TDD – Test Driven Development

Module: Project & Job

Mục tiêu

- Trình bày được tư duy Test-First
- Trình bày được mô hình TDD
- Mô tả được vòng vận hành của TDD
- Trình bày được hoạt động refactoring và lợi ích của refactoring
- Trình bày được khái niệm UnitTest
- Giải thích cách viết Test-Case đơn giản
- Triển khai được Unit Test với Junit
- Áp dụng tư duy Test-first vào code
- Viết được UnitTest
- Chạy được UnitTest

Test First

Kiểm thử

Test First

Kiểm thử

- Kiểm thử (testing) là các nghiên cứu được thực hiện để khẳng định chất lượng của sản phẩm phần mềm
- Kiểm thử giúp quản lý được rủi ro của sản phẩm
- Các kỹ thuật kiểm thử bao gồm việc thực thi chương trình để tìm ra các bug (lỗi hoặc khiếm khuyết) và đảm bảo chương trình phù hợp để sử dụng
- 2 thao tác quan trọng khi kiểm thử đó là: thiết kế ca kiểm thử (test case) và thực thi ca kiểm thử

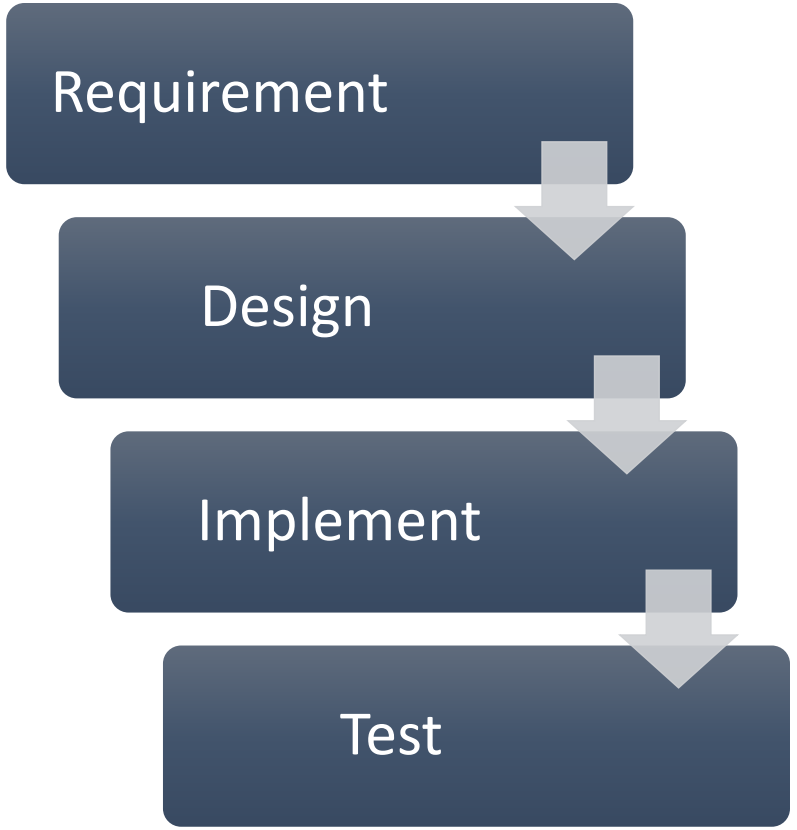
Các vai trò tham gia trong quá trình kiểm thử

- Tester
- QA
- Developer
- ScrumMaster
- PM
- Product Manager
- Product Owner
- BA
- Customer?

Developer có vai trò rất quan trọng trong việc đảm bảo chất lượng phần mềm thông qua kiểm thử.

Mô hình kiểm thử truyền thống

- Trong mô hình truyền thống, thao tác kiểm thử được thực hiện ở giai đoạn cuối của quá trình phát triển
- Các lỗi được phát hiện muộn sẽ dẫn đến:
  - Rủi ro tăng cao
  - Chi phí tăng cao
  - Không đạt được tiến độ phát hành
  - Mất cơ hội kinh doanh

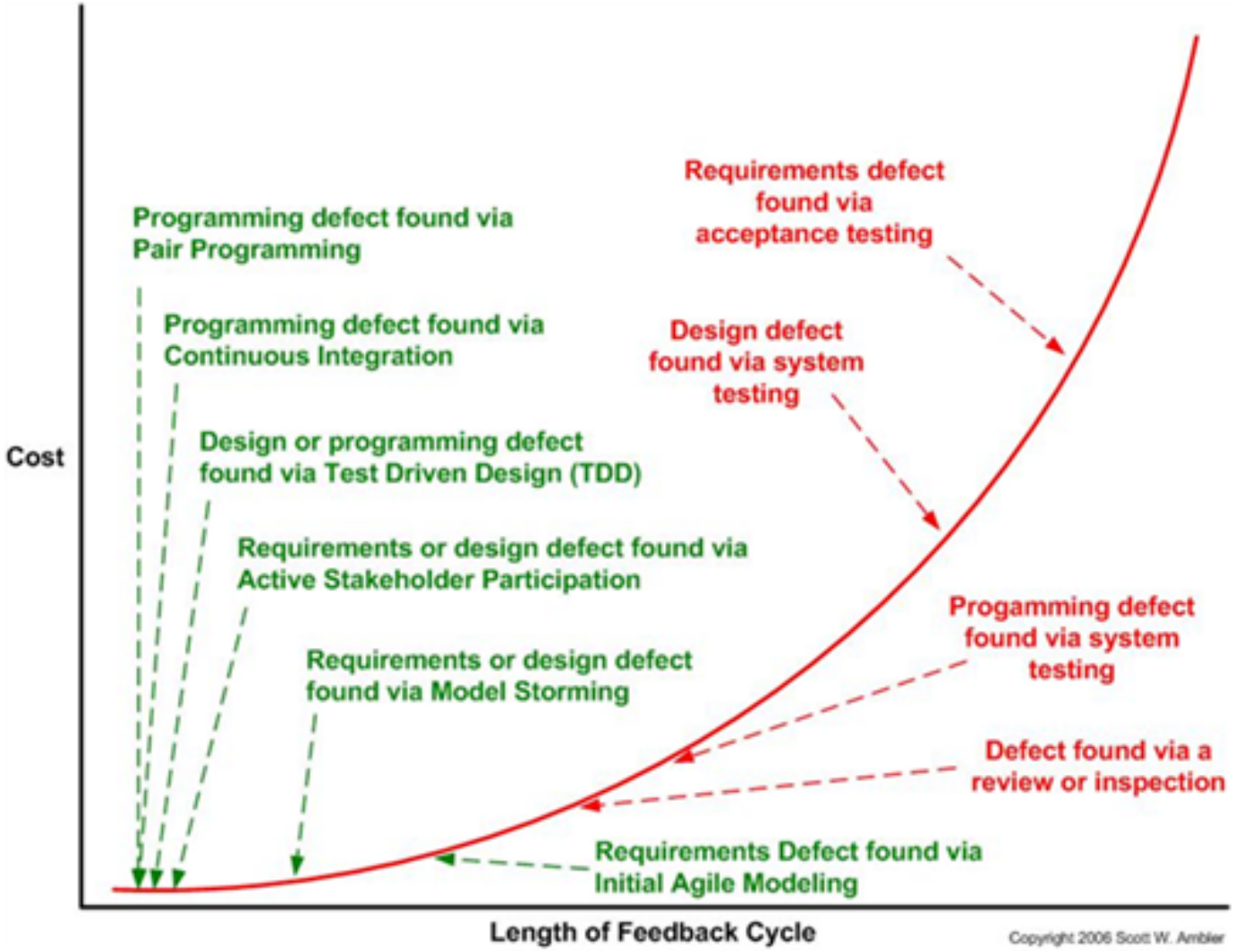


Test First và TDD

- Test First là một *cách tiếp cận* mới so với mô hình phát triển phần mềm truyền thống, trong đó việc lập trình bắt đầu bằng cách viết các bài kiểm thử trước khi bắt tay vào viết mã nguồn của chương trình
- TDD (Test Driven Development) là một *quy trình* lập trình, trong đó bao gồm nhiều giai đoạn nhỏ lặp đi lặp lại, mỗi giai đoạn bao gồm các bước:
    1. Viết các bài kiểm thử
    2. Viết mã nguồn
    3. Tái cấu trúc mã nguồn.

Kiểm thử càng sớm càng tốt

- Nếu các bug được phát hiện càng muộn thì chi phí sẽ càng cao

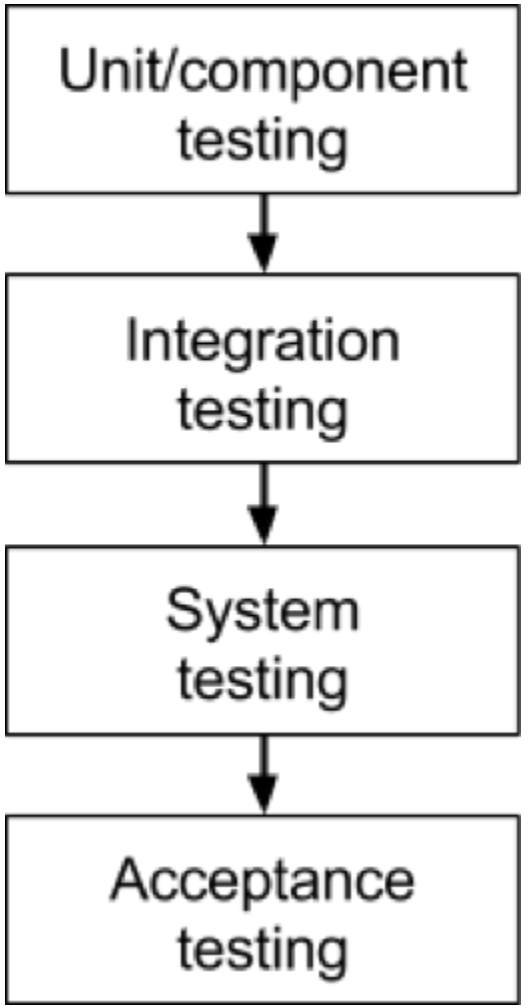


Unit Test và TDD

Các mức kiểm thử (Testing Level)
Unit Test
Kiểm thử tự động
TDD

Các mức kiểm thử

- Dựa vào đối tượng được kiểm thử, có thể chia thành 4 mức độ kiểm thử phổ biến như sau:
  - Kiểm thử Đơn vị (Unit Testing)
  - Kiểm thử Tích hợp (Integration Testing)
  - Kiểm thử Hệ thống (System Testing)
  - Kiểm thử Người dùng/Kiểm thử Chấp nhận (User Testing/Acceptance Testing)



Kiểm thử Đơn vị

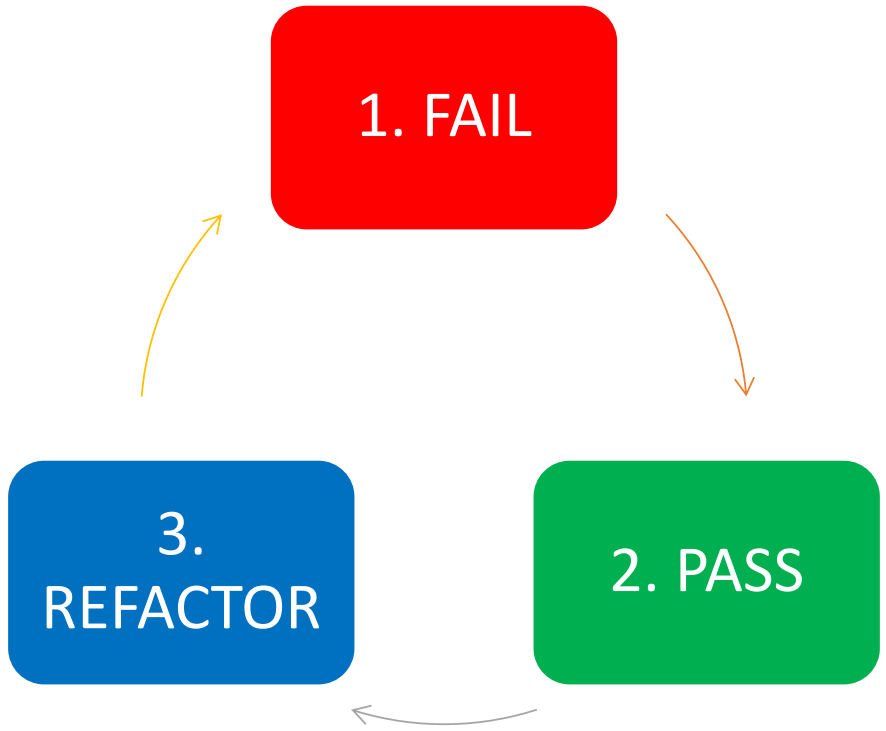
- Kiểm thử Đơn vị (Unit Testing) là thao tác kiểm thử được thực hiện trên các đơn vị/thành phần nhỏ của chương trình
- Kiểm thử Đơn vị được dùng để kiểm tra tính chính xác của các đơn vị mã nguồn nhỏ khi chúng được thực thi độc lập
- Một ”đơn vị” có thể là một khối lệnh, một phương thức, một lớp, một module…
- Một đơn vị thường là “nhỏ”: bao gồm ít đầu vào, ít đầu ra và không quá phức tạp
- Đơn vị thường được áp dụng phổ biến là ở mức phương thức
- Kiểm thử đơn vị thường được thực hiện bởi Lập trình viên

Kiểm thử tự động

- Kiểm thử tự động (automated testing) là hình thức sử dụng các công cụ để tự động thực thi các test case, khác với kiểm thử thủ công (manual testing – do con người thực hiện)
- Lợi ích của kiểm thử tự động:
  - Tăng tốc độ kiểm thử
  - Giảm chi phí
  - Dễ thực hiện kiểm thử hồi quy (regression testing)
- Kiểm thử đơn vị thường được tiến hành dưới hình thức tự động
- Một số công cụ hỗ trợ kiểm thử tự động phổ biến cho Java: Junit, TestNG, JTest, Mockito

Vòng thực thi của TDD

1. Viết Test
   - Fail bởi vì chưa có mã nguồn
   - Thông thường các IDE sẽ báo message màu đỏ
2. Viết mã nguồn
   - Viết mã nguồn để vượt qua test
   - Thông thường các IDE sẽ báo message màu xanh
3. Tái cấu trúc mã nguồn
   - Chỉnh sửa mã nguồn để "tốt" hơn
   - Cần đảm bảo mã nguồn vẫn vượt qua được các bài test



Refactoring

- Refactoring là quá trình tái cấu trúc mã nguồn (sắp xếp lại mã nguồn) mà không ảnh hưởng đến hành vi của chương trình nhằm giúp cho mã nguồn trở nên “tốt” hơn
- Các mục đích chính của refactoring:
  - Mã nguồn dễ bảo trì hơn
  - Mã nguồn dễ mở rộng hơn
- Có nhiều kỹ thuật refactor khác nhau, chẳng hạn như: Đổi tên biến, tách biến, tách phương thức, tách đối tượng, áp dụng Design Pattern…

JUnit

- JUnit là một Testing Framework được sử dụng cho ngôn ngữ Java
- Website: https://junit.org
- Junit thường được tích hợp sẵn trong các IDE thông dụng như Netbeans, Eclipse, IntelliJ IDEA
- Ngoài JUnit thì có TestNG là một Testing Framework thông dụng cho Java
- Có thể sử dụng JUnit kết hợp với các công cụ khác như Mockito, PowerMock, EvoSuite…

Viết test case trong Junit: Ví dụ

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

- Trong đó:
  - @Test được sử dụng để khai báo một phương thức test
  - assertEquals dùng để so sánh bằng 2 giá trị

Một lớp Test cơ bản

```
class StandardTests {
    @BeforeAll
    static void initAll() {
    }

    @BeforeEach
    void init() {
    }

    @Test
    void succeedingTest() {
    }

    @Test
    void failingTest() {
        fail("a failing test");
    }
```

```
...

    @Test
    @Disabled("for demonstration purposes")
    void skippedTest() {
        // not executed
    }

    @AfterEach
    void tearDown() {
    }

    @AfterAll
    static void tearDownAll() {
    }

}
```

Các assertions

- assertEquals()
- assertTrue()
- assertFalse()
- assertNull()
- assertNotNull()
- assertThrows()
- assertTimeout()
- assertArrayEquals()

[Thực hành] Ứng dụng máy tính đơn giản

[Thực hành] Tìm giá trị tuyệt đối

[Bài tập] Tìm ngày tiếp theo

[Bài tập] Phân loại tam giác

Tổng kết

- Kiểm thử là việc nghiên cứu để khẳng định chất lượng sản phẩm
- Test First là một cách tiếp cận mới về phát triển phần mềm
- Trong TDD, các test case được viết trước khi viết mã nguồn
- TDD bao gồm các bước: Viết test, viết mã nguồn, refactor
- Refactor là hoạt động tái cấu trúc mã nguồn nhằm giúp cho mã nguồn tốt hơn
- JUnit là một Testing Framework dành cho Java