FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.0

Prepared by: Student 2
SoftFlux Engineer

SoftFlux

May 6, 2023

Contents

Revision History

Name	Date	Reason for Changes	Version

1 Introduction/Purpose

1.1 Purpose

The purpose of this document is to define and describe a set of functional tests that will be used to test a software system called Encost Smart Graph Project (ESGP). The tests in this document are based on all high priority requirements specified in the Software Requirements Specification (SRS), and detailed in the Software Design Specification (SDS) for the Encost Smart Graph Project.

1.2 Document Conventions

This document uses the following conventions:

ESGP:	Encost Smart Graph Project
SRS:	Software Requirements Specification
SDS:	Software Design Specification
FSTP:	Functional Software Test Plan

1.3 Intended Audience and Reading Suggestions

- Developers: developers implementing the software should refer to this document to understand how their implementation will be tested so that they can create the system so that it passes the detailed tests.
- Project Manager: the project manager is able to use this document to observe if there are any gaps in the specified tests or test coverage.
- User: users are able to review this document, (particularly the black-box section), to see if they find any gaps in the specified tests or test coverage.

1.4 Project Scope

Encost is a new and emerging Smart Home development company. They manufacture a series of Smart Home and IoT solutions, including Wifi Routers, Smart Hubs and Controllers, Smart Light Bulbs, Smart Appliances, and Smart Whiteware. Encost is interested in investigating how their smart devices are being used and connected within households across New Zealand. They have worked, in partnership with energy companies and their users, to gather information about the smart devices that have been in use in 100 New Zealand homes between April 2020 and April 2022 (called the Encost Smart Homes Dataset). The Encost Smart Graph Project (ESGP) is a software system that enables the visualisation of Encost's devices using a graph data structure. When provided with the Encost Smart Homes Dataset, ESGP enables users to view all of the devices in the dataset, along with their connection to one another. It also provides verified users with summary statistics on device distribution, location, and connectivity.

2 Specialized Requirements Specification

2.1 Software Design Specification Document

This Test Plan is created from the SDS prepared by Student # 3.

2.2 Username and Password Validation

The SDS specifies when an Encost User is logging in the username is read, validated, and if the provided username is valid it then prompts for the password, which then gets validated. If the provided username is invalid, an error message is displayed and the user is asked if they would like to provide credentials to login as an Encost User or if they would like to login as a Community User. The SRS explains the process as the user being prompted to provide their username, then password, and then validating them together and an error message is displayed if the pairing is invalid. Rather than giving the user the possibility to uncover a username for the system through providing multiple usernames, and being informed if they are valid/invalid, the FSTP works on the user having to provide their username and password before they are validated. If the pairing is invalid an error message explaining 'invalid username and/or password provided' is displayed.

2.3 Encost Smart Devices

Encost Wifi Routers have two device types: Router and Extender. Each household in the dataset only has one Encost Wifi Router with a device type of "Router" (referred to as Household Router). However, a household can have zero or more Encost Wifi Routers with device type of "Extender", (referred to as Household Extender). A Household Router does not have a router connection, however a Household Extender does (it is the router of the household it exists in). An Encost Smart Device is connected to an Encost Wifi Router and the device ID of the router is stored with the device as the router connection. The number of Encost Hubs/Controllers that are sending commands to an Encost Smart Device is determined by the number of Hubs/Controllers that have the same router connection as an Encost Smart Device.

2.4 User Navigation

The SDS does not specify how the user will indicate the options they would like to perform. The options available to the user at each step will be associated with a number, to select an option they will have to type the number associated with the option and then press enter. This is done to avoid the user having to type out the full description of the option they want to perform as it is likely they will not write the option as it appears on the console. When an Encost user has to provide a username and password or if they select to load in a custom data set and have to provide a filepath, the user will have to type these out in full as the input for these three items is unlimited.

2.5 Additional Methods

In order to test certain functionality of the system additional methods need to be added to some classes:

2.5.1 UserVerifier Class

In the SDS currently no methods return a boolean indicating if the username and password are for an Encost verified user. An additional method called verifyUser(String username, String password) will be added to the class. This method will take two parameters; the username and password to check, and returns a boolean to indicate if the provided pair of credentials are valid or invalid. If the username and password provided are a valid set of credentials the return value will be true and if the provided credentials are invalid the returned value will be false. The username field and the verifyUsername(username: String) and verifyPassword(password: String) methods can be removed from the class.

An additional method called checkUserType(char input) will be created which will be passed the console input received in the getUserVersion() method of the ConsoleApp class. checkUserType(char input) will return a string categorising the user from their provided input. It's parameter input represents the user-type the user has entered (1 or 2). checkUserType will return a string value that categorises the user. If checkUserType is passed the char '1' it will return a string with the value "community", if it is passed the char '2' it will return a string with the value "encost-unverified". Any char value that is not 1 or 2 will return a string with the value "invalid".

2.5.2 DeviceGraph Class

The getNeighbors(deviceId: String) method of the class returns an array of Devices that represent all the neighbors of the device with the passed in deviceId. If an invalid deviceId is passed in or the deviceId passed in has no neighbors it will return null.

2.5.3 DeviceType Enumeration

The DeviceTypes for each of the Device categories will be the following:

- WIFI_ROUTER for Encost WifiRouter
- HUB_CONTROLLER for Encost Hub/Conroller
- WHITEWARE for Encost Smart Whiteware
- LIGHTING for Encost Smart Lighting
- APPLIANCE for Encost Smart Appliance

2.5.4 GraphVisualiser class

To test that it is correctly converting the deviceGraph into a Graph (using convertGraph(deviceGraph: DeviceGraph)), an extra method will be created in the GraphVisualiser class called getNumNodes(), it will take no parameters and return the number of nodes in the graph (zero if graph is null). Another method will be created called getNumEdgeCount(), it will take no parameters and return the number of edges in the graph (zero if graph is null).

2.5.5 ConsoleApp Class

In selectFeature() after the user has selected a feature information will be displayed and then the ESGP Feature Options for the usertype will be reprinted so that the User can select another option.

2.5.6 DataSummary Class

A method will be created called calcNumDevicesByCategory() which will be used by the calculateDeviceDistribution() method. calcNumDevicesByCategory() will take no parameters and will return an enum map where each of the enums in the enum type DeviceType is used as the key, and the Integer value is the number of devices in that device category. If the deviceGraph in DataSummary is null it will return null.

3 Black-box Testing

3.1 Categorizing Users

3.1.1 Description

Users of the application should be able to indicate if they are a Community User or an Encost User. As described in Section 2.4 the user will navigate the system by typing numbers and pressing enter. The user will enter '1' to indicate they are a community user and '2' to indicate they are an Encost user. Retrieving the user type is handled by the getUserVersion() method in the ConsoleApp class. A method will be created called checkUserType(char input) which will be passed the console input received in the getUserVersion() and will return a string categorising the user from their provided input. checkUserType will take a single character as a parameter which represents the user-type the user has entered (1 or 2). checkUserType will return a string value that categorises the user. If checkUserType is passed the char '1' it will return a string with the value "community", if it is passed the char '2' it will return a string with the value "encost-unverified". Any char value that is not 1 or 2 will return a string with the value "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 test

- **Test technique:** Expected inputs, edge cases, boundary cases

3.1.4 Test Cases

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

3.2 ESGP Account Login

3.2.1 Description

Encost employees should be able to log in to the system as an Encost User to be able to access additional features. Logging in as an Encost User is be handled by the verifyUser() method in the ConsoleApp class. The verifyUser() method will make use of the UserVerifier class to perform the verification. The UserVerifier class will be modified to have a method called verifyUser(String username, String password). This method will take two parameters; the username and password to check, and returns a boolean to indicate if the provided pair of credentials are valid or invalid. If the user provides a valid set of credentials the return value will be true and if the provided credentials were invalid the returned value will be false.

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;

3.2.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Decision Table, equivalence partitioning

3.2.4 Test Cases

Equivalence partitions (in the order they appear in in the Decision Table):

- Partition of all invalid usernames with an invalid password
- Partition of all invalid usernames with a valid password
- Partition of all valid usernames with an invalid password

- Partition of all valid usernames with a valid password

Input (Username)	**Input (Password)**	**Expected Output**
Invalid	Invalid	False
Invalid	Valid	False
Valid	Invalid	False
Valid	Valid	True

3.3 ESGP Feature Options

3.3.1 Description

Users need to be able to indicate whether they want to (1) load a custom dataset; (2) visualise a graph representation of the data; or (3) view summary statistics. Retrieving the feature is handled by the selectFeature() method. It takes no parameters and does not return a value. selectFeature() handles calling the appropriate method to handle the feature the user selects which then prints the appropriate message/prompt to console.

For Community Users:

- The user will enter ’1’ to indicate to visualise a graph representation of the data (Encost Smart Homes Dataset).

For Encost Users:

- The user will enter ’1’ to indicate to visualise a graph representation of the data (Encost Smart Homes Dataset or the loaded custom dataset).
- The user will enter ’2’ to indicate to load a custom dataset.
- The user will enter ’3’ to indicate to view summary statistics of the data (Encost Smart Homes Dataset or the loaded custom dataset).

Expected Console Output for each option:

1. ’1’ - the ESGP Feature Options for the usertype being reprinted to the screen (so that the user can select another option after they close the graph visualisation)
2. ’2’ - prompt asking for the full file path of the location of the custom dataset: ”Please provide the full filepath of the custom dataset: ”
3. ’3’ - immediate message stating ”Summary Statistics calculating...”. After they are calculated, the summary statistics (Device Distribution, Device Location and Device Connectivity) of the data (Encost Smart Homes Dataset or the loaded custom dataset) followed by the ESGP Feature Options for the usertype being reprinted to the screen (so that the user can select another option).
4. Any other char will print a message ”Please provide a valid Feature Option e.g. ’1’ to visualise a graph representation of the data”.

3.3.2 Functional Requirements Tested

SRS 4.3 REQ-2 Once the user has selected a feature, the system should provide them with the prompt/information for that feature.

3.3.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Expected inputs, edge cases, boundary cases

3.3.4 Test Cases

Input	Expected Output (Printed to Console)
'1'	The ESGP Feature Options for the usertype being reprinted to the screen (so that the user can select another option after the close the graph visualisation)
'2'	"Please provide the full filepath of the custom dataset: "
'3'	"Summary Statistics calculating...". After they are calculated, the summary statistics (Device Distribution, Device Location and Device Connectivity) of the data (Encost Smart Homes Dataset or the loaded custom dataset) followed by the ESGP Feature Options for the usertype.
'0'	"Please provide a valid Feature Option e.g. '1' to visualise a graph representation of the data"
'4'	"Please provide a valid Feature Option e.g. '1' to visualise a graph representation of the data"
'a'	"Please provide a valid Feature Option e.g. '1' to visualise a graph representation of the data"

3.4 Loading the Encost Smart Homes Dataset

3.4.1 Description

The parseFile() method of the FileParser class is used to extract the Encost Device information from the lines of a file. The parseFile method takes a String, (filepath; the full file path of where the file is located), as a parameter and returns an array of Devices extracted from the passed in file, if the file exists and is correctly formatted, otherwise it returns null. Each line in the Encost Dataset text file contains the information to create one Device object and it is assumed to be correctly formatted. E.g. If the Encost Dataset file contains fifteen lines then parseFile() should return a Device array with a length of fifteen.

3.4.2 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 (an example from the Encost Smart Homes Dataset is included below).

3.4.3 Test Type

- **Level of test:** Black-box testing, unit testing
- **Test technique:** Expected cases, boundary cases

3.4.4 Test Cases

Please note the number of devices described in the table does not represent the actual Encost Dataset it is just for demonstrating that parseFile() correctly extracts the information from the file.

Input	**Expected Output**
Valid Encost Dataset File Path (Containing 15 lines)	Device Array with a length equal to the number of lines in the Encost Dataset File (15) and each Device in the array is not null
Invalid Encost Dataset Filepath	Null
Empty string	Null
Null string	Null

3.5 Categorizing Smart Home Devices

3.5.1 Description

The system should categorize each Encost Smart Device into one of five categories: Encost Wifi Router, Encost Hubs/Controllers, Encost Smart Lighting, Encost Smart Appliances and Encost Smart Whiteware. These categories are used in the graph visualisation and summary statistics. The parseFile() method of the FileParser class to extract the Encost Device information and categorises the devices. This method takes a String, (filepath - the full file path of where the file is located), as a parameter and returns an array of Devices extracted from the passed in file, (if the file exists and is correctly formatted), otherwise it returns null. Each line in the Encost Dataset text file or a custom dataset contains the information to create one Device object. It is assumed the Encost Dataset is correctly formatted. E.g. If the Encost Dataset file contains fifteen lines then parseFile() should return a Device array with a length of fifteen. The categorisation is seen in the deviceType field of the Device class as specified in the SDS. The deviceType field should be one of the values: WIFI_ROUTER, APPLIANCE, LIGHTING, HUB_CONTROLLER and WHITEWARE. The other fields of the Device class:

deviceId (String), dateConnected (Date), deviceName (String), householdId (String), routerConnection (String), sends (boolean), receives (boolean) should also be set when creating the Device. The router connection of an Encost Wifi Router is assumed to be set to an empty string. This means all Devices in the array returned from parseFile() should have a deviceId (String), dateConnected (Date), deviceName (String), householdId (String), routerConnection (String); i.e. each of these fields should not return null.

3.5.2 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.

3.5.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Expected cases, boundary cases

3.5.4 Test Cases

Please note the number of devices described in the table does not represent the actual Encost Dataset it is just for demonstrating that parseFile() correctly extracts the information from the file and categorises the devices.

Input	**Expected Output**
Dataset filepath containing 5 lines for Encost WifiRouters and 4 lines for Encost Hubs	Null (File is incorrectly formatted)
Encost Dataset File Path (Containing 15 lines): 5 lines for Encost Wifi Routers, 4 lines for Encost Smart Appliances, 3 lines for Encost Smart Lighting, 2 lines for Encost Smart Hubs/Controller and 1 line for Encost Smart Whiteware	Device Array with a length equal to the number of lines in the Encost Dataset File (15) containing 5 Device objects with the DeviceType WifiRouter, 4 Device objects with the DeviceType SmartAppliance, 3 Device objects with the DeviceType SmartLighting, 2 Device objects with the DeviceType HubController and 1 Device object with the DeviceType SmartWhiteware. Each of these Devices fields: deviceId, dateConnected, deviceName, householdId, and routerConnection should not return null.

3.6 Building a Graph Data Type

3.6.1 Description

The system needs to create a graph data structure to store each individual Encost Smart Device Object. The created graph is used to visualise the Encost Smart Devices and to generate summary statistics. The graph data type is represented by the DeviceGraph class which consists of an array of Devices and an adjacency matrix which represents the connections between devices. The parseFile() method of the FileParser class is used to extract the Encost Device information from a passed in file. This method takes a String, (filepath - the full file path of where the file is located), as a parameter and returns an array of Devices extracted from the passed in file, if the file exists and is correctly formatted, otherwise it returns null. Each line in the Encost Dataset text file contains the information to create one Device object and is assumed to be correctly formatted. E.g. If the Encost Dataset file contains fifteen lines then parseFile() should return a Device array with a length of fifteen. The DeviceGraph constructor takes one parameter; an array of Devices, (which will be obtained using parseFile()), and does not return any values. The getDevices() method of the DeviceGraph class returns the array of Devices in the class. To test DeviceGraph is correctly storing the devices in the graph data structure, after intialising a DeviceGraph with the returned array from parseFile(), the length of the device array returned from getDevices() will be checked with the length of the returned array from parseFile().

3.6.2 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;

3.6.3 Test Type

- **Level of test:** Black-box testing, integration: FileParser and DeviceGraph
- **Test technique:** Expected cases, boundary cases

3.6.4 Test Cases

For demonstration purposes:
The file at "Valid example file path v1" contains the following devices:
WifiRouter (id: EWR-1234), SmartAppliance (router id: EWR-1234), SmartWhiteware (router id: EWR-1234), HubController (router id: EWR-1234), WifiRouter (id: EWR-6789), SmartAppliance (router id: EWR-6789).

The file at "Valid example file path v2" contains the following devices:
WifiRouter (id: EWR-1234), WifiRouter (id: EWR-6789).

Filename given to parseFile()	**Expected Number of Devices**
””	0
Invalid file path	0
Valid example file path v1	6
Valid example file path v2	2

3.7 Graph Visualisation

3.7.1 Description

The system needs to allow the user to view a visualisation of the created graph data structure. Creating the visulisation is handled by the GraphVisualiser class. It is passed a DeviceGraph object in its convertGraph() method which converts it into a Graph-Stream graph. The created graph is then used to produce the visualisation. If the passed in DeviceGraph is null the graph will not be created. To test that it is correctly converting the deviceGraph into a Graph an extra method will be created in the GraphVisualiser class called getNumNodes() it will take no parameters and return the number of nodes in the graph (zero if graph is null). Another method will be created called getNumEdgeCount() it will take no parameters and return the number of edges in the graph (zero if graph is null). To easily test this the parseFile() method of the FileParser class will be used to obtain an array of devices. parseFile() takes one string parameter (filepath - the full file path of where the file is located), and returns an array of Devices which can then be given to DeviceGraph to create a DeviceGraph object.

3.7.2 Functional Requirements Tested

SRS 4.8 REQ-1 The graph visualisation must be implemented using the Graph-Stream library;

SRS 4.8 REQ-2 The graph visualisation must show all nodes in the graph data structure;

SRS 4.8 REQ-3 The graph visualisation must show all connections between nodes (i.e. edges) in the graph data structure;

3.7.3 Test Type

- **Level of test:** Black-box testing, integration: FileParser, DeviceGraph and GraphVisuliser
- **Test technique:** Expected cases, boundary cases

3.7.4 Test Cases

For demonstration purposes: The file at ”Valid example file path v1” contains the following devices:

WifiRouter (id: EWR-1234), SmartAppliance (router id: EWR-1234), SmartWhiteware (router id: EWR-1234), HubController (router id: EWR-1234), WifiRouter (id: EWR-6789), SmartAppliance (router id: EWR-6789)
In summary, it has a total of six nodes in the graph it will construct. Router EWR-1234 is connected to three devices, and Router EWR-6789 is connected to one device, so the number of edges in the graph this file will construct is four.

The file at "Valid example file path v2" contains the following devices:
WifiRouter (id: EWR-1234), WifiRouter (id: EWR-6789).
In summary, it has a total of two nodes in the graph it will construct. Neither Router is connected to any devices, so the number of edges in the graph this file will construct is zero.

Filename given to parseFile()	**Expected Number of Nodes**	**Expected Number of Edges**
""	0	0
Invalid file path	0	0
Valid example file path v1	6	4
Valid example file path v2	2	0

3.8 Calculating Device Distribution

3.8.1 Description

The system should give Users logged in as verified Encost Users the ability to view the distribution of devices across category and type, based on the information stored in the graph data structure. The DataSummary class handles calculating summary statistics. A method will be created called calcNumDevicesByCategory() it takes no parameters and will return an enum map where each of the enums in the enum type DeviceType is used as the key, and the Integer value is the number of devices in that device category. If the deviceGraph in DataSummary is null it will return null. The parseFile() method of the FileParser class is used to extract the Encost Device information from a passed in file. This method takes a String, (filepath - the full file path of where the file is located), as a parameter and returns an array of Devices extracted from the passed in file, if the file exists and is correctly formatted, otherwise it returns null. Each line in the Encost Dataset text file contains the information to create one Device object and is assumed to be correctly formatted. E.g. If the Encost Dataset file contains fifteen lines then parseFile() should return a Device array with a length of fifteen. The DeviceGraph constructor takes one parameter; an array of Devices, (which will be obtained using parseFile()), and does not return any values. The getDevices() method of the DeviceGraph class returns the array of Devices in the class, this will be passed to the DataSummary class constructor so that calcNumDevicesByCategory() can be tested.

3.8.2 Functional Requirements Tested

SRS 4.9 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;

3.8.3 Test Type

- **Level of test:** Black-box testing, integration: FileParser, DeviceGraph, and DataSummary
- **Test technique:** Expected cases, boundary cases

3.8.4 Test Cases

For demonstration purposes: The file at "Valid example file path v1" contains the following devices:
WifiRouter (id: EWR-1234), SmartAppliance (router id: EWR-1234), SmartWhiteware (router id: EWR-1234), Smart Lighting (router id: EWR-1234), HubController (router id: EWR-1234), WifiRouter (id: EWR-6789), SmartAppliance (router id: EWR-6789), SmartAppliance (router id: EWR-6789), HubController (router id: EWR-6789)

The file at "Valid example file path v2" contains the following devices:
WifiRouter (id: EWR-1234), WifiRouter (id: EWR-6789).

	Expected Number of Devices by Category				
Filename given to parseFile()	**WifiRouter**	**SmartAppliance**	**SmartLighting**	**SmartWhiteware**	**HubController**
""	Null	Null	Null	Null	Null
Invalid file path	Null	Null	Null	Null	Null
Valid example file path v1	2	3	1	1	2
Valid example file path v2	2	0	0	0	0

4 White-box testing

4.1 Minimum and Maximum Number of Devices Connected to an Encost Wifi Router Pseudocode

4.1.1 Description

The system should use the information stored in the graph data structure to calculate the minimum number and maximum number of devices that an Encost Wifi Router is connected to.

4.1.2 Functional Requirements Tested

SRS 4.11 REQ-2 The system should use the information stored in the graph data structure to calculate the minimum number and maximum number of devices that an Encost Wifi Router is connected to;

4.1.3 Test Type

- **Level of test:** White-box testing, unit test
- **Test technique:** Expected cases, boundary cases, coverage

4.1.4 Pseudocode

```
public int [ ] calcMinMaxRouterConnectedDevices(Device [ ] devices) {
  IF device array is not null AND length is one or greater //br1
    Declare minimum number of devices variable and set it to negative one
    Declare maximum number of devices variables and set it to zero
    FOR all the devices in the device array //br2
      IF devicetype is WifiRouter //br3
        Get device id
        Get neighbors from the deviceGraph using the retrieved device id
        IF neighbors is not null //br4
          IF number of neighbors is less than minimum number of devices
          OR minimum number of devices is negative one //br5
            Set minimum number of devices to number of neighbors
          END IF
          IF number of neighbors is greater than maximum number of devices //br6
            Set maximum number of devices to number of neighbors
          END IF
        ELSE //br7
            Set minimum number of devices to zero
         ENDIF
```

```
          ENDIF
      ENDFOR
      RETURN minimum number of devices and maximum number of devices
   ELSE //br8
      RETURN null
ENDIF

}
```

4.2 Branch Coverage Testing

Contents of Device Array	**Branches Covered**	**Expected Minimum**	**Expected Maximum**
Case 1: Null array	br8	Null	Null
Case 2: Empty array	br8	Null	Null
Case 3: WifiRouter (id: EWR-1234), WifiRouter (id: EWR-4321)	br1, br2, br3, br7	0	0
Case 4: WifiRouter (id: EWR-1234), SmartAppliance (router id: EWR-1234), SmartWhiteware (router id: EWR-1234), HubController (router id: EWR-1234), WifiRouter (id: EWR-6789), SmartAppliance (router id: EWR-6789)	br1, br2, br3, br4, br5, br6	1	3

Total Coverage: br1, br2, br3, br4, br5, br6, br7, br8 = 8/8 = 100% branch coverage

5 Mutation Testing

5.1 Mutant #1

5.1.1 Description

Check for if the device type is a WifiRouter is inverted.

5.1.2 Pseudocode

```
IF devicetype is NOT WifiRouter
```

5.1.3 Test Type

- **Level of test:** White-box testing, unit test
- **Test technique:** Expected cases, boundary cases, coverage, mutation

5.2 Mutant #2

5.2.1 Description

Check for if number of neighbors removes handling of checking if this is the first time the neighbors of a WifiRouter are being checked (negative one flag).

5.2.2 Pseudocode

```
IF number of neighbors is less than minimum number of devices //no OR condition
```

5.2.3 Test Type

- **Level of test:** White-box testing, unit test
- **Test technique:** Expected cases, boundary cases, coverage, mutation

5.3 Mutant #3

5.3.1 Description

Updating the maximum number of devices rather than the minimum number of devices if number of neighbors is less than minimum number of devices:

5.3.2 Pseudocode

```
IF number of neighbors is less than minimum number of devices OR minimum number of devices is negative one
    Set maximum to number of neighbors
```

5.3.3 Test Type

- **Level of test:** White-box testing, unit test
- **Test technique:** Expected cases, boundary cases, coverage, mutation

5.4 Mutant #4

5.4.1 Description

Update the maximum number of devices if the number of neighbors is equal to the maximum number of devices:

5.4.2 Pseudocode

```
IF number of devices is equal to the maximum number of devices
    Set maximum to number of neighbors
```

5.4.3 Test Type

- **Level of test:** White-box testing, unit test
- **Test technique:** Expected cases, boundary cases, coverage, mutation

5.5 Test Cases For All Mutations

	Contents of Device Array	**Expected Minimum**	**Expected Maximum**
Case 1:	WifiRouter (id: EWR-1234), WifiRouter (id: EWR-4321)	0	0
Case 2:	WifiRouter (id: EWR-1234), SmartAppliance (router id: EWR-1234), SmartWhiteware (router id: EWR-1234), HubController (router id: EWR-1234), WifiRouter (id: EWR-6789), SmartAppliance (router id: EWR-6789)	1	3

5.6 Mutation Score

	Test		**Mutant 1**	**Mutant 2**	**Mutant 3**	**Mutant 4**	**Non-Mutant**
Output	**Case 1**	**Min**	-1	0	0	0	0
		Max	0	0	0	0	0
	Case 2	**Min**	1	-1	-1	0	1
		Max	1	3	1	0	3

Mutations caught: 4/4
Mutation Score: 100%
However it is important to note that without Case 2 three out of the four mutants would not have been caught. These are good mutants to include as they are easy mistakes to make in the implementation, e.g., checking something is not equal to something else rather than being equal, (mutant 1), checking something is equal to something else rather than greater than it, (mutant 4), and updating the wrong but similar named variable, (mutant 3). Mutant 2 is an important mutant to include as the OR condition that is omitted handles setting the minimum number of devices to the first encountered WifiRouters connected devices. This is important as we cannot assume the minimum

number of devices connected to a WifiRouter will be 0 hence it being set as a flag value (-1) to enable it to be cleanly assigned. Removing this OR statement means the number of connected devices will never be less than -1 which will result in an unexpected minimum value being returned.