**Tên sinh viên:**

1. **Nguyễn Văn Lâm - 20152135**
2. **Nguyễn Thị Ngọc - 20152693**
3. **Nguyễn Đình Phúc - 20152864**
4. **Nguyễn Thùy Lan - 20152107**
5. **UNIT TEST**
6. Định nghĩa Unit Test:

* Một Unit test là một phần của source code, thực thi một phần code chính xác và so sánh kết quả thực tế với kết quả mong đợi.

1. Tầm quan trọng của Unit test:

* Lợi ích của Unit test:

+ Đảm bảo chất lượng từng Unit trong phần mềm.

+ Phát hiện lỗi sớm và chỉnh sửa kịp thời.

+ Giảm chi phí.

+ Tái sử dụng được.

+ Giúp chúng ta Design.

1. Phương pháp xây dựng Unit Test:

* Phân tích các tình huống có thể xảy ra đối với mã.
* Mọi UT phải bắt đầu với trạng thái “fail” và chuyển trạng thái “pass” sau một số thay đổi hợp lý đối với mã chính.
* Khi viết một đoạn mã quan trọng, hãy viết các UT tương ứng.
* Số lượng Test Case đủ lớn để phát hiện điểm yếu của mã
* Sớm nhận biết các đoạn mã không ổn định và có nguy cơ gây lỗi cao, viết UT tương ứng để khống chế.
* Ứng với mỗi đối tượng nghiệp vụ (business object) hoặc đối tượng truy cập dữ liệu (data access object), nên tạo ra một lớp kiểm tra riêng vì những lỗi nghiêm trọng có thể phát sinh từ các đối tượng này.
* Để ngăn chặn các lỗi có thể phát sinh trở lại thực thi tự động tất cả UT mỗi khi có một sự thay đổi quan trọng, hãy làm công việc này mỗi ngày. Các UT lỗi cho chúng ta biết thay đổi nào là nguyên nhân gây lỗi.
* Để tăng hiệu quả và giảm rủi ro khi viết các UT, cần sử dụng nhiều phương thức kiểm tra khác nhau. Hãy viết càng đơn giản càng tốt.
* Cuối cùng, viết UT cũng đòi hỏi sự nỗ lực, kinh nghiệm và sự sáng tạo như viết PM.

1. Các đối tượng thay thế (**MOCK OBJECTS**)

* Đặc điểm:
* Đơn giản hơn đối tượng thực nhưng vẫn giữ được sự tương tác với các đối tượng khác.
* Không lặp lại nội dung đối tượng thực.
* Cho phép thiết lập các trạng thái riêng trợ giúp kiểm tra.

1. **Sử dụng mocks khi nào thì thích hợp(when):**

* Có thể sử dụng “mock object” để mô phỏng các đối tượng thật(real object) sau:
* Không có hành vi cụ thể => không thể đoán trước kết quả
* Khó cài đặtXử lý chậm
* Khó xảy ra và dễ gây lỗi
* Object liên quan giao diện người dùng
* Object chưa tồn tại dạng mã (mới trên bản thiết kế)

1. **MOCK OBJECTS(tt)**

* 3 bước quan trọng để sử dụng các đối tượng thay thế trọng kiểm thử:
* Đưa ra interface để mô tả đối tượng.
* Viết nội dung cho đối tượng thực dựa trên interface
* Trích interface từ đối tượng thật và triển khai mock object(MO) dựa trên interface đó.

1. **Các dạng Mocks**

Có thể dùng nhiều kĩ thuật để test các module trong chương trình:

* Stubs
* Mock
* Test a servlet

1. **Simple Stubs**

**Giả sử gọi hàm getTime() để trả về thời gian hiện tại:**

public long getTime()

{

return System.currentTimemillis();

}

public long getTime()

{

if(debug)

{

return debug\_cur\_time;

}

else{

return System.currentTimemillis();

**}**

**}**

1. **Mock**

**Tạo 1 interface: Kiểm thử 1 lớp truy xuất cơ sở dữ liệu:**

public class RealDAO

{

public string getName( long id)

{

//….

}

}

Tạo 1 interface:

public interface Environmental

{

public long getTime();

// Other methods omitted...

**}**

1. **MOCK(tt)**

**Áp dụng class interface cho lớp muốn kiểm thử:**

public class SystemEnvironment implements

Environmental {

public long getTime()

{

return System.currentTimeMillis();

}

// other methods ...

}

1. **MOCK(tt)**

**Tạo class mock kế thừa từ interface này:**

public class MockSystemEnvironment

implements Environmental {

public long getTime() {

return current\_time;

}

public void setTime(long aTime) {

current\_time = aTime;

}

private long current\_time;

// ...

}

1. **ACCEPTANCE TEST**
2. Định nghĩa acceptance test:

* Acceptance testing thường là trách nhiệm của khách hàng hoặc người dùng của hệ thống; các bên liên quan khác cũng có thể có liên quan.
* Aceptance Test là giai đoạn test có mục đích chính là làm thỏa mãn và tìm kiếm lòng tin của khách hàng với sản phẩm phần mềm của mình

1. Mục tiêu của acceptance test:

* Kiểm tra các qui trình xử lý công việc trên hệ thống đã được tích hợp đầy đủ nhất.
* Các qui trình hoạt động và bảo trì
* Các thủ tục người dùng (ví dụ: phân quyền dựa trên user login)
* Các form (ví dụ, các màn hình nhập liệu)
* Các Report (ví dụ các bản báo cáo như phiếu thu để in, báo cáo doanh thu…)
* Mục tiêu của acceptance testing là xác nhận lại sự tin tưởng vào hệ thống, các đặc tính thuộc về chức năng hoặc phi chức năng của hệ thống. Tìm kiếm lỗi không phải là trọng tâm chính của Acceptance testing. Acceptance testing có thể đánh giá sự sẵn sàng của hệ thống để triển khai và sử dụng, mặc dù không nhất thiết phải là mức cuối cùng của việc kiểm thử. Ví dụ, một cuộc kiểm thử tích hợp hệ thống ở quy mô lớn có thể được thực hiện sau khi đã thực hiện acceptance test đối với một hệ thống.

1. Một số loại acceptance testing thông thường:

* User acceptance testing (Kiểm thử chấp nhận người dùng)

Thông thường dùng để kiểm tra tính phù hợp với người dùng của hệ thống, công việc này được thực hiện bởi người dùng của doanh nghiệp.

* Operational (acceptance) testing (Kiểm thử chấp nhận hoạt động)
* Chấp nhận hệ thống bởi các quản trị viên hệ thống (system administrators), bao gồm:
* Kiểm thử phần backup/restore (sao lưu - phục hồi hệ thống)
* Khôi phục lại hệ thống sau khi có sự cố (Disaster recovery) như cúp điện, hỏa hoạn...
* Quản trị người dùng (User management) như phân quyền, lần lượt login bằng tất cả các quyền.
* Các nhiệm vụ bảo trì (Maintenance tasks)
* Các nhiệm vụ tải data và di chuyển data. (Data load and migration tasks)
* Kiểm tra các lỗ hổng bảo mật định kỳ. (Periodic checks of security vulnerabilities)
* Contract and regulation acceptance testing (Kiểm thử chấp nhận hợp đồng và thỏa thuận)

Contract acceptance testing được thực hiện với tiêu chí chấp nhận một hợp đồng nâng cấp - phát triển phần mềm. Tiêu chuẩn chấp nhận cần được xác định khi các bên thoả thuận hợp đồng. Điều lệ chấp nhận thử nghiệm được thực hiện đối với bất kỳ quy định phải được dựa vào, chẳng hạn như quy định của chính phủ, pháp luật hoặc các điều lệ an toàn.

* Alpha and beta (or field) testing (Kiểm thử alpha và beta)

Phát triển của thị trường hoặc COTS, phần mềm thường muốn nhận được phản hồi từ khách hàng tiềm năng hoặc khách hàng trong thị trường của họ trước khi sản phẩm phần mềm được đóng gói để thương mại. Alpha testing được thực hiện tại nơi tổ chức phát triển nhưng không phải do nhóm phát triển test mà do khách hàng thực hiện test. Beta testing hoặc field-testing được thực hiện bởi khách hàng hoặc các khách hàng tiềm năng tại nơi của họ (khách hàng download bản beta và cài vào máy mình rồi sử dụng).

1. **Stability testing**
2. Khái niệm:

* Stability testing là 1 quá trình non-functional của việc test hành vi của các ứng dụng bằng cách áp dụng tải tối đa. Nó được thực hiện để tìm ra khả năng mở rộng của hệ thống trong 1 môi trường nhất định.
* Stability testing là khả năng duy trì hoạt động của sản phẩm xuyên suốt và vượt quá thời gian sử dụng của nó mà không hỏng hoặc xảy ra lỗi. Đây là một kỹ thuật non-functional, với mục đích đòi hỏi khả năng chịu tải của phần mềm tới mức tối đa. Stability testing được thực hiện để kiểm tra hiệu quả của 1 sản phẩm được phát triển vượt qua mức hoạt động bình thường, hay tới 1 điểm dừng.

1. Lợi ích:

Stability testing cho phép bạn kiểm tra:

* Ước lượng độ tin cậy trong sự ổn định của hệ thống của bạn đang thử nghiệm
* Đảm bảo rằng hệ thống của bạn có thể xử lý các chương trình lớn
* Theo dõi hiệu quả của hệ thống của bạn
* Kiểm tra hệ thống ổn định theo yêu cầu

1. Công cụ cho performance testing như: webload, loadRunner, Apache JMeter, NeoLoad, CloudTest, Loadstorm, LoadUI, WAPT, LoadImpact
2. **Automation testing**
3. Khái niệm:

Kiểm thử tự động hay còn gọi là Automation testing đóng một vai trò quan trọng góp phần nâng cao năng suất kiểm thử, giảm thiểu lỗi cũng như sự nhàm chán với việc kiểm thử bằng tay trong một thời gian dài hoặc lặp đi lặp lại.

1. Ưu, nhược điểm:

* Ưu điểm:
  + Độ tin cậy cao
  + Khả năng lặp
  + Khả năng tái sử dụng
  + Tốc độ cao
  + Chi phí thấp
* Nhược điểm:
  + Khó mở rộng, bảo trì
  + Khả năng bao phủ thấp
  + Vấn đề về công cụ và nhân lực

1. Automation tester:

Để trở thành automation tester cần có kiến thức về:

* types testing
* testing techiques
* Nắm rõ về HTML và XPATH,
* Thành thạo ít nhất một ngôn ngữ lập trình thong dụng , hỗ trợ cho việc scripting trên test tools,
* Sử dụng thành thạo thư viện của Selenium WebDriver API bởi Selenium open source,
* Sử dụng thành thạo ít nhất 1 framework testing: Junit/TestNG/NUnit/...
* Tìm hiểu về software design pattern để build framework/common library
* Sử dụng/build framework thành thạo từ Page Object Model pattern.
* Kỹ năng về coding/IDE
* Học hỏi công nghệ mới trong mảng automation testing