ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

TÌM HIỂU VỀ UNIT TEST COVERAGE VÀ BEST PRACTICES

MÔN HỌC: THIẾT KẾ PHẦN MỀM

LỚP: 22KTPM3

Giảng viên hướng dẫn:

Trần Duy Thảo

Hồ Tuấn Thanh

Nguyễn Lê Hoàng Dũng

Sinh viên thực hiện:

Trần Đức Tùng – 22127442

Contents

I. Sinh viên thực hiện

- Họ và tên: Trần Đức Tùng
- Mã số sinh viên: 22127442
- Lớp: Thiết Kế Phần Mềm - 22KTPM3

II. Giới thiệu về unit testing

Unit testing được xem là nền tảng quan trọng trong việc đảm bảo chất lượng phần mềm, vì nó không chỉ giúp phát hiện sớm các lỗi logic mà còn khẳng định rằng mỗi đơn vị code – từ hàm đến module – hoạt động đúng theo thiết kế ban đầu. Một trong những yếu tố chủ chốt của unit testing là việc đo lường mức độ "coverage", tức là tỷ lệ phần trăm mã nguồn được chạy qua các test case. Báo cáo này cung cấp một cái nhìn tổng quan về các dạng coverage chính cũng như các best practices trong việc xây dựng unit test hiệu quả, từ đó giúp tăng cường khả năng phát hiện lỗi, cải thiện quá trình kiểm thử và hỗ trợ việc bảo trì cũng như mở rộng hệ thống trong tương lai. Qua đó, unit test không chỉ là công cụ phát hiện lỗi mà còn là tài liệu sống phản ánh hành vi của hệ thống, đóng góp vào việc tạo nên một sản phẩm phần mềm bền vững và dễ dàng nâng cấp.

III. Các Loại Code Coverage

1. Line Coverage

Line Coverage đo lường số dòng code đã được thực thi trong quá trình chạy test. Mục tiêu là đạt được mức càng cao càng tốt; tuy nhiên, việc đạt 100% line coverage không đồng nghĩa với việc code được kiểm thử đầy đủ về logic.

Ưu điểm:

- Đơn giản, dễ đo lường.

Nhược điểm:

- Không đảm bảo rằng tất cả các điều kiện logic (if/else) đều được kiểm thử.

2. Branch Coverage

Branch Coverage kiểm tra tất cả các nhánh (if/else, switch-case) trong code. Đây là chỉ số quan trọng để đảm bảo rằng mọi điều kiện đều được xét đến.

Ưu điểm:

- Bao phủ được nhiều trường hợp logic phức tạp.

Nhược điểm:

- Có thể tăng số lượng test cases cần viết, đặc biệt với các biểu thức phức tạp.

3. Function Coverage

Function Coverage đảm bảo rằng tất cả các hàm và phương thức được gọi ít nhất một lần trong quá trình chạy test. Điều này giúp kiểm tra sự tồn tại của các hàm, nhưng không đảm bảo tính đúng đắn của logic bên trong hàm.

Ưu điểm:

- Xác minh mọi hàm đều được gọi.

Nhược điểm:

- Không đảm bảo rằng các hàm được kiểm tra với các đầu vào khác nhau.

4. Path Coverage

Path Coverage là chỉ số kiểm tra tất cả các đường đi có thể xảy ra trong quá trình thực thi code. Đây là mức kiểm thử chi tiết nhất nhưng cũng khó đạt được đối với các chương trình phức tạp.

Ưu điểm:

- Đảm bảo kiểm thử toàn diện các luồng logic.

Nhược điểm:

- Số lượng đường đi có thể tăng rất nhanh, dẫn đến việc viết test case trở nên phức tạp và không thực tế.

5. Statement Coverage

Statement Coverage đo lường số lượng các câu lệnh (statements) trong code đã được thực thi trong quá trình chạy test. Nó cho biết phần trăm các câu lệnh được kiểm tra.

Ưu điểm:

- Dễ hiểu và đo lường, cung cấp cái nhìn tổng quát về việc chạy test.

Nhược điểm:

- Không phản ánh đầy đủ các nhánh logic và điều kiện phức tạp trong code.

6. Class/Module Coverage

Class/Module Coverage đảm bảo rằng tất cả các lớp hoặc module trong code đều được gọi và kiểm thử ít nhất một lần. Điều này giúp đánh giá phạm vi kiểm thử của cấu trúc phần mềm.

Ưu điểm:

- Đảm bảo rằng mọi thành phần cấu trúc (lớp/module) đều được bao phủ trong quá trình kiểm thử.

Nhược điểm:

- Không đảm bảo rằng các logic phức tạp bên trong các lớp/module đó đều được kiểm thử kỹ lưỡng.

IV. Mức Coverage Tối Thiểu và Định Nghĩa Trong Industry

Trong thực tế, không có con số cố định cho mức code coverage tối thiểu, tuy nhiên nhiều dự án phần mềm và công ty thường đặt mục tiêu từ 70% đến 80% coverage. Điều này đảm bảo rằng phần lớn các logic quan trọng được kiểm thử mà không quá tập trung vào việc đạt 100% – điều này đôi khi không khả thi và có thể gây lãng phí tài nguyên.

Một số tiêu chuẩn trong industry khuyến nghị:

- 70-80% Coverage: Đối với hầu hết các ứng dụng.
- 80-90% Coverage: Đối với hệ thống có mức độ rủi ro cao (ví dụ: hệ thống tài chính, y tế).
- Không cần ép buộc 100% Coverage: Nếu các đoạn code không có ý nghĩa kinh doanh hoặc là code xử lý lỗi không thể xảy ra trong điều kiện thực tế.

V. Best Practices Khi Viết Unit Test

Để đảm bảo các bài kiểm tra đơn vị hiệu quả và dễ bảo trì, người phát triển nên tuân thủ một số nguyên tắc sau:

1. Tách rời Dependencies

- **Sử dụng Mock và Stub:** Tránh phụ thuộc trực tiếp vào database, file system, hay các API bên ngoài. Sử dụng các framework như unittest.mock giúp mô phỏng hành vi của các dependency này, làm cho unit test chạy nhanh và độc lập.
- **Dependency Injection:** Cho phép truyền các dependency vào các module thay vì khởi tạo bên trong, từ đó dễ dàng thay thế bằng các đối tượng giả lập trong test.

2. Viết Test Cho Cả Happy Case và Edge Case

- **Happy Case:** Kiểm thử các trường hợp hoạt động bình thường theo thiết kế.
- **Edge Case:** Kiểm thử các đầu vào không hợp lệ, các tình huống ngoại lệ hoặc giới hạn, đảm bảo rằng hệ thống xử lý đúng khi gặp lỗi.

3. Giữ Cho Unit Test Nhỏ Gọn và Độc Lập

- Mỗi unit test nên kiểm tra một chức năng cụ thể.
- Tránh để các test case phụ thuộc vào nhau, vì điều này gây khó khăn trong việc xác định nguyên nhân khi có lỗi xảy ra.

4. Không Phụ Thuộc Vào Implementation Details

- Tập trung kiểm thử hành vi (behavior) của module, thay vì cách thức hiện thực bên trong.
- Điều này giúp cho bộ test không bị phá vỡ khi code được refactor mà không thay đổi chức năng.

5. Sử Dụng Framework và Công Cụ Hỗ Trợ

- **Pytest:** Là framework test mạnh mẽ, cung cấp khả năng mở rộng qua fixture, parameterized test và tích hợp với coverage.py.
- **Coverage.py:** Giúp đo lường mức độ kiểm thử của code, từ đó lập tức biết được phần nào chưa được kiểm tra.

6. Kiểm Tra Sớm và Thường Xuyên

- Tích hợp unit test vào quy trình CI/CD để chạy tự động mỗi khi có thay đổi code.
- Việc này giúp phát hiện lỗi sớm và giảm thiểu rủi ro khi triển khai vào môi trường sản xuất.

Những best practices này không chỉ giúp cải thiện chất lượng code mà còn tạo ra một bộ test có khả năng mở rộng, bảo trì và phát triển trong tương lai.

VI. Công Cụ Hỗ Trợ và Ứng Dụng

Trong quá trình viết unit test, việc lựa chọn công cụ phù hợp sẽ giúp quá trình kiểm thử trở nên hiệu quả hơn. Một số công cụ được sử dụng phổ biến bao gồm:

1. coverage.py (Freetuts, n.d.)

- **Chức năng:** Đo lường mức độ code coverage của ứng dụng.
- **Ưu điểm:** Cung cấp báo cáo chi tiết về số dòng code được test, các nhánh chưa được kiểm thử, giúp phát hiện các khu vực cần bổ sung thêm test case.

2. Pytest

- **Chức năng:** Framework test cho Python hỗ trợ viết unit test với cú pháp đơn giản và khả năng mở rộng cao.
- **Ưu điểm:** Hỗ trợ fixtures, parameterized test, cho phép viết các test case độc lập và dễ đọc.

3. Unittest.mock

- **Chức năng:** Thư viện tích hợp trong Python để tạo mock objects, giúp giả lập các dependency ngoài.
- **Ưu điểm:** Giảm thiểu sự phụ thuộc vào các yếu tố bên ngoài, làm cho unit test trở nên nhanh chóng và ổn định.

4. CI/CD Tools (docuwriter.ai, n.d.)

- **Ví dụ:** Jenkins, GitHub Actions, Travis CI.
- **Chức năng:** Tích hợp chạy unit test tự động mỗi khi code được commit, giúp đảm bảo chất lượng phần mềm liên tục.
- **Ưu điểm:** Phát hiện lỗi sớm, báo cáo kết quả kiểm thử ngay lập tức cho team phát triển.

Những công cụ này được áp dụng rộng rãi trong industry, đảm bảo rằng các bài kiểm tra không chỉ chạy được mà còn mang lại những thông tin hữu ích để cải thiện code.

VII. Kết Luận và Đề Xuất Cải Tiến

1. Kết Luận

Qua quá trình nghiên cứu, ta nhận thấy:

- Code Coverage là một chỉ số hữu ích để đánh giá mức độ kiểm thử của hệ thống, tuy nhiên nó không phản ánh đầy đủ chất lượng của test case.
- Việc áp dụng các best practices trong unit testing như tách rời dependencies, kiểm thử cả happy case lẫn edge case, và sử dụng mock giúp nâng cao hiệu quả kiểm thử.
- Các công cụ như pytest, coverage.py và CI/CD là nền tảng hỗ trợ mạnh mẽ cho quá trình kiểm thử trong các dự án phần mềm hiện đại.

2. Đề Xuất Cải Tiến Thiết Kế

Để ứng dụng quản lý sinh viên trở nên dễ kiểm thử và bảo trì hơn, cần cân nhắc:

- **Phân Tách Module:** Tách các phần logic xử lý kinh doanh ra khỏi các thao tác I/O, giúp viết test cho logic riêng biệt.
- **Dependency Injection:** Áp dụng để giảm sự phụ thuộc cứng vào các dịch vụ bên ngoài, tạo điều kiện cho việc sử dụng mock.
- **Tách Biệt Giao Diện và Logic:** Thiết kế hệ thống theo mô hình MVC hoặc service layer để giao diện người dùng không ảnh hưởng đến các unit test.
- **Tăng Cường Sử Dụng Fixture:** Sử dụng các fixture trong pytest để chuẩn bị dữ liệu test một cách nhất quán, giảm thiểu sự lặp lại trong các test case.

Việc cải tiến thiết kế không chỉ giúp cho quá trình test trở nên dễ dàng mà còn giúp dự án dễ dàng mở rộng và bảo trì lâu dài.