FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.1

Prepared by: Student 5
SoftFlux Engineer

SoftFlux

May 7, 2023

Contents

List of Figures

List of Tables

List of Algorithms

Revision History

Name	Date	Reason for Changes	Version
Student 5	May 3rd 2023	Exit Function written	V1.1

1 Introduction/Purpose

1.1 Purpose

This document is Functional Software Test Plan for the software Encost Smart Graph Project (ESGP) for the company Encost. The purpose of this document is to provide a blueprint for the software development process and outline what tests need to be completed in order for the ESGP to function correctly as outlined by the clients Encost. This functional test plan will outline the black-box testing, white-box testing and the mutation testing required for development along with any specialised requirement specifications.

This functional software test plan is derived off the Software Design Specification 1 document.

1.2 Document Conventions

This document will use the following conventions:
ESGP: Encost Smart Graph Project
CLI: Command Line Interface
UI: User Interface
ESHD: Encost Smart Home Dataset
CSV: Comma-Separated Values

1.3 Intended Audience and Reading Suggestions

The intended audiences for this document includes; Developers, testers and project managers.

- Developer: May use this document as an outline for development to ensure the program which is developed meets the requirements outlined in the SRS through passing the tests outlined.
- Tester: May use this document as a reference in order to test the completed ESGP meets the requirements outlined in the SRS by passing the tests outlined within this document in the specified and correct manner as expected.

1.4 Project Scope

The ESGP is a software system with aims to enable the Encost smart devices to be visualised in a graph data structure from the command line. With both accessibility to Encost verified users and community users but with restricted access to the software capability. There will be no hardware integration required for the ESGP nor for the testing purposed of the software. SoftFlux is responsible to the complete test plan of the ESGP and the insurance that the software meets the requirements by passing the tests outlined in this document.

2 Specialized Requirements Specification

- The local/global dataset location has not be specified within the SDS for the ESHD. The location of the dataset must be found within the same directory location as the ESGP.
- In the SDS the functionality of the exit function was not specified. After discussion with the client it was decided that the exit function will close the ESGP.

3 Black-box Testing

3.1 Categorizing Users

3.1.1 Description

As per the SDS document in section 4.5 the user types are either 'community' or 'encost-verified', however to ensure flexibility due to Encost future expansion this is subject to increase overtime if necessary. To the black box test of the high priority section of categorizing users is error checking. At the beginning of a users experience with the ESGP they may be unaware of the correct keys or input to add to get the required output within the command line by creating a error checking case for vulnerabilities we can ensure that a user will be able to safely begin using the program within causing faults or errors.

3.1.2 Functional Requirements Tested

REQ-1 A prompt should be included that allows the user to indicate whether they are: (1) a member of the community; or (2) a member of Encost;

3.1.3 Test Type

- **Level of Test:** Black-box testing, Unit test
- **Test Technique:** Expected input, edge cases, boundary cases

3.1.4 Test Cases

The ESGP should only pass for two cases '1' and '2' as outlined in section 4.4.1 of the SDS. For all cases listed below the test should fail.

- Error checking cases (must fail all items):
  - ☐ one
  - ☐ two
  - ☐ community
  - ☐ user
  - ☐ encost
  - ☐ Encost
  - ☐ community user
  - ☐ 2 Encost
  - ☐ null/no entry

☐ 11111111111111111111111111111111

☐ COMMUNITY

☐ admin

3.2 ESGP Account Login

3.2.1 Description

For the black box test of the ESGP account login the requirement must pass a decision table test. This decision table will have 15 test cases which are outlined below in table 3.1. The final produced ESGP must pass all 15 test cases outlined in the table and produce the required output. The rules to which this decision table test my follow are outlined in table 3.2, each test case must follow all four rules in order to pass the test.

3.2.2 Functional Requirements Tested

REQ-1 The ESGP Account Login prompt should first prompt the user to enter their username. It should then prompt the user to enter their password.

REQ-2 Once the username and password have been entered, the system should check that the inputs are valid.

3.2.3 Test Type

- **Level of Test:** Black-box testing, Unit test
- **Test Technique:** Decision table with expected inputs and boundary values

3.2.4 Test Cases

	Cases 1-10	**Case 11**	**Case 12**	**Case 13**	**Case 14**	**Case 15**
Username	Encost Given Usernames 1-10	Encost username 1	"username"	Encost password 8	Encost password 6	"admin"
Password	Encost Given Passwords 1-10	Encost password 4	"password"	Encost username 8	Encost username 3	"Encost"

Table 3.1: ESGP account login test cases.

3.3 ESGP Feature Options

3.3.1 Description

Due to the multiple options which the feature options can display and that the options vary due to the the different user types the optimal black box test type is a decision

	Rule 1	Rule 2	Rule 3	Rule 4
Username	Correct	Incorrect	Incorrect	Correct
Password	Incorrect	Correct	Incorrect	Correct
Output	User made aware of error. User type is asked again.	User made aware of error. User type is asked again.	User made aware of error. User type is asked again.	ESGP Feature Options are displayed.

Table 3.2: Rules for the ESGP account login decision table.

table test. Each key stroke labeled in table 3.3 is a case for the decision table. The result is what is expected output of the ESGP. For every key stroke provided the ESGP must produced the expected result in order to pass this black box test.

3.3.2 Functional Requirements Tested

REQ-1 The ESGP Feature Options prompt should provide the user with a selection of features that they can pick from. For a Community User there is only one feature available: visualising a graph representation of the data. For a verified Encost User there are three features: (a) loading a custom dataset; (b) visualising a graph representation of the data; or (c) viewing summary statistics.

3.3.3 Test Type

- **Level of Test:** Black-box testing, Unit test
- **Test Technique:** Decision table with expected inputs and boundary values

3.3.4 Test Cases

The test cases can be seen in table 3.3. This table gives expected outputs for the valid test cases along with boundary values and expectant results. These test cases cover both community and Encost-verified users.

3.4 Loading Encost Smart Homes Dataset

3.4.1 Description

As per the specifications document in section 4.2 the ESHD is loaded in from a CSV formatted file. The black box testing that must be completed for this high priority requirements is a checksum test. When reading in the ESHD it can be tested to ensure the data is being read out correctly by writing the dataset back out to a test CSV file labeled 'test.csv'. Then by checking the MD5 checksum of both files the dataset loading code can be tested.

User Type	Features	Key	Result
Community	Graph Data	1	Graph Stream window opens
	Exit	x	Close ESGP.
	Test case 1	9	Error message is displayed. Feature options re-displayed.
	Test case 2	h	Error message is displayed. Feature options re-displayed.
	Test case 3	e	Error message is displayed. Feature options re-displayed.
	Test case 4	?	Error message is displayed. Feature options re-displayed.
Encost Verified	Custom Dataset	1	Custom dataset function begins
	Graph Data	2	Graph Stream window opens
	Summary Statistics	3	Statistics are output to console
	Exit	x	Close ESGP.
	Test case 5	s	Error message is displayed. Feature options re-displayed.
	Test case 6	5	Error message is displayed. Feature options re-displayed.
	Test case 7	p	Error message is displayed. Feature options re-displayed.
	Test case 8	!	Error message is displayed. Feature options re-displayed.

Table 3.3: Table of ESGP features, key strokes and results

This is a manual test. There is no JUnit test created to automate the functionality of this test.

3.4.2 Functional Requirements Tested

REQ-3 The system should be able to read the Encost Smart Homes Dataset line by line and extract the relevant device information.

3.4.3 Test Type

- **Level of Test:** Black-box testing, Unit test
- **Test Technique:** Manual, Data integrity check

3.4.4 Test Cases

This test is passed when the ESGP can produce an identical MD5 checksum for both the files.

3.5 Categorising Smart Home Devices

3.5.1 Description

To test the categorising of the Smart Home devices both of the two requirements outlined in the SRS document need to be full filled. To test this two CSV formatted datasets are given below. The first dataset should successfully create one object from each device category with all the information stored correctly. The second CSV should be rejected as there is a device category given which is not found in the ESGP. The test CSV can be sound in table 3.4 and table 3.5.

The reading dataset function will create the dataset and inform the ESGP of the success. This will be a Boolean result sent back from the function.

3.5.2 Functional Requirements Tested

REQ-1 The system should determine the device category for each device, based on the information provided on each line of the Encost Smart Homes Dataset (or custom dataset).

REQ-2 The system should create an Object for each device. This object should hold all of the information for that device.

3.5.3 Test Type

- **Level of Test:** Black-Box testing, Unit test
- **Test Technique:** Expected input, edge cases, boundary cases

3.5.4 Test Cases

```
Device ID,Date Connected,Device Name,Device Type,Household ID,Router Connection,Sends,Receives
EWR-1234,25/04/23,Encost Router 360,Router,WKO-1234,-,yes,yes
EHC-2468,25/04/23,Encost Smart Hub,Hub/Controller,WKO-1235,EWR-1234,yes,yes
ELB-4567,25/04/23,Encost Smart Bulb B22 (multi colour),LightBulb,WKO-1236,EWR-1234,no,yes
EK-9867,25/04/23,Encost Smart Jug,Kettle,WKO-1237,EWR-1234,no,yes
ESW-7569,25/04/23,Encost Smart Washer,Washing Machine,WKO-1238,EWR-1234,no,yes
```

Table 3.4: Valid test case for device categorising

```
Device ID,Date Connected,Device Name,Device Type,Household ID,Router Connection,Sends,Receives
EPR-1234,25/04/23,Encost Public Test 360,Test,WKO-1234,-,no,yes
ETT-1324/04/23,Encost Smart Test,Fail,WKO-1235,EwR-1234,yes,no
```

Table 3.5: Valid test case for device categorising

3.6 Building a Graph Data Type

3.6.1 Description

To test the building of a graph data type the test CSV files in table 3.4 and table 3.5 must be used again. To ensure that all data points within the test CSV are loaded correctly each device name will be collected and stored into a list for comparison. The test list should return an empty list as no valid devices are found within the invalid test CSV. The valid test should return the results given in section 3.6.4.

3.6.2 Functional Requirements Tested

REQ-2 All unique data points should be included in the graph.
REQ-3 All households should be represented in the graph.

3.6.3 Test Type

- **Level of Test:** Black-Box testing, Unit test
- **Test Technique:** Expected input, edge cases, boundary cases

3.6.4 Test Cases

Test inputs are stored in table 3.4 and table 3.5.
To pass the test the returned list should be comprised of these device names:

- Encost Router 360
- Encost Smart Hub
- Encost Smart Bulb B22 (multi colour)
- Encost Smart Jug
- Encost Smart Washer

3.7 Graph Visualisation

3.7.1 Description

The black box test for the visualisation of the graph objects it to ensure that the resulting graph in a new UI window is correctly displaying the dataset. Using the given CSV

dataset in table 3.6 the resulting visualisation of the graph should look like figure 3.1. This visualisation following the SDS document and design guide given in section 4.4.4 of the SDS.

This is a manual test which requires no JUnit form. This test ensures that the visual functions provided by the Graph Stream library match the intended design outlined in the SDS document.

3.7.2 Functional Requirements Tested

REQ-1 The graph visualisation must be implemented using the Graph Stream library.

REQ-2 The graph visualisation must show all nodes in the graph data structure.

REQ-3 The graph visualisation must show all connections between nodes in the graph data structure.

REQ-4 The graph visualisation must distinguish between different device categories.

REQ-5 The graph visualisation must, in some way, illustrate each device's ability to send and receive commands from other devices.

3.7.3 Test Type

- **Level of Test:** Black-Box testing, Integration test with Graph Stream Library.
- **Test Technique:** Manual expected case test

3.7.4 Test Cases

```
Device ID,Date Connected,Device Name,Device Type,Household ID,Router Connection,Sends,Recieves
EWR-1234,25/04/23,Encost Router 360,Router,WKO-1234,-,yes,yes
EHC-2468,25/04/23,Encost Smart Hub,Hub/Controller,WKO-1235,EWR-1234,yes,yes
ELB-4567,25/04/23,Encost Smart Bulb B22 (multi colour),LightBulb,WKO-1236,EWR-1234,no,yes
EK-9867,25/04/23,Encost Smart Jug,Kettle,WKO-1237,EWR-1234,no,yes
ESDP-7569,20/03/23,Encost Smart Dishwasher Pro,Washing Machine,WKO-8451,EWR-1234,no,yes
EWRP-4759,21/03/23,Encost Router Plus,Router,WKO-8452,-,yes,yes
EHC-2468,22/03/23,Encost Smart Hub,Hub/Controller,WKO-8453,EWRP-4759,yes,yes
ELB-7419,23/03/23,Encost Smart Bulb B22 (white),LightBulb,WKO-8454,EWRP-4759,no,yes
EWK-3842,24/03/23,Encost Smart Whistling Kettle,Kettle,WKO-8455,EWRP-4759,no,yes
```

Table 3.6: CSV formatted data for graph visualisation testing

3.8 Calculating Device Distribution

3.8.1 Description

In order to calculate the device distribution the ESGP must use the graph data structure class and its information. The requirements for this feature as outlined in section 4.9 of the SRS document detail that the system must calculate the number of devices in each

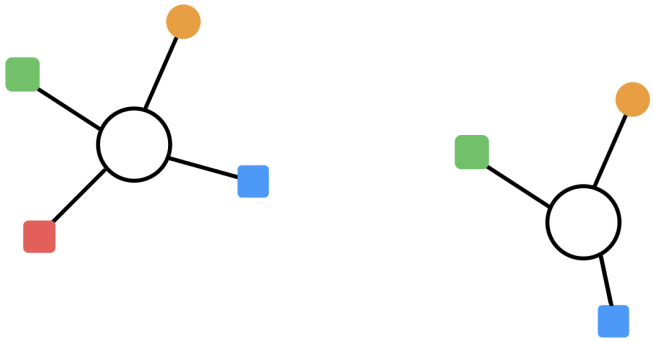

Figure 3.1: The ESGP graph visualisation example

category and also for each type of device. The SDS document outlines how this information must be displayed to the CLI in figure 4.7. This shows they must be separated by category, type and name of each device with the calculated number to the right.

To test the calculations required for the device distribution a boundary value analysis needs to be used. This is to ensure so and valid dataset that can be loaded the calculations for the device distribution should hold. The cases are outlined below for the analysis.

3.8.2 Functional Requirements Tested

REQ-1 The system should use the information stored in the graph data structure to calculate the number of devices that exist in each device category. REQ-2 For each device category, the system should also calculate the number of devices that exist for each device type.

3.8.3 Test Type

- **Level of Test:** Black-Box testing, Unit test
- **Test Technique:** Boundary Values Testing

3.8.4 Test Cases

Note: As per the SDS document the device names must also be listed and calculated, however for testing simplicity the name distribution of the above devices can be up to the digression of the tester when making the CSV files required for testing.

Minimum	Above Minimum	Normal	Below Maximum	Maximum
2x Router	3x Routers	15x Routers	80x Routers	100x Routers
1x Extender	3x Extender	5x Extenders	80x Hub/controllers	100x Extenders
	1x Light Bulb	6x Hub/controllers	80x Light Bulb	100x Hub/controllers
	1x Coffee Maker	12x Light Bulb	80x Strip Lighting	100x Light Bulb
	1x Dishwasher	4x Strip Lighting	80x Kettle	100x Strip Lighting
		18x Other Lighting	80x Toaster	100x Other Lighting
		20x Kettle	80x Coffee Maker	100x Kettle
		24x Toaster	80x Washing Machine\Dryer	100x Toaster
		12x Coffee Maker	80x Refrigerator\Freezer	100x Coffee Maker
		17x Washing Machine\Dryer		100x Washing Machine\Dryer
		32x Refrigerator\Freezer		100x Refrigerator\Freezer
		17x Dishwasher		100x Dishwasher

Table 3.7: Boundary Values for device distribution

4 White-box testing

4.1 Device Distribution Pseudocode

The device distribution pseudocode is written to meet the requirements of the SDS outlined in the SDS figure 4.7. This pseudocode makes use of the graph class methods 'getDevicesByCategory()', 'getDevicesByType()', 'getDevicesByProduct()' as stated in 4.1.2 of the SDS documentation. Below in algorithm 1 is the described pseudocode for device distribution.

The pseudocode found in algorithm 1 only focuses on the calculation code not the display code.

4.2 Branch Coverage Testing

4.2.1 Description

The branch coverage test will be parameterized unit tests. Within the device distribution pseudocode shown in algorithm 1 there are 17 branches which must be covered. These branches cover each given device category and type from Encost. Because of the device class structure shown in section 4.1.1 of the SDS document each device has its category and type internally stored. Due to this for each device which is counted within algorithm 1 is considered a branch which needs to be covered. The test inputs are within table 4.1. Table 4.1 is a list of the branches which give 100% coverage to algorithm 1. For testing a CSV file containing one object of each type listed in table 4.1 will ensure the coverage of every category and type for the device distribution code.

Algorithm 1 Device Distribution Pseudocode

```
1: function Public DeviceDistribtion(Graph graph)
2:     INITIALISE empty array of length 12, for device types.
3:     INITIALISE array with all device types.
4:     INITIALISE empty array of length 5, for device categories.
5:     INITIALISE array with all device categories.
6:     INITIALISE empty array of length 32, for products.
7:     INITIALISE array with all device products.
8:     INITIALISE a total counter for type, category and product.
9:     Run the for loops within a try-catch structure to ensure that errors are caught.
10:    for every category do
11:        CALL getDevicesByCategory(category)
12:        categoryArray[currentIndex] ← categoryList.size()
13:        categorySum+ = categoryList.size()
14:    end for
15:    for every type do
16:        CALL getDevicesByType(type)
17:        typeArray[currentIndex] ← typeList.size()
18:        typeSum+ = typeList.size()
19:    end for
20:    for every product do
21:        CALL getDevicesByProduct(product)
22:        productArray[currentIndex] ← productList.size()
23:        productSum+ = productList.size()
24:    end for
25:    Check the sum of all products = sum of all types = sum of all categories
26:    if if sums do not match then
27:        Throw error
28:
```

4.2.2 Functional Requirements Tested

REQ-1 The system should use the information stored in the graph data structure to calculate the number of devices that exist in each device category.

REQ-2 For each device category, the system should also calculate the number of devices that exist for each device type.

4.2.3 Test Type

- **Level of Test:** White-Box testing, Unit test
- **Test Technique:** Branch Coverage

4.2.4 Test Cases

Test inputs are stored in table 3.4 and table 3.5.

Test Branch	Category	Type	Coverage
1	WiFiRouter	Router	2/17
2	WiFiRouter	Extender	1/17
3	Hub/Controller	Hub	2/17
4	Lighting	Light Bulb	2/17
5	Lighting	Strip Lighting	1/17
6	Lighting	Other Lighting	1/17
7	Appliance	Kettle	2/17
8	Appliance	Toaster	1/17
9	Appliance	Coffee Maker	1/17
10	Whiteware	Washer\Dryer	2/17
11	Whiteware	Fridge\Freezer	1/17
12	Whiteware	Dishwasher	1/17

Table 4.1: Branch coverage for device distribution test

5 Mutation Testing

5.0.1 Description

The main areas which require mutation within the white-box test pseudocode is the type and category checking. The device categories, types and products are returned as a list object. Then the length of each list is stored as the count for that device. The mutations which can be found are below are decision mutations to the logic of the function. The mutations below will be used to test the device distribution function.

5.0.2 Test Type

- **Level of Test:** White-Box testing, Unit test
- **Test Technique:** Mutation Testing

5.1 Mutant #1

The first mutation for the white box test is shown on line 2 of algorithm 2. This mutation is simply to change the array size for the categories count. This mutation will

Device Categories	Count	Device Type	Count
Wifi Router	2	Router	1
Hub/Controller	1	Extender	1
Lighting	3	Hub	1
Appliance	3	Lightbulb	1
Whiteware	3	Strip Lighting	1
Total	12	Other Lighting	1
		Kettle	1
		Toaster	1
		Coffee Maker	1
		Washer\Dryer	1
		Fridge/Freezer	1
		Dishwasher	1
			12

Table 4.2: The expected output of the branch coverage test.

only increase the size of the array by 1.

- int[] categoryArray = new int[6]

The returned array will be larger than the expected array but there will not be a category to collect information on within the loop. Due to loop increments being the array with the category names the loop should close before the 6th index of the count array.

5.2 Mutant #2

Mutation 2 is shown on line 11 of algorithm 2. This mutation is a statement mutation. The category count is changed to a type count.

- typeArray[index] += category.size()

Mutation two should be caught by the check sum count at the end of the function. In each graph the number of devices will remain constant during the time the distribution is calculated (more devices can be added but should not be during this point of the ESGP). Thus the sum of categories, products and types should be equal unless an error has occurred. If such an error does occur then an appropriate error should be thrown.

5.3 Mutant #3

On line 7 of algorithm 2 is the the 3rd mutation test. This test is a statement mutation which increments the index of the category array.

- categoryArray[index + 1] = categoryList.size();

Algorithm 2 Device Distribution Pseudocode Mutations

```
1: function PUBLIC DEVICEDISTRIBTION(Graph graph)
2:     INITIALISE empty array of length 12, for device types.
3:     INITIALISE array with all device types.
4:     INITIALISE empty array of length 6, for device categories          ▷ Mutation 1
5:     INITIALISE array with all device categories.
6:     INITIALISE empty array of length 32, for products.
7:     INITIALISE array with all device products.
8:     INITIALISE a total counter for type, category and product.
9:     + Run the for loops within a try-catch structure to ensure that errors are caught.
10:    for every category do
11:        CALL getDevicesByCategory(category)
12:        categoryArray[currentIndex + 1] ← categoryList.size()          ▷ Mutation 3
13:        categorySum+ = categoryList.size()
14:    end for
15:    for every type do
16:        CALL getDevicesByType(type)
17:        typeArray[currentIndex] ← categoryList.size()                  ▷ Mutation 2
18:        typeArray[currentIndex] ← typeList.size()                      ▷ Mutation 4
19:        typeSum+ = typeList.size()
20:    end for
21:    for every product do
22:        CALL getDevicesByProduct(product)
23:        productArray[currentIndex] ← productList.size()
24:        productSum+ = productList.size()
25:    end for
26:    Check the sum of all products = sum of all types = sum of all categories
27:    if if sums do not match then
28:        Throw error
29:
```

This mutation should be caught by the try-catch structure within the device distribution function. The try-catch structure should be caught on the last device with an index array out of bounds error and handled accordingly to the developer.

5.4 Mutant #4

Mutation 4 is on line 12 of algorithm 2 and it is a repetition of line 11. This should mutate the output of every device type tested.

- typeArray[index] = type.size()

Mutation four should be caught at the end of the function when the sums are compared. Much like mutation two the sum of categories, products and types should be equal unless

an error has occurred. If such an error does occur then an appropriate error should be thrown.

5.5 Mutation Score

Using the two input test sets below the mutation test scores can be calculated. In table 5.1 we can see the coverage of the mutations. Using the score calculation below the mutation score is 100% as all mutations should be caught by the code provided in algorithm 2.

Mutation score = (number of killed mutants/total number of mutants) x 100
Mutation score = (4/4) x 100 = 100%

- Input Set 1: Toaster, Toaster, Kettle, Router, Extender, Fridge, Dishwasher, Light bulb, Lightbulb, stripLight, kettle, freezer
- Input Set 2: Toaster, Extender, Fridge, Dishwasher, Lightbulb, Lightbulb, Extender, Hub, otherLighting, Kettle, Washer, Toaster

	Mutation 1	**Mutation 2**	**Mutation 3**	**Mutation 4**
Input 1	Pass	Pass	Pass	Pass
Input 2	Pass	Pass	Pass	Pass

Table 5.1: Mutation testing results

5.6 Conclusion

This functional software test plan uses the software design specification document SDS_-1. This document is to be used as a basis for testing the Encost Smart Graph Project along with being used as a guideline for developer when building the ESGP.