**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 gì?**  
  Testing (kiểm thử) là quá trình đánh giá một hệ thống phần mềm hoặc một thành phần của nó để xác minh rằng nó đáp ứng các yêu cầu đã đặt ra và hoạt động đúng như mong đợi. Testing bao gồm việc thực hiện các kịch bản kiểm thử (test cases) để phát hiện lỗi, đảm bảo chất lượng và hiệu suất của phần mềm.
* **Tại sao testing quan trọng trong việc giảm thiểu rủi ro phần mềm?**  
  Testing giúp:
  + **Phát hiện lỗi sớm**: Giảm thiểu khả năng lỗi xuất hiện trong môi trường thực tế, từ đó tiết kiệm chi phí sửa chữa.
  + **Đảm bảo chất lượng**: Đảm bảo phần mềm đáp ứng các yêu cầu chức năng và phi chức năng, mang lại trải nghiệm tốt cho người dùng.
  + **Giảm rủi ro**: Giảm thiểu các rủi ro như mất dữ liệu, lỗi hệ thống, hoặc ảnh hưởng đến uy tín của tổ chức.
  + **Tăng độ tin cậy**: Xây dựng niềm tin cho người dùng và khách hàng về tính ổn định và bảo mật của phần mềm.
  + **Tuân thủ yêu cầu**: Đảm bảo phần mềm đáp ứng các tiêu chuẩn pháp lý hoặc quy định của ngành.

**b. Ví dụ thực tế về rủi ro phần mềm nếu không thực hiện testing**  
Ví dụ: Một ứng dụng ngân hàng trực tuyến không được kiểm thử kỹ lưỡng có thể cho phép chuyển tiền vượt quá số dư tài khoản. Nếu không phát hiện lỗi này, người dùng có thể rút tiền không hợp lệ, gây thiệt hại tài chính lớn cho ngân hàng và làm mất niềm tin của khách hàng.

**2. Phân biệt Testing và Debugging**

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

* **Testing**: Là quá trình thực hiện các kiểm tra để tìm ra lỗi hoặc xác minh rằng hệ thống hoạt động đúng như yêu cầu. Testing tập trung vào việc phát hiện các vấn đề, không sửa chữa chúng.
  + Mục tiêu: Xác định lỗi, đảm bảo chất lượng, và xác minh yêu cầu.
  + Người thực hiện: Thường là tester hoặc nhóm QA.
  + Công cụ: Công cụ kiểm thử như Selenium, JUnit, hoặc các kịch bản kiểm thử thủ công.
* **Debugging**: Là quá trình phân tích và sửa chữa lỗi đã được phát hiện trong quá trình testing. Debugging tập trung vào việc tìm nguyên nhân gốc rễ của lỗi và khắc phục nó.
  + Mục tiêu: Xác định và sửa lỗi trong mã nguồn.
  + Người thực hiện: Thường là lập trình viên hoặc nhà phát triển.
  + Công cụ: Công cụ gỡ lỗi như Visual Studio Debugger, Chrome DevTools, hoặc các IDE.

**b. Ví dụ cụ thể về lỗi hệ thống**

* **Lỗi**: Ứng dụng web bị treo khi người dùng nhập dữ liệu không hợp lệ (ví dụ: nhập chuỗi ký tự vào trường số điện thoại).
* **Cách testing phát hiện lỗi**:
  + Tester thiết kế kịch bản kiểm thử để kiểm tra trường nhập số điện thoại với các dữ liệu không hợp lệ (ví dụ: nhập "abc" thay vì số).
  + Thực hiện kịch bản kiểm thử và ghi nhận rằng hệ thống bị treo (crash) thay vì hiển thị thông báo lỗi.
* **Cách debugging giải quyết lỗi**:
  + Lập trình viên sử dụng công cụ gỡ lỗi để theo dõi luồng thực thi của mã nguồn.
  + Phát hiện rằng mã không xử lý trường hợp dữ liệu không hợp lệ (ví dụ: thiếu kiểm tra định dạng đầu vào).
  + Sửa mã bằng cách thêm kiểm tra định dạng (ví dụ: chỉ chấp nhận số) và hiển thị thông báo lỗi thân thiện với người dùng.

**3. Mục tiêu của Testing**

**a. Liệt kê ít nhất 5 mục tiêu chính của testing**

1. **Xác minh yêu cầu**: Đảm bảo phần mềm đáp ứng đầy đủ các yêu cầu chức năng và phi chức năng đã đặt ra.
2. **Đảm bảo chất lượng**: Đảm bảo phần mềm hoạt động ổn định, đáng tin cậy và mang lại trải nghiệm tốt cho người dùng.
3. **Phát hiện lỗi**: Tìm ra các lỗi hoặc khiếm khuyết trong phần mềm trước khi triển khai.
4. **Đánh giá hiệu suất**: Kiểm tra khả năng hoạt động của phần mềm dưới các điều kiện tải khác nhau.
5. **Đảm bảo tính bảo mật**: Xác minh rằng phần mềm được bảo vệ trước các mối đe dọa như tấn công mạng hoặc rò rỉ dữ liệu.

**b. Giải thích một mục tiêu và tầm quan trọng của nó**

* **Mục tiêu được chọn**: Xác minh yêu cầu.
* **Tầm quan trọng**:
  + Xác minh yêu cầu đảm bảo rằng phần mềm đáp ứng đúng những gì khách hàng hoặc người dùng mong đợi, tránh tình trạng sản phẩm không đáp ứng nhu cầu thực tế.
  + Nếu không xác minh yêu cầu, phần mềm có thể hoạt động tốt về mặt kỹ thuật nhưng không đáp ứng mục đích sử dụng, dẫn đến lãng phí thời gian, chi phí và làm mất lòng tin của khách hàng.
  + Ví dụ: Trong một ứng dụng đặt vé máy bay, nếu không xác minh yêu cầu rằng hệ thống chỉ cho phép chọn ngày bay trong tương lai, người dùng có thể đặt vé sai, gây thiệt hại cho cả khách hàng và hãng hàng không.

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

**a. Định nghĩa**

* **Error**: Là sai lầm của con người (thường là lập trình viên) khi viết mã, dẫn đến hành vi không mong muốn của phần mềm.
  + Ví dụ: Lập trình viên viết sai logic kiểm tra dữ liệu đầu vào.
* **Defect**: Là lỗi trong mã nguồn hoặc hệ thống, được phát hiện trong quá trình testing.
  + Ví dụ: Một hàm không xử lý đúng dữ liệu không hợp lệ.
* **Failure**: Là biểu hiện của lỗi khi hệ thống không hoạt động đúng như mong đợi trong môi trường thực tế.
  + Ví dụ: Hệ thống trả về kết quả sai hoặc bị treo khi người dùng nhập dữ liệu.
* **Root Cause**: Là nguyên nhân gốc rễ dẫn đến error, defect hoặc failure.
  + Ví dụ: Thiếu kiểm tra đầu vào hoặc thiết kế không đầy đủ.

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

* **Error**: Lập trình viên sử dụng sai công thức tính toán số tiền (ví dụ: tính thuế sai hoặc không làm tròn số).
* **Defect**: Hàm tính toán số tiền trong mã nguồn không xử lý đúng các trường hợp đặc biệt (ví dụ: số thập phân).
* **Failure**: Người dùng nhận được hóa đơn với số tiền sai (ví dụ: 100.99 USD thay vì 100.00 USD).
* **Root Cause**: Thiếu yêu cầu cụ thể trong tài liệu thiết kế về cách làm tròn số tiền hoặc không kiểm tra đầy đủ các trường hợp tính toán trong giai đoạn phát triển.

**Phần 2: Bài tập thực hành**

**1. Phân tích trường hợp thực tế**

**Ứng dụng đặt vé máy bay gặp lỗi: Hệ thống cho phép đặt vé cho 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 logic của ứng dụng.
* **Defect**: Hàm kiểm tra ngày bay trong mã nguồn không bao gồm điều kiện so sánh ngày chọn với ngày hiện tại.
* **Failure**: Hệ thống cho phép người dùng đặt vé và hoàn tất giao dịch với ngày bay trong quá khứ (ví dụ: ngày 15/10/2024 khi hiện tại là 24/04/2025).
* **Root Cause**: Thiếu yêu cầu rõ ràng trong tài liệu thiết kế rằng hệ thống chỉ cho phép chọn ngày bay trong tương lai, hoặc thiếu kiểm thử cho trường hợp này trong giai đoạn phát triển.

**b. Đề xuất quy trình testing để phát hiện lỗi này trước khi triển khai**

1. **Phân tích yêu cầu**: Xem xét tài liệu yêu cầu để xác minh rằng hệ thống phải từ chối ngày bay trong quá khứ.
2. **Thiết kế kịch bản kiểm thử**: Tạo các test case kiểm tra các trường hợp ngày bay, bao gồm ngày trong quá khứ, ngày hiện tại, và ngày trong tương lai.
3. **Thực hiện kiểm thử thủ công**: Tester nhập các ngày bay không hợp lệ (ví dụ: ngày trong quá khứ) và kiểm tra phản hồi của hệ thống.
4. **Kiểm thử tự động**: Sử dụng công cụ như Selenium để tự động kiểm tra nhiều trường hợp ngày bay khác nhau.
5. **Kiểm thử tích hợp**: Đảm bảo rằng module chọn ngày bay tương tác đúng với các module khác (như thanh toán).
6. **Kiểm thử hồi quy**: Chạy lại các kịch bản kiểm thử sau mỗi lần cập nhật mã nguồn để đảm bảo lỗi không tái diễn.
7. **Báo cáo và sửa lỗi**: Ghi nhận các lỗi phát hiện được và làm việc với lập trình viên để sửa chữa trước khi triển khai.

**2. Thiết kế kịch bản kiểm thử**

**Ứng dụng đặt vé máy bay - Tính năng chọn ngày bay**

Dưới đây là 5 kịch bản kiểm thử (test cases) để kiểm tra tính năng chọn ngày bay:

1. **Kịch bản 1: Kiểm tra chọn ngày bay trong quá khứ**
   * **Mô tả**: Kiểm tra phản hồi của hệ thống khi người dùng 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 đặt vé.
2. **Kịch bản 2: Kiểm tra chọn ngày bay là ngày hiện tại**
   * **Mô tả**: Kiểm tra xem hệ thống có cho phép đặt vé cho ngày hiện tại hay không.
   * **Dữ liệu đầu vào**: Ngày 24/04/2025 (ngày hiện tại).
   * **Kết quả mong đợi**: Hệ thống hiển thị thông báo: "Không thể đặt vé cho ngày hiện tại, vui lòng chọn ngày trong tương lai" hoặc cho phép đặt vé nếu yêu cầu cho phép.
3. **Kịch bản 3: Kiểm tra chọn ngày bay trong tương lai hợp lệ**
   * **Mô tả**: Kiểm tra xem hệ thống xử lý đúng khi chọn ngày bay trong tương lai.
   * **Dữ liệu đầu vào**: Ngày 01/05/2025.
   * **Kết quả mong đợi**: Hệ thống chấp nhận ngày bay và cho phép tiếp tục quá trình đặt vé.
4. **Kịch bản 4: Kiểm tra chọn ngày bay ngoài phạm vi cho phép**
   * **Mô tả**: Kiểm tra phản hồi khi chọn ngày bay quá xa trong tương lai (ví dụ: vượt quá 1 năm).
   * **Dữ liệu đầu vào**: Ngày 01/05/2026.
   * **Kết quả mong đợi**: Hệ thống hiển thị thông báo lỗi: "Không thể đặt vé cho ngày quá xa trong tương lai" và không cho phép tiếp tục.
5. **Kịch bản 5: Kiểm tra chọn ngày không hợp lệ (định dạng sai)**
   * **Mô tả**: Kiểm tra phản hồi của hệ thống khi nhập định dạng ngày không hợp lệ.
   * **Dữ liệu đầu vào**: Nhập "abc" vào trường ngày bay.
   * **Kết quả mong đợi**: Hệ thống hiển thị thông báo lỗi: "Định dạng ngày không hợp lệ, vui lòng nhập theo định dạng DD/MM/YYYY".

**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**

1. **Đặt vé sai thời điểm**: Người dùng có thể đặt vé cho ngày trong quá khứ, dẫn đến giao dịch không hợp lệ, gây thiệt hại tài chính hoặc mất uy tín của hãng hàng không.
2. **Tải hệ thống không cần thiết**: Hệ thống có thể xử lý các giao dịch không hợp lệ, gây lãng phí tài nguyên hoặc làm chậm hiệu suất.
3. **Trải nghiệm người dùng kém**: Người dùng gặp lỗi hoặc không nhận được thông báo rõ ràng, dẫn đến sự không hài lòng và giảm lòng tin vào ứng dụng.

**b. Đề xuất các mục tiêu testing cụ thể để giảm thiểu rủi ro**

1. **Xác minh rằng hệ thống từ chối các ngày không hợp lệ**: Đảm bảo hệ thống ngăn chặn việc chọn ngày trong quá khứ hoặc định dạng sai, giảm nguy cơ giao dịch không hợp lệ.
2. **Đảm bảo thông báo lỗi rõ ràng**: Kiểm tra rằng hệ thống hiển thị thông báo lỗi dễ hiểu khi người dùng nhập ngày không hợp lệ, cải thiện trải nghiệm người dùng.
3. **Kiểm tra hiệu suất của tính năng chọn ngày**: Đảm bảo tính năng hoạt động nhanh và ổn định dưới các điều kiện tải khác nhau, tránh lãng phí tài nguyên hệ thống.