FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.1

Prepared by: Student 3
SoftFlux Engineer

SoftFlux

May 6, 2023

Contents

List of Figures

List of Tables

Revision History

Name	Date	Reason for Changes	Version
SoftFlux Engineer	06/05/23	Completed Functional Test Plan	1.0
SoftFlux Engineer	07/05/23	Correction to white box testing	1.1

1 Introduction/Purpose

1.1 Purpose

The purpose of this document is to provide an overview of the testing for the Encost Smart Graph Project. This document includes black box tests and white box test. This document also comes with a Functional Test Suite in JUnit.

1.2 Document Conventions

The document will use the following conventions:

ESGP:	Encost Smart Graph Project
CLI:	Command Line Interface
CL:	Command Line
UI:	User Interface
EOF:	End of File
ESH:	Encost Smart Home
SDS:	Software Design Specifications

1.3 Intended Audience and Reading Suggestions

This document is intended for use by any individual user, developer, or tester. Each reader type has their own individual uses for this document:

- User: may use this document to ensure the software works as expected
- Developer: may use this document to aid in the development of the software

- Tester: may use this document to develop tests for the software

1.4 Project Scope

This Functional Software Test Plan was created using SDS #2. The black box tests in this document only cover the high priority requirements covered in the SRS. The white box tests only cover one of the summary statistics - and is only expected to cover branch coverage testing and mutation testing.

2 Specialized Requirements Specification

Below covers points regarding testing that has either been; mentioned in the SDS, slightly altered from the SDS, or are just worth mentioning. As well as notes on the testing datasets.

Points mentioned in the SDS:

- If any invalid CLI inputs are found, inform the user their input is invalid and re-prompt the user
- The user can type 'home' to return to the feature options page of the respective edition (community or encost-verified) they are using

Alterations to points mentioned in the SDS:

- For user type and feature selection input validation, the SDS states "sub-strings of the feature, the number of the feature, or 'c/community'" should be expected. This test plan ignores the 'sub-strings', and instead suggests one key word.
- For input validation, the SDS also states that "at any point within the ESGP if a user enters 'back' the CLI will return to the previous prompt". Due to the user potentially mistyping and needing to use the backspace character; as well as the need to read the users input as it is typed, this has also been changed. Instead the word 'back' must be entered.

Further points worth mentioning:

- If there is an issue with one line of the dataset, an appropriate message should be displayed to the user at the end of reading the file, the rest of the file should be read as usual.
- Where 'displays error message' is an outcome, an appropriate and clear error message is expected which states the error. For example: file not found should display an error which states that the file has not been found.

- The SDS does not discuss how the graph data structure is implemented. The test plan assumes this object is created within the CommunityUser class (and is inherited in the EncostUser class).

Multiple testing datasets can be found in the Appendix. These are created to have everything be represented within a small dataset for testing purposes. Different tests may use different testing datasets to fit the requirements of the test. Ideally, two datasets would be used for each test to ensure that values have not been hardcoded to cater towards one dataset.

3 Black-box Testing

3.1 Categorising Users

The purpose of this test is to ensure users are correctly categorised.

The level of this test is unit. This is because only one unit (function) is involved in this process.

The test technique used is a decision table. This is shown in table 3.1. The test inputs are valid inputs for community user, valid inputs for encost user, and invalid inputs for both. According to the SDS, the valid inputs for this are: 1, 2, e, encost, c, and community.

These tests involve 3 conditions to check: the validity of the input, the correct outcome has been displayed, and the created user object is of the correct class (CommunityUser or EncostUser).

Input	Type	Outcome	User-type
1	Valid	Displays feature options	community
c	Valid	Displays feature options	community
C	Valid	Displays feature options	community
community	Valid	Displays feature options	community
COMMUNITY	Valid	Displays feature options	community
2	Valid	Displays Encost login prompt	encost
e	Valid	Displays Encost login prompt	encost
E	Valid	Displays Encost login prompt	encost
encost	Valid	Displays Encost login prompt	encost
ENCOST	Valid	Displays Encost login prompt	encost
3	Invalid	Display error message and re-prompt user	-
user	Invalid	Display error message and re-prompt user	-
	Invalid	Display error message and re-prompt user	-
home	Invalid	Display error message and re-prompt user	-

Table 3.1: Decision table for indicating user type

As well as displaying the appropriate next steps for valid input (feature options for community users and login prompt for Encost users), the user-type needs to be stored. Following the SDS, for a community user, a CommunityUser object is created for the user; and for an Encost user, an EncostUser object is created.

3.2 ESGP Account Login

The purpose of this test is to ensure login is only successful with a valid username and password combination.

The level of this test is integration. This is because multiple units are involved in the username/password verification process. The encryption of the entered password is one unit, and then username and encrypted password verification is another unit.

The test technique used is a decision table. This is shown in table 3.2. The test inputs are the different username and password combinations:

- Incorrect username and password
- Incorrect username and correct password (for other user)
- Correct username and incorrect password
- Correct username and password

Username	Password	Outcome
false	false	Display error message and re-prompt user
false	true	Display error message and re-prompt user
true	false	Display error message and re-prompt user
true	true	Display features options and set user type

Table 3.2: Decision table for ESGP account login

3.3 ESGP Feature Options

The purpose of this test is to ensure valid input and features are chosen as expected.

The level of this test is integration. This is because multiple units are involved in this process. One unit for validating input,and multiple for starting the feature.

The test technique for this is two input/output tables. Two tables are needed because the community user and Encost user have different options available to pick and therefore will have different tests.

The first thing to test is that the correct feature options have been displayed. Table 3.3 shows what features should be displayed for each user type.

User type	Features displayed
Community	(1) Data graph visualisation
Encost	(1) Load custom dataset
	(2) Data graph visualisation
	(3) View summary statistics

Table 3.3: Features to display based off user type

The second thing to test is that invalid inputs are appropriately handled and that valid inputs result in the correct next stage. Table 3.4 shows the decision table for the community user feature selection, and Table 3.5 shows the decision table for the Encost user feature selection.

The SDS states that "all subsets of the feature" should be able to be correctly entered. This had been changed to allow the number, full phrase, or the main key work only. The invalid tests for the community features need to ensure that entering Encost features do not result in those being displayed.

Input	**Type**	**Outcome**
1	Valid	Display graph message to console and open graph
data graph visualisation	Valid	Display graph message to console and open graph
DATA GRAPH VISUALISATION	Valid	Display graph message to console and open graph
graph	Valid	Display graph message to console and open graph
2	Invalid	Display error message and re-prompt user
3	Invalid	Display error message and re-prompt user
load custom dataset	Invalid	Display error message and re-prompt user
load	Invalid	Display error message and re-prompt user
view summary statistics	Invalid	Display error message and re-prompt user
home	Valid	Re-display feature options
HOME	Valid	Re-display feature options
back	Valid	Display user selection prompt

Table 3.4: Decision table of feature selection for community users

Input	**Type**	**Outcome**
1	Valid	Display prompt for dataset file path
load custom dataset	Valid	Display prompt for dataset file path
LOAD CUSTOM DATASET	Valid	Display prompt for dataset file path
load	Valid	Display prompt for dataset file path
2	Valid	Display graph message to console and open graph
data graph visualisation	Valid	Display graph message to console and open graph
graph	Valid	Display graph message to console and open graph
3	Valid	Display summary statistics
view summary statistics	Valid	Display summary statistics
summary	Valid	Display summary statistics
4	Invalid	Display error message and re-prompt user
load graph	Invalid	Display error message and re-prompt user
home	Valid	Re-display feature options
HOME	Valid	Re-display feature options
back	Valid	Display user selection prompt

Table 3.5: Decision table of feature selection for Encost users

The feature should only be considered to have passed if every input correctly maps to the given output. If a single test fails, the entire feature is considered to have failed.

Note: 'graph message' as stated in the SDS is "Opening graph UI window for X.csv" where X is the filename.

3.4 Loading the Encost Smart Homes Dataset

The purpose of this test is to ensure that the Encost Smart Home Dataset is open, read, and processed correctly.

The level of this test is unit. This is because this feature only involves one unit (function) - opening and reading the file.

The test technique used is and event and outcome table.

The dataset should be called **'ESH_dataset.csv'** and should be located in the same directory as the ESGP source program in a directory called **'datasets'**.

Event	**Type**	**Outcome**
File location unknown	Invalid	Displays error message
File not found	Invalid	Displays error message
File wrong format (not .csv)	Invalid	Displays error message
File already open	Invalid	Displays error message
File opens correctly	Valid	Proceed to read file
Line not enough arguments	Invalid	Continues to next line and at EOF, display error message
Line contains invalid data types	Invalid	Continues to next line and at EOF, display error message
Line contains repeating device ID	Invalid	Continues to next line and at EOF, display error message
Line reads correctly	Valid	Create device object (Section 3.5)
End of file	Valid	Close file, no CL output

Table 3.6: Decision Table regarding loading and reading from the dataset

To test these, as shown in the Functional Test Suite, the variable storing the location of the ESH dataset should be made temporarily public. This allows the tests to change the file to test for the events listed in Table 3.6.

3.5 Categorising Smart Home Devices

The purpose of this test is to ensure a device object is created for each device, with the correct device category.

The level of this test is integration testing. This is because there are 2 units involved in this process - getting the category from the type, and creating the device object.

The test technique used is and input/output table.

The process of this test would involve loading the testing datasets in the place of the ESH dataset. This file is read and device objects are created for each line. The list of these device objects should be iterated through and the method returning the devices ID and category should be used for the testing. These should be compared to Table 3.7 to ensure all device categories are correct. The test should only be considered to have passed if all of the device IDs and categories match.

Device ID	Category
EWR-1234	Encost Wifi Routers
ELB-4567	Encost Smart Lighting
EK-9876	Encost Smart Appliances
EHC-2468	Encost Hubs/Controllers
ESW-5555	Encost Smartware
EWR-2345	Encost Wifi Routers
ESW-3333	Encost Smartware

Table 3.7: Device categories for each device in Testing Dataset 1

3.6 Building a Graph Data Type

The purpose of this test is to ensure devices are stored in a graph data structure.

It is not mentioned in the SDS how the graph data type is implemented or where it is stored. This document assumes the object is located in the CommunityUser class.

The level of this test is multiple unit tests. One for checking that the device has the correct category for its type. This can test the method that is given a type and returns the category. The table containing the device category and type can be found in the SRS. The second test checks that every device is in the structure.

Using section 6.2, the Graph data structure should contain 2 node with the IDs - EWR-1357 and ELB-3579.

These tests should include:

- Checking that each device has the correct category for its type.
- Checking every device is in the structure

3.7 Graph Visualisation

The purpose of this test is to ensure the data is correctly represented using the GraphStream library.

The level of this test is integration. This is because multiple units are involved in the whole visualisation process.

To test this, a testing dataset should be used which covers all requirement combinations for the graphing feature. This dataset has a list of what the graph needs to include. The test should only be considered to pass if **all** of the requirements are met.

For the dataset (Section 6.1), the following requirements should be met:

- Nodes (7)
  - Rounded square: EWR-1234
  - Rounded square: EWR-2345
  - Cross: EHC-2468

  - Circle: ELB-4567
  - Square: EK-9876
  - Diamond: ESW-5555
  - Diamond: ESW-3333
- Connections (5)
  - EWR-1234 to ELB-4567: arrow towards ELB
  - EWR-1234 to EK-9876: arrow towards EWR
  - EWR-1234 to EHC-2468: arrow both ways
  - EWR-1234 to ESW-55555: arrow towards ESW
  - EWR-2345 to ESW-3333: arrow both ways
- Groups (2)
  - 1 group with router EWR-1234 and devices: ELB-4567, EK-9876, EHC-2468 and ESW-5555
  - 1 group with router EWR-2345 and device: ESW-3333

Note: as this is a visual test, a JUnit test in the test suite does not exist for this test.

3.8 Calculating Device Distribution

The purpose of this test is to ensure the distribution of devices across each device category and type is correctly calculated.

The level of this test is integration tests. This is because there are multiple units involved for the entire feature - finding types for each category; and calculating the amount per type.

The test technique for this is a input/output table for the overall feature. This checks that everything is calculated and shown as expected.

The first thing to test is that all categories and types are shown to the CL, with the types being grouped with the correct category. This table linking the device types to their category is located in the SRS (Section 4.7).

Using the dataset Section 6.1, Table 3.8 shows what numbers are expected for each category and type. For types not listed, the count is expected to be '0'.

Category/Type	**Count**
Encost Wifi Routers	2
>> Router	2
Encost Hubs/Controllers	1
>> Hub/Controller	1
Encost Smart Appliances	1
>> Kettle	1
Encost Smart Lighting	1
>> Light bulb	1
Encost Smartware	2
>> Washing	2

Table 3.8: Expected device distribution

4 White-box testing

The purpose of this test is to ensure every path through the summary statistic 'Calculating Device Location' has been tested, and that correct outputs are produced.

The level of this level is multiple unit tests. Where each unit test is testing one of the four functions required in this statistic.

The test should include a description of the mutations, the pseudo-code for the mutations, a series of test case inputs, the corresponding test case outputs, the level of the test, and the test technique(s). The mutation score must be calculated and justification and/or a description for why these are good mutations should be included.

The summary statistic chosen is 'Calculating Device Location'. This involves calculating statistics grouping data into their geographical location. This summary statistic can be broken down into four methods, each of which can have branch coverage testing. Providing the other routines to load and store the data have been tested to work, it can be safe to assume that the data is in a state which could not cause errors caused from the data.

Due to late changes in the test plan, for dealing with division involving 0, the if statement shown in Section 4.3 should be implemented in the two other flow charts, pseudocode, and branch coverage. This is to prevent dividing by zero errors occuring when there are no households in a region.

Public lists with definitions:

- List<String> regions: the ISO code of the regions in alphabetical order
- List<String> categories: the different Encost device categories in alphabetical order

- List<int> numHouseholds: number of unique households in each region (indicated by index - index 0 is AUK, 1 is BOP, ...)
- List<int> numDevices: total number of devices in each region (indicated by index)
- List<List<int>> numDevicesPerCategory: total number of devices in each category in each region. The first index indicates the region, and the second index indicates the category

4.1 Get device data

The function of this method is to fill lists with appropriate information which can then be extracted by later methods to output the statistics to the console.

The pseudocode of the method is as follows:

```
function getDeviceData
    for each device
        get device region
        find region index in list
        add one to num devices at same index
        get device type
        if device type is a router
            increment households for the region
        else
            get device category
            find category index in list
            add one to list[region index][category index]
        end if
    end for
end function
```

Figure 4.1 shows the flow diagram of this method, with each branch to cover being labelled with a letter in a red box.

The dataset that provides 100% branch coverage for this test is in Section 6.2. Table 4.1 shows that this provides full branch coverage. Before iterating through the devices occurs, branches A and B are covered. The first device, which is a router, covers the branches C, D, E, F, and G; and then after the if, it covers H and I, and N to continue to the next device. The second device, which is not a router, also covers branches C - G; but then after the if, it covers branches J, K, L and M. This is the final device, so the final if is false and branch O is covered. Because each of the branches shown in Figure 4.1 is covered using this dataset, it has 100% branch coverage.

Stage	Branches Covered
1 (before iteration)	A, B
2 (router)	C, D, E, F, G, H, I, N
3 (non-router)	C, D, E, F, G, J, K, L, M
4 (no more devices)	O

Table 4.1: Branch coverage

To test that this method has worked as expected, the lists should be as follows:

- numHouseholds[1] = 1, all other indexes should be 0
- numDevices[1] = 2, all other indexes should be 0
- numDevicesPerCategory[1][2] = 1, all other indexes should be 0

If any of these requirements are not met, the test is considered to be failed.

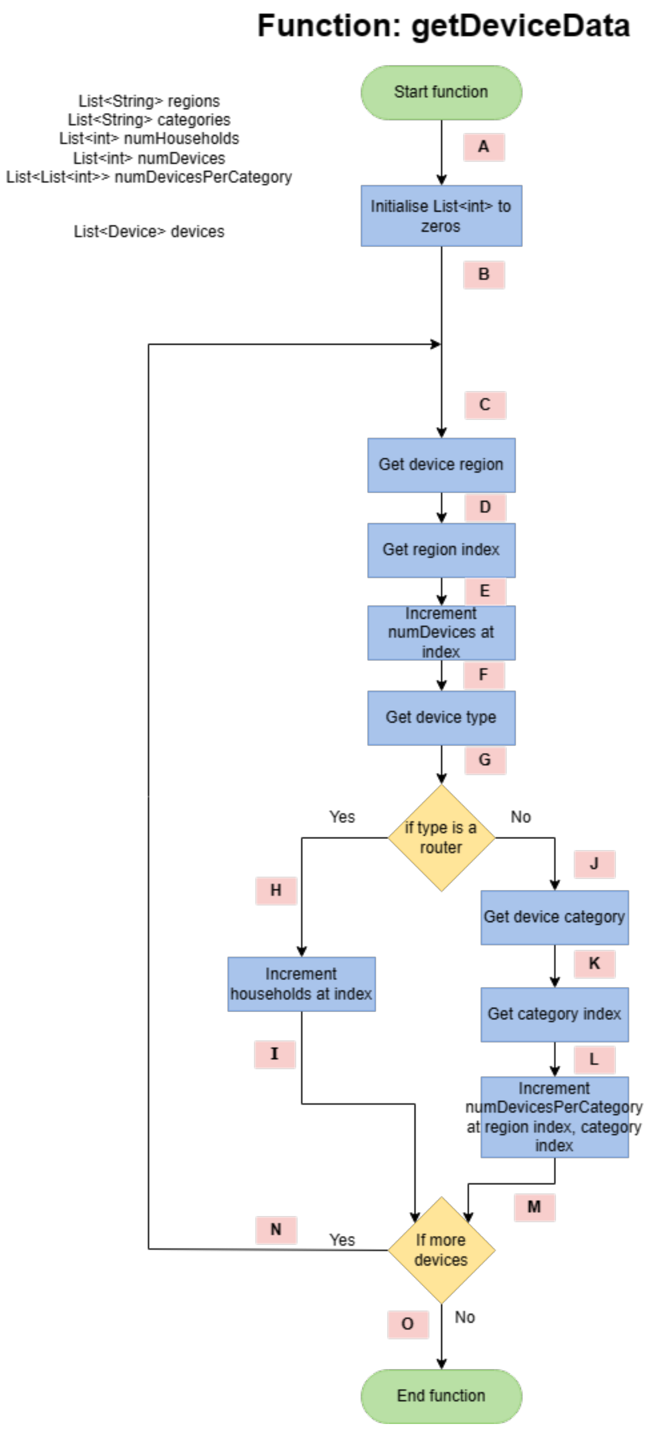

Figure 4.1: Branches for function *getDeviceData()*

4.2 Devices per household by region

The function of this method is to perform internal calculations and to output the results to the console of the number of devices in each region. The pseudocode of the method is as follows:

```
function devicesPerHouseholdByRegion
    print header for table
    print divider for table
    for each region
        calculate average devices (devices per region/households per region)
        print region
        print num households
        print num devices
        print avg devices
        print divider
    end for
end function
```

Figure 4.2 shows the flow diagram of this method, with each branch to cover being labelled with a letter in a red box.

The dataset that provides 100% branch coverage for this test is in Section 6.2 (all other valid testing dataset provide 100% branch coverage due to branches not being determined by data, but to compare correct outputs, this dataset should be used). Table 4.2 shows that this provides full branch coverage. Before iterating through the regions, branches A, B, and C are covered. For the first region (AUK), branches D, E, F, G, H, I, and J are covered. As there is are more regions, it loops and branch K is covered. The rest of the regions (excluding the last - WTC) cover the same branches. The final region covers branches D-J, but as there are no more regions, it breaks the loop and branch L is covered. Because each of the branches shown in Figure 4.2 is covered, it has 100% branch coverage.

Stage	Branches Covered
1 (before iterating)	A, B, C
2 (AUK region)	D, E, F, G, H, I, J, K
3 (more regions)	D, E, F, G, H, I, J, K
4 (WTC/last region)	D, E, F, G, H, I, J, L

Table 4.2: Branch coverage

To test that this method has worked as expected, the console output should contain the following information:

- AUK region: 1 household, 1 total device, 1 average device
- BOP region: 2 households, 3 total devices, 1.5 average devices

- All other regions should have 0 for all

If any of these requirements are not met, the test is considered to be failed.

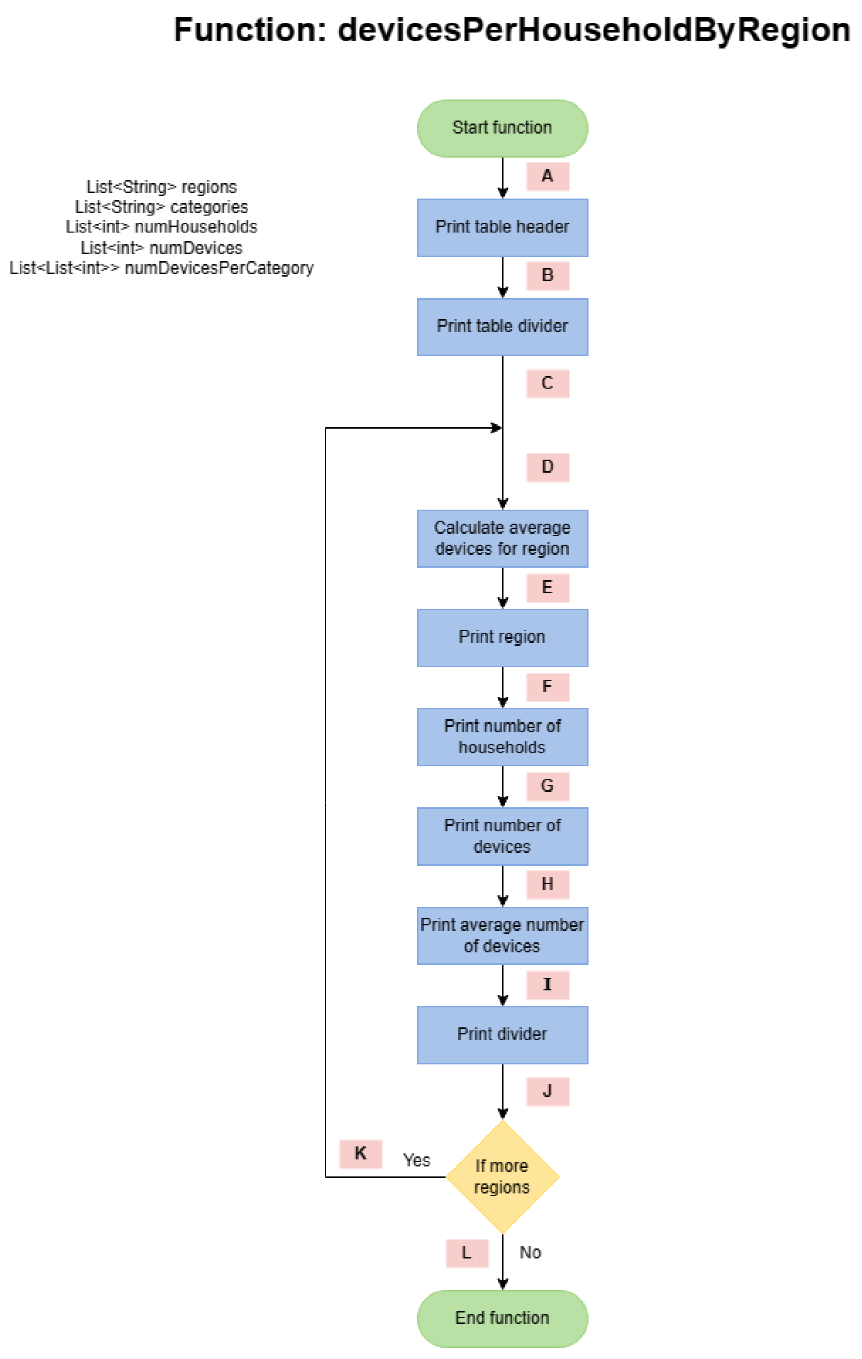

Figure 4.2: Branches for function *devicesPerHouseholdByRegion()*

4.3 Devices per household by category

The function of this method is to perform the internal calculations and output to the console the results for average number of devices in each household for each category in each region. The pseudocode of the method is as follows:

```
function devicesPerHouseholdByCategory
    print header for table
    print table divider
    for each region
        print region
        for each category
            print category
            calculate average per household
                ^ (list[region][category]/ households[region])
            print average devices
        print divider
        end for
    end for
end function
```

Figure 4.3 shows the flow diagram of this method, with each branch to cover being labelled with a letter in a red box.

The dataset that provides 100% branch coverage for this test is in Section 6.4. Table 4.3 shows that this provides full branch coverage. Before iterating, branches A, B, and C are covered. Going through the first region (AUK), branches D and E are covered. Going through the first category for AUK, goes through branches F, G, H, I, K and L. As there are more categories remaining for the region, it also covers M. For the last category for AUK, it covers F-I, K and L, and because there are no more categories, it covers branches N and O. There are more regions so P is also covered. The rest of the regions cover these same branches, excluding the last region (WTC). For any region that has no households, it covers branches D-G, then covers H and J through the if statement. It then covered L, and then M for the next category. The last category for the last region covers branches F-I, K and L, then as there are no more categories branches N and O are covered. There are no more regions, so Q is also covered. Because each of the branches shown in Figure 4.3 is covered, it has 100% branch coverage.

If the dataset is a full dataset, with every region/household/category has values in it, there is only 88% coverage (15/17). This is because the branches to prevent dividing by 0 errors are not executed due to the data being full. This means to test full branch coverage, a partial dataset must be used - where an entire region has no households.

Stage	Branches Covered
1 (before iterating)	A, B, C
2 (AUK region)	D, E
3 (First category for AUK)	F, G, H, I, K, L
4 (Last category for AUK)	F, G, H, I, K, N, O, P
5 (Any region with no household)	D, E, F, G, H, J, L..
5 (Last region last category)	F, G, H, I, K, L, N, O, Q

Table 4.3: Branch coverage

To test that this method has worked as expected, the console output should contain the following information:

- AUK region: 1 Smartware, 0 all other categories
- BOP region: 1 Smart Appliances, 0.5 Smartware, 0 all other categories
- All other regions have 0 for all categories

If any of these requirements are not met, the test is considered to be failed.

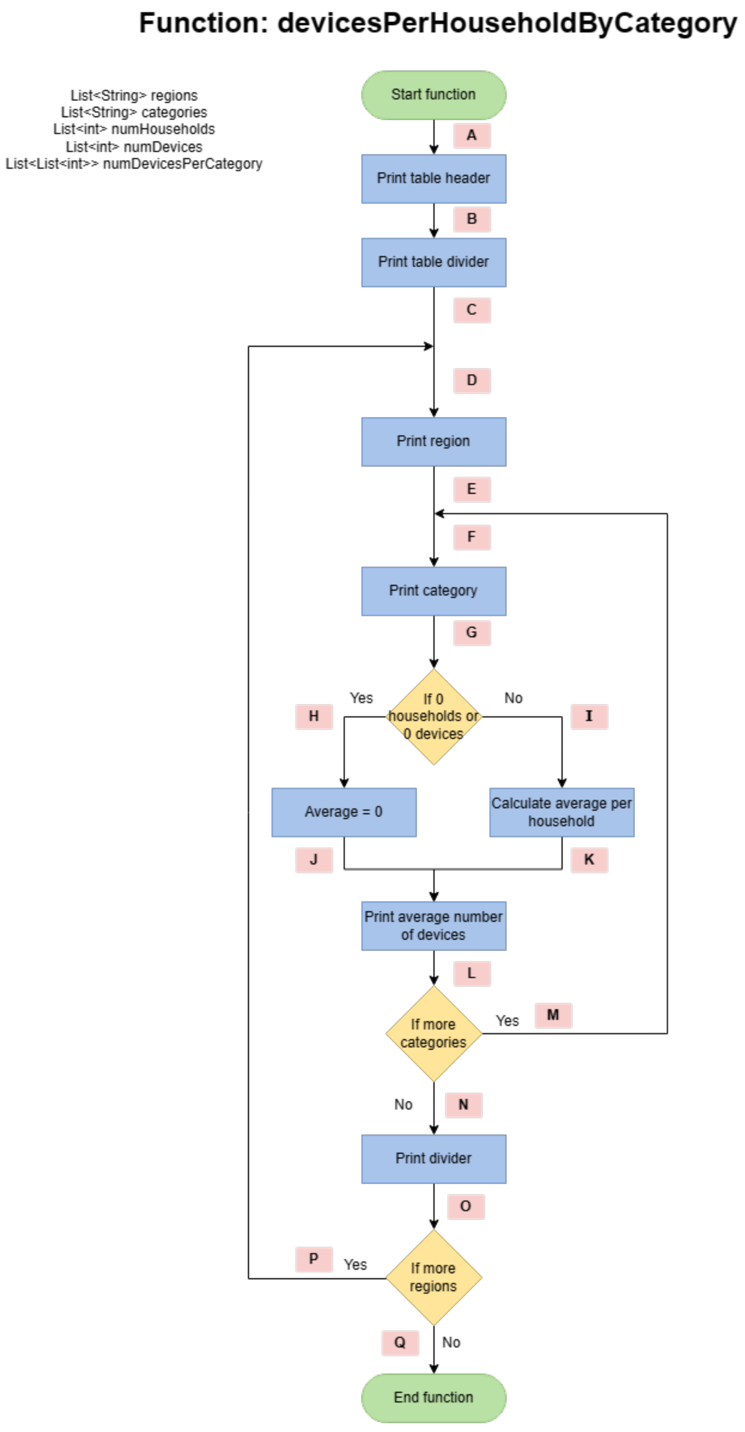

Figure 4.3: Branches for function *devicesPerHouseholdByCategory()*

4.4 Devices per region by category

The function of this method is to perform the internal calculations and output to the console the results for number of devices for each category in each region. The pseudocode of the method is as follows:

```
function devicesPerRegionByCategory
    print header for table
    print table divider
    for each region
        print region
        for each category
            print category
            print devices list[region][category]
        print divider
    end for
end function
```

Figure 4.4 shows the flow diagram of this method, with each branch to cover being labelled with a letter in a red box.

The branch coverage and dataset is the same as Section 4.3.

To test that this method has worked as expected, the console output should contain the following information:

- AUK region: 1 Smartware, 0 all other categories
- BOP region: 2 Smart Appliances, 1 Smartware, 0 all other categories
- All other regions have 0 for all categories

If any of these requirements are not met, the test is considered to be failed.

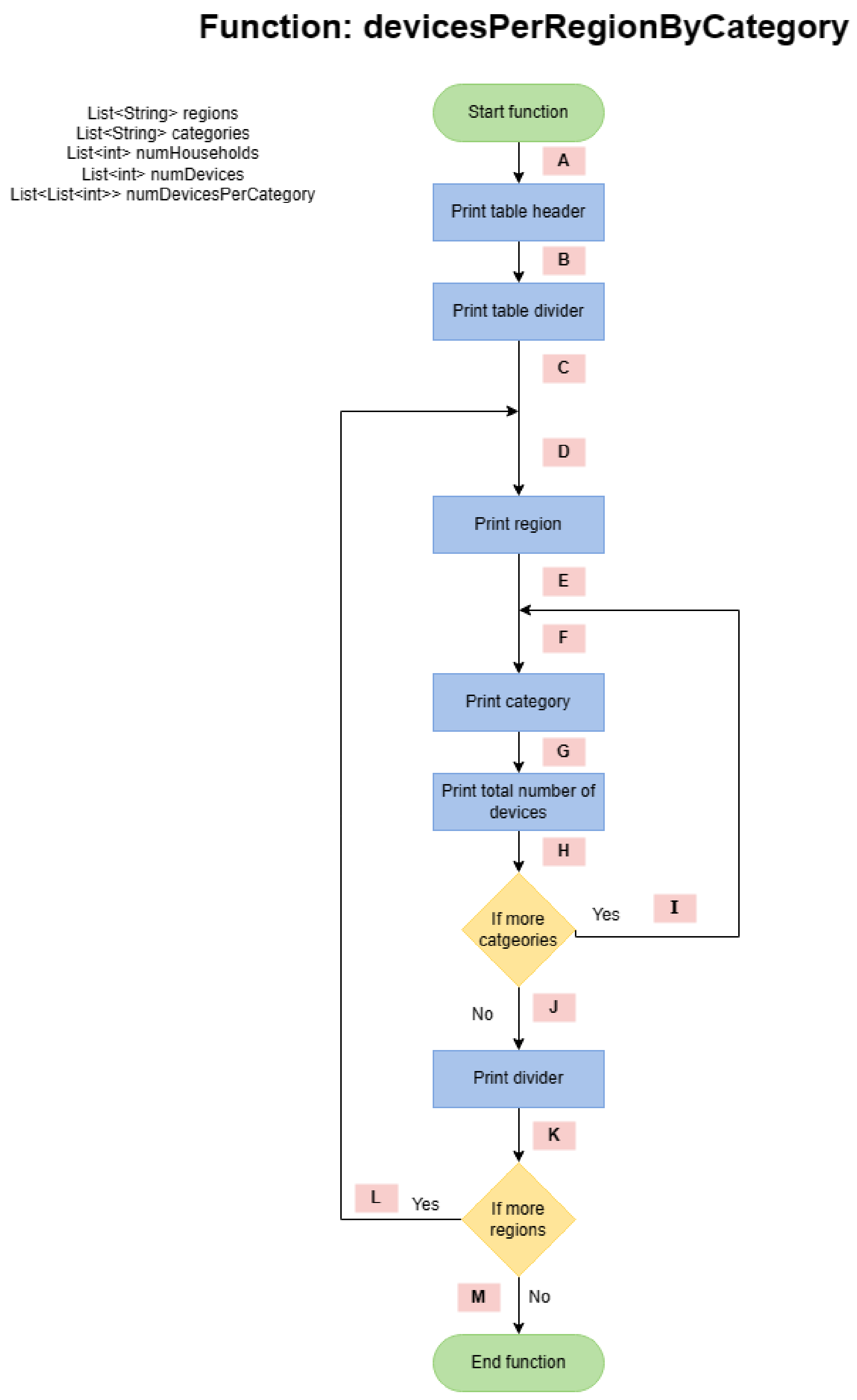

Figure 4.4: Branches for function *devicesPerRegionByCategory()*

5 Mutation Testing

Mutants are specifically chosen for Section 4.3 for ease of comparison. In the tables showing expected and mutant outputs, where a number is not specified (region or device category), '0' should be assumed.

5.1 Mutant #1

This mutant deals with the average being calculated. Instead of dividing, one is subtracted from the other. The change in pseudocode is:

```
...
list[region][category] / households[region]
...

...
list[region][category] - households[region]
...
```

The level of this test is a unit, mutant test. It is a unit test because only 1 unit is tested. The test technique for this test is comparing expected outputs with the mutant outputs.

This is a good mutation that should be included because there is a chance that some results would be the same. For example 4/2 and 4-2 both are equal to 2.

Test file	Expected output	Mutant output
Section 6.4	AUK: 1 Smartware	AUK: 0 Smartware
	BOP: 1 Smart Applicances, 0.5 Smartware	BOP: 0 Smart Appliances, -1 Smartware
Section 6.5	AUK: 0.5 Smartware, 1.5 Smart Lighting	AUK: -1 Smartware, 1 Smartware
	BOP: 1 Smart Appliances, 0.5 Smartware	BOP: 0 Smart Appliances, -1 Smartware

Table 5.1: Expected and mutant outputs for Mutant 1

The mutant is caught using both datasets. This is shown by all values from the mutant program being different from the expected output.

Mutant score = Mutants caught/total tests = 2/2 = 100%

5.2 Mutant #2

This mutant also deals with the average being calculated. Instead of using dividing by the total devices in the correct region, it always divides by the first region (index 0). The change in pseudocode is:

```
...
list[region][category] / households[region]
...

...
list[region][category] / households[0]
...
```

The level of this test, is a unit, mutant test. It is a unit test because only 1 unit is tested. The test technique for this test is comparing expected outputs with the mutant outputs.

This is a good mutation that should be included because it is capable and can have a high likelihood of producing the correct output. If two regions have the same number of devices, then the mutant would not be caught. It is likely that with one dataset, that some regions statistics will be unaffected, while other regions will have different values.

Test file	Expected output	Mutant output
Section 6.4	AUK: 1 Smartware	AUK: 1 Smartware
	BOP: 1 Smart Applicances, 0.5 Smartware	BOP: 2 Smart Appliances, 1 Smartware
Section 6.5	AUK: 0.5 Smartware, 1.5 Smart Lighting	AUK: 0.5 Smartware, 1.5 Smart Lighting
	BOP: 1 Smart Appliances, 0.5 Smartware	BOP: 1 Smart Appliances, 0.5 Smartware

Table 5.2: Expected and mutant outputs for Mutant #2

The mutant is caught using the first dataset (Section 6.4). This is shown with the BOP region having different values. But, the mutant is not caught using the second dataset. This is because the mutant output is the same as the expected output.

Mutant score = Mutants caught/total tests = 1/2 = 50%

5.3 Mutant #3

This mutation deals with calculating the average. Instead of looking up two different indexes in the 2D array, the same index is looked up twice. The change in pseudocode is:

```
...
list[region][category] / households[region]
...

...
list[category][category] / households[region]
...
```

The level of this test is a unit, mutant test. It is a unit test because only 1 unit is tested. The test technique for this test is comparing expected outputs with the mutant outputs.

This is a good mutant test that should be included because there is a chance that some of the results are the same.

The category index was chosen to be repeated for the mutant test rather than the region index due to list indexing issues. As the number of regions is greater than the number of categories, the code will fail due to an out of bounds error. This is not a good example of a mutation test because it will be caught 100% of the time due to an error being thrown rather than the wrong values being displayed.

Test file	**Expected output**	**Mutant output**
Section 6.4	AUK: 1 Smartware BOP: 1 Smart Applicances, 0.5 Smartware	All regions: 2 Smart Appliances
Section 6.5	AUK: 0.5 Smartware, 1.5 Smart Lighting BOP: 1 Smart Appliances, 0.5 Smartware	All regions: 2 Smart Appliances

Table 5.3: Expected and mutant outputs for Mutant #3

The mutant is caught using both datasets. This is shown by all values from the mutant program being different from the expected output.

Mutant score = Mutants caught/total tests = 2/2 = 100%

5.4 Mutant #4

This mutation deals with checking the value before calculating the average. The pseudocode for this change is:

```
...
if (households in region == 0){
...

...
if (households in region >= 0){
...
```

The level of this test is a unit, mutant test. It is a unit because only 1 unit is tested. The test technique used fort his test is comparing expected outputs with the mutant outputs.

This is a good mutant test because it changes the branch coverage, completely ignoring the branches from the 'else' branch.

Test file	Expected output	Mutant output
Section 6.4	AUK: 1 Smartware	All regions:
	BOP: 1 Smart Applicances, 0.5 Smartware	0 for all
Section 6.5	AUK: 0.5 Smartware, 1.5 Smart Lighting	All regions:
	BOP: 1 Smart Appliances, 0.5 Smartware	0 for all

Table 5.4: Expected and mutant outputs for Mutant #4

The mutant is caught using both datasets. This is shown by some values from the mutant program being different from the expected output.

Mutant score = Mutants caught/total tests = 2/2 = 100%

6 Appendix

6.1 Testing Dataset 1

This dataset provides the minimum requirements to ensure graph visualisation (Section 3.7) meets all the requirements.

```
EWR-1234,01/01/2023,Encost Router 360,Router,WKO-1234,-,Yes,Yes
ELB-4567,01/01/2023,Encost Smart Bulb B22 (white),Light bulb,WKO-1234,EWR-1234,No,Yes
EK-9876,01/01/2023,Encost Smart Jug,Kettle,WKO-1234,EWR-1234,Yes,No
EHC-2468,01/01/2023,Encost Smart Hub 2.0,Hub/Controller,WKO-1234,Yes,Yes
ESW-5555,01/01/2023,Encost Smart Washer,Washing,WKO-1234,No,Yes
EWR-2345,01/01/2023,Encost Router Plus,Router,AUK-2345,-,Yes,Yes
ESW-3333,01/01/2023,Encost Smart Washer,Washing,AUK-2345,Yes,Yes
```

6.2 Testing Dataset 2

The dataset provides the minimum requirements to meet full branch coverage for testing the method in Section 4.1.

```
EWR-1357,02/02/2023,Encost Router 360,Router,BOP-7654,-,Yes,Yes
ELB-3579,02/02/2023,Encost Novelty Light (giraffe),Other Lighting,BOP-7654,EWR-1357,No,Y
```

6.3 Testing Dataset 3

The dataset provides more requirements than Section 6.2 to ensure correct average calculation and multiple regions work.

```
EWR-1357,02/02/2023,Encost Router 360,Router,BOP-7654,-,Yes,Yes
ELB-3579,02/02/2023,Encost Novelty Light (Giraffe),Other Lighting,BOP-7654,EWR-1357,No,Y
EWR-1234,02/02/2023,Encost Router 360,Router,BOP-2222,-,Yes,Yes
EWR-7890,02/02/2023,Encost Router 360,Router,AUK-3333,-,Yes,Yes
```

6.4 Testing Dataset 4

```
EWR-1357,03/03/2023,Encost Router 360,Router,BOP-7654,-,Yes,Yes
EK-9876,03/03/2023,Encost Smart Jug,Kettle,BOP-7654,EWR-1357,Yes,No
ESW-5555,03/03/2023,Encost Smart Washer,Washing,BOP-7654,EW-1357,Yes,Yes
EWR-1234,03/03/2023,Encost Router 360,Router,BOP-2222,-,Yes,Yes
EK-6789,03/03/2023,Encost Smart Jug,Kettle,BOP-2222,EWR-1234,Yes,No
EWR-7890,03/03/2023,Encost Router 360,Router,AUK-3333,-,Yes,Yes
ESW-4444,03/03/2023,Encost Smart Washer,Washing,AUK-3333,EWR-7890,Yes,Yes
```

6.5 Testing Dataset 5

```
EWR-1357,03/03/2023,Encost Router 360,Router,BOP-7654,-,Yes,Yes
EK-9876,03/03/2023,Encost Smart Jug,Kettle,BOP-7654,EWR-1357,Yes,No
ESW-5555,03/03/2023,Encost Smart Washer,Washing,BOP-7654,EW-1357,Yes,Yes
EWR-1234,03/03/2023,Encost Router 360,Router,BOP-2222,-,Yes,Yes
EK-6789,03/03/2023,Encost Smart Jug,Kettle,BOP-2222,EWR-1234,Yes,No
EWR-7890,03/03/2023,Encost Router 360,Router,AUK-3333,-,Yes,Yes
ESW-4444,03/03/2023,Encost Smart Washer,Washing,AUK-3333,EWR-7890,Yes,Yes
EWR-2222,04/04/2023,Encost Router 360,Router,AUK-3456,-,Yes,Yes
ELB-4567,04/04/2023,Encost Smart Bulb B22 (white),Light bulb,AUK-3456,EWR-2222,No,Yes
ELB-4568,04/04/2023,Encost Smart Bulb E26 (white),Light bulb,AUK-3456,EWR-2222,No,Yes
ELB-4569,04/04/2023,Encost Smart Bulb E26 (white),Light bulb,AUK-3456,EWR-2222,No,Yes
```

6.6 Invalid Datasets

In the Functional Test Suite, there are multiple invalid datasets included. These are for the purpose of testing issues with reading in data from the csv (incorrect number of arguments, incorrect data types, duplicate data).