Testing Plan Document

for

MindMeter

at The Department of Electrical and Computer Engineering,

The University of the West Indies

St Augustine Campus

Trinidad

Version 0.1

Prepared by:

Raja Bissessar

CSL Group 4

23 March 2024

Based on the ISO/IEC/IEEE Std 29148:2018

Document Control

Title: CSL Group 4 MindMeter Testing Plan Document.

Version: 0.1

Date: 23/03/2024

Author(s): Raja Bissessar

Document Signoff

|  |  |  |  |
| --- | --- | --- | --- |
| Nature of Signoff | Person | Signature with Date | Role |
| Author  Reviewers |  |  | Project Manager, Requirements Analyst and Consultant  QA Officer |

Document Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Change Details |
| 20 Mar 2024 | 0.0 | Raja Bissessar | Initial template commit. |
| 23 Mar 2024 | 0.1 | Raja Bissessar | Completed all sections. |

Table of Contents

[1. Introduction 5](#_Toc162448332)

[1.1 Objectives 5](#_Toc162448333)

[1.2 Scope 5](#_Toc162448334)

[1.3 Major Constraints 5](#_Toc162448335)

[2. Test Plan 6](#_Toc162448336)

[2.1 Testing Strategy 6](#_Toc162448337)

[2.1.1 Unit 6](#_Toc162448338)

[2.1.2 Integration Testing 6](#_Toc162448339)

[2.1.3 Black Box Testing 7](#_Toc162448340)

[2.1.4 White Box Testing 8](#_Toc162448341)

[2.2 Test Resources and Staffing 10](#_Toc162448342)

[2.3 Test Work Products 11](#_Toc162448343)

[2.4 Test Record Keeping 11](#_Toc162448344)

[2.5 Test Metrics 11](#_Toc162448345)

[2.6 Testing Tools and Environment 11](#_Toc162448346)

[2.6.1 Python 12](#_Toc162448347)

[2.6.2 Discord 12](#_Toc162448348)

[2.6.3 GitHub 12](#_Toc162448349)

[2.6.4 Visual Studio 12](#_Toc162448350)

[2.6.5 Microsoft Word 12](#_Toc162448351)

[2.7 Test Schedule 13](#_Toc162448352)

List of Tables

Table 1: Describing the Unique Test Cases 14

Table 2: System Testing Details 15

Table 3: Unit Test Details 16

Table 4 Testing Engineer Information 18

# Introduction

Testing is an integral phase in software development projects, ensuring the reliability, functionality, and performance of software’s. This document outlines the testing approach, strategies, and methodologies employed to guarantee the delivery of a high-quality product. Through meticulous testing, the aim is to identify and rectify defects, validate requirements, and ultimately provide the user with a seamless and error-free software experience. This concise guide serves as a roadmap for the testing phase, with emphasis on the commitment to deliver a robust and reliable software solution.

## Objectives

* Requirement Analysis.
* Identifying the process for testing.
* Developing test cases based on analysis.
* List the test details.
* Describe the testing strategies implemented.

## Scope

This test plan document will outline the testing approaches taken to test the MindMeter application. The functionality and testing procedures will be implemented with respect to the requirements outlined for the application and the beneficiary’s needs will be prioritized.

## Major Constraints

Constraints generally refer to limitations or restrictions that can affect the design and development of a software design project. This project was mainly affected by time and scope constraints factors. This project was allotted a period of 10 weeks for development, documentation and deployment. However, despite the time constraints, the features and functionalities to be included in this application were met in its entirety.

# Test Plan

## Testing Strategy

This document was derived from the functionality, reliability and performance requirements outlined in the System Requirement specification (SyRS) document. The testing methods utilized in this document to meet the customer’s requirements were as follows:

* Unit Testing
* Integration Testing
* Black Box Testing
* White Box Testing
* Performance Testing

### Unit

Unit testing is a software testing technique where individual units or components of a software application are tested in isolation. The goal of this technique is to validate that each unit of the software performs as designed. A “unit” refers to the smallest testable part of an application such as a function, method, or procedure.

### Integration Testing

Integration testing is the process of testing the interface between two software units or modules. The aim of this testing technique is to identify any problems or bugs that arise when different components are combined with each other to determine the correctness of the interface and expose the faults in the interaction between the integrated units. Integration testing is only complete once all the modules have been unit tested. There are four types of integration testing.

* Big-Bang Integration Testing - This is where all the modules are combined, and the functionality is verified after the completion of individual module testing. Hence, all the modules of the system are put together and tested. This approach is applicable to small systems because it is difficult to locate an error as the error may belong to any of the modules being integrated.
* Bottom-Up Integration Testing – This is where each module at the lower level is tested with higher modules until all modules are tested. The aim of this technique is to test the interface among various modules making up the subsystem.
* Top-Down Integration Testing – This is where the behaviour of the lower-level modules is simulated before they are integrated. This form of testing uses the top to bottom approach. First, high level modules are tested and then low-level modules. Low-level modules are then integrated to high level to ensure that the system is working.
* Mixed Integration Testing – This testing is also known as sandwiched testing and involves a combination of top down and bottom-up testing approaches. This approach overcomes the shortcoming of top-down and bottom-up approaches.

### Black Box Testing

Black-box testing is a software testing method where the tester assesses the functionality of a system without delving into its internal workings. Instead, the focus is on verifying that the software meets specified requirements or specifications. It can be performed through various methods:

* Syntax-Driven Testing - This is applied to systems represented by a language, such as compilers using context-free grammar. During this process, test cases are systematically generated to ensure the utilization of each grammar rule at least once.
* Equivalence Partitioning-This testing involves grouping similar inputs to streamline testing. By identifying equivalence classes, one input from each group is tested, covering both valid and invalid scenarios in the process.
* Boundary Value Analysis – This method focuses on scrutinizing input boundaries, recognizing that errors often manifest at these critical points. It includes testing not only the typical valid and invalid inputs but also values at the edges of the input domain.
* Cause-Effect Graphing: This technique establishes a connection between logical inputs (causes) and their corresponding actions (effects). It also employs Boolean graphs to visually represent the relationships, and the process further entails transforming the graph into a decision table. Subsequently, the decision table rules are converted into practical test cases for comprehensive testing.
* Requirement-based Testing - Includes validating the software system requirements specified in the Software Requirements Specification (SRS).
* Compatibility Testing - This considers various infrastructure parameters, such as processor type, architecture, and operating system (OS). The objective is to verify that the software functions seamlessly across diverse environments, accommodating different processor types, architectural configurations, and operating systems. This ensures the software's adaptability and performance across a range of system setups.

Black box testing is characterized by several key features. It involves independent testing conducted by individuals not involved in the application development, ensuring unbiased assessments from a user's perspective. Testers lack access to internal code, enabling a focus on evaluating external behaviour and functionality. This testing approach is typically requirement-based, ensuring adherence to specified specifications, and employs diverse techniques such as functional, usability, acceptance, and regression testing. Black box testing is easily automated with various tools, reducing testing time and effort, and its scalability can be adjusted based on the application's size and complexity. Testers possess limited knowledge of the application, enhancing the realism of assessments to mimic end users' interactions.

**Advantages of Black Box Testing**:

* No extensive functional knowledge or programming skills are required for implementation.
* Efficient for testing in larger systems.
* Tests are conducted from the user's or client's perspective.
* Test cases can be easily reproduced.
* Useful for identifying ambiguity and contradictions in functional specifications.

**Disadvantages of Black Box Testing**:

* Possibility of repeating the same tests during the testing process.
* Difficulty in implementing test cases without clear functional specifications.
* Execution of test cases is challenging due to complex inputs at different testing stages.
* Sometimes, the reason for test failure cannot be identified.
* Certain programs within the application may go untested.
* Does not reveal errors in the control structure.
* Working with a large sample space of inputs can be exhaustive and time-consuming.

### White Box Testing

White box testing, also known as glass box testing, clear box testing, or structural testing, is a software testing approach that scrutinizes not only the functionality but also the internal structures such as data structures, design, and code of a software application. In contrast to black box testing, where the focus is solely on functionality, white box testing provides the tester with access to the source code. Leveraging this access, testers can design test cases to examine the software's internal logic, flow, and structure, ensuring that code paths align with specified requirements. This testing method, also termed transparent or open box testing, facilitates a thorough examination of the software at the code level. Its operation follows the following process:

* Input: Requirements, functional specifications, design documents, source code.
* Processing: Conducting risk analysis to navigate the entire process.
* Thorough Test Planning: Developing test cases to encompass the entire code. Iteratively executing and refining until error-free software is achieved. Communication of results is also undertaken.
* Output: Compiling a comprehensive final report summarizing the entire testing process.

**Testing Techniques**:

* Statement Coverage involves traversing all statements at least once, ensuring that every line of code or node in a flowchart is tested. This technique is valuable for pinpointing faulty code, and a minimum of 2 test cases is required to cover all nodes.
* Branch Coverage focuses on designing test cases to traverse each branch from all decision points at least once. In a flowchart, this means covering all edges. A minimum of 4 test cases is necessary to ensure all branches are covered.
* Condition Coverage requires the coverage of all individual conditions, while Multiple Condition Coverage tests all possible combinations of outcomes at least once.
* Basis Path Testing involves creating control flow graphs from code or a flowchart and calculating Cyclomatic complexity to determine the number of independent paths. This guides the design of the minimal number of test cases for each independent path.
* Loop Testing is crucial for the thorough examination of loops, which are fundamental to many algorithms. For simple loops, test cases include scenarios like skipping the loop entirely, one pass, two passes, and m passes, where m is less than n. Nested loops are tested from the innermost loop outward, and concatenated loops are treated like independent or nested loops based on their relationship.

**Characteristics of White Box Testing**:

* Code Coverage Assessment: White box testing facilitates the examination of code coverage within an application, aiding in the detection of areas that lack testing.
* Access to Source Code: Conducting white box testing necessitates access to the source code of the application, enabling the testing of individual functions, methods, and modules.
* Proficiency in Programming Languages: Testers engaged in white box testing must possess knowledge of programming languages such as Java, C++, Python, and PHP to comprehend code structure and formulate tests.
* Detection of Logical Errors: White box testing is instrumental in pinpointing logical errors in the code, such as infinite loops or incorrect conditional statements.
* Integration Testing: White box testing proves valuable for integration testing, enabling the verification of seamless interaction among different components of an application.
* Unit Testing: White box testing is employed for unit testing, scrutinizing individual units of code to ensure their correct functionality.
* Code Optimization: White box testing aids in the optimization of code by identifying performance issues, redundant code, or areas that can be enhanced.
* Security Testing: White box testing can be utilized for security testing, enabling testers to uncover vulnerabilities in the application's code.

Advantages:

* Thorough examination of entire code and structures.
* Optimizes code and removes errors.
* Can start at an earlier stage in the Software Development Life Cycle.
* Easy to automate.
* Facilitates easy code optimization.
* Identifies defects not detected by other testing techniques.
* Enables creation of comprehensive test cases covering all code paths.
* Ensures adherence to coding standards and performance optimization.

Disadvantages:

* Expensive.
* Requires rewriting test cases during code redesign.
* Testers need in-depth knowledge of code and programming language.
* May lead to a focus on internal workings, potentially missing external issues.
* Risk of biased views due to familiarity with the code.
* Missing functionalities may not be detected.
* Complex and not always realistic.
* Higher chances of errors in production.

## Test Resources and Staffing

To ensure thorough testing of the MindMeter application, an array of test resources and a dedicated testing team were employed. The team collaborated closely throughout the testing phase, leveraging their expertise to conduct comprehensive evaluations.

The testing was exclusively performed on laptops and computers, the primary platforms for the MindMeter.

In addition to the testing team, various tools were utilized to facilitate the testing process. These tools included laptops and computers for the execution of tests, the MindMeter application installed, tabulated test results for efficient tracking, and an internal text file for documentation purposes.

The comprehensive nature of the testing team, combined with the diverse set of tools, guaranteed a meticulous and systematic testing approach. The focus was on validating the functionality and reliability of the MindMeter, aligning with the overall objective of delivering a robust application.

## Test Work Products

A meticulously crafted test plan served as the guiding framework for the testing team. This plan facilitated an organized and systematic approach, preventing redundant testing and enhancing the efficiency of the testing phase. Test scenarios were thoughtfully designed to be feasible, testable, and aligned with the application's specific requirements. While most tests were directly linked to these requirements, additional tests were conducted to validate the overall functionality of the application.

## Test Record Keeping

To maintain a comprehensive record of the testing process, testing results were diligently documented in a "Testing Summary" document. This document included tabulated descriptions and analyses of each test, offering valuable insights into the performance of the MindMeter. Additional testing data, such as the date a test was performed, was also added to the documentation.

## Test Metrics

Test metrics for the MindMeter project were established to gauge the success or failure of tests. These metrics were complemented by detailed comments providing insights into the nuances of test results. The combination of status indicators and comments ensured an extensive understanding of the application's performance during testing.

## Testing Tools and Environment

The tests were conducted on MindMeter v0.3. The tools and software/environments employed for evaluating the application's functionality are outlined below.

Tools:

1. Laptops and Computers
2. Installed MindMeter application
3. Tabulated test results
4. Internal text file

Software/Environments:

1. Python (Programming Language)
2. Discord (Task tracking, collaboration, and communication)
3. GitHub (Version control and distribution of the application)
4. Visual Studio (IDE for application development and testing)
5. Microsoft Word (Documentation and analysis of test results)

Details on Specific Tools and Software:

### Python

Python was chosen as the programming language for developing the MindMeter, emphasizing its versatility and suitability for desktop friendly applications.

### Discord

Discord played a pivotal role in facilitating task tracking, collaboration, and communication among the testing team. Discord served as the virtual meeting point for testers to discuss testing procedures and activities.

### GitHub

GitHub played a crucial role in version control, ensuring the most up-to-date version of the desktop application was distributed to multiple team members for testing.

### Visual Studio

Visual Studio served as the Integrated Development Environment (IDE) for developing and testing the MindMeter application. It supported Python programming, debugging, and provided tools for comprehensive testing.

### Microsoft Word

Microsoft Word played an instrumental role in the documentation process. It was used for tabulating, discussing, and analyzing test results, making documents shareable across multiple devices, and enhancing group collaboration and productivity.

## Test Schedule

The comprehensive testing schedule for MindMeter (V0.3) incorporated a variety of testing methodologies to ensure the application's robustness and reliability.

Unit testing played a central role, allowing specific components of the application to undergo rigorous testing for errors and ensuring their individual functionality. This focused approach was instrumental in early error detection and promoting the independent performance of each unit.

Integration testing was another crucial facet, evaluating the functionality and seamless integration of larger interconnected components within the MindMeter. The methodology employed, whether top-down, sandwich, or bottom-up, was determined by the flow of one component to the next, ensuring optimal cohesion.

Primarily, black box testing took precedence, enabling testers to evaluate the application from an end-user's perspective. This approach aimed at enhancing the overall user experience by validating the application's adherence to specified requirements and identifying potential issues users might encounter during interaction.

In specific cases, white box testing was applied to scrutinize specific routes of the application, delving into the code's internal structure. This method helped identify bugs or logic errors, contributing to overall code quality improvement.

Each test case was meticulously described and indexed with a unique code, as shown in Table 1 below, facilitating a structured and systematic testing process. This approach ensured a nuanced understanding of the application's performance, ultimately meeting the highest standards of quality and reliability.

Table 1: Describing the Unique Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **Test Case** | **Test Case Type** |
| TCN-01 | Verify that the application displays all tabs (General, Stress test, Depression test, Self-Esteem test, Preview, Database). | Unit |
| TCN-02 | Confirm that the application interface is user-friendly and easy to interpret. | Unit |
| TCN-03 | Ensure that each test provides instructions for users to read and navigate the app and answer the test questions appropriately. | Unit |
| TCN-04 | Check if the database is accessible from the main page. | Integration (Bottom-Up) |
| TCN-05 | Verify that all student data is shown in the database. | Integration (Bottom-Up) |
| TCN-06 | Ensure that navigation between the different functionalities of the application is possible. | Integration (Top-Down) |
| TCN-07 | Complete an assessment and ensure that the application performs an overall tally of the test results and displays a score based on the student’s answers. | Unit |
| TCN-08 | Test if the application stores past tests completed by students as well as their general information by searching for a student’s name for an assessment previously completed. | Integration (Bottom-Up) |
| TCN-09 | Print a test previously completed to confirm that the print function is working. | Unit |
| TCN-10 | Attempt to enter invalid information to confirm that the database only stores valid information. | Unit |
| TCN-11 | Fill in information in general tab and then check if the preview tab reflects the information filled in the general tab. | Integration (Top-Down) |
| TCN-12 | Attempt to do more than one test at a time to confirm that the users are not able to do more than one form without having a stored file in the database. | Integration (Sandwich) |

Table 2: System Testing Details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MindMeter Testing** | | | | |
| **Functional**  **Requirements**  **#ID** | **Functional and Non-Functional Requirement/Unit Case** | **Priority** | **Test Case #ID** | **Testing Type** |
| R.1 | Includes General, Stress test, Depression test, Self-Esteem test, preview and database tabs. | Must Have | TCN-01 | Unit |
| R.2 | User friendly and easy to interpret. | Should Have | TCN-02 | Unit |
| R.3 | Contains instructions for users to read and navigate the app. | Should Have | TCN-03 | Unit |
| R.4 | Database accessible from the main page | Should Have | TCN-04 | Integration (Bottom-Up) |
| R.5 | Database shows all student data | Must Have | TCN-05 | Integration (Bottom-Up) |
| R.6 | Database shows the relevant form done by the student | Must Have | TCN-08 | Integration (Bottom-Up) |
| R.7 | Performs an overall tally of the test results | Must Have | TCN-07 | Unit |
| R.8 | Database stores data that can be found by a search | Must Have | TCN-08 | Integration (Bottom-Up) |
| R.9 | Database allows user to print data. | Should Have | TCN-09 | Unit |
| R.10 | The database only stores valid information | Should Have | TCN-10 | Unit |
| R.11 | The preview tab depends on the information filled in the general tab. | Must Have | TCN-11 | Integration (Top-Down) |
| R.12 | The code implemented for tallying the scores in the database depends on the options the student chooses | Must Have | TCN-07 | Unit |
| R.13 | The users are not able to do more than one form without having a stored file in the database | Must Have | TCN-12 | Integration (Sandwich) |

Table 3: Unit Test Details

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Test Case** | **Pre- Conditions** | **Test Steps** | **Test Data** | **Results** |
| TCN-01 | Verify that the application displays all tabs (General, Stress test, Depression test, Self-Esteem test, Preview, Database). | Application is installed and launched | Open the application and check the tabs. | N/A | All tabs are displayed. |
| TCN-02 | Confirm that the application interface is user-friendly and easy to interpret. | Application is installed and launched | Navigate through the application ensuring all buttons work and look aesthetically pleasing. | N/A | The application interface is user-friendly and easy to interpret. |
| TCN-03 | Ensure that each test provides instructions for users to read and navigate the app and answer the test questions appropriately. | Application is installed and launched | Open each test and read the instructions. | N/A | Each test provides clear instructions. |
| TCN-04 | Check if the database is accessible from the main page. | Application is installed and launched | Try to access the database from the main page. | N/A | The database is accessible from the main page. |
| TCN-05 | Verify that all student data is shown in the database. | Application is installed and launched | Open the database and check the student data. | N/A | All student data is shown in the database. |
| TCN-06 | Ensure that navigation between the different functionalities of the application is possible. | Application is installed and launched | Try to navigate between different functionalities. | N/A | Navigation between different functionalities is possible. |
| TCN-07 | Complete an assessment and ensure that the application performs an overall tally of the test results and displays a score based on the student’s answers. | Application is installed and launched | Complete a known negative test and check the results. Complete a known positive test and check the results. | A known positive test and a known negative test. | The application performs an overall tally of the test results and displays a score. (Positive for the known positive test and negative for the known negative test) |
| TCN-08 | Test if the application stores past tests completed by students as well as their general information by searching for a student’s name for an assessment previously completed. | Application is installed and launched | Search for a student’s name and check the past tests. | Student’s name and a completed assessment for them in the system | The application stores past tests and general information. |
| TCN-09 | Print a test previously completed to confirm that the print function is working. | Application is installed and launched | Try to print a completed test. | N/A | The print function works correctly and the test prints successfully. |
| TCN-10 | Attempt to enter invalid information to confirm that the database only stores valid information. | Application is installed and launched | Try to enter invalid information into the database. | Invalid information | The database rejects invalid information. |
| TCN-11 | Fill in information in general tab and then check if the preview tab reflects the information filled in the general tab. | Application is installed and launched | Fill in information in the general tab and then check the preview tab. | Information to be filled in | The preview tab reflects the information filled in the general tab. |
| TCN-12 | Attempt to do more than one test at a time to confirm that the users are not able to do more than one form without having a stored file in the database. | Application is installed and launched | Try to do more than one test at a time. | N/A | Users are not able to do more than one form without having a stored file in the database. |