TÌM HIỂU VỀ UNIT TESTING

1. Tìm Hiểu Về Các Loại Code Coverage Quan Trọng

1.1. Line coverage (Kiểm tra tỷ lệ dòng code được thực thi)

- Line coverage (bao phủ dòng) là một phiên bản cơ bản của test coverage (bao phủ kiểm thử). Khi một bộ test được chạy, code coverage sẽ ghi nhận những dòng mã nào đã được thực thi. Do đó, line coverage được tính bằng tổng số dòng đã chạy chia cho tổng số dòng trong codebase.
- Line coverage giúp đánh giá mức độ kiểm thử của mã nguồn, nhưng nó không đảm bảo rằng mọi trường hợp có thể xảy ra trong logic của chương trình đều được kiểm tra. Ví dụ: nếu một dòng mã chứa điều kiện (if statement) nhưng chỉ một nhánh được chạy, line coverage vẫn có thể đạt 100%, dù chưa kiểm tra tất cả các luồng thực thi.
- Ví dụ:
    - Chúng ta có hai hàm, add (cộng) và divide (chia). Cả hai đều nhận vào hai số a và b. Tuy nhiên, trong hàm divide, chúng ta kiểm tra xem b có bằng 0 hay không. Điều này hợp lý vì chúng ta không muốn thực hiện phép chia cho 0.

```
def add(a, b):
    return a + b
def divide(a, b):
    if b == 0:
        return "undefined"
    return a/b
```

    - Nếu a = 1 và b = 2 thì điều kiện if b == 0: là sai, do đó dòng return “undefined” sẽ không được chạy.

```
def add(a, b):
    return a + b
def divide(a, b):
    if b == 0:
        return "undefined" # Dòng này bị bỏ qua
    return a/b
```

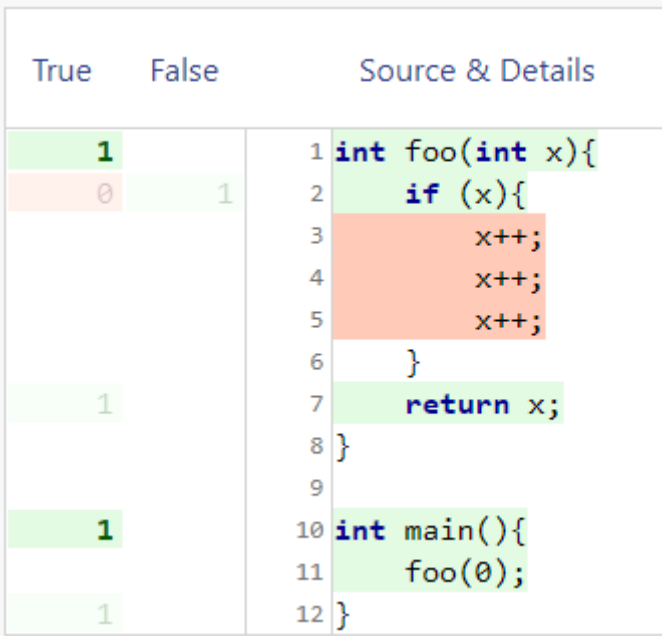
    - Trong trường hợp của chúng ta, tỷ lệ bao phủ (coverage percentage) sẽ là 83.3% vì có 5 dòng được thực thi trên tổng số 6 dòng.

1.2. Branch Coverage (Đảm bảo tất cả các nhánh (if, else, switch) được kiểm thử)

- Các nhánh (branches) thường xuất hiện trong các câu lệnh if, khi có hai hướng đi dựa trên kết quả đánh giá điều kiện. Do đó, branch coverage đo lường số lượng nhánh đã được thực thi so với tổng số nhánh có trong mã nguồn.
- Ví dụ:
    - Trong mã của chúng ta, chỉ có một câu lệnh if với hai nhánh: nhánh true: Khi b == 0 và nhánh false: Khi b != 0.
    - Branch coverage sẽ kiểm tra xem cả hai nhánh này có được thực thi hay không khi chạy bộ kiểm thử.

1.3. Function Coverage (Xác minh tất cả các hàm/method được gọi ít nhất một lần)

- Function Coverage xem xét đến việc các hàm (hoặc method) trong chương trình có được gọi ít nhất một lần trong quá trình test hay không. Điều này đảm bảo rằng mọi chức năng do lập trình viên viết ra đều đã được "chạm tới" khi chạy unit test. Một hàm không bao giờ được gọi trong quá trình test có thể tiềm ẩn lỗi mà chưa được phát hiện.
- Trong ví dụ sau, độ bao phủ hàm đầy đủ (2/2) được đạt được vì cả hai hàm, foo và main, đều đã được gọi.



1.4. Path Coverage (Đảm bảo các đường đi logic quan trọng đều được test)

- Path Coverage là dạng phức tạp và mạnh mẽ nhất, khi nó yêu cầu kiểm tra tất cả các đường đi logic có thể xảy ra trong chương trình – bao gồm cả các tổ hợp điều kiện phức tạp. Mức path coverage cao thể hiện rằng hệ thống đã được kiểm thử trong tất cả các trạng thái có thể, nhưng việc đạt được mức path coverage 100% thường khó khăn do số lượng tổ hợp logic tăng rất nhanh.

2. Mức Code Coverage Tối Thiểu Trong Thực Tế

- Trong thực tiễn phát triển phần mềm, không phải lúc nào mục tiêu cũng là đạt được 100% code coverage. Thay vào đó, người ta thường xác định mức độ phù hợp dựa trên loại hệ thống và tính chất của từng dự án cụ thể.
- Đối với các ứng dụng đơn giản như hệ thống quản lý CRUD (Create – Read – Update – Delete), mức coverage từ 70% đến 80% được xem là đủ tốt để đảm bảo các chức năng chính đều được kiểm thử. Tuy nhiên, nếu ứng dụng thuộc nhóm có yêu cầu cao về độ an toàn và ổn định – chẳng hạn như hệ thống tài chính, y tế, hoặc xử lý thông tin nhạy cảm – thì mức coverage yêu cầu có thể lên tới 95% hoặc thậm chí là tuyệt đối.
- Nhiều tổ chức và công ty phần mềm lớn cũng thường áp dụng ngưỡng tối thiểu là 80% code coverage để xem xét một pull request hoặc một tính năng mới có thể được merge vào nhánh chính hay không. Dù vậy, đây không nên là một con số cứng nhắc. Lập trình viên và nhóm phát triển nên linh hoạt, đặt trọng tâm vào việc kiểm thử đầy đủ các logic quan trọng thay vì theo đuổi con số cao một cách máy móc.
- Điều cần lưu ý là: một đoạn mã có coverage cao vẫn có thể chứa lỗi nếu test không kiểm tra đúng điều kiện hoặc chỉ test "cho có". Ngược lại, coverage thấp không hẳn là xấu nếu phần không được test là những logic ít quan trọng như logging, config hoặc đoạn mã rất khó có thể xảy ra.

3. Best Practices khi viết Unit Test

3.1. AAA (Arrange, Act, Assert)

- Arrange: Thiết lập dữ liệu, khởi tạo đối tượng, xác định kết quả mong đợi.
- Act: Gọi hàm cần kiểm thử và lưu kết quả.
- Assert: So sánh kết quả thu được với kết quả mong đợi.

3.2. Ngắn gọn & Đơn giản

- Viết kiểm thử ngắn, dễ hiểu, không chứa logic phức tạp.
- Tránh trùng lặp, áp dụng nguyên tắc DRY (Don't Repeat Yourself).

- Nếu cần thiết, sử dụng các hàm xác nhận tùy chỉnh.

3.3. Viết Test Trước Code (TDD)

- Tạo test case trước khi viết code giúp đảm bảo tính kiểm thử và dễ bảo trì.
- Giúp phát hiện lỗi và lỗ hổng trong yêu cầu sớm hơn.

3.4. Headless Testing

- Dùng PhantomJS hoặc Headless Chrome để chạy kiểm thử không cần giao diện đồ họa, giúp tiết kiệm tài nguyên.
- Không phù hợp với các kiểm thử cần hiển thị trực quan.

3.5. Kiểm thử Đầy Đủ Kịch Bản

- Kiểm thử cả kết quả mong đợi và không mong đợi.
- Bao gồm các trường hợp biên (min/max giá trị).

3.6. Tuân thủ Tiêu Chuẩn Ngành

- Đảm bảo kiểm thử tuân thủ quy định như HIPAA, PCI-DSS nếu cần.
- Ghi lại quá trình kiểm thử để hỗ trợ kiểm toán.

3.7. Kiểm thử Một Yêu Cầu Một Lần

- Không kiểm thử toàn bộ phương thức trong một test case.
- Khi yêu cầu thay đổi, chỉ cần cập nhật test tương ứng.

TÀI LIỆU THAM KHẢO

- [1] Hu, T. (2022, October 12). *Line or branch coverage: Which type of coverage is right for you?* Codecov. Retrieved April 4, 2025, from https://about.codecov.io/blog/line-or-branch-coverage-which-type-is-right-for-you/
- [2] Đinh, T. (n.d.). *Cách thực hiện unit test chi tiết và những lưu ý quan trọng*. TesterPro.vn. Retrieved April 4, 2025, from https://testerpro.vn/unit-test/