Gideon Clottey

(202289192)

ENGI 9839 Assignment 2

Q1a)

In order to assess a fully integrated software system against the established criteria, system testing is a crucial validation step in the Software Development Life Cycle (SDLC). After unit and integration testing, it is carried out at the top of the testing pyramid to make sure that all modules and subsystems work properly together in a production-like setting.

Main Objectives of System Testing are as follows:

- **Validation of End-to-End System Functionality**
  Verifying that the system functions as intended when all of its parts are integrated is the main goal. This involves confirming that user interactions through interfaces (such as GUIs or APIs) cause the appropriate internal processes to be triggered and produce the appropriate outputs.
- **Verification Against Functional and Non-Functional Requirements**
  Testing the system guarantees that it satisfies both functional (like checking out a book or logging in) and non-functional (like system dependability, security, and performance under load) requirements. Usually specified at the start of the project, these requirements serve as the foundation for creating system test cases.
- **Detection of Interface and Integration Defects**
  System tests evaluate the entire system as a black box through its public interfaces in order to detect interaction flaws, such as timing problems, misuse of the interface, or misinterpretation of data contracts, whereas unit tests validate specific classes or methods in isolation.
- **Support for Acceptance and Deployment Readiness**
  System testing assists in verifying that the program is reliable and functional enough to move forward with production deployment or user acceptance testing (UAT). Prior to the system being made available to end users, this stage serves as the last quality check.

Three Distinct Types of System Tests with Examples

The following lists the three main types of system testing and provides examples relevant to web-based library management systems (LMS):

1. **Functional Testing**
   Verifying that the system satisfies its functional requirements and carries out the planned tasks as delineated during the design phase is the main goal of this kind of testing. "Does the system work as promised?" is addressed. Business logic, user interactions, input/output processing, and data flows are frequently examined during functional testing.

Example: Testing the “Search Book” feature to ensure a user can search by title, author, or ISBN and receive accurate results.

Test Case: Input: "Harry Potter"; Expected output: All matching books with title “Harry Potter” are listed.

2. **Performance Testing:**
   Assesses how the library system responds to different load scenarios, such as when several users are working at once. Important parameters like throughput, response time, and server resource usage are tracked. For example, we could use Apache JMeter to simulate 1000 concurrent visitors trying to check out a book. All queries should be answered by the system in less than two seconds, and CPU and memory use should be kept within reasonable bounds. Testing for load, stress, and soak would assist guarantee the system's scalability and resilience to stress?

   Example: Checking if the system can handle 1000 concurrent users searching for books or borrowing books simultaneously.

   Test Case: Simulate 1000 users borrowing different books at once; Expected result: All requests are completed within acceptable response time (e.g., under 2 seconds).

3. **Security Testing:**

   The goal of security testing is to confirm that a system's built-in defenses against intrusions, breaches, and other vulnerabilities are effective. It seeks to detect any dangers and guarantee the security of user information and sensitive data.

   Example: Testing that a user cannot access admin features like deleting books without admin credentials.

   Test Case: A regular user tries to access /admin/deleteBook?id=1001; Expected result: “Access Denied” message or redirection to login.

Q1b)

System Under Test (SUT):
Web-based Library Management System

Feature Being Validated:
Borrowing a book by a registered user.

Test Scenario: Borrow a book successfully.

Assumptions:

- User is registered and logged in.
- Book exists in the system and is available for borrowing.
- User has not exceeded the borrowing limit.

Preconditions:

- User account exists: User123
- Book available: "Clean Code" with Book ID: B101
- User is logged in and on the book details page.

Test Steps & Expected Results:

Step	Action	Expected Result
1	Navigate to the "Search" section of the library system	The search interface is displayed.
2	Enter "Clean Code " in the search bar and click "Search"	Search results display "Clean Code" with its details (Author, ISBN, and Availability: Available).
3	Click on the book to view details	The detailed book page is displayed, showing full information and a "Borrow" button.
4	Click on "Borrow" button	System checks availability
5	Confirm borrowing	A confirmation message appears, e.g., "Clean Code." The book's status in the system updates from "Available" to "Borrowed" by "User123". The book is added to " User123"'s list of borrowed books
6	View borrowed books page	Book B101 appears in user's borrowed list with the borrowing date and due date.
7	Attempt to borrow again	The search results show the book's status as "Borrowed" or "Unavailable" in the general catalog.

System Test Scenario: Unsuccessful Book Borrowing (Book Unavailable)

Assumptions:

- The user has a valid, active account in the library system and has logged in.
- The specific book (identified by ISBN/Book ID) is not available for borrowing (e.g., already borrowed by another user).
- The system's database and network connectivity are stable and operational.

Preconditions:

- User account exists: Jane Smith
- Book available: " Clean Code " with Book ID: B101

Step	Action	Expected Result
1	Navigate to the "Search" section of the library system	The search interface is displayed.
2	Enter " Clean Code " in the search bar and click "Search"	Search results display " Clean Code " with its details (Author, ISBN, and Availability: Borrowed/Unavailable).
3	Click on the " Clean Code " book title to view its detailed page.	The detailed book page is displayed, showing full information. The "Borrow" button is either absent, disabled, or replaced with a message like "Currently Unavailable" or "Already Borrowed".
4	Attempt to click on a disabled "Borrow" button or an equivalent action if available	The system displays an error message or a notification indicating that the book is

		currently unavailable for borrowing, e.g., "This book is currently unavailable. Please check back later or place a hold.” The book's status in the system remains "Borrowed" or "Unavailable" and is not added to "Jane Smith"'s borrowed books.
5	Navigate to "My Account" or "Borrowed Books" section for "Jane Smith".	“Clean Code” is not listed under “Jane Smith’s” borrowed books.

Q1c)

By comparing the behavior and performance of the integrated system to these predetermined standards, system testing plays a crucial role in confirming that software satisfies both functional and non-functional requirements.

Functional Requirements:

These describe the "what" of the system the precise functions, features, and behaviors that it must have in order to satisfy user needs. They specify how the system ought to react to different inputs in various scenarios.

Example (Library Management System):

Requirement	Testing Approach
User Login – Users should be able to log in with valid credentials.	Create test cases with correct and incorrect credentials. Confirm successful login redirects to the dashboard, and failed login shows an error like “Invalid password.”
Book Search – Users should be able to search for books by title, author, or ISBN.	Search for different keywords and verify if the correct results are displayed. Check edge cases like partial matches and no results.
Borrow Book – Registered users should be able to borrow available books.	Simulate the borrowing process. Validate that the book status updates to "borrowed" and appears in the user’s loan history.
Add New Book (Librarian Only) – Admins should be able to add new books.	Log in as a librarian, access the “Add Book” form, submit valid data, and verify that the book appears in the catalog. Also test that non-admins cannot access this feature.

Non-Functional Requirements:

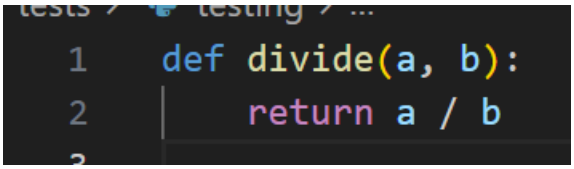
These specify the features, limitations, and characteristics of the system, emphasizing "how" the system functions, including scalability, performance, security, dependability, and usability. Even if they have nothing to do with a particular function, they still affect the user experience.

Example (Library Management System):

Sn	Requirement Type	Requirement	Testing Approach
1	Performance	The system must return search results within **2 seconds** for up to **100 concurrent users**.	Use tools like JMeter or Locust to simulate user load. Measure response time and ensure it stays under the threshold.
2	Security	All login credentials must be **encrypted** during transmission and storage.	Use penetration testing or packet sniffers (e.g., Wireshark) to verify encrypted transmission. Inspect database to ensure passwords are hashed (e.g., using SHA-256).
3	Usability	The book borrowing interface must be intuitive for users with **basic computer skills**.	Conduct usability testing with novice users. Observe their interaction flow, collect feedback, and analyze metrics like click count and task completion time.
4	Reliability	The system should have **zero crashes** during extended use over **24 hours**.	Run soak tests where the system is used continuously, simulating user behavior to check for memory leaks or failures.

Q2a)

Given code

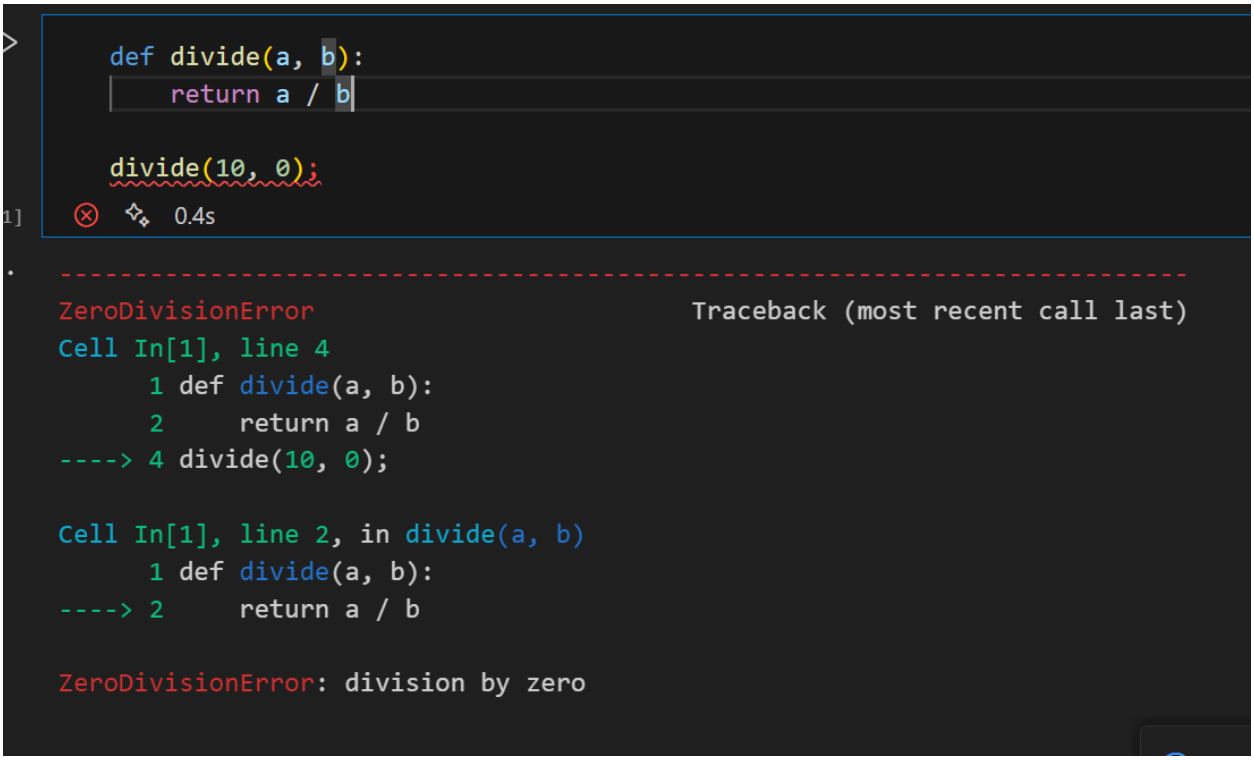


Identified Issues:

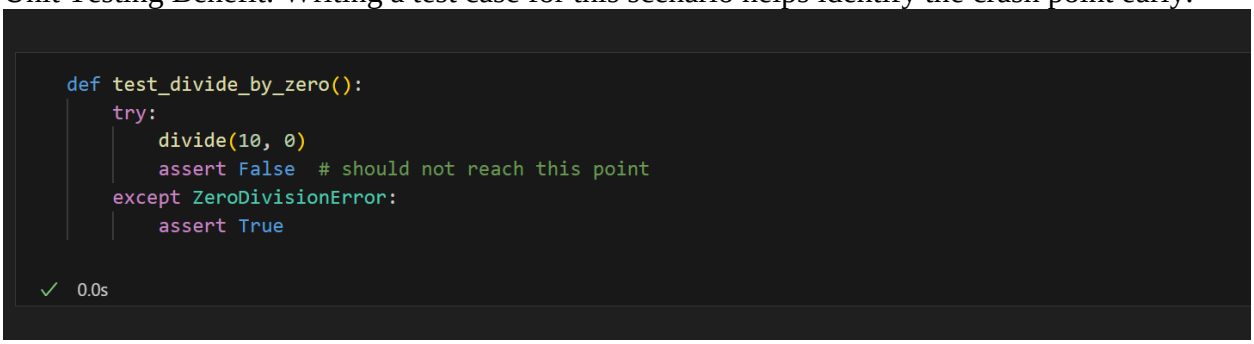
1. **Division by Zero**

Any number divided by zero in Python results in a ZeroDivisionError. The application may crash or behave strangely during runtime if the system is unable to handle this issue.

Example Error:
divide(10, 0) → ZeroDivisionError: division by zero



Unit Testing Benefit: Writing a test case for this scenario helps identify the crash point early:



2. Non-Numeric Input Types
Explanation: If either a or b is a non-numeric value like a string, the function raises a TypeError.

```
divide("10", 2)
```

0.0s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 divide("10", 2)

Cell In[1], line 2, in divide(a, b)
      1 def divide(a, b):
----> 2     return a / b

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Unit Testing Benefit: A test case for type validation ensures robustness:

```
def test_divide_string_input():
    try:
        divide("10", 2)
        assert False
    except TypeError:
        assert True
```

0.0s

Improved Version of the Code:

```python
def divide(a, b):
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        return "Error: Both inputs must be numeric."
    if b == 0:
        return "Error: Division by zero is not allowed."
    return a / b
divide("10", 2)
```

27] ✓ 0.0s

```
'Error: Both inputs must be numeric.'
```

```python
divide(10, 0)
```

28] ✓ 0.0s

```
'Error: Division by zero is not allowed.'
```

Why These Changes Matter:

- Type Checking: Prevents improper data types that could crash the function.
- Zero Check: Prevents division errors before they occur.
- User Feedback: Returns friendly error messages, improving usability and debugging.
- Improved Robustness: Function is now more resilient and suitable for use in larger applications or APIs.

Q2b)

Test Case 1: Successful Payment Transaction

Objective: Ensure that a payment is processed successfully when correct details are provided.

Preconditions:

- User is logged into the system (UserID: 1001)
- Shopping cart contains items worth $50.
- Valid Visa credit card saved to user profile.
- Network connectivity is stable.
- The product is in stock.

Assumptions:

- Payment gateway is available and operational.
- User's card has sufficient funds and is not expired.
- System is correctly integrated with payment API.
- The e-commerce system is integrated with a payment gateway.
- The payment gateway is capable of processing the specified card type.
- The user's account is in good standing.

Steps	Actions	Expected results
1	User navigates to the shopping cart	Cart contents are displayed with total amount: $50
2	User clicks "Checkout"	Redirected to checkout page
3	User selects saved Visa card ending in 1234	Payment form auto-fills with saved card details
4	User confirms billing address	Billing details are shown correctly
5	User clicks "Pay Now"	Payment request sent to payment gateway
6	System receives approval from payment gateway	Success response received
7	System confirms payment	Message shown: "Your payment of $50.00 was successful!"
8	System redirects to "Order Confirmation" page	Page shows order summary and transaction ID
9	System sends receipt to user's email	User receives confirmation email with receipt and order details
10	User navigates to order history	Order is listed under recent purchases

Test Case 2: Failed Payment Due to Invalid Card Details

Objective: Verify that the system handles failed payments correctly and provides appropriate feedback.

Preconditions:

- User is logged into the system.

- Shopping cart contains items worth $75.
- No valid card saved; user will enter card manually.

Assumptions:

- Payment gateway provides appropriate error messages for invalid inputs.
- No changes are made to inventory or order history for failed transactions.

Steps	**Actions**	**Expected Results**
1	User navigates to the shopping cart	Cart contents are displayed with total amount: $75
2	User clicks "Checkout"	Redirected to checkout page
3	User selects "Add New Card" option	Payment form allows manual card input
4	User enters invalid card number 1234 5678 9999 0000 and expiry 12/20	Card info appears on the screen, but expired
5	User clicks "Pay Now"	Payment request sent to gateway
6	System receives rejection from payment gateway	Failure response received
7	System displays payment failure message	Message shown: **"Payment failed: Invalid or expired card. Please try again."**
8	System does not proceed to order confirmation	User remains on the checkout page
9	User given option to re-enter or select different payment method	"Try Again" and "Change Payment Method" options are visible
10	No charges are applied to user’s account	No transactions recorded in user account or order history