



Automation Test & TDD

Module: Advanced Programming with JavaScript



Mục tiêu

- Trình bày được khái niệm kiểm thử
- Trình bày được khái niệm kiểm thử tự động
- Trình bày được lợi ích của kiểm thử tự động
- Trình bày được các cấp độ trong kiểm thử
- Triển khai được kiểm thử đơn vị
- Triển khai được kiểm thử tích hợp
- Thực thi được các test case
- Trình bày được khái niệm Test-First
- Trình bày được khái niệm TDD



Kiểm thử & Kiểm thử tự động



Kiểm thử là gì?

- Kiểm thử phần mềm là quá trình thực thi 1 chương trình với mục đích tìm ra lỗi.
- Quá trình này đảm bảo sản phẩm phần mềm đáp ứng chính xác, đầy đủ và đúng theo yêu cầu đã đưa ra.
- Kiểm thử phần mềm cũng cung cấp mục tiêu, cái nhìn độc lập về phần mềm, điều này cho phép việc đánh giá và hiểu rõ các rủi ro khi thực thi phần mềm.
- Kiểm thử phần mềm được phân loại thành hai phương pháp:
 - Kiểm thử thủ công (Manual Testing)
 - Kiểm thử tự động (Automated Testing)

Kiểm thử tự động là gì?



- Kiểm thử tự động (automation testing) là việc sử dụng phần mềm đặc biệt (tách biệt với phần mềm đang được kiểm thử) để kiểm soát việc thực hiện các bài kiểm tra và so kết quả thực tế với kết quả dự đoán.
- Tự động kiểm thử có thể tự động hóa một số nhiệm vụ lặp đi lặp lại nhưng cần thiết trong một quá trình thử nghiệm đã được chính thức hóa, hay là các kiểm thử bổ sung nhưng sẽ khó thực hiện thủ công

Lợi ích của kiểm thử tự động



- Giúp tiết kiệm được thời gian và chi phí phát triển hơn so với kiểm thử thủ công
- Giúp hạn chế các sai sót do thao tác thủ công của con người
- Kiểm thử tự động không cần quá nhiều sự can thiệp của con người. Có thể chạy kiểm thử tự động liên tục mà không cần giám sát.
- Giúp tăng tốc độ thực hiện kiểm thử
- Giúp tăng phạm vi kiểm thử

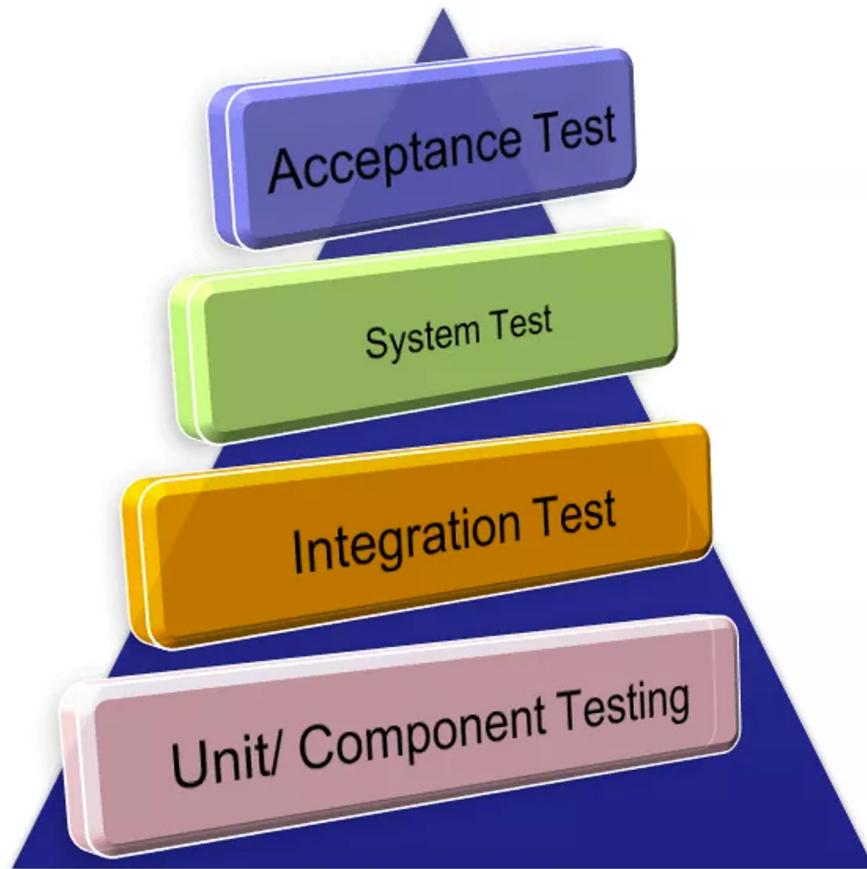
Vai trò lập trình viên trong kiểm thử



Đối với lập trình viên, sau khi xây dựng xong một hàm hoặc một chức năng cụ thể, chúng ta phải kiểm tra lại tính đúng đắn của những dòng mã đã được viết. Có hai cách để thực hiện việc này:

- Kiểm tra thủ công bằng cách chạy lại hàm hoặc chức năng vừa viết.
- Kiểm tra tự động bằng cách xây dựng sẵn một kịch bản thực thi cho hàm, hoặc chức năng.

Các cấp độ trong kiểm thử



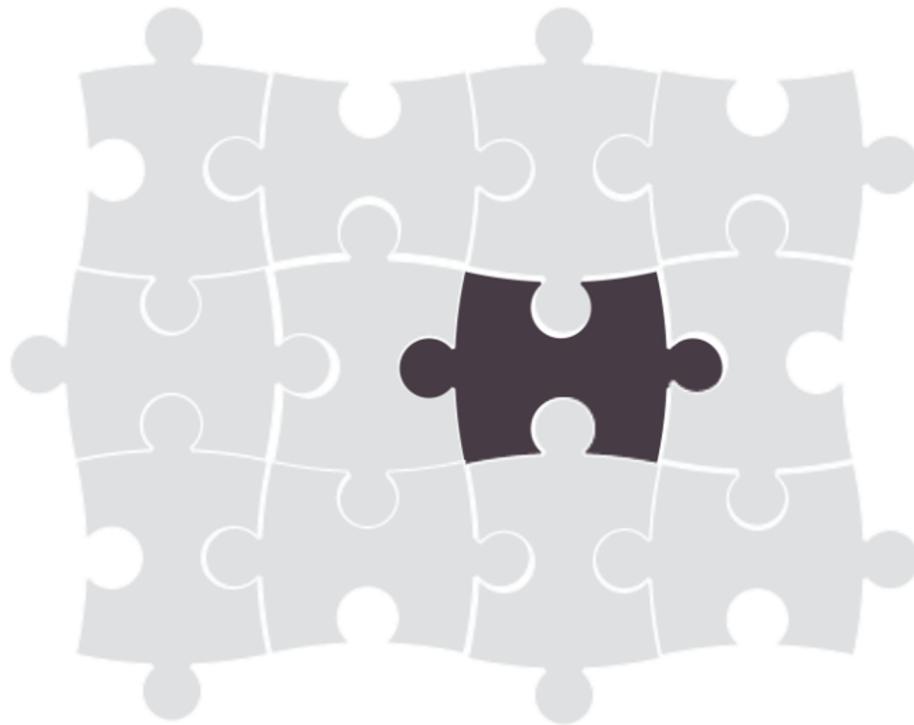


Unit Testing

Unit Testing



- Kiểm thử đơn vị là quá trình kiểm thử từng đơn vị phần mềm xem có hoạt động đúng như thiết kế hay không.





Unit là gì?

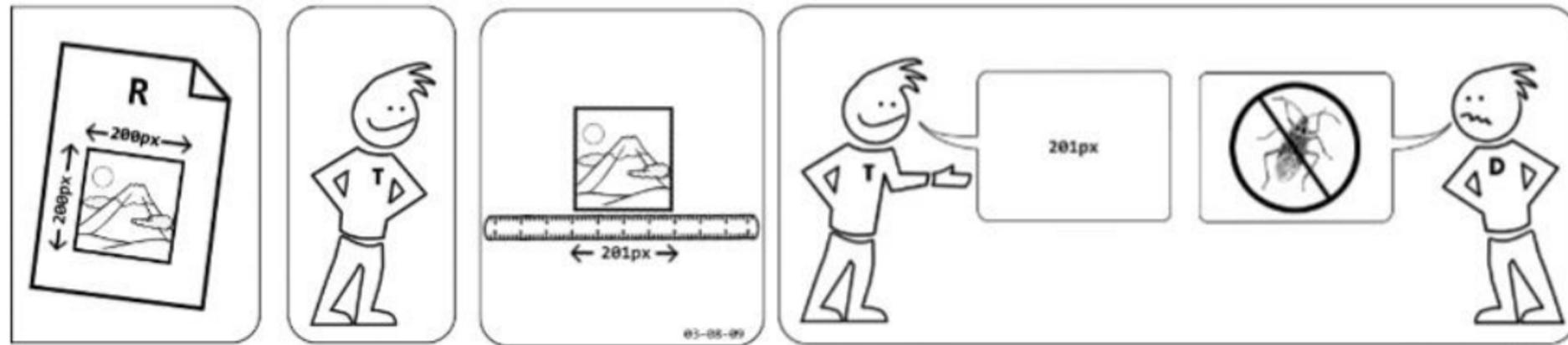
- Một dòng lệnh
- Một hàm, phương thức
- Một module
- Một chương trình



Tại sao phải thực hiện unit testing



- Phát hiện lỗi nhanh
- Cải tiến quy trình và thiết kế sản phẩm
- Giảm chi phí sửa lỗi

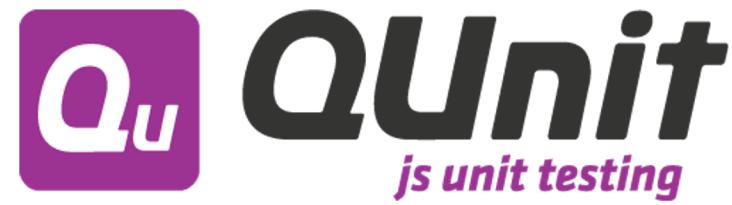


Đặc điểm Unit Test



- Unit Test có các đặc điểm sau:
 - Đóng vai trò như những người sử dụng đầu tiên của hệ thống.
 - Chỉ có giá trị khi chúng có thể phát hiện các vấn đề tiềm ẩn hoặc lỗi kỹ thuật.
- Unit Test có 3 trạng thái cơ bản:
 - Fail (trạng thái lỗi)
 - Ignore (tạm ngừng thực hiện)
 - Pass (trạng thái làm việc)

Một số Framework hỗ trợ kiểm thử





Jest



Tìm hiểu về Jest

- Jest là một trong những trình chạy thử nghiệm phổ biến nhất hiện nay và đang là lựa chọn mặc định cho các dự án React, Jest có các đặc tính sau:
 - **Đơn giản, dễ hiểu:** bạn ko cần phải đi mò giữa nhiều thư viện khác nhau, chỉ lên trang chủ Jest là đủ
 - **Không cần cấu hình gì cả:** vâng, hoàn toàn không. Chỉ cần kéo thư viện về là bạn có thể bắt đầu viết test và test code được rồi
 - **All in one:** một mình Jest là đã cân đầy đủ test runner, assert và mock. Ngoài ra còn có thêm cả Coverage reports....
 - **Nhanh:** phải nói là rất nhanh, ngoài ra terminal của test rất đẹp và thân thiện



Demo

Thực hành Unit test với Jest



Integration Testing

Kiểm thử tích hợp là gì?



- Kiểm thử tích hợp (Integration testing) hay còn gọi là tích hợp và kiểm thử (integration and testing, viết tắt: I&T) là một giai đoạn trong kiểm thử phần mềm.
- Kiểm thử tích hợp xảy ra sau kiểm thử đơn vị (Unit Test) và trước kiểm thử xác nhận.
- Kiểm thử tích hợp nhận các module đầu vào đã được kiểm thử đơn vị, nhóm chúng vào các tập hợp lớn hơn, áp dụng các ca kiểm thử đã được định nghĩa trong kế hoạch kiểm thử tích hợp vào tập hợp đó, và cung cấp đầu ra cho hệ thống tích hợp.



Tại sao cần kiểm thử tích hợp?

- Một module được thiết kế bởi một lập trình viên có hiểu biết và logic lập trình có thể khác với các lập trình viên khác. Kiểm thử tích hợp là cần thiết để đảm bảo tính hợp nhất của phần mềm.
- Tại thời điểm phát triển module, sẽ có nhiều lúc khách hàng đưa ra những thay đổi về yêu cầu, những thay đổi này có thể không được kiểm tra ở giai đoạn unit test trước đó.
- Giao diện và cơ sở dữ liệu của các module có thể chưa hoàn chỉnh khi được ghép lại.
- Khi tích hợp hệ thống các module có thể không tương thích với cấu hình chung của hệ thống.
- Xử lý những trường hợp ngoại lệ một cách không phù hợp có thể gây ra các vấn đề khác



Demo

Thực hành Integration Testing với Jest



Test First & TDD

Test First Development



- TFD – Phát triển kiểm thử đầu tiên tức là phải viết ra các chương trình kiểm thử trước khi viết mã lệnh

Tại sao cần Test-First



Không Test-First	Test-First
Nhận bài toán\vấn đề, cụ thể là chức năng cần viết code.	Nhận bài toán\vấn đề, cụ thể là chức năng cần viết code.
Dùng IDE viết hoàn chỉnh chức năng đó.	Dùng IDE để thiết kế Test-Case cho chức năng đó.
Kiểm thử chức năng bằng cách viết code hoặc tích hợp vào đâu đó rồi chạy thử.	Dùng IDE để viết hoàn chỉnh chức năng. Thường phối hợp nhịp nhàng giữa viết test và viết mã cho chức năng. VD, TDD (Test-Driven Development) là một kỹ thuật triển khai TF với các bước cụ thể.
Dùng hoặc không dùng framework cho Unit Test.	Chắc chắn dùng Framework cho Unit Test.



TDD là gì

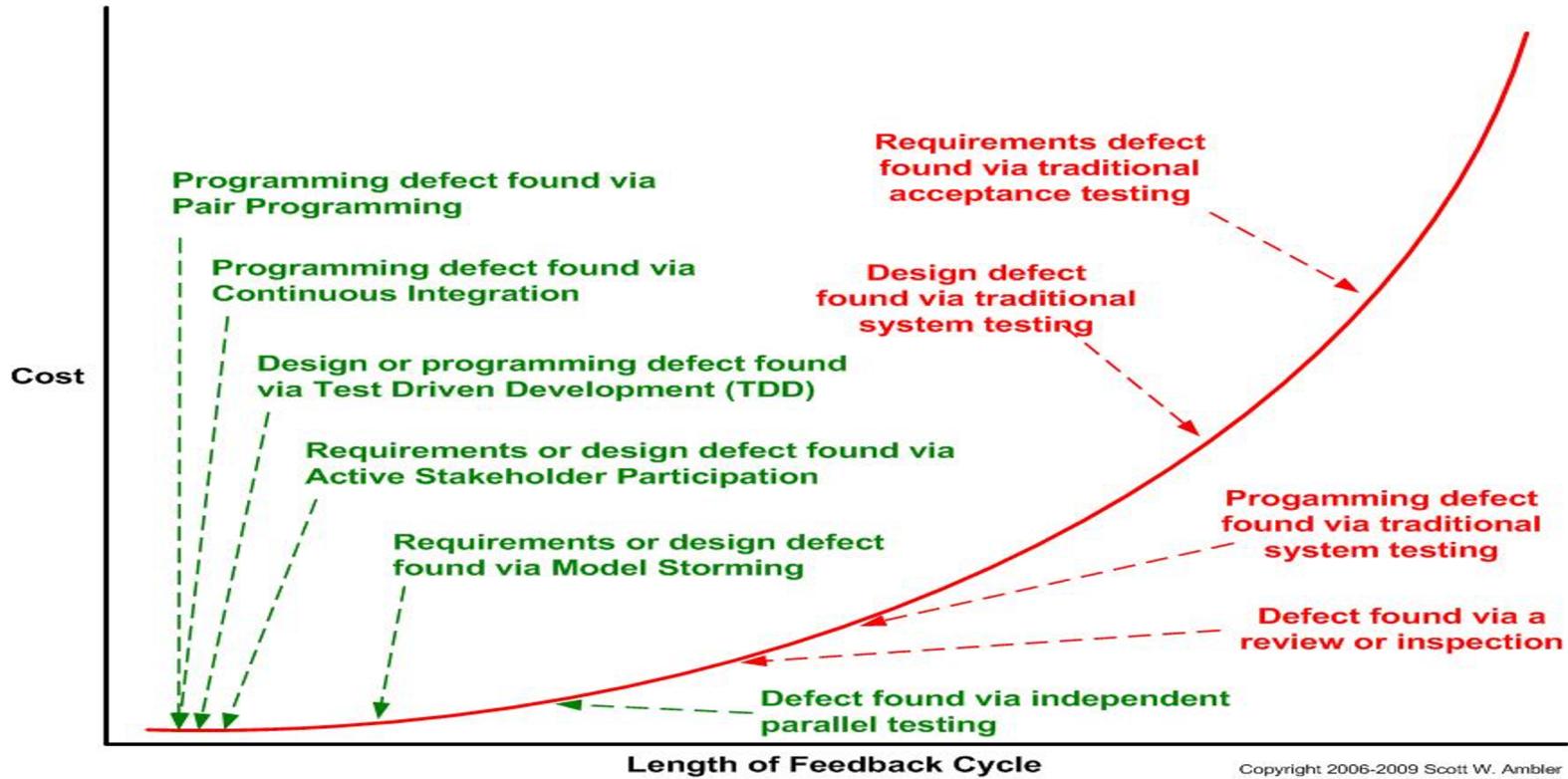
- TDD là kỹ thuật lập trình nhằm giúp nâng cao năng suất và hiệu quả phẩm mềm.
- TDD bao gồm sự kết hợp của hai thành tố: *TFD - Test-First Development* (Phát triển kiểm thử đầu tiên - tức là bạn cần phải viết ra các trường hợp kiểm thử trước khi viết mã lệnh) và *Refactoring* (Tái cấu trúc – thay đổi cấu trúc đoạn mã sau khi các kiểm thử được thực thi để cải tiến đoạn mã tốt hơn).

Tại sao nên sử dụng TDD để phát triển phần mềm

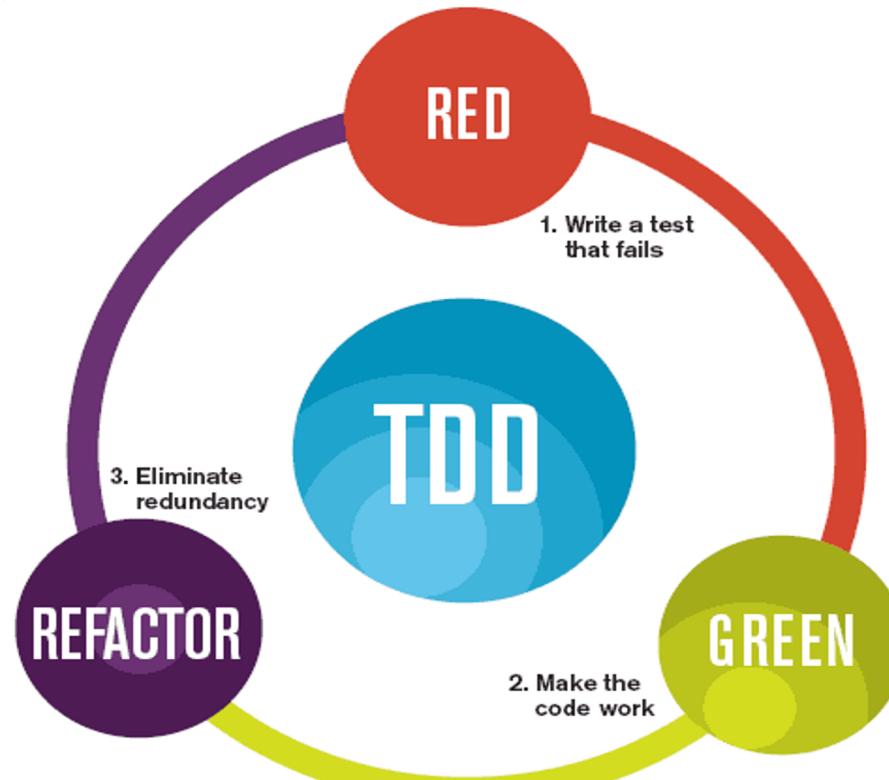


- Hiểu đúng bài toán cần giải quyết
- Hướng vào mục tiêu rõ ràng, tránh được việc viết những đoạn chương trình thừa
- Các thành phần của chương trình làm đến đâu chắc chắn đến đấy do đó khả năng bảo trì, mở rộng và kế thừa cao.
- Làm ví dụ minh họa cho nhà phát triển

Phát triển phần mềm sử dụng TDD so với mô hình truyền thống



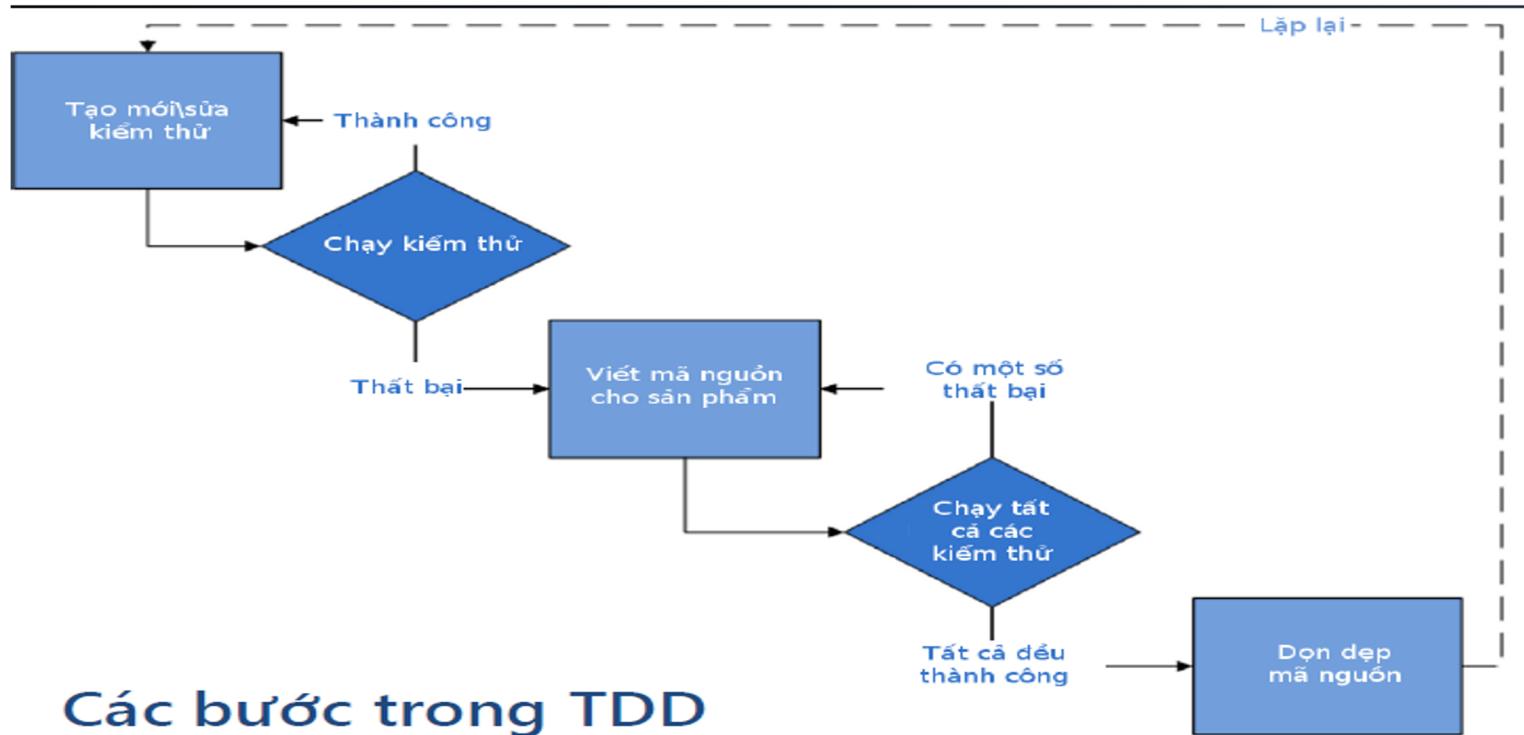
Quy trình làm việc với TDD



The mantra of Test-Driven Development (TDD) is “red, green, refactor.”

Các chiến lược khi viết kiểm thử dựa trên ba màu cơ bản: red (thất bại) – khi kiểm thử gắp lỗi, green (thành công) – khi một kiểm thử thông qua, blue (cải tiến) – cấu trúc lại mã nguồn

Luồng công việc chính trong TDD



Các bước trong TDD

Nguồn ảnh: [Excirial](http://upload.wikimedia.org/wikipedia/en/9/9c/Test-driven_development.PNG) (http://upload.wikimedia.org/wikipedia/en/9/9c/Test-driven_development.PNG)

Bài toán FizzBuzz



- Xây dựng chức năng read (hàm read()) để đọc một số nguyên từ 1 - 100 theo quy tắc FizzBuzz
- Tạo test-first và thực hiện kiểm thử theo hướng dẫn

Tổng kết



Qua bài học này, chúng ta đã đạt được các mục tiêu sau:

- Trình bày được khái niệm kiểm thử
- Trình bày được khái niệm kiểm thử tự động
- Trình bày được lợi ích của kiểm thử tự động
- Trình bày được các cấp độ trong kiểm thử
- Triển khai được kiểm thử đơn vị
- Triển khai được kiểm thử tích hợp
- Thực thi được các test case