FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.0

Prepared by: Student 1
SoftFlux Engineer

SoftFlux

May 6, 2023

Contents

Revision History

Name	Date	Reason for Changes	Version

1 Introduction/Purpose

1.1 Purpose

This document is Software Design Specification for the software Encost Smart Graph Project (ESGP) for the company Encost, which has the purpose of optimising the use and connectivity between devices. This software allows the user to view the devices and their connections, provides graph visualisation of Encost's devices, as well as providing summary statistics about each device. The purpose of this Software Test plan is to specify how the software's implementation of the accepted design specification will be checked for its completeness, robustness, and fulfilment of the software's purpose.

1.2 Document Conventions

- ESGP - Encost Smart Graph Project
- SDS - Software Design Specification
- SRS - Software Requirements Specification

1.3 Intended Audience and Reading Suggestions

This document is for developers and testers to read, and implement, to ensure catch errors in the system and ensure a robust system is developed.

- Developers - Use this document to ensure function names fit the testing suite, and to know the assumptions testers will work on when developing an implementation.
- Tester - Use this document to implement the testing, and evaluate the software.

1.4 Project Scope

The test plan will follow the accepted Software Design Specification #3. Softflux is only responsible for the base version of the software. This test plan only accounts for an English version, with the following functionalities:

- User login and categorisation
- User input and output
- Graph visualising
- Data processing and statistics
- File parsing

2 Specialized Requirements Specification

2.1 Display all categories in device summary

When generating device statistics, all categories and types will be printed even if there are 0 of that category/type. This has been confirmed by the client.

2.2 Login process

The accepted SDS has incorrectly implemented the login process, specified in the SRS. The logic will be assumed to follow the specified SRS logic. The username and password will be read and validated simultaneously, rather than individually.

2.3 Select Feature function

It will be assumed there is an internal function called 'determineFeatureSelected' which takes a string input and returns a string for the type of feature that is selected. It will return 'summary', 'graph', 'load', or 'invalid' based on the validity of the selection. This method will be used internally by 'selectFeature'. This assumption has been confirmed by the client.

2.4 GetUserVersion internal function

It will be assumed there is an internal function called 'determineVersionType' which takes a string input and returns a string for the type of user type that is selected. It will return 'unverified-enocst', 'community' or 'invalid' based on the validity of the selection. This method will be used internally by 'getUserVersion'. This assumption has been confirmed by the client.

2.5 Device Category

The specified SDS does not include an implementation for the Device object's category or region, which is a required feature by the system. It will be assumed there is a function that returns these values called 'getDeviceCategory()' and 'getHouseholdRegion()', returning a string value for the category/region.

2.6 Empty account details

It is assumed that a password could be empty as hash functions will still return a hash for an empty string. So the system will not reject an 'empty' password. However, it is assumed that a username cannot be empty and is therefore invalid.

2.7 Case sensitivity of username/password pairings

It will be assumed usernames are NOT case-sensitive, and passwords will be case-sensitive, as per common convention.

2.8 Valid username characters

The provided username/password pairings will be assumed to only use characters found on a US English keyboard. Provided usernames in the will be assumed to not have spaces, and only include characters a-z, 0-9, ._- as per common convention to support the largest range of keyboards. It will be assumed that any of the 10 username/password pairings provided will be valid according to these restrictions.

2.9 Return type of verifyUsername and verifyPassword

It will be assumed that the verifyUsername and verifyPassword functions will return a boolean to inform the program if a verification step was completed successfully.

2.10 Devices without a router connection

If a device does NOT have a router connection, and no other devices connect to it, it will be assumed to be an isolated device which is still valid. Devices do not need to be all connected together. i.e. Non-router devices can also validly have an empty 'routerConnection'

2.11 Invalid category/region

It will be assumed if the Device class cannot determine the category/region from the householdId or deviceType/deviceName/deviceId a null value will be returned by the 'getDeviceRegion()' and 'getDeviceCategory()' functions.

2.12 Empty routerConnection

It will be assumed if the routerConnection of a device is unspecified in the dataset file it will be entered as 'null' into the Device class constructor.

2.13 Method of determining category

It will be assumed that internally the system uses the 'deviceID' to determine the Device category, as this is consistent with how the Region is calculated from the Household ID.

2.14 Device IDs

It will be assumed the device ID device type initials are in the form of the first letters in the device type, prepended by an 'E'. For example, 'Kettle'='EK', 'Hub/Controller'='EHC', 'Light Bulb'='ELB'. It will also be assumed the number (e.g. '-1234') combined with the initials of the device type will be unique to all other devices, for example, there will never be two devices of 'EWR-1234', but there can be another of 'EHC-1234'.

2.15 Hashing method

The passwords will not be 'encrypted', they will be 'hashed' according to the SDS, but the hashing algorithm was unspecified. It will be assumed the hashing function used to hash the passwords with be an SHA-256 hashing algorithm.

2.16 User/Password pairs

The following tests will reference 10 example username/plain text password pairs used to test ESGP Account Login, these will obviously not be used by the client but will be assumed to be used during testing. These will be entered in a file located at the constant 'USER_DETAILS_FILEPATH' location.

Username	Password
'user1'	'valid1'
'user2'	'valid2'
'user3'	'valid3'
'user4'	'valid4'
'user5'	'valid5'
'user6'	'valid6'
'user7'	'valid7'
'user8'	'valid-8+&!.'
'user_9.-'	'valid9'
'user10'	''

Note: In the file storing these pairs the passwords will be hashed with SHA-256, the specified above is plain text version

3 Black-box Testing

JUnit is a popular automated testing platform, which leverages testing individual function calls in isolation to test the overall reliability, completeness, and correctness of a program. JUnit is the recommended tool for testing the following tests in an automated fashion.

3.1 Categorising Users

3.1.1 Description

Users should be able to indicate whether they are a community user, or an encost user. The user type selection will be represented by the userIsVerified boolean, after the user finishes the selection stage. A function exists called 'getUserVersion()' which takes a single character as input, which can be '1' for Community User, and '2' for Encost User. The 'getUserVersion()' function will return a string of 'encost-unverified', 'community', or 'invalid', which will then determine what steps the program takes next. Any character other than '1' or '2' will cause the function to return 'invalid'.

3.1.2 Functional Requirements Tested

SRS 4.1 REQ-2 The system should store the user-type that the user has selected (community or encost-unverified).

3.1.3 Test Type

- Level of test: Black-box testing, unit testing
- Test Technique: Expected inputs, the equivalence class of invalid inputs, boundary and edge cases

3.1.4 Test Cases

Input	Expected Output
'1'	'community'
'2'	'encost-unverified'
'3'	'invalid'
"	'invalid'
'a'	'invalid'

3.2 ESGP Account Login

3.2.1 Description

The ESGP Account Login feature is implemented using the UserVerifier class and will allow the user to enter a username and password to be compared against 10 username/hashed-password pairings found in the file 'USER_DETAILS_FILEPATH. The UserVerifier class has two functions, verifyUsername and verifyPassword which both take a string as a parameter representing the user's username/password respectively. Both verifyUsername and verifyPassword will return a boolean indicating if the verification was successful. The verifyUsername will return false if the username is not part of a valid username/-password pair. The verifyPassword function will only return 'true' if the password when hashed matches the associated password hash of the username(Note: verifyUsername must be run before with a valid username).

Username/Password pairings are specified in 2.16.

3.2.2 Functional Requirements Tested

SRS 4.2 REQ-2 Once the username and password have been entered, the system should check that the inputs are valid.

SRS 4.2 REQ-5 Ten username and password pairs will be provided. The passwords should be encrypted before being stored in the application (see 2.16)

3.2.3 Test Type

- Test Level: Black-box testing, integration test between local storage (Username/-password pairs file) and account login
- Test Techniques: Equivalence classes of invalid usernames and invalid passwords, edge cases, and boundary

3.2.4 Test cases

Username	Password	Expected Output
'user1'	'valid1'	True
'user2'	'valid2'	True
'user3'	'valid3'	True
'user4'	'valid4'	True
'user5'	'valid5'	True
'user6'	'valid6'	True
'user7'	'valid7'	True
'user8'	'valid-8+&!.'	True
'user_9.-'	'valid9'	True
'user10'	''	True
'USER1'	'valid1'	True
'user1'	'invalid'	False
'user1'	'valid2'	False
'user1'	'VALID1'	False
'invalid'	-	False
''	-	False

3.3 ESGP Feature Options

3.3.1 Description

The 'determineFeatureSelected()' (see 2.4) handles the selection between the individual features by accepting a string and a boolean. The string is the user's selection of '1' for graph visualisation, '2' for loading a custom dataset, '3' for summary statistics, and any other input is invalid; the function will return 'graph', 'load', 'summary', or 'invalid' based on the selection. The boolean input specifies if the user is verified, and should have access to options '2' and '3', if 'true' then return the corresponding feature name, otherwise return 'invalid'. Option '1' will always return 'graph'.

3.3.2 Functional Requirements Tested

SRS 4.3 REQ-1 The ESGP Feature Options prompt should provide the user with a selection of features that they can prick 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, (c) viewing summary statistics.

3.3.3 Test Type

- Test Level; Black-box testing, unit test
- Test Techniques: Decision table, expected input, equivalence class of invalid inputs, and boundary and edge cases

3.3.4 Test Cases

Input - selection	Input - isVerified	Expected output
’1’	True	’graph’
’1’	False	’graph’
’2’	True	’load’
’2’	False	’invalid’
’3’	True	’summary’
’3’	False	’invalid’
’a’	True	’invalid’
’a’	False	’invalid’
”	True	’invalid’
”	False	’invalid’

3.4 Loading Encost Dataset - FileParser

3.4.1 Description

The Loading Encost Dataset feature is handled by the ’loadFile’ function. However, the ’loadFile’ function uses the internal FileParser object’s parseFile function, which takes in a single parameter ’filePath’ and returns an array of Device objects. If the file path to the ENCOST_DATASET_FILEPATH is passed, the parseFile function should return all the devices inside. If an invalid file path is given, or the dataset is invalid, ’null’ will be returned.

3.4.2 Functional Requirements Tested

SRS 4.4 REQ-2 The system should know the default location of the Encost Smart Homes Dataset

SRS 4.4 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

- Test Level; Black-box testing, integration between encost dataset (Local storage), Device class, and fileparser.
- Test Techniques: Equivalence class of all valid devices and all valid household IDs, and edge case.

3.4.4 Test Cases

Input	Excepted Output
ENCOST_DATASET_FILEPATH	a non-empty array of Devices - 4 of the devices will match the first 4 entries in Table 3.1 (Exclude 'ED-2468')
ENCOST_DATASET_FILEPATH	100 unique HouseholdIDs exist over all devices - one of which will have ID 'WKO-1234'
'/fake/path.txt'	null

3.5 Loading Encost Dataset - loadFile()

3.5.1 Description

The loadFile() function is present in the ConsoleApp class and combines the ENCOST_-DATASET_FILEPATH with the FileParser component. The loadFile function takes no inputs but returns an array of Device objects associated with the Encost Dataset. If the function failed to parse the file it would return 'null', however, the encost dataset should not return null as the system is designed to accommodate it.

The following is an example of devices from the Encost dataset that can be used to test the validity of the result.

3.5.2 Functional Requirements Tested

SRS 4.4 REQ-3 The system should be able to read the Encost Smart Homes Dataset line by line and extract the relevant device information.

3.5.3 Test Type

- Test Level: Black-box testing, integration test between FileParser, Device class, and ConsoleApp class (contains loadFile() function), and Encost dataset (Local storage).
- Test Techniques: Expected input and output, the Equivalence class of all device entries, and the Equivalence class of all valid household IDs

Device ID	Date Connected	Device name	Device type	Household ID	Router Connection	Sends	Receives
EWR-1234	01/04/22	Encost Router 360	Router	WKO-1234	-	Yes	Yes
ELB-4567	01/04/22	Encost Smart Bulb B22 (multi colour)	Light bulb	WKO-1234	EWR-1234	No	Yes
EK-9876	07/05/22	Encost Smart Jug	Kettle	WKO-1234	EWR-1234	No	Yes
EHC-2468	01/04/22	Encost Smart Hub 2.0	Hub/ Controller	WKO-1234	EWR-1234	Yes	Yes
ED-2468	01/04/22	Encost Dishwasher Pro	Dishwasher	WKO-1234	EWR-1234	No	Yes

Table 3.1: Data in Encost Smart Homes Dataset + Example whiteware entry

3.5.4 Test Cases

Function call	Excepted Output
loadFile()	Non-empty array of Devices - Four devices match Table 3.1 (Exclude entry 'ED-2468')
loadFile()	100 unique HouseholdIDs exist over all devices - One of which has the ID 'WKO-1234'

3.6 Categorising Smart Home Devices - Categorising

3.6.1 Description

The Categorising Smart Home Devices feature is handled internally by the Device class constructor, which takes eight parameters and allows the user to hold all of a device's information and categorise the user by it. Based on the deviceId (see 2.13), the system will determine the device's category which can be retrieved with 'getDeviceCategory()' called on the Device object.

3.6.2 Functional Requirements Tested

SRS 4.6 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).

SRS 4.6 REQ-2 The system should create an object for each device. This object should hold all of the information for that device.

3.6.3 Test Type

- Test Level: Black-box testing, unit test
- Test Techniques: Equivalence classes of invalid device types and device numbers, boundary, expected inputs
- Functions used: getDeviceCategory

3.6.4 Test Cases

Input DeviceID	getDeviceCategory() Output
"EWR-1234"	'Encost Wifi Routers'
"EWE-1234"	'Encost Wifi Routers'
"ELB-4567"	'Encost Smart Lighting'
"ESL-4567"	'Encost Smart Lighting'
"EOL-4567"	'Encost Smart Lighting'
"EK-9876"	'Encost Smart Appliances'
"ET-9876"	'Encost Smart Appliances'
"ECM-9876"	'Encost Smart Appliances'
"EHC-2468"	'Encost Hubs/Controllers'
"ED-1246"	'Encost Smart Whiteware'
"EWMD-1246"	'Encost Smart Whiteware'
"ERF-1246"	'Encost Smart Whiteware'
"IV-0000"	null
""	null

3.7 Categorising Smart Home Devices - Object data

3.7.1 Description

The Categorising Smart Home Devices feature is handled internally by the Device class constructor, which takes eight parameters. The Device class holds all of a device's information. Device information can be accessed through get functions of the same name; all 'get' functions return a string, except for DateConnected (Date), DeviceType (DeviceType enum), and isSender/isReceiver (Boolean).

3.7.2 Functional Requirements Tested

SRS 4.6 REQ-2 The system should create an object for each device. This object should hold all of the information for that device.

3.7.3 Test Type

- Test Level: Black-box testing, unit test
- Test Techniques: Equivalence classes of all valid device ids, date connected, name, type, HouseholdId, RouterConnection, and sending/receiving. Expected inputs

3.7.4 Test Cases

Each table specifies a different input of a specific part of the Device constructor that we are testing, parameters not specified in each test can be arbitrarily chosen from Table 3.1. To avoid duplicate descriptions, and restating duplicate information each of the following single case tests will be stated here.

Input DeviceID	getDeviceId() Output
”EWR-1234”	”EWR-1234”

Input DateConnected	getDateConnected() Output
new Date(”1/1/22”)	equivalent to new Date(”1/1/22”)

Input DeviceName	getDeviceName() Output
”Encost Router 360”	”Encost Router 360”

Input DeviceType	getDeviceType() Output
DeviceType.Router	DeviceType.Router

Input HouseholdID	getHouseholdId() Output
”WKO-1234”	”WKO-1234”

Input RouterConnection	getRouterConnection() Output
”EWR-1234”	”EWR-1234”

Input Sends	isSender() Output
False	False

Input Receives	isReceiver() Output
True	True

3.8 Building a Graph Data Type - getDevices()

3.8.1 Description

The Graph Data Type is handled by the DeviceGraph class, which holds an array of devices (Nodes) with an adjacencyMatrix object holding the connections (Edges) between devices. The GraphData constructor has a single parameter which is an array of Device objects, the constructor automatically populates the adjacencyMatrix object and allows the user to run calculations on devices based on their directed connections with one another. All devices (nodes) can be retrieved with the 'getDevices' function, which is a member function of the DeviceGraph class, takes no parameters, and returns a Device array.

3.8.2 Functional Requirements Tested

SRS 4.7 REQ-1 Each Encost Smart Device object should be stored in the graph data structure. The objects should be the nodes in the graph. The connection between objects should be the edges.

SRS 4.7 REQ-2 All unique data points should be included in the graph.

SRS 4.7 REQ-3 All households should be represented in the graph.

3.8.3 Test Type

- Test Level; Black-box testing, Bottom up integration test of DeviceGraph and Device class
- Test Techniques: Equivalence class of all valid devices, and householdId, boundary case

3.8.4 Test Cases

Devices in DeviceGraph	getDevices() Output
See Table 3.1	Array Device objects created from Table 3.1
See Table 3.1	All devices have householdID 'WKO-1234'
0 devices	empty array

3.9 Building a Graph Data Type - Neighbours of nodes

3.9.1 Description

The Graph Data Type is handled by the DeviceGraph class, which holds an array of devices (Nodes) with an adjacencyMatrix object holding the connections (Edges) between devices. The GraphData constructor has a single parameter which is an array of Device objects, the constructor automatically populates the adjacencyMatrix object and allows the user to run calculations on devices based on their directed connections with one another. The 'getNeighbour' function takes a string deviceID as a parameter and returns an array of Device objects which can receive commands from the device (or null if device not found). To be the neighbour of a device, the device needs to have the ability to 'send' commands, and the neighbouring devices need to have the ability to 'receive' commands (isSender() and isReceiver() values).

3.9.2 Functional Requirements Tested

SRS 4.7 REQ-1 Each Encost Smart Device object should be stored in the graph data structure. The objects should be the nodes in the graph. The connection between objects should be the edges.

3.9.3 Test Type

- Test Level: Black-box testing, Integration test between DeviceGraph class and Device class
- Test Techniques: The equivalence class of all devices that sends to neighbours, and receives from neighbours. Edge and boundary case

3.9.4 Test Cases

Devices in DeviceGraph	getNeighbour Input	Expected Output
See Table 3.1	"EWR-1234"	array with 4 Device objects
See Table 3.1	"ELB-4567"	empty Device array
See Table 3.1	"EHC-2468"	Device array with a single Device object
1 Device of with deviceID 'EWR-1234'	"EWR-1234"	empty array
0 Devices	"IV-000"	null

3.10 Calculating Device Distribution

3.10.1 Description

The Calculating Device Distribution feature is handled by DataSummary class with the 'calculateDeviceDistribution' function. The DataSummary constructor takes a DeviceGraph as a parameter, which is the DeviceGraph object on which calculations are done. The calculateDeviceDistribution function does not take any parameters and returns a string with all the summary information.

3.10.2 Functional Requirements Tested

SRS 4.7 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. F

SRS 4.7 REQ-2 For each device category, the system should also calculate the number of devices that exist for each device type.

SRS 4.7 REQ-3 The system should output these figures to the console in a clear and concise manner.

3.10.3 Test Type

- Test Level: Blackbox testing, integration test between the device class, graph data type, and summary information.
- Test Techniques: Equivalence class, Expected input, boundary and edge case

3.10.4 Test Cases

Example full device information can be found in Table 3.1.

Devices in DeviceGraph	Expected Output

0 devices	`Category Encost Wifi Routers device count: 0`
	`  - Type ’Router’ device count: 0`
	`  - Type ’Extender’ device count: 0`
	`Category Encost Hub/Controller device count: 0`
	`  - Type ’Hub/Controller’ device count: 0`
	`Category Encost Smart Lighting device count: 0`
	`  - Type ’Light Bulb’ device count: 0`
	`  - Type ’Strip Lighting’ device count: 0`
	`  - Type ’Other Lighting’ device count: 0`
	`Category Encost Smart Appliances device count: 0`
	`  - Type ’Kettle’ device count: 0`
	`  - Type ’Toaster’ device count: 0`
	`  - Type ’Coffee Maker’ device count: 0`
	`Category Encost Smart Whiteware device count: 0`
	`  - Type ’Washing Machine/Dryer’ device count: 0`
	`  - Type ’Refrigerator/Freezer’ device count: 0`
	`  - Type ’Dishwasher’ device count: 0`

1 of each of the following device types: Router, Extender	Category Encost Wifi Routers device count: 2 - Type ’Router’ device count: 1 - Type ’Extender’ device count: 1 Category Encost Hub/Controller device count: 0 - Type ’Hub/Controller’ device count: 0 Category Encost Smart Lighting device count: 0 - Type ’Light Bulb’ device count: 0 - Type ’Strip Lighting’ device count: 0 - Type ’Other Lighting’ device count: 0 Category Encost Smart Appliances device count: 0 - Type ’Kettle’ device count: 0 - Type ’Toaster’ device count: 0 - Type ’Coffee Marker’ device count: 0 Category Encost Smart Whiteware device count: 0 - Type ’Washing Machine/Dryer’ device count: 0 - Type ’Refrigerator/Freezer’ device count: 0 - Type ’Dishwasher’ device count: 0
1 Device of DeviceType.HubController	Category Encost Wifi Routers device count: 0 - Type ’Router’ device count: 0 - Type ’Extender’ device count: 0 Category Encost Hub/Controller device count: 1 - Type ’Hub/Controller’ device count: 1 Category Encost Smart Lighting device count: 0 - Type ’Light Bulb’ device count: 0 - Type ’Strip Lighting’ device count: 0 - Type ’Other Lighting’ device count: 0 Category Encost Smart Appliances device count: 0 - Type ’Kettle’ device count: 0 - Type ’Toaster’ device count: 0 - Type ’Coffee Marker’ device count: 0 Category Encost Smart Whiteware device count: 0 - Type ’Washing Machine/Dryer’ device count: 0 - Type ’Refrigerator/Freezer’ device count: 0 - Type ’Dishwasher’ device count: 0
1 of each of the following device types: LightBulb, StripLighting, OtherLighting	Category Encost Wifi Routers device count: 0 - Type ’Router’ device count: 0 - Type ’Extender’ device count: 0 Category Encost Hub/Controller device count: 0 - Type ’Hub/Controller’ device count: 0 Category Encost Smart Lighting device count: 3 - Type ’Light Bulb’ device count: 1 - Type ’Strip Lighting’ device count: 1 - Type ’Other Lighting’ device count: 1 Category Encost Smart Appliances device count: 0 - Type ’Kettle’ device count: 0 - Type ’Toaster’ device count: 0 - Type ’Coffee Marker’ device count: 0 Category Encost Smart Whiteware device count: 0 - Type ’Washing Machine/Dryer’ device count: 0 - Type ’Refrigerator/Freezer’ device count: 0 - Type ’Dishwasher’ device count: 0

1 of each of the following device types: Kettle, Toaster, CoffeMaker	`Category Encost Wifi Routers device count: 0`
	`  - Type ’Router’ device count: 0`
	`  - Type ’Extender’ device count: 0`
	`Category Encost Hub/Controller device count: 0`
	`  - Type ’Hub/Controller’ device count: 0`
	`Category Encost Smart Lighting device count: 0`
	`  - Type ’Light Bulb’ device count: 0`
	`  - Type ’Strip Lighting’ device count: 0`
	`  - Type ’Other Lighting’ device count: 0`
	`Category Encost Smart Appliances device count: 3`
	`  - Type ’Kettle’ device count: 1`
	`  - Type ’Toaster’ device count: 1`
	`  - Type ’Coffee Marker’ device count: 1`
	`Category Encost Smart Whiteware device count: 0`
	`  - Type ’Washing Machine/Dryer’ device count: 0`
	`  - Type ’Refrigerator/Freezer’ device count: 0`
	`  - Type ’Dishwasher’ device count: 0`

1 of each of the following device types: WashingMachineDryer, RefrigeratorFreezer, and Dishwasher	`Category Encost Wifi Routers device count: 0`
	`  - Type ’Router’ device count: 0`
	`  - Type ’Extender’ device count: 0`
	`Category Encost Hub/Controller device count: 0`
	`  - Type ’Hub/Controller’ device count: 0`
	`Category Encost Smart Lighting device count: 0`
	`  - Type ’Light Bulb’ device count: 0`
	`  - Type ’Strip Lighting’ device count: 0`
	`  - Type ’Other Lighting’ device count: 0`
	`Category Encost Smart Appliances device count: 0`
	`  - Type ’Kettle’ device count: 0`
	`  - Type ’Toaster’ device count: 0`
	`  - Type ’Coffee Marker’ device count: 0`
	`Category Encost Smart Whiteware device count: 3`
	`  - Type ’Washing Machine/Dryer’ device count: 1`
	`  - Type ’Refrigerator/Freezer’ device count: 1`
	`  - Type ’Dishwasher’ device count: 1`

1 x Device of DeviceType Router, 2x Device of DeviceType Extender	`Category Encost Wifi Routers device count: 3`
	`  - Type ’Router’ device count: 1`
	`  - Type ’Extender’ device count: 2`
	`Category Encost Hub/Controller device count: 0`
	`  - Type ’Hub/Controller’ device count: 0`
	`Category Encost Smart Lighting device count: 0`
	`  - Type ’Light Bulb’ device count: 0`
	`  - Type ’Strip Lighting’ device count: 0`
	`  - Type ’Other Lighting’ device count: 0`
	`Category Encost Smart Appliances device count: 0`
	`  - Type ’Kettle’ device count: 0`
	`  - Type ’Toaster’ device count: 0`
	`  - Type ’Coffee Maker’ device count: 0`
	`Category Encost Smart Whiteware device count: 0`
	`  - Type ’Washing Machine/Dryer’ device count: 0`
	`  - Type ’Refrigerator/Freezer’ device count: 0`
	`  - Type ’Dishwasher’ device count: 0`

4 White-box testing

4.1 calculateDeviceDistribution Pseudocode

```
string calculateDeviceDistribution(){
    IF this.deviceGraph is null
        return "Device Distribution Summary Failed: Graph is uninitialised"
    ENDFOR
    SET devices = this.deviceGraph.getDevices()
    SET validCategories = Hashmap of categories -> Hashmap of types->0
    FOR EACH device in the devices
        IF the device’s category is null
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE IF the device’s type is NOT in validCategories[category]
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE
            INCREMENT type’s count by 1
        ENDIF
    ENDFOR
    FOR EACH category in the hashmap
        SET categoryDeviceCount = 0
        FOR EACH type associated with the category
            ADD type’s count to categoryDeviceCount
        ENDFOR
        ADD category’s name and categoryDeviceCount to outputString
        FOR EACH type associated with the category in the hashmap
            ADD type name and type’s count to outputString
        ENDFOR
    ENDFOR
    RETURN outputString
}
```

4.2 Branch Coverage Testing

4.2.1 Description

The Calculating Device Distribution feature is handled by DataSummary class with the calculateDeviceDistribution() function, which prints out the number of devices for each device category and then the breakdown by device type. The DataSummary constructor takes a DeviceGraph as a parameter to run all summary functions on. The calculateDeviceDistribution function does not take any parameters and returns a string with the summary information, it will return an error string if the device graph is null - if any devices have invalid statistics (e.g. category or type is invalid) they will be skipped and excluded from any statistics. The device ID is assumed to be used to determine the category (see 2.13), so DeviceType and device category can be null independent of one another.

4.2.2 Functional Requirements Tested

SRS 4.9 REQ-1, SRS 4.9 REQ-2, SRS 4.9 REQ-3

4.2.3 Test Type

- Test Level; Whitebox testing, Integration test between DeviceGraph and Summary statistics

- Test Techniques: Equivalence class of all device types, edge case

4.2.4 Test Cases

The following test is shared with 3.10.

DeviceGraph input	Expected Output

1x Device of DeviceType.Router, 2x Device of DeviceType.Extender	`Category Encost Wifi Routers device count: 3`
	`  - Type 'Router' device count: 1`
	`  - Type 'Extender' device count: 2`
	`Category Encost Hub/Controller device count: 0`
	`  - Type 'Hub/Controller' device count: 0`
	`Category Encost Smart Lighting device count: 0`
	`  - Type 'Light Bulb' device count: 0`
	`  - Type 'Strip Lighting' device count: 0`
	`  - Type 'Other Lighting' device count: 0`
	`Category Encost Smart Appliances device count: 0`
	`  - Type 'Kettle' device count: 0`
	`  - Type 'Toaster' device count: 0`
	`  - Type 'Coffee Marker' device count: 0`
	`Category Encost Smart Whiteware device count: 0`
	`  - Type 'Washing Machine/Dryer' device count: 0`
	`  - Type 'Refrigerator/Freezer' device count: 0`
	`  - Type 'Dishwasher' device count: 0`

null deviceGraph	`"Device Distribution Summary Failed: Graph is uninitialised"`
1x DeviceType.Router with no deviceID	`"Device Distribution Summary Failed: Some devices have invalid data"`
1x Device with a null DeviceType	`"Device Distribution Summary Failed: Some devices have invalid data"`

4.2.5 Branch coverage

There are 11 branches in 4.1:

- Four For loops - Two branches for each for-all loop (1 entry, and 1 exit branch)
- Four If statements - One branch for each (Else statement included)

All categories, and all types, will be represented in the output string so a majority of the code can be tested with a single test, however, branches that return immediately with errors need individual tests to enter each individual branch. The first test covers 9/12 branches printing to console both zero and non-zero entries. The second test covers

1/12 branches catching the first error handling IF statement and returns immediately. The second test covers 2/12 branches, passing the first FOR loop, and returning early after error checking the first IF statement. The second test covers 2/12 branches, passing the first FOR loop, and returning early after error checking the second IF statement. In total, 100% branch coverage is achieved. For robustness, branch testing seems to be not the best approach as it can largely be covered by a single test (except for the null handling). Code path coverage could be a better method to confirm the robustness of this code, by tracking the paths the logic can take through the system.

5 Mutation Testing

5.1 Mutant #1 - Device type count starts at 1

5.1.1 Description

Small syntatical change to start the count of all types at 1, will not be caught unless the file RETURN statement is reached. Sets the count to 1 for the initial device type counts.

5.1.2 Pseudo-code

```
string calculateDeviceDistribution(){
    IF this.deviceGraph is null
        return "Device Distribution Summary Failed: Graph is uninitialised"
    ENDFOR
    SET devices = this.deviceGraph.getDevices()
    SET validCategories = Hashmap of categories -> Hashmap of types->1 <--MOD
    FOR EACH device in the devices
        IF the device's category is null
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE IF the device's type is NOT in validCategories[category]
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE
            INCREMENT type's count by 1
        ENDIF
    ENDFOR
    FOR EACH category in the hashmap
        SET categoryDeviceCount = 0
        FOR EACH type associated with the category
            ADD type's count to categoryDeviceCount
        ENDFOR
        ADD category's name and categoryDeviceCount to outputString
        FOR EACH type associated with validCategory[category]
            ADD type name and type's count to outputString
        ENDFOR
    ENDFOR
    RETURN outputString
}
```

5.2 Mutant #2 - type's count is decremented

5.2.1 Description

Small syntatical change to remove 1 from the type's count, instead of adding.

5.2.2 Pseudo-code

```
string calculateDeviceDistribution(){
    IF this.deviceGraph is null
        return "Device Distribution Summary Failed: Graph is uninitialised"
    ENDFOR
    SET devices = this.deviceGraph.getDevices()
    SET validCategories = Hashmap of categories -> Hashmap of types->0
    FOR EACH device in the devices
        IF the device's category is null
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE IF the device's type is NOT in validCategories[category]
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE
            DECREMENT type's count by 1 <--MOD
        ENDIF
    ENDFOR
    FOR EACH category in the hashmap
        SET categoryDeviceCount = 0
        FOR EACH type associated with the category
            ADD type's count to categoryDeviceCount
        ENDFOR
        ADD category's name and categoryDeviceCount to outputString
        FOR EACH type associated with validCategory[category]
            ADD type name and type's count to outputString
        ENDFOR
    ENDFOR
    RETURN outputString
}
```

5.3 Mutant #3 - Add category count instead of type count to output

5.3.1 Description

Small syntactical change which makes the types print the same number of devices as its parent category. Can cause types to not add up.

5.3.2 Pseudo-code

```
string calculateDeviceDistribution(){
    IF this.deviceGraph is null
        return "Device Distribution Summary Failed: Graph is uninitialised"
    ENDFOR
    SET devices = this.deviceGraph.getDevices()
    SET validCategories = Hashmap of categories -> Hashmap of types->0
    FOR EACH device in the devices
        IF the device's category is null
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE IF the device's type is NOT in validCategories[category]
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE
            INCREMENT type's count by 1
        ENDIF
    ENDFOR
    FOR EACH category in validCategory
        SET categoryDeviceCount = 0
        FOR EACH type associated with the category
            ADD type's count to categoryDeviceCount
        ENDFOR
        ADD category's name and categoryDeviceCount to outputString
        FOR EACH type associated with validCategory[category]
            ADD type name and category's count to outputString  <--MOD
        ENDFOR
    ENDFOR
    RETURN outputString
}
```

5.4 Mutant #4 - device's type IS in validCategories[category]

5.4.1 Description

Small syntactical change to flip check for if device is or is not in the category's hashmap. Removes switches the branch to consume the loop.

5.4.2 Pseudo-code

```
string calculateDeviceDistribution(){
    IF this.deviceGraph is null
        return "Device Distribution Summary Failed: Graph is uninitialised"
    ENDFOR
    SET devices = this.deviceGraph.getDevices()
    SET validCategories = Hashmap of categories -> Hashmap of types->0
    FOR EACH device in the devices
        IF the device's category is null
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE IF the device's type is in validCategories[category] <---MOD
            RETURN "Device Distribution Summary Failed: Devices have invalid data"
        ELSE
            INCREMENT type's count by 1
        ENDIF
    ENDFOR
    FOR EACH category in the hashmap
        SET categoryDeviceCount = 0
        FOR EACH type associated with the category
            ADD type's count to categoryDeviceCount
        ENDFOR
        ADD category's name and categoryDeviceCount to outputString
        FOR EACH type associated with validCategory[category]
            ADD type name and type's count to outputString
        ENDFOR
    ENDFOR
    RETURN outputString
}
```

5.5 Mutation Test Sets

5.5.1 Test Type

- Test Level; Whitebox testing, integration test between DeviceGraph and Summary Statistics
- Test Techniques: Equivalence class of all device types, mutation testing, boundary testing

5.5.2 Test Cases

DeviceGraph input	Expected Output

1x Device of DeviceType.Router, 2x Device of DeviceType.Extender	`Category Encost Wifi Routers device count: 3`
	`  - Type 'Router' device count: 1`
	`  - Type 'Extender' device count: 2`
	`Category Encost Hub/Controller device count: 0`
	`  - Type 'Hub/Controller' device count: 0`
	`Category Encost Smart Lighting device count: 0`
	`  - Type 'Light Bulb' device count: 0`
	`  - Type 'Strip Lighting' device count: 0`
	`  - Type 'Other Lighting' device count: 0`
	`Category Encost Smart Appliances device count: 0`
	`  - Type 'Kettle' device count: 0`
	`  - Type 'Toaster' device count: 0`
	`  - Type 'Coffee Marker' device count: 0`
	`Category Encost Smart Whiteware device count: 0`
	`  - Type 'Washing Machine/Dryer' device count: 0`
	`  - Type 'Refrigerator/Freezer' device count: 0`
	`  - Type 'Dishwasher' device count: 0`

1x Device with a null DeviceType	`"Device Distribution Summary Failed: Some devices have invalid data"`

5.6 Mutation Score

5.6.1 Mutation Testing Results

- Mutation #1 is caught by input 1
- Mutation #2 is caught by inputs 1
- Mutation #3 is caught by input 1 & 2
- Mutation #4 is caught by input 1

There are 4 individual code paths the system could follow, 1. Exit at deviceGraph == null, 2. Device Category is null, 3. type is not in a category, and all devices are valid. The mutants affect how the code path goes, and none of the error-handling branches have extra nested logic so as long as there is one test case that returns a valid device distribution summary string, with more than 1 device in a category, all mutants are caught. Mutation score: 3/4 = 100%

The aforementioned mutants are good because they are small syntactical changes, which if the full functionality of the code is tested it can be found to ensure the full coverage of the branches/code paths, but if only small cases are tested e.g. 1 singular Hub/Controller, or returns early, they cannot be found. A particularly good mutant is where the category sum is printed instead of the type's count impacting cases where only 1 device may be tested, however, the tests ensure this case is handled by providing more than 1 device for a single category and type.