Test Document

Team 2 - Increment 3

3 April 2013

Table 1: Team

Name	ID Number
Kevin Cameron	9801448
Addison Rodomista	1967568
Dragos Dinulescu	6304826
Adrian Max McCrea	9801448
Ghazal Zamani	1971158
Karim Kaidbey	9654726
Carmelo Fragapane	6298265
Long Wang	9547967
Simone Ovilma	9112510
Nicholas Constantinidis	6330746
Asmaa Alshaibi	9738231

1 Introduction

The objectives of this document are to outline the overall Test Plan created for ensuring best performance of the Fun Sheets application. This document outlines the full test schedule and the major phases of testing, how the tests are designed and what their results are, and how the tests relate to the requirements of the program itself.

2 Test Plan

***Increment 1 (Weeks 1-4)**: Unit Testing*
***Increment 2 (Weeks 5-8)**: Additional Unit Testing and Subsystem Testing*
***Increment 3 (Weeks 9-11)**: Integration Testing and Regression Testing*

The phases of the test plan were built using a Bottom-Up approach. The test cases for the first 2 increments relied heavily on JUnit Tests because features were continuously being re-adjusted to appropriately match the system requirements. Throughout all phases, Regression testing was used to ensure additional features did not disrupt previously passed test cases. Each unit test was designed to specify a specific functionality of the program, and each of the tests had approximately 5 variations to ensure consistency among multiple inputs.

***The requirements tested for the application include**:*
- Inputting values or formulas into cells
- Adding cells together
- Testing for various exceptions
- Proper file handling
- Testing equation and grammar operations within cells
- Testing input line and validation
- Testing message line for appropriate outputs
- Testing Cut, Copy, and Paste functionalities
- Verifying the multiple data formats
- Interactivity testing with GUI elements of the program
- Testing hotkey implementation

3 Unit Level Test Cases

All test cases for testing at the system level.

3.1 FileHandler Tests

1. consistencyEmptyTest

Purpose
The purpose of these tests is checking if the functions: consistencyEmpty() works from the filehandler class, testing the loaded empty data is the same as the saved empty data.

Input Specification
Calling FileHandler class for a new file test. Then the first new Grid a is saved in the new file then showing both grids on screen. Test returning a Boolean value if the saved Grid a is the same in the now loaded file of Grid b.

Expected Output
The test should return true assuming Grid a and b are equal

Traces to Use Cases
Traces to the class FileHandler.java in JUnit, validating test class consistencyEmpty().

2. consistencyDoubleTest

Purpose
The purpose of these tests is checking if the functions: consistencyDouble() works from the filehandler class, testing the loaded data is the same as the saved data concerning type double values.

Input Specification
Calling FileHandler class for a new file test. The first Grid a with inserted doubled values is saved in the new file test then returning a Boolean value to check if the saved Grid a is the same in the now loaded file of Grid b. Calling assertEquals for a consistency check of Grid a and b to check if both files are equal.

Expected Output
I expect the test works, returning True, because Grid a has saved double values, then loading the file test thus Grid b will have the same inputted values.

Traces to Use Cases
Traces to the class FileHandler.java in JUnit, validating test class consistencyDouble().

3. loadTest

Purpose
The purpose of these tests is checking if the functions: loadTest() works from the filehandler class, testing if the file can be loaded or not. **Input Specification**
Calling FileHandler class for a new file test. A new Grid a and b. Grid a is saved in file test having inserted values and/or formula(s). Loading it to see if the file exists in comparison with Grid b, and asserting if both of the Grids are equal.

Expected Output
I expect the test works, returning true, so testing if the file test can be loaded.

Traces to Use Cases
Traces to the class FileHandler.java in JUnit, validating test class loadTest().

3.2 Cell Tests

4. TestCreationCells

Purpose
The purpose of these tests are to ensure that the creation of a new Cell object properly assigns the appropriate data to the default variables.

Input Specification
Both default constructors are tested along with parameterized one. For the parameterized tests the input data includes a double representing the cell value, followed by two ints representing their x and y coordinate for the grid.

Expected Output
The tests should confirm that the creation of the Cell objects allocate proper data to default variables.

Traces to Use Cases
The Cell objects are the core of the system, and as such are necessary requirements for creating the entire application. These test cases in particular ensure that the smallest unit of our system can be created as expected.

5. TestCellGetters

Purpose
The purpose of these tests are to ensure that the Get method calls function as expected for their various uses in the program. These include returning cell data, rows, and columns.

Input Specification
Sample data passed include a double value representing the Cells content, followed by two ints representing their x and y coordinates in the grid.

Expected Output
The expected output should be consistent with the parameters passed for input.

Traces to Use Cases
Getting specific cells relates directly with the applications ability to communicate between cells and allows functionalities like adding multiple cells together to work appropriately.

6. TestCellSetters

Purpose
The purpose of these tests are to ensure that the Set method calls allocate the appropriate data to the specified cells

Input Specification
Sample data passed include a double value representing the Cells content, followed by two ints representing their x and y coordinates in the grid.

Expected Output
The expected output should be consistent with the parameters passed for input.

Traces to Use Cases
Setting specific cells directly relates to the applications ability to accept input from the user and manipulate the grid. These use cases ensure functionality.

7. TestCellContent

Purpose
The purpose of these tests are to determine the actual content within Cells.

Input Specification
No data is passed for these tests.

Expected Output
The expected output should consist of a boolean statement specifying whether the given cells contain Values or Formulas.

Traces to Use Cases
These use cases are traced to ensure that all cells within the grid contain content and can never be completely empty.

3.3 CommandClass Tests

8. testValidations

Purpose
The purpose of these tests is checking if the functions: isValid(), testIsInRange() , testIsValidEquation() , testIsAlphaNumeric() from the command class works as it is expected to or not. The isValid() and testisInRange() functions are just returning a Boolean object true. The isValidEquation() function checks for the equations to be in the required format the isAlphaNumeric() function is also called within this function to check the pattern of the passed equation.

Input Specification
We create a new object from the command class that is named command for testing. The

functions are then run on this sample object. No parameter is needed to be passed to these functions.

Expected Output
We expect the output to be true for testIsValid(), testIsInRange() , testIsValidEquation() and false for testIsAlphaNumeric().

Traces to Use Cases
All of these functions are to check the validities of the equations which are passed by the user to our grid. These functions are used within the Command class only to make sure no invalid equation or value is passed to the grid.

9. testUpdateGrid

Purpose
The purpose of these tests is checking if the functions: updateGrid() is working correctly or not. The function goes through the whole grid and update all cells with the new data.

Input Specification
We create a new object from the Command class that is named command and a new object from the Grid class for testing. The function is then run on this sample objects. We need to pass two parameters to the function one Grid object and the cell name as a String.

Expected Output
We expect the output to be according to all the updates applied on every cells even after applying the formula. It updates the grid after each command if applicable. For example if you have the following commands:
A1 = 5
B1 = 10 + A1
A1 = 10
When A1 = 10 updateGrid checks for dependencies and will not only update A1 but also the value of B1 will be updated from 15 to 20. Test Case: testReplaceCellNamesBy-Value1() , testReplaceCellNamesByValue2() , testReplaceCellNamesByValue3():

Traces to Use Cases
The function is used in the Command class to verify that all cells values are up to date according to the last changes made by the user to the grid even if they are already used elsewhere in an equation.

10. testReplacingCells

Purpose
The purpose of these tests is checking if the functions: testReplaceCellNamesByValue1() is working correctly or not. This function basically shows the equation or the value that is recorded in the cell.

Input Specification
We create a new object from the Command class that is named command for testing. The function is then run on this sample object. We need to pass two parameters to the function one equation as a String and a double value.

Expected Output
For this test cases we change the three sample cells value to 0.0 and then check if the change we made was successful or not (So we check if the value of that specific cell is 0.0 or not).

Traces to Use Cases
The function is used in the Command class to basically edit the cell to hold an equation instead of the value. It can be some equation on two cells (addition, subtraction, etc.), two numbers or a number and another cell. The test checks if the final record in the cell is in the cell accordingly.

3.4 Grid Tests

11. testGridCellGetters

Purpose
These functions check if the getCellRow() function works properly or not. Basically the functions job is to return the integer representing cells row.

Input Specification
We first create a new object from Grid class and run the function on this object to test its functionality. The only parameter passed to getCellRow() is the cells name as a String.

Expected Output
Since the functionality of the function is such that the row of any cell name is the number minus one, we for example expect the output for cell B6 to be 5 and D1 to be 0.

Traces to Use Cases
The function GetCellRow() retrieves the cell row index from the cell name, it gets the numerical part of cell name and subtract it by one to return the row.

12. testGridCellAlphanemericInputs

Purpose
These functions are all different test cases for checking the functionality of the AlphanumericInput() function that checks if there are cell names after the = sign i.e check for C4 F7 in A1 = C4 + 5 - F7.

Input Specification
First we made an object from Grid class to run the function on it, then we either call the insertValue() on the grid object that we created to insert our sample values or we just test the cell the way it was initialized. After that we test the function with different

sample equations.

Expected Output
We check if the equations that were inserted in the cells are working properly meaning the correct equation is returned by the function.

Traces to Use Cases
Every time a cell name is found i.e A1 the function retrieves its index by using getCellRow getCellColumn if you for example C4 is found, then its index is row = 4 column = 3. Once the index of the cell is known, its value is retrieved and the name of the cell is replaced by its value. In the newEquation string i.e if newEquation = "B1 + 4 + C2" and B1=7, C2=9; then newEquation becomes "7 + 4 + 9".

13. testGridCellNumericInputs

Purpose
These functions check the numericInput() function for its functionality. The functions job is to get a numeric input or a equation with numbers such as 0.0 + 4.0 as input and record it accordingly.

Input Specification
The input which is passed to the function as parameter is basically an expression to be calculated.

Expected Output
The function should return a double value representing the new value assigned to the cell.

Traces to Use Cases
For this test case we assign different values to cells either by a single double (5.0) or by an expression (0.0 + 0.3), and the final cell value should be record accordingly. The function actually evaluate the arithmetic expression.

14. testGridGetCells

Purpose
These functions are to check the functionality of the getCell() function from Grid class that basically returns the value of the cell by taking its column and row.

Input Specification
First we create an object from Grid class and call insertValue() function on this object to give the girds cells some sample values. For input we need to pass dimensions of the cell as parameters.

Expected Output
We run the test to check if the assigned value is correctly recorded or not.

Traces to Use Cases
The function getCell itself is an easy function that return the the Cell at the x,y position.

(x and y are the dimensions of the cell on the grid)

15. testGridGetCellColumns

Purpose
These functions check if the getCellColumn() function works properly or not. Basically the functions job is to return the integer representing cells column.

Input Specification
We first create a new object from Grid class and run the function on this object to test its functionality. The only parameter passed to getCellColumn() is the cells name as a String.

Expected Output
The output is expected to be an integer representing the column number for example for A1 we expect 0 or for G9 we expect 6.

Traces to Use Cases
The function GetCellColumn() retrieves the cell column index from the cell name i.e. A1 =¿ column index is 0 (65 - 65), ASCII(A) = 65

4 Subsystem Level Test Cases

4.1 Document Editing Tests

1. Testing Copy Functionality

Summary
Copy the value of the cell whether it's a primitive or formula value. Paste the value of the last copied value or cut. Undo-ing the last event that has been done should be counteracted with a redo event.

Preconditions Cell(s) must be selected **Main Scenario**

1. Copy the value of the selected cell
2. Temporary storage will store the value for the next event

Expected Results
The copied value should be stored in temporary storage.

2. Testing Cut Functionality

Summary

Cut the value of the cell whether it is a primitive or formula value.

Preconditions

Cell must be selected

Main Scenario

1. Cut the value of the selected cell
2. Temporary storage will store the value for the next event

Expected Results

The result of the selected value will disappear on the view and will be storage in temporary storage.

3. Testing Paste Functionality

Summary

Paste the value of the last copied or cut value to a cell

Preconditions

Cell must be selected, followed by the command to paste.

Main Scenario

1. User Selects a cell
2. Copies contents of specified cell
3. User Selects a different cell
4. Issues paste command through GUI or Hotkeys
5. Value is inputted into cells

Expected Results

The value(s) held in temporary storage will be pasted to the specified cell

4. Testing Undo Functionality

Summary

Retrieve the previous event(s)

Preconditions

The user must have made at least 1 change to the application at the time of using the Undo command.

Main Scenario

1. User Selects a cell
2. User edits a cell
3. User issues an Undo command
4. Cell contents return to previous state

Expected Results

Per use of an Undo command, the spreadsheet should move back to its immediately previous state

5. Testing Primitive Formatting Functionality

Summary

Adjusts the cells primitive formatting to either Monetary, Integer, or Scientific formatting

Preconditions

A cell must be selected and for "Format" option must be selected from the menu GUI.

Main Scenario

1. User Selects a cell
2. User selects the Format tab
3. User selects desired primitive format
4. Cell contents are adjusted appropriately

Expected Results

The cells primitive format adjusts according to the set format.

6. Testing Theme Functionality

Summary

Adjusts the grid design and overall graphical user interface to users preference.

Preconditions

Theme option from the options tab must be selected

Main Scenario

1. User selects "Theme Tab"
2. User selects "Select Theme"

3. User selects desired theme from list
4. Spreadsheets UI is changed as desired

Expected Results

The spreadsheet application appropriately adjusts its theme to the desired settings.

7. Testing Formula Input Functionality

Summary

Once a formula has been inputted into a cell, the appropriate calculations are made and the primitive result is displayed in the cells view.

Preconditions

A cell must be selected

Main Scenario

1. User selects a cell
2. User enters a formula into the input fields
3. Primitive results are displayed in cells view

Exceptions

1. When selected cell address appears in formula, throw exception
2. Warning message displayed on UI

Expected Results

Correct calculations are made from users inputted formula and are displayed in the cell.

5 Integration Testing

5.1 Subsystem Testing

File Manager Test

Name
Create, Save and open

Summary
Open a new instance of FunSheets with a default grid. Modify the values of the grid using primitive and formula values. Save the sheet to a new file, and then re-open it.

Precondition
None

Main Scenario

1. Open Funsheets using the default Icon or executable.
2. Modify one or several cell blocks.
3. Select the 'File' menu item and select 'Save'. Give file a new name
4. Close the instance of FunSheets
5. Open a new instance of FunSheets, Select 'File' menu item and select 'Load'.
6. Navigate to the previously saved file and select it.
7. Verify the primitive and formula values are consistent.

Exceptions

1. User must have write privileges to the save location.

Expected Results

1. If User has write privilages to system location FunSheets will create a .csv file located at the directory selected in the dialog box
2. If the User does not have write privilages FunSheets will report and access denied error and no files will be saved

Postcondition And Results

1. User created .csv file is saved to hard disk. Results are successful

Priority
High (Main requirment)

Name
Open a third-party modified .csv

Summary
Open the .CSV using a text editor, or third-party spreadsheet application and modify the cell values.

Precondition
Pre-Existing .csv file created from FunSheets

Main Scenario

1. Open the funSheets .csv file is default text editor
2. Modify one or several cell elements and save the file
3. Open a new instance of FunSheets, Select 'File' menu item and select 'Load'.
4. Navigate to the previously saved file and select it.
5. Varify the primitive and formula values are consistant.

Exceptions

1. If third-party modifications do NOT follow the formatting requirments of FunSheets the .csv file will not open. Internally we will trap an exception and report that the file is not valid.

Expected Results

1. If the .csv file is modified and the changes adhear to the formatting requirments of FunSheets the spreadsheet application will open successfully and the changes will be visable to the user.
2. If the .csv file is modified and the changes conflict with the formatting requirments of FunSheets the file is deemed incompatable and an error message should be displayed.

Postcondition and Results

1. Modified .csv that conforms to formatting was open successfully and the changes were viable. Test Successful
2. Modified .csv that conflicts to formatting was open successfully, error were not reported. Test failed see example input IT_SUB_FM_Test2.csv

Priority
Low (Modifing .csv outside FunSheets is not supported)

Name
Create a new .csv overwrite old data

Summary
Open and existing .csv, use the 'File' menu option to create a new spreadsheet

Precondition
Pre-Existing .csv file created from FunSheets

Main Scenario

1. Open Funsheets using the default Icon or executable.
2. Select the 'File' menu item and select 'New' or press 'CTRL + N' for new
3. Accept warning for unsaved data.
4. Select the 'File' menu and select 'Save' or press 'CTRL + S'
5. Overwrite existing filename

Exceptions

1. None

Expected Results

1. Warning message is displayed to warn the user about overwriting data.
2. Accepting the warning will overwrite the saved file and data is lost
3. Canceling the warning message will preserve the old file without datalost

Postcondition and Results

1. No warning message is display. (Test failed)

Priority
High - Loss of data is unacceptable without a warning message to the user

Name
Open an existing file without saving the current document

Summary
After working on a file proceed to open an existing file without saving the current project

Precondition
Pre-Existing .csv file created from FunSheets

Main Scenario

1. Open Funsheets using the default Icon or executable.
2. Modify one or several fields in the grid
3. Select the 'File' menu and select 'Load' or press 'CTRL + L'
4. Navigate to an existing .csv file and select it.
5. Review warning message.

Exceptions

1. None

Expected Results

1. Warning message is displayed to warn the user about unsaved data
2. Accepting the warning will open the existing file, and all current work will be lost
3. Canceling the warning message will preserve the current instance and allow the use to save the file.

Postcondition and Results

1. Warning message is displayed and information is clear. Pressing 'Cancel' preserves the instance without data loss. Test successful
2. Warning message is displayed and information is clear. Pressing 'Ok' will load the selected file and all current data is lost. Test successful

Priority
Successful

5.2 Portability Testing

Windows XP 32-Bit

Name
Windows XP 32-BIT deployment

Summary
Port an executable version of FunSheets to a PC running Windows XP 32-BIT and test basic functionality and requirments.

Precondition

1. Java project compiled without errors and all Major unit and subsystem tests have passed during development
2. Target PC has properly configured Java Run-Time-Enviornment (JRE) or Java Development Tool-kit (JDK) installed and working.

Main Scenario

1. Using official version of FunSheets media (USB or DVD) install or copy the executable file to the target PC
2. Launch an instance of FunSheets by doubleclicking on the executable.
3. Testing various functionality according to Subsystem and Integration testing outlined in section XXX.XX

Exceptions

1. None

Expected Results

1. Subsystem test number XXX.XX to XXX.XX are successful.

Postcondition And Results

1. Results are successful

Priority
Successful

Windows 7 32-Bit

Name
Windows 7 32-BIT deployment

Summary
Port an executable version of FunSheets to a PC running Windows 7 32-BIT and test basic functionality and requirments.

Precondition

1. Java project compiled without errors and all Major unit and subsystem tests have passed during development
2. Target PC has properly configured Java Run-Time-Enviornment (JRE) or Java Development Tool-kit (JDK) installed and working.

Main Scenario

1. Using official version of FunSheets media (USB or DVD) install or copy the executable file to the target PC
2. Launch an instance of FunSheets by doubleclicking on the executable.
3. Testing various functionality according to Subsystem and Integration testing outlined in section XXX.XX

Exceptions

1. Windows 7 UAC (User Account Controls) can provide complications when running third-party software

Expected Results

1. Subsystem test number XXX.XX to XXX.XX are successful.

Postcondition And Results

1. In some cases UAC controls need to be set to allow FunSheets to run correctly. Firewall exceptions should be created.
2. Results are successful

Priority
Successful

Windows 7 64-Bit

Name
Windows 7 64-BIT deployment

Summary
Port an executable version of FunSheets to a PC running Windows 7 64-BIT and test basic functionality and requirments.

Precondition

1. Java project compiled without errors and all Major unit and subsystem tests have passed during development
2. Target PC has properly configured Java Run-Time-Enviornment (JRE) or Java Development Tool-kit (JDK) installed and working.

Main Scenario

1. Using official version of FunSheets media (USB or DVD) install or copy the executable file to the target PC
2. Launch an instance of FunSheets by doubleclicking on the executable.
3. Testing various functionality according to Subsystem and Integration testing outlined in section XXX.XX

Exceptions

1. Windows 7 UAC (User Account Controls) can provide complications when running third-party software

Expected Results

1. Subsystem test number XXX.XX to XXX.XX are successful.

Postcondition And Results

1. In some cases UAC controls need to be set to allow FunSheets to run correctly. Firewall exceptions should be created.
2. Results are successful

Priority
Successful

Windows 8 32-Bit

Name
Windows 8 32-BIT deployment

Summary
Port an executable version of FunSheets to a PC running Windows 8 32-BIT and test basic functionality and requirments.

Precondition

1. Java project compiled without errors and all Major unit and subsystem tests have passed during development
2. Target PC has properly configured Java Run-Time-Enviornment (JRE) or Java Development Tool-kit (JDK) installed and working.

Main Scenario
1. Using official version of FunSheets media (USB or DVD) install or copy the executable file to the target PC
2. Launch an instance of FunSheets by doubleclicking on the executable.
3. Testing various functionality according to Subsystem and Integration testing outlined in section XXX.XX

Exceptions

1. Windows 8 UAC (User Account Controls) can provide complications when running third-party software
2. FunSheets does not have an integrated Windows 8 Metro interface. Can only run from Desktop view

Expected Results

1. Subsystem test number XXX.XX to XXX.XX are successful.

Postcondition And Results

1. In some cases UAC controls need to be set to allow FunSheets to run correctly. Firewall exceptions should be created.
2. A Desktop icon should be created and used rather then Metro interface.
3. Results are successful

Priority
Successful

Windows 8 64-Bit

Name
Windows 8 64-BIT deployment

Summary
Port an executable version of FunSheets to a PC running Windows 8 64-BIT and test basic functionality and requirments.

Precondition

1. Java project compiled without errors and all Major unit and subsystem tests have passed during development
2. Target PC has properly configured Java Run-Time-Enviornment (JRE) or Java Development Tool-kit (JDK) installed and working.

Main Scenario
1. Using official version of FunSheets media (USB or DVD) install or copy the executable file to the target PC
2. Launch an instance of FunSheets by doubleclicking on the executable.
3. Testing various functionality according to Subsystem and Integration testing outlined in section XXX.XX

Exceptions

1. Windows 8 UAC (User Account Controls) can provide complications when running third-party software
2. FunSheets does not have an integrated Windows 8 Metro interface. Can only run from Desktop view

Expected Results

1. Subsystem test number XXX.XX to XXX.XX are successful.

Postcondition And Results

1. In some cases UAC controls need to be set to allow FunSheets to run correctly. Firewall exceptions should be created.
2. A Desktop icon should be created and used rather then Metro interface.
3. Results are successful

Priority
Successful

Apple OS X

Name
Apple OS X deployment

Summary
Port an executable version of FunSheets to a PC running Apple OS X and test basic functionality and requirments.

Precondition

1. Java project compiled without errors and all Major unit and subsystem tests have passed during development
2. Target PC has properly configured Java Run-Time-Enviornment (JRE) or Java Development Tool-kit (JDK) installed and working.

Main Scenario

1. Using official version of FunSheets media (USB or DVD) install or copy the executable file to the target PC
2. Launch an instance of FunSheets by doubleclicking on the executable.
3. Testing various functionality according to Subsystem and Integration testing outlined in section XXX.XX

Exceptions

1. User account as approprate privilages to install software

Expected Results

1. Subsystem test number XXX.XX to XXX.XX are successful.

Postcondition And Results

1. Results are successful

Priority
Successful

6 Test Results

List the tests, indicating which passed and which did not pass. List requirements indicating the percentage of tests that passed for that requirement.

6.1 CellTests

testCellCreations

Success rate 100 percent (16/16 passed)

testCellGetters

Success rate of 100 percent (8/8 passed)

testCellSetters

Success rate of 100 percent (5/5 passed)

testValidations

Success rate of 100 percent (17/17 passed)

6.2 CommandTest

testValidations

Success rate of 100 percent (7/7 passed)

testUpdatingGrid

Success rate of 100 percent (1/1 passed)

6.3 GridTest

testGridInputs

Success rate of 100 percent (8/8 passed)

testGridGetters

Success rate of 66 percent (8/12 passed)
testGetCell[0-4] resulted in failed runs

6.4 FileHandlerTests

controlTest

Success rate of 100 percent (1/1 passed)

consistencyTests

Success rate of 25 percent (1/ 4 passed)
consistencyDouble, consistencyFormula, and consistencyBoth led to failed runs

saveTest

Success rate of 0 percent (0/1 passed)
saveTest lead to failure

loadTest

Success rate of 0 percent (0/1 passed)
loadTest lead to failure

7 References