**Bài tập tổng hợp về Kỹ năng và Tích hợp Testing**

**Phần 1: Lý thuyết**

**Kỹ năng thiết yếu của một Tester**

**a. Liệt kê ít nhất 5 kỹ năng thiết yếu của một tester và giải thích tại sao mỗi kỹ năng quan trọng**

1. **Tư duy logic**:
   * **Tầm quan trọng**: Tư duy logic giúp tester phân tích yêu cầu, xác định các trường hợp kiểm thử, và tìm ra nguyên nhân gốc rễ của lỗi. Nó hỗ trợ việc xây dựng các kịch bản kiểm thử toàn diện và dự đoán các vấn đề tiềm ẩn.
2. **Chú ý chi tiết**:
   * **Tầm quan trọng**: Tester cần nhận ra các chi tiết nhỏ trong giao diện, chức năng, hoặc yêu cầu để phát hiện lỗi mà người khác có thể bỏ qua, đảm bảo sản phẩm đạt chất lượng cao.
3. **Giao tiếp hiệu quả**:
   * **Tầm quan trọng**: Tester phải truyền đạt lỗi, yêu cầu, và đề xuất giải pháp một cách rõ ràng với các bên liên quan (developer, product owner), giúp giảm hiểu lầm và tăng hiệu quả hợp tác.
4. **Kỹ năng phân tích**:
   * **Tầm quan trọng**: Kỹ năng này giúp tester hiểu rõ yêu cầu, phân tích rủi ro, và đánh giá tác động của lỗi, từ đó ưu tiên các trường hợp kiểm thử quan trọng.
5. **Hiểu biết công nghệ**:
   * **Tầm quan trọng**: Hiểu về công nghệ (như API, cơ sở dữ liệu, hoặc trình duyệt) giúp tester thực hiện kiểm thử kỹ thuật và phối hợp tốt hơn với đội phát triển.

**b. Tình huống thực tế: Kiểm thử ứng dụng ngân hàng**

* **Tình huống**: Trong ứng dụng ngân hàng, người dùng báo cáo rằng giao dịch chuyển khoản đôi khi thất bại mà không có thông báo lỗi rõ ràng.
* **Ứng dụng tư duy logic**:  
  Tester phân tích quy trình chuyển khoản, chia nhỏ thành các bước (nhập thông tin, xác thực, gửi yêu cầu API, xử lý phản hồi). Họ kiểm tra nhật ký hệ thống (log) để xác định điểm lỗi, ví dụ: API trả về mã lỗi 500 do quá tải máy chủ. Tester cũng xem xét các trường hợp biên, như số tiền chuyển khoản vượt giới hạn hoặc tài khoản không đủ số dư.
* **Ứng dụng kỹ năng giao tiếp**:  
  Tester viết báo cáo lỗi rõ ràng, bao gồm các bước tái hiện, kết quả mong đợi, và kết quả thực tế. Họ tổ chức một cuộc họp ngắn với developer và product owner, giải thích lỗi bằng ngôn ngữ dễ hiểu, đồng thời đề xuất kiểm tra lại cấu hình API hoặc thêm thông báo lỗi thân thiện với người dùng.

**Whole Team Approach**

**a. Whole team approach là gì và lợi ích**

* **Whole team approach là gì?**  
  Đây là phương pháp trong đó tất cả thành viên trong nhóm (developer, tester, product owner, designer, v.v.) cùng chịu trách nhiệm về chất lượng sản phẩm. Thay vì chỉ tester đảm nhiệm kiểm thử, mọi người cùng tham gia vào các hoạt động kiểm thử và đảm bảo chất lượng từ đầu đến cuối dự án.
* **Lợi ích**:
  1. **Tăng cường hợp tác**: Các thành viên chia sẻ trách nhiệm, giảm khoảng cách giữa các vai trò, dẫn đến giao tiếp hiệu quả hơn.
  2. **Phát hiện lỗi sớm**: Mọi người tham gia kiểm thử từ giai đoạn đầu, giúp phát hiện và sửa lỗi trước khi chúng trở nên phức tạp.
  3. **Cải thiện chất lượng sản phẩm**: Sự tham gia của nhiều góc nhìn (kỹ thuật, kinh doanh, thiết kế) đảm bảo sản phẩm đáp ứng tốt yêu cầu người dùng.

**b. Ví dụ về whole team approach**

* **Tình huống**: Phát triển ứng dụng di động cho phép người dùng đặt lịch hẹn bác sĩ.
* **Cách áp dụng**:
  + **Hoạt động hợp tác**: Product owner tổ chức buổi họp yêu cầu với tester và developer để làm rõ các kịch bản người dùng (ví dụ: đặt lịch, hủy lịch, thông báo nhắc nhở). Tester đề xuất các trường hợp kiểm thử biên (như đặt lịch trùng giờ). Developer chia sẻ các hạn chế kỹ thuật (như giới hạn API). Designer đảm bảo giao diện dễ dùng.
  + **Kết quả**: Nhóm thống nhất yêu cầu rõ ràng, giảm lỗi do hiểu sai yêu cầu. Tester và developer thực hiện kiểm thử đôi (pair testing) để phát hiện lỗi giao diện ngay trong giai đoạn phát triển, tăng chất lượng ứng dụng.

**Tích hợp Testing vào vòng đời phát triển phần mềm**

**a. Tích hợp testing trong DevOps và shift-left**

* **Trong DevOps**:  
  Testing được tích hợp liên tục vào quy trình phát triển (CI/CD). Tester tham gia từ giai đoạn lập kế hoạch, viết các kịch bản kiểm thử tự động (unit test, integration test, end-to-end test), và giám sát chất lượng trong môi trường sản xuất. Công cụ như Selenium, JUnit, hoặc Jenkins được sử dụng để chạy kiểm thử tự động mỗi khi có thay đổi mã nguồn.
* **Trong shift-left**:  
  Testing bắt đầu ngay từ giai đoạn phân tích yêu cầu. Tester hợp tác với product owner để đánh giá yêu cầu, xác định rủi ro, và viết các trường hợp kiểm thử trước khi phát triển. Điều này giúp phát hiện vấn đề sớm, giảm chi phí sửa lỗi.

**b. Shift-left testing và ví dụ**

* **Khái niệm**: Shift-left testing là việc đẩy các hoạt động kiểm thử lên sớm trong vòng đời phát triển phần mềm, thay vì chỉ kiểm thử ở giai đoạn cuối.
* **Ví dụ**: Trong dự án phát triển ứng dụng đặt vé máy bay, tester tham gia vào giai đoạn phân tích yêu cầu. Họ kiểm tra xem yêu cầu "hiển thị giá vé theo thời gian thực" có rõ ràng không, xác định các trường hợp như giá vé không tải được khi mất kết nối mạng. Tester viết các trường hợp kiểm thử và chia sẻ với developer trước khi coding, giúp đảm bảo mã nguồn xử lý đúng các trường hợp lỗi.

**Lợi ích của việc bắt đầu Testing sớm**

**a. Lợi ích của testing sớm**

1. **Tiết kiệm chi phí**: Phát hiện lỗi ở giai đoạn đầu (như yêu cầu hoặc thiết kế) rẻ hơn nhiều so với sửa lỗi ở giai đoạn triển khai hoặc sản xuất.
2. **Cải thiện chất lượng sản phẩm**: Testing sớm đảm bảo yêu cầu rõ ràng và đúng ngay từ đầu, giảm lỗi nghiêm trọng trong sản phẩm cuối.
3. **Tăng hiệu quả phát triển**: Phát hiện lỗi sớm giúp developer sửa lỗi nhanh hơn, tránh phải làm lại nhiều phần mã nguồn.

**b. Ví dụ về lỗi được phát hiện sớm**

* **Lỗi**: Trong một ứng dụng thương mại điện tử, hệ thống không kiểm tra số lượng hàng tồn kho trước khi cho phép đặt hàng, dẫn đến việc khách hàng đặt hàng sản phẩm đã hết.
* **Phát hiện sớm**: Trong giai đoạn phân tích yêu cầu, tester nhận thấy yêu cầu không đề cập đến kiểm tra tồn kho. Họ đề xuất thêm kiểm tra này vào yêu cầu và viết các trường hợp kiểm thử (như đặt hàng khi tồn kho bằng 0).
* **Tiết kiệm chi phí**: Nếu lỗi này được phát hiện ở giai đoạn sản xuất, công ty có thể phải hoàn tiền, xử lý khiếu nại khách hàng, và sửa mã nguồn với chi phí cao hơn. Testing sớm giúp tránh các chi phí này và tăng sự hài lòng của khách hàng.

**Phần 2: Bài tập thực hành**

**Áp dụng kỹ năng Tester**

**a. Sử dụng tư duy logic để phân tích lỗi**

* **Lỗi**: Người dùng không thể thêm sản phẩm vào giỏ hàng khi sử dụng trình duyệt cũ (ví dụ: Internet Explorer 8).
* **Phân tích**:
  1. **Xác định phạm vi lỗi**: Tester kiểm tra xem lỗi xảy ra trên tất cả trình duyệt cũ hay chỉ một số phiên bản cụ thể.
  2. **Phân tích kỹ thuật**: Tester kiểm tra console trình duyệt để tìm lỗi JavaScript (ví dụ: hàm addToCart sử dụng API không được hỗ trợ trên trình duyệt cũ, như fetch).
  3. **Kiểm tra trường hợp biên**: Tester thử thêm sản phẩm với các điều kiện khác nhau (sản phẩm có số lượng lớn, không đăng nhập, v.v.) để xác định phạm vi lỗi.
  4. **Kết luận**: Nguyên nhân có thể là mã JavaScript không tương thích với trình duyệt cũ. Tester đề xuất sử dụng polyfill hoặc thay thế hàm fetch bằng XMLHttpRequest.

**b. Sử dụng kỹ năng giao tiếp để báo cáo lỗi**

* **Báo cáo lỗi**:  
  Tester viết báo cáo lỗi với các thông tin:
  + **Tiêu đề**: "Không thể thêm sản phẩm vào giỏ hàng trên Internet Explorer 8".
  + **Mô tả**: Lỗi xảy ra khi người dùng nhấn nút "Thêm vào giỏ hàng". Không có phản hồi, và console báo lỗi "fetch is not defined".
  + **Bước tái hiện**:
    1. Mở Internet Explorer 8.
    2. Truy cập trang sản phẩm.
    3. Nhấn nút "Thêm vào giỏ hàng".
  + **Kết quả mong đợi**: Sản phẩm được thêm vào giỏ hàng, hiển thị thông báo thành công.
  + **Kết quả thực tế**: Không có phản hồi, sản phẩm không được thêm.
  + **Đề xuất**: Thay thế hàm fetch bằng XMLHttpRequest hoặc thêm polyfill để hỗ trợ trình duyệt cũ.
* **Giao tiếp**: Tester tổ chức một cuộc họp ngắn với developer, trình bày lỗi bằng ngôn ngữ rõ ràng, cung cấp ảnh chụp màn hình và log console. Họ cũng thảo luận với product owner để xác định mức độ ưu tiên của lỗi (dựa trên số lượng người dùng sử dụng trình duyệt cũ).

**Thiết kế kế hoạch hợp tác Whole Team Approach**

* **Ứng dụng**: Tính năng "thêm sản phẩm vào giỏ hàng" trong ứng dụng thương mại điện tử.
* **Kế hoạch hợp tác**:
  + **Vai trò tham gia**:
    - **Tester**: Thiết kế và thực hiện các trường hợp kiểm thử, báo cáo lỗi.
    - **Developer**: Phát triển và sửa lỗi mã nguồn, hỗ trợ kiểm thử tự động.
    - **Designer**: Đảm bảo giao diện nút "Thêm vào giỏ hàng" rõ ràng và thân thiện.
    - **Product owner**: Làm rõ yêu cầu và ưu tiên các trường hợp sử dụng.
  + **Hoạt động cụ thể**:
    - **Buổi họp yêu cầu**: Product owner trình bày yêu cầu (ví dụ: thêm sản phẩm nhanh, hiển thị số lượng trong giỏ hàng). Tester đề xuất các trường hợp kiểm thử (như thêm sản phẩm hết hàng, thêm khi chưa đăng nhập).
    - **Kiểm thử đôi (pair testing)**: Tester và developer cùng kiểm thử tính năng trên các trình duyệt khác nhau, phát hiện lỗi tương thích sớm.
    - **Xây dựng pipeline CI/CD**: Developer tích hợp kiểm thử tự động (unit test cho hàm addToCart, end-to-end test cho quy trình thêm sản phẩm) vào pipeline CI/CD. Tester xem xét kết quả kiểm thử.
    - **Đánh giá giao diện**: Designer và tester kiểm tra giao diện trên các thiết bị để đảm bảo nút "Thêm vào giỏ hàng" hoạt động đúng và trực quan.
  + **Kết quả mong đợi**:
    - Tính năng hoạt động ổn định trên tất cả trình duyệt và thiết bị.
    - Giảm lỗi do hiểu sai yêu cầu nhờ giao tiếp chặt chẽ.
    - Tăng sự hài lòng của người dùng nhờ giao diện thân thiện và hiệu suất tốt.

**Áp dụng Shift-Left Testing**

**a. Hoạt động testing sớm trong dự án quản lý kho hàng**

1. **Kiểm thử yêu cầu**: Tester hợp tác với product owner để xem xét yêu cầu, ví dụ: "hệ thống hiển thị số lượng hàng tồn kho theo thời gian thực". Tester đặt câu hỏi để làm rõ, như "Điều gì xảy ra nếu dữ liệu tồn kho không đồng bộ?" và đề xuất thêm kiểm tra lỗi đồng bộ.
2. **Phân tích rủi ro**: Tester xác định các rủi ro, như nhập kho âm do lỗi tính toán. Họ đề xuất các trường hợp kiểm thử (ví dụ: nhập số lượng âm, nhập số vượt quá giới hạn).
3. **Viết kịch bản kiểm thử sớm**: Tester viết các trường hợp kiểm thử (unit test, integration test) trước khi developer bắt đầu coding, đảm bảo mã nguồn đáp ứng yêu cầu ngay từ đầu.

**b. Lợi ích của các hoạt động này**

* **Cải thiện chất lượng sản phẩm**:  
  Kiểm thử yêu cầu giúp phát hiện các yêu cầu không rõ ràng hoặc mâu thuẫn (ví dụ: không xác định hành vi khi tồn kho âm). Điều này đảm bảo hệ thống được thiết kế đúng ngay từ đầu, giảm lỗi nghiêm trọng.
* **Giảm chi phí**:  
  Phát hiện lỗi ở giai đoạn yêu cầu hoặc thiết kế rẻ hơn nhiều so với sửa lỗi ở giai đoạn triển khai. Ví dụ, việc thêm kiểm tra tồn kho âm trong yêu cầu tránh được việc sửa mã nguồn và tái kiểm thử ở giai đoạn sau.
* **Tăng hiệu quả phát triển**: Developer nhận được các trường hợp kiểm thử rõ ràng từ đầu, giúp họ viết mã đúng yêu cầu và tránh làm lại.