CSC13010 - Thiết kế phần mềm

Unit Test Report

Mục lục

I. Thông tin thành viên

Họ tên	MSSV
Giang Đức Nhật	22120252
Nguyễn Quốc Tường	22120413
Nguyễn Trường Vũ	22120441
Bùi Đình Gia Vỹ	22120450

II. Unit Test

A. Tổng quan về Unit Test

1. Unit Test là gì?

Trước khi tìm hiểu về Unit Test, ta cần hiểu khái niệm "Unit". Unit được định nghĩa là một hành vi đơn lẻ trong hệ thống cần kiểm thử (SUT). Thông thường, trong lập trình hướng đối tượng (OOP), Unit có thể là một phương thức (method) hoặc một lớp (class). Tuy nhiên, điều này không có nghĩa rằng mọi phương thức hay lớp đều được coi là một Unit. Chẳng hạn, một "enum chỉ chứa các constant" không được xem là một Unit trong Unit Test vì nó không có hành vi nào để test cả.

Sau khi đã hiểu về Unit, ta có thể dễ dàng tiếp cận khái niệm Unit Test. Đơn giản, Unit Test là quá trình kiểm tra hoạt động của từng Unit để đảm bảo chúng vận hành đúng như mong đợi. Trong Unit Test, ta viết code để kiểm thử tất cả các trường hợp có thể xảy ra, nhằm đảm bảo coverage (*sẽ được giải thích ở phần sau*) cao nhất có thể.

2. Tại sao cần Unit Test?

Sau khi tìm hiểu về Unit Test, có lẽ ta đã hình dung được tầm quan trọng của nó trong quá trình phát triển phần mềm. Trước hết, Unit Test giúp đảm bảo rằng code hoạt động đúng như mong đợi. Thứ hai, khi cập nhật hoặc chỉnh sửa code, việc testing trở nên dễ dàng hơn và tiết kiệm công sức. Cuối cùng, Unit Test còn giúp chứng minh chất lượng của từng dòng code. Chính vì vậy, Unit Test luôn là một bước không thể thiếu khi phát triển bất kỳ feature nào.

B. Các loại Code Coverage quan trọng

1. Line Coverage

Là tỷ lệ dòng code được thực thi trong quá trình test, được tính bằng công thức:

$$Line\ Coverage = \frac{số\ dòng\ code\ được\ thực\ thi}{tổng\ số\ dòng\ code} * 100\%$$

Chỉ số này giúp đánh giá coverage của Unit Test và đảm bảo rằng các dòng quan trọng trong code đều được kiểm tra.

2. Branch Coverage

Là tỷ lệ các nhánh if, else, switch case được thực thi trong quá trình test, được tính bằng công thức:

$$Branch\ Coverage = \frac{số\ nhánh\ được\ thực\ thi}{tổng\ số\ nhánh} * 100\%$$

Chỉ số này giúp đảm bảo mọi nhánh logic đều được test, tránh lỗi do các edge cases.

3. Function Coverage

   Xác định xem các function/method có được gọi trong quá trình test hay không, đồng thời đảm bảo rằng tất cả function/method đều được test đầy đủ. Điều này giúp phát hiện các đoạn code chưa được test và cải thiện mức độ Coverage, đảm bảo rằng không có phần nào trong code bị bỏ sót.

4. Path Coverage

   Đảm bảo rằng tất cả các đường đi logic trong một đoạn code đều được kiểm tra bằng cách test mọi nhánh có thể xảy ra. Điều này giúp phát hiện các edge cases trong if, else, switch-case và đảm bảo rằng mọi kịch bản có thể xảy ra đều được xử lý đúng cách.

C. Mức độ Coverage

Mức độ Coverage giúp đánh giá chất lượng phần mềm, nhưng không có nghĩa là phần mềm không có bug. Mỗi doanh nghiệp thường đặt ra tiêu chuẩn về Coverage, thường nằm trong khoảng 80-90%. Việc tuân thủ tiêu chuẩn này là bắt buộc, nhưng không nên chạy đua theo chỉ số Coverage một cách máy móc.

Bên cạnh Unit Test, còn có Integration Test để đảm bảo các luồng hoạt động trơn tru hay Performance Test để đánh giá khả năng chịu tải của hệ thống. Do đó, Unit Test chỉ nên được xem là một bước giúp xác minh từng Unit hoạt động đúng theo mong muốn của lập trình viên, chứ không đảm bảo business logic hoàn toàn chính xác.

D. Best practices khi viết Unit Test

1. Viết test dễ đọc, dễ hiểu và dễ maintain
   - Nên tách biệt test logic với implement
   - Nên sử dụng mock, stub để không đụng đến database hay các third-party
   - Tránh hardcode
   - Nên tuân thủ các naming convention, các best practice tương như lúc code features
2. Không nên phụ thuộc vào implementation
   - Hãy giữ quan điểm “Viết test để tìm bug, không phải để luôn pass” nên hãy viết để đảm bảo logic chạy đúng
   - Không nên kiểm thử những đoạn code không có logic quan trọng như getter, setter,...
   - Không nên chạy theo coverage một cách máy móc
3. Kiểm thử mọi test cases
   - Không nên chỉ kiểm tra happy case mà phải kiểm tra cả edge cases

4. Một vài best practices khác
   - Tuân thủ FIRST Principles
   - Tích hợp Unit Test vào DI
   - Áp dụng các patterns phổ biến như: Given-When-Then, Arrange-Act-Assert,...

E. Công cụ hỗ trợ

- **SonarQube**: giúp kiểm tra chất lượng code (naming convention, duplication,..) và test coverage trong suốt quá trình coding.
- **SonarCloud**: là công cụ có thể được tích hợp vào GitHub, Bitbucket, Gitlab. Giúp kiểm tra coverage, các issues của code mới được pushed lên một cách tự động.
- **Các IDE**: một vài IDE có tích hợp sẵn chạy unit test và tính coverage mà không cần cài đặt gì thêm (*IntelliJ, PyCharm, WebStorm,...*).

F. Kết luận

Việc viết unit test không chỉ để đạt được tỷ lệ coverage cao mà quan trọng là kiểm thử đủ các luồng logic quan trọng. Áp dụng best practices giúp unit test hiệu quả, dễ bảo trì và đảm bảo chất lượng phần mềm.