UNIT TEXT

1. Khái niệm về Unit Test

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

Vì Unit được chọn để kiểm tra thường có kích thước nhỏ và chức năng hoạt động đơn giản, chúng ta không khó khăn gì trong việc tổ chức, kiểm tra, ghi nhận và phân tích kết quả kiểm tra nên việc  phát hiện lỗi sẽ dễ dàng xác định nguyên nhân và khắc phục cũng tương đối dễ dàng vì chỉ khoanh vùng trong một Unit đang kiểm tra.

Mỗi UT sẽ gửi đi một thông điệp và kiểm tra câu trả lời nhận được đúng hay không, bao gồm:

* Các kết quả trả về mong muốn
* Các lỗi ngoại lệ mong muốn

Các đoạn mã UT hoạt động liên tục hoặc định kỳ để thăm dò và phát hiện các lỗi kỹ thuật trong suốt quá trình phát triển, do đó UT còn được gọi là kỹ thuật kiểm nghiệm tự động.  
UT có các đặc điểm sau:

* Đóng vai trò như những người sử dụng đầu tiên của hệ thống.
* Chỉ có giá trị khi chúng có thể phát hiện các vấn đề tiềm ẩn hoặc lỗi kỹ thuật.

**Khi làm Unit test chúng ta thường thấy các khái niệm sau:**

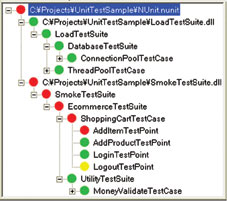
1. **Assertion:** Là một phát biểu mô tả các công việc kiểm tra cần tiến hành, thí dụ: AreEqual(), IsTrue(), IsNotNull()… Mỗi một UT gồm nhiều assertion kiểm tra dữ liệu đầu ra, tính chính xác của các lỗi ngoại lệ ra và các vấn đề phức tạp khác như:  
   – Sự tồn tại của một đối tượng  
   – Điều kiện biên: Các giá trị có vượt ra ngoài giới hạn hay không  
   – Thứ tự thực hiện của các luồng dữ liệu  
   …
2. **Test Point:** Là một đơn vị kiểm tra nhỏ nhất, chỉ chứa đơn giản một assertion nhằm khẳng định tính đúng đắn của một chi tiết mã nào đó. Mọi thành viên dự án đều có thể viết một test point.
3. **Test Case:** Là một tập hợp các test point nhằm kiểm tra một đặc điểm chức năng cụ thể, thí dụ toàn bộ giai đoạn người dùng nhập dữ liệu cho đến khi thông tin được nhập vào cơ sở dữ liệu. Trong nhiều trường hợp kiểm tra đặc biệt và khẩn cấp có thể không cần đến test case.
4. **Test Suite:** Là một tập hợp các test case định nghĩa cho từng module hoặc hệ thống con.
5. **Regression Testing (hoặc Automated Testing):** Là phương pháp kiểm nghiệm tự động sử dụng một phần mềm đặc biệt. Cùng một loại dữ liệu kiểm tra giống nhau nhưng được tiến hành nhiều lần lặp lại tự động nhằm ngăn chặn các lỗi cũ phát sinh trở lại. Kết hợp Regression Testing với Unit Testing sẽ đảm bảo các đoạn mã mới vẫn đáp ứng yêu cầu thay đổi và các đoạn mã cũ sẽ không bị ảnh hưởng bởi các hoạt động bảo trì.
6. **Production Code:** Phần mã chính của ứng dụng được chuyển giao cho khách hàng.
7. **Unit Testing Code:**Phần mã phụ để kiểm tra mã ứng dụng chính, không được chuyển giao cho khách hàng.

### 2. Vòng đời Unit Test

UT có 3 trạng thái cơ bản:

* Fail (trạng thái lỗi)
* Ignore (tạm ngừng thực hiện)
* Pass (trạng thái làm việc)

Toàn bộ UT được vận hành trong một hệ thống tách biệt. Có rất nhiều PM hỗ trợ thực thi UT với giao diện trực quan. Thông thường, trạng thái của UT được biểu hiện bằng các màu khác nhau: màu xanh (pass), màu vàng (ignore) và màu đỏ (fail)

.

UT chỉ thực sự đem lại hiệu quả khi:

* Được vận hành lặp lại nhiều lần
* Tự động hoàn toàn
* Độc lập với các UT khác.

3. Thiết kế Unit test

Mỗi UT đều được tiết kế theo trình tự sau:

* Thiết lập các điều kiện cần thiết: khởi tạo các đối tượng, xác định tài nguyên cần thiết, xây dựng các dữ liệu giả…
* Triệu gọi các phương thức cần kiểm tra.
* Kiểm tra sự hoạt động đúng đắn của các phương thức.
* Dọn dẹp tài nguyên sau khi kết thúc kiểm tra.

4. Ứng dụng Unit test

* Kiểm tra mọi đơn vị nhỏ nhất là các thuộc tính, sự kiện, thủ tục và hàm.
* Kiểm tra các trạng thái và ràng buộc của đối tượng ở các mức sâu hơn mà thông thường chúng ta không thể truy cập được.
* Kiểm tra các quy trình (process) và mở rộng hơn là các khung làm việc(workflow – tập hợp của nhiều quy trình)

5. Lợi ích của việc áp dụng Unit test

Thời gian đầu, người ta thường do dự khi phải viết UT thay vì tập trung vào code cho các chức năng nghiệp vụ. Công việc viết Unit Test có thể mất nhiều thời gian hơn code rất nhiều nhưng lại có lợi ích sau:

* Tạo ra môi trường lý tưởng để kiểm tra bất kỳ đoạn code nào, có khả năng thăm dò và phát hiện lỗi chính xác, duy trì sự ổn định của toàn bộ PM và giúp tiết kiệm thời gian so với công việc gỡ rối truyền thống.
* Phát hiện các thuật toán thực thi không hiệu quả, các thủ tục chạy vượt quá giới hạn thời gian.
* Phát hiện các vấn đề về thiết kế, xử lý hệ thống, thậm chí các mô hình thiết kế.
* Phát hiện các lỗi nghiêm trọng có thể xảy ra trong những tình huống rất hẹp.
* Tạo hàng rào an toàn cho các khối mã: Bất kỳ sự thay đổi nào cũng có thể tác động đến hàng rào này và thông báo những nguy hiểm tiềm tàng.

Trong môi trường làm việc Unit Test còn có tác dụng rất lớn đến năng suất làm việc:

* Giải phóng chuyên viên QA khỏi các công việc kiểm tra phức tạp.
* Tăng sự tự tin khi hoàn thành một công việc. Chúng ta thường có cảm giác không chắc chắn về các đoạn mã của mình như liệu các lỗi có quay lại không, hoạt động của module hiện hành có bị tác động không, hoặc liệu công việc hiệu chỉnh mã có gây hư hỏng đâu đó…
* Là công cụ đánh giá năng lực của bạn. Số lượng các tình huống kiểm tra (test case) chuyển trạng thái “pass” sẽ thể hiện tốc độ làm việc, năng suất của bạn.

6. Cách code hiệu quả với Unit Test

Phân tích các tình huống có thể xảy ra đối với mã. Đừng bỏ qua các tình huống tồi tệ nhất có thể xảy ra, thí dụ dữ liệu nhập làm một kết nối cơ sở dữ liệu thất bại, ứng dụng bị treo vì một phép toán chia cho không, các thủ tục đưa ra lỗi ngoại lệ sai có thể phá hỏng ứng dụng một cách bí ẩn…

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.

Mỗi khi viết một đoạn mã quan trọng, hãy viết các UT tương ứng cho đến khi bạn không thể nghĩ thêm tình huống nào nữa.

Nhập một số lượng đủ lớn các giá trị đầu vào để phát hiện điểm yếu của mã theo nguyên tắc:

* Nếu nhập giá trị đầu vào hợp lệ thì kết quả trả về cũng phải hợp lệ
* Nếu nhập giá trị đầu vào không hợp lệ thì kết quả trả về phải không hợp lệ

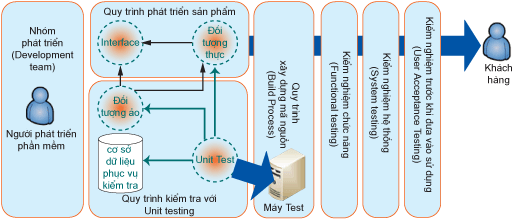
Sớm nhận biết các đoạn mã không ổn định và có nguy cơ gây lỗi cao, viết UT tương ứng để khống chế.

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.



Video hướng dẫn sử dụng: https://www.youtube.com/watch?v=oKj5aDF3EwA