Báo Cáo: Unit Test Coverage và Best Practices

1. Giới Thiệu

Unit test là một phẫn quan trông trong qui trình phát triển phần mềm, giúp kiểm thử tự động và đảm bảo chất lượng code. Trong báo cáo này, chúng ta sẽ tìm hiểu về **unit test coverage** và **best practices** khi viết unit test cho ứng dụng quản lý sinh viên.

2. Unit Test Coverage

Unit test coverage là chỉ số đo lường tỷ lệ code được kiểm thử bằng unit test. Dưới đây là các loại code coverage quan trọng:

2.1 Line Coverage: Đo tỷ lệ dòng code được thực thi khi chạy test.

- Định nghĩa: Đo lường số dòng mã nguồn đã được thực thi trong quá trình chạy test.
- Ví dụ: Nếu một hàm có 10 dòng lệnh, test thực thi 8 dòng → line coverage là 80%.
- Ý nghĩa:
  - Dễ đo, phổ biến nhất.
  - Giúp phát hiện những dòng code chưa được kiểm thử.
- Hạn chế: Không đảm bảo rằng tất cả các nhánh điều kiện hoặc logic phức tạp đã được test.

2.2 Branch Coverage: Kiểm tra tất cả các nhánh (if-else, switch-case) đã được test hay chưa.

- Định nghĩa: Đo số lượng nhánh rẽ nhánh điều kiện (ví dụ: if, else, case) đã được test.
- Ví dụ: Trong một điều kiện if-else, nếu chỉ test if đúng, thì branch coverage chỉ là 50%.
- Ý nghĩa:
  - Giúp phát hiện các nhánh logic chưa được kiểm tra.
  - Phù hợp cho các hàm chứa nhiều điều kiện rẽ nhánh.
- Cách cải thiện: Cần thiết kế test case sao cho mỗi nhánh đều được đi qua ít nhất một lần.

2.3 Function Coverage: Xác minh tất cả các hàm/method đã được gọi ít nhất một lần.

- Định nghĩa: Kiểm tra xem tất cả các hàm hoặc phương thức trong chương trình có được gọi ít nhất một lần hay không.
- Ví dụ: Nếu bạn có 10 hàm, nhưng test chỉ gọi 7 hàm → function coverage là 70%.
- Ý nghĩa:
  - Phù hợp với các hệ thống lớn có nhiều hàm chưa chắc đã được dùng/test đầy đủ.
  - Hạn chế: Không đảm bảo nội dung trong hàm đã được test kỹ.

2.4 Path Coverage: Đảm bảo tất cả các đường đi logic quan trọng đều được kiểm thử.

- Định nghĩa: Đảm bảo tất cả các đường đi logic có thể xảy ra qua chương trình đều được kiểm thử.
- Ý nghĩa:
  - Rất toàn diện, đảm bảo nhiều tổ hợp điều kiện được kiểm tra.
  - Phù hợp với hệ thống có logic phức tạp, yêu cầu cao về độ tin cậy.
- Hạn chế:
  - Rất tốn thời gian và khó đạt được 100% (số đường đi tăng theo cấp số nhân).
  - Cần phân tích kỹ các nhánh, vòng lặp, điều kiện lồng nhau.

3. Mức Code Coverage Tối Thiểu

- Trong thực tế, **không có con số "chuẩn tuyệt đối"** áp dụng cho mọi dự án. Mức tối thiểu phụ thuộc vào:
  - **Mức độ phức tạp** của hệ thống
  - **Mức độ rủi ro** (liên quan đến dữ liệu người dùng, tài chính, sức khỏe,…)
  - **Ngân sách và thời gian dự án**
- Tuy nhiên, có một số **mốc tham khảo chung trong ngành phần mềm**:

Loại hệ thống / ngành	Mức Coverage khuyến nghị
Dự án thông thường	**70% - 80%**
Hệ thống lớn, critical	**85% - 90%**
Ngành tài chính, y tế	**90% - 100%**
MVP, PoC nhỏ	**50% - 60%**

- **Ý Nghĩa Của Từng Mức Coverage**

  - **< 50%**: Rất thấp, hầu như không thể đảm bảo chất lượng. Rất rủi ro nếu có thay đổi code.
  - **50%-70%**: Tạm chấp nhận cho các sản phẩm MVP, nội bộ, hoặc đang phát triển nhanh.
  - **70%-85%**: Mức tốt cho đa số ứng dụng sản phẩm. Đã bao phủ phần lớn logic.
  - **85%-95%**: Mức lý tưởng, yêu cầu test kỹ nhiều edge case, nhánh phụ.
  - **> 95%**: Gần như toàn bộ logic đã được kiểm thử – phù hợp cho hệ thống đòi hỏi độ chính xác cao (fintech, healthcare, IoT,...)

4. Best Practices Khi Viết Unit Test

4.1. Nguyên Tắc Viết Test Hiệu Quả

- **Test tự độc lập**: Mỗi test case không được phụ thuộc vào nhau.
- **Sử dụng Mock khi cần thiết**: Tránh kết nối với database hoặc API bên ngoài.
- **Bao quát Happy Case và Edge Case**: Kiểm tra cả trường hợp hệ thống hoạt động bình thường và trường hợp dữ liệu biên.
- **Tránh test trùng lắp**: Viết test ngắn gọn, tránh lắp lại logic.

4.2. Các Mô Hình Viết Test Tốt

- **AAA (Arrange - Act - Assert)**:
  - *Arrange*: Chuẩn bị dữ liệu.
  - *Act*: Gọi hàm/method cần test.
  - *Assert*: Kiểm tra kết quả mong muốn.
- **FIRST**:
  - *Fast*: Test nhanh.
  - *Independent*: Tự độc lập.
  - *Repeatable*: Kết quả nhất quán.
  - *Self-validating*: Tự kiểm tra kết quả.
  - *Timely*: Viết test song song với code.

---

5. Kết Luận

Unit test coverage giúp đảm bảo chất lượng và độ tin cậy của code. Tuy nhiên, việc đạt 100% coverage không phải lúc nào cũng cần thiết. Quan trọng nhất là viết test chất lượng, dễ bảo trì, bao quát đủ các trường hợp quán trọng.

Các bước tiếp theo:

- Áp dụng best practices vào việc viết test cho ứng dụng quản lý sinh viên.
- Sử dụng các công cụ như **JUnit (Java), PyTest (Python), Jest (JavaScript)** để đo và phân tích code coverage.

📌 **Tài liệu tham khảo**:

- "Unit Testing Principles, Practices, and Patterns" - Vladimir Khorikov
- Documentation của các framework test như JUnit, PyTest, Jest.
