**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO BÀI TẬP**

**Unit Test Coverage và Best Practices**

**1. Các Loại Code Coverage Quan Trọng**

**a. Line Coverage**

* Đo lường số lượng dòng code đã được thực thi khi chạy test.
* Nếu một dòng code không được thực thi trong quá trình kiểm thử, thì có khả năng nó không được sử dụng hoặc có thể chứa lỗi tiềm ẩn.
* Line coverage cao giúp phát hiện các đoạn code chết hoặc không được sử dụng.
* **Ví dụ:**

A black background with white text

AI-generated content may be incorrect.

* + Để đạt line coverage tốt, cần có test case gọi phương thức add(2,3).

**b. Branch Coverage**

* Xác minh rằng tất cả các nhánh điều kiện (if, else, switch) đều được kiểm thử.
* Giúp phát hiện lỗi xảy ra khi một số nhánh điều kiện không được kiểm tra.
* Đảm bảo rằng không có nhánh nào trong chương trình bị bỏ sót khi kiểm thử.
* **Ví dụ:**

A screen shot of a computer code

AI-generated content may be incorrect.

* + Cần có test cases:
    - checkNumber(5) để test nhánh if.
    - checkNumber(-3) để test nhánh else.

**c. Function Coverage**

* Kiểm tra xem tất cả các hàm/method trong code có được gọi ít nhất một lần hay không.
* Nếu một hàm không có test case nào gọi đến, có thể nó không cần thiết hoặc chưa được sử dụng đúng cách.
* Đảm bảo mỗi chức năng trong ứng dụng đều hoạt động chính xác.
* **Ví dụ:**

A screen shot of a computer screen

AI-generated content may be incorrect.

* + Cần có test case gọi printMessage() để đảm bảo function này được kiểm thử.

**d. Path Coverage**

* Đảm bảo rằng tất cả các đường đi logic quan trọng trong chương trình đều được kiểm thử.
* Đặc biệt quan trọng trong các hệ thống có nhiều nhánh rẽ hoặc vòng lặp.
* Path coverage cao giúp phát hiện các lỗi logic trong chương trình.
* **Ví dụ:**

A screen shot of a computer program

AI-generated content may be incorrect.

* + Cần có test cases:
    - calculate(5, 3, true) để test nhánh if.
    - calculate(5, 3, false) để test nhánh else.

**2. Mức Coverage Tối Thiểu Trong Industry**

* Không có tiêu chuẩn cố định cho code coverage, nhưng một số mức phổ biến:
  + **70-80%**: Được coi là mức chấp nhận được cho hầu hết các hệ thống phần mềm.
  + **90% trở lên**: Thường chỉ áp dụng cho các hệ thống quan trọng, yêu cầu tính ổn định cao như phần mềm tài chính, y tế.
  + **Dưới 50%**: Thường không đủ để đảm bảo chất lượng phần mềm.
* Mức coverage hợp lý phụ thuộc vào tính chất của dự án:
  + **Ứng dụng doanh nghiệp**: Cần kiểm thử các quy trình nghiệp vụ quan trọng.
  + **Ứng dụng thời gian thực**: Cần test kỹ các phần xử lý đồng thời, cạnh tranh tài nguyên.
  + **Hệ thống nhúng**: Quan trọng để kiểm thử các điều kiện lỗi và hành vi bất thường.
* Các tổ chức lớn thường có quy định nội bộ về coverage tối thiểu:
  + Google: Thường yêu cầu tối thiểu 80% test coverage.
  + Microsoft: Khuyến khích đạt trên 85% với các thành phần quan trọng.
  + Facebook: Tập trung vào test các module có rủi ro cao hơn là đặt ra ngưỡng coverage cụ thể.

**3. Best Practices Khi Viết Unit Test**

**a. Viết Test Dễ Bảo Trì**

* Tránh phụ thuộc vào database hoặc persistent storage.
* Sử dụng mock hoặc stub cho các dependency bên ngoài.
* Viết test có thể dễ dàng hiểu và cập nhật khi code thay đổi.
* Viết test theo cấu trúc **AAA (Arrange - Act - Assert)** để dễ đọc và bảo trì.
* Ví dụ sử dụng Mockito trong Java:

A screen shot of a computer program

AI-generated content may be incorrect.

**b. Kiểm Thử Happy Case và Edge Case**

* **Happy Case**: Test với dữ liệu hợp lệ, kỳ vọng đầu ra chính xác.
* **Edge Case**: Kiểm thử với dữ liệu không hợp lệ, giá trị biên để đảm bảo chương trình xử lý đúng lỗi.
* **Exception Handling**: Kiểm tra cách chương trình xử lý ngoại lệ.
* Đảm bảo kiểm thử đầy đủ các tình huống có thể xảy ra.
* Ví dụ:

A computer screen shot of text

AI-generated content may be incorrect.

**c. Tránh Viết Test Trùng Lặp**

* Không lặp lại test case cho cùng một logic.
* Viết test ở mức hợp lý, tập trung vào kiểm thử hành vi hơn là implementation details.
* Ví dụ không nên kiểm thử cả getter và setter nếu chúng chỉ gán và trả về giá trị.

A screen shot of a computer program

AI-generated content may be incorrect.

* + Không cần test setter/getter vì chúng không có logic phức tạp.
* Thay vì viết nhiều test case riêng lẻ, sử dụng **Parameterized Test** để kiểm tra nhiều giá trị đầu vào.

A computer screen shot of text

AI-generated content may be incorrect.

* Với các best practices trên, việc viết unit test không chỉ giúp tăng coverage mà còn giúp phát hiện lỗi sớm, đảm bảo code chất lượng và dễ bảo trì hơn.

**4. Kết Luận**

Unit test coverage là một chỉ số quan trọng nhưng không nên chạy theo số lượng. Điều quan trọng hơn là đảm bảo kiểm thử đầy đủ các logic quan trọng, sử dụng các best practices để có unit test hiệu quả, dễ bảo trì và không phụ thuộc vào implementation details. Các doanh nghiệp lớn như Google, Microsoft đều có các tiêu chuẩn riêng về coverage để đảm bảo chất lượng phần mềm. Cuối cùng, mục tiêu chính của unit test không chỉ là đạt được coverage cao mà còn là đảm bảo hệ thống hoạt động ổn định và đáng tin cậy.