Software Testing Plan
Library Management System

Meghana N Naik – 2023BCSE07AED584
CSE General

1. Introduction

The Library Management System (LMS) is a software application developed to automate and manage library activities such as book cataloging, member registration, book issue and return, fine calculation, and report generation. Since the system plays a critical role in daily library operations and data handling, systematic testing is essential to ensure reliability, accuracy, security, and performance.

This Software Testing Plan describes the overall testing strategy, objectives, testing environment, test cases, and defect management process for the Library Management System. The primary goal of this plan is to identify defects early, verify that the system meets user requirements, and ensure that the final product is of high quality and ready for deployment.

2. Test Strategy

The test strategy defines the levels of testing, testing techniques, and responsibilities involved in testing the Library Management System. Multiple testing levels are used to ensure complete system validation.

2.1 Unit Testing

Unit testing focuses on testing individual modules of the Library Management System independently to verify that each component works correctly.

Examples of unit testing include:

- Testing the login module for valid and invalid usernames and passwords.
- Testing the book addition module to ensure book details are saved correctly.
- Testing fine calculation logic for overdue books.

Unit testing is usually performed by developers using test cases and automated testing tools wherever possible.

2.2 Integration Testing

Integration testing verifies the interaction between different modules of the system after unit testing is completed.

Examples include:

- Integration between user management and book issue modules.
- Integration between book return and fine calculation modules.
- Integration between database and reporting modules.

The objective is to detect interface defects and data flow issues between integrated components.

2.3 System Testing

System testing evaluates the complete Library Management System as a whole to ensure it meets the specified functional and non-functional requirements.

System testing includes:

- Functional testing of all features such as search, issue, return, and reports.
- Performance testing to verify response time during multiple user access.
- Security testing to ensure only authorized users can access sensitive data.
- Usability testing to verify ease of use for librarians and users.

System testing is carried out in an environment similar to the real production environment.

2.4 Acceptance Testing

Acceptance testing is performed to validate the system against user expectations and requirements.

- Conducted by library staff or end users.
- Ensures the system meets business needs.
- Confirms that workflows such as book issue, return, and report generation work correctly.

Once acceptance testing is successful, the system is approved for deployment.

3. Test Environment

The test environment defines the hardware, software, and test data required to execute testing activities.

3.1 Hardware Requirements

- Computer systems with minimum 4 GB RAM
- Processor: Intel i3 or above
- Hard disk space: Minimum 20 GB
- Network connectivity for database access

3.2 Software Requirements

- Operating System: Windows / Linux
- Database: MySQL / Oracle
- Front-end: HTML, CSS, JavaScript
- Back-end: Java / Python / PHP
- Web Browser: Chrome, Firefox
- Testing Tools: Manual testing tools or automation tools if required

3.3 Test Data Requirements

- Sample book records with different categories
- Sample user accounts (students, staff, librarian)
- Issue and return records
- Fine calculation data for overdue scenarios

Test data should represent real-life usage to ensure accurate testing results.

4. Test Cases

Test cases are designed to verify system functionality based on inputs, expected outputs, and pass/fail criteria.

4.1 Sample Test Case Structure

- Test Case ID
- Test Description
- Input Data
- Expected Output
- Actual Output
- Pass/Fail Status

4.2 Example Test Cases

Test Case 1: Login Functionality

- Input: Valid username and password
- Expected Output: Successful login and dashboard display
- Pass Criteria: User is redirected to dashboard

Test Case 2: Book Issue

- Input: Valid book ID and user ID
- Expected Output: Book issued successfully and database updated
- Pass Criteria: Issue record created

Test Case 3: Book Return

- Input: Returned book after due date
- Expected Output: Fine calculated correctly
- Pass Criteria: Fine displayed accurately

Proper documentation of test cases helps ensure repeatable and consistent testing.

5. Defect Management

Defect management is the process of identifying, recording, tracking, and resolving defects found during testing.

5.1 Defect Logging

- Defects are logged using a defect tracking tool or spreadsheet.
- Each defect includes ID, description, severity, priority, and status.
- Screenshots or logs are attached when necessary.

5.2 Defect Classification

Defects are classified based on severity:

- Critical – System crashes or data loss
- Major – Core functionality failure
- Minor – UI or non-critical issues
- Trivial – Cosmetic issues

5.3 Defect Resolution

- Defects are assigned to developers for fixing.
- Fixed defects are re-tested to verify resolution.
- Regression testing is performed to ensure fixes do not introduce new defects.