**1. Overview: (1 tiếng)**

* **Unit test là gì:**
* Unit Test là một kiểu kiểm thử phần mềm trong đó các đơn vị hoặc thành phần riêng lẻ của một phần mềm được kiểm tra.
* Mục đích: xác nhận rằng mỗi đơn vị của mã phần mềm hoạt động như mong đợi.
* Được thực hiện trong quá trình phát triển (giai đoạn viết mã) của một ứng dụng bởi dev.
* Tách một phần mã và xác minh tính đúng đắn của nó. Một unit có thể là một chức năng, phương pháp, thủ tục, mô-đun hoặc đối tượng riêng lẻ, hàm (Function), thủ tục (Procedure), lớp (Class), hoặc các phương thức (Method).
* **Nếu không sử dụng unit test thì sao ==> Tại sao phải sử dụng unit test:**
* Thực tế không phải chỉ riêng mỗi nghề viết code mới có sử dụng Unit test, các nghề khác như siêu thị chẳng hạn, việc làm này là thường xuyên mỗi ngày, thậm chí mỗi giờ.

**Ví dụ**: check xem trứng có hư chưa, sữa có tươi, rau có thối không...vv

=> Chính vì vậy có thể kết luận luôn là Unit test là rất quan trọng.

Đương nhiên là cần thiết với một Lập Trình Viên có trách nhiệm với những dòng code của chính bản thân mình.

—-------------------------------—-----------------------------------------------------------------------------------

**Trong một ví dụ cụ thể:**

Khi chúng ta có một hàm cần được thay thế, nhưng chúng ta cần đảm bảo cái hàm mới chúng ta viết phải cho ra kết quả giống như hàm ban đầu, thì trong trường hợp này chúng ta cần unit test để kiểm tra, lúc này chúng ta sẽ nhanh chóng phát hiện ra những chỗ không đúng một cách nhanh chóng.

**Nhưng** **nếu trong trường hợp nếu không có unit test thì sao?**: chúng ta phải tự kiểm tra, điều này thì không đảm bảo được hàm mới sẽ cho ra kết quả giống như hàm ban đầu, giả sử như người dev có thể kiểm tra được hết thì việc này cũng mất rất nhiều thời gian không đáng có, gây nên lãng phí.

Trong quá trình code không tránh khỏi những lỗi, hoặc trong quá trình chỉnh sửa thêm tính năng khác cần:

**Đảm bảo chất lượng code:**

* Việc viết test trước khi code sẽ khiến bạn phải suy nghĩ về nhiều khía cạnh của chức năng giúp giảm thiểu các lỗi và nâng cao chất lượng code, phát hiện các thuật toán thực thi không hiệu quả, chạy vượt quá giới hạn thời gian.

**Phát hiện lỗi sớm:**

* Khi viết ra một chức năng đầy đủ và bắt đầu kiểm tra nó thì xảy ra lỗi, khi đó bạn sẽ phải lật lại từ đầu debug từng đoạn để tìm xem lỗi từ đâu.
* Công đoạn đó thực sự rất tốn thời gian và công sức. Trong khi đó nếu bạn viết Unit Test trước khi bắt đầu code thì mọi lỗi sẽ được phát hiện và kiểm soát từ ngay bước đầu.

**Dễ dàng bảo trì và thay đổi:**

* Áp dụng thư viên mới thì dễ nhưng để đảm bảo tất cả các chức năng cũ vẫn chạy ngon lành thì không phải đơn giản. Tuy nhiên Unit test sinh ra để giúp bạn làm việc đó.
* Công việc của bạn chỉ cần chạy lại unit test cho từng chức năng và kiểm tra.

**Cung cấp tài liệu rõ ràng:**

* Việc viết Unit Test không những giúp đảm bảo chất lượng của từng chức năng mà nó còn là một tài liệu rất rõ ràng mô tả chức năng đó với nhiệm vụ gì.

**Giảm chi phí sửa lỗi.**

**Tăng sự tự tin:**

* Khi hoàn thành một công việc, chúng ta thường có cảm giác không chắc chắn về các đoạn mã của mình như liệu các lỗi có quay lại không, hoạt động của module hiện hành có bị tác động không, hoặc liệu công việc hiệu chỉnh mã có gây hư hỏng đâu đó…

**2. Chiến lược triển khai:**

Khi có quá nhiều function trong một hàm:

*func big\_func() {*

*a = f\_small\_1()*

*if a {*

*f\_small\_2()*

*} else {*

*f\_small\_3()*

*}*

*f\_small\_4()*

*...*

*f\_small\_7()*

*}*

Chúng ta có thể triển khai bằng nhiều phương án:

* Top down: kiểm tra từ trên xuống và các hàm nhỏ hơn sẽ được thay thế bằng các giá trị "giả" hay còn được gọi là stub để kiểm tra tính đúng đắn của big\_function trước, sau rồi sau đó thì sẽ đến các cấp tiếp theo.
* Bottom Up: ta sẽ kiểm tra từ dưới lên, có nghĩa là ta sẽ kiểm tra tính đúng đắn của các hàm nhỏ hơn trước, ví dụ f\_small\_4,.., rồi sau đó nếu đúng sẽ được thay thế vào vào hàm big\_func để kiểm tra.

**3. Cách sử dụng code hiệu quả trong unit test**

* **Một số khái niệm cần thiết trong unit test:**
* Test case: mỗi một function thì cần nhiều testcase, ứng với mỗi trường hợp testcase chạy.
* Setup: Khai báo dữ liệu, cài đặt các biến.
* Teardown: Dùng để xóa dữ liệu, giải phóng bộ nhớ.
* Assert: Kiểm tra tính đúng đắn của hàm.
* Mock: Giả sử chương trình của bạn được chia làm 2 module: A và B. Module A đã code xong, B thì chưa. Để test module A, ta dùng mock để làm giả module B, không cần phải đợi tới khi module B code xong mới test được.
* **Các bước thiết lập một unit test:**
* Thiết lập các điều kiện cần thiết bao gồm khởi tạo các đối tượng, xác định tài nguyên cần thiết, xây dựng các dữ liệu giả.
* Gọi các phương thức cần kiểm tra.
* Kiểm tra sự hoạt động đúng đắn của các phương thức.
* Dọn dẹp tài nguyên sau khi kết thúc kiểm tra.
* **Một số lưu ý khi viết Unit Test:**
* Code Unit Test phải được viết ngắn gọn, dễ hiểu, dễ đọc.
* Unit Test cần phải nhanh, vì Unit Test sẽ được chạy để kiểm định lỗi mỗi lần build. Do đó, trong Unit Test phải hạn chế các nhiệm vụ tốn thời gian như gọi I/O, database, network,…
* Luôn đảm bảo lỗi phải được xác định trong quá trình Unit Test được sửa trước khi chuyển sang giai đoạn tiếp theo.
* Không nên có quá nhiều assert trong một test case vì khi một điều kiện không thỏa mãn, các assert khác sẽ bị bỏ qua.
* Mọi Unit Test phải bắt đầu với trạng thái “fail” và chuyển trạng thái “pass” sau một số thay đổi hợp lý đối với mã chính.
* Mỗi khi viết một đoạn mã quan trọng, hãy viết các Unit Test tương ứng cho đến khi không thể nghĩ thêm tình huống nào nữa.

**4. Auto vs manual:**

* **Khi nào sử dụng auto/manual**
* Manual testing: là việc tiến hành kiểm thử phần mềm mà các test case được thực hiện thủ công bởi người kiểm thử (QA/QC/Tester) mà không sử dụng bất kì công cụ tự động nào hỗ trợ.
* Automation testing: là quá trình kiểm thử phần mềm thông qua việc sử dụng các công cụ tự động để chạy các test scripts.
* **Những khác biệt chính giữa Manual testing và Automation testing**

| **Yếu tố** | **Manual testing** | **Automation testing** |
| --- | --- | --- |
| Chi phí thời gian | Tốn thời gian để thực hiện kiểm thử | Thời gian kiểm thử rút ngắn. Đặc biệt có thể chạy daily tự động trên CI, không cần người khởi chạy. |
| Giao diện thay đổi | Thay đổi nhỏ về không ảnh hưởng đến kịch bản test | Dễ bị ảnh hưởng bởi source code sản phẩm |
| Kiến thức về lập trình | Không đòi hỏi người kiểm thử có kiến thức về lập trình | Người kiểm thử cần có kiến thức về lập trình để hiện thực test scripts |
| Kiểm thử khám phá | Dễ dàng triển khai Exploratory testing | Không thể thực hiện Exploratory testing |
| Thực thi song song | Khi cần kiểm thử trên nhiều môi trường cần tăng nguồn lực QA lên để thực hiện | Dễ dàng thực thi song song trên nhiều môi trường nhờ vào sự hỗ trợ của các công cụ tự động |
| Khi nào áp dụng? | Dự án nhỏ, thời gian thực hiện ngắn hay các test case chỉ đòi hỏi chạy ít lần | Dự án lớn hay các test case đòi hỏi chạy đi chạy lại nhiều lần |

* **Auto sử dụng những thư viện/framework nào**

*Framework:*

* Các automated testing framework là một tập hợp các thành phần hỗ trợ thực hiện các thử nghiệm và báo cáo toàn diện về kết quả thử nghiệm.
* Các framework dành cho kiểm thử tự động thường gặp nhất:
* Jest: Jest được Facebook sử dụng cho đến nay để kiểm thử tất cả các mã nguồn JavaScript. Hỗ trợ kiểm thử chạy độc lập và không gián đoạn mà không có bất kỳ xung đột nào.
* Mocha: Điểm mạnh là tính đơn giản. Hỗ trợ các tính năng phong phú như thời gian chờ dành riêng cho kiểm thử, JavaScript API,…
* Jasmine: Là Framework phát triển theo hướng hành vi để kiểm thử đơn vị trong JavaScript. Nó được sử dụng để kiểm tra mã nguồn JavaScript đồng bộ và bất đồng bộ.
* Storybook: là một môi trường giúp chúng ta có thể dễ dàng dev và test UI. Với nó chúng ta có thể test UI với những state ảo và test xem UI sẽ hoạt động ra sao.
* Cypress: là front end testing tool được xây dựng cho các ứng dụng Web hiện đại. Đây là một công cụ hỗ trợ hữu hiệu cho developers và QA trong kiểm thử ứng dụng Web hiện đại.
* Puppeteer: là một thư viện NodeJS cung cấp các high-level API để sử dụng cho các ứng dụng dành riêng cho trình duyệt như kiểm tra thu thập dữ liệu, testing UI, xem các vấn đề về hiệu suất, …
* React Testing Library: là một tập hợp các trình trợ giúp cho phép bạn kiểm tra các thành phần React mà không cần dựa vào chi tiết triển khai của chúng.
* WebdriverIO cho phép bạn tự động hóa bất kỳ ứng dụng nào được viết bằng các khuôn khổ web hiện đại như React, Angular, Polymeror, Vue.js cũng như các ứng dụng di động gốc dành cho Android và iOS.

**5. Chọn thư viện để thực hành xây dựng demo:**

* **Dựa trên điều kiện của project:**
* Độ phổ biến của thư viện (tham khảo khi cần thiết).
* Dễ bảo trì cần biết một số thông tin để chuẩn bị khi sau này thư viện có thay đổi, khi thư viện gốc thay đổi thì thư viện xây dựng trên nó cũng bị thay đổi:
* Thư viện WebdriverIO được xây dựng dựa trên Mocha và Jasmine.
* Thư viện Puppeteer được xây dựng dựa trên Jest.
* Một số thư viện được dùng nhiều cho việc test thiết kế
* Thư viện Storybook là một công cụ thiết kế và phát triển những UI Components cho ứng dụng của bạn trên một môi trường hoàn toàn biệt lập.
* React Testing Library là một tập hợp các trình trợ giúp cho phép bạn kiểm tra các thành phần React

**6. Nhược điểm:**

* Không thể được mong đợi Unit Test bắt hết mọi lỗi trong một chương trình. Bản chất của Unit Test là tập trung vào một đơn vị mã. Do đó, nó không thể bắt các lỗi tích hợp hoặc lỗi cấp hệ thống rộng.
* Không có thời gian cho Unit Test. Viết Unit Test là tốn thời gian đó là lý do tại sao rất khó để đáp ứng thời hạn.
* Đối với một số trường hợp Unit Test rất khó viết (trong các hệ thống lớn, hay trong phần gia diện).

**7. References:**

1. <https://www.niemvuilaptrinh.com/article/Unit-Test-La-Gi-10-Frameworks-Unit-Test-Cho-Javascript2020>
2. <https://topdev.vn/blog/unit-test-la-gi/>
3. <https://viblo.asia/p/cac-test-automation-framework-pho-bien-nhat-hien-nay-va-uu-nhuoc-diem-cua-tung-loai-4dbZNpkQ5YM>
4. <https://www.guru99.com/unit-testing-guide.html>
5. <https://co-well.vn/nhat-ky-cong-nghe/unit-testing-la-gi-va-vai-tro-trong-kiem-thu-phan-mem/#:~:text=M%E1%BB%99t%20quy%20tr%C3%ACnh%20Unit%20Testing,code%20s%E1%BA%BD%20%C4%91%C6%B0%E1%BB%A3c%20th%E1%BB%AD%20nghi%E1%BB%87m>.
6. <https://wiki.tino.org/unit-test-la-gi/>
7. <https://viblo.asia/p/unit-test-co-ban-jvElaGGoKkw>
8. <https://toidicodedao.com/2015/04/02/viet-unit-test-cho-javascript-voi-jasmine/#:~:text=Unit%20test%3A%20%C4%90%C3%A2y%20l%C3%A0%20test,sau%20khi%20ta%20s%E1%BB%ADa%20code>.
9. <https://tech.cybozu.vn/test-spec-noi-the-hien-kich-ban-test-addf4/>
10. <https://tech.cybozu.vn/page-object-model-lua-chon-de-test-automation-di-duong-dai-a0fcb/>
11. <https://tech.cybozu.vn/automation-testing-va-manual-testing-diem-manh-va-diem-yeu-tung-loai-773d1/>