[Unit Test](https://techmaster.vn/posts/33618/unit-test-dung-de-lam-gi-va-kinh-nghiem-viet-unit-test-tot-nhat)

Unit test có thể làm tăng đáng kể chất lượng dự án của chúng ta, nhưng với một điều kiện là unit test được viết một cách hợp lý và có vai trò nhất quán, rõ ràng. Nếu không, unit test sẽ lấy mất rất nhiều công sức của team mà không mang lại rất ít giá trị và rất khó để có thể nhận ra mục đích của những method unit test như vậy hoặc có thể đơn giản là nó chả có mục đích gì hết, việc tìm hiểu nó chỉ là phí công mà thôi.

**A. Unit Test không phải là pass hết test case**

Khi tôi bắt đầu viết unit test, hầu như tất cả mọi cố gắng của tôi là làm sao cho tất cả các test case đều xanh. Nhưng mục đích của unit test không phải là như vậy, **hãy viết unit test sao cho mọi việc của bạn là modify source code implement chứ không phải là nỗ lực chỉnh sửa code test**. Nhìn thấy **test case fail mới chính là điều chúng ta mong đợi** từ unit test, điều đó có nghĩa là unit test của chúng ta đã hoạt động và thực sự có ý nghĩa. Chúng ta nên vui mừng vì điều đó, qua unit test fail chúng ta nhận thấy rõ ràng có sự sai sót trong code implement và có thể nhanh chóng fix nó tại bước unit test này.

Còn trường hợp thứ hai là chúng ta nhận được phản hồi về sai sót của chương trình từ người khác. Hãy kiểm tra lại test case của class test liên quan, bổ sung unit test còn thiếu và sau đó là run lại method test với hy vọng là test method sẽ fail. Vì như thế bug của chương trình sẽ được khoanh vùng lại tại method test của unit test thôi và một bug unit test sẽ đơn giản hơn rất nhiều một bug từ intergration test.

=> Màu đỏ trong unit test mới chính là thứ chúng ta mong chờ nhiều nhất từ unit test. Vậy mục tiêu của unit test là gì?

**B. Unit Test dùng để xác định lại requirement và hỗ trợ cho detail design của chương trình**

Có vẻ mục đích tôi đưa ra không liên quan gì nhiều đến từ "unit test", nhưng qua công việc, tôi thấy rằng đây mới chân chính là mục đích chân chính của việc viết unit test. Chúng tôi làm việc theo mô hình TDD, lên unit test luôn được ưu tiên viết trước khi implement code. Điều đó có nghĩa rằng chúng tôi cần hình dung ra code unit test trong quá trình thiết kế source code, chứ không phải trong quá trình test.

Unit test cần được viết base trên các ý có trong spec requirement, từ các phân tích spec, chúng ta define ra các thành phần phần mềm, các class, interface v.v.. cần thiết và các logic tương tác giữa các unit này được chuyển từ mô tả trong spec sang test case method trong unit test.

**1. Unit test mô tả tất cả requirement**

Yêu cầu tối thiểu của unit test là không được để sót các ý có sẵn trong spec. Một khi chúng ta đã đạt được yêu cầu tối thiểu của unit test tức là đã đạt được mục tiêu đầu tiên đó là verify, xác định chính xác source code đã tuân theo requirement được yêu cầu.

Và đương nhiên, unit test cũng cần phải được modify, chỉnh sửa mỗi khi có sự thay đổi của requirement liên quan. Hãy đảm bảo các bug hướng chức năng của chương trình luôn được cập nhật vào code unit test vì khi có bug liên quan đến chức năng chương trình, chúng ta luôn phải confirm lại về spec và việc add thêm hoặc chỉnh sửa các test case liên quan là cần thiết. Một khi unit test không bám sát được spec, thì đó chính là unit test chết, nó sẽ mất dần ý nghĩa tồn tại của chinh bản thân code test.

**2. Unit test & detail design**

Bản thân tôi, khi làm việc theo mô hình TDD, việc viết unit test chính là định hướng căn bản cho việc detail design của chương trình. Tôi sử dụng behavior test case method để xác định khung chương trình trước khi bắt tay vào implement code thực thi.

Việc này có một ưu điểm rất nổi bật đó là giữ cho design luôn luôn đơn giản và dễ hiểu. Vì bạn không thể viết một đoạn mã code unit test lằng nhằng được khi công việc đó luôn được thực thi trước khi viết code implement. => Vậy là bạn luôn luôn phải tuân theo nguyên tắc thứ nhất KISS.

Ưu điểm thứ hai là code luôn luôn bám sát spec, bạn sẽ không viết bất kỳ dòng code nào mà không có yêu cầu được đưa ra. Tại sao vậy? Đơn giản vì unit test dựa trên spec và nó được viết trước code implement -> bạn chẳng thể có cơ hội nào để chèn thêm những thứ nào khác ngoài spec cả. => Bạn đã bị ép buộc tuân theo nguyên tắc thứ hai YAGNI.

Viết method tuân thủ theo Single Responsibility Principle - “Một class - một method chỉ được có 1 nhiệm vụ”, bạn sẽ gặp rất nhiều khó khăn khi viết unit test mà không tuân theo nguyên tắc trên. Vì với unit test, bạn cần viết test cho một logic, một nhiệm vụ duy nhất trong một case. Đừng đi ngược chiều chảy của dòng nước, nếu bạn cố viết những method với hơn 1 nhiệm vụ, bạn sẽ gặp khó khăn rất lớn với unit test của chúng đấy.

=> Theo một cách rất tự nhiên, unit test đã định hướng detail desin của class implement tuân theo một số nguyên tắc cơ bản và khiến code trở lên thông thoáng và dễ hiểu hơn rất nhiều.

**3. Unit test & Maintaince**

Xuất phát từ lý do unit test chính là sự thể hiện trực quan ý hiểu của developer đối với spec => Đọc test case chúng ta có thể hiểu được bối cảnh của logic, các hành vi (behavior) và kêt quả xử lý mong muốn đạt được của logic. Ở bước này, chúng ta cần giữ số lượng test method tối thiểu mà vẫn thể hiện được đầy đủ các unit test, với tên method rõ ràng. Như vậy developer sau có thể dễ dàng chỉnh sửa code test hoặc bổ sung code test do có sự thay đổi từ spec.

Unit test hỗ trợ rất tốt việc refactor source code, vì nó đảm bảo source code sau khi refactor vẫn đáp ứng đủ các yêu cầu requirement đề ra, nếu chúng ta có thể pass tất cả các test case và bộ unit test case là đầy đủ.

Tuy nhiên, unit test không thể trợ giúp các mục đích sau:

**4. Unit test không thể phát hiện hết bug của chương trình**

Vì đơn giản, unit test là các kịch bản, các hành vi của các thành phần lập trình được viết theo spec và theo ý hiểu của người lập trình. Chúng ta không thể mong chờ unit test xác định được hoàn toàn code chạy chính xác hay chưa. Vì unit test chỉ hỗ trợ xác định tính đúng đắn của các unit lập trình một cách độc lập. Khi kết hợp các class, các module lại với nhau, chúng ta có thể gặp rất nhiều các loại lỗi khác uni test không thể bao phủ hết được. Đặc biệt các yêu cầu phi chức năng thì không thể yêu cầu unit test xác minh được.

Với unit test, chúng ta có thể tránh được hầu hết các lỗi đơn giản của chương trình. Nhưng với các lỗi phức tạp khác thì unit test không thể trợ giúp gì cho bạn được.

**5. Pass unit test chưa hẳn code của unit class đã đúng**

Nếu bạn viết sai ngày từ công đoạn viết unit test, thì chắc chắn rằng code của bạn cũng sẽ sai và unit test không thể trợ giúp gì cho bạn trong trường hợp này, Việc đầu tiên bạn phải làm là correct lại code unit test và chỉnh sửa lại code thực thi sau đó. Việc chỉnh sửa này sẽ tránh cho tương lai bạn không còn bị những lỗi tương tự xảy ra nữa. Nhưng một cách khách quan thì unit test bó tay với trường hợp này. => Unit test method cũng cần được review một cách kĩ lưỡng.

**C. Nguyên tắc viết Unit Test**

Sau đây, tôi sẽ đưa ra những nguyên tắc của tôi khi viết unit test, mong các bạn góp ý thêm. Nó trợ giúp tôi rất nhiều khi viết unit test.

**1. Viết unit test theo nguyên tắc Top to Down**

Hãy bắt đầu viết unit test case cho logic chính và bắt đầu với method test cho behavior, hành vi của method trước. Ví dụ chúng ta có spec như sau: Chức năng Login, người dùng nhập user và password, Nếu đúng redirect tới trang quản lý với user là staff, trang home với user là customer. Nếu sai thì show error message.

Chúng ta bắt đầu với unit test cho case main follow 1: redirectOtherPageWhenPassAuthenticate trong method này chúng ta có 2 behavior nhất định phải qua là: authenticateUser(account, password), (mock method với giả thiết method trả về class User và isAuthenticate = true) và sau đó là redirectPage(user). Chúng ta chưa cần quan tâm đến authenticateUser và redirectPage cần implement như thế nào, chỉ đơn giản là viết các ý theo spec yêu cầu.

case thứ 2 là: showErrorWhenFailAuthenticate, có 2 method: authenticateUser(account, password), (mock method với giả thiết method trả về class User và isAuthenticate = false) và sau đó là showErrorLoginMessage method.

Tiếp theo chúng ta đọc tiếp spec ở các mục nhỏ hơn và dần hoàn thiện unit test code. Ưu điểm của phương pháp này là code chúng ta implement theo hướng tự nhiên nhất có thể, luôn bám sát vào spec. Cả code unit test và code thực thi sẽ dần được hoàn thiện theo cấu trúc của spec.

**2. Viết unit test theo đúng nghĩa unit test (độc lập và không phụ thuộc vào các thành phần lập trình khác.)**

**Với mỗi một ý có trong spec, bạn chỉ nên viết một và chỉ một test case** và luôn tuân theo một chiều duy nhất, viết cho main logic trên cùng và xuống đần cho các sub logic. Nguyên tắc này giúp tôi tránh được việc lặp lại test case một cách vô ích. Để giúp người khác có thể hiểu được bạn đang làm gì thì việc định hướng cách viết nhất quán là một điều quan trọng. Nếu bạn viết thừa test case, sẽ không ai cảm ơn bạn đâu, vì một ai đó sẽ phải phân vân không hiểu tại sao lại có thêm 1 test case nữa và mục đích của nó là gì.

**Viết unit test code cho chỉ một unit tại một thời điểm**. Chúng ta cần viết unit test theo class, một class test cho một class thực thi và khi class có nhiều method, chúng ta cần nhóm các method test cho cùng một method con vào nested class của chính class test đó. Bằng cách này chúng ta có thể quản lý test case hiệu quả hơn. Nhất là với các class support và common, chúng có thể bao gồm rất nhiều method.

**Giả lập (mock hoặc fake) tất cả các method, logic được call từ các thành phần lập trình khác** Làm vậy để tránh các logic test ánh hưởng đến nhau, data từ các thành phần unit khác sẽ tác động đến nhiều test case => Một khi có một test case bị sai logic, chúng ta dễ dàng fix được chúng mà không bị báo đỏ ở quá nhiều method test khác. Nếu không, việc thay đổi một unit có thể ảnh hưởng ra bên ngoài và gây ra fail test ở khắp mọi nơi.

**3. Đặt tên các unit test của bạn một cách rõ ràng và nhất quán**

Chắc chắn việc đặt tên rõ ràng và nhất quán là rất quan trọng và cần thiết để cho mọi người có thể hiểu nhanh hơn và rõ hơn về logic bạn định viết.

**Unit test cho behavior logic không quan trọng đến giá trị trả về** Không giống như assertion test, behavior test không quan trọng đến kết quả chúng ta mong muốn nhận được hay không mà nó như một tấm bản đồ chỉ đường cho ta biết lộ trình của logic cần đi.