Test Plan

for

Personal Budget Manager Application

Version 1.4

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 Siming Huang Tony Lac Danny Shash

TABLE OF CONTENTS

List of Figures

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

1.4 References

- Pressman, Roger S. Software Engineering: A Practitioner's Approach. 5th ed. Toronto: McGraw-Hill, 2001.
- Dr. Nora Houari, "COMP 354 Software Engineering – Validation and Verification, Testing" https://moodle.concordia.ca/moodle/pluginfile.php/3554573/mod_label/intro/se-vv-test.pdf (Current April 1, 2018)

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 Model

3.1.1.1 CompositeBillTest

3.1.1.1.1 add Function

3.1.1.1.1.1 Black Box Testing

Name:	Danny		Test Date:	April 11,2019
Class name:	compositeBillTest	Method name:	testAdd()	File Name:
				compositeBillTest.java
Objects:	CompositeBill comp_bill	List items		
	Expense b1			
Test Case:	comp_bill.add(b1)			
Excepted Output:	Expense b1 should be added to items Iteration and int noOfSubIteam increases by 1			
Actual Outpit:	Expense b1 is added to items Iteration and int noOfSubIteam increases by 1			
Bugs?"	No			

3.1.1.1.1.2 White Box Testing

```
public void add(Expense expense) {
```

```
    expense.setParent(this);                                  1

    items.add(expense);                                       2

    this.setNoOfSubItems(this.getNoOfSubItems()+1);           3
```

```
}
```

Path 1	1-2-3

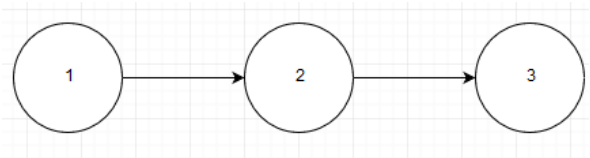


Figure 1. Path diagram for function: add

3.1.1.1.2 remove Function

3.1.1.1.2.1 Black Box Testing

Name:	Danny		Test Date:	April 11,2019
Class name:	compositeBillTest	Method name:	testRemove()	File Name:
				compositeBillTest.java
Objects:	CompositeBill comp_bill	List items		
	Expense b1	Expense b2		
Test Case:	comp_bill.add(b1)	comp_bill.add(b2)	comp_bill.add(b1)	
Excepted Output:	Expense b1 should be added to items Iteration and int noOfSubIteam increases by 1	Expense b2 should be added to items Iteration and int noOfSubIteam increases by 1	Expense b1 should be removed from items Iteration and int noOfSubIteam decreases by 1	
Actual Outpit:	Expense b1 is added to items Iteration and int noOfSubIteam increases by 1	Expense b2 is added to items Iteration and int noOfSubIteam increases by 1	Expense b1 is removed from items Iteration and int noOfSubIteam decreases by 1	
Bugs?"	No	No	No	

3.1.1.1.2.2White Box Testing

```
public boolean remove(Expense expense) {

        if(items.size()>0) {                                   1

                items.remove(expense);
                this.setNoOfSubItems(this.getNoOfSubItems()-1);   2

                return true;

        }
        return false;                                          3
}
```

Path 1	1-3
Path 2	1-2

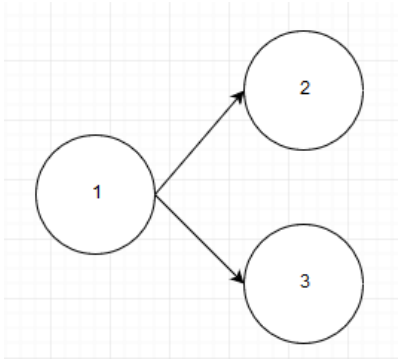


Figure 2. Path diagram for function: remove

3.1.1.1.3 getBillsList Function

3.1.1.1.3.1 Black Box Testing

Class name:	compositeBillTest	Method name:	testGetBillsList()	File Name:			
				compositeBill Test.java			
Objects:	CompositeBill comp_bill	List items					
	Expense b1	Expense b2					
	Expense b3	Expense b4					
	CompositeBill test						
Test Case:	comp_bill.add(b1)	comp_bill.add(b2)	comp_bill.add(test)	test.add(b3)	test.add(b3)	comp_bill.getBills List()	test.getBillsLi st()
Excepted Output:	Expense b1 should be added to comp_bill as a child	Expense b2 should be added to comp_bill as a child	CompositeBill test should be added to comp_bill as a child	Expense b3 should be added to test as a child	Expense b3 is added to test as a child	5 Bill expenses should be children of comp_bill	2 Bill expenses should be children of test
Actual Outpit:	Expense b1 is added to comp_bill as a child	Expense b2 is added to comp_bill as a child	CompositeBill test is added to comp_bill as a child	Expense b3 is added to test as a child	Expense b3 is added to test as a child	5 Bill expenses are children of comp_bill	2 Bill expenses are children of test
Bugs?"	No	No	No	No	No	No	No

3.1.1.1.3.2White Box Testing

```
items=new ArrayList<Expense>()

private ArrayList<Expense> getBillsList(ArrayList<Expense> list) {
```

```
Iterator<Expense> compBill = items.iterator();          1

while(compBill.hasNext())                               2

                {


                Expense e = compBill.next();            3

                list.add(e);

                if(e.getType().ordinal()>1) {           4

                CompositeBill ce= (CompositeBill)e;     5

                ce.getBillsList(list);

        }
        }
    return list;                                        6
}
```

Path 1	1-2-6
Path 2	1-2-3-2(loop)-6
Path 3	1-2-3-4-5-1(loop)-2(loop)-6

Path 1	1-2-6
Objects	List list, Iterator compBill, List items
Result	Only one expense item in the Iterator. Therefore it cannot be a composite. Therefore *return list*.

Path 2	1-2-3-2(loop)-6
Objects	List list, Iterator compBill, List items, Expense e, Expense ce
Result	More than expense item in the Iterator. However, none of them are composites because there *getType* index number is 1 or less.Step 2 is repeated until there are no more objects in the iterator and exits to step 6 and *return list*.

Path 3	1-2-3-4-5-1(loop)-2(loop)-6
Objects	List list, Iterator compBill, List items, Expense e, Expense ce
Result	*CompositeBill ce* is created at step 5, and subsequent Bill *Expenses e* are added to the list. If e.getType is larger is 2+, further *CompositeBill ce* are created and cycle repeats itself from step 1 until step2 cannot be activated and the *return list* is activated

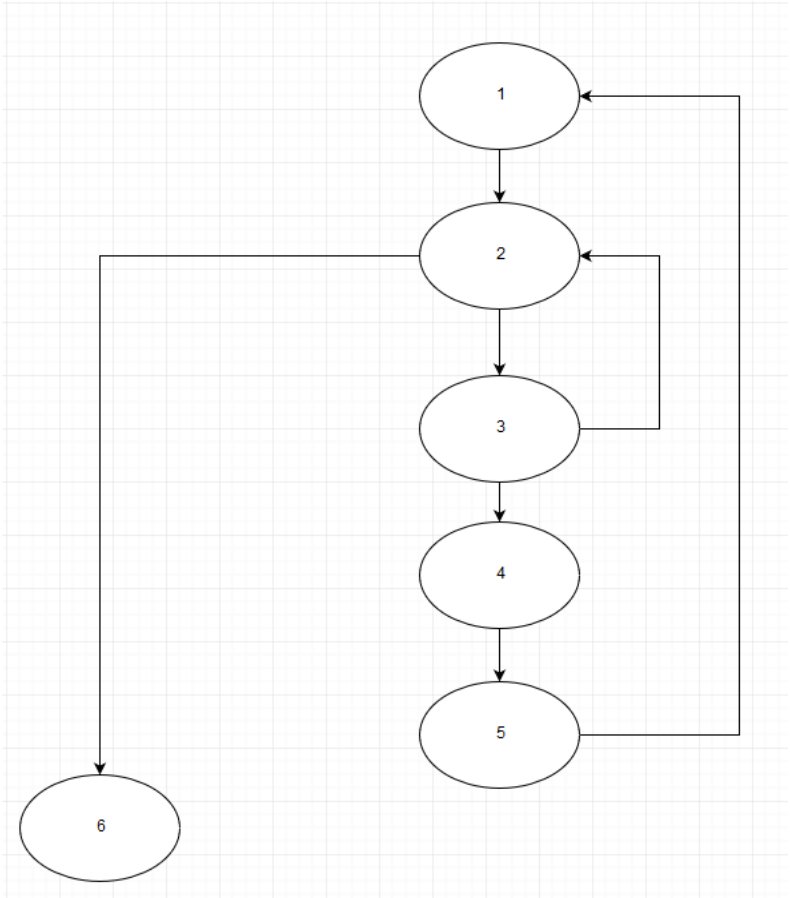


Figure 3. Path diagram for function: getBillsList

3.1.2 View

3.1.2.1 ExpenseObserverImplTest

3.1.2.1.1 testUpdate Function

3.1.2.1.1.1 Black Box Testing

Name:	Danny		Test Date:	April 11,2019
Class name:	ExpenseObserverImplTest	Method name:	testUpdate()	File Name:
				ExpenseObserverImplTest.java
Objects:	ExpenseSubject subject	ExpenseObserver observer		
	List data	Map dataItem		
Variables	int index:	0,1,2,3		
Test Case:	data.get(index)	data.get(index).size()	observer.update(data)	
Excepted Output:	should return Map Expense corresponding to index(key): 0-Purcharse, 1-Bill,2-comp_purchase,3-comp_bill	should return amount of expense correpsonding to Map index	Entire List data viewed by observer should be updated	
Actual Outpit:	returns Map Expense corresponding to index(key): 0-Purcharse, 1-Bill,2-comp_purchase,3-comp_bill	returns amount of expense correpsonding to Map index	Entire List data viewed by observer is updated	
Bugs?"	No	No	No	

3.1.2.1.1.2 White Box Testing

```
public void update(List<Map<ExpenseKey , Expense>> data) {

    this.ldata = data;                                          1

    System.out.println("Data state changed, please refresh the view! items="+
                        ldata.get(0).size()+", " + //purchase
                        ldata.get(1).size()+", " + //bill
                        ldata.get(2).size()+", " + //comp_purchase
                        ldata.get(3).size()        //comp_bill
                        );                                      2

    UserInterface.getInstance().getTableModel().refresh();      3

    subject.resetStateChange();                                 4
```

}

Path 1	1-2-3-4

Path 1	1-2-3-4
Object	List data
Expected Results	Step2:"`Data state changed, please refresh the view! items=2, 3, 1, 1"//Amount of expensese in maps for purchase, bill, comp_purchase and comp_bill are updated in the Idata Arraylist.` Step3: Userinterface table model is updated Step4: if any changes is in the UI Expenses not saved is lost.

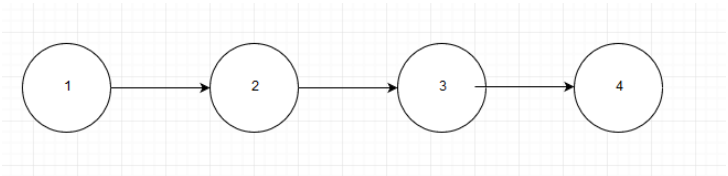


Figure 4. Path diagram for function: update

3.1.2.1.2 testGetData Function

3.1.2.1.2.1 Black Box Testing

Name:	Danny		Test Date:	April 11,2019	
Class nam	ExpenseObserverImplTest	Method name:	testGetData()	File Name:	
				ExpenseObserverImplTest.java	
Objects:	ExpenseSubject subject	ExpenseObserver observer			
	List<Map<ExpenseKey , Expense>> data	Map<ExpenseKey , Expense> dataItem			
	DisplayParameters params				
	Expense p1//purchase expense	Map mp//map of purchase_exp			
	Expense p2//purchase expense	Map cp//values of collection of purchase_exp			
Variables	int index:	0 //Purcharse	String expenseType		
		1 //Bill			
		2 //comp_purchase			
		3 //comp_bill			
Test Case:	data.get(0)	data.get(0).size()	observer.update(data)	params.type = ExpenseType.PURCHASE;	assertEquals(2, observer.getData(params).size());
Excepted Output:	should return "p". Map name of purchase expenses	should return "2" index amount of purchases expenses in map "p"	observer view of data should be updated.	purchase types should only shown to the viewer	Verify that observer is shown the available amount of purchase expenses, which are 2.
Actual Outpit:	return "p". Map name of purchase expenses	returns "2" index amount of purchases expenses in map "p"	observer view of data is updated.	purchase types are only shown to the viewer	Verify that observer is shown the available amount of purchase expenses, which are 2.
Bugs?"	No	No	No	No	No

3.1.2.1.2.2 White Box Testing

```
private static final ExpenseObserver expenseObserver = new ExpenseObserverImpl();
        private ExpenseSubject subject;
        private List<Map<ExpenseKey , Expense>> ldata=null;

public ArrayList<Expense> getData(DisplayParameters params) {

ArrayList<Expense> expList = null;                    1

if(ldata.get(params.type.ordinal()) == null) {
System.out.println("reurning as data is null, refresh");   2
return expList;
                }

switch(params.type.ordinal())                         3
```

```
{
case 0:
        expList = new ArrayList<Expense>();
        ArrayList<Expense> purchase_List =
        new ArrayList<Expense>(ldata.get(0).values());
        expList.addAll(purchase_List);
break;
```

4

```
case 2:
        expList = new ArrayList<Expense>();
        ArrayList<Expense> comp_purchase_List =
        new ArrayList<Expense>(ldata.get(2).values());
        expList.addAll(comp_purchase_List);
        break;
```

5

```
case 1:
        expList = new ArrayList<Expense>();
        ArrayList<Expense> bill_List = new ArrayList<Expense>(ldata.get(1).values());
```

6

```
        expList.addAll(bill_List);
        break;
case 3:
        expList = new ArrayList<Expense>();
        ArrayList<Expense> comp_bill_List =
        new ArrayList<Expense>(ldata.get(3).values());
        expList.addAll(comp_bill_List);
        break;
```

7

```
default:
throw new RuntimeException("getExpenses error");
}
```

8

```
return expList;
```

9

```
}
```

Path 1	1-2-9
Path 2	1-3-8-9
Path 3	1-3-4-9
Path 4	1-3-5-9
Path 5	1-3-6-9
Path 6	1-3-7-9

Path 1	1-2-9
Object	DisplayParameters params, ArrayList expList, ExpenseType type
Expected Results	System.out.println("returning as data is null, refresh");

Path 2	1-3-8-9
Object	DisplayParameters params, ArrayList expList, ExpenseType type
Expected Results	RuntimeException("getExpenses error");

Path 3	1-3-4-9
Object	DisplayParameters params, ArrayList expList, ExpenseType type
Expected Results	return expList.addAll(purchase_List);

Path 4	1-3-5-9
Object	DisplayParameters params, ArrayList expList, ExpenseType type
Expected Results	return expList.addAll(comp_purchase_List);

Path 5	1-3-6-9
Object	DisplayParameters params, ArrayList expList, ExpenseType type
Expected Results	return expList.addAll(bill_List);

Path 6	1-3-6-9
Object	DisplayParameters params, ArrayList expList, ExpenseType type
Expected Results	return expList.addAll(comp_bill_List);

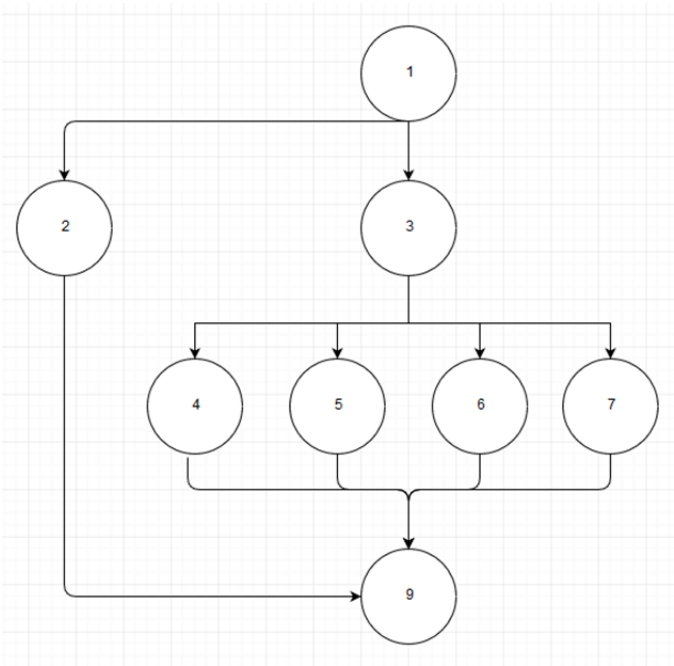


Figure 5. Path diagram for function: getData

3.1.3 Controller

3.1.3.1 InMemoryStoreTest

3.1.3.1.1 put Function

3.1.3.1.1.1 Black Box Testing

Name:	Danny		Test Date:	April 11,2019
Class name:	InMemoryStoreTest	Method name:	put()	File Name:
				InMemoryStoreTest.java
Objects:	InMemoryStore inMemoryStore		Purchase purchase	
	List list		Bill bill	
	Map purchaseMap		CompositeBill compositeBill	
	Map compositePurchase Map		CompositePurchase compositePurchase	
	Map compositeBillMap			
Test Case:	inMemoryStore.put (purchase)	inMemoryStore.put (bill);	inMemoryStore.put(purcharseBill)	inMemoryStore.put(composit ePurchase)
Excepted Output:	purchase should be added	bill should be added	purchaseBill should be added	compositeBill should be added
Actual Outpit:	purchase added	bill added	purchaseBill added	compositeBill added
Bugs?"	No	No	No	No

3.1.3.1.1.2 White Box Testing

```
private List<Map<ExpenseKey , Expense>> expenseData;
private Map<ExpenseKey , Expense> purchases;
private Map<ExpenseKey , Expense> comp_purchases;
private Map<ExpenseKey , Expense> bill;
private Map<ExpenseKey , Expense> comp_bill;


public void put(Expense expense) throws IOException {
```

```
switch(expense.getType().ordinal()){
```

1

```
case 0 : purchases.put(expense.getKey(), expense);
       break;
```

2

```
case 1 : bill.put(expense.getKey(), expense);
       break;
```

3

```
case 2 : comp_purchases.put(expense.getKey(), expense);
```

4

```
            break;
        case 3 : comp_bill.put(expense.getKey(), expense);
            break;                                                  5
        default:
            throw new RuntimeException("Invalid Expense type");     6
        }
}
```

Path 1	1-2
Path 2	1-3
Path 3	1-4
Path 4	1-5
Path 5	1-6

Path 1	1-2
Objects	Expense expense, Map purchases
Result	expense is added to Map purchase with unique key

Path 2	1-3
Objects	Expense expense, Map bill
Result	expense is added to Map bill with unique key

Path 3	1-4
Objects	Expense expense, Map comp_purchases
Result	expense is added to Map comp_purchases with unique key

Path 4	1-5
Objects	Expense expense, Map comp_bill
Result	expense is added to Map comp_bill with unique key

Path 5	1-6
Objects	Expense expense
Result	RuntimeException("Invalid Expense type");

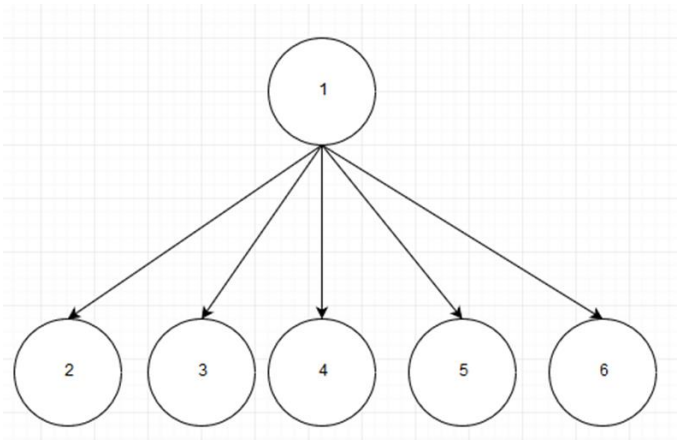


Figure 6. Path diagram for function: put

3.1.3.1.2 remove Function

3.1.3.1.2.1 Black Box Testing

Name:	Danny		Test Date:	April 11,2019
Class name:	InMemoryStoreTest	Method name:	remove()	File Name:
				InMemoryStoreTest.java
Objects:	List list		Purchase purchase	
	InMemoryStore inMemoryStore			
	Map purchaseMap			
Variables	int purchaseMapSize			
Test Case:	inMemoryStore.remove (purchase)			
Excepted Output:	purchase should be removed			
Actual Outpit:	purchase removed			
Bugs?"	No			

3.1.3.1.2.2 White Box Testing

```
private List<Map<ExpenseKey , Expense>> expenseData;
private Map<ExpenseKey , Expense> purchases;
private Map<ExpenseKey , Expense> comp_purchases;
private Map<ExpenseKey , Expense> bill;
private Map<ExpenseKey , Expense> comp_bill;
```

```
public void remove(Expense expense) throws IOException {

    switch(expense.getType().ordinal(){

      case 0 : purchases.remove(expense.getKey(), expense);
             break;

      case 1 : bill.remove(expense.getKey(), expense);
             break;

      case 2 :comp_purchases.remove(expense.getKey(), expense);
                    break;
```

1

2

3

4

```
case 3 : comp_bill.remove(expense.getKey(), expense);
             break;
```

5

```
default:
       throw new RuntimeException() ;
}
```

6

```
        }

    }
```

Path 1	1-2
Path 2	1-3
Path 3	1-4
Path 4	1-5
Path 5	1-6

Path 1	1-2
Objects	Expense expense, Map purchases
Result	expense is removed from Map purchase with its unique key

Path 2	1-3
Objects	Expense expense, Map bill
Result	expense is removed from Map bill with its unique key

Path 3	1-4
Objects	Expense expense, Map comp_purchases
Result	expense is removed from Map comp_purchases with its with unique key

Path 4	1-5
Objects	Expense expense, Map comp_bill
Result	expense is removed from Map comp_bill with its unique key

Path 5	1-6
Objects	Expense expense
Result	throw new RuntimeException();

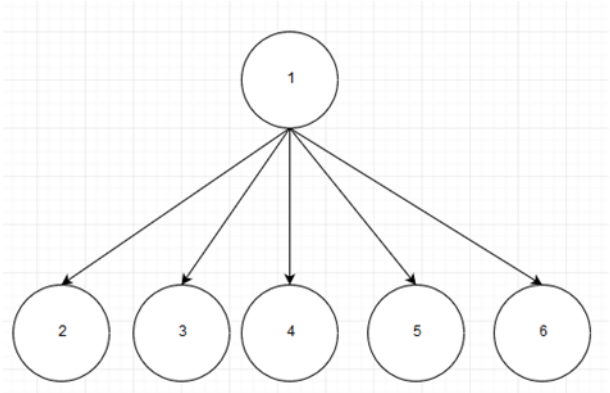


Figure 7. Path diagram for function: remove

3.1.3.1.3 modify Function

3.1.3.1.3.1 Black Box Testing

Name:	Danny		Test Date:	April 11,2019
Class name:	InMemoryStoreTest	Method name:	modify()	File Name:
				InMemoryStoreTest.java
Objects:	List list		Purchase	
	InMemoryStore inMemoryStore		Purchase purchase2	
	Map purchaseMap		ExpenseKey expenseKey	
Variables	String location:	Montreal		
Test Case:	purchase2.setLocation ("Montreal")	inMemoryStore.modify (purchase1, purchase2);		
Excepted Output:	purchase2 location should be set to Montreal	purchase1 should be replaced by purchase2		
Actual Outpit:	purchase2 location is set to Montreal	purchase1 is replaced by purchase2		
Bugs?"	No	No		

3.1.3.1.3.2 White Box Testing

```
public boolean modify(Expense from, Expense to) {

        //Let's check if they are same
        if(!from.iseqal(to)) {
                return false;
        }

        try {
```

1

2

```
                this.remove(from);


        }
        catch (IOException e) {
                // TODO Auto-generated catch block        3
                e.printStackTrace();
        }
        switch(from.getType().ordinal()){                 4
                case 0 : purchases.put(to.getKey(), to);
                        break;                            5
                case 1 : bill.put(to.getKey(), to);
                        break;                            6
                case 2 : comp_purchases.put(to.getKey(), to);
                        break;                            7
                case 3 : comp_bill.put(to.getKey(), to);
                        break;                            8
                default:
                        throw new RuntimeException() ;    9
        }
        return true;                                      10
}
```

Path 1	1
Path 2	1-3
Path 3	1-2-4-9
Path 4	1-2-4-5-10
Path 5	1-2-4-6-10
Path 6	1-2-4-7-10
Path 7	1-2-4-8-10

Path 1	1
Objects	Expense from, Expense to
Result	return false;// The expense has not been modified for any changes

Path 2	1-3
Objects	Expense from, Expense to
Result	e.printStackTrace();// Expense *from* cannot be removed due to system bug.

Path 3	1-2-4-9

Objects	Expense from, Expense to
Result	throw new RuntimeException();// Expense from type number is out of bound

Path 4	1-2-4-5-10
Objects	Expense from, Expense to
Result	expense *from* is removed and expense *to* is put into Map purchases with its unique key

Path 5	1-2-4-6-10
Objects	Expense from, Expense to
Result	Expense *from* is removed and expense *to* is put into Map *bill* with its unique key

Path 6	1-2-4-7-10
Objects	Expense from, Expense to
Result	Expense *from* is removed and expense *to* is put into Map *comp_purchases* with its unique key

Path 7	1-2-4-8-10
Objects	Expense from, Expense to
Result	Expense *from* is removed and expense *to* is put into Map *comp_purchases* with its unique key

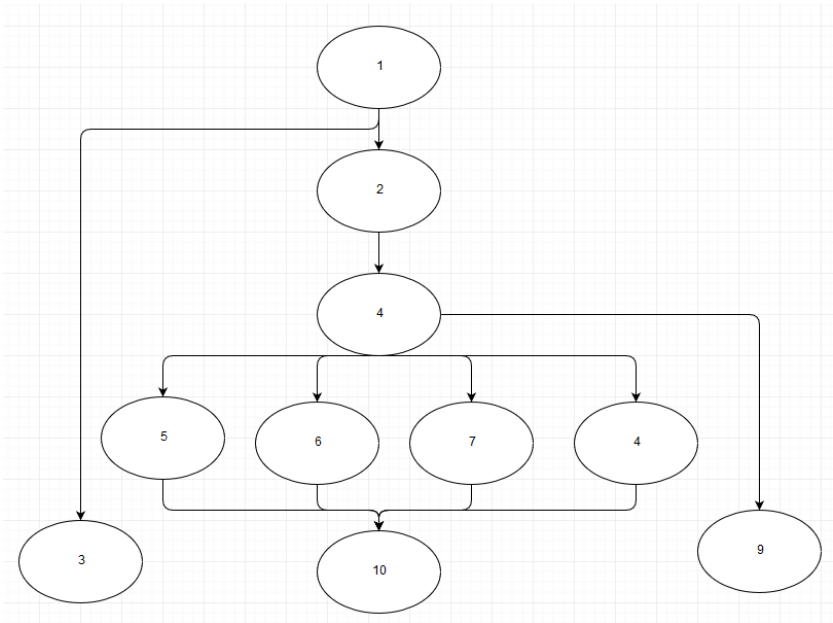


Figure 8. Path diagram for function: modify

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

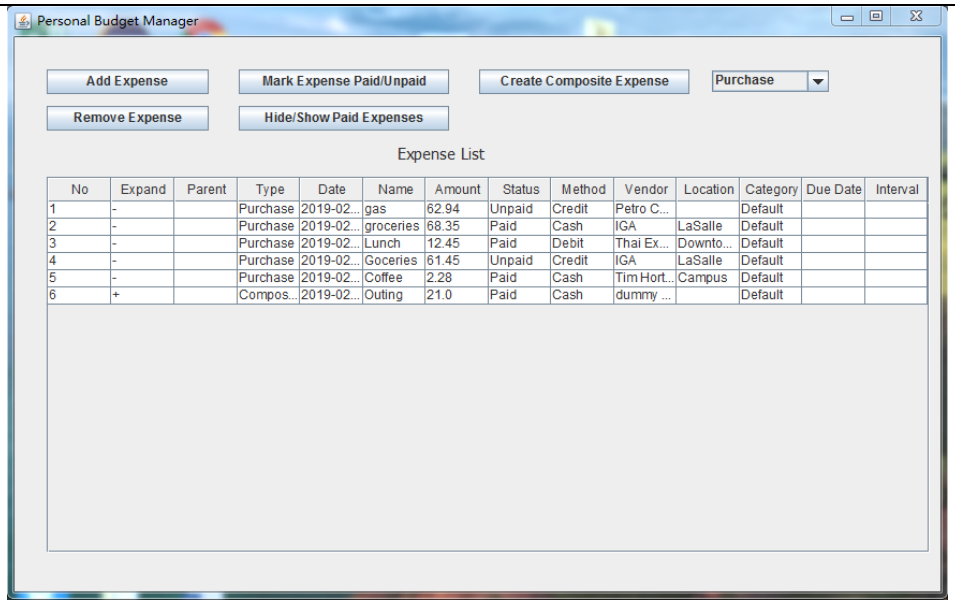
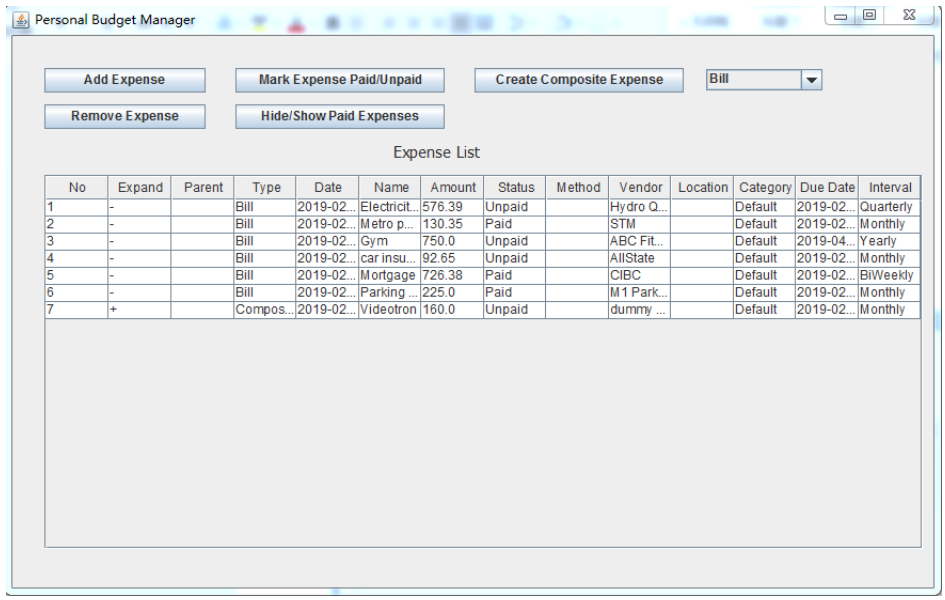
Test Case	show Paid Purchase
Test Case Description	1. open the app 2. click “Hide/Show Paid Expenses” button on the main panel to hide the paid bills 3. click “Hide/Show Paid Expenses” button on the main panel to show the paid bills again
Test data	Type: bill Date: 2018-08-19 Name: jenny Amount: 50 Status: 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

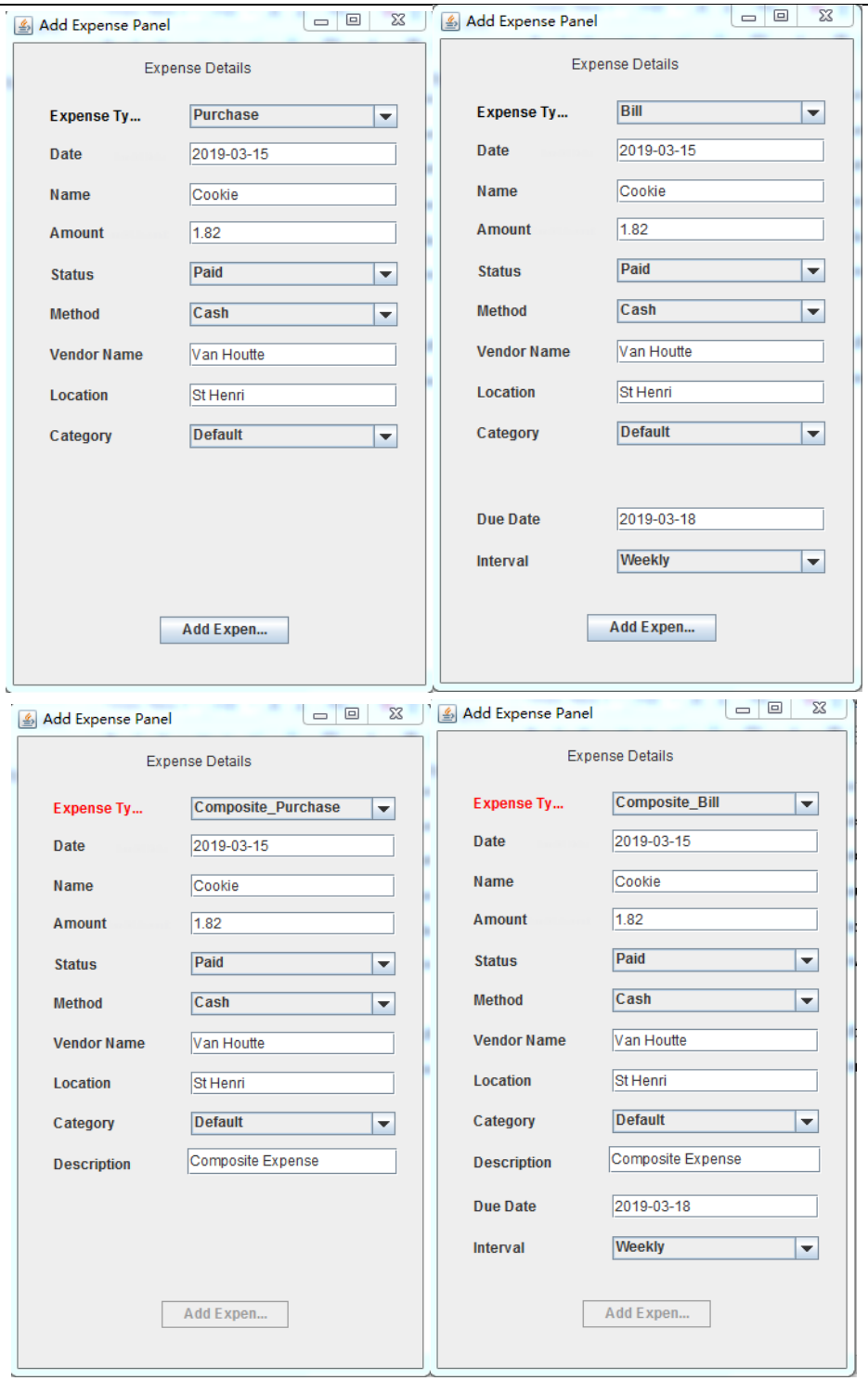
3.4 User Interface Testing

This section is to test the user interface described in the design. To test the User Interface, each functionality described in the design document will be verified to see if it has been implemented correctly.

3.4.1 Main Panel

Interface	 
What is tested?	Display function
Operation	Users can switch purchase and bill list by clicking drop down button.
Expected result	If a user clicks ‘purchase’, it will show all purchase expense. If a user clicks ‘bill’, it will show all bill expense.
Effective result	After clicking ‘purchase’, it shows purchase list. After clicking ‘bill’, it shows bill list.

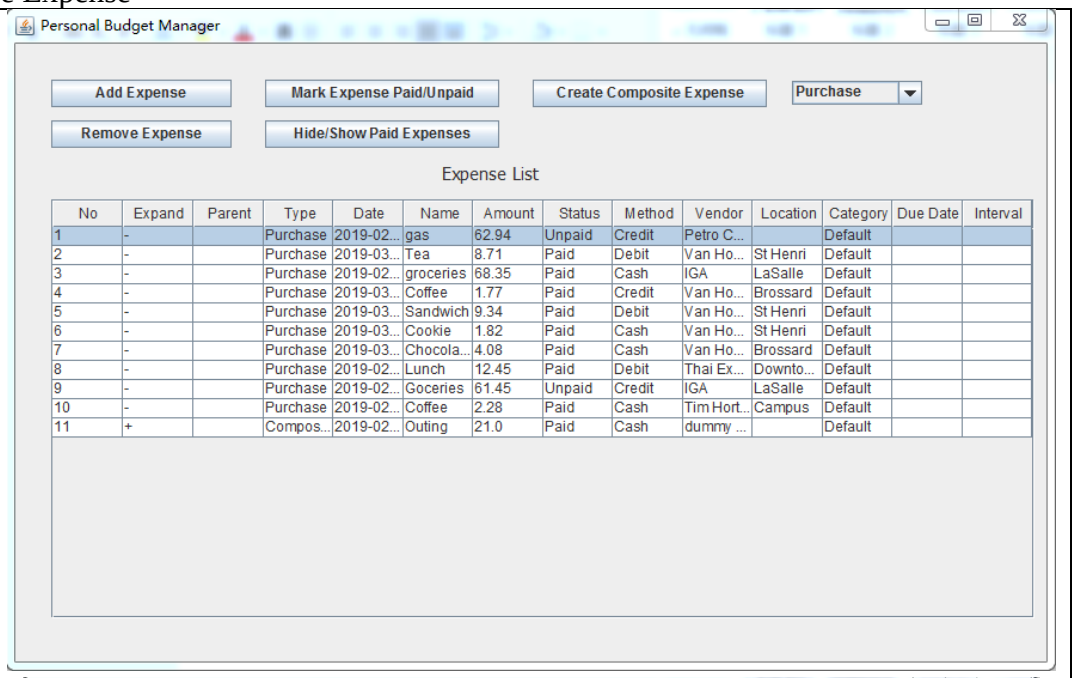
3.4.2 Add Expense

Interface	
What is tested?	Add expense
Operation	Users can add different types of expenses by clicking ‘Add Expense’ button on the main panel.

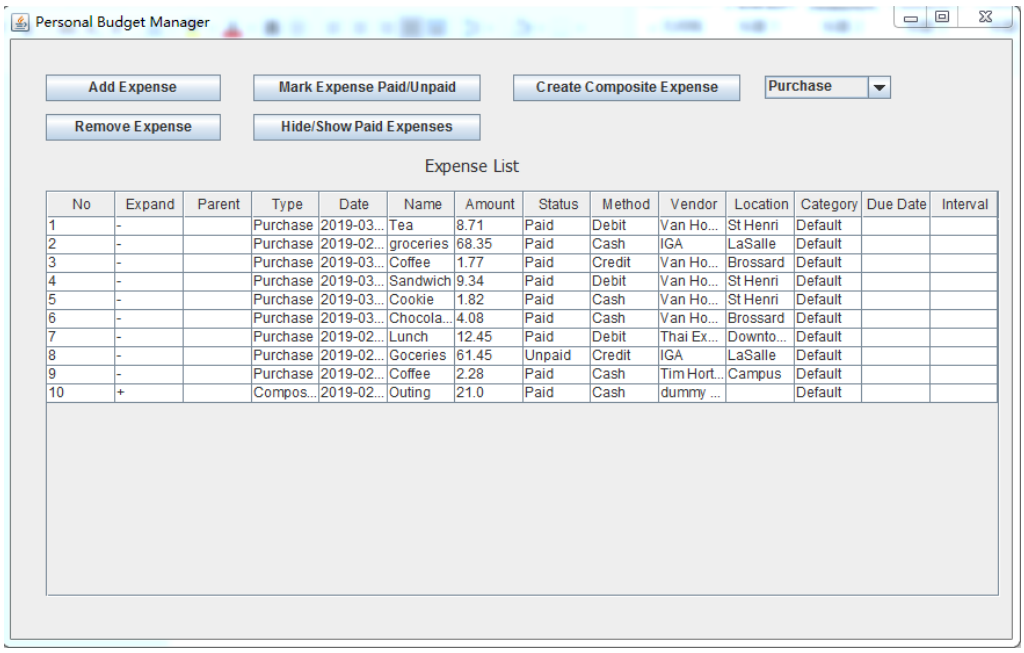
Expected result	When a user clicks ‘Add Expense’ button on the main panel, a new interface will pop up. Here they can input description of their expenses. If a user clicks ‘purchase’, they can input information such as date, name, amount, status, method, vendor name, location and category. If a user clicks ‘bill’, they can input above information and due date and interval. If a user clicks ‘Composite_Purchase’ or ‘Composite_Bill’, the color of expense type will be changed to prompt error. After completing all the information, click the button ‘add expense’ then the data will be stored in database and added on the list.
Effective result	As expected.

3.4.3 Remove Expense

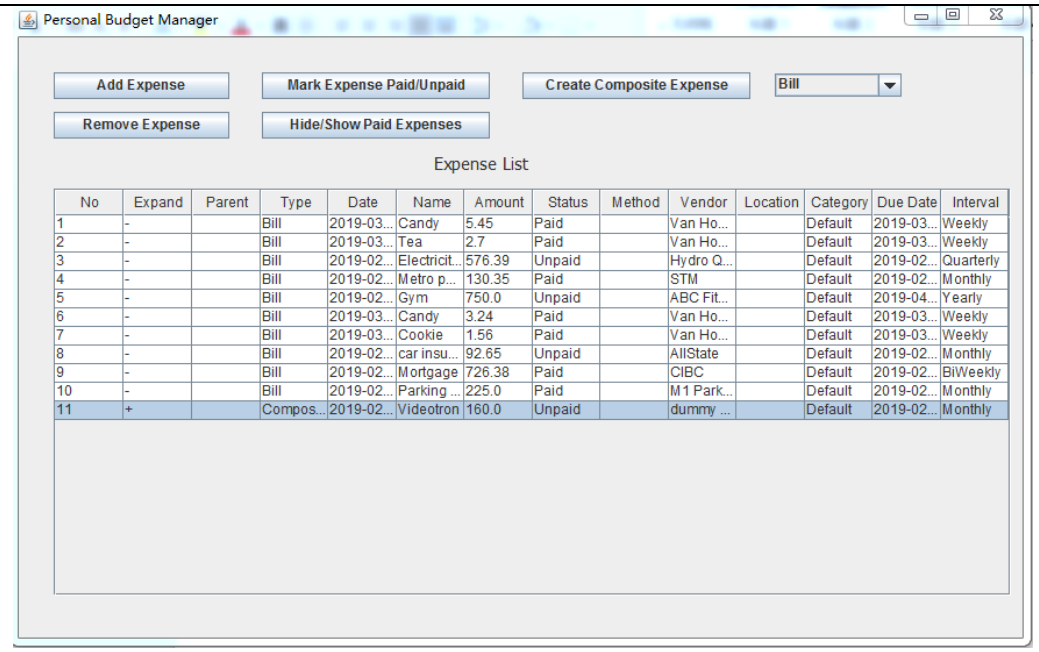
Interface



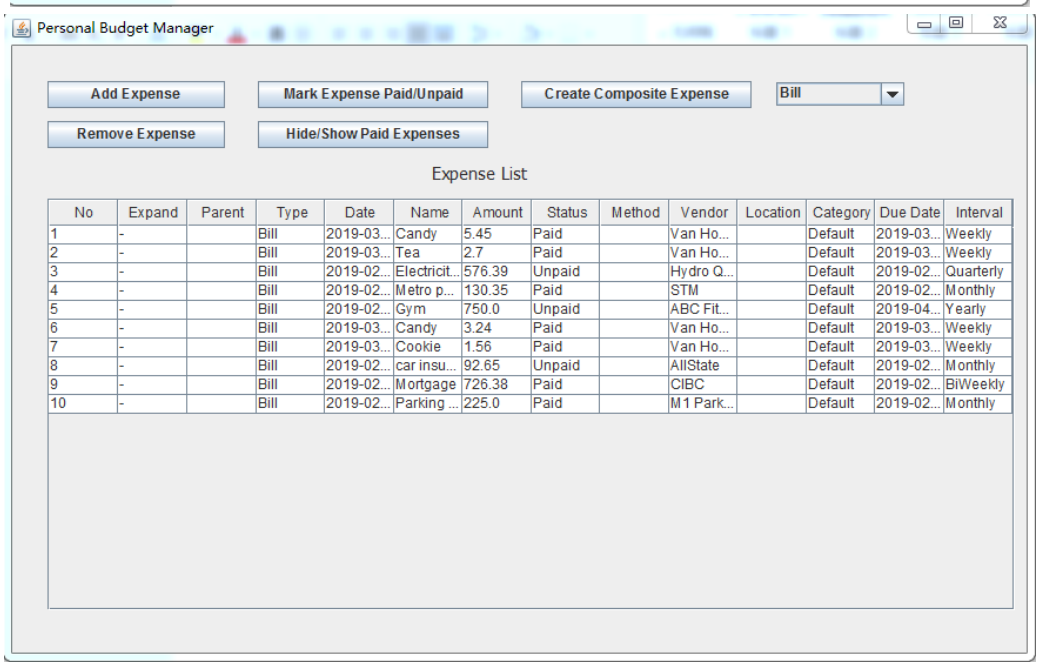
No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-02...	gas	62.94	Unpaid	Credit	Petro C...		Default		
2	-		Purchase	2019-03...	Tea	8.71	Paid	Debit	Van Ho...	St Henri	Default		
3	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
4	-		Purchase	2019-03...	Coffee	1.77	Paid	Credit	Van Ho...	Brossard	Default		
5	-		Purchase	2019-03...	Sandwich	9.34	Paid	Debit	Van Ho...	St Henri	Default		
6	-		Purchase	2019-03...	Cookie	1.82	Paid	Cash	Van Ho...	St Henri	Default		
7	-		Purchase	2019-03...	Chocola...	4.08	Paid	Cash	Van Ho...	Brossard	Default		
8	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
9	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
10	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
11	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		



No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-03...	Tea	8.71	Paid	Debit	Van Ho...	St Henri	Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-03...	Coffee	1.77	Paid	Credit	Van Ho...	Brossard	Default		
4	-		Purchase	2019-03...	Sandwich	9.34	Paid	Debit	Van Ho...	St Henri	Default		
5	-		Purchase	2019-03...	Cookie	1.82	Paid	Cash	Van Ho...	St Henri	Default		
6	-		Purchase	2019-03...	Chocola...	4.08	Paid	Cash	Van Ho...	Brossard	Default		
7	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
8	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
9	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
10	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		



No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Bill	2019-03...	Candy	5.45	Paid		Van Ho...		Default	2019-03...	Weekly
2	-		Bill	2019-03...	Tea	2.7	Paid		Van Ho...		Default	2019-03...	Weekly
3	-		Bill	2019-02...	Electricit...	576.39	Unpaid		Hydro Q...		Default	2019-02...	Quarterly
4	-		Bill	2019-02...	Metro p...	130.35	Paid		STM		Default	2019-02...	Monthly
5	-		Bill	2019-02...	Gym	750.0	Unpaid		ABC Fit...		Default	2019-04...	Yearly
6	-		Bill	2019-03...	Candy	3.24	Paid		Van Ho...		Default	2019-03...	Weekly
7	-		Bill	2019-03...	Cookie	1.56	Paid		Van Ho...		Default	2019-03...	Weekly
8	-		Bill	2019-02...	car insu...	92.65	Unpaid		AllState		Default	2019-02...	Monthly
9	-		Bill	2019-02...	Mortgage	726.38	Paid		CIBC		Default	2019-02...	BiWeekly
10	-		Bill	2019-02...	Parking ...	225.0	Paid		M1 Park...		Default	2019-02...	Monthly
11	+		Compos...	2019-02...	Videotron	160.0	Unpaid		dummy ...		Default	2019-02...	Monthly

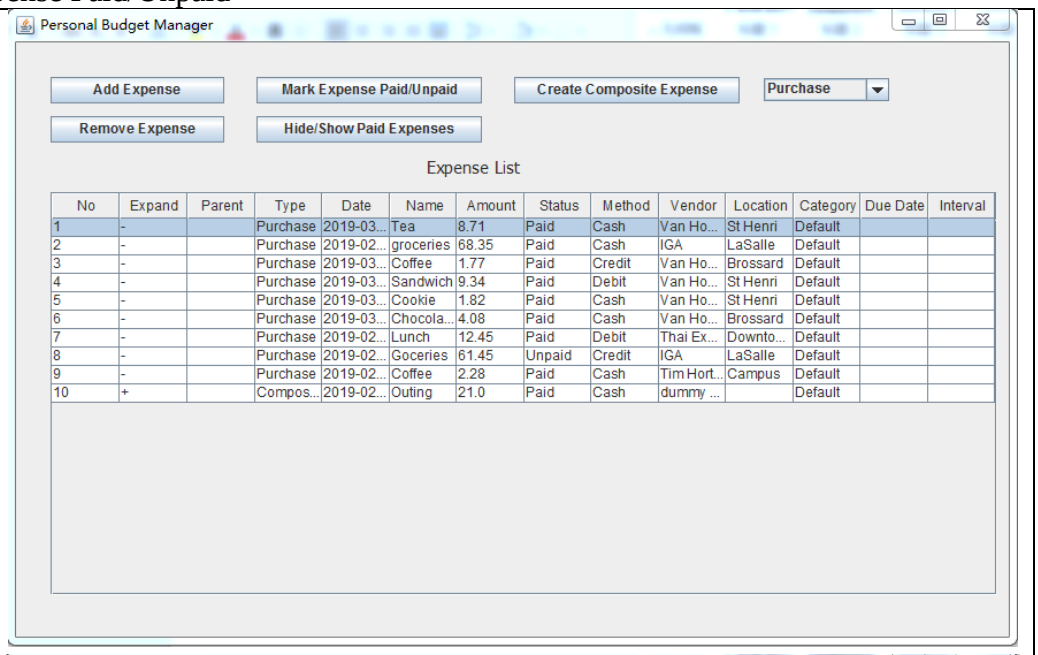
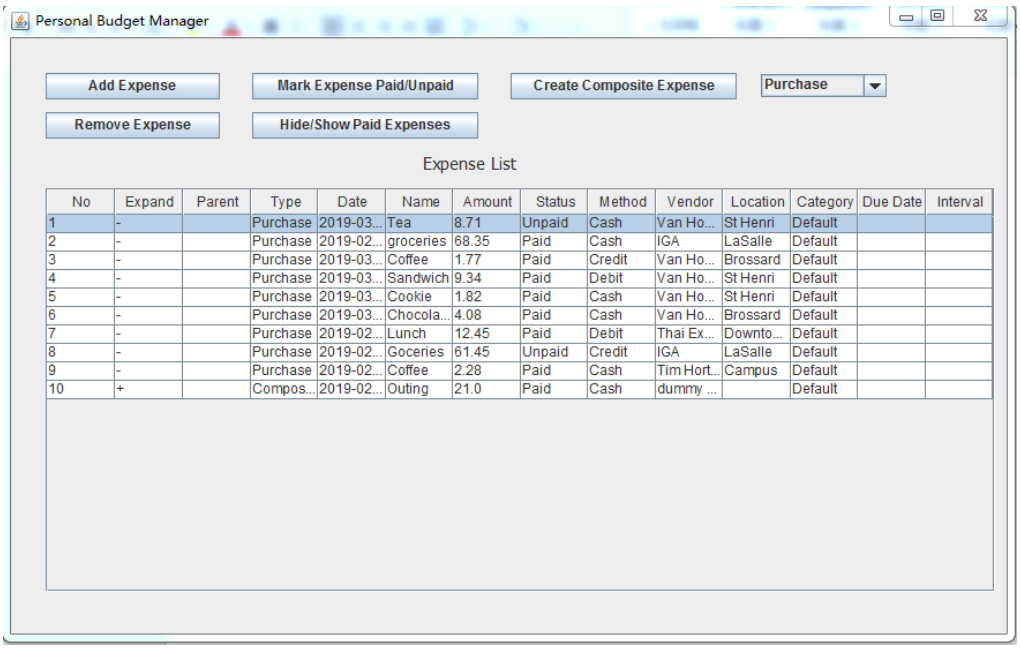


No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Bill	2019-03...	Candy	5.45	Paid		Van Ho...		Default	2019-03...	Weekly
2	-		Bill	2019-03...	Tea	2.7	Paid		Van Ho...		Default	2019-03...	Weekly
3	-		Bill	2019-02...	Electricit...	576.39	Unpaid		Hydro Q...		Default	2019-02...	Quarterly
4	-		Bill	2019-02...	Metro p...	130.35	Paid		STM		Default	2019-02...	Monthly
5	-		Bill	2019-02...	Gym	750.0	Unpaid		ABC Fit...		Default	2019-04...	Yearly
6	-		Bill	2019-03...	Candy	3.24	Paid		Van Ho...		Default	2019-03...	Weekly
7	-		Bill	2019-03...	Cookie	1.56	Paid		Van Ho...		Default	2019-03...	Weekly
8	-		Bill	2019-02...	car insu...	92.65	Unpaid		AllState		Default	2019-02...	Monthly
9	-		Bill	2019-02...	Mortgage	726.38	Paid		CIBC		Default	2019-02...	BiWeekly
10	-		Bill	2019-02...	Parking ...	225.0	Paid		M1 Park...		Default	2019-02...	Monthly

What is tested?	Remove expense
Operation	Users can remove all kinds of expenses by clicking ‘Remove Expense’ button on the main panel.
Expected result	If a user chose a purchase or bill expense, click the button ‘Remove expense’, then the data will be deleted from the list and database. If a user chose a composite expense, click the button ‘Remove expense’, then all the sub-columns will be deleted from the list and database.

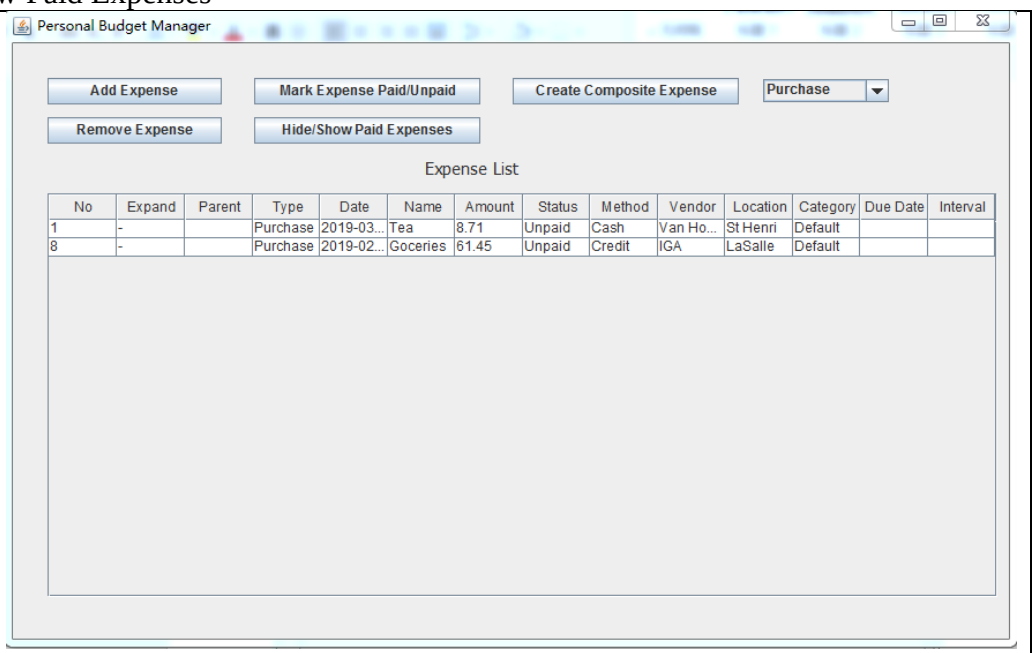
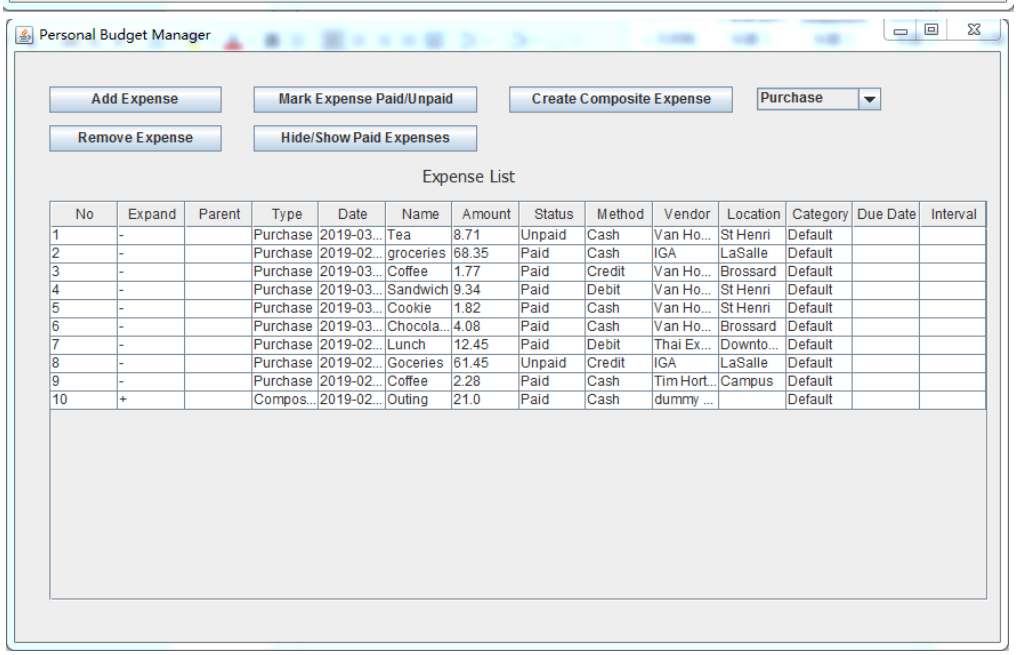
Effective result	As expected.

3.4.4 Mark Expense Paid/Unpaid

Interface	 No Expand Parent Type Date Name Amount Status Method Vendor Location Category Due Date Interval 1 - Purchase 2019-03... Tea 8.71 Paid Cash Van Ho... St Henri Default 2 - Purchase 2019-02... groceries 68.35 Paid Cash IGA LaSalle Default 3 - Purchase 2019-03... Coffee 1.77 Paid Credit Van Ho... Brossard Default 4 - Purchase 2019-03... Sandwich 9.34 Paid Debit Van Ho... St Henri Default 5 - Purchase 2019-03... Cookie 1.82 Paid Cash Van Ho... St Henri Default 6 - Purchase 2019-03... Chocola... 4.08 Paid Cash Van Ho... Brossard Default 7 - Purchase 2019-02... Lunch 12.45 Paid Debit Thai Ex... Downto... Default 8 - Purchase 2019-02... Goceries 61.45 Unpaid Credit IGA LaSalle Default 9 - Purchase 2019-02... Coffee 2.28 Paid Cash Tim Hort... Campus Default 10 + Compos... 2019-02... Outing 21.0 Paid Cash dummy ... Default  No Expand Parent Type Date Name Amount Status Method Vendor Location Category Due Date Interval 1 - Purchase 2019-03... Tea 8.71 Unpaid Cash Van Ho... St Henri Default 2 - Purchase 2019-02... groceries 68.35 Paid Cash IGA LaSalle Default 3 - Purchase 2019-03... Coffee 1.77 Paid Credit Van Ho... Brossard Default 4 - Purchase 2019-03... Sandwich 9.34 Paid Debit Van Ho... St Henri Default 5 - Purchase 2019-03... Cookie 1.82 Paid Cash Van Ho... St Henri Default 6 - Purchase 2019-03... Chocola... 4.08 Paid Cash Van Ho... Brossard Default 7 - Purchase 2019-02... Lunch 12.45 Paid Debit Thai Ex... Downto... Default 8 - Purchase 2019-02... Goceries 61.45 Unpaid Credit IGA LaSalle Default 9 - Purchase 2019-02... Coffee 2.28 Paid Cash Tim Hort... Campus Default 10 + Compos... 2019-02... Outing 21.0 Paid Cash dummy ... Default
What is tested?	Mark expense paid/unpaid
Operation	Users can mark expenses paid or unpaid by clicking ‘Mark expense paid/unpaid’ button on the main panel.

Expected result	If a user chooses a line, click the button 'Mark expense paid/unpaid', the expense status will be changed and stored in the database.
Effective result	As expected.

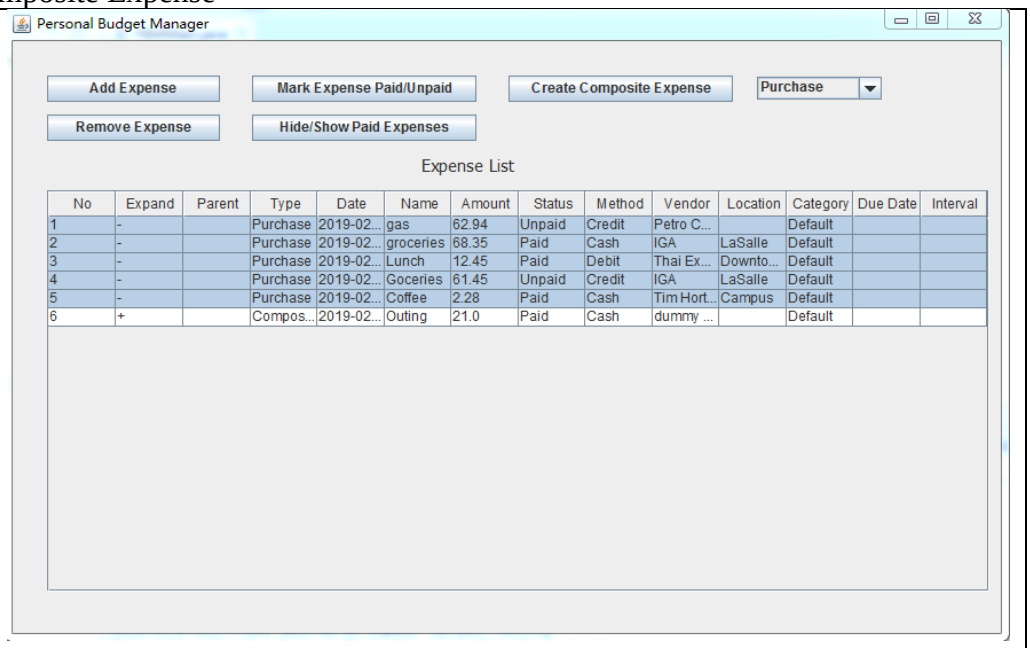
3.4.5 Hide/Show Paid Expenses

Interface	 
What is tested?	Hide/show paid expense

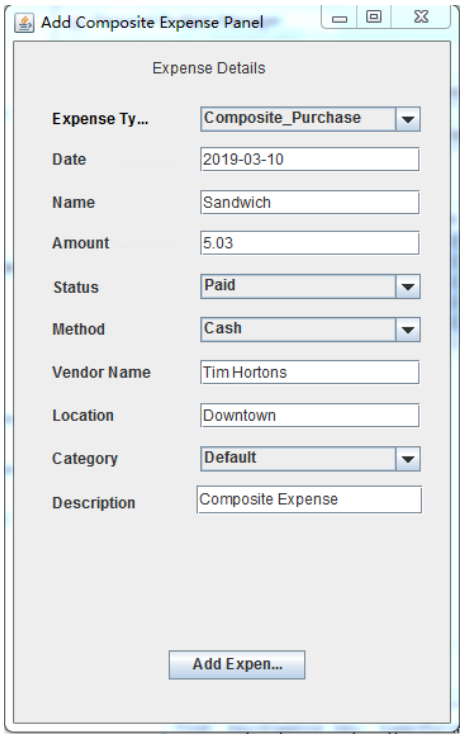
Operation	Users can hide paid expenses by clicking ‘Hide/show paid expense’ button on the main panel.
Expected result	If a user click the button ‘hide/show paid expense’, the list will only show the unpaid expenses. Click the button again, the list will back to original look.
Effective result	As expected.

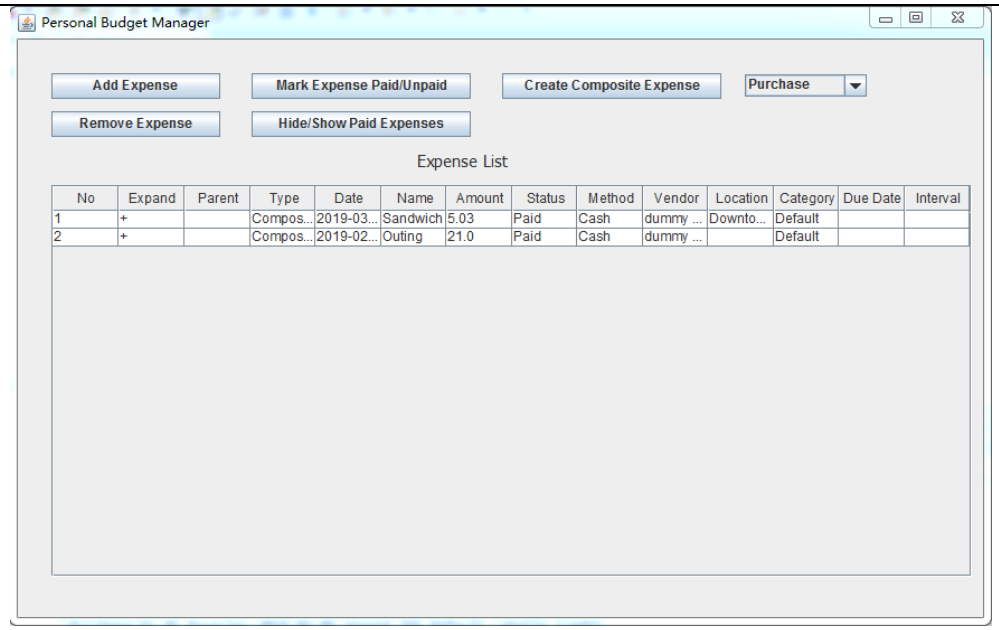
3.4.6 Create Composite Expense

Interface

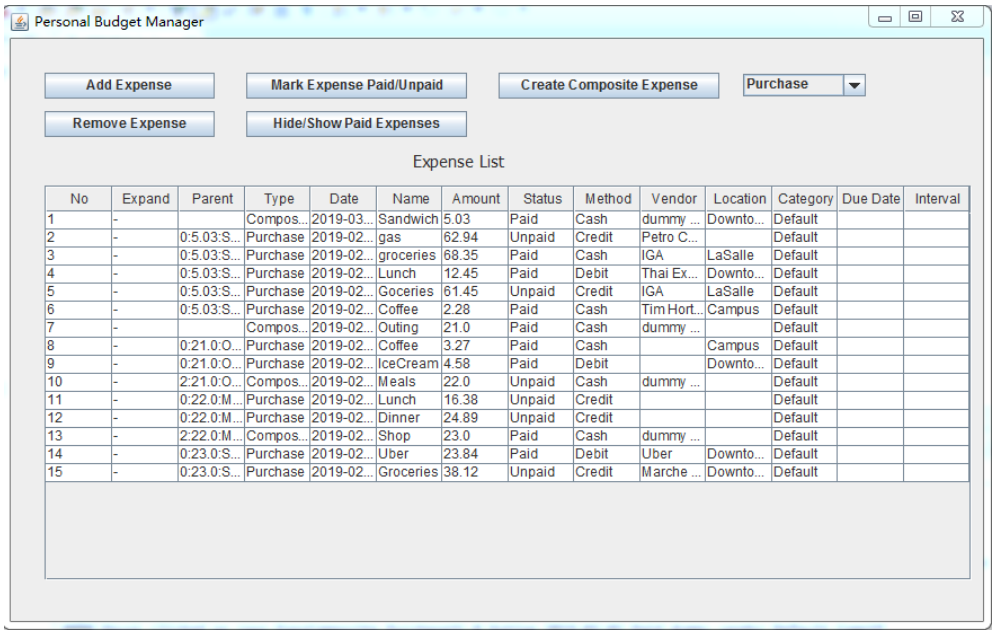


No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Purchase	2019-02...	gas	62.94	Unpaid	Credit	Petro C...		Default		
2	-		Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
3	-		Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
4	-		Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
5	-		Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
6	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		





No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	+		Compos...	2019-03...	Sandwich	5.03	Paid	Cash	dummy ...	Downto...	Default		
2	+		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		



No	Expand	Parent	Type	Date	Name	Amount	Status	Method	Vendor	Location	Category	Due Date	Interval
1	-		Compos...	2019-03...	Sandwich	5.03	Paid	Cash	dummy ...	Downto...	Default		
2	-	0:5.03:S...	Purchase	2019-02...	gas	62.94	Unpaid	Credit	Petro C...		Default		
3	-	0:5.03:S...	Purchase	2019-02...	groceries	68.35	Paid	Cash	IGA	LaSalle	Default		
4	-	0:5.03:S...	Purchase	2019-02...	Lunch	12.45	Paid	Debit	Thai Ex...	Downto...	Default		
5	-	0:5.03:S...	Purchase	2019-02...	Goceries	61.45	Unpaid	Credit	IGA	LaSalle	Default		
6	-	0:5.03:S...	Purchase	2019-02...	Coffee	2.28	Paid	Cash	Tim Hort...	Campus	Default		
7	-		Compos...	2019-02...	Outing	21.0	Paid	Cash	dummy ...		Default		
8	-	0:21.0:O...	Purchase	2019-02...	Coffee	3.27	Paid	Cash		Campus	Default		
9	-	0:21.0:O...	Purchase	2019-02...	IceCream	4.58	Paid	Debit		Downto...	Default		
10	-	2:21.0:O...	Compos...	2019-02...	Meals	22.0	Unpaid	Cash	dummy ...		Default		
11	-	0:22.0:M...	Purchase	2019-02...	Lunch	16.38	Unpaid	Credit			Default		
12	-	0:22.0:M...	Purchase	2019-02...	Dinner	24.89	Unpaid	Credit			Default		
13	-	2:22.0:M...	Compos...	2019-02...	Shop	23.0	Paid	Cash	dummy ...		Default		
14	-	0:23.0:S...	Purchase	2019-02...	Uber	23.84	Paid	Debit	Uber	Downto...	Default		
15	-	0:23.0:S...	Purchase	2019-02...	Groceries	38.12	Unpaid	Credit	Marche ...	Downto...	Default		

What is tested?	Create composite expense
Operation	Users can create composite expenses by clicking ‘Add Expense’ button on the main panel.
Expected result	If a user chooses lines, click the button “Create Composite Expense”, another interface will pop up. Here they can input description of the composite expense. Click the button “Add Expense”, a composite expense can be created. Double click “+” it will show details.
Effective result	As expected.

3.4.7 Moving the Window

Interface	Main panel
What is tested?	Moving the window
Operation	User clicks on the title bar of the window to move it elsewhere on the screen.
Expected result	The window should be moved and placed where the user wants.
Effective result	As expected.

3.4.8 Exit the Application

Interface	Main panel
What is tested?	Exit the application
Operation	User clicks '×' to Exit the application.
Expected result	The window should be closed.
Effective result	As expected.

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

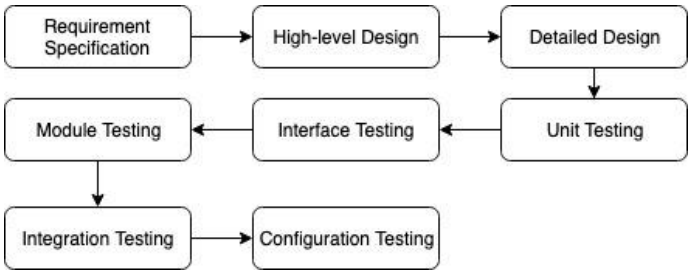
This section describes the procedures and guidelines followed during the tests.

4.1 Test Plan & Software Engineering process

The relationship between test plan and the software engineering process of the project are:

- Software Design Document guides unit-testing plan
- Software Design Document guides integration-testing plan
- Software Requirement Documents guides the testing of the PBM's features

Test plan & SE process

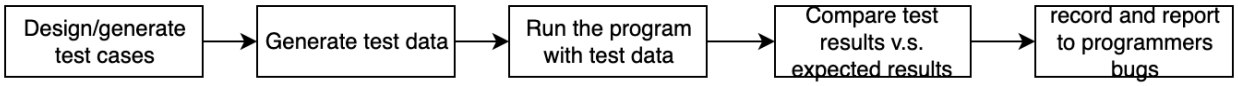


4.2 Work Flow of a Test

For each test, the workflow will be:

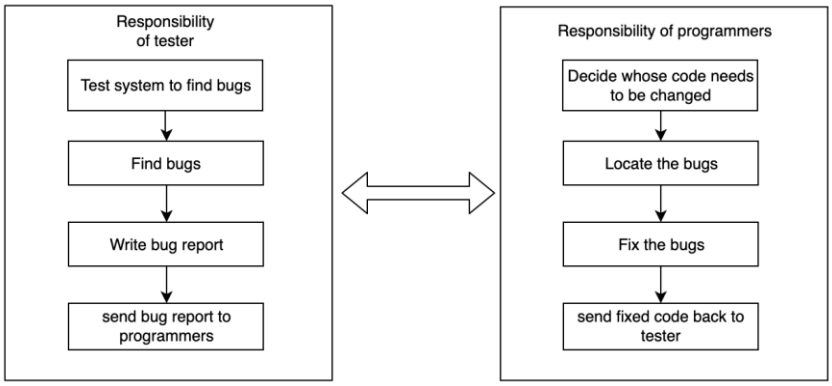
- Design test cases and generate test cases
- Generate test data
- Run the program with the test data.
- Compare the test output with expected results
- Record and report to programmers if the test output does not meet the expected results

The following is the diagram for the test workflow. The diagram applies to unit tests, interface tests, integration tests and configuration tests.



4.3 Workflow of Fixing Bugs

There is a standard workflow for fixing bugs and there are interactions among testers and programmers. The tester is responsible for designing test cases, test PBM application, fill out the bug template and report to programmers. Programmers will fix the bugs and notify the tester the bugs are fixed.



5. Iteration Milestones

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

			January				February				March				April	
Milestone	Days	Who	7	14	21	28	4	11	18	25	4	11	18	25	1	8
General																
Test Plan Template Creation	1d	J														
Test Plan Document	30d	J/D/S														
Phase 3 Deliverables		J														
Testing																
Testing & QA	8d	D/S														
Implementation																
Code	45d	T														
Build 1	20d	T														
Build 2	20d	A														
Build 3	15d	A														

Notes: J (Jenny), D (Danny), S (Siming), T (Tony), A (Amar)