### **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**:
   * **Lý do quan trọng**: Tester cần phân tích yêu cầu, thiết kế test case và xác định nguyên nhân lỗi một cách có hệ thống. Tư duy logic giúp phát hiện các kịch bản bất thường và đảm bảo kiểm thử toàn diện.
   * **Ví dụ**: Khi kiểm thử một chức năng, tester sử dụng tư duy logic để xây dựng các trường hợp kiểm thử (test case) bao quát cả trường hợp hợp lệ và không hợp lệ.
2. **Giao tiếp hiệu quả**:
   * **Lý do quan trọng**: Tester cần truyền đạt lỗi, yêu cầu và kết quả kiểm thử rõ ràng với các bên liên quan (developer, product owner). Giao tiếp tốt giúp giảm hiểu lầm và tăng hiệu quả xử lý vấn đề.
   * **Ví dụ**: Báo cáo lỗi một cách chi tiết, dễ hiểu giúp developer nhanh chóng xác định và sửa lỗi.
3. **Chú ý chi tiết**:
   * **Lý do quan trọng**: Tester phải nhận biết những khác biệt nhỏ trong giao diện, chức năng hoặc dữ liệu để phát hiện lỗi mà người khác có thể bỏ qua.
   * **Ví dụ**: Phát hiện một lỗi hiển thị sai font chữ hoặc giá trị tính toán sai trong ứng dụng.
4. **Kỹ năng phân tích và giải quyết vấn đề**:
   * **Lý do quan trọng**: Tester cần phân tích nguyên nhân gốc rễ của lỗi và đề xuất giải pháp hoặc tái hiện lỗi để hỗ trợ developer.
   * **Ví dụ**: Xác định lỗi xảy ra do xung đột dữ liệu trong cơ sở dữ liệu và đề xuất kiểm tra ràng buộc dữ liệu.
5. **Hiểu biết về lĩnh vực (domain knowledge)**:
   * **Lý do quan trọng**: Hiểu biết về lĩnh vực của ứng dụng (ví dụ: ngân hàng, thương mại điện tử) giúp tester thiết kế các kịch bản kiểm thử phù hợp với yêu cầu thực tế.
   * **Ví dụ**: Trong ứng dụng ngân hàng, tester cần hiểu các quy tắc giao dịch để kiểm thử các tính năng như chuyển khoản.

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

* **Tình huống**: Một người dùng báo cáo rằng khi chuyển khoản qua ứng dụng ngân hàng, số tiền đôi khi bị hiển thị sai trên giao diện xác nhận.
* **Cách sử dụng tư duy logic**:
  + Tester phân tích luồng dữ liệu từ giao diện người dùng đến hệ thống backend, kiểm tra xem lỗi xảy ra ở giao diện (UI), logic xử lý hay cơ sở dữ liệu.
  + Tester thiết kế các test case để kiểm tra các trường hợp: số tiền lớn, số tiền lẻ, giao dịch liên tục.
  + Tester tái hiện lỗi bằng cách lặp lại các bước người dùng thực hiện, ghi lại các điều kiện cụ thể (ví dụ: trình duyệt, loại tài khoản).
* **Cách sử dụng kỹ năng giao tiếp**:
  + Tester viết báo cáo lỗi chi tiết, bao gồm: mô tả lỗi, bước tái hiện, ảnh chụp màn hình, và tác động (ví dụ: gây nhầm lẫn cho người dùng).
  + Tester tổ chức một buổi họp ngắn với developer và product owner để thảo luận nguyên nhân lỗi, đề xuất kiểm tra định dạng dữ liệu ở giao diện.
  + Tester sử dụng ngôn ngữ rõ ràng, không kỹ thuật hóa khi giải thích với product owner để đảm bảo họ hiểu mức độ nghiêm trọng.

#### **Whole Team Approach**

**a. Whole Team Approach là gì? Lợi ích của phương pháp này**

* **Whole Team Approach**: Là cách tiếp cận trong đó mọi thành viên trong nhóm dự án (developer, tester, product owner, designer, v.v.) cùng chịu trách nhiệm về chất lượng sản phẩm. Testing không chỉ là nhiệm vụ của tester mà là sự hợp tác của cả đội.
* **Lợi ích**:
  1. **Tăng cường chất lượng sản phẩm**: Mọi người cùng tham gia kiểm thử từ sớm, giúp phát hiện lỗi ở giai đoạn đầu, giảm chi phí sửa lỗi.
  2. **Cải thiện giao tiếp và hợp tác**: Các vai trò làm việc chặt chẽ, giảm hiểu lầm về yêu cầu và tăng tính minh bạch.
  3. **Tăng tốc độ phát triển**: Phát hiện lỗi sớm và phản hồi nhanh giúp dự án tiến triển mượt mà hơn, đặc biệt trong các mô hình Agile hoặc DevOps.

**b. Ví dụ cụ thể về Whole Team Approach**

* **Tình huống**: Phát triển tính năng đặt hàng trên ứng dụng di động thương mại điện tử.
* **Cách hợp tác**:
  + **Buổi họp yêu cầu**: Product owner trình bày yêu cầu (ví dụ: cho phép đặt hàng nhanh trong 1 cú nhấp chuột). Tester đặt câu hỏi về các trường hợp ngoại lệ (ví dụ: khi hết hàng). Developer đề xuất giải pháp kỹ thuật.
  + **Kiểm thử đôi (Pair Testing)**: Tester và developer cùng kiểm thử tính năng trên môi trường thử nghiệm, phát hiện lỗi giao diện ngay lập tức.
  + **Phản hồi liên tục**: Designer cung cấp mockup giao diện, tester kiểm tra tính khả dụng, và product owner xác nhận tính năng phù hợp với nhu cầu khách hàng.
* **Kết quả**: Tính năng được phát triển đúng yêu cầu, ít lỗi hơn, và thời gian phát hành được rút ngắn.

#### **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 vào quy trình CI/CD (Continuous Integration/Continuous Deployment). Mỗi lần code được đẩy lên, các bài kiểm thử tự động (unit test, integration test) được chạy để phát hiện lỗi ngay lập tức.
  + Tester hợp tác với developer để xây dựng pipeline kiểm thử tự động, đảm bảo chất lượng ở mọi giai đoạn (từ phát triển đến triển khai).
  + Ví dụ: Trong một dự án, tester thiết lập kiểm thử API tự động để kiểm tra tính năng thanh toán trước khi triển khai lên production.
* **Trong Shift-Left**:
  + Shift-Left Testing chuyển các hoạt động kiểm thử sang giai đoạn sớm của vòng đời phát triển (phân tích yêu cầu, thiết kế).
  + Tester tham gia từ giai đoạn xác định yêu cầu để đảm bảo yêu cầu rõ ràng, khả thi và có thể kiểm thử.
  + Ví dụ: Tester phân tích tài liệu yêu cầu để phát hiện mâu thuẫn (ví dụ: yêu cầu không rõ ràng về xử lý lỗi khi người dùng nhập sai mật khẩu).

**b. Khái niệm Shift-Left Testing và ví dụ**

* **Khái niệm**: Shift-Left Testing là việc thực hiện kiểm thử sớm trong vòng đời phát triển phần mềm, thay vì chỉ kiểm thử ở giai đoạn cuối (như triển khai hoặc release).
* **Ví dụ áp dụng**:
  + **Dự án**: Phát triển ứng dụng quản lý học sinh.
  + **Hoạt động Shift-Left**: Tester tham gia buổi phân tích yêu cầu, phát hiện rằng yêu cầu "xóa hồ sơ học sinh" không đề cập đến việc kiểm tra quyền truy cập. Tester đề xuất thêm kiểm tra quyền admin trước khi xóa.
  + **Kết quả**: Lỗi bảo mật tiềm ẩn được phát hiện và sửa ngay từ giai đoạn thiết kế, tránh chi phí sửa lỗi lớn ở giai đoạn sau.

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

**a. Phân tích 3 lợi ích của việc bắt đầu testing sớm**

1. **Tiết kiệm chi phí**: Phát hiện lỗi ở giai đoạn sớm (yêu cầu, thiết kế) rẻ hơn nhiều so với sửa lỗi ở giai đoạn triển khai.
   * Ví dụ: Sửa lỗi logic trong yêu cầu tốn ít thời gian hơn sửa lỗi trong code đã triển khai.
2. **Cải thiện chất lượng sản phẩm**: Kiểm thử sớm đảm bảo yêu cầu rõ ràng, giảm lỗi nghiêm trọng trong sản phẩm cuối cùng.
   * Ví dụ: Kiểm tra tính khả dụng giao diện ngay từ mockup giúp tránh thiết kế sai lệch.
3. **Rút ngắn thời gian phát hành**: Phát hiện và sửa lỗi sớm giúp giảm thời gian lặp lại (iteration) và đẩy nhanh tiến độ dự án.
   * Ví dụ: Phát hiện lỗi tích hợp API sớm giúp developer điều chỉnh trước khi triển khai.

**b. Ví dụ về lỗi phát hiện sớm và tiết kiệm chi phí**

* **Lỗi**: Trong một ứng dụng thương mại điện tử, tính năng tính thuế không được xác định rõ ràng trong yêu cầu (ví dụ: không quy định cách xử lý thuế cho các quốc gia khác nhau).
* **Phát hiện sớm**: Tester tham gia phân tích yêu cầu, phát hiện thiếu sót này và yêu cầu product owner làm rõ quy tắc tính thuế.
* **Tiết kiệm chi phí**: Nếu lỗi này chỉ được phát hiện ở giai đoạn triển khai, đội phát triển phải sửa đổi code, chạy lại kiểm thử và triển khai lại, tốn thời gian và nguồn lực. Phát hiện sớm giúp điều chỉnh yêu cầu trước khi coding, tiết kiệm chi phí sửa đổi.

### **Phần 2: Bài tập thực hành**

#### **Áp dụng kỹ năng Tester**

**Tình huống**: Ứng dụng thương mại điện tử, lỗi "người dùng không thể thêm sản phẩm vào giỏ hàng khi sử dụng trình duyệt cũ."

**a. Sử dụng tư duy logic để phân tích và xác định nguyên nhân lỗi**

* **Bước 1: Phân tích lỗi**:
  + Tester xác định các yếu tố liên quan: trình duyệt (loại, phiên bản), hệ điều hành, và hành vi cụ thể (nút "Thêm vào giỏ hàng" không hoạt động hay không phản hồi).
  + Tester kiểm tra xem lỗi xảy ra trên tất cả trình duyệt cũ hay chỉ một số phiên bản cụ thể (ví dụ: Internet Explorer 9).
* **Bước 2: Tái hiện lỗi**:
  + Tester sử dụng trình duyệt cũ (ví dụ: IE9) và thực hiện các bước: truy cập trang sản phẩm, nhấn nút "Thêm vào giỏ hàng".
  + Tester kiểm tra console trình duyệt để tìm lỗi JavaScript hoặc kiểm tra network để xem API có được gọi đúng không.
* **Bước 3: Xác định nguyên nhân**:
  + Giả thuyết 1: Code JavaScript sử dụng cú pháp hiện đại (ES6) không tương thích với trình duyệt cũ.
  + Giả thuyết 2: API không phản hồi đúng do trình duyệt không hỗ trợ một số giao thức (như HTTPS).
  + Tester chạy thử trên các trình duyệt khác nhau và xác nhận lỗi do thiếu polyfill cho JavaScript ES6.

**b. Sử dụng kỹ năng giao tiếp để báo cáo lỗi và đề xuất giải pháp**

* **Báo cáo lỗi**:
  + Tester tạo ticket trên hệ thống quản lý lỗi (ví dụ: Jira) với nội dung:
    - **Tiêu đề**: "Không thể thêm sản phẩm vào giỏ hàng trên trình duyệt IE9."
    - **Mô tả**: Lỗi xảy ra khi nhấn nút "Thêm vào giỏ hàng" trên IE9. Console báo lỗi "Arrow function không được hỗ trợ."
    - **Bước tái hiện**:
      1. Mở IE9.
      2. Truy cập trang sản phẩm.
      3. Nhấn nút "Thêm vào giỏ hàng".
    - **Ảnh chụp màn hình/Video**: Đính kèm console lỗi và giao diện.
    - **Tác động**: Ảnh hưởng đến người dùng sử dụng trình duyệt cũ, làm giảm trải nghiệm.
  + Tester gửi email ngắn gọn đến đội phát triển, tóm tắt lỗi và đề xuất họp nhanh nếu cần.
* **Đề xuất giải pháp**:
  + Tester gợi ý thêm polyfill (như Babel) để hỗ trợ cú pháp JavaScript trên trình duyệt cũ.
  + Tester đề xuất kiểm tra danh sách trình duyệt được hỗ trợ trong yêu cầu để xác định xem có cần hỗ trợ IE9 hay không.

#### **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**:

* **Các vai trò tham gia**:
  + **Tester**: Thiết kế và thực hiện test case, kiểm tra tính tương thích trình duyệt.
  + **Developer**: Phát triển và sửa lỗi tính năng "Thêm vào giỏ hàng".
  + **Designer**: Đảm bảo giao diện nút "Thêm vào giỏ hàng" thân thiện và nhất quán trên các trình duyệt.
  + **Product Owner**: Xác định yêu cầu, ưu tiên hỗ trợ trình duyệt cũ dựa trên đối tượng người dùng.
* **Các hoạt động cụ thể**:
  + **Buổi họp yêu cầu**:
    - Product owner trình bày yêu cầu (ví dụ: hỗ trợ các trình duyệt phổ biến và một số trình duyệt cũ).
    - Tester đặt câu hỏi về các trường hợp ngoại lệ (ví dụ: khi giỏ hàng đầy, trình duyệt không hỗ trợ JavaScript).
    - Developer đề xuất các giải pháp kỹ thuật (như sử dụng polyfill).
  + **Kiểm thử đôi (Pair Testing)**:
    - Tester và developer cùng kiểm thử tính năng trên các trình duyệt khác nhau, đặc biệt là trình duyệt cũ, để phát hiện lỗi sớm.
  + **Kiểm tra giao diện với designer**:
    - Tester và designer kiểm tra tính khả dụng của nút "Thêm vào giỏ hàng" trên các thiết bị và trình duyệt.
  + **Phản hồi liên tục**:
    - Tester cập nhật kết quả kiểm thử trên công cụ quản lý dự án (như Jira) và tổ chức họp ngắn để thảo luận lỗi nghiêm trọng.
* **Kết quả mong đợi**:
  + Tính năng "Thêm vào giỏ hàng" hoạt động ổn định trên các trình duyệt, bao gồm cả trình duyệt cũ.
  + Giảm số lượng lỗi được báo cáo bởi người dùng sau khi triển khai.
  + Tăng cường sự hiểu biết lẫn nhau giữa các vai trò, cải thiện hiệu quả làm việc nhóm.

#### **Áp dụng Shift-Left Testing**

**Dự án**: Ứng dụng quản lý kho hàng.

**a. Đề xuất 3 hoạt động testing ở giai đoạn sớm**

1. **Kiểm thử yêu cầu**:
   * Tester tham gia phân tích tài liệu yêu cầu, kiểm tra tính rõ ràng và khả thi của các chức năng (ví dụ: kiểm tra yêu cầu về nhập kho có bao gồm xử lý lỗi khi nhập sai mã sản phẩm không).
2. **Phân tích rủi ro**:
   * Tester cùng product owner xác định các rủi ro tiềm ẩn (ví dụ: lỗi tính toán số lượng tồn kho có thể gây thiệt hại lớn). Tester đề xuất ưu tiên kiểm thử các chức năng quan trọng như nhập/xuất kho.
3. **Thiết kế test case sớm**:
   * Tester xây dựng test case dựa trên yêu cầu (ví dụ: kiểm tra nhập kho với số lượng âm, số lượng vượt quá giới hạn).

**b. Phân tích cách các hoạt động này cải thiện chất lượng và giảm chi phí**

* **Cải thiện chất lượng sản phẩm**:
  + **Kiểm thử yêu cầu**: Phát hiện mâu thuẫn trong yêu cầu (ví dụ: thiếu quy định về giới hạn tồn kho) giúp đảm bảo sản phẩm đáp ứng đúng nhu cầu.
  + **Phân tích rủi ro**: Tập trung kiểm thử vào các chức năng quan trọng giúp giảm lỗi nghiêm trọng (như sai lệch số lượng tồn kho).
  + **Thiết kế test case sớm**: Test case được chuẩn bị trước giúp phát hiện lỗi ngay từ giai đoạn phát triển, đảm bảo sản phẩm ổn định hơn.
* **Giảm chi phí**:
  + Sửa lỗi ở giai đoạn yêu cầu hoặc thiết kế tốn ít thời gian và nguồn lực hơn so với sửa lỗi sau khi triển khai.
  + Ví dụ: Phát hiện lỗi yêu cầu về nhập kho sai sớm giúp tránh việc sửa đổi code và cơ sở dữ liệu ở giai đoạn sau, tiết kiệm chi phí phát triển và kiểm thử lại.