ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

MÔN:

THIẾT KẾ PHẦN MỀM

Bài tập 04 – Unit Test

Giảng viên phụ trách : Trần Duy Thảo
Đề tài : Bài tập 04
Nhóm : niceScore

Thành phố Hồ Chí Minh, tháng 4 năm 2025

MỤC LỤC

I. DANH SÁCH THÀNH VIÊN

- MSSV: 22120062 – Nguyễn Đăng Điền
- MSSV: 2212016 – Huỳnh Tấn Lộc
- MSSV: 22120199 – Trần Lượng

II. NỘI DUNG

1. Tổng quan về Unit Test Coverage

Unit Test Coverage (Tỷ lệ bao phủ mã nguồn trong unit test) là một kỹ thuật giúp xác định xem các trường hợp thử nghiệm (test case) có đủ bao quát hay không.

Trong Unit Test, đôi khi các trường hợp thử nghiệm (test case) không đủ bao quát, khiến một số đoạn mã, hàm, đường đi logic, … không được kiểm thử đầy đủ. Test coverage sẽ giúp tối ưu quy trình kiểm thử, xác định lỗ hổng, phạm vi còn thiếu trong unit test, cũng như ngăn ngừa lỗi rò rỉ.

2. Các loại Code Coverage quan trọng

2.1. Line Coverage

Line Coverage được định nghĩa là tỷ lệ dòng lệnh được thực thi trong kiểm thừ. Line Coverage được tính theo công thức:

$$Line\ Coverage = \frac{Số\ dòng\ lệnh\ được\ thực\ thi}{Tổng\ số\ dòng\ lệnh} * 100$$

Line Coverage có tỷ lệ thấp thể hiện trong mã nguồn được kiểm thử có nhiều mã "chết" (mã không được thực thi), hoặc trường hợp thử nghiệm (test case) không đủ bao quát, bỏ sót một số dòng lệnh chưa kiểm thử. Truy ngược về những dòng chưa được thực thi, ta có thể kiểm tra lại mã nguồn hoặc thêm test case phù hợp.

2.2. Statement Coverage

Statement Coverage là tỷ lệ câu lệnh được thực thi trong kiểm thử so với tổng số câu lệnh:

$$Statement\ Coverage = \frac{Số\ câu\ lệnh\ được\ thực\ thi}{Tổng\ số\ câu\ llệnh} * 100$$

Statement Coverage tương tự với Line Coverage, khác nhau ở việc một số câu lệnh (statement) cò thể nằm trên nhiều dòng, hoặc 1 dòng chứa nhiều câu lệnh. Hai Code Coverage này hoàn toàn có thể dùng thay thế nhau.

2.3. Branch Coverage (Decision Coverage)

Branch Coverage đo lường tỷ lệ nhánh và đường dẫn trong mã nguồn được kiểm thử, dựa trên công thức:

$$Branch\ Coverage = \frac{Số\ nhánh/kết\ quả\ lựa\ chọn\ được\ thực\ thi}{Tổng\ số\ nhánh/kết\ quả\ lựa\ chọn} * 100$$

Branch Coverage đảm bảo rằng mọi nhánh của cấu trúc điều khiển (if-else, switch, …) đều được kiểm thử. Branch Coverage có tính bao quát hơn Line Coverage/ Statement Coverage.

2.4. Function Coverage (Tỷ lệ bao phủ hàm)

Function Coverage là tỷ lệ hàm được gọi và thực thi ít nhất một lần trong bài kiểm thử:

$$Function\ Coverage = \frac{Số\ hàm\ được\ thực\ thi\ ít\ nhất\ 1\ lần}{Tổng\ số\ hàm} * 100$$

Function Coverage giúp phát hiện hàm thừa/không được sử dụng, hoặc trường hợp kiểm thử bị sót.

2.5. Path Coverage (Tỷ lệ bao phủ đường)

Path Coverage tính toán số lượng đường đi trong unit test trên tổng số đường đi có thể có trong dự án.

$$Path\ Coverage = \frac{Số\ đường\ đi\ trong\ unit\ test}{Tổng\ số\ đường\ có\ thể\ đi} * 100$$

Khác với Branch Coverage chỉ quan tâm mỗi nhánh có được đi qua hay không, Path Coverage còn quan tâm đến tổ hợp kết quả của các nhánh đó. Trong unit test thực tế, Path Coverage có thể rất lớn, tốn thời gian và phức tạp, nhưng phạm vi kiểm tra rộn và đảm bảo hơn Branch Coverage nếu đủ tốt.

2.6. Condition Coverage

Condition Coverage kiểm tra xem liệu mọi biểu thức đúng – sai đã được kiểm thử chưa. Condition Coverage đòi hỏi bài test phải có đủ 2 kết quả cho mỗi điều kiện

đúng – sai. Nếu điều kiện được ghép bởi các điều kiện nhỏ hơn, cần đảm bảo các điều kiện con nhận cả hai giá trị đúng – sai trong các trường hợp.

2.7. Loop Coverage

Loop Coverage tính toán xem mỗi vòng lặp có được thực thi không, và số lần lặp trong thử nghiệm. Một bộ thử nghiệm tốt, đạt Loop Coverage cao phải được thử 0 lần (điều kiện dừng từ đầu), 1 lần và nhiều lần lặp cho các vòng lặp, đảm bảo vòng lặp vận hành chính xác ổn định.

Loop Coverage giúp phát hiện lỗi liên quan đến điều kiện dừng và khả năng xử lý dữ liệu lặp trong chương trình.

3. Mức Coverage tối thiểu

3.1. Mức độ Code Coverage với các hệ thống tương tự:

- DO-178B: Tiêu chuẩn hàng không 1992, yêu cầu code coverage 100% cho các hệ thống an toàn khẩn cấp. Do liên quan đến ngành hàng không, yêu cầu tiêu chuẩn rất cao.

Ảnh hưởng của sự cố hệ thống	**Ví dụ**	**Code Coverage yêu cầu**
Thảm họa	Rơi máy bay	100% condition/branch coverage, 100% statement coverage
Nguy hiểm	Hành khách tử vong	100% branch coverage, 100% statement coverage
Lớn	Hành khách bị thương	100% statement coverage

Nhỏ	Thay đổi lịch trình bay	Không
Hầu như không	Hệ thống giải trí gặp sự cố	Không

- IEC 61508:2010: An toàn chức năng của hệ thống điện tử

Mức độ an toàn	**100% statements coverage**	**100% branches coverage**	**100% conditions coverage**
1 (ít nghiêm trọng)	Được khuyến nghị	Được khuyến nghị	Được khuyến nghị
2	Yêu cầu	Được khuyến nghị	Được khuyến nghị
3	Yêu cầu	Yêu cầu	Được khuyến nghị
4 (nghiệm trọng nhất)	Yêu cầu	Yêu cầu	Yêu cầu

- ISO 26262: "An toàn chức năng cho phương tiên đường bộ"

Phương pháp	**ASIL**			
	A (ít nghiêm trọng)	**B**	**C**	**D (nghiêm trọng nhất)**

Statement coverage	Yêu cầu	Yêu cầu	Được khuyến nghị	Được khuyến nghị
Branch coverage	Được khuyến nghị	Yêu cầu	Yêu cầu	Yêu cầu
Modified Condition/Decision Coverage	Được khuyến nghị	Được khuyến nghị	Được khuyến nghị	Yêu cầu

3.2. Mức độ Code Coverage phù hợp

- Google đề xuất 60%: Chấp nhận được – 75%: Đáng khen – 90%: Xuất sắc
- Bullseye Testing Technology đề xuất 70% ~ 80% cho đa số các dự án.
- Trong hệ thống lớn, bài kiểm tra đạt tỷ lệ bao phủ mã nguồn cao (100%) là không cần thiết:
    - Chi phí, thời gian, nguồn lực khổng lồ (đặc biệt là Path Coverage)
    - Mục đích chính của Code Coverage là phát hiện những đoạn mã chưa được kiểm tra trong kiểm thử. Code Coverage không đánh giá toàn diện chất lượng của mã nguồn hay bài kiểm tra. Một số bài kiểm tra chất lượng thấp (lặp kiểm tra, tốn thời gian, …) có thể đạt Code Coverage cao, nhưng thiếu ý nghĩa thực tế.
    - Một số đoạn mã là tính năng trong các phiên bản tới, chưa cần thiết hoặc được ứng dụng ở phiên bản hiện tại.

  Do đó, khi độ bao phủ mã nguồn đạt mức độ tốt (70~80%), ta nên dành nhân lực, thời gian cho thay đổi, sửa chữa mã nguồn, cân bằng giữa Code Coverage và hiệu quả kiểm thử.

4. Best Practice khi viết Unit Test

- Không Phụ Thuộc Vào Database: Sử dụng mock dependencies thay vì truy cập trực tiếp vào database để kiểm thử dễ dàng và bảo trì.

- Kiểm Thử Happy Case và Edge Case: Bao gồm các trường hợp thành công và đặc biệt để đảm bảo hệ thống xử lý đúng mọi tình huống.
- Tránh Trùng Lặp và Phụ Thuộc Vào Implementation Details: Mỗi test kiểm tra tình huống riêng biệt, không phụ thuộc vào chi tiết triển khai.

Ngoài ra, cần tuân thủ thêm best practices:

- Viết test trong quá trình phát triển để phát hiện lỗi sớm.
- Mỗi test chỉ có một assertion, giúp kiểm thử đơn giản và dễ hiểu.
- Kiểm thử với giá trị biên và tránh logic phức tạp để giữ cho các test dễ đọc.
- Tránh logic phức tạp: Giữ cho các bài kiểm thử dễ đọc và dễ hiểu.

III. TÀI LIỆU THAM KHẢO

- Code Coverage trong Unit Test https://www.geeksforgeeks.org/code-coverage-testing-in-software-testing/
- Tại sao Test Coverage là một phần quan trọng của Kiểm thử phần mềm? https://topdev.vn/blog/tai-sao-test-coverage-la-mot-phan-quan-trong-cua-kiem-thu-phan-mem/
- What is code coverage? https://www.atlassian.com/continuous-delivery/software-testing/code-coverage
- Minimum Acceptance Code Coverage https://www.bullseye.com/minimum.html
- Code Coverage Best Practice https://testing.googleblog.com/2020/08/code-coverage-best-practices.html
- What is a reasonable code coverage % for unit test (and why)? https://stackoverflow.com/questions/90002/what-is-a-reasonable-code-coverage-for-unit-tests-and-why
- Test Coverage https://martinfowler.com/bliki/TestCoverage.html
- The role of code coverage in regulations and standards https://about.codecov.io/blog/the-role-of-code-coverage-in-regulations-and-standards/
- Unit Test Best Practice: https://medium.com/@kaanfurkanc/unit-testing-best-practices-3a8b0ddd88b5

- Best Practice for Unit Testing: https://www.browserstack.com/guide/unit-testing-best-practices#:~:text=Best%20Practices%20for%20Unit%20Testing%201%20One%20Assertion,Write%20tests%20during%20development%2C%20not%20after%20it%20