảng trong quá trình phát triển ta cần tạo một kiểm thử mà không cài đặt một thời điểm xác định tuy nhiên ta muốn tạo một thông báo khi việc tạo bị hoãn lại. Trong trường hợp đó ta sẽ cần sử dụng phương thức fail() mà luôn luôn thất bại (fail) với một thông báo do ta tự đặt như sau

```
48 public void testNotImplementedYet() {
49     fail("Not implemented yet");
50 }
51 }
```

#### 3.3.2. Các phương thức ViewAssertion

Khi kiểm thử giao diện người dùng ta sẽ phải sử dụng nhiều phương thức phức tạp liên quan nhiều đến giao diện View. Trong trường hợp này, Android cung cấp một class với nhiều phương thức assertion là android.test.ViewAsserts sẽ kiểm thử các View và mối quan hệ giữa chúng trên UI.

Dưới đây là các phương thức sẽ cung cấp cho ta các tiện ích khác nhau khi sử dụng chúng

-assertBaselineAligned: xem xét hai View có cùng nằm trên một đường thẳng hay không (trên cùng trục y) baseline.

-assertBottomAligned: xem xét hai View có được căn cảng đáy theo trục y hay không

- `assertGroupNotContains`: xem xét một ViewGroup xác định mà không chứa một ViewChild xác định nào đó
- `assertRightAligned`: xem xét hai view có cùng căn bên phải theo trục x hay không.
- `assertTopAligned`: xem xét hai view có được căn theo cùng cạnh y hay không

Ví dụ sau đây sẽ minh họa cách sử dụng ViewAssert để kiểm thử giao diện người dùng

```

39  public void testUserInterfaceLayout() {
40      final int margin = 0;
41      final View origin = mActivity.getWindow().getDecorView();
42      assertOnScreen(origin, mMessage);
43
44      assertOnScreen(origin, mCapitalize);
45      assertRightAligned(mMessage, mCapitalize, margin);
46  }
47

```

Phương thức `assertOnScreen` sử dụng để kiểm tra các View. Trong trường hợp này ta sử dụng một cây phân cấp View để phân tích, nếu ta không cần di sâu vào các nhánh của cây hoặc cách tiếp cận này không phù hợp với kiểm thử thì ta có thể chọn một nhánh View gốc khác trong cây phân cấp.

-Một số phương thức Viewassertion khác như

- +`assertAssignableFrom`: xem xét hai đối tượng có cùng thuộc một class hay không
- +`assertContainsRegex`: xem xét dạng của đối tượng có đúng với biểu thức chính quy hay không.
- +`assertNotEmpty`: xem xét các tập hợp trong hệ thống có rỗng hay không.
- +`checkEqualsAndHashCodeMethods`: một tiện ích kiểm thử kết quả các phương thức `equal()` và `hashcode()` ở cùng một thời điểm. Kiểm thử phương thức `equals()` cũng áp dụng đối với cả các đối tượng trùng hợp với các kết quả xác định.

Phương thức sau đây sẽ kiểm thử một lỗi error trong suốt quá trình phương thức được gọi thông qua sự kiện click vào một button

```

24  @UiThreadTest
25  public void testNoErrorInCapitalization() {
26      final String msg = "this is a sample";
27
28      mMessage.setText(msg);
29      mCapitalize.performClick();
30
31      final String actual = mMessage.getText().toString();
32      final String notExpectedRegexp = "(?i:ERROR)";
33
34      assertNotContainsRegex("Capitalization found error:", notExpectedRegexp, actual);
35
36  }
37

```

### 3.3.3. Class TouchUtils

Thỉnh thoảng khi kiểm thử giao diện UI, thật hữu ích khi giả lập được các sự kiện touch khác nhau. Có nhiều sự kiện touch nhưng class `android.test.TouchUtils` là class đơn giản nhất để sử dụng. Những phương thức trong class này cho phép giả lập các tương tác với giao diện thông qua việc kiểm thử. `TouchUtils` cung cấp các phương thức tạo ra các sự kiện sử dụng UI hay luồng chính - vì vậy không có xử lý đặc biệt nào ở đây cả.

Các phương thức được hỗ trợ ở đây là

- click vào một View và thả ra
- Nhấp vào một View: đó là chạm vào nó và thả nhanh ra

- Nhấn lùi vào một view
  - Kéo màn hình
  - Kéo View
- Dưới đây là minh họa cho cách sử dụng các sự kiện này

```

2  public void testListScrolling() {
3
4      mListview.scrollTo(0, 0);
5
6      TouchUtil.dragQuarterScreenUp(this, mActivity);
7      TouchUtil.dragQuarterScreenUp(this, mActivity);
8      TouchUtil.dragQuarterScreenUp(this, mActivity);
9      TouchUtil.dragQuarterScreenUp(this, mActivity);
10     TouchUtil.tapView(this, mListview);
11
12     final int expectedItemPosition = 0;
13     final int actualItemPosition = mListview.getFirstVisiblePosition();
14
15     assertEquals("Wrong position",
16                 expectedItemPosition, actualItemPosition);
17
18     final String expected = "Anguilla";
19     final String actual = mListview.getAdapter().getItem(expectedItemPosition).toString();
20
21     assertEquals("Wrong content", actual, expected);
22 }

```

### 3.3.4. Đối tượng giả lập: Mock Object

Ở đây ta sẽ xem xét các Mock Object mà Android SDK đã cung cấp sẵn cho ta trong gói android.test.mock để giúp ta xử lý các tác vụ. Một điểm chung là tất cả các phương thức trong các class đều không có chức năng gì cả và tung ra một ngoại lệ là UnsupportedOperationException

- MockApplication: một class kế thừa từ class Application.
- MockContentProvider: class kế thừa từ class ContentProvider
- MockContentResolver: class kế thừa từ class ContentResolver giúp cho việc test không tác động trực tiếp lên hệ thống thực từ các hệ thống thực.
- MockContext: class kế thừa từ class Context, mục đích là để thêm các phụ thuộc khác.
- MockCursor: class kế thừa từ class Cursor, mục đích là để cài đặt các mã kiểm thử từ các cài đặt Cursor.
- MockDialogInterface: class kế thừa từ class DialogInterface.
- MockPackageManager: class kế thừa từ class PackageManager.
- MockResources: class kế thừa từ class Resource..

#### a) Class MockContext

Đối tượng này có thể được sử dụng để tạo ra các phụ thuộc, các đối tượng giả khác hay quản lý các class kiểm thử. Một rộng class này sẽ cung cấp các phương thức cần thiết hữu ích cho các class kiểm thử sau này.

#### b) IsolatedContext class

Trong một vài trường hợp ta muốn cài đặt Activity với các thành phần khác. Để thực hiện điều này Android SDK cung cấp một class là android.test.IsolatedContext, một class đối tượng giả Context nhằm ngăn chặn tương tác với các hệ thống bên dưới nhưng vẫn thỏa mãn các tương tác với thành phần hay gói khác như Services hay ContentProvider.

#### c) MockContentResolver class

Nếu ta kế thừa từ class này mà không cài đặt các phương thức của nó thì sẽ có một UnsupportedOperationException ném ra. Lý do ta sử dụng class này là để cài đặt các kiểm thử với dữ liệu thật.

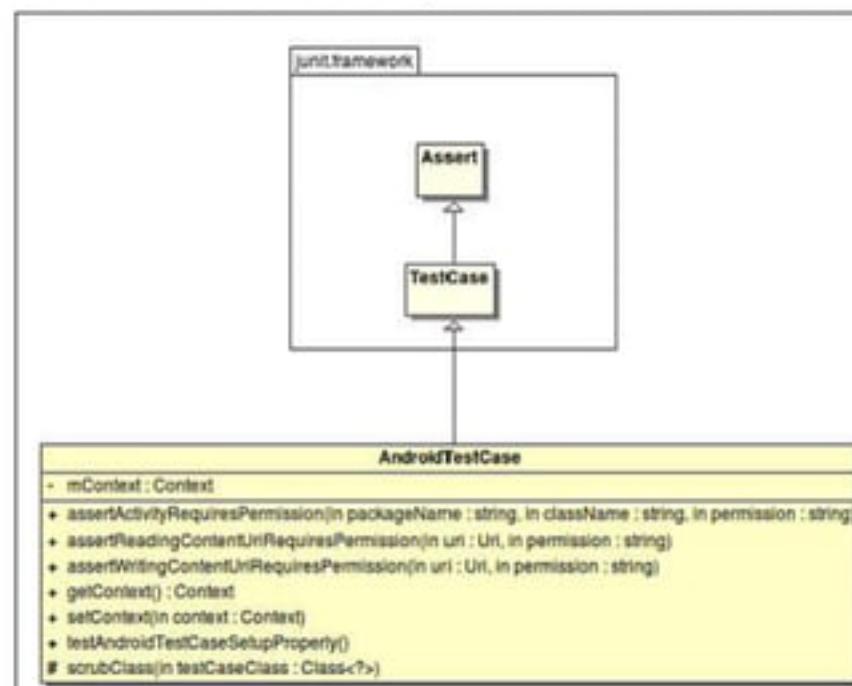
Vấn đề đầu tiên ta gặp phải là có sự không rõ ràng khi ta đưa đối tượng MockContentResolver vào trong ca kiểm thử của ta để sử dụng kiểm thử cơ sở dữ liệu với ContentProvider. Và tương tự với đối tượng của class MockContext. Để giải quyết vấn đề này ta sẽ thảo luận trong phần các nguyên lý kiểm thử android.

### 3.3.5. Xây dựng các class theo các testcase

Đến phần này ta sẽ đi sâu vào tìm hiểu kĩ các class trong Android Testing API

#### a) Android Testcase base class

Class này được sử dụng như là class cơ sở cho việc thiết kế các testcase trong Android. Đây là mô hình UML cho thiết kế testcase này



Sử dụng class này khi ta muốn truy cập vào một Activity Context như tài nguyên, cơ sở dữ liệu hay file hệ thống. Context được lưu trong một trường có tên là mContext, và có thể được sử dụng trong Testcase nếu cần thiết. Phương thức getContext() cũng sẽ cần được gọi đến.

Các class dựa theo testcase này có thể start nhiều Activity bằng cách gọi hàm startActivity(). Có nhiều test case trong Android kế thừa từ lớp cơ sở này là

- ✓ ApplicationTestCase<T extends Application>
- ✓ ProviderTestCase<T extends ContentProvider>
- ✓ ServiceTestCase<T extends Service>

-Phương thức assertActivityRequiresPermission()

Signer	Mô tả
<pre>public void assertActivityRequiresPermission (String packageName, String className, String permission)</pre>	<p>phương thức này kiểm tra một Activity khi bắt đầu chạy có được cho phép bởi một quyền cụ thể nào đó không. Nó đưa vào ba tham số:</p> <ul style="list-style-type: none"> <li>+String packageName: tên package chứa Activity sẽ khởi chạy.</li> <li>+String className: tên của class Activity sẽ khởi chạy.</li> <li>+String permission: quyền cho phép của Activity này với hệ thống.</li> </ul>

-Phương thức assertReadingContentUriRequiresPermission:

Signer	Miêu tả
<pre>public void assertReadingContentUriRequiresPermission(Uri uri, String permission)</pre>	<p>method này kiểm tra quyền đọc một URI xác định có cho phép hay không</p> <p>Tham số của method:</p> <ul style="list-style-type: none"> <li>+uri: URI mà method muốn truy vấn</li> <li>+permission: quyền truy vấn uri đó</li> </ul> <p>Nếu một ngoại lệ SecurityException bị tung ra chưa quyền truy cập thì method này đã thực hiện đúng</p>

-Method assertWritingContentUriRequiresPermission

Signer	Miêu tả
<pre>public void assertWritingContentUriRequiresPermission(Uri uri, String permission)</pre>	<p>Miêu tả: method này kiểm tra quyền được thêm vào một URI xác định có cho phép hay không</p> <p>Tham số của method:</p> <ul style="list-style-type: none"> <li>+uri: URI mà method muốn truy vấn</li> <li>+permission: quyền truy vấn uri đó</li> </ul> <p>Nếu một ngoại lệ SecurityException bị tung ra chưa quyền truy cập thì method này đã thực hiện đúng</p>

### b) Instrumentation

Instrumentation được khởi tạo bởi hệ thống trước khi bắt kì ứng dụng nào được chạy, cho phép nó quản lý bất kì các tương tác nào giữa hệ thống và ứng dụng

Như đối với bắt kì thành phần ứng dụng của Android, để khai báo Instrumentation ta dùng thẻ <instrumentation>. Ví dụ trong một project test, ta sẽ khai báo Instrumentation như sau:

<instrumentation>

    android:targetPackage="com.example.aatg.myfirstproject"

    android:name="android.test.InstrumentationTestRunner"

    android:label="MyFirstProject Tests"/>

Thuộc tính targetPackage định nghĩa tên gói test, label là đoạn văn bản sẽ hiển thị khi Instrumentation được liệt kê.

-Subclass ActivityMonitor: class Instrumentation dùng để quản lý tương tác giữa hệ thống và ứng dụng hoặc các Activity trong quá trình test. Inner class Instrumentation.ActivityMonitor cho phép quản lý từng Activity riêng lẻ của một ứng dụng.

Ví dụ: giả sử ta có một TextView trong Activity chứa một URL mà tự động chở tới một trang web nào đó với thuộc tính autoLink như sau:

```

2   <TextView
3     android:layout_width="fill_parent"
4     android:layout_height="wrap_content"
5     android:text="@string/home"
6     android:layout_gravity="center"
7     android:gravity="center"
8     android:autoLink="web" />
9     android:id="@+id/link" />
10

```

Nếu ta muốn khi click vào link thì một Browser sẽ tự động kích hoạt thì ta sẽ tạo một test như sau:

```

2     public void testFollowLink() {
3         final Instrumentation inst = getInstrumentation();
4         IntentFilter intentFilter = new IntentFilter(Intent.ACTION_VIEW);
5
6         intentFilter.addDataScheme("http");
7         intentFilter.addCategory(Intent.CATEGORY_BROWSABLE);
8         ActivityMonitor monitor = inst.addMonitor(intentFilter, null, false);
9
10        assertEquals(0, monitor.getHits());
11        TouchUtils.clickView(this, mLink);
12        monitor.waitForActivityWithTimeout(5000);
13        assertEquals(1, monitor.getHits());
14        inst.removeMonitor(monitor);
15    }
16

```

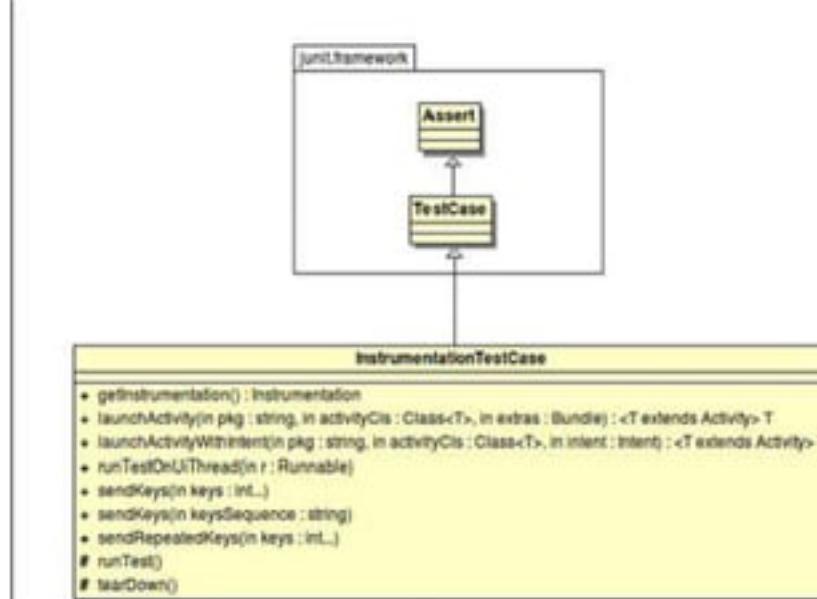
Ở đây ta đã lấy ra một instrumentation, thêm vào một IntentFilter và monitor, giới hạn timeout, cuối cùng là loại bỏ monitors. Sử dụng monitors, ta có thể test ngay cả các tương tác phức tạp giữa hệ thống và các Activity. Đây là một công cụ mạnh mẽ trong kiểm thử tích hợp.

#### -Class testcase Instrumentation

Class InstrumentationTestCase là lớp cha cho rất nhiều testcase mà có truy cập đến Instrumentation. Dưới đây là các subclass quan trọng của class này

- ↳ ActivityTestCase
- ↳ ProviderTestCase<T extends ContentProvider>
- ↳ SingleLaunchActivityTestCase<T extends Activity>
- ↳ SyncBaseInstrumentation
- ↳ ActivityInstrumentationTestCase<T extends Activity>
- ↳ ActivityUnitTestCase<T extends Activity>

Đây là mô hình UML về class InstrumentationTestCase và những subclass của nó



Class InstrumentationTestCase nằm trong gói android.test, và kế thừa từ class junit.framework.TestCase

+Các method launchActivity và launchActivityWithIntent

Đây là những phương thức tiện ích dùng để kích hoạt các Activity trong test. Nếu tham số Intent thứ 2 không được xác định thì một Intent mặc định sẽ được sử dụng. Chữ kí của phương thức:

```
public final T launchActivity( String pkg, Class<T> activityCls, Bundle extras)
```

Nếu cần tạo một Intent riêng, ta có thể sử dụng cách sau để thêm vào một Intent

```
public final T launchActivityWithIntent( String pkg, Class<T> activityCls, Intent intent)
```

+ Các method sendKeys và sendRepeatedKeys

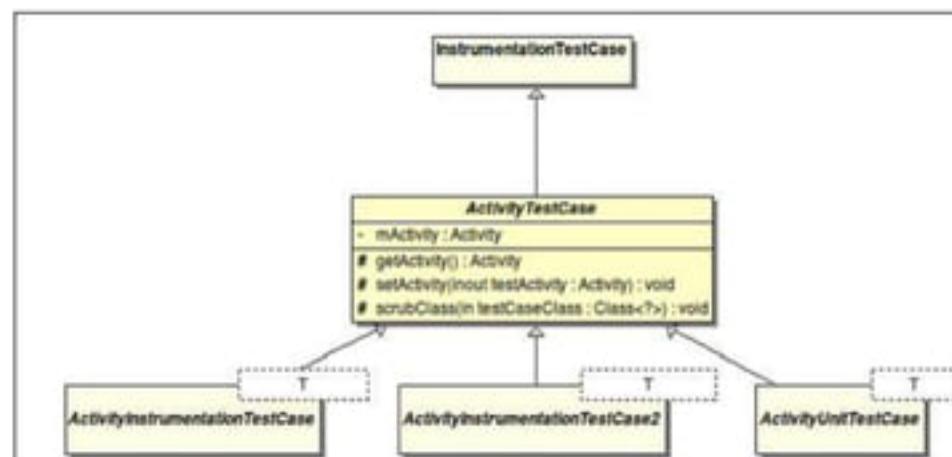
Khi kiểm thử các Activity, ta sẽ cần giả lập tương tác giữa bàn phím ảo và các phím để hoàn thành các trường dữ liệu hay điều hướng giữa các thành phần trong ứng dụng.

+ Method runTestOnUiThread

Đây là một phương thức trợ giúp cho việc chạy các phần test trên luồng UI. Ngoài ra, ta cũng có thể sử dụng kí hiệu @UiThreadTest để chạy các test trên luồng UI. Nhưng đôi khi ta chỉ cần chạy một phần của testcase trên luồng UI vì những phần khác có thể không phù hợp hay đang sử dụng các phương thức trợ giúp khác. Các mô hình phổ biến nhất là thay đổi focus trước khi gửi phím.

- Class ActivityTestCase

Đây là class chủ yếu hỗ trợ cho việc truy cập vào Instrumentation. Ta có thể sử dụng class này nếu muốn cài đặt các hành vi đặc biệt của testcase. Sau đây là mô hình UML của class này và các subclass của nó:

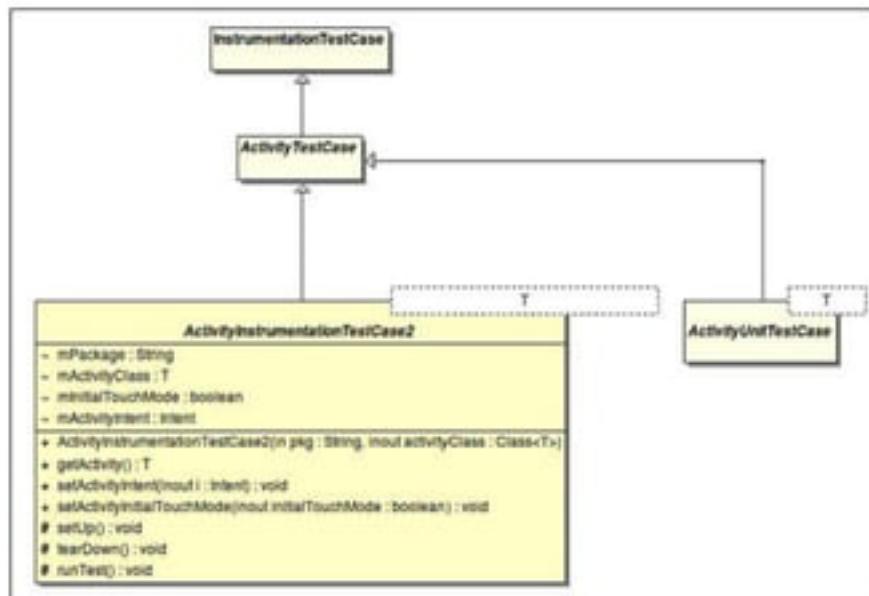


Abstract class ActivityTestCase kế thừa từ InstrumentationTestCase và các class khác (ActivityInstrumentationTestCase, ActivityInstrumentationTestCase2 và ActivityUnitTestCase).

Method scrubClass trong class này được kích hoạt từ phương thức tearDown() để giải phóng các biến đã được khởi tạo để tránh phải tham chiếu tới chúng. Điều này để ngăn chặn việc rò rỉ bộ nhớ trong các testcase lớn. Một ngoại lệ IllegalStateException sẽ bị ném ra nếu có vấn đề truy cập các biến.

- Class ActivityInstrumentationTestCase2

Class này sẽ cung cấp cho các chức năng kiểm thử đối với một Activity riêng lẻ. Class này truy cập vào Instrumentation và sẽ tạo một Activity trong kiểm thử sử dụng kiến trúc hệ thống bằng cách gọi phương thức InstrumentationTestCase.launchActivity(). Đây là biểu đồ UML:



Class `ActivityInstrumentationTestCase2` kế thừa từ class `ActivityTestCase`. Activity có thể thao tác và giám sát sau khi tạo xong. Kiểm thử chức năng là rất hữu ích cho việc kiểm thử tương tác giữa giao diện người dùng như sự kiện già lập hành vi của người dùng.

+Hàm khởi tạo: `ActivityInstrumentationTestCase2(Class<T> activityClass)`: sẽ được khởi tạo với một thê hiện của một Activity tương tự cùng class.

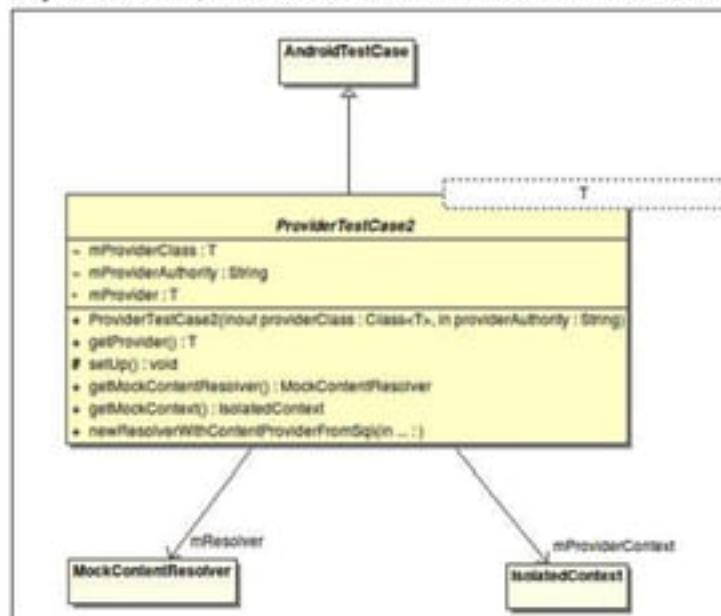
+Hàm `setUp`: Đây là hàm để khởi tạo các biến và các thành phần cố định khác của ứng dụng

+Hàm `tearDown`: Hàm này để giải phóng các biến đã được khởi tạo trong hàm `setUp`

+ Hàm `testPreconditions`: Hàm này để kiểm tra các điều kiện khởi tạo để chạy các test của chúng ta đúng

-Class `ProviderTestCase2<T>`

Đây là class được thiết kế để kiểm thử ContentProvider. Sau đây là mô hình UML cho class này



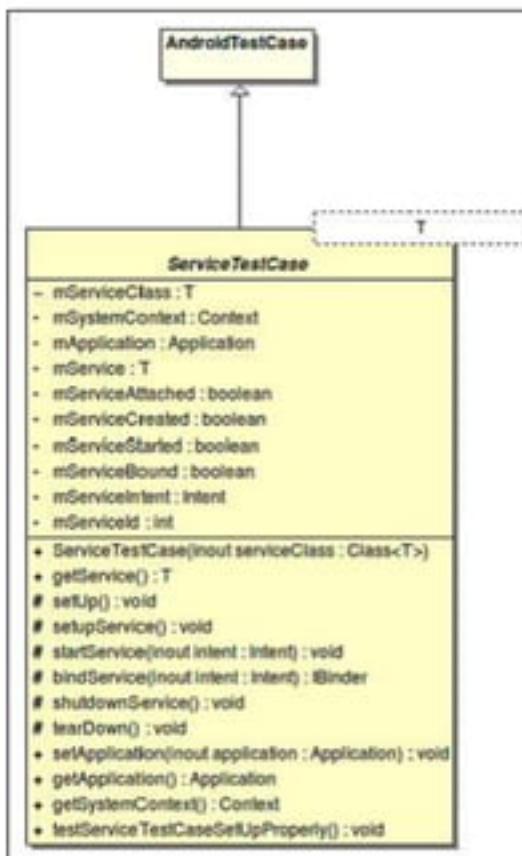
Class `android.test.ProviderTestCase2` cũng kế thừa từ class `AndroidTestCase`

+Hàm khởi tạo: `ProviderTestCase2(Class<T> providerClass, String providerAuthority)`

Tham số cho hàm class `ContentProvider` và tham số thứ hai là xâu xác thực của class `ContentProvider` đó.

-Class `ServiceTestCase<T>`

Class này dùng để kiểm tra Service trong Android. Sau đây là mô hình UML của class



Các method để thực hiện vòng đời Service cũng được đưa vào class này như setupService, startService, bindService, và shutDownService.

+Hàm khởi tạo: ServiceTestCase(Class<T> serviceClass)

Tham số đầu vào là một class Service để kích hoạt chính class này.

- Class TestSuiteBuilder.FailedToCreateTests

Đây là một class đặc biệt để chỉ ra lỗi trong suốt quá trình build. Đó là trong suốt quá trình tạo một dãy các test, nếu có một lỗi được tìm thấy thì sẽ có một ngoại lệ bị ném ra.

## 4. Ví dụ về TDD trên Android

Sau khi đã tìm hiểu kĩ về các class trong Android testing API, em xin trình bày một ví dụ về quy trình phát triển phần mềm TDD áp dụng trên Android, qua đó cũng sẽ nói chi tiết hơn về các class được hỗ trợ kiểm thử của Android testing API

### 4.1.Tạo mới các Project

- Project minh họa này là một ứng dụng để chuyển đổi nhiệt độ từ độ C sang độ F và ngược lại, các yêu cầu của ứng dụng bao gồm

+ chuyển đổi nhiệt độ C sang độ F và ngược lại.

+ UI gồm hai EditText cho phép user nhập vào giá trị độ C (hay độ F) cần chuyển đổi

+ Khi một giá trị nhiệt độ được nhập vào một trường thi EditText kia sẽ tự động cập nhật giá trị

+ Nếu giá trị User nhập vào không hợp lệ thi sẽ hiển thị thông báo lỗi ngay tại nơi nhập liệu

+ Các EditText khi khởi chạy ứng dụng sẽ trống, không chứa giá trị nào cả

+ Giá trị nhập vào là các số thập phân với hai chữ số ở phần thập phân

+ Chữ số được căn phai EditText

+ Giá trị phải được giữ lại khi ứng dụng bị Pause

-Thiết kế UI



Tiếp theo, như đúng quy trình phát triển TDD, ta sẽ xây dựng các testcase và sau đó viết mã thực thi các testcase để các testcase đều pass như mô hình sau

#### 4.2.Xây dựng các TestCase

##### 4.2.1.Kiểm thử hàm khởi tạo:

Mục đích của testcase này là kiểm tra xem các đối tượng đã được khởi tạo đúng chưa? (có bị null không)

STT	Mục tiêu kiểm thử	Kết quả
01	Xác định xem activity dùng trong testcase đã được khởi tạo trong hàm setUp()	Yes
02	Xác định các EditText đã được khởi tạo chưa trong hàm setUp()	Yes
03	Xác định xem giá trị ban đầu của các EditText có trống hay không	Yes

##### 4.2.2.Kiểm thử giao diện

Phần này sẽ kiểm tra các đặc tính của giao diện có phù hợp với yêu cầu đặt ra ban đầu không

STT	Mục tiêu kiểm thử	Kết quả
01	Xác định các thành phần View có hiển thị trên UI screen không? (EditText,...)	Yes
02	Kiểm thử vị trí tương đối giữa các thành phần View (các EditText, TextView,...) có đúng với thiết kế ban đầu không?	Yes
03	Kiểm thử kích cỡ các font chữ của các thành phần View (font chữ phải đúng 24dip)	Yes
04	Kiểm thử các thuộc tính căn trái, phải, độ rộng của các View đúng với yêu cầu không?	Yes

Kết quả sau khi chạy các Testcase

Finished after 33.662 seconds		
Runs:	Errors:	Failures:
13/13	0	0
com.example.temperatureconverter.test.TemperatureConverterActivityTest (33.333 s)		
testAlignment (1.023 s)		
testCelsiusInputFieldCoverEntireScreen (0.174 s)		
testFahrenheitInputFieldCoverEntireScreen (0.970 s)		
testFahrenheitToCelsiusConversion (2.705 s)		
testFieldsOnScreen (2.981 s)		
testFieldsShouldStartEmpty (2.448 s)		
testFontSizes (2.225 s)		
testH茱InputFields (1.025 s)		
testInputFilter (4.793 s)		
testJustification (2.340 s)		
testMargins (2.334 s)		
testPreconditions (2.878 s)		
testVirtualKeyboardSpaceReserved (2.799 s)		

#### 4.2.3. Kiểm thử chức năng

Phần này sẽ trình bày các testcase về kiểm thử chức năng chính của ứng dụng là chuyển giá trị nhập vào từ độ C sang độ F và ngược lại:

STT	Mục tiêu, yêu cầu kiểm thử	Kết quả
01	Kiểm thử chức năng chuyển giá trị độ C sang °F	Pass
02	Kiểm thử chức năng chuyển giá trị độ F sang °C	Pass
03	Kiểm thử chức năng lọc giá trị user nhập vào (với $^{\circ}\text{C} \geq -273$ , $^{\circ}\text{F} \geq -459$ ), nếu không thỏa mãn sẽ thông báo lỗi tại nơi nhập	Pass
04	Kiểm thử chức năng tự động hiển thị kết quả chuyển đổi khi user nhập xong giá trị một trường	Pass

Kết quả sau khi chạy các testcase này

Finished after 0.22 seconds		
Runs:	Errors:	Failures:
4/4	0	0
com.example.temperatureconverter.test.TemperatureConverterTest (0.365 s)		
testCelsiusToFahrenheit (0.000 s)		
testExceptionForLessThanAbsoluteZeroC (0.000 s)		
testExceptionForLessThanAbsoluteZeroF (0.000 s)		
testFahrenheitToCelsius (0.365 s)		

#### 4.2.4. Kiểm thử tích hợp

Kiểm thử các thành phần trong ứng dụng kết hợp với cơ sở hệ thống: Activity, Context, Application, ...

a) *Activity*

STT	Mục tiêu kiểm thử	Kết quả
01	Kiểm thử khởi tạo Activity trong ứng dụng đã khởi tạo thành công chưa?	Pass
02	Kiểm thử vòng đời của Activity	Pass

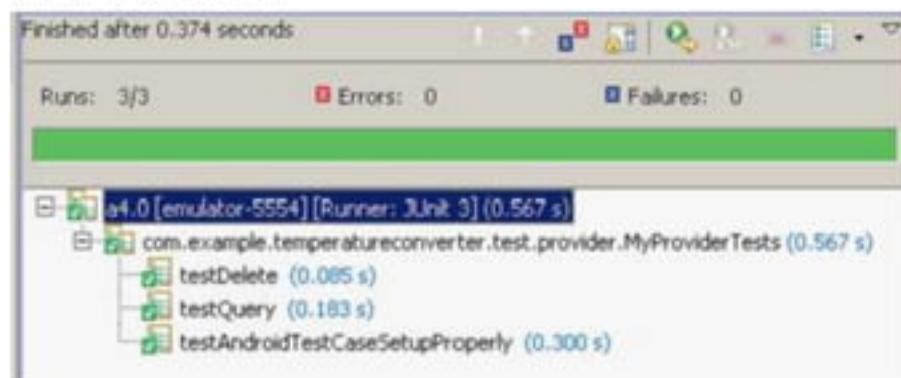
Kết quả chạy testcase

Finished after 2.470 seconds		
Runs:	Errors:	Failures:
1/1	0	0
com.example.temperatureconverter.test.TemperatureConverterActivityUnitTest (2.470 s)		
testLifecycleCreate (1.243 s)		
testPreconditions (1.227 s)		

b) *ContentProvider*

STT	Mục tiêu kiểm thử	Kết quả
01	Kiểm thử hàm truy vấn (query) trong ContentProvider	Pass
02	Kiểm thử hàm xóa (delete) trong ContentProvider	Pass

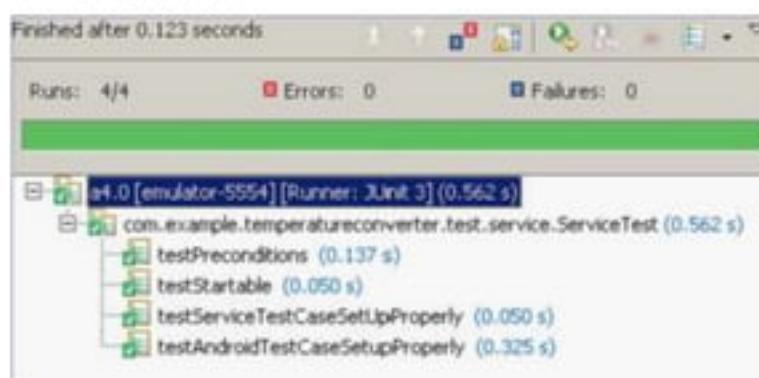
Kết quả thực hiện test



c) Service

STT	Mục tiêu kiểm thử	Kết quả
01	Kiểm thử service đã được khởi tạo hay chưa	Pass
02	Kiểm thử đã tạo kết nối tới service thành công không	Pass

Kết quả minh họa



### 4.3. Street Test

Như đã trình bày ở phần trên, Android hỗ trợ cho phép ta kiểm thử khả năng chịu tải (Street Test) của ứng dụng thông qua một công cụ kiểm thử tự động đã được tích hợp sẵn là Monkey runner.

Lý thuyết về monkey (monkey theorem) chỉ ra rằng một con khỉ nếu gõ vào bàn phím trong một khoảng thời gian vô hạn thì sẽ hoàn thành được một đoạn văn bản cho trước, có thể đó là cả một tác phẩm văn học nổi tiếng như Kinglear của William Shakespeare.

Lý thuyết này áp dụng vào Android ta sẽ thấy monkey runner sẽ tự động sinh ra ngẫu nhiên hàng loạt các sự kiện tương tác với ứng dụng như chạm, bấm, xoay màn hình,... mà có thể làm ứng dụng bị đóng (force close) trong một khoảng thời gian là hữu hạn.

Để chạy Monkey runner ta sẽ sử dụng công cụ adb của Android, tại màn hình command prompt, ta thực hiện lệnh sau:

```
adb -e shell monkey -p com.example.temperatureconverter -v -v 1000
```

Monkey runner sẽ gửi các sự kiện tới package của ứng dụng (-p) và sẽ hiển thị log trong Logcat dạng verbose manner(-v -v). Số lượng các sự kiện ở đây là 1000.



Hiển thị trên logcat

Và trên command

```
C:\Windows\System32\cmd.exe
: schedstat=< 1044238368 290688324 1660 > utm=69 stm=35 core=0
at dalvik.system.NativeStart.run(Native Method)

"JDWP" daemon prio=5 tid=4 UMINIT
: group="system" sCount=1 dsCount=0 obj=0x410168c8 self=0x9eaaff
: systid=149 nice=0 sched=0/0 cgrp=default handle=1313352
: schedstat=< 21010687 218165211 21 > utm=2 stm=0 core=0
at dalvik.system.NativeStart.run(Native Method)

"Signal Catcher" daemon prio=5 tid=3 RUNNABLE
: group="system" sCount=0 dsCount=0 obj=0x410167d8 self=0xic8a58
: systid=148 nice=0 sched=0/0 cgrp=default handle=1259400
: schedstat=< 21542257 87943155 8 > utm=0 stm=1 core=0
at dalvik.system.NativeStart.run(Native Method)

"GC" daemon prio=5 tid=2 UMINIT
: group="system" sCount=1 dsCount=0 obj=0x410166f0 self=0xicd190
: systid=145 nice=0 sched=0/0 cgrp=default handle=1260864
: schedstat=< 202592094 1241968168 46 > utm=16 stm=4 core=0
at dalvik.system.NativeStart.run(Native Method)

----- end 143 -----

----- pid 177 at 2013-05-02 07:41:57 -----
Cmd line: com.android.phone

DALVIK THREADS:
(mutexes: tll=0 tsl=0 tscl=0 ghl=0)
"main" prio=5 tid=1 NATIVE
: group="main" sCount=1 dsCount=0 obj=0x409c1468 self=0x12818
: systid=177 nice=0 sched=0/0 cgrp=default handle=1074002952
: schedstat=< 2557000863 17173188191 2854 > utm=185 stm=79 core=0
at android.os.MessageQueue.nativePollOnce(Native Method)
at android.os.MessageQueue.next(MessageQueue.java:118)
```

#### **4.4. Behavior Driven Development (BDD)**

#### 4.3.1. Khái niệm:

Theo Wikipedia, BDD là một quy trình phát triển phần mềm dựa trên TDD, BDD kết hợp các nguyên lý, kĩ thuật chung của TDD với những ý tưởng từ domain-driven design (DDD, một cách tiếp cận phát triển phần mềm cho những yêu cầu phức tạp bằng cách thực thi các mô hình tiền hóa) và phân tích thiết kế hướng đối tượng (OOAD) để cung cấp cho các nhà phát triển phần mềm và khách hàng một công cụ chung trong quy trình phát triển phần mềm.

Có thể hiểu đơn giản BDD là sự tiền hóa và kết hợp của cả TDD và kiểm thử chấp nhận Acceptance Testing. BDD lần đầu tiên được giới thiệu bởi Dan North vào năm 2003 để miêu tả một kĩ thuật mà tập trung vào sự kết hợp giữa lập trình viên và những bên liên quan bằng cách sử dụng một quy trình thường được gọi là phát triển phần mềm *outside-in*. Mục tiêu chính của nó là làm thỏa mãn nhu cầu kinh doanh của khách hàng.

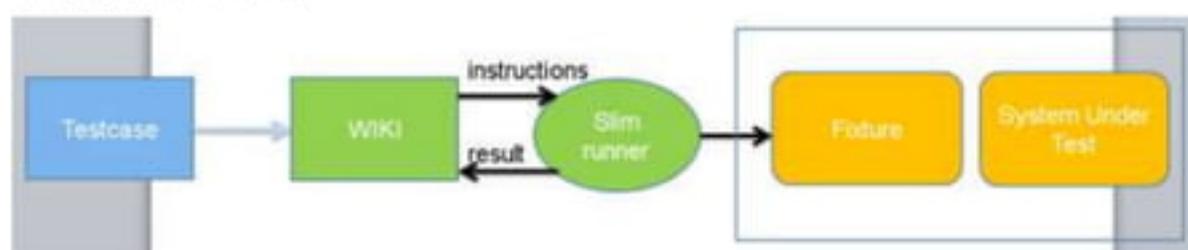
BDD được phát triển từ một thí nghiệm tư duy dựa trên kĩ thuật của lập trình ngôn ngữ tư duy (Neuro Linguistic Programming, NLP).

#### 4.3.2. Fitnesse

Fitnesse là một công cụ cộng tác phát triển phần mềm, một framework mã nguồn mở được tạo ra cho việc kiểm thử. Nó già lập sự cộng tác trong phát triển phần mềm bằng cách cung cấp một framework kiểm thử mạnh mẽ WIKI cho phép cả khách hàng, lập trình viên, tester chỉnh sửa test theo một cách độc lập với nền tảng. Fitnesse dựa trên framework kiểm thử tích hợp của Ward Cunningham và bây giờ được phát triển bởi Robert C. Martin.

Fitnesse được thiết kế để hỗ trợ kiểm thử chức năng (hay kiểm thử chấp nhận) bằng cách tích hợp ở cấp độ doanh nghiệp. Mặc dù Fitnesse không hỗ trợ BDD theo một dạng đặc biệt nào đó nhưng bằng cách kết hợp các Script test và bảng kịch bản cung cấp một công cụ tốt cho BDD.

##### Kiến trúc của Fitnesse:



Fitnesse làm việc bằng cách thực thi các trang Wiki gọi các Fixtures được viết tùy biến. Fixtures ở đây đóng vai trò như cầu nối giữa các trang Wiki và System Under Test (SUT), hệ thống mà ta muốn kiểm thử. Fixtures có thể được viết bằng nhiều ngôn ngữ lập trình như Java, C#, Ruby,... Bất cứ khi nào Wiki test thực thi, Fixtures làm việc bằng cách gọi SUT với tham số phù hợp, thực thi một phần logic nghiệp vụ trong hệ thống phần mềm và đưa kết quả nếu có của SUT trả về các trang Wiki và hiển thị kết quả test có pass hay không.

Fitnesse có hai hệ thống test, SLIM và FIT. Trong đó FIT là hệ thống kiểm thử cũ và đã không còn được phát triển nữa. Tuy nhiên những kế hoạch gần đây cho thấy FIT đang dần được phát triển tiếp. Song FIT khó bảo trì và phức tạp trên nhiều nền tảng khác nhau nên SLIM đã được tạo ra nhằm khắc phục các vấn đề này. SLIM cũng là một phiên bản gọn nhẹ dựa trên FIT. Một trong những mục tiêu của SLIM là có thể cài đặt được nhiều ngôn ngữ khác nhau. Trái ngược với FIT, SLIM không yêu cầu bắt kì sự phụ thuộc nào vào framework Fitnesse trong mã Fixtures, điều này làm cho việc viết mã trong Fixtures trở nên dễ dàng hơn.

##### a) Cài đặt Fitnesse

Đầu tiên ta cần download bộ cài đặt Fitnesse từ trang chủ, sau đó trong commandline gõ lệnh sau:

```
java -jar fitnesse.jar -p 9000
```

Sau đó vào trình duyệt và nhập vào địa chỉ <http://localhost:9000> ta sẽ thu được kết quả sau

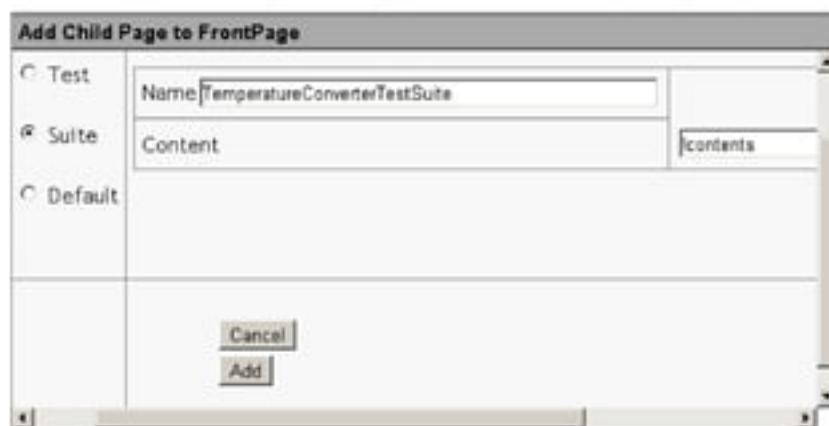
The screenshot shows the FitNesse FrontPage interface. On the left is a sidebar with buttons for Edit, Properties, Where Used, Search, Files, Versions, Recent Changes, User Guide, and Test History. The main content area has a title "WELCOME TO FitNesse!" and a subtitle "THE FULLY INTEGRATED STAND-ALONE ACCEPTANCE TESTING FRAMEWORK AND WIKI". Below this is a note: "To add your first 'page', click the [Edit](#) button and add a [WikiWord](#) to the page." A "To Learn More..." section contains links to "A One-Minute Description", "A Two-Minute Example", "User Guide", and "Acceptance Tests". At the bottom of the sidebar is the text "FitNesse v20100303". In the main content area, there is a "My Tests" section with a link to "TemperatureConverterTests".

Từ đây ta dễ dàng thêm các test muốn để bắt đầu kiểm thử (bằng cách nhấp vào nút add child, sau đó thêm các subpage chứa kịch bản kiểm thử)

#### b) Thực thi các test

Đầu tiên để có thể chạy các testcase trong FitNesse, ta cần tạo ra một trang subwiki chứa các test ta sẽ định chạy, cách tạo một trang mới như sau:

Click chuột vào link [add child], FitNesse sẽ xuất hiện hộp thoại như sau:



Điền đầy đủ các thông tin cho ChildPage và sau đó nhấn Add, kết quả nếu thực hiện thành công

## TemperatureConverterTestSuite [add child]

Contents:

Tiếp tục, tại child page này ta tạo một child page của nó, cách làm tương tự như trên. Sau khi đã tạo xong các trang cần thiết ta sẽ tiến hành chỉnh sửa các thông số của trang vừa tạo để phù hợp với Testcase, nhấp vào Menu Edit bên phải của trang sẽ ra một khung edit text cho phép ta khai báo các thành phần cần thiết:

-Dòng đầu tiên là khai báo hệ thống test, sử dụng SLIM như đã trình bày bên trên.

- Tiếp theo là khai báo đường dẫn đến các thư mục classes chứa các class đã biên dịch của ứng dụng, ở đây là TemperatureConverter.
- Dòng tiếp là import package chứa class dùng để test
- Tiếp đến là bảng quyết định decision table chứa các giá trị mà ta muốn kiểm thử, tên của bảng này là tên của class bên trên

#### EDIT PAGE:

```

`define TEST_SYSTEM (alias)
`path C:\Users\quynh\workspace\TemperatureConverter\bin\classes
`path C:\Users\quynh\workspace\TemperatureConverterTest\bin\classes
`import
`com.example.temperatureconverter.test.fitnesse

`contents
Here is detail of test

`TestBDDWithFitnesse
`celcius|convertToCelsiusToFahrenheit|
0.0 |--> 32 |
100.0 |212.0 |
-1.0 |30.2 |
-100.0 |-140.0 |
32.0 |89.6 |
-40.0 |-40.0 |
-273.0 |-- -459.4 |
-273 |-- -459.4 |
-273 |-- -459 |
-273 |-- -459.4000000000003 |
-273 |-- -459.4000000000003 |
-273 |-- -459.41 < _ < -459.40 |
-274.0 |Invalid temperature: -274.00C below absolute zero|
15.99 |--> 41.99 |
178 | 120<_<190|

```

#### c) Kết quả

Sau khi đã xong các bước bên trên, ta nhấn vào menu Test bên phải để thực thi Testcase, nếu kết quả đúng như mong đợi thì sẽ có màn hình xanh

FrontPage: TemperatureConverterTestSuite.

### TemperatureConverterTestCase

Test Result: passed

**Assertions: 18 right, 0 wrong, 0 Ignored, 0 exceptions**

Variables defined: TEST\_SUITE=case

Variables: C:\Users\quynh\workspace\TemperatureConverter\bin\classes

Imports: C:\Users\quynh\workspace\TemperatureConverterTest\bin\classes

Import

com.example.temperatureconverter.test.Fitnesse

Contents:

Here is detail of test

TestBDDWithFitnesse	
celcius	convertToCelsiusToFahrenheit
0.0	32.0--32
100.0	212.0
-1.0	30.2
-100.0	-140.0
32.0	89.6
-40.0	-40.0
-273.0	-459.4--459.4
-273	-459.4000000000003--459.4
-273	-459.4000000000003--459
-273	-459.4000000000003--459.4000000000003
-273	-459.41--459.4000000000003--459.40
-274.0	Invalid temperature: -274.00C below absolute zero

#### d) GivenWhenThen

Với việc sử dụng các annotation trong GivenWhenThen framework, ta sẽ dễ dàng ăn đi sự phức tạp trong các đoạn mã lập trình, thay vào đó ta chỉ cần sử dụng các biểu thức chính quy để so khớp với các phương thức mà ta muốn kiểm thử. Điều này giúp cho khách hàng hay các bên liên quan trong quy trình phát triển phần mềm dù không hiểu nền tảng bên dưới vẫn có thể tham gia kiểm thử được ứng dụng. GivenWhenThen cũng là một framework dựa trên Fitnesse và SLIM nên việc sử dụng cũng giống như Fitnesse, để cài đặt GivenWhenThen ta cũng cần download bộ cài đặt từ trang chủ và sau đó thực hiện như giống với Fitnesse. Các bước trong GivenWhenThen:

- Given: là các điều kiện tiên quyết cho testcase
- Wen: miêu tả hành động của user
- Then: kết quả của hành động

Chính kiến trúc đơn giản và sử dụng ngôn ngữ tự nhiên giúp cho mọi khách hàng có thể hiểu được ứng dụng hoạt động như thế nào. Áp dụng vào chương trình TemperatureConverter, ta sẽ kiểm thử chức năng chuyển đổi giá trị  $^{\circ}\text{C}$  sang  $^{\circ}\text{F}$ , kết quả sau khi thực hiện testcase trên Fitnesse

```
FitNesse Test Results [Details]
Assertions: 3 right, 0 wrong, 0 ignored, 0 exceptions (0.966 seconds)
* Precondition Listener
* Included page: /TemperatureConverterTests.html

@Given I am using the TemperatureConverter
@When I enter 100 into Celsius field
@Then I obtain 212 in Fahrenheit field

given: I am using the TemperatureConverter
when: I enter 100 into Celsius field
then: I obtain 212 in Fahrenheit field

given: I am using the TemperatureConverter
when: I enter -100 into Celsius field
then: I obtain -140 in Fahrenheit field

given: I am using the TemperatureConverter
when: I enter -100 into Fahrenheit field
then: I obtain -73.33 in Celsius field
Show Then Celsius -73.33333333333333
```

#### 4.4.Kiểm thử hiệu năng

Khi phát triển ứng dụng, đặc biệt là trên các thiết bị di động, do sự giới hạn về bộ nhớ cũng như tốc độ xử lý và thời gian làm việc của CPU nên vẫn đề hiệu năng cần được tối ưu sao cho chương trình sử dụng ít tài nguyên nhất mà vẫn thực hiện đầy đủ các chức năng yêu cầu. Tài nguyên ở đây có thể bao gồm bộ nhớ, lượng tiêu thụ Pin, thời gian CPU xử lý. Do đó vẫn đề theo dõi quá trình xử lý của bất kì một lệnh hay hàm thực thi nào của chương trình cũng là điều cần thiết. Hiện có nhiều phương pháp để đánh giá hiệu quả hoạt động của một chương trình trên Android như sử dụng công cụ Trace View được Android hỗ trợ sẵn qua công cụ phân tích log, các class Debug của Android API, hay như đếm số lệnh thực hiện bởi chương trình từ lúc khởi chạy cho tới khi bị giải phóng tài nguyên,...

##### 4.4.1.Tính thời gian thực thi của một hàm hay sự kiện nào đó (TextChanged, onClick,...) hay tổng số lệnh thực hiện của chương trình

Để tính thời gian thực thi của bất kì một sự kiện nào đó đơn giản là ta sẽ xác định thời gian lúc CPU bắt đầu thực thi sự kiện đó tới khi kết thúc xong công việc, giả sử trong ứng dụng TemperatureConverter thì sự kiện có lẽ là quan trọng nhất đó là onTextChanged khi ta nhập giá trị cho một trường nhiệt độ thì trường nhiệt độ kia sẽ tự động cập nhật giá trị, do đó trước khi thực thi sự kiện này ta sẽ đặt vào đầu hàm một biến ghi nhận giá trị thời gian tại thời điểm bắt đầu thực hiện là:

```

if (BENCHMARK_TEMPERATURE_CONVERSION) {
    t0 = System.currentTimeMillis();
    Debug.startMethodTracing("TemperatureChangedWatcher");
}

```

Và sau khi thực hiện xong ta sẽ tính thời gian đã trôi qua

```

if (BENCHMARK_TEMPERATURE_CONVERSION) {
    Debug.stopMethodTracing();
    t = System.currentTimeMillis() - t0;
    Log.e("On Text Changed ==>", "t= " + t);
}

```

Kết quả sẽ được in ra Logcat của Eclipse: (Đơn vị thời gian tính theo millisecond)

Log	TOTAL
On Text Changed ==>	t= 1996

Một cách tiếp cận khác trong vấn đề này là đếm số lệnh mà chương trình đã sử dụng để chạy (lệnh ở đây không phải là số dòng mã lập trình vì để chạy chương trình thì Android sẽ phải gọi nhiều lệnh khác hỗ trợ tùy thuộc mỗi ứng dụng). Áp dụng vào ứng dụng TemperatureConverter ta sẽ đặt một object của class InstructionCount để đếm số lệnh ngay khi Activity bắt đầu vòng đời của nó

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    instructionCount = new Debug.InstructionCount();
    instructionCount.resetAndStart();
}

```

và khi kết thúc ta sẽ hiển thị ra tổng số lệnh mà chương trình đã cần

```

@Override
protected void onDestroy() {
    if (instructionCount.collect()) {
        Log.e("Total instructions executed==>",
              "" + instructionCount.globalTotal());
        Log.e("Method invocations==>",
              "" + instructionCount.globalMethodInvocations());
    }
    super.onDestroy();
}

```

Kết quả cũng sẽ được hiển thị tại Logcat

com.example.temp	On Text Changed ==>	t= 1996
	Total instructions executed==>	650195
	Method invocations==>	52237

#### 4.4.2. Sử dụng Trace View

Traceview là một giao diện đồ họa ghi lại log thực thi, được tạo bằng class Debug để theo dõi quá trình chạy chương trình. Ngoài ra Traceview cũng giúp ta debug và đánh giá hiệu năng của ứng dụng. Traceview tái file log và sẽ hiển thị lên một giao diện đồ họa gồm 2 panel:

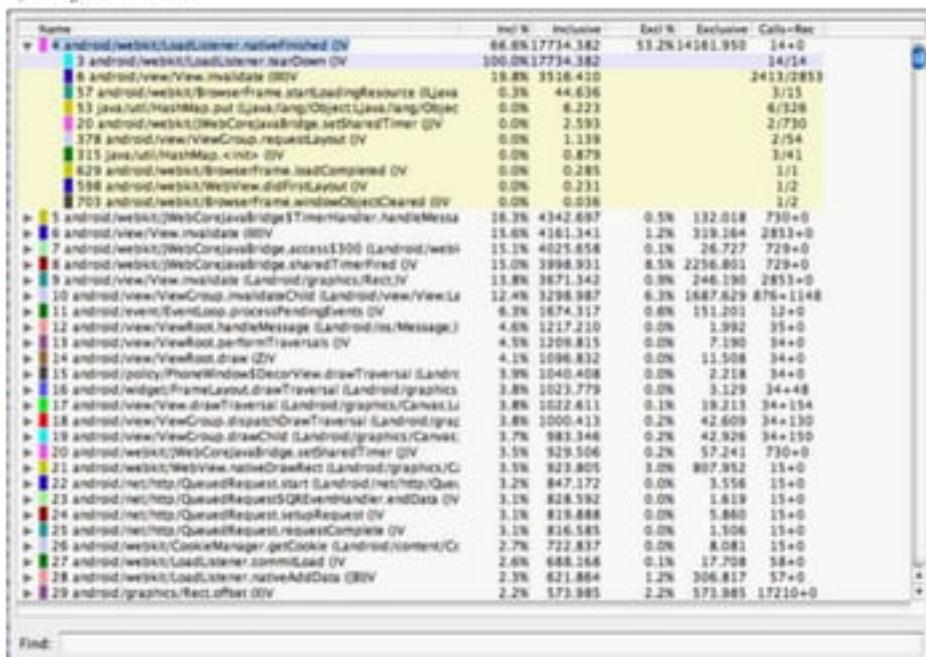
- Timeline Panel: miêu tả mỗi luồng (thread) và phương thức bắt đầu và kết thúc.
- Profile Panel: cung cấp một cách chi tiết những gì xảy ra trong một hàm.

a) Timeline panel:



Mỗi luồng thực thi được biểu diễn trên một dòng riêng biệt với thời gian tăng dần về bên phải. Mỗi phương thức được hiển thị theo một màu sắc riêng. Đường thẳng mảnh nằm dưới dòng đầu tiên thể hiện mức độ của tất cả các lời gọi đối với phương thức được chọn, ở đây là LoadListener.nativeFinished() và nó sẽ hiển thị chi tiết trong Profile Panel.

#### b) Profile Panel:



Bảng trên biểu diễn cả thời gian inclusive time và exclusive time. Exclusive time chỉ thời gian mà hàm sử dụng để thực thi. Inclusive time là thời gian chạy của hàm cộng với thời gian của bất kỳ hàm nào mà có gọi đến nó. Các hàm mà gọi đến hàm đó gọi là hàm cha, các hàm mà hàm đó gọi đến là hàm con. Khi một hàm được click vào nó sẽ hiển thị ra các hàm cha và hàm con. Hàm cha hiển thị trong vùng màu tím, hàm con trong vùng màu vàng. Cột cuối cùng của bảng thể hiện số lời gọi đến hàm (bao gồm cả lời gọi đệ quy), trong ví dụ thì LoadListener.nativeFinished() có 14 lời gọi đến.

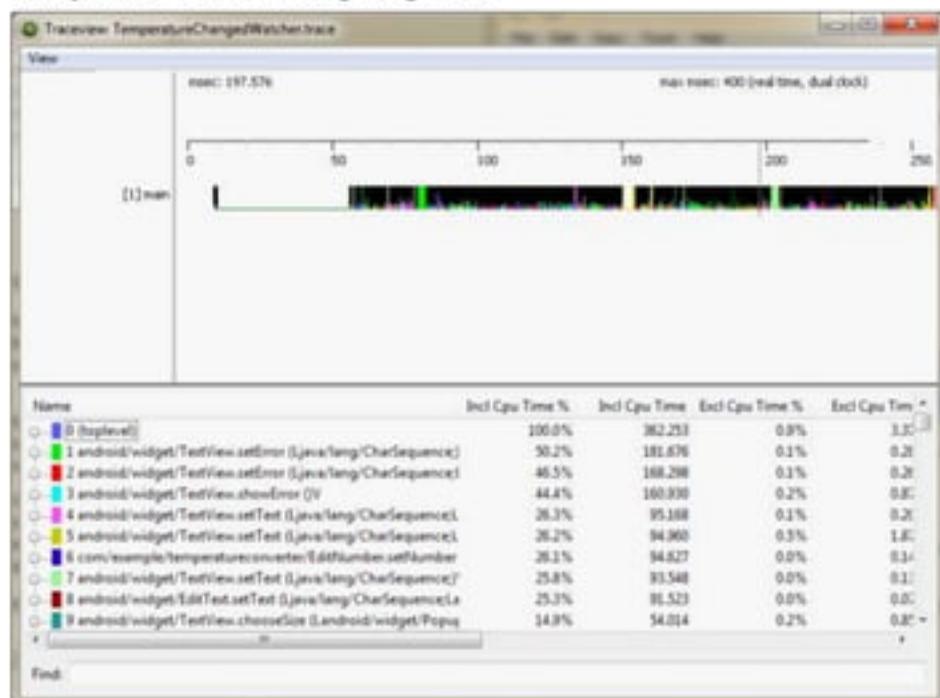
Để tạo ra file log này ta sẽ đặt một lệnh tiện ích của class Debug trước khi thực hiện hàm:

```
if (BENCHMARK_TEMPERATURE_CONVERSION) {
    t0 = System.currentTimeMillis();
    Debug.startMethodTracing("TemperatureChangedWatcher");
}
```

Và sau khi hàm thực hiện xong, trước khi thoát khỏi hàm ta gọi lệnh Debug.stopMethodTracing()

```
if (BENCHMARK_TEMPERATURE_CONVERSION) {
    Debug.stopMethodTracing();
    t = System.currentTimeMillis() - t0;
    Log.e("On Text Changed ==>", "t= " + t);
}
```

Kết quả hiển thị đối với ứng dụng ví dụ



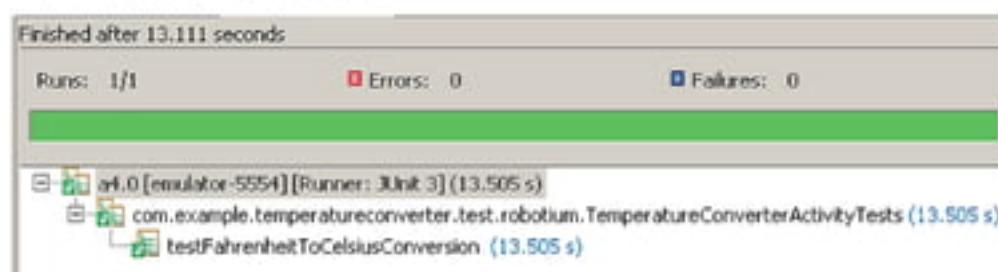
#### 4.5.Kiểm thử với Robotium framework

Robotium là một framework mã nguồn mở nhỏ gọn nhưng đầy mạnh mẽ và linh hoạt giúp cho việc kiểm thử tự động trên Android đơn giản hơn rất nhiều. Kiểm thử viên có thể kiểm thử cả hộp đen và hộp trắng trên công cụ này. Với Robotium, ta có thể dễ dàng kiểm thử chức năng, hệ thống, các kịch bản kiểm thử chấp nhận,... Ngoài ra Robotium cũng có thể kiểm thử các phần mềm mà ta không có mã nguồn (chỉ có file đã đóng gói dạng \*.apk). Robotium cũng hỗ trợ kiểm thử giao diện đối với mọi thành phần View như Activity, Dialogs, Toast, Menu, Context Menu,... Sau đây em xin trình bày áp dụng Robotium vào ứng dụng demo ở các phần trước là TemperatureConverter, testcase. Ví dụ

STT	Mục tiêu kiểm thử	Kết quả
01	Kiểm thử chuyển đổi giá trị nhập vào từ $^{\circ}\text{F}$ sang $^{\circ}\text{C}$	Pass

Một trong những điểm mạnh của Robotium là xác định hay tham chiếu đến các View thông qua tên (thuộc tính android:text) hay thử tự xuất hiện trên UI mà không cần sử dụng đến hàm activity.findViewById(int id) truyền thống của Android API, ngoài ra khi ta muốn kích hoạt một sự kiện nào đó trên một đối tượng View thì ta chỉ cần gọi hàm clickOnYYY, YYY là EditText, Button,... mà không cần gọi các hàm phức tạp trong Android testing API .

Kết quả sau khi chạy testcase:



## Chương IV Áp dụng Nunit vào kiểm thử phần mềm trên Asp.net MVC

### 1.Giới thiệu

Như trên đã trình bày, Nunit là một công cụ mã nguồn mở rất hữu dụng trong kiểm thử phần mềm tự động, ta có thể dễ dàng áp dụng công cụ này một cách linh hoạt trên bất kì một dạng phần mềm nào có hỗ trợ nền tảng Microsoft .Net framework và ASP.net MVC là một trong những công nghệ đó. Trong phần này em xin trình bày một demo áp dụng Nunit để kiểm thử các đơn vị lên một phần mềm được xây dựng với công nghệ ASP.net MVC là Sport Shop. Phần mềm này được viết bằng C# 4.0 và các công nghệ khác hỗ trợ đi kèm trong việc kiểm thử mà em sẽ trình bày ngay sau đây

#### 1.1.Mô tả tổng quan chương trình

Xây dựng một phần mềm để bán hàng Online sử dụng công nghệ Asp.net MVC và các công nghệ hỗ trợ khác, Nunit 2.6 để kiểm thử các chức năng trong phần mềm

##### 1.1.1.Phạm vi phần mềm

- Ứng dụng trong phạm vi luận văn tốt nghiệp, nhằm minh họa việc kiểm thử phần mềm, kiểm thử đơn vị, công cụ mã nguồn mở Nunit và các công nghệ khác
- Sản phẩm là một chương trình bán hàng Online trên nền web cho phép thực hiện các thao tác đơn giản nhưng vẫn đầy đủ các chức năng cơ bản.

##### 1.1.2.Phân tích yêu cầu

Phần mềm có hai loại đối tượng người dùng, ứng với mỗi loại đối tượng phần mềm sẽ có các chức năng tương ứng như sau:

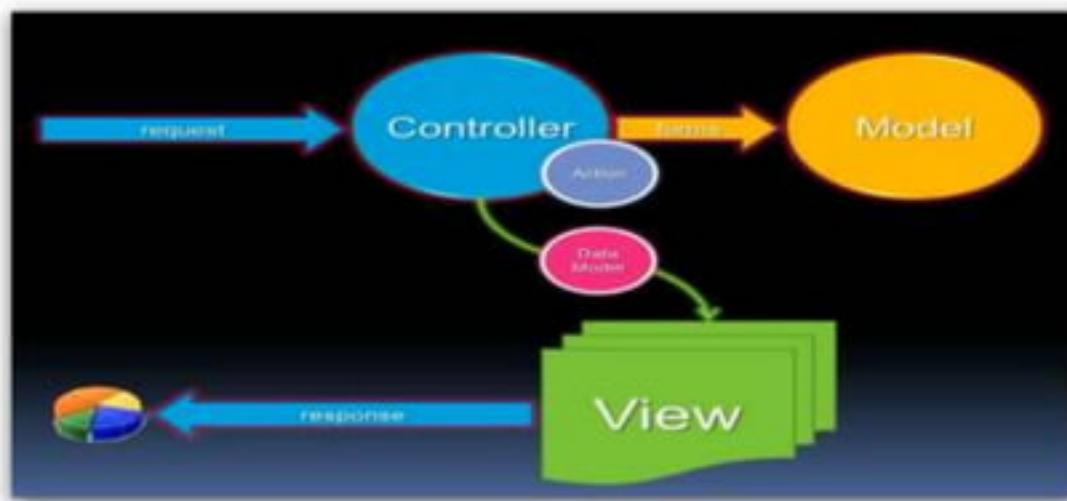
Actors	Major functional
Khách hàng (Customer)	<ul style="list-style-type: none"><li>• Xem danh sách các mặt hàng trong hệ thống theo từng loại.</li><li>• Tìm kiếm theo tên của mỗi mặt hàng</li><li>• Quản lý giỏ hàng của mình: thêm hàng, xóa hàng</li><li>• Check out: xác nhận việc mua hàng</li></ul>
Quản trị viên (Administrator)	<ul style="list-style-type: none"><li>• Đăng nhập, đăng xuất hệ thống</li><li>• Quản lý danh sách các mặt hàng: thêm, sửa, xóa.</li><li>• Tìm kiếm theo tên mặt hàng</li></ul>

### 1.2.Các công nghệ sử dụng

#### 1.2.1.Asp.net MVC

*Kiến trúc Của Asp.net MVC:*

Nền tảng MVC bao gồm các thành phần dưới đây:



**Hình 01: Mô hình Model – View – Controller**

**Models:** Các đối tượng Models là một phần của ứng dụng, các đối tượng này thiết lập logic của phần dữ liệu của ứng dụng. Thông thường, các đối tượng model lấy và lưu trạng thái của model trong CSDL. Ví dụ như, một đối tượng Product (sản phẩm) sẽ lấy dữ liệu từ CSDL, thao tác trên dữ liệu và sẽ cập nhật dữ liệu trở lại vào bảng Products ở SQL Server.

Trong các ứng dụng nhỏ, model thường là chỉ là một khái niệm nhằm phân biệt hơn là được cài đặt thực thụ, ví dụ, nếu ứng dụng chỉ đọc dữ liệu từ CSDL và gởi chúng đến view, ứng dụng không cần phải có tầng model và các lớp liên quan. Trong trường hợp này, dữ liệu được lấy như là một đối tượng model (hơn là tầng model).

**Views:** Views là các thành phần dùng để hiển thị giao diện người dùng (UI). Thông thường, view được tạo dựa vào thông tin dữ liệu model. Ví dụ như, view dùng để cập nhật bảng Products sẽ hiển thị các hộp văn bản, drop-down list, và các check box dựa trên trạng thái hiện tại của một đối tượng Product.

**Controllers:** Controller là các thành phần dùng để quản lý tương tác người dùng, làm việc với model và chọn view để hiển thị giao diện người dùng. Trong một ứng dụng MVC, view chỉ được dùng để hiển thị thông tin, controller chịu trách nhiệm quản lý và đáp trả nội dung người dùng nhập và tương tác với người dùng. Ví dụ, controller sẽ quản lý các dữ liệu người dùng gửi lên (query-string values) và gởi các giá trị đó đến model, model sẽ lấy dữ liệu từ CSDL nhờ vào các giá trị này.

### 1.2.2. Razor View engine

Đây là một View Engine được tích hợp sẵn vào từ ASP.net MVC 3.0 trở nên để hiển thị giao diện và đóng vai trò là View trong mô hình MVC. Có thể hiểu đơn giản Razor là sự kết hợp của C# và HTML trong cùng một trang văn bản HTML trong đó thì các mã C# xử lý dữ liệu động và HTML xử lý phần tĩnh (các tài nguyên tĩnh của trang web). Khi làm việc với Razor View ta thấy thật sự đây là một View engine đơn giản nhưng lại đầy mạnh mẽ và thông minh. Razor có thể dễ dàng nhận ra đâu là bắt đầu một đoạn mã xử lý C# mà ta chỉ cần thêm vào kí pháp @ trước các đối tượng trong C#. Cú pháp trong Razor cũng rất ngắn gọn và đơn giản hơn nhiều so với ASP.net View engine truyền thống (các trang \*.aspx) chính vì vậy mà Microsoft đã tạo ra hẳn một công nghệ phát triển web mới mà chỉ dựa trên sức mạnh của Razor view đó là Web Matrix và đang hứa hẹn ngày càng nhiều lập trình viên ưa thích công nghệ phát triển phần mềm dựa web mới này.

### 1.2.3. Một vài công nghệ khác

- ✓ ADO.net Entity framework Code First hỗ trợ cho việc truy xuất database đơn giản hơn
- ✓ Dependency Injection (DI): hỗ trợ cho việc kiểm thử (trong phần mềm này sử dụng một thư viện mã nguồn mở của DI là Ninject)

### 1.3. Môi trường thực thi

- ✓ Phần mềm được triển khai trên nền ASP.Net MVC framework 4, Ado.net entity framework 6.0.0 Alpha new với sự hỗ trợ của bộ công cụ Microsoft Visual Studio 2012.
- ✓ Để cài đặt phần mềm này ta cần có .Net framework 4.0 trở lên, một server web (IIS) và một server database như Microsoft SQL Server

### 1.4. Cấu trúc và hướng dẫn sử dụng phần mềm

#### 1.4.1. Cấu trúc phần mềm

Phần mềm bao gồm 3 project chính là

- ❖ SportStore.Domain: chứa các class entity, các interface truy cập trực tiếp dữ liệu, cung cấp các xử lý nghiệp vụ cho phần mềm.
- ❖ SportStore.UnitTests: chứa toàn bộ các class testcase trong phần mềm.
- ❖ SportStore.WebUI: dùng để hiển thị các trang web và điều hướng phần mềm.

#### 1.4.2. Hướng dẫn cài đặt và sử dụng

Để cài đặt phần mềm, ta có hai cách

- Cài đặt IIS làm server ảo trên máy, sau đó host phần mềm lên một thư mục ảo mà ta sẽ tạo trên server ảo này, cấu hình các thông số cho hợp lệ (chuỗi kết nối cơ sở dữ liệu, port server,...) và sau đó trên một trình duyệt bất kì nhập vào địa chỉ [http://localhost:\[port\]](http://localhost:[port]) (port ở đây là cổng kết nối tới phần mềm).
- Hay nếu máy tính có kết nối internet thì ta đơn giản chỉ cần truy cập vào địa chỉ <http://testsportshop.somee.com> là có thể sử dụng phần mềm như bình thường.

### 1.5. Giao diện phần mềm

Trang chủ



## Xem danh sách các sản phẩm của cửa hàng

[View cart](#)

Todays is 10-May-11

Type a query and hit  [Search](#)

**CATEGORIES PRODS**

- All products
- Apparel Sport
- Ball Sport
- Bicycle Sport
- Car Sport
- Glass Sport
- Gloves Sport
- Hat Sport

## List of All products

Name of Product	Description	Category	Price	Add To Cart
Nike sport hat	RIP CURL Men's Shred The Search Cap	Hat Sport	\$27.00	<a href="#">Add To Cart</a>
Adidas sport ball	Men's Soccer Finale Wembley Original Match Ball	Ball Sport	\$49.95	<a href="#">Add To Cart</a>
FILA sport shoes			\$25.67	<a href="#">Add To Cart</a>

## Giỏ hàng

[View cart](#)

Todays is 10-May-11

Type a query and hit  [Search](#)

**CATEGORIES PRODS**

- All products
- Apparel Sport
- Ball Sport
- Bicycle Sport
- Car Sport
- Glass Sport
- Gloves Sport

## Your cart

At here you will see all the products which you have added to your cart, you can check the total of price of all ones. And if you want to buy, you can click at "checkout" now.

We hope all you have comfortable about our services.

Thank you!

Quantity	Item	Price	Subtotal	Options
1	Nike sport hat	\$27.00	\$27.00	<a href="#">Remove from cart</a>
1	Adidas sport ball	\$49.95	\$49.95	<a href="#">Remove from cart</a>
1	FILA sport shoes	\$25.50	\$25.50	<a href="#">Remove from cart</a>
1	Nike sport shoes	\$25.67	\$25.67	<a href="#">Remove from cart</a>
Total			\$122.12	

[Continue shopping](#) [Checkout now](#)

## Tim kiếm

[View cart](#)

Todays is 10-May-11

Type a query and hit  [Search](#)

**CATEGORIES PRODS**

- All products
- Apparel Sport
- Ball Sport
- Bicycle Sport
- Car Sport

## Search for customer

← Back

Result for name:

- Name: Nike sport hat  
Price: \$27.00
- Name: Nike sport shoes  
Price: \$25.67
- Name: Nike sport ball  
Price: \$29.95

Dăng nhập (chỉ dành cho admin)



Sau khi đăng nhập thành công ta có thể quản trị hệ thống

Identify(ID)	Product name	Product price	Actions
1.	Nón sport...	\$27.00	Edit   Delete   Detail
2.	Adidas sp...	\$40.95	Edit   Delete   Detail
3.	Fila sport...	\$19.50	Edit   Delete   Detail

**Edit Product**

Product price(đ)

07.00

Product category

Hat Sport

Product Image mime type

image/png

Product description

R&P CURL Men's Shred The Search Cap

## 2.Thiết kế các test case cho phần mềm

### 2.1.Kiểm thử giao diện

STT	Yêu cầu Test	Yêu cầu kết quả
01	Các màu sắc hiển thị trang web	Các màu sắc của các mục, liên kết, đường dẫn phải đúng như ban đầu để ra, hiển thị trên cả web server và client browser
02	Kích thước của các đối tượng trên web	Kích thước các đối tượng không bị thay đổi so với ban đầu, hiển thị tốt cho mọi client browser
03	Vị trí tương đối của các phần tử trang web	Các vị trí các phần tử không bị lệch đi, ví dụ như trong màn hình đăng nhập (dành cho Admin) thì khoảng cách hai textfield và nút submit là không được thay đổi, nút submit luôn nằm dưới hai textfield này.
04	Giao diện đơn giản, thân thiện, dễ sử dụng	Các phần tử phải được bố trí hợp lý trên mọi trang web trong phần mềm cho phép người dùng thao tác thuận tiện nhất. Ví dụ khi user nhập xong dữ liệu cho một textfield thì cho phép dùng phím tab để chuyển đến thành phần khác mà không cần thao tác đến chuột, hay như nhập xong ô tìm kiếm thì chỉ cần nhấn Enter hệ thống sẽ làm việc ngay mà không cần dùng chuột click vào nút "Search".

### 2.2.Kiểm thử chức năng

STT	Yêu cầu Test	Yêu cầu kết quả
01	Kiểm thử chức năng duyệt danh sách các sản phẩm	Các sản phẩm phải được sắp xếp theo danh mục từng loại
02	Kiểm thử chức năng quản lý giỏ hàng	Thực hiện thêm, xóa các mặt hàng trong giỏ, tổng số tiền của các mặt hàng phải đúng với công thức $\sum(\text{số lượng một sản phẩm} * \text{giá/một sản phẩm})$
03	Kiểm thử chức	Sau khi đã có một số mặt hàng trong giỏ, khách hàng có thể đặt mua

	năng checkout	hang, khi thực hiện thành công mua hàng, toàn bộ mặt hàng trong giỏ sẽ bị xóa bỏ.
04	Kiểm thử chức năng tìm kiếm	Kết quả tìm kiếm phải chứa toàn bộ từ khóa mà user đã nhập và hiển thị theo dạng danh sách
05	Kiểm thử chức năng LogOn	Admin sau khi nhập đúng Username và Password thi sẽ truy cập được vào chức năng quản trị hệ thống của phần mềm
06	Kiểm thử chức năng quản lý sản phẩm	Sau khi đã LogOn thành công User sẽ có quyền Admin quản trị hệ thống với các thao tác nghiệp vụ như thêm, cập nhật, xóa sản phẩm, tìm kiếm,...
07	Kiểm thử chức năng LogOut	Cho phép thoát khỏi hệ thống sau khi đã thực hiện xong các công việc.

### 2.3.Kiểm thử các module: xây dựng testcase cho các module sau

- +Module phân trang
- +Module gửi dữ liệu từ controller đến View trong module phân trang
- +Module phân loại các sản phẩm theo loại category
- +Module tạo ra danh sách các categories của sản phẩm
- +Module đếm số sản phẩm trong một category xác định
- +Module shopping cart
- +Module checkout
- +Module Admin: hiển thị danh sách các sản phẩm
- +Module Admin: Update product
- +Module Admin: Edit product
- +Module Admin: delete product
- +Module Admin: thêm và xóa, hiển thị danh mục loại sản phẩm
- +Module Admin: LogOn

## 3.Thực thi các testcase

### 3.1.Kiểm thử giao diện

STT	Yêu cầu Test	Yêu cầu kết quả	Kết quả
01	Các màu sắc hiển thị trang web	Các màu sắc của các mục, liên kết, đường dẫn phải đúng như ban đầu để ra, hiển thị trên cả web seerver và client browser	True
02	Kích thước của các đối tượng trên web	Kích thước các đối tượng không bị thay đổi so với ban đầu, hiển thị tốt cho mọi client browser	True
03	Vị trí tương đối của các phần tử trang web	Các vị trí các phần tử không bị lệch đi, ví dụ như trong màn hình đăng nhập (dành cho Admin) thì khoảng cách hai textfield và nút submit là không được thay đổi, nút submit luôn nằm dưới hai textfield này.	True
04	Giao diện đơn giản, thân thiện, đơn giản, dễ sử dụng	Các phần tử phải được bố trí hợp lý trên mọi trang web trong phần mềm cho phép người dùng thao tác thuận tiện nhất. Ví dụ khi user nhập xong dữ liệu cho một textfield thi cho phép dùng phím tab để chuyển đến thành phần khác mà không cần thao tác đến chuột, hay như nhập xong ô tìm kiếm thi	True

		chi cần nhấn Enter hệ thống sẽ làm việc ngay mà không cần dùng chuột click vào nút “Search”.	
--	--	----------------------------------------------------------------------------------------------	--

### 3.2.Kiểm thử chức năng

STT	Yêu cầu Test	Yêu cầu kết quả	Kết quả
01	Kiểm thử chức năng duyệt danh sách các sản phẩm	Các sản phẩm phải được sắp xếp theo danh mục từng loại	Pass
02	Kiểm thử chức năng quản lý giỏ hàng	Thực hiện thêm, xóa các mặt hàng trong giỏ, tổng số tiền của các mặt hàng phải đúng với công thức $\Sigma(\text{số lượng một sản phẩm} * \text{giá/ một sản phẩm đó})$	Pass
03	Kiểm thử chức năng checkout	Sau khi đã có một số mặt hàng trong giỏ, khách hàng có thể đặt mua hàng, khi thực hiện thành công mua hàng, toàn bộ mặt hàng trong giỏ sẽ bị xóa bỏ.	Pass
04	Kiểm thử chức năng tìm kiếm	Kết quả tìm kiếm phải chứa toàn bộ từ khóa mà user đã nhập và hiển thị theo dạng danh sách	Pass
05	Kiểm thử chức năng LogOn	Admin sau khi nhập đúng Username và Password thì sẽ truy cập được vào chức năng quản trị hệ thống của phần mềm	Pass
06	Kiểm thử chức năng quản lý sản phẩm	Sau khi đã LogOn thành công User sẽ có quyền Admin quản trị hệ thống với các thao tác nghiệp vụ như thêm, cập nhật, xóa sản phẩm, tìm kiếm,...	Pass
07	Kiểm thử chức năng LogOut	Cho phép thoát khỏi hệ thống sau khi đã thực hiện xong các công việc.	Pass

### 3.3.Kiểm thử các Module

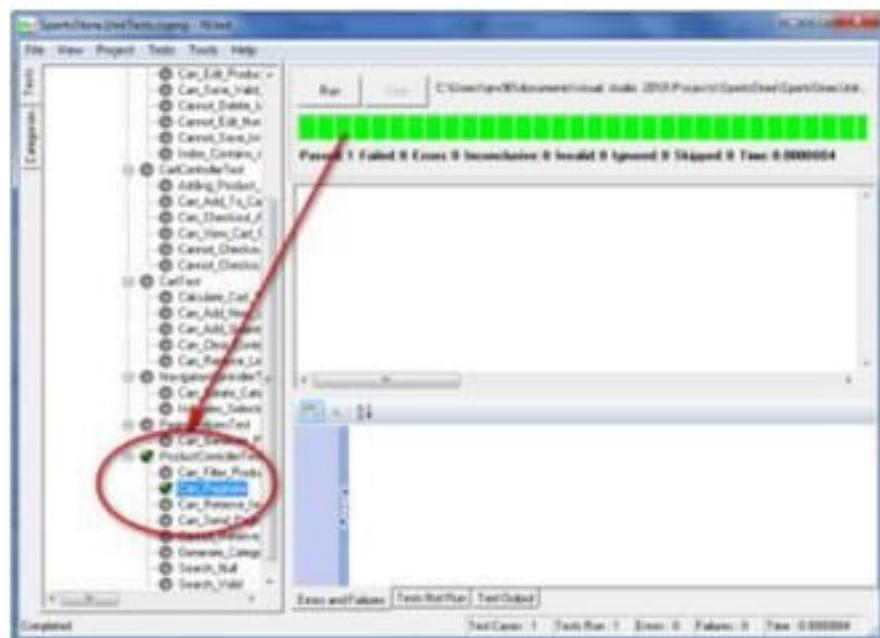
+Module phân trang: chúng ta có thể kiểm thử tính năng phân trang bằng cách tạo một đối tượng giả lập truy xuất dữ liệu, gọi là mock repository, và gắn nó (injecting) vào hàm khởi tạo của class ProductController và sau đó gọi phương thức List() để yêu cầu một trang cụ thể. Sau đó chúng ta có thể so sánh các Product mà ta lấy được với dữ liệu kiểm thử trong cài đặt mock.

```

21  [Test]
22  public void Can_Paginate()
23  {
24
25      // Arrange
26      // - create the mock repository
27      Mock<IProductRepository> mock = new Mock<IProductRepository>();
28      mock.Setup(m => m.Products).Returns(new Product[]
29      {
30          new Product {ProductID = 1, Name = "P1"},
31          new Product {ProductID = 2, Name = "P2"},
32          new Product {ProductID = 3, Name = "P3"},
33          new Product {ProductID = 4, Name = "P4"},
34          new Product {ProductID = 5, Name = "P5"}
35      }.AsQueryable());
36
37      // create a controller and make the page size 3 items
38      ProductController controller = new ProductController(mock.Object);
39      controller.PageSize = 3;
40
41      // Action
42      ProductListViewModel result = (ProductListViewModel)controller.List(null, 2).Model;
43
44      // Assert
45      Product[] prodArray = result.Products.ToArray();
46      NUnit.Framework.Assert.IsTrue(prodArray.Length == 3);
47      NUnit.Framework.Assert.AreEqual(prodArray[0].Name, "P4");
48      NUnit.Framework.Assert.AreEqual(prodArray[1].Name, "P5");
49  }

```

## Kết quả khi chạy với Nunit



- + Module gửi dữ liệu từ controller đến View trong module phân trang: chúng ta cần đảm bảo đúng dữ liệu phân trang được gửi từ controller đến View

```
[Test]
public void Can_Send_Pagination_View_Model()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();

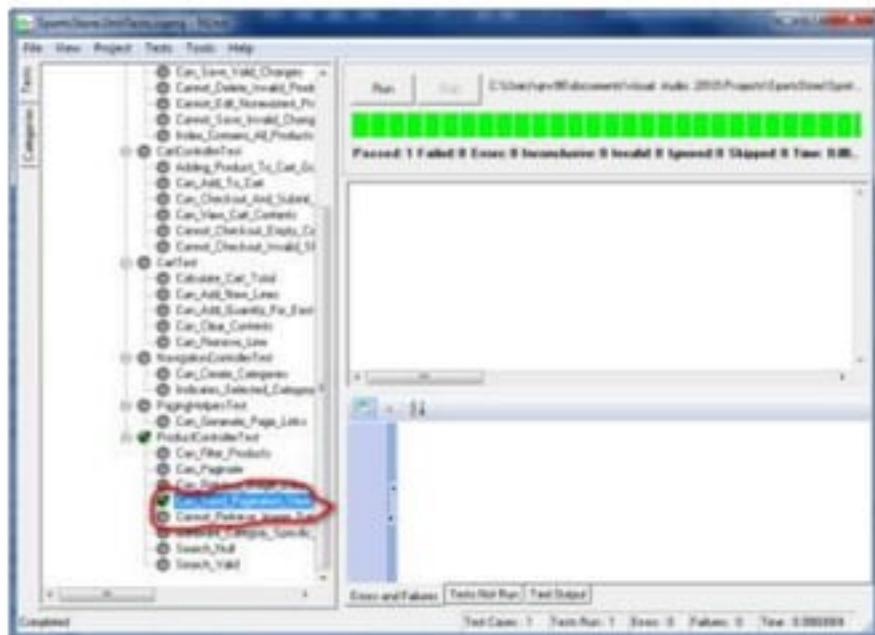
    mock.Setup(m => m.Products).Returns(new Product[] {
        new Product {ProductID = 1, Name = "P1"},
        new Product {ProductID = 2, Name = "P2"},
        new Product {ProductID = 3, Name = "P3"},
        new Product {ProductID = 4, Name = "P4"},
        new Product {ProductID = 5, Name = "P5"}
    }).AsQueryable();

    ProductController control = new ProductController(mock.Object);
    control.PageSize = 3;

    ProductsListViewModel rs = (ProductsListViewModel)control.List(null, 2).Model;
    PagingInfo info = rs.PagingInfo;

    NUnit.Framework.Assert.AreEqual(info.CurrentPage, 2);
    NUnit.Framework.Assert.AreEqual(info.ItemsPerPage, 3);
    NUnit.Framework.Assert.AreEqual(info.TotalItems, 5);
    NUnit.Framework.Assert.AreEqual(info.TotalPages, 2);
}
```

Kết quả sau khi thực thi



+Module phân loại các sản phẩm theo loại category: chúng ta cần kiểm thử tính năng phân loại sản phẩm theo từng loại để đảm bảo ta chỉ nhận được các sản phẩm của một mặt hàng nhất định nào đó

```
[Test]
public void Can_Filter_Products()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();

    mock.Setup(m => m.Products).Returns(new Product[]
    {
        new Product {ProductID = 1, Name = "P1", Category = "Cat1"},
        new Product {ProductID = 2, Name = "P2", Category = "Cat2"},
        new Product {ProductID = 3, Name = "P3", Category = "Cat1"},
        new Product {ProductID = 4, Name = "P4", Category = "Cat2"},
        new Product {ProductID = 5, Name = "P5", Category = "Cat3"}
    }).AsQueryable();

    ProductController control = new ProductController(mock.Object);
    control.PageSize = 3;

    Product[] rs = ((ProductsListViewModel)control.
        List("Cat2", 1).Model).
        Products.ToArray();

    NUnit.Framework.Assert.AreEqual(rs.Length, 2);
    NUnit.Framework.Assert.IsTrue(rs[0].Name == "P2" && rs[0].Category == "Cat2");
    NUnit.Framework.Assert.IsTrue(rs[1].Name == "P4" && rs[0].Category == "Cat2");
}
```

+ Module đếm số sản phẩm trong một category xác định: ta sẽ tạo một đối tượng mock repository chứa các dữ liệu cho trước trong một tập các danh mục hàng và sau đó gọi phương thức List() để lấy về các sản phẩm thuộc cùng một mặt hàng và nếu tên mặt hàng này là null thì sẽ lấy về toàn bộ các sản phẩm

```

[TestMethod]
public void Generate_Category_Specific_Product_Count()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    mock.Setup(m => m.Products).Returns(new Product[]
    {
        new Product {ProductID = 1, Name = "P1", Category = "Cat1"},
        new Product {ProductID = 2, Name = "P2", Category = "Cat2"},
        new Product {ProductID = 3, Name = "P3", Category = "Cat1"},
        new Product {ProductID = 4, Name = "P4", Category = "Cat2"},
        new Product {ProductID = 5, Name = "P5", Category = "Cat3"}
    })
    .AsQueryable();

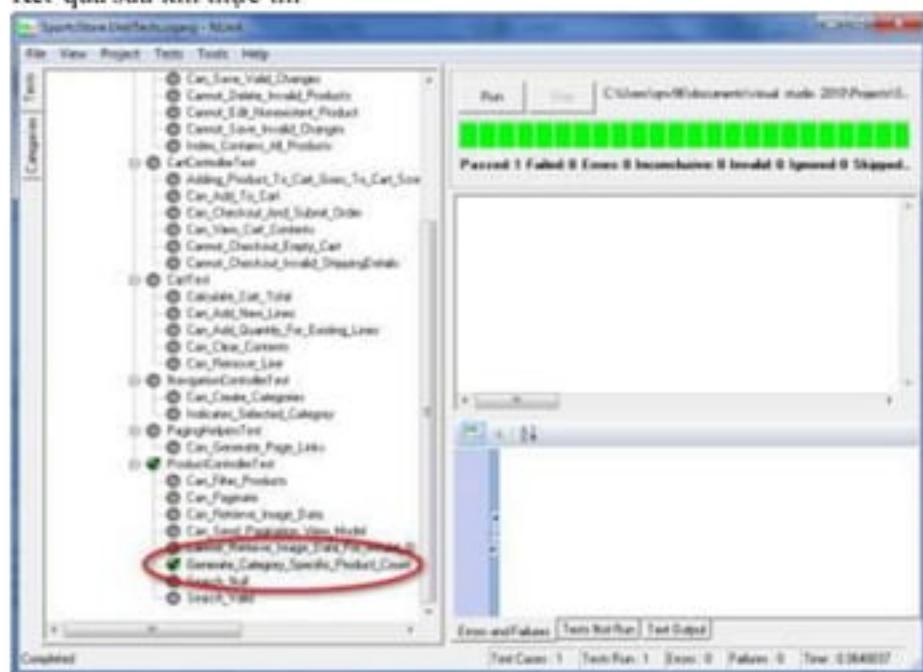
    ProductController target = new ProductController(mock.Object);
    target.PageSize = 3;

    // Action - test the product counts for different categories
    int res1 = ((ProductsListViewModel)target.List("Cat1").Model).PagingInfo.TotalItems;
    int res2 = ((ProductsListViewModel)target.List("Cat2").Model).PagingInfo.TotalItems;
    int res3 = ((ProductsListViewModel)target.List("Cat3").Model).PagingInfo.TotalItems;
    int resAll = ((ProductsListViewModel)target.List(null).Model).PagingInfo.TotalItems;

    // Assert
    NUnit.Framework.Assert.AreEqual(res1, 2);
    NUnit.Framework.Assert.AreEqual(res2, 2);
    NUnit.Framework.Assert.AreEqual(res3, 1);
    NUnit.Framework.Assert.AreEqual(resAll, 5);
}

```

Kết quả sau khi thực thi



+ Module shopping cart: thêm một sản phẩm vào giỏ hàng, nếu đây là lần đầu tiên sản phẩm được thêm vào giỏ hàng thì ta thêm mới một đối tượng CartLine

```

[Test]
public void Can_Add_New_Lines()
{
    Product p1 = new Product { ProductID = 1, Name = "P1" };
    Product p2 = new Product { ProductID = 2, Name = "P2" };

    Cart cart = new Cart();
    cart.AddItem(p1, 1);
    cart.AddItem(p2, 1);

    CartLine[] rs = cart.Lines.ToArray();

    Assert.AreEqual(rs.Length, 2);
    Assert.AreEqual(rs[0].Product, p1);
    Assert.AreEqual(rs[1].Product, p2);
}

```

Tuy nhiên nếu sản phẩm đã có trong giỏ thì ta chỉ việc tăng số lượng của sản phẩm đó lên một đơn vị

```

[Test]
public void Can_Add_Quantity_For_Existing_Lines()
{
    Product p1 = new Product { ProductID = 1, Name = "P1" };
    Product p2 = new Product { ProductID = 2, Name = "P2" };

    Cart cart = new Cart();
    cart.AddItem(p1, 1);
    cart.AddItem(p1, 10);
    cart.AddItem(p2, 1);

    CartLine[] rs = cart.Lines.OrderBy(c => c.Product.ProductID).ToArray();

    Assert.AreEqual(rs.Length, 2);
    Assert.AreEqual(rs[0].Quantity, 11);
    Assert.AreEqual(rs[1].Quantity, 1);
}

```

Tiếp theo ta sẽ kiểm thử chức năng xóa bỏ một sản phẩm ra khỏi giỏ hàng

```

[Test]
public void Can_Remove_Line()
{
    Product p1 = new Product { ProductID = 1, Name = "P1" };
    Product p2 = new Product { ProductID = 2, Name = "P2" };
    Product p3 = new Product { ProductID = 3, Name = "P3" };

    Cart cart = new Cart();
    cart.AddItem(p1, 1);
    cart.AddItem(p2, 1);
    cart.AddItem(p2, 3);
    cart.AddItem(p3, 5);

    cart.RemoveLine(p2);

    int rs = cart.Lines.Where(c => c.Product == p2).Count();

    Assert.AreEqual(rs, 0);
    Assert.AreEqual(cart.Lines.Count(), 2);
}

```

Sau đó ta cũng sẽ xem xét việc tính toán tổng giá trị các sản phẩm trong giỏ có đúng không

```

[Test]
public void Calculate_Cart_Total()
{
    Product p1 = new Product { ProductID = 1, Name = "P1" , Price = 100m};
    Product p2 = new Product { ProductID = 2, Name = "P2" , Price = 50m};

    Cart cart = new Cart();
    cart.AddItem(p1, 1);
    cart.AddItem(p1, 3);
    cart.AddItem(p2, 1);

    decimal rs = cart.ComputeTotalValue();

    Assert.AreEqual(rs, 450m);
}

```

Và cuối cùng là giờ hàng sẽ bị xóa bỏ khi user thoát khỏi session hay thực hiện checkout

```

[Test]
public void Can_Clear_Contents()
{
    Product p1 = new Product { ProductID = 1, Name = "P1", Price = 100m };
    Product p2 = new Product { ProductID = 2, Name = "P2", Price = 50m };

    Cart cart = new Cart();
    cart.AddItem(p1, 1);
    cart.AddItem(p2, 1);

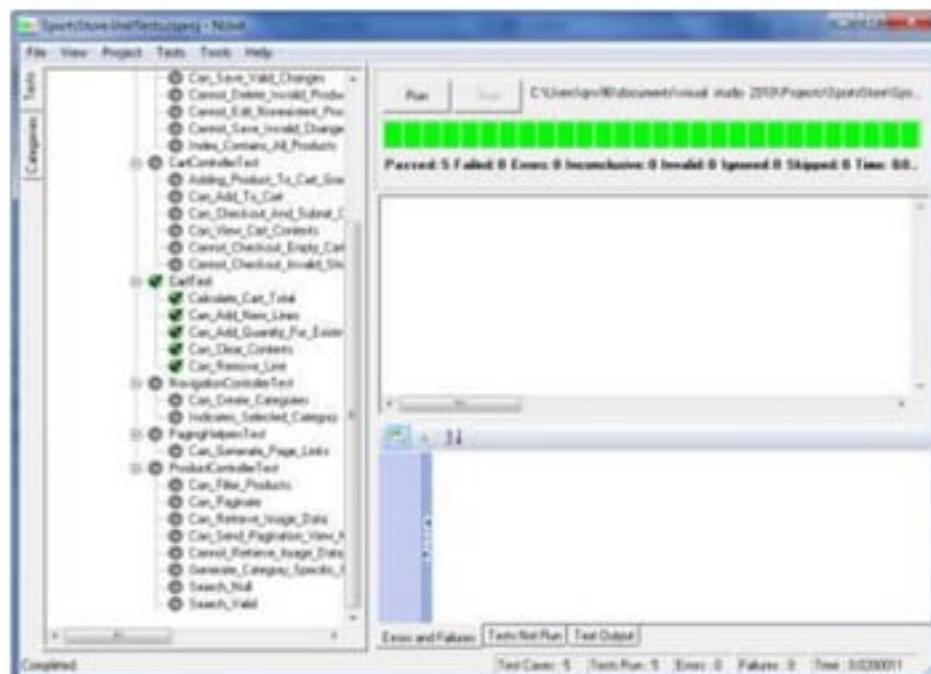
    cart.Clear();

    int rs = cart.Lines.Count();

    Assert.AreEqual(rs, 0);
}

```

Kết quả của toàn bộ các thao tác trên khi tiến hành trên Nunit



+Module checkout: khi giờ hàng chưa có sản phẩm nào mà khách hàng thực hiện checkout thì sẽ đưa đến một thông báo lỗi nhắc nhở khách hàng, ta kiểm tra bằng cách xác thực rằng phương thức ProcessOrder() của đối tượng mock IorderProcessor sẽ không bao giờ được gọi

```
[Test]
public void Cannot_Checkout_Empty_Cart()
{
    Mock<IOrderProcessor> mock = new Mock<IOrderProcessor>();
    Cart cart = new Cart();
    ShippingDetails shippingDetails = new ShippingDetails();
    CartController control = new CartController(null, mock.Object);

    ViewResult rs = control.Checkout(cart, shippingDetails);
    mock.Verify(m => m.ProcessOrder(It.IsAny<Cart>(), It.IsAny<ShippingDetails>()), Times.Never());

    Assert.AreEqual("", rs.ViewName);
    Assert.AreEqual(false, rs.ViewData.ModelState.IsValid);
}
}
```

Tiếp theo ta cũng sẽ kiểm tra xem nếu như khách hàng nhập thiếu thông tin cần thiết trong phần checkout thì ta cũng sẽ xuất ra một thông báo lỗi trong ModelState

```
[Test]
public void Cannot_Checkout_Invalid_ShippingDetails()
{
    Mock<IOrderProcessor> mock = new Mock<IOrderProcessor>();
    Cart cart = new Cart();
    cart.AddItem(new Product(), 1);

    CartController control = new CartController(null, mock.Object);
    control.ModelState.AddModelError("error", "error");

    ViewResult rs = control.Checkout(cart, new ShippingDetails());
    mock.Verify(m => m.ProcessOrder(It.IsAny<Cart>(), It.IsAny<ShippingDetails>()), Times.Never());

    Assert.AreEqual("", rs.ViewName);
    Assert.AreEqual(false, rs.ViewData.ModelState.IsValid);
}
}
```

Và cuối cùng nếu mọi thông tin đều hợp lệ thì ta sẽ kiểm tra xem việc checkout có thực hiện thành công không

```
[Test]
public void Can_Checkout_And_Submit_Order()
{
    Mock<IOrderProcessor> mock = new Mock<IOrderProcessor>();
    Cart cart = new Cart();
    cart.AddItem(new Product(), 1);

    CartController control = new CartController(null, mock.Object);

    ViewResult rs = control.Checkout(cart, new ShippingDetails());
    mock.Verify(m => m.ProcessOrder(It.IsAny<Cart>(), It.IsAny<ShippingDetails>()), Times.Once());

    Assert.AreEqual("Completed", rs.ViewName);
    Assert.AreEqual(true, rs.ViewData.ModelState.IsValid);
}
}
```

+Module Admin: hiển thị danh sách các sản phẩm: kiểm tra xem danh sách sản phẩm trả về là trong cơ sở dữ liệu, ta kiểm tra bằng cách tạo một đối tượng mock repository và so sánh dữ liệu thử với dữ liệu trả về của phương thức

```

[Test]
public void Index_Contains_All_Products()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();

    mock.Setup(p => p.Products).Returns(new Product[]
    {
        new Product{ ProductID =1, Name="P1"},
        new Product{ ProductID =2, Name="P2"},
        new Product{ ProductID =3, Name="P3"},

    }).AsQueryable();

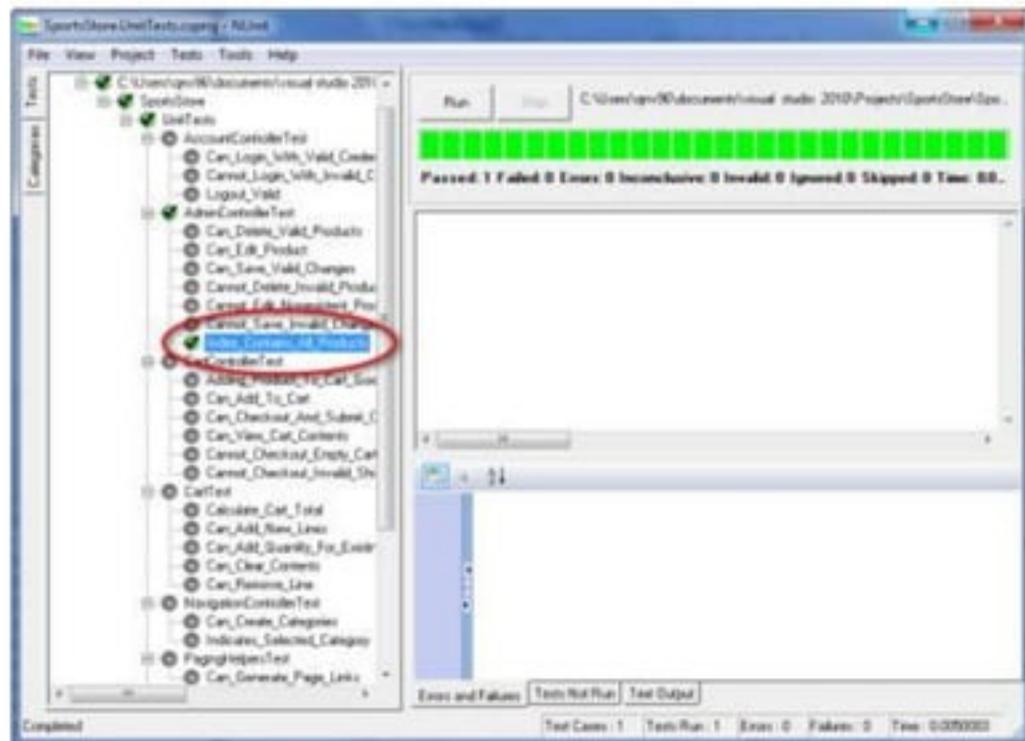
    AdminController target = new AdminController(mock.Object);

    Product[] rs = ((IEnumerable<Product>)target.Index().ViewData.Model).ToArray();

    Assert.AreEqual(rs.Length, 3);
    Assert.AreEqual("P1", rs[0].Name);
    Assert.AreEqual("P2", rs[1].Name);
    Assert.AreEqual("P3", rs[2].Name);
}

```

Kết quả sau khi thực thi



+Module Admin: Update product: ta sẽ kiểm thử xem nếu ta nhập vào ID của một product thì ta sẽ lấy được đúng sản phẩm ấy, ngược lại nếu ID đó không tồn tại trong kho dữ liệu (repository) thì ta sẽ không lấy được bất kỳ sản phẩm nào

```

[Test]
public void Can_Edit_Product()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    mock.Setup(p => p.Products).Returns(new Product[]
    {
        new Product{ ProductID =1, Name="P1"},
        new Product{ ProductID =2, Name="P2"},
        new Product{ ProductID =3, Name="P3"},

    }.AsQueryable());
    AdminController target = new AdminController(mock.Object);

    Product p1 = target.Edit(1).ViewData.Model as Product;
    Product p2 = target.Edit(2).ViewData.Model as Product;
    Product p3 = target.Edit(3).ViewData.Model as Product;

    Assert.AreEqual(1, p1.ProductID);
    Assert.AreEqual(2, p2.ProductID);
    Assert.AreEqual(3, p3.ProductID);
}

[Test]
public void Cannot_Edit_Nonexistent_Product()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    mock.Setup(p => p.Products).Returns(new Product[]
    {
        new Product{ ProductID =1, Name="P1"},
        new Product{ ProductID =2, Name="P2"},
        new Product{ ProductID =3, Name="P3"},

    }.AsQueryable());
    AdminController target = new AdminController(mock.Object);

    Product rs = (Product) target.Edit(4).ViewData.Model;
    Assert.IsNotNull(rs);
}

```

Sau khi đã có được sản phẩm để update thông tin , ta cũng cần kiểm tra xem khi ta nhập sai thông tin thì phần mềm sẽ không cho phép lưu vào cơ sở dữ liệu, ngược lại nếu dữ liệu nhập đúng ta sẽ lưu vào cơ sở dữ liệu các giá trị đã thay đổi

```

[TestMethod]
[Obsolete]
public void Cannot_Save_Invalid_Changes()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    AdminController admin = new AdminController(mock.Object);
    Product product = new Product { Name = "Test" , ProductID=1};

    admin.ModelState.AddModelError("Error", "Error");

    ActionResult rs = admin.Edit_Post(1,null);

    mock.Verify(p => p.SaveProduct(It.IsAny<Product>()), Times.Never());
    Assert.IsInstanceOfType(typeof(ViewResult), rs,"Co loi xay ra");
}

[TestMethod]
public void Can_Delete_Valid_Products()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    Product prod = new Product { ProductID = 2, Name = "Test" };

    mock.Setup(p => p.Products).Returns(new Product[]
    {
        new Product{ ProductID =1, Name="P1"},
        prod,
        new Product{ ProductID =3, Name="P3"},

    }.AsQueryable());

    AdminController admin = new AdminController(mock.Object);

    admin.Delete(prod.ProductID);
    mock.Verify(p => p.DeleteProduct(prod));
}

```

+Module Admin: delete product: đầu tiên ta sẽ kiểm tra xem khi ta đưa đúng vào một ID của một sản phẩm thì phương thức sẽ gọi hàm DeleteProduct của repository và nhận về đúng sản phẩm đã bị xóa

```

[TestMethod]
public void Can_Delete_Valid_Products()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    Product prod = new Product { ProductID = 2, Name = "Test" };

    mock.Setup(p => p.Products).Returns(new Product[]
    {
        new Product{ ProductID =1, Name="P1"},
        prod,
        new Product{ ProductID =3, Name="P3"},

    }.AsQueryable());

    AdminController admin = new AdminController(mock.Object);

    admin.Delete(prod.ProductID);
    mock.Verify(p => p.DeleteProduct(prod));
}

```

Và tiếp theo ta cần xác nhận rằng nếu ta đưa vào một ID mà không tương ứng với bất kì sản phẩm nào thì phương thức DeleteProduct sẽ không được gọi

```

[Test]
public void Cannot_Delete_Invalid_Products()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();

    mock.Setup(p => p.Products).Returns(new Product[]
    {
        new Product{ ProductID =1, Name="P1"},
        new Product{ ProductID =2, Name="P2"},
        new Product{ ProductID =3, Name="P3"},

    }).AsQueryable();

    AdminController admin = new AdminController(mock.Object);

    admin.Delete(100);
    mock.Verify(p => p.DeleteProduct(It.IsAny<Product>()), Times.Never());
}

```

+Module Admin hiển thị, thêm và xóa danh mục các loại mặt hàng

Testcase này sẽ kiểm thử chức năng thêm và xóa bỏ các danh mục hàng trong hệ thống. Nếu thêm một danh mục hàng thành công thì hệ thống sẽ chuyển đến trang hiển thị danh sách các loại mặt hàng

```

[Test]
[Obsolete]
public void CreateCategory()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    AdminController admin = new AdminController(mock.Object);

    Category category = new Category { ID = 4, Name = "C4" };
    |
    ActionResult rs = admin.Addcategory_Test(category);

    mock.Verify(c => c.Addcategory(category));
    Assert.IsInstanceOfType(typeof(RedirectToRouteResult), rs, "Co loi xay ra");
}

```

Tương tự với chức năng xóa danh mục mặt hàng

```

[Test]
[Obsolete]
public void DeleteCategory()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    Category cate_delete = new Category { ID = 2, Name = "C2" };

    mock.Setup(c => c.Categories).Returns(new Category[]
    {
        new Category{ID= 1,Name = "C1"}, 
        cate_delete,
        new Category{ID= 3,Name = "C3"},

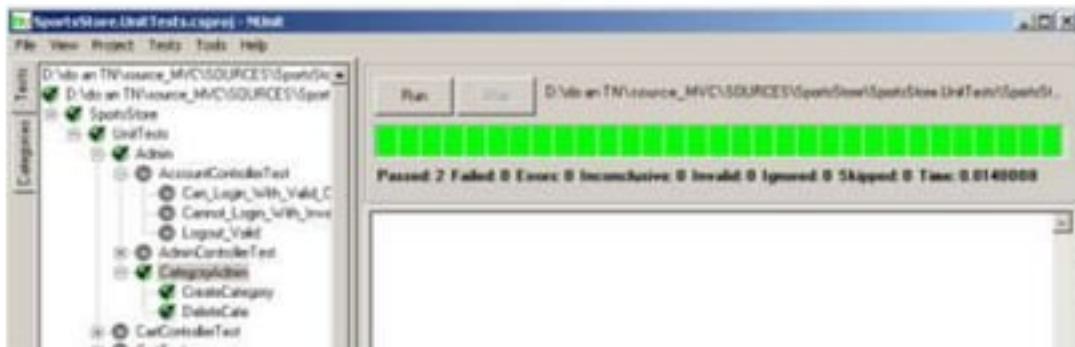
    }).AsQueryable();

    AdminController admin = new AdminController(mock.Object);
    ActionResult rs = admin.Deletecategory(cate_delete.ID);

    mock.Verify(c => c.Deletecategory(cate_delete));
    Assert.IsInstanceOfType(typeof(RedirectToRouteResult), rs, "Co loi xay ra");
}

```

Kết quả sau khi chạy hai testcase này



+Module Admin: LogOn: trong phần này ta sẽ kiểm tra xem nếu User nhập vào đúng Username và Password thì sẽ Login vào hệ thống với quyền Admin và ngược lại. Ở đây ta tạo một đối tượng giả lập mock (thực chất là một interface) là IauthProvider và kiểm tra kiểu và kết quả trả về

```
[Test]
[Obsolete]
public void Can_Login_With_Valid_Credentials()
{
    Mock<IAuthProvider> mock = new Mock<IAuthProvider>();
    mock.Setup(a => a.Authenticate("admin", "secret")).Returns(true);

    LogOnViewModel model = new LogOnViewModel
    {
        Username="admin",
        Password="secret"
    };

    AccountController account = new AccountController(mock.Object);
    ActionResult rs = account.LogOn(model, url);

    Assert.IsInstanceOfType(typeof(RedirectResult), rs, "co error");
    Assert.AreEqual(url, ((RedirectResult)rs).Url);
}
```

-Kiểm thử một vài module khác

+Module tìm kiếm: khi user nhập vào một từ trong tên sản phẩm thì phần mềm sẽ liệt kê ra toàn bộ các sản phẩm có tên chứa cụm từ đã nhập vào

```
[Test]
public void Search_Valid()
{
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    mock.Setup(p => p.Products).Returns(new Product[]
    {
        new Product{ProductID=1, Name="Tim Ball"},
        new Product{ProductID=2, Name="Tennis Hat"},
        new Product{ProductID=3, Name="Goal Hat"},
        new Product{ProductID=4, Name="Sport shoes"}
    });

    ProductController control = new ProductController(mock.Object);
    ActionResult tk = control.Search("Hat");
    ViewResult rs = (ViewResult)tk;

    IEnumerable<Product> data = rs.ViewData.Model as IEnumerable<Product>;
    NUnit.Framework.Assert.AreEqual(data.Count(), 2);
}
```

+Module Logout: cho phép Admin sau khi đã thực hiện xong công việc thi Logout ra khỏi hệ thống

```

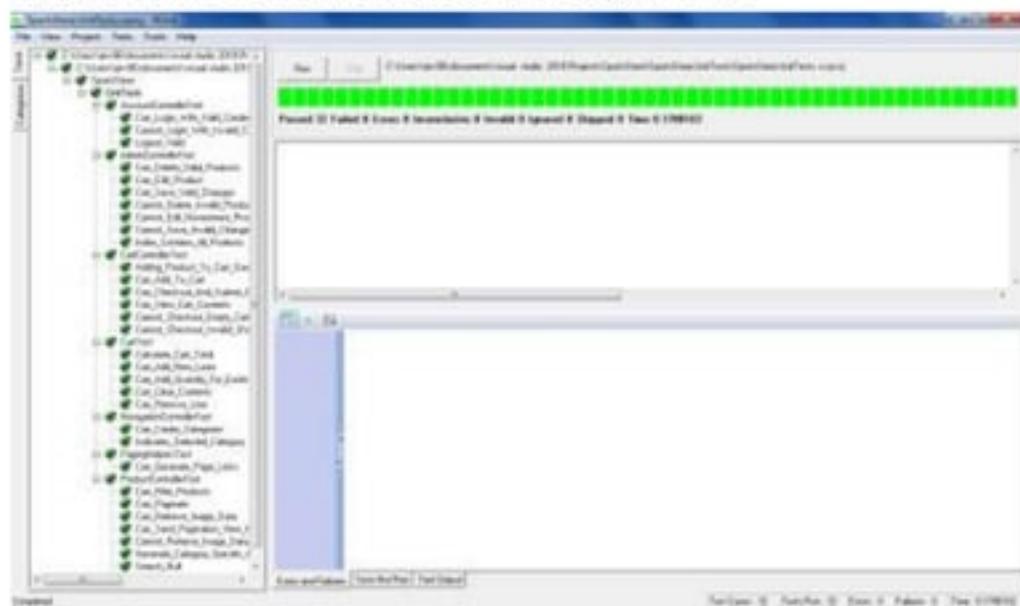
[Test]
[Obsolete]
public void Logout_Valid()
{
    Mock<IAAuthProvider> mock = new Mock<IAAuthProvider>();
    AccountController control = new AccountController(mock.Object);

    ActionResult rs = control.LogOut(true) as ViewResult;

    Assert.IsNotNull(rs);
    Assert.IsInstanceOfType(typeof(ViewResult), rs, "error");
}

```

Kết quả sau khi thực thi toàn bộ các testcase trong phần mềm



## Kết luận

Kiểm thử phần mềm là một hoạt động quan trọng nhằm đảm bảo chất lượng phần mềm.

Việc nghiên cứu lựa chọn các kỹ thuật và chiến lược kiểm thử phần mềm phù hợp giúp cho việc kiểm thử có hiệu quả, giảm chi phí và thời gian. Việc xây dựng tài liệu kiểm thử phần mềm hợp lý sẽ giúp cho việc tổ chức, quản lý và thực hiện kiểm thử có hiệu quả.

## Những vấn đề đã đạt được

Trong thời gian làm thực tập tốt nghiệp với sự định hướng và giúp đỡ tận tình của thầy ThS. Thạc Binh Cường, em đã đạt được những kết quả sau:

- Nắm được tổng quan về kiểm thử phần mềm: các phương pháp, kỹ thuật kiểm thử phần mềm và các vấn đề liên quan.....
- Tìm hiểu và nắm được phương pháp thiết kế test case trong kiểm thử phần mềm và áp dụng phương pháp đó với bài toán cụ thể.
- Nghiên cứu những chức năng và cách thức hoạt động kiểm thử công cụ mã nguồn mở NUnit và sử dụng nó để kiểm thử cho những phần mềm hoàn chỉnh viết bằng .NET.

Hướng phát triển của đề tài trong tương lai

Khi nghiên cứu về kiểm thử phần mềm nói chung và công cụ Nunit, JUnit nói riêng, em đã hiểu được kiểm thử là rất quan trọng trong quy trình sản xuất phần mềm, đảm bảo chất lượng phần mềm. Sự áp dụng với kiến thức tìm hiểu được mới chỉ dừng lại ở một bài toán nhỏ. Hướng phát triển của thực tập tốt nghiệp là:

- Thực hiện kiểm thử trên mô hình bài toán phần mềm rộng hơn, phức tạp hơn.
- Tìm hiểu và nghiên cứu thêm về các công cụ kiểm thử tự động, kiểm thử website, kiểm thử tái, kiểm thử cơ sở dữ liệu....

### Tài liệu tham khảo

#### {Sách tham khảo}

- [1]. Bài giảng điện tử môn học Kiểm thử và đảm bảo chất lượng phần mềm, Th.s.Thạc Bình Cường, Bộ môn CNPM, Viện CNTT&TT, Đại học Bách Khoa Hà Nội.
- [2]. The Art Of Software Testing – Second Edition - Glenford J. Myers
- [3]. McGraw Hill - Software Engineering - A Practitioner's Approach - Pressman (5th Ed)(2001)
- [4]. Sybex - Effective Software Test Automation
- [5]. Happy About® Global Software Test Automation - Hung Q. Nguyen, Michael Hackett, Brent K. Whitlock

#### {Các trang web tham khảo}

- [6]. [http://en.wikipedia.org/wiki/Unit\\_testing](http://en.wikipedia.org/wiki/Unit_testing)
- [7]. [http://en.wikipedia.org/wiki/Software\\_testing](http://en.wikipedia.org/wiki/Software_testing)
- [8]. [http://msdn.microsoft.com/en-us/library/ms243147\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/ms243147(v=vs.80).aspx)
- [9]. <http://students.depaul.edu/~slouie/QTUsersGuide.pdf>
- [10]. Professional ASP.net MVC 4 Wrox publishing house 2012
- [11]. Android GUI testing
- [12]. Internet,...