### **Phần 1: Lý thuyết**

#### **1. Khái niệm Testing**

**a. Testing là gì? Tại sao testing lại quan trọng trong việc giảm thiểu rủi ro phần mềm?**

**Testing** là quá trình kiểm tra và đánh giá một phần mềm để đảm bảo rằng nó hoạt động đúng như yêu cầu, không có lỗi và đáp ứng được nhu cầu của người dùng. Testing bao gồm việc chạy phần mềm trong các điều kiện khác nhau để phát hiện lỗi, kiểm tra tính năng và đảm bảo chất lượng.

**Tầm quan trọng của testing**:

**Giảm thiểu rủi ro**: Testing giúp phát hiện lỗi sớm, giảm nguy cơ phần mềm bị lỗi khi triển khai, từ đó tránh các hậu quả như mất dữ liệu, thất thoát tài chính, hoặc làm mất lòng tin của người dùng.

**Đảm bảo chất lượng**: Testing xác minh rằng phần mềm đáp ứng các yêu cầu chức năng và phi chức năng, đảm bảo trải nghiệm người dùng tốt.

**Tiết kiệm chi phí**: Phát hiện và sửa lỗi trong giai đoạn phát triển rẻ hơn nhiều so với sửa lỗi sau khi triển khai.

**Tăng độ tin cậy**: Phần mềm được kiểm tra kỹ lưỡng sẽ đáng tin cậy hơn, giảm thiểu sự cố không mong muốn.

**b. Ví dụ thực tế về rủi ro phần mềm nếu không thực hiện testing**:

Một ứng dụng ngân hàng trực tuyến không được kiểm tra kỹ lưỡng có thể cho phép chuyển khoản vượt quá số dư tài khoản. Nếu không phát hiện lỗi này trước khi triển khai, khách hàng có thể thực hiện các giao dịch sai lệch, dẫn đến tổn thất tài chính lớn cho ngân hàng và mất uy tín với khách hàng.

#### **2. Phân biệt Testing và Debugging**

**a. Sự khác biệt giữa testing và debugging**:

**Testing**: Là quá trình thực hiện các bài kiểm tra để phát hiện lỗi hoặc xác minh rằng phần mềm hoạt động đúng như mong đợi. Testing tập trung vào việc tìm ra các vấn đề (defect) mà không cần xác định nguyên nhân cụ thể.

Mục tiêu: Phát hiện lỗi (defect) hoặc xác nhận phần mềm đáp ứng yêu cầu.

Người thực hiện: Tester hoặc nhóm QA.

**Debugging**: Là quá trình phân tích, xác định nguyên nhân gốc rễ (root cause) của lỗi đã được tìm thấy trong quá trình testing và sửa chữa lỗi đó.

Mục tiêu: Xác định và sửa lỗi trong mã nguồn.

Người thực hiện: Lập trình viên hoặc nhóm phát triển.

**b. Ví dụ cụ thể về lỗi hệ thống**:

**Tình huống**: Ứng dụng web bị treo khi người dùng nhập dữ liệu không hợp lệ (ví dụ: nhập ký tự đặc biệt vào trường số điện thoại).

**Testing**:

Tester thực hiện kiểm tra bằng cách nhập dữ liệu không hợp lệ (ví dụ: nhập “abc@#$” vào trường số điện thoại).

Kết quả: Hệ thống bị treo hoặc hiển thị lỗi không rõ ràng. Tester ghi nhận đây là một defect.

**Debugging**:

Lập trình viên phân tích mã nguồn để tìm nguyên nhân gốc rễ (root cause), ví dụ: hàm xử lý đầu vào không kiểm tra định dạng dữ liệu.

Lập trình viên sửa mã nguồn bằng cách thêm kiểm tra định dạng (chỉ cho phép số) và xử lý lỗi (hiển thị thông báo “Vui lòng nhập số hợp lệ”).

#### **3. Mục tiêu của Testing**

**a. 5 mục tiêu chính của testing**:

**Xác minh yêu cầu**: Đảm bảo phần mềm đáp ứng tất cả các yêu cầu chức năng và phi chức năng đã được định nghĩa.

**Đảm bảo chất lượng**: Đánh giá hiệu suất, độ tin cậy và trải nghiệm người dùng để đảm bảo phần mềm đạt tiêu chuẩn chất lượng.

**Phát hiện lỗi**: Tìm ra các lỗi hoặc defect trong phần mềm trước khi triển khai.

**Đảm bảo tính tương thích**: Kiểm tra phần mềm hoạt động tốt trên các nền tảng, thiết bị hoặc trình duyệt khác nhau.

**Tăng độ tin cậy và an toàn**: Đảm bảo phần mềm hoạt động ổn định và bảo vệ dữ liệu người dùng, đặc biệt trong các hệ thống nhạy cảm như tài chính hoặc y tế.

**b. Giải thích một mục tiêu – Phát hiện lỗi**:

**Tại sao quan trọng**: Phát hiện lỗi sớm trong quy trình phát triển giúp giảm chi phí sửa chữa và ngăn chặn các sự cố nghiêm trọng khi phần mềm được triển khai. Ví dụ, nếu một lỗi trong hệ thống thanh toán không được phát hiện, nó có thể dẫn đến giao dịch sai hoặc mất dữ liệu, gây tổn thất lớn. Testing giúp phát hiện các lỗi này để lập trình viên có thể sửa trước khi phần mềm đến tay người dùng.

#### **4. Phân biệt Error, Defect, Failure và Root Cause**

**a. Định nghĩa**:

**Error**: Là sai lầm do con người gây ra trong quá trình phát triển, ví dụ: lập trình viên viết sai logic hoặc bỏ sót một trường hợp.

**Defect**: Là vấn đề hoặc lỗi trong phần mềm được phát hiện trong quá trình testing, bắt nguồn từ error.

**Failure**: Là hiện tượng phần mềm không hoạt động đúng như mong đợi khi được sử dụng, thường là kết quả của defect.

**Root Cause**: Là nguyên nhân gốc rễ dẫn đến error, defect hoặc failure, cần được xác định để sửa chữa triệt để.

**b. Phân tích trường hợp cụ thể – Hệ thống thanh toán trực tuyến trả về sai số tiền**:

**Tình huống**: Khi người dùng thanh toán 100.000 VND, hệ thống hiển thị số tiền là 1.000.000 VND.

**Error**: Lập trình viên viết sai logic tính toán, ví dụ: nhân số tiền với 10 thay vì giữ nguyên giá trị.

**Defect**: Trong quá trình testing, tester phát hiện rằng hệ thống hiển thị sai số tiền khi thực hiện giao dịch.

**Failure**: Người dùng nhận được thông báo giao dịch với số tiền sai (1.000.000 VND thay vì 100.000 VND).

**Root Cause**: Thiếu kiểm tra định dạng số tiền trong mã nguồn hoặc không có kiểm tra xác minh số tiền trước khi hiển thị.

### **Phần 2: Bài tập thực hành**

#### **1. Phân tích trường hợp thực tế – Ứng dụng đặt vé máy bay**

**Tình huống**: Hệ thống cho phép người dùng đặt vé máy bay với ngày bay trong quá khứ và hoàn tất giao dịch.

**a. Xác định error, defect, failure và root cause**:

**Error**: Lập trình viên không thêm kiểm tra để ngăn chặn việc chọn ngày trong quá khứ trong mã nguồn.

**Defect**: Trong quá trình testing, tester phát hiện hệ thống không chặn việc chọn ngày bay trong quá khứ.

**Failure**: Hệ thống hoàn tất giao dịch với ngày bay không hợp lệ (ví dụ: ngày 15/10/2024 khi hôm nay là 24/04/2025).

**Root Cause**: Thiếu logic kiểm tra ngày hiện tại so với ngày được chọn trong tính năng chọn ngày bay.

**b. Đề xuất quy trình testing để phát hiện lỗi này**:

**Kiểm tra yêu cầu**: Xác minh rằng yêu cầu của hệ thống bao gồm việc ngăn chặn chọn ngày trong quá khứ.

**Thiết kế test cases**: Tạo các kịch bản kiểm thử cho tính năng chọn ngày bay, bao gồm:

Chọn ngày trong quá khứ.

Chọn ngày hiện tại.

Chọn ngày trong tương lai.

Chọn ngày không hợp lệ (ví dụ: định dạng sai).

**Thực hiện kiểm thử**:

Sử dụng kiểm thử thủ công hoặc tự động để chạy các test cases.

Kiểm tra giao diện người dùng (UI) để đảm bảo thông báo lỗi hiển thị đúng.

**Kiểm thử tích hợp**: Đảm bảo rằng tính năng chọn ngày bay hoạt động đúng với các module khác (ví dụ: thanh toán, xác nhận vé).

**Kiểm thử hồi quy**: Sau khi sửa lỗi, kiểm tra lại toàn bộ hệ thống để đảm bảo không có lỗi mới phát sinh.

#### **2. Thiết kế kịch bản kiểm thử**

Dưới đây là 5 kịch bản kiểm thử cho tính năng chọn ngày bay của ứng dụng đặt vé máy bay:

**Test Case 1**:

**Mô tả**: Kiểm tra chọn ngày bay trong quá khứ.

**Dữ liệu đầu vào**: Ngày 15/10/2024 (giả sử ngày hiện tại là 24/04/2025).

**Kết quả mong đợi**: Hệ thống hiển thị thông báo lỗi “Không thể chọn ngày trong quá khứ” và không cho phép tiếp tục giao dịch.

**Test Case 2**:

**Mô tả**: Kiểm tra chọn ngày bay là ngày hiện tại.

**Dữ liệu đầu vào**: Ngày 24/04/2025 (ngày hiện tại).

**Kết quả mong đợi**: Hệ thống cho phép chọn ngày hiện tại và tiếp tục quy trình đặt vé.

**Test Case 3**:

**Mô tả**: Kiểm tra chọn ngày bay trong tương lai.

**Dữ liệu đầu vào**: Ngày 01/05/2025.

**Kết quả mong đợi**: Hệ thống cho phép chọn ngày và tiếp tục quy trình đặt vé mà không có lỗi.

**Test Case 4**:

**Mô tả**: Kiểm tra chọn ngày bay với định dạng không hợp lệ.

**Dữ liệu đầu vào**: Nhập “abc” vào trường ngày bay.

**Kết quả mong đợi**: Hệ thống hiển thị thông báo lỗi “Vui lòng nhập định dạng ngày hợp lệ (DD/MM/YYYY)”.

**Test Case 5**:

**Mô tả**: Kiểm tra chọn ngày bay vượt quá giới hạn cho phép (ví dụ: hơn 1 năm).

**Dữ liệu đầu vào**: Ngày 01/05/2026 (giả sử hệ thống chỉ cho phép đặt vé trong vòng 1 năm).

**Kết quả mong đợi**: Hệ thống hiển thị thông báo lỗi “Ngày bay không được vượt quá 1 năm kể từ ngày hiện tại”.

#### **3. Phân tích rủi ro và mục tiêu testing**

**a. 3 rủi ro tiềm ẩn nếu tính năng chọn ngày bay không được kiểm tra kỹ lưỡng**:

**Đặt vé không hợp lệ**: Người dùng có thể đặt vé cho các ngày không khả dụng (quá khứ hoặc quá xa trong tương lai), dẫn đến lỗi hệ thống hoặc thiệt hại tài chính.

**Mất uy tín**: Khách hàng mất niềm tin vào ứng dụng nếu hệ thống cho phép đặt vé sai, ảnh hưởng đến danh tiếng của công ty.

**Tổn thất tài chính**: Nếu vé được đặt cho ngày không hợp lệ, hãng hàng không hoặc công ty cung cấp dịch vụ có thể phải hoàn tiền hoặc chịu chi phí xử lý giao dịch sai.

**b. Mục tiêu testing cụ thể để giảm thiểu rủi ro**:

**Xác minh rằng hệ thống từ chối các ngày không hợp lệ**: Kiểm tra rằng các ngày trong quá khứ hoặc vượt quá giới hạn (ví dụ: hơn 1 năm) bị từ chối với thông báo lỗi rõ ràng.

**Đảm bảo tính chính xác của giao diện người dùng**: Kiểm tra rằng lịch chọn ngày chỉ hiển thị các ngày hợp lệ và không cho phép chọn ngày không khả dụng.

**Kiểm tra tích hợp với hệ thống thanh toán**: Đảm bảo rằng các ngày bay được chọn hợp lệ được truyền đúng sang các module khác (ví dụ: thanh toán, xác nhận vé).