TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

THIẾT KẾ PHẦN MỀM

UNIT TEST COVERAGE - BEST PRACTICES

Thành viên

Mã số	Họ và tên
21126028	Vũ Huyền Thiên Lý
22127410	Lưu Thanh Thuý
22127424	Nguyễn Phước Minh Trí
22127435	Võ Lê Việt Tú
22127444	Trần Thị Cát Tường

Giảng viên hướng dẫn

Trần Duy Thảo

Hồ Tuấn Thanh

Nguyễn Lê Hoàng Dũng

Mục lục

Chương 1

Code Coverage

Trong unit test, `code coverage` là một giá trị thể hiện độ bao phủ của `code` bởi các `test case` được biểu thị dưới dạng %, nó cho ta thấy được tỉ lệ `code` đã được thực thi bằng `unit test` trên tổng số `code` của một chương trình. `Code coverage` được tính bằng công thức sau:

$$\text{code coverage} = \frac{\text{items executed}}{\text{total number of code}}$$

1.1 Line Coverage

1.1.1 Định nghĩa

`Line coverage` thể hiện số câu lệnh của chương trình đã được thực thi trong quá trình test, chúng ta có thể hiểu nôm na `line coverage` ở đây chính là độ bao phủ các dòng lệnh, tức là số dòng lệnh đã được chạy qua khi test. Một dòng lệnh được tính là đã được thực thi khi chỉ một hoặc toàn bộ `code` của dòng lệnh đó được thực thi, vì thế khi độ bao phủ câu lệnh không đảm bảo rằng chương trình của bạn 100% không có lỗi.

Nói cách khác, `line coverage` kiểm tra để xác định tất cả các dòng code trong thiết kế đã được thực thi trong quá trình mô phỏng hay chưa.

1.1.2 Ví dụ

```
def foo(decision)
    a = 0
    if decision
        a = 1
    end
    a = 1 / a
    return a
end
```

Ở ví dụ trên dễ dàng đặt được 100% `line coverage` bằng cách test với input `foo(1)`, khi đó câu lệnh trong điều kiện `if` sẽ được thực thi, tuy nhiên nếu input là giá trị 0 thì ở dòng lệnh `return` sẽ xảy ra lỗi `division by zero`

1.2 Branch Coverage

1.2.1 Định nghĩa

`Branch Coverage Testing` là một kỹ thuật kiểm thử phần mềm tập trung vào việc kiểm tra tất cả các nhánh của cấu trúc điều kiện (`if, else, switch-case`) trong mã nguồn. Mục tiêu của kỹ thuật này là đảm bảo rằng tất cả các điều kiện và nhánh trong mã đều được kiểm thử để phát hiện lỗi và cải thiện độ tin cậy của phần mềm.

`Branch Coverage` được tính bằng công thức:

$$\text{Branch Coverage} = \left(\frac{\text{Số nhánh được kiểm tra}}{\text{Tổng số nhánh}} \right) \times 100\%$$

1.2.2 Ví dụ

```
def check_number(num):
    if num > 0:
        print("Positive")
    elif num < 0:
        print("Negative")
    else:
        print("Zero")
```

- Xác định các nhánh trong đoạn mã
  - Nhánh 1: `if (num > 0)` → Điều kiện đúng, in "Positive Number".
  - Nhánh 2: `else if (num < 0)` → Điều kiện đúng, in "Negative Number".
  - Nhánh 3: `else` → Khi cả hai điều kiện trước sai, in "Zero".
- Kiểm thử với các giá trị đầu vào

Test Case	Giá trị num	Nhánh được kiểm tra
TC1	5	Nhánh 1
TC2	-3	Nhánh 2
TC3	0	Nhánh 3

- Tính toán `Branch Coverage`

  Với 3 test case trên, chúng ta đã kiểm tra tất cả các nhánh có thể có trong đoạn mã, do đó:

$$\text{Branch Coverage} = \left(\frac{3}{3}\right) \times 100$$

- Kết luận
  - Để đạt 100% `Branch Coverage`, chúng ta phải kiểm thử tất cả các nhánh có thể xảy ra trong đoạn mã.
  - Kiểm thử với số lượng test case ít hơn có thể bỏ sót một số nhánh, dẫn đến việc giảm độ bao phủ kiểm thử.

1.3 Function Coverage

1.3.1 Định nghĩa

`Function Coverage` là một kỹ thuật trong kiểm thử phần mềm nhằm đo lường xem tất cả các hàm (functions) trong mã nguồn có được gọi ít nhất một lần trong quá trình kiểm thử hay không.

- Khi thực hiện kiểm thử, công cụ phân tích sẽ theo dõi các hàm nào đã được gọi và các hàm nào chưa được gọi.
- Nếu một hàm không được thực thi trong bất kỳ test case nào, có thể xảy ra lỗi tiềm ẩn vì hàm đó chưa được kiểm tra.
- `Function Coverage` giúp đảm bảo rằng tất cả các hàm ít nhất đã được thực thi một lần trong quá trình kiểm thử.

Công thức tính `Function Coverage`:

$$\text{Function Coverage} = \left(\frac{\text{Số hàm được gọi}}{\text{Tổng số hàm}}\right) \times 100\%$$

1.3.2 Ví dụ

Nếu một chương trình có 10 hàm nhưng chỉ có 8 hàm được gọi trong quá trình kiểm thử, `Function Coverage` sẽ là:

$$\left(\frac{8}{10}\right) \times 100\% = 80\%$$

Có nhiều công cụ hỗ trợ đo `Function Coverage` trong các ngôn ngữ lập trình khác nhau:

Ngôn ngữ	Công cụ đo `Function Coverage`
Python	`coverage.py`
Java	`JaCoCo, Cobertura`
C/C++	`gcov, lcov`
JavaScript	`Istanbul, Jest Coverage`

1.4 Path Coverage

1.4.1 Định nghĩa

`Path Coverage` đảm bảo rằng tất cả các đường đi có thể có qua mã nguồn đều đã được thực thi ít nhất một lần trong quá trình kiểm thử.

- Một chương trình có thể có nhiều nhánh điều kiện (`if-else, switch-case, vòng lặp for, whil`
- Một đường đi (path) là một chuỗi các câu lệnh được thực thi từ điểm bắt đầu đến điểm kết thúc của chương trình.
- `Path Coverage` kiểm tra tất cả các đường đi có thể có, giúp phát hiện nhiều lỗi hơn so với `Branch Coverage`.

Công thức tính `Path Coverage`:

$$\text{Path Coverage} = \left(\frac{\text{Số đường đi được kiểm tra}}{\text{Tổng số đường đi có thể có}}\right) \times 100\%$$

1.4.2 Ví dụ

```
def process_list(lst):
    for num in lst:
        if num % 2 == 0:
            print("Even")
        else:
            print("Odd")
```

Các đường đi có thể có:

- Danh sách rỗng (`lst = []`) → Không có vòng lặp nào chạy.
- Danh sách chỉ có số chẵn (`lst = [2, 4, 6]`) → Luôn in "even".
- Danh sách chỉ có số lẻ (`lst = [1, 3, 5]`) → Luôn in "odd".
- Danh sách có cả số chẵn và số lẻ (`lst = [2, 3, 4]`) → Vừa in "even", vừa in "odd".

Chương 2

Mức Coverage tối thiểu

Mức độ `Code Coverage` phù hợp có thể khác nhau tùy thuộc vào loại kiểm thử và yêu cầu cụ thể của dự án. Dưới đây là một số hướng dẫn chung:

- Kiểm thử đơn vị (Unit Testing): Mức độ bao phủ mã lệnh thường được khuyến nghị là khoảng 90%. Điều này đảm bảo rằng hầu hết các chức năng nhỏ nhất của ứng dụng đều được kiểm tra kỹ lưỡng.
- Kiểm thử tích hợp (Integration Testing): Mức độ bao phủ có thể thấp hơn, khoảng 80%. Kiểm thử tích hợp tập trung vào việc đảm bảo các mô-đun hoặc thành phần khác nhau hoạt động cùng nhau một cách chính xác.
- Kiểm thử hệ thống (System Testing): Mức độ bao phủ thường vào khoảng 70%. Giai đoạn này kiểm tra toàn bộ hệ thống để đảm bảo rằng tất cả các yêu cầu chức năng và phi chức năng đều được đáp ứng.

Chương 3

Best Practices khi viết Unit Test

Unit Test là một phần quan trọng trong kiểm thử phần mềm, giúp đảm bảo từng thành phần nhỏ nhất của hệ thống hoạt động chính xác. Tuy nhiên, để viết unit test hiệu quả, cần tuân theo các best practices nhằm tối ưu hiệu suất, dễ bảo trì và đảm bảo chất lượng code.

3.1 Viết Unit Test dễ bảo trì

Hạn chế phụ thuộc vào database hoặc persistent storage

- Vấn đề: Kiểm thử phụ thuộc vào database làm chậm tốc độ, khó kiểm soát dữ liệu test, và dễ bị ảnh hưởng khi môi trường thay đổi.
- Giải pháp:
  - Dùng mock dependencies thay vì database thật.
  - Nếu cần database, sử dụng in-memory database như SQLite (Python, Node.js) hoặc H2 (Java).

3.2 Kiểm thử cả Happy case và Edge case

- Happy case: Kiểm thử các đầu vào hợp lệ, đảm bảo chức năng hoạt động đúng.
- Edge case: Kiểm thử các trường hợp bất thường như giá trị rỗng, null, số âm, dữ liệu vượt quá giới hạn.

Cần kiểm thử cả đường đi thành công và các trường hợp lỗi để tránh bug không mong muốn

3.3 Không viết Test trùng lặp hoặc quá phụ thuộc vào Implementation Details

- Vấn đề:
  - Test trùng lặp làm tốn tài nguyên và khó bảo trì.
  - Viết test quá chi tiết vào cách triển khai có thể gây lỗi nếu code thay đổi, dù kết quả đầu ra không đổi.
- Giải pháp:
  - Test nên tập trung vào kết quả mong đợi (expected output) hơn là cách nội bộ chương trình hoạt động.
  - Tránh test trùng lặp: Nếu một hàm đã được kiểm thử thông qua một phương thức khác, không cần kiểm thử lại.

Cách tối ưu: Sử dụng parameterized tests để tránh trùng lặp.

3.4 Kiểm thử độc lập (Isolate Tests)

- Vấn đề:
  - Test không nên phụ thuộc vào trạng thái của test trước đó.
  - Nếu một test case bị lỗi, nó không được ảnh hưởng đến các test khác.
- Giải pháp:
  - Mỗi test case cần có dữ liệu riêng biệt, tránh dùng chung object hoặc trạng thái.
  - Dùng mock/stub thay vì gọi API hoặc database thật.

Nguồn tham khảo

Cornett, Steve. 2025. “What Is the Minimum Acceptable Code Coverage?” 2025. https://www.bullseye.com/minimum.html.

Pittet, Sten. 2025. “Code Coverage.” 2025. https://www.atlassian.com/continuous-delivery/software-testing/code-coverage.

Testim. 2025. “Unit Testing Best Practices.” 2025. https://www.testim.io/blog/unit-testing-best-practices/.

Vu, Phan Dang Hai. 2025. “Các Tiêu Chí Tính Độ Bao Phủ Code Trong Unit Test.” 2025. https://viblo.asia/p/cac-tieu-chi-tinh-do-bao-phu-code-trong-unit-test-GrLZD0w2Zk0.