Chapter 12: Testing

Trước khi bạn lướt qua phần này và nói rằng “tôi không phải là một tester”, hãy dừng lại một chút và suy nghĩ. Testing là một khâu quan trọng đối với một kĩ sư phần mềm, bởi lí do này nên những câu hỏi liên quan đến testing có thể xuất hiện trong cuộc phỏng vấn của bạn. Dĩ nhiên, nếu bạn ứng cử bản thân cho vị trí Tester thì đây là lúc bạn nên chú ý đó!

Những vấn đề của Testing thường rơi vào 1 trong 4 hạng mục: (1) Test 1 vật thể thực tại (ví dụ như 1 cây bút bi); (2) Test 1 phần của phần mềm; (3) Viết test code cho một chức năng; (4) Sửa chữa một vấn đề đang hiện hữu. Chúng ta sẽ cùng tiếp cận từng vấn đề trong 4 loại nêu trên.

Hãy nhớ rằng 4 loại nêu trên yêu cầu bạn đừng cho rằng input hay user sẽ hoạt động tốt. Hãy trông chờ vào những lỗi có thể xuất hiện và lên kế hoạch cho chúng.

**Người phỏng vấn đang tìm kiếm những gì từ bạn?**

Bề ngoài, những câu hỏi testing dường như chỉ là những danh sách test case phổ thông, và đúng rồi đó thêm một chút mở rộng nữa! Bạn cần phải có một danh sách test case hợp lý.

Hơn nữa, người phỏng vấn muốn kiểm tra những điều sau:

* Big Picture Understanding (hiểu biết về bức tranh toàn cảnh): Bạn có phải là một người hiểu rõ thực sự phần mềm là gì không? Bạn có thể ưu tiên những test case đúng cách? Ví dụ, giả sử bạn được yêu cầu kiểm thử hệ thống e-commerce như Amazon. Thật tuyệt khi chắc rằng những hình ảnh của sản phẩm xuất hiện đúng chỗ, nhưng thậm chí quan trọng hơn là việc thanh toán hoạt động tin một cách đáng tin cậy, những sản phẩm được thêm vào hàng đợi vận chuyển và những khách hàng sẽ không bao giờ bị tính phí gấp đôi.
* Knowing How the Pieces Fit Together (Hiểu biết về những mảnh ghép sẽ chắp nối với nhau như thế nào): Bạn có hiểu cách hoạt động của phần mềm và nó có thể sẽ tương thích với một hệ sinh thái lớn hơn? Giả sử bạn được yêu cầu test Google Spreadsheets. Việc quan trọng là bạn test phần mở đầu, lưu trữ và chỉnh sửa tài liệu. Nhưng Google Spreadsheets là một phần của 1 hệ sinh thái lớn hơn. Bạn cần phải test sự tương tác với Gmail, với plug-ins và một số các thành phần khác.
* Organization (Tổ chức): Bạn có tiếp cận vấn đề bằng một cách thức có cấu trúc hay bạn chỉ trút ra những gì nghĩ ra trong đầu? Một số ứng viên khi được yêu cầu đưa ra những test case cho 1 camera sẽ chỉ viết ra bất kì điều gì và tất cả những gì họ có thể nghĩ ra. Một ứng viên giỏi sẽ chia các phần ra thành các hạng mục như Chụp hình, Quản lí hình ảnh, Cài đặt,… Cách tiếp cận có cấu trúc này cũng sẽ giúp bạn làm một cách triệt để hơn với việc tạo ra những test case.
* Practicality(Tính thực tiễn): Bạn có thể thực sự tạo ra một kế hoạch testing hợp lý? Ví dụ, nếu 1 user báo rằng phần mềm bị hỏng khi họ mở một bức ảnh nhất định và bạn chỉ nói với họ rằng hãy cài đặt lại phần mềm, nó thường là không thực tế cho lắm. Kế hoạch testing của bạn phải khả thi và thực tế cho công ty có thể triển khai.

Chứng minh những khía cạnh trên sẽ thế hiện rằng bạn sẽ là một thành viên giá trị của một team testing.

**Testing một vật thể trong thế giới thực**

Một số ứng viên bị bất ngờ khi được hỏi rằng test một cây bút mực như thế nào. Sau tất cả thì bạn sẽ phải testing phần mềm đúng không? Có thể lắm, nhưng những câu hỏi “đời thực” vẫn rất phổ biến. Hãy cùng giải quyết nó với một ví dụ sau.

Câu hỏi: Bạn test 1 cái kẹp giấy như thế nào?

Bước 1: Ai sẽ sử dụng nó? Tại sao?

Bạn cần phải thảo luận với người phỏng vấn rằng ai đang sử dụng sản phẩm đó và mục đích của họ là gì. Câu trả lời có thể không giống như những gì bạn nghĩ. Câu trả lời có thể là “những giáo viên, dùng để kẹp giữ giấy tờ với nhau”, hoặc cũng có thể “những nghệ sĩ, dùng để bẻ thành hình dáng của con vật” hoặc là cả hai. Câu trả lời của câu hỏi này sẽ định hình cách trả lời của bạn cho những câu sau.

Bước 2: Những trường hợp sử dụng (use case)?

Sẽ rất hữu ích nếu bạn tạo một danh sách use case. Trong trường hợp này, use case có thể là đơn giản giữ các tờ giấy với nhau mà không làm hư giấy.

Với một số câu hỏi khác, có thể sẽ có nhiều use case. Ví dụ như là sản phẩm đó cần có khả năng gửi và nhận nội dung hoặc viết hoặc tẩy xóa,…

Step 3: Việc sử dụng nằm trong khuôn khổ nào?

Khuôn khổ sử dụng nghĩa là giữ một lúc 30 tờ giấy mà không gây tổn hại (ví dụ như bị bẻ cong) và 50 tờ giấy với việc bị cong 1 phần nhỏ vĩnh viễn.

Khuôn khổ cũng được mở rộng ra các nhân tố môi trường. Ví dụ, cái kẹp giấy có thể sử dụng trong nhiệt độ ấm (90 – 110 độ F)? Còn nếu rất lạnh thì thế nào?

Bước 4: Những điều kiện thất bại?

Không có sản phẩm “không thể hỏng”, vì vậy việc phân tích các điều kiện dẫn tới hỏng hóc là một phần thiết yếu của việc testing. Sẽ tốt hơn nếu bạn có thể thảo luận với người phỏng vấn rằng khi nào sự hỏng hóc đó có thể chấp nhận được (thậm chí là cần thiết) và sự hỏng hóc đó có ý nghĩa gì.

Ví dụ, nếu bạn đang test một cái máy giặt, có thể bạn quyết định rằng cái máy đó có thể xử lý ít nhất 30 cái áo hoặc quần. 30-45 cái có thể dẫn đến 1 số hệ quả nhỏ như là quần áo không được giặt sạch. Trong trường hợp nhiều hơn 45 cái áo quần, hệ quả cực lớn có thể được chấp nhận. Tuy nhiên, hệ quả cực lớn trong trường hợp này chỉ nên có thể là máy giặt không cung cấp nước chứ không phải là “lửa hay lụt”.

Step 5: Bạn sẽ thực hiện việc testing như thế nào?

Trong một số trường hợp, nó cũng có thể liên quan đến việc thảo luận về chi tiết việc thực hiện testing. Ví dụ, nếu bạn có thể đảm bảo rằng một cái ghế có thể chịu được việc sử dụng bình thường trong 5 năm, ắt hẳn rằng bạn không thể để nó trong nhà và chờ 5 năm sau. Thay vào đó, bạn sẽ cần phải tính toán việc sử dụng “bình thường” là gì (Sẽ có bao nhiều lần “ngồi” trên cái ghế đó hàng năm? Tay vịn thì sao?). Sau đó, để có thế testing bằng tay, bạn sẽ muốn có một cái máy tự động làm những điều đó.

**Testing một mảnh của phần mềm**

Thực ra việc testing một mảnh của phần mềm rất giống với testing một thực thể. Điều khác biệt lớn đó là testing phần mềm thường nhấn mạnh vào các chi tiết việc thực hiện testing.

Ghi nhớ rằng testing phần mềm có 2 mặt trọng tâm:

* Testing thủ công và tự động (Manual vs. Automated Testing): Trong một thế giới lý tưởng, chúng ta có thể thích mọi thứ đều tự động, nhưng mức độ khả thi của nó thì hiếm lắm. Một số thứ đơn giản là sẽ hiệu quả hơn với testing thủ công vì những đặc tính của nó quá là cảm tính để một máy tính có thể kiểm tra được hiệu quả (ví dụ như những nội dung đại diện cho nội dung khiêu dâm ). Hơn nữa, trong khi một máy tính thường chỉ có thể xác định những vấn đề đã được dạy, việc quan sát của con người có thể tiết lộ ra những vấn đề mới chưa được kiểm tra một cách đặc biệt. Cả con người và máy tính định hình một phần thiết yếu trong quá trình testing.
* Testing hộp đen vs. Testing hộp trắng (Black Box Testing vs. White Box Testing): Sự phân biệt này dựa vào mức độ can thiệp vào phần mềm. Với testing hộp đen, ta chỉ được nhận các lỗi tiềm ẩn của phần mềm và test nó. Với hộp trắng testing, ta có thêm quyền can thiệp vào chương trình để test một số chức năng độc lập. Ta cũng có thể cho chạy tự động hộp đen mặc dù chắc chắn việc đó khó hơn nhiều.

Chúng ta hãy cùng tóm lược lại nhé.

*Bước 1: Chúng ta đang sử dụng Black Box hay White Box testing?*

Mặc dù câu hỏi này có thể bị hoãn lại tới các bước sau nhưng tôi thích loại bỏ chúng ra sớm hơn. Hãy chắc rằng bạn đang làm việc với black box hay white box testing, hoặc cả hai.

*Bước 2: Ai sẽ sử dụng nó, tại sao?*

Phần mềm thường có một hoặc nhiều người dùng chiến lược và những đặc tính được thiết kế theo họ. Ví dụ, nếu bạn được yêu cầu test phần mềm phụ huynh quản lý trên một trình duyệt web, người dùng chính của bạn bao gồm phụ huynh (người chặn) và trẻ em (người bị chặn). Bạn có thể có “khách” (người mà không nên bị chặn hay được chặn).

*Step 3: Những trường hợp sử dụng?*

Trong trường hợp phần mềm chặn trẻ em, use cases của phụ huynh bao gồm cài đặt phần mềm, cập nhật điều khiển, xóa điều khiển và tất nhiên là việc sử dụng internet riêng tư của họ. Đối với trẻ em, uses case bao gồm truy cập vào những nội dung hợp lệ cũng như “không hợp lệ”.

Hãy nhớ rằng không phải một điều kỳ diệu nào đó mà bạn quyết định các use cases, mà đó là một cuộc đối thoại giữa bạn với người phỏng vấn.

*Bước 4: Phạm vi sử dụng?*

Bây giờ chúng ta đã có định nghĩa mơ hồ về use cases, ta cần phải tìm ra ý nghĩa của nó. 1 website bị chặn là như thế nào? Có phải chỉ trang “không hợp lệ” bị chặn, hay là cả website? Có phải thiết bị được cho là “học” như thế nào là nội dung xấu, hay nó chỉ dựa trên danh sách trắng hoặc đen? Nếu nó được học những nội dung không phù hợp, mức độ không phù hợp tiêu cực hay tích cực như thế nào là có thể chấp nhận?

*Bước 5: Những điều kiện thất bại, xảy ra lỗi*

Khi một phần mềm hỏng – một điều không tránh khỏi – lúc bấy giờ trông nó sẽ như thế nào? Rõ ràng, lỗi của phần mềm không nào phá hỏng cả máy tính. Thay vào đó, nó nên chỉ là cho phép một trang bị chặn, hoặc chặn một trang được phép. Trong trường hợp sau này, bạn có thể sẽ muốn bàn bạc về khả năng ghi đè với một mật khẩu từ phụ huynh.

Bước 6: Test cases là gì? Bạn thực hiện việc testing như thế nào?

Đây là nơi mà sự phân biệt giữa testing thủ công và tự động, giữa black box và white box testing thực sự xảy ra.

Bước 3 và bước 4 đã gần như định nghĩa use cases. Ở bước 6, ta đã định nghĩa xa hơn và tranh luận làm thế nào để thực hiện testing. Bạn đang testing những tình huống cụ thể nào? Những bước nào trong đây có thể tự động hóa? Bước nào cần sự can thiệp của con người?

Hãy nhớ rằng trong khi tự động hóa cho phép bạn làm rất nhiều việc, nó vẫn có một số hạn chế nhất định. Testing thủ công thường trở thành 1 phần trong phương pháp test.

Khi bạn đi qua cái danh sách này, đừng kể liền một hơi về những viễn cảnh bạn có thể nghĩ ra. Nó thật sự là không có tổ chức và chắc hẳn bạn sẽ bỏ lỡ những hạng mục quan trọng. Thay vào đó, tiếp cận nó với một phương thức có tổ chức. Chia phần testing của bạn ra các thành phần chính và bắt đầu từ đó. Điều đó sẽ không chỉ giúp bạn có một danh sách test cases hoàn thiện hơn mà nó còn thể hiện bạn là một người có tổ chức, một người làm việc có phương pháp!

**Kiểm tra một chức năng (Testing a Function)**

Trong nhiều trường hợp, testing một chức năng là kiểu testing dễ dàng nhất. Cuộc đối thoại thường sẽ ngắn gọn hơn và ít mơ hồ vì đối với việc testing thường bị hạn chế về mặt xác thực input và output.

Tuy nhiên, đừng bỏ lơ giá trị của một số cuộc đối thoại với người phỏng vấn bạn. Bạn nên tranh luận bất kỳ tình huống giả định nào với họ, đặc biệt là bạn nên xử lý những tình huống cụ thể với sự tôn trọng.

Giả sử rằng bạn được yêu cầu viết code để test sort(int[] array) tức sắp xếp một mảng các số nguyên. Bạn có thể thực hiện theo dưới các bước sau.

*Bước 1: Định nghĩa test case*

Thông thường, bạn nên nghĩ về những loại test case sau:

* Case thông thường: Nó có tạo ra output đúng với các input đưa vào không? Hãy nhớ rằng có những vấn đề tiềm ẩn. Ví dụ, vì việc sắp xếp thường yêu cầu một số phân đoạn, có lý khi mà một thuật toán có thể hỏng vì các mảng có một số lẻ các phần tử, từ đó nó không thể được phân chia đều nhau. Test case của bạn nên liệt kê cả 2 ví dụ.
* Case khó đỡ (The extremes): Chuyện gì sẽ xảy ra khi bạn nhận một cái mảng rỗng? hay một cái rất nhỏ (mảng 1 phần tử)? hoặc là một cái rất lớn?
* Input rỗng và “không hợp lệ” (Nulls and “illegal” input): Thật đáng giá để nghĩ về việc code sẽ thể hiện như thế nào khi được nhận một input không hợp lệ. Ví dụ, nếu bạn đang test một chức năng để tạo một dãy n số Fibonacci, test case của bạn chắc hẳn nên chứa cả trường hợp n là số âm.
* Input “lạ” (Strange input): Một dạng input thứ tư đôi lúc xuất hiện: strange input. Chuyện gì xảy ra khi bạn nhận một mảng đã được sắp xếp? hay một mảng được sắp với chiều ngược lại?

Tạo ra những test này đòi hỏi kiến thức về các chức năng bạn đang viết. Nếu bạn còn chưa rõ về những ràng buộc, bạn sẽ cần phải hỏi người phỏng vấn bạn đầu tiên.

*Bước 2: Định nghĩa kết quả mong đợi*

Đôi lúc kết quả mong đợi thật là rõ ràng: một output đúng! Tuy nhiên, trong vài trường hợp bạn có thể muốn xác thực một số khía cạnh khác. Ví dụ, nếu phương thức sắp xếp trả về một bản copy của mảng thì bạn chắc hẳn phải xác thực rằng mảng nguyên thủy chưa bị tác động vào.

*Bước 3: Viết code test*

Một khi bạn đã có test case và định nghĩa kết quả thì việc viết code để bắt đầu việc test nên được bắt tay vào luôn. Code của bạn có thể sẽ như thế này:

1. void testAddThreeSorted() {
2. MyList list = new MyList();
3. list.addThreeSorted(3,1,2); // Adds 3 items in sorted order
4. assertEquals(list.getElement(0),1);
5. assertEquals(list.getElement(1),2);
6. assertEquals(list.getElement(2),3);
7. }

**Câu hỏi sửa chữa (Troubleshooting Questions)**

Một dạng câu hỏi cuối là hãy giải thích bạn sẽ debug hay sửa chữa một vấn đề đang hiện hữu như thế nào. Nhiều ứng viên thường lẩn tránh câu hỏi như đưa ra những câu trả lời không thực tế như “cài đặt lại phần mềm”. Bạn có thể tiếp cận những câu hỏi như thế này bằng một phương pháp có cấu trúc như mọi thứ khác thôi.

Hãy cùng bước qua bấn đề này với một ví dụ: Bạn đang làm việc cho Google Chrome và khi bạn nhận được một bug report: Chrome bị crash khi khởi động. Bạn sẽ làm gì?

Cài đặt lại browser có thể giải quyết vấn đề của người dùng, nhưng nó sẽ không giúp những người dùng khác có cùng vấn đề. Mục tiêu của bạn là hiểu rõ những gì đang thực sự xảy ra và như vậy các developer có thể sửa chữa nó.

*Bước 1: Hiểu về những gì đang xảy ra*

Điều đầu tiên bạn nên làm là hỏi để hiểu rõ về tình hình nhiều nhất có thể.

* Người dùng đã phải trải qua vấn đề này bao lâu rồi?
* Phiên bản browser là đang dùng là gì? Hệ điều hành đang dùng?
* Lỗi này có xảy ra một cách hệ thống hay không, hoặc tần suất xảy ra lỗi? Nó xảy ra khi nào?
* Có error report hiện ra không?

*Step 2: Chia nhỏ vấn đề*

Bây giờ bạn đã hiểu chi tiết về những gì đang xảy ra, bạn muốn chia vấn đề ra thành các thành phần nhỏ có thể test được. Trong trường hợp này , bạn có thể tưởng tượng dòng chảy của nó như sau:

1. Đi đến cửa số Windows
2. Click biểu tượng Chrome
3. Browser instance starts
4. Browser loads settings
5. Browser issues HTTP request for homepage
6. Browser gets HTTP response
7. Browser parses webpage
8. Browser displays content

Tại một số thời điểm trong quá trình này, một cái gì đó gây ra lỗi và nó gây ra hiện tượng crash cho browser. Một tester giỏi sẽ lặp lại những tác nhân này và chẩn đoán vấn đề.

*Bước 3: Tạo ra các test cụ thể và dễ quản lý*

Mỗi thành phần trên nên có một hướng dẫn thực tế - điều mà bạn có thể yêu cầu người dùng làm, hoặc những điều mà bạn có thể tự làm (như là lặp lại các bước trên máy móc của bạn). Trong thế giới thực, bạn sẽ thỏa thuận với khách hàng và bạn không thể đưa cho họ các hướng dẫn mà họ không thể hoặc không làm.