**Test Plan**

**for**

**Personal Budget Manager Application**

**Version 1.3**

**Prepared by**

|  |  |  |
| --- | --- | --- |
| **Name** | **Student ID** | **e-mail** |
| **Danny Shash** | **29548912** | **danny.shash@gmail.com** |
| **Ganji Amarendher** | **25764246** | **a\_ganji@encs.concordia.ca** |
| **Siming Huang** | **40081588** | **hsim47@163.com** |
| **Tony Lac** | **40049123** | **tony.lac@polymtl.ca** |
| **Xinjie Zeng** | **27238223** | **xinjiezeng@gmail.com** |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 31/01/2019 | 1.0 | Test plan for increment 1 | Xinjie Zeng |
| 18/03/2019 | 1.1 | Test plan for increment 2 | Xinjie Zeng |
| 03/04/2019 | 1.3 | Test plan for increment 3 | Xinjie Zeng |

TABLE OF CONTENTS

[1. Introduction 3](#_Toc5181428)

[1.1 Purpose 3](#_Toc5181429)

[1.2 Scope 3](#_Toc5181430)

[1.3 Document Terminology and Acronyms 3](#_Toc5181431)

[1.3.1 Definitions 3](#_Toc5181432)

[1.3.2 Acronyms 3](#_Toc5181433)

[2. Target Test Items 3](#_Toc5181434)

[2.1 Unit Testing 3](#_Toc5181435)

[2.2 Integration Testing 4](#_Toc5181436)

[2.3 Function Testing 4](#_Toc5181437)

[2.4 User Interface Testing 4](#_Toc5181438)

[2.5 Configuration Testing 4](#_Toc5181439)

[3. Test Approach 5](#_Toc5181440)

[3.1 Unit Testing 5](#_Toc5181441)

[3.1.1 Function add 5](#_Toc5181442)

[3.2.1 Function remove 5](#_Toc5181443)

[3.3.1 Function modify payment status 5](#_Toc5181444)

[3.2 Integration Testing 5](#_Toc5181445)

[3.2.1 userInterface panel 5](#_Toc5181446)

[3.2.2 addCompositeExpense panel 5](#_Toc5181447)

[3.2.3 addExpense panel 6](#_Toc5181448)

[3.3 Function Testing 6](#_Toc5181449)

[3.3.1 Add an expense 6](#_Toc5181450)

[3.3.2 Create Composite Expense 9](#_Toc5181451)

[3.3.3 Mark Expense Paid/Unpaid 10](#_Toc5181452)

[3.3.4 Remove Expense 15](#_Toc5181453)

[3.3.5 Hide/Show Paid Expenses 17](#_Toc5181454)

[3.4 User Interface Testing 21](#_Toc5181455)

[3.5 Configuration Testing 21](#_Toc5181456)

[4. Testing Workflow 21](#_Toc5181457)

[5. Iteration Milestones 21](#_Toc5181458)

# **1.** **Introduction**

The primary goal of this project is to develop a personal budget manager application, which users can manage and track their personal expenses. This is the final phase of the project, which contains a comprehensive list of tests that will be performed along with a workflow of how the tests will be executed.

## **1.1 Purpose**

The purpose of the test plan is to gather all of the information necessary to plan and control the test effort for this phase. The Test Plan document supports the following objectives:

* + List the recommended test requirements
  + Describe the testing strategies and approaches to be employed
  + Describe the workflow of the testing process that must be executed
  + Provide a timeline with milestones for the testing phase

## **1.2** **Scope**

This test plan is to test the Personal Budget Manager Application. The test plan will cover unit, integration, function, and user interface testing. Testing techniques that will be performed include white box, black box testing as well as boundary testing. A test plan workflow will also be included along with milestones for this phase.

## **1.3 Document Terminology and Acronyms**

**1.3.1 Definitions**

Purchase A type of day-to-day expense

Bill A type of recurring expense

**1.3.2 Acronyms**

PBM Personal Budget Manager Application

SRS Software Requirement Specification

SDD Software Design Document

# **2. Target Test Items**

In this section, we will list all the target test items and the detailed test plans.

## **2.1 Unit Testing**

Unit testing consists of testing different units of the system. We test classes and methods in isolation using white box and black box techniques. The list of test items for unit testing will not cover all the classes and methods. It will focus on classes and methods that implement major functions (please refer to the design document). Below is a list of the test items:

* Function add
* Function remove
* Function modify payment status

## **2.2 Integration Testing**

We will test components (models, views and controller) separately, and then integrating them together and test it again.

## **2.3 Function Testing**

It will consist of all the requirements and specifications in the SRS. We include detailed test cases for all the functionalities. Below is the list of functions that were tested:

* Add a purchase
* Add a bill
* Add a composite purchase
* Add a composite bill
* Remove a purchase
* Remove a bill
* Remove a composite purchase
* Remove a composite bill
* Modify payment status
* Show/hide the expense
* View all expenses on the major panel

## **2.4 User Interface Testing**

User interface testing is to make sure the user interface works as required in the software requirement document and software design document. For the user interface, the possible interactions will be tested in great detail. The user interface will be covered include:

* addCompositeExpense panel
* addExpense panel
* userInterface panel

## **2.5 Configuration Testing**

Configuration testing makes sure the PBM application runs successfully in different environment configurations. We have tested the PBM application under different operation system:

* Windows
* Mac

# **3. Test Approach**

The test approach describes the strategies to design and implement the tests. In this section, we will describe the details of the tests for each target test item in section 3.

## **3.1 Unit Testing**

### 3.1.1 Function add

3.1.1.1 Black Box Testing

3.1.1.2 White Box Testing

### 3.2.1 Function remove

3.2.1.1 Black Box Testing

3.2.1.2 White Box Testing

### 3.3.1 Function modify payment status

3.3.1.1 Black Box Testing

3.3.1.2 White Box Testing

## **3.2 Integration Testing**

The purpose of the integration testing is to ensure the proper navigation of PBM application and ease of use for users. We will navigate through window to window, verify key and mouse movement. For each integration test, we design several test cases. In each test case, exactly one new component will be analyzed.

### 3.2.1 userInterface panel

|  |  |
| --- | --- |
| Test Case 1 | Initialize the window |
| Test Case Description | To test if the window can initialize normally  This test case should be done when we run the code |
| Test Result | OK |

|  |  |
| --- | --- |
| Test Case 2 | Open addCompositeExpense panel |
| Test Case Description | Select multiple purchases or bills (not both) and click the Create Composite Expense button to test if the addCompositeExpense panel can be opened. |
| Test Result | OK |

|  |  |
| --- | --- |
| Test Case 3 | Open addExpense panel |
| Test Case Description | Click the Add Expense button to test if the addExpense panel can be opened. |
| Test Result | OK |

### 3.2.2 addCompositeExpense panel

|  |  |
| --- | --- |
| Test Case 1 | show new composite expense on userInterface panel |
| Test Case Description | The new composite expense should be displayed on the userInterface panel when user clicks Add Expense button on addCompositeExpense panel. |
| Test Result | OK |

|  |  |
| --- | --- |
| Test Case 2 | Close addCompositeExpense panel |
| Test Case Description | The addCompositeExpense panel should be closed automatically when user clicks Add Expense button on addCompositeExpense panel. |
| Test Result | OK |

### 3.2.3 addExpense panel

|  |  |
| --- | --- |
| Test Case 1 | show new expense on userInterface panel |
| Test Case Description | The new expense should be displayed on the userInterface panel when user clicks Add Expense button on addExpense panel. |
| Test Result | OK |

|  |  |
| --- | --- |
| Test Case 2 | Close addExpense panel |
| Test Case Description | The addExpense panel should be closed automatically when user clicks Add Expense button on addExpense panel. |
| Test Result | OK |

## **3.3 Function Testing**

This section tests the functions of the software. Each requirement is associated with a set of test cases, with valid data and invalid data.

### 3.3.1 Add an expense

|  |  |
| --- | --- |
| Test case | Add a purchase |
| Test Case Description | 1. open the app  2. click “Add Expense” button on main panel  3. type in info  4. press “Add Expense button” on Add Expense panel |
| Test data | Type: purchase  Date: 2018-07-19  Name: candy  Amount: 2.62  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | store the info in the data base and show it on the main panel |
| Actual result | Successfully stores the info in the data base and update it on the main panel |

|  |  |
| --- | --- |
| Test case | Add a bill |
| Test Case Description | 1. open the app  2. click “Add Expense” button on main panel  3. type in info  4. press “Add Expense button” on Add Expense panel |
| Test data | Type: bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: unpaid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly |
| Expected result | store the info in the data base and update it on the main panel |
| Actual result | Successfully stores the info in the data base and update it on the main panel |

|  |  |
| --- | --- |
| Test case | Add a purchase (invalid date) |
| Test Case Description | 1. open the app  2. click “Add Expense” button on main panel  3. type in info  4. press “Add Expense button” on Add Expense panel |
| Test Data | Type: purchase  Date: dkejide  Name: candy  Amount: 2.62  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | The “Add Expense” button become unclickable. There is a red sentence besides the Date to ask user to type in date in correct format like 2019-09-09. If the info is correct, “Add Expense” button will become clickable and the red word disappears. |
| Actual result | Successfully implement the above scenario |

|  |  |
| --- | --- |
| Test case | Add a purchase (invalid amount) |
| Test Case Description | 1. open the app  2. click “Add Expense” button on main panel  3. type in info  4. press “Add Expense button” on Add Expense panel |
| Test data | Type: purchase  Date: 2019-03-07  Name: candy  Amount: dfsdf  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | The “Add Expense” button become unclickable. There is a red sentence besides the amount to ask user to type in correct format like 74.55. If the info is correct, “Add Expense” button will become clickable and the red word disappears. |
| Actual result | Successfully implement the above scenario |

|  |  |
| --- | --- |
| Test case | Add a bill (invalid date) |
| Test Case Description | 1. open the app  2. click “Add Expense” button on main panel  3. type in info  4. press “Add Expense button” on Add Expense panel |
| Test data | Type: bill  Date: sdfsdfew  Name: jenny  Amount: 50  Status: unpaid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly |
| Expected result | The “Add Expense” button become unclickable. There is a red sentence besides the Date to ask user to type in date in correct format like 2019-09-09. If the info is correct, “Add Expense” button will become clickable and the red word disappears. |
| Actual result | Successfully implement the above scenario |

|  |  |
| --- | --- |
| Test case | Add a bill (invalid amount) |
| Test Case Description | 1. open the app  2. click “Add Expense” button on main panel  3. type in info  4. press “Add Expense button” on Add Expense panel |
| Test data | Type: purchase  Date: 2019-03-07  Name: candy  Amount: dfsdf  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | The “Add Expense” button become unclickable. There is a red sentence besides the amount to ask user to type in correct format like 74.55. If the info is correct, “Add Expense” button will become clickable and the red word disappears. |
| Actual result | Sucessfully implement the above scenario |

### 3.3.2 Create Composite Expense

|  |  |
| --- | --- |
| Test case | Create a composite bill |
| test steps | 1. open the app  2. select multiple bills on the main panel  3. click the “Create Composite Expense” button |
| Test Case Description | Type: bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: unpaid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly  Type: bill  Date: 2019-01-01  Name: Gym  Amount: 750  Status: paid  Vendor Name: ABC Fitness  Location:  Method:  Category: Default  Due date: 2019-04-01  Interval: Monthly |
| Expected result | These two bills are shown under the composite bill on the panel. |
| Actual result | Successfully create composite bill |

|  |  |
| --- | --- |
| Test case | Create a composite purchase |
| Test Case Description | 1. open the app  2. select multiple purchases on the main panel  3. click the “Create Composite Expense” button |
| Test data | Type: purchase  Date: 2019-03-07  Name: candy  Amount: 30.5  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food  Type: purchase  Date: 2019-01-27  Name: cakes  Amount: cocobun  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | These two purchases are shown under the composite purchase on the panel. |
| Actual result | Successfully creates a composite purchase |

### 3.3.3 Mark Expense Paid/Unpaid

|  |  |
| --- | --- |
| Test Case | Mark a purchase unpaid to paid |
| Test Case Description | 1. open the app  2. select an expense  3. click “Mark Expense Paid/Unpaid” button |
| Test data | Type: purchase  Date: 2019-03-18  Name: gas  Amount: 62.94  Status: unpaid  Method: credit  Vendor: Petrol Canada  Location:  Category: default  Due Date: 2019-03-28 |
| Expected result | The purchase status changed from unpaid to paid. |
| Actual result | Successfully updated the purchase status to paid. |

|  |  |
| --- | --- |
| Test Case | Mark a purchase paid to unpaid |
| Test Case Description | 1. open the app  2. select an expense  3. click “Mark Expense Paid/Unpaid” button |
| Test data | Type: purchase  Date: 2019-03-18  Name: gas  Amount: 62.94  Status: paid  Method: credit  Vendor: Petrol Canada  Location:  Category: default  Due Date: 2019-03-28 |
| Expected result | The purchase status changed from paid to unpaid. |
| Actual result | Successfully updated the purchase status to unpaid. |

|  |  |
| --- | --- |
| Test case | Mark a bill unpaid to paid |
| Test Case Description | 1. open the app  2. select an expense  3. click “Mark Expense Paid/Unpaid” button |
| Test data | Type: bill  Date: 2019-01-18  Name: Electricity  Amount: 576.93  Status: unpaid  Method:  Vendor: Hydro Quebec  Location:  Category: default  Due Date: 2019-03-18  Interval: Quarterly |
| Expected result | The bill status changed from unpaid to paid. |
| Actual result | Successfully updated the bill status to paid. |

|  |  |
| --- | --- |
| Test Case | Mark a bill paid to unpaid |
| Test Case Description | 1. open the app  2. select an expense  3. click “Mark Expense Paid/Unpaid” button |
| Test data | Type: bill  Date: 2019-03-18  Name: gas  Amount: 62.94  Status: paid  Method: credit  Vendor: Petrol Canada  Location:  Category: default  Due Date: 2019-03-28  Interval: Monthly |
| Expected result | The purchase status changed from paid to unpaid. |
| Actual result | Successfully updated the purchase status to unpaid. |

|  |  |
| --- | --- |
| Test Case | mark a composite purchase unpaid to paid |
| Test Case Description | 1. open the app  2. select composite purchase on the main panel  3. click the “Mark Expense Paid/Unpaid” button |
| Test data | Type: composite purchase  Date: 2019-03-07  Name: candy  Amount: 30.5  Status: unpaid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food  Date: 2019-01-27  Name: cakes  Amount: cocobun  Status: unpaid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | The status of two purchases shown under the composite purchase will be updated to paid. |
| Actual result | The status of two purchases shown under the composite purchase update successfully.  The status of composite purchase updates successfully. |

|  |  |
| --- | --- |
| Test Case | mark a composite purchase paid to unpaid |
| Test Case Description | 1. open the app  2. select composite purchase on the main panel  3. click the “Mark Expense Paid/Unpaid” button |
| Test data | Type: composite purchase  Date: 2019-03-07  Name: candy  Amount: 30.5  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food  Date: 2019-01-27  Name: cakes  Amount: cocobun  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | The status of two purchases shown under the composite purchase will be updated to unpaid. |
| Actual result | The status of two purchases shown under the composite purchase update successfully.  The status of composite purchase updates successfully. |

|  |  |
| --- | --- |
| Test Case | mark a composite bill unpaid to paid |
| Test Case Description | 1. open the app  2. select composite bill on the main panel  3. click the “Mark Expense Paid/Unpaid” button |
| Test data | Type: composite bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: unpaid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly  Date: 2019-01-01  Name: Gym  Amount: 750  Status: unpaid  Vendor Name: ABC Fitness  Location:  Method:  Category: Default  Due date: 2019-04-01  Interval: Monthly |
| Expected result | The status of two bills shown under the composite bills will be updated to paid. |
| Actual result | The status of two bills shown under the composite bill update successfully.  The status of composite bill updates successfully. |

|  |  |
| --- | --- |
| Test Case | mark a composite bill paid to unpaid |
| Test Case Description | 1. open the app  2. select composite bill on the main panel  3. click the “Mark Expense Paid/Unpaid” button |
| Test data | Type: composite bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: paid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly  Date: 2019-01-01  Name: Gym  Amount: 750  Status: paid  Vendor Name: ABC Fitness  Location:  Method:  Category: Default  Due date: 2019-04-01  Interval: Monthly |
| Expected result | The status of two bills shown under the composite bills will be updated to unpaid. |
| Actual result | The status of bills shown under the composite bill update successfully.  The status of composite bill updates successfully. |

### 3.3.4 Remove Expense

|  |  |
| --- | --- |
| Test Case | Remove a bill |
| Test Case Description | 1. open the app  2. select a bill on the main panel  2. click “Remove Expense” button on the main panel |
| Test data | Type: bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: unpaid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly |
| Expected result | data is successfully removed from the main panel |
| Actual result | Data is successfully removed from the main panel |

|  |  |
| --- | --- |
| Test Case | Remove a purchase |
| Test Case Description | 1. open the app  2. select a purchase on the main panel  2. click “Remove Expense” button on the main panel |
| Test data | Type: purchase  Date: 2018-07-19  Name: candy  Amount: 2.62  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | data is successfully removed from the main panel |
| Actual result | Data is successfully removed from the main panel |

|  |  |
| --- | --- |
| Test scenario | Remove a composite bill |
| Test Case Description | 1. open the app  2. select a composite bill  3. click the “Remove Expense” button |
| Test data | Type: composite bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: paid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly  Date: 2019-01-01  Name: Gym  Amount: 750  Status: paid  Vendor Name: ABC Fitness  Location:  Method:  Category: Default  Due date: 2019-04-01  Interval: Monthly |
| Expected result | These two bills should be removed from the main panel |
| Actual result | The composite bill is removed successfully. |

|  |  |
| --- | --- |
| Test Case | Remove a composite purchase |
| Test Case Description | 1. open the app  2. select composite purchase on the main panel  3. click the “Remove Expense” button |
| Test data | Type: composite purchase  Date: 2019-03-07  Name: candy  Amount: 30.5  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food  Date: 2019-01-27  Name: cakes  Amount: cocobun  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food |
| Expected result | The two purchases will be both removed from the main panel |
| Actual result | The composite purchase is removed successfully. |

### 3.3.5 Hide/Show Paid Expenses

|  |  |
| --- | --- |
| Test Case | Hide Paid Purchase |
| Test Case Description | 1. open the app  2. click “Hide/Show Paid Expenses” button on the main panel |
| Test data | Type: purchase  Date: 2018-07-19  Name: candy  Amount: 2.62  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food  Type: purchase  Date: 2018-03-09  Name: cakes  Amount: 5.5  Status: unpaid  Method: credit  Vendor Name: cocobun  Location: Downtown  Category: Food  Type: purchase  Date: 2018-02-19  Name: groceries  Amount: 100  Status: unpaid  Method: credit  Vendor Name: super c  Location: Downtown  Category: Food |
| Expected result | Paid purchases should be hidden from the main panel |
| Actual result | Paid purchases are successfully hidden from the main panel |

|  |  |
| --- | --- |
| Test Case | show Paid Purchase |
| Test Case Description | 1. open the app  2. click “Hide/Show Paid Expenses” button on the main panel to hide the paid purchase  3. click “Hide/Show Paid Expenses” button on the main panel to show the paid purchase again |
| Test data | Type: purchase  Date: 2018-07-19  Name: candy  Amount: 2.62  Status: paid  Method: debit  Vendor Name: Tim Hortons  Location: Downtown  Category: Food  Type: purchase  Date: 2018-03-09  Name: cakes  Amount: 5.5  Status: unpaid  Method: credit  Vendor Name: cocobun  Location: Downtown  Category: Food  Type: purchase  Date: 2018-02-19  Name: groceries  Amount: 100  Status: unpaid  Method: credit  Vendor Name: super c  Location: Downtown  Category: Food |
| Expected result | Paid data should be reappeared on the main panel |
| Actual result | Paid data is successfully reappeared on the main panel |

|  |  |
| --- | --- |
| Test Case | Hide Paid bill |
| Test Case Description | 1. open the app  2. click “Hide/Show Paid Expenses” button on the main panel |
| Test data | Type: bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: unpaid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly  Type: bill  Date: 2018-08-19  Name: electricity  Amount: 150  Status: unpaid  Vendor Name: Hydro Quebec  Location:  Method: credit  Category: default  Due date: 2018-11-19  Interval: Quarterly  Type: bill  Date: 2019-01-19  Name: parking  Amount: 150  Status: paid  Vendor Name: Indigo  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-02-19  Interval: Monthly |
| Expected result | Paid bills should be hidden from the main panel |
| Actual result | Paid bills are successfully hidden from the main panel |

|  |  |
| --- | --- |
| Test Case | show Paid Purchase |
| Test Case Description | 1. open the app  2. click “Hide/Show Paid Expenses” button on the main panel to hide the paid bills  3. click “Hide/Show Paid Expenses” button on the main panel to show the paid bills again |
| Test data | Type: bill  Date: 2018-08-19  Name: jenny  Amount: 50  Status: unpaid  Vendor Name: Fido  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-09-19  Interval: Monthly  Type: bill  Date: 2018-08-19  Name: electricity  Amount: 150  Status: unpaid  Vendor Name: Hydro Quebec  Location:  Method: credit  Category: default  Due date: 2018-11-19  Interval: Quarterly  Type: bill  Date: 2019-01-19  Name: parking  Amount: 150  Status: paid  Vendor Name: Indigo  Location: Downtown  Method: credit  Category: Utilities  Due date: 2019-02-19  Interval: Monthly |
| Expected result | Paid bills should be reappeared on the main panel |
| Actual result | Paid bills are successfully reappeared on the main panel |

## **3.4 User Interface Testing**

This section is to test the user interface described in the design.

## **3.5 Configuration Testing**

This section is to test the PBM application under different environment configurations the users may have.

|  |  |
| --- | --- |
| Test Case | Windows |
| Test Case Description | To ensure that the PBM application runs properly under Windows |
| Input | 1. Copy the PBM application and all files needed to execute it on Windows. 2. Re-test the integration tests in 3.2 3. Re-test the function tests in 3.3 4. Re-test the user interface tests in 3.4 |

|  |  |
| --- | --- |
| Test Case | Mac |
| Test Case Description | To ensure that the PBM application runs properly under Mac |
| Input | 1. Copy the PBM application and all files needed to execute it on Windows 2. Re-test the integration tests in 3.2 3. Re-test the function tests in 3.3 4. Re-test the user interface tests in 3.4 |

# **4. Testing Workflow**

# **5. Iteration Milestones**

The following are the milestones that were set in this iteration.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | January | February | March | April |
|  |  |  |  |  |