FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.0

Prepared by: Student 5
SoftFlux Engineer

SoftFlux

May 6, 2024

Contents

1 Introduction/Purpose

1.1 Purpose

The purpose of this document is to outline the functional testing approach for the Encost Smart Graph Project (ESGP). This test plan will ensure that the software meets its intended functional requirements and specifications.

1.2 Document Conventions

Throughout this document, the following conventions will be utilised:

- **ESGP:** Encost Smart Graph Project
- **QA:** Quality Assurance

1.3 Intended Audience

This document is targeted towards the following audience:

- **Software Developers:** Developers involved in the implementation of the ESGP software will benefit from understanding the testing approach and test scenarios outlined in this document. It will provide insights into the expected behaviour of the software and help in identifying potential areas for improvement.
- **Quality Assurance Engineers:** QA engineers responsible for testing the functionality of the ESGP software will find this document valuable for creating detailed test cases and executing functional tests. It will guide them in ensuring that the software meets the specified functional requirements and behaves as expected under different scenarios.
- **Project Managers:** Project managers overseeing the development and testing phases of the ESGP software will gain an understanding of the testing strategy outlines in this document. It will enable them to make informed decisions regarding resource allocation, scheduling, and risk management, ultimately contributing to the successful delivery of the software.

1.4 Reading Suggestions

For an effective understanding of the functional testing approach for the ESGP software, the following reading suggestions are provided:

- **Software Design Specification (SDS) Document:** It is recommended to review the SDS document prepared by Student 3, which provides detailed information about the software design, architecture, and component specifications. Understanding the design decisions and functional requirements outlined in the SDS will facilitate the creation of relevant test cases and scenarios.
- **Software Requirements Specification (SRS) Document:** Familiarise yourself with requirements specified for the ESGP software. This will serve as a reference point for designing test cases and validating the functional behaviour of the software during testing.

1.5 Project Scope

The primary focus of this document is to cover the functional testing of the ESGP software, including, user interface, backend functionality, data processing, and visualisation features.

2 Specialized Requirements Specification

Have you found any ambiguities, missing detail, missing features, etc. in the SDS document? Have you discussed them with your client (your lecturers) and received confirmation/clarification? If so, this information should go here.

- **displayNextPrompt():** A method named displayNextPrompt needs to be created in the EncostSmartGraphProject class which will read the user-type from the from the private field and determine what prompt to display next based on the user type, ESGP Feature Options, ESGP Account Login, or an error message.
- **askforLoginDetails():** A method named askForLoginDetails needs to be created in the EncostSmartGraphProject class which will be in charge of printing to the console, asking the user to input their username and then their password. This method will store the user responses and parse them to the authenticate login method.
- **displayFeatureOptions():** A method named displayFeatureOptions needs to be created in the EncostSmartGraphProject class which will be in charge of printing the ESGP Feature Options to the console based on what the user-type is.

- **displayFeatureChoice():** A method named displayFeatureChoice needs to be created in the EncostSmartGraphProject class which will be in charge of reading the user input from the ESGP Feature Options and displaying the corresponding feature.
- **categoriseDevice():** A method named categoriseDevice needs to be created in the EncostSmartGraphProject class which will be in charge of categorising devices based on their device type. It will take a string as a parameter and will determine the device category based on the validity of that string. It will return the device category as a string.
- **Device class:** A Device class needs to be created which will hold information for each device. The constructor will take fields for all required Device information. This class will be connected to the main class.
- **DeviceGraph class:** A DeviceGraph class needs to be created which will be in charge of creating a graph data structure that holds all the devices. This class holds the devices within the graph data structure in a private field. This class is connected to the main class.

3 Black-box Testing

3.1 Categorizing Users

Users should be able to indicate whether they are a Community User or an Encost User.

3.1.1 Displaying User-Type Choice

3.1.1.1 Description

Verify that the system displays a prompt allowing the user to indicate their user-type as a member of the community or Encost.

3.1.1.2 Functional Requirement Tested

SRS 4.1 REQ-1	A prompt should be included that allows the user to indicate whether they are: (1) a member of the community; or (2) a member of Encost;

3.1.1.3 Test Cases

- **Level of test:** Black-box testing, integration test
  - **User Interface Component:** This component handles the interaction with the user, including displaying prompts and receiving user input.
  - **Main Application Logic:** This component orchestrates the overall flow of the application, including handling user input and directing the user to appropriate actions based on their input.
- **Test technique:** Event-driven testing

Stimulus	Expected Response
Application starts	Prompt asking the user to indicate user-type

3.1.2 Storing User-Type

3.1.2.1 Description

As described in the SDS, users will enter ‘1’ for a community user and ‘2’ for an Encost User. An internal method (defineUserType()) reads this input and categorises users as either ‘community’ or ‘encost-unverified’. The expected output of this test is stored in the User.type variable then the User object is stored as a global class variable within the EncostSmartGraphProject class

3.1.2.2 Functional Requirement Tested

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

3.1.2.3 Test Cases

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

Input	Expected Output
‘1’	“community”
‘2’	“encost-unverified”
‘’	“invalid”
‘9’	“invalid”
‘a’	“invalid”

3.1.3 Displaying Next Prompt Based On User-Type

3.1.3.1 Description

Verify that after the user indicates their user-type, the system present the appropriate prompt for the next action. A method named displayNextPrompt will be executed and the string output of that method will be held in an output variable. The output is then checked to be equal to what should be displayed next, ESGP Feature Options, ESGP Account Login.

3.1.3.2 Functional Requirement Tested

SRS 4.1 REQ-3	Once the user has indicated which group they belong to, the system should provide them with the next prompt. For Community Users, the next prompt will be the ESGP Feature Options. For Encost Users, the next prompt will be the ESGP Account Login.

3.1.3.3 Test Cases

- **Level of test:** Black Box Testing, integration test
  - **User Interface Component:** Responsible for displaying the initial prompt and capturing user input.
  - **Backend Component:** Handles the logic for processing the user's input and determining the next prompt based on the user-type.
  - **defineUserType:** This method manages user-related operations, including updating the user type to "community" or "encost-unverified".
- **Test technique:** Expected inputs, edge cases, boundary cases

Description	Input	Expected Output
User is a Community User	'1'	ESGP Feature Options
User is an Encost User	'2'	ESGP Account Login
Invalid input (other than '1' or '2')	'3'	Error message

3.2 ESGP Account Login

Encost employees should be able to login to the system to access additional features. As described in the SDS, users will be prompted to enter a username and password to determine their verification status.

3.2.1 ESGP Account Login Prompt

3.2.1.1 Description

Verify that the ESGP Account Login correctly prompts the user for their username first and then their password, as specified in the requirements.

The authenticateLogin method will be created which takes these two inputs and compares them with the locally stored username and password pairs. If the inputted username and password is valid, the user-type is updated to be ”encost-verified”.

Verify that once the username and password have been entered, the system checks that the inputs are valid.

Verify that if the username and/or password are invalid, the system informs the user and prompts them to enter their credentials again.

Verify that if the username and password are valid, the system updates the user-type to be “encost-verified” and provides the user with the ESGP Feature Options.

3.2.1.2 Functional Requirement Tested

SRS 4.2 REQ-1	The ESGP Account Login prompt should first prompt the user to enter their username. It should then prompt the user to enter their password;
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-3	If the username and/or password are invalid, the system should inform the user that they have entered an invalid username and/or password and prompt them to enter their credentials again;
SRS 4.2 REQ-4	If the username and password are valid, the system should update the user type to be “encost-verified” and should provide the user with the ESGP Feature Options;

3.2.1.3 Test Cases

- **Level of test:** Black Box Testing, integration test
  - **User Interface Component:** Responsible for displaying the login prompts to the user.
  - **Backend Component:** Responsible for managing the logic behind the login process, including validating user inputs and controlling the flow of prompts.
  - **authenticateLogin:** This method is responsible for validating the credentials entered by the user.
- **Test technique:** State transitioning

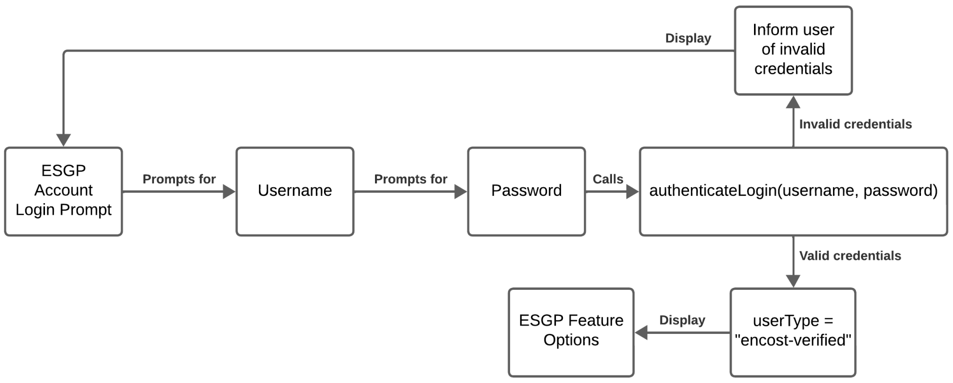

Inform user of invalid credentials
Display
Invalid credentials
ESGP Account Login Prompt
Prompts for
Username
Prompts for
Password
Calls
authenticateLogin(username, password)
Valid credentials
userType = "encost-verified"
Display
ESGP Feature Options

Figure 3.1: Login Prompt Test

3.3 ESGP Feature Options

Users should be able to select whether they want to (a) load a custom dataset; (b) visualise a graph representation of the data; or (c) view summary statistics.

3.3.1 Displaying ESGP Feature Options

3.3.1.1 Description

Verify that the ESGP Feature Options prompt provides the user with the correct selection of features based on their user-type.

3.3.1.2 Functional Requirement Tested

SRS 4.3 REQ-1 The ESGP Feature Options prompt should provide the user with a selection of features that they can pick from. For a Community User there is only one feature available: visualising a graph representation of the data. For a verified Encost User there are three features: (a) loading a custom dataset; (b) visualising a graph representation of the data; or (c) viewing summary statistics;

3.3.1.3 Test Cases

- **Level of test:** Black Box Testing, integration test
  - **User Interface Component:** Responsible for displaying the ESGP Feature Options prompt to the user.
  - **Backend Component:** Handles the logic for determining the available features based on the user type.

  - **defineUserType:** This method is responsible for storing the user type to determine available features.

- **Test technique:** Equivalence partitioning

User Type	Expected Features
Community User	Visualising a graph representation of the data
Encost User	Loading a custom dataset, Visualising a graph representation of the data, Viewing summary statistics

3.3.2 Displaying ESGP Feature Choice

3.3.2.1 Description

Verify that once the user has selected a feature from the ESGP Feature Options, the system displays the corresponding prompt or information for that feature.

3.3.2.2 Functional Requirement 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.

3.3.2.3 Test Cases

- **Level of test:** Black Box Testing, integration test
  - **User Interface Component:** Responsible for displaying the ESGP Feature Options prompt to the user.
  - **Backend Component:** Handles the logic for determining what do display next based off of the user's feature choice.
  - **loadDataset:** This method is responsible for loading and processing a dataset and creating a graph of the data.
  - **generateSummaryStatistics:** This method is responsible for loading and processing a dataset and generating the summary statistics.
- **Test technique:** Equivalence partitioning, boundary values

Selected Feature	Expected Output
Visualising a graph representation of the data ("1")	System displays graph visualisation of Encost Smart Homes Dataset
Loading a custom dataset ("2")	File path prompt for custom dataset
Viewing summary statistics ("3")	Summary statistics are shown for the selected dataset

3.4 Loading the Encost Smart Homes Dataset

The system should be able to read and process the Encost Smart Homes Dataset.

3.4.1 Reading the Encost Smart Homes Dataset

3.4.1.1 Description

The system should be able to read and process the Encost Smart Homes Dataset. This will be done by the processData method within the EncostData class which takes a File object as a parameter. The Encost Smart Homes Dataset is stored locally within a private variable in the EncostSmartGraphProject class.

3.4.1.2 Functional Requirement Tested

SRS 4.4 REQ-1	The Encost Smart Homes Dataset file should be located inside the system;
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.

3.4.1.3 Test Cases

- **Level of test:** Black Box Testing, integration test
  - **loadDataset**: This method is responsible for receiving the dataset file as input and parsing the dataset, extraction relevant device information, and storing it for further processing.
  - **EncostData:** Responsible for containing the Encost Smart Homes Dataset and relevant methods.
- **Test technique:** State transitioning

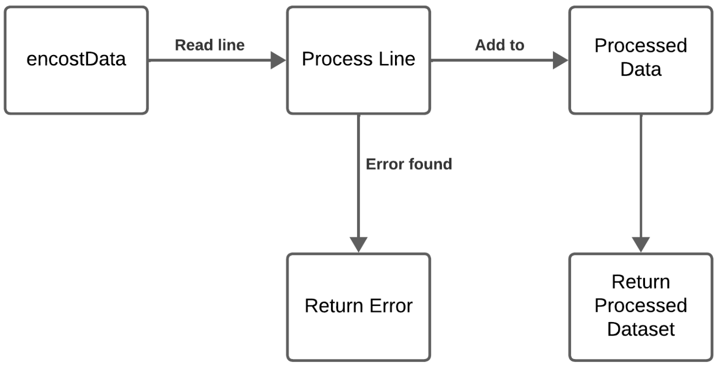


Figure 3.2: Reading Encost Smart Homes Dataset

3.5 Categorizing Smart Home Devices

The system should be able to categorize each Encost Smart Device into one of five categories. These categories will be used for the graph visualisation and summary statistics.

3.5.1 Categorizing Devices

3.5.1.1 Description

Verify that the system determines the device category for each device based on the information provided on each line of the Encost Smart Homes Dataset (or custom dataset). Device categories are pre-defined and should be assigned accurately according to the dataset information.

3.5.1.2 Functional Requirement 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). Device categories are shown in the table below;

3.5.1.3 Test Cases

- **Level of test:** Black Box Testing, unit test
- **Test technique:** Decision table

Device Type	Expected Category
Router, Extender	Encost Wifi Router
Hub/Controller	Encost Hub/Controller
Light Bulb, Strip Lighting, Other Lighting	Encost Smart Lighting
Kettle, Toaster, Coffee Maker	Encost Smart Appliance
Washing Machine, Dryer, Refrigerator, Freezer, Dishwasher	Encost Smart Whiteware

3.5.2 Object Creation

3.5.2.1 Description

Verify that the system creates an object for each device, containing all the information relevant to that device.

3.5.2.2 Functional Requirement 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.

3.5.2.3 Test Cases

- **Level of test:** Black Box Testing, unit test
- **Test technique:** Decision table

Input	Expected Output
Valid device information	Object created with device info
Invalid or incomplete device information	Error or object not created
Empty device information	Error or object not created
Incorrect data format	Error or object not created

3.6 Building a Graph Data Type

The system should create a graph data structure to store all of the Encost Smart Device Objects. This graph will be used to visualise the Encost Smart Devices and to generate summary statistics.

3.6.1 Storing Objects

3.6.1.1 Description

Verify that each Encost Smart Device Object is stored in the graph data structure, with the objects representing the nodes in the graph and the connections between objects representing the edges.

3.6.1.2 Functional Requirement 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;

3.6.1.3 Test Cases

- **Level of test:** Black Box Testing, unit test
- **Test technique:** Decision table

Input	Expected Output
Valid Encost Smart Device Objects	Graph structure created with nodes and connections
No Encost Smart Device Objects provided	Empty graph structure
Duplicate Encost Smart Device Objects	Graph structure created without duplicates

3.7 Graph Visualisation

The system should allow the user to view a visualisation of the graph data structure.

3.7.1 Visualising Connections

3.7.1.1 Description

Verify that the graph visualisation shows all connections between nodes in the graph data structure.

3.7.1.2 Functional Requirement Tested

SRS 4.7 REQ-3	The graph visualisation must show all connections between nodes (i.e. edges) in the graph data structure;

3.7.1.3 Test Cases

- **Level of test:** Black Box Testing, unit test
- **Test technique:** Decision table

Input	Expected Output
Valid graph data structure	All connections between nodes are shown
Empty graph data structure	Error

3.7.2 Visualising Different Device Categories

3.7.2.1 Description

Verify that the graph visualisation distinguishes between different device categories (visually).

3.7.2.2 Functional Requirement Tested

SRS 4.7 REQ-4	The graph visualisation must distinguish between different Device Categories. For example, Encost Smart Lighting nodes should be visually different to Encost Smart Appliances nodes, and so on;

3.7.2.3 Test Cases

- **Level of test:** Black Box Testing, unit test
- **Test technique:** Decision table

Device Category Input	Expected Output
Encost Wifi Routers	Blue node
Encost Hubs/Controllers	Red node
Encost Smart Lighting	Yellow node
Encost Smart Appliances	Green node
Encost Smart Whiteware	White node

3.7.3 Visualising Device Ability

3.7.3.1 Description

Verify that the graph visualisation illustrates each device's ability to send and receive commands from other devices.

3.7.3.2 Functional Requirement Tested

SRS 4.7 REQ-5	The graph visualisation must, in some way, illustrate each device's ability to send and receive commands from other devices. For example, it should be clear that an Encost Smart Hub 2.0 can both send and receive commands, while an Encost Smart Jug can receive commands but cannot send them.

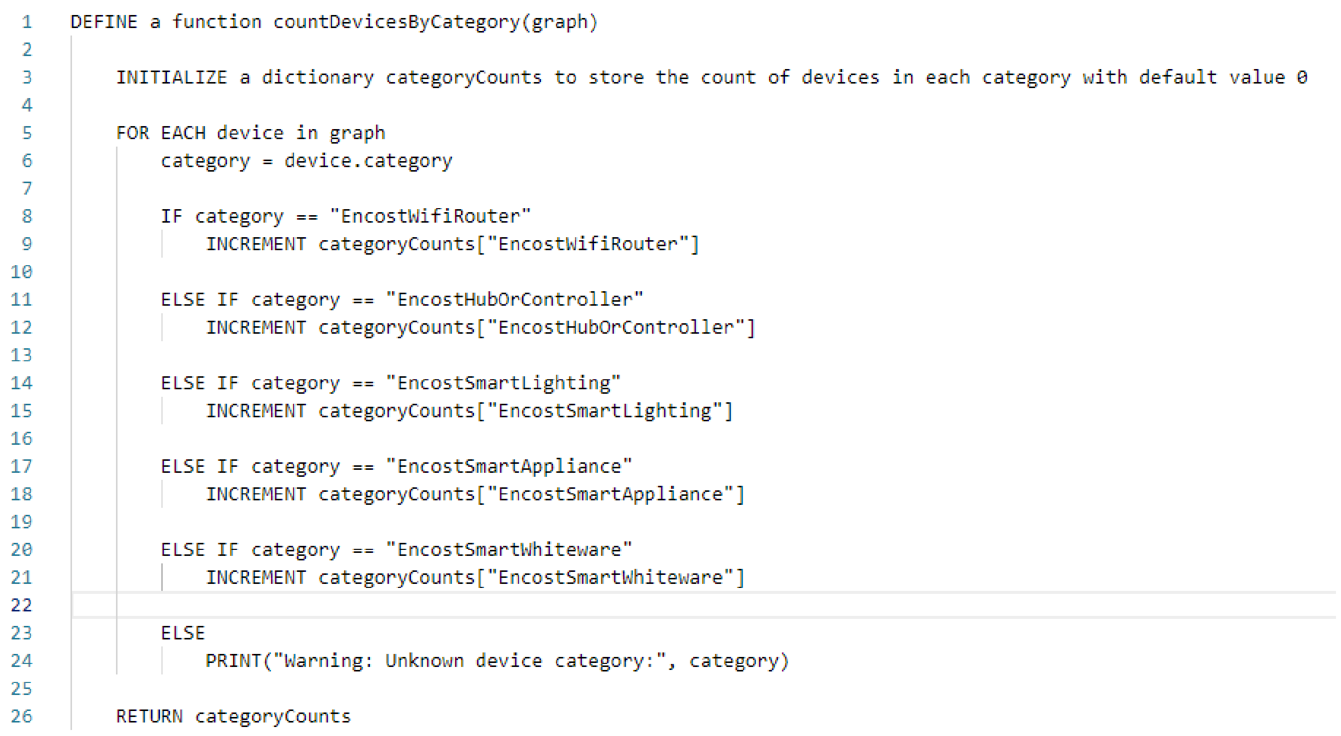
3.7.3.3 Test Cases

- **Level of test:** Black Box Testing, unit test
- **Test technique:** Decision table

Device Category Input	Graph Displays Sends	Graph Displays Receives
Encost Router 360	Yes	Yes
Encost Smart Bulb B22	No	Yes
Encost Smart Jug	No	Yes
Encost Smart Hub 2.0	Yes	Yes

4 White-box testing

4.1 Calculating The Number Of Devices In Each Category Pseudocode



```
DEFINE a function countDevicesByCategory(graph)

    INITIALIZE a dictionary categoryCounts to store the count of devices in each category with default value 0

    FOR EACH device in graph
        category = device.category

        IF category == "EncostWifiRouter"
            INCREMENT categoryCounts["EncostWifiRouter"]

        ELSE IF category == "EncostHubOrController"
            INCREMENT categoryCounts["EncostHubOrController"]

        ELSE IF category == "EncostSmartLighting"
            INCREMENT categoryCounts["EncostSmartLighting"]

        ELSE IF category == "EncostSmartAppliance"
            INCREMENT categoryCounts["EncostSmartAppliance"]

        ELSE IF category == "EncostSmartWhiteware"
            INCREMENT categoryCounts["EncostSmartWhiteware"]


        ELSE
            PRINT("Warning: Unknown device category:", category)

    RETURN categoryCounts
```

4.2 Branch Coverage Testing

4.2.1 Test Case 1: Empty Graph

- **Description:** This tests checks if the function handles an empty graph
- **Input:** An empty graph (no devices)
- **Expected Output:** An empty dictionary *categoryCounts*
- **Branch Coverage:** Ensures the loop iterates zero times

4.2.2 Test Case 2: Single Device - Known Category

- **Description:** This tests verifies the code increments the count for a known categoy
- **Input:** A graph with one device having a known category
- **Expected Output:** A dictionary *categoryCounts* with the device category set to one.
- **Branch Coverage:** Covers the loop iterating once and the first *if* statement condition being true.

4.2.3 Test Case 3: Single Device - Unknown Category

- **Description:** This test checks how the code handles a device with an unknown category
- **Input:** An empty graph with one device having an unknown category
- **Expected Output:** An empty dictionary *categoryCounts*
- **Branch Coverage:** Covers the loop iterating once and the first *if* statement condition being true

4.2.4 Test Case 4: Multiple Devices - Different Categories

- **Description:** This test ensures the code correctly categorises and counts devices with different categories
- **Input:** A graph with multiple devices having different known categories
- **Expected Output:** A dictionary *categoryCounts* with accurate counts for each category based on the input devices
- **Branch Coverage:** Covers the loop iterating multiple times, different *if* statement conditions being true based on device categories

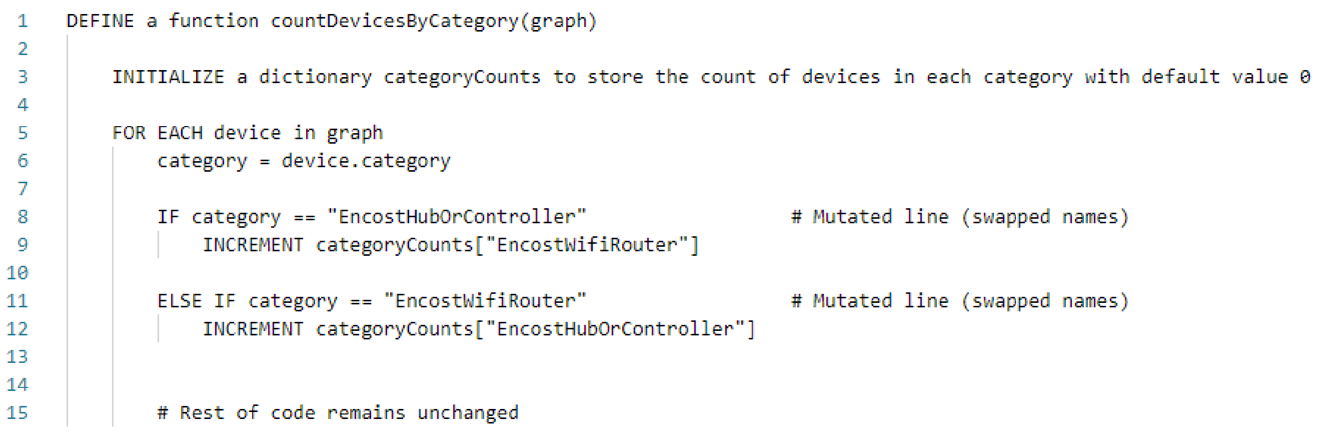
4.2.5 Test Case 5: Multiple Devices - Mix of Known and Unknown Categories

- **Description:** This test combines scenarios from previous tests with devices having know and unknown categories
- **Input:** A graph with multiple devices having a mix of known and unknown categories
- **Expected Output:** A dictionary *categoryCounts* with accurate counts for each known and unknown category based on the input devices
- **Branch Coverage:** Ensures comprehensive loop iterations and execution of all branches

5 Mutation Testing

5.1 Swapping "EncostWifiRouter" With Another Category

This mutant swaps the category names in the first two *if* statements



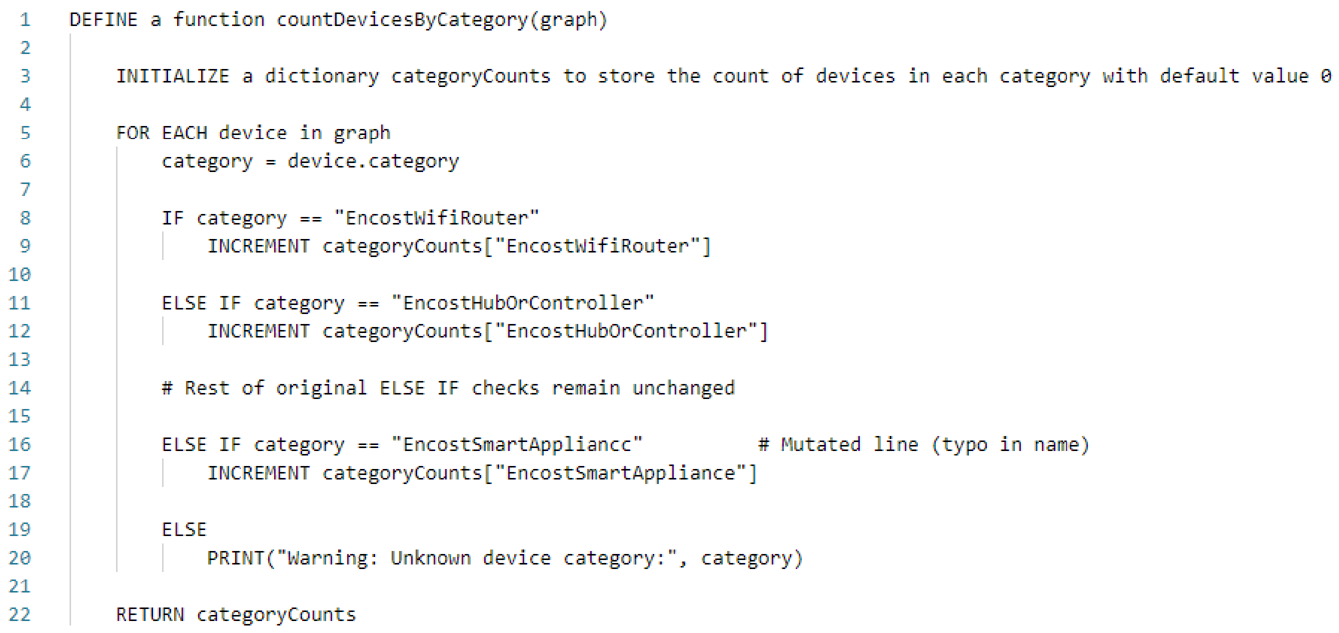
```
DEFINE a function countDevicesByCategory(graph)

    INITIALIZE a dictionary categoryCounts to store the count of devices in each category with default value 0

    FOR EACH device in graph
        category = device.category

        IF category == "EncostHubOrController"              # Mutated line (swapped names)
            INCREMENT categoryCounts["EncostWifiRouter"]

        ELSE IF category == "EncostWifiRouter"              # Mutated line (swapped names)
            INCREMENT categoryCounts["EncostHubOrController"]


        # Rest of code remains unchanged
```

Figure 5.1: Category Swap Mutant Pseudocode

Swapping category names is a common error that can occur during development. This mutation tests if the code behaves as expected when categories are accidentally mixed up.

5.2 Incorrect Category Name

This mutant introduces a typo n the category name for ”EncostSmartAppliance”.



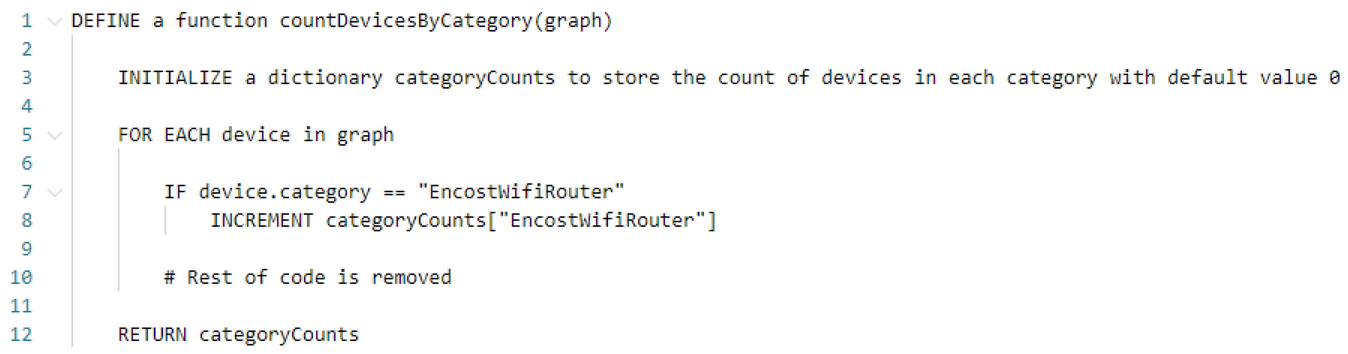
```
DEFINE a function countDevicesByCategory(graph)

    INITIALIZE a dictionary categoryCounts to store the count of devices in each category with default value 0

    FOR EACH device in graph
        category = device.category

        IF category == "EncostWifiRouter"
            INCREMENT categoryCounts["EncostWifiRouter"]

        ELSE IF category == "EncostHubOrController"
            INCREMENT categoryCounts["EncostHubOrController"]

        # Rest of original ELSE IF checks remain unchanged

        ELSE IF category == "EncostSmartAppliancc"          # Mutated line (typo in name)
            INCREMENT categoryCounts["EncostSmartAppliance"]

        ELSE
            PRINT("Warning: Unknown device category:", category)

    RETURN categoryCounts
```

Figure 5.2: Incorrect Category Name Mutant Pseudocode

This data mutation simulates a potential typo in the category name. This might lead to devices being incorrectly categorised or not counted at all, highlighting the importance of data validation.

5.3 Modified IF Conditions

This mutant changes the IF conditions to count only devices with a known category (”EncostWifiRouter”)



```
DEFINE a function countDevicesByCategory(graph)

    INITIALIZE a dictionary categoryCounts to store the count of devices in each category with default value 0

    FOR EACH device in graph

        IF device.category == "EncostWifiRouter"
            INCREMENT categoryCounts["EncostWifiRouter"]

        # Rest of code is removed

    RETURN categoryCounts
```

Figure 5.3: Modified Loop Condition Mutant Psuedocode

This mutant simulates a scenario where the loop might not be iterating over all devices as intended, potentially leading to inaccurate category counts.

5.4 Incorrect Increment Variable

This mutant changes the variable being incremented in the ”EncostSmartLighting” category check.

```
DEFINE a function countDevicesByCategory(graph)

  INITIALIZE a dictionary categoryCounts to store the count of devices in each category with default value 0
  INITIALIZE misc_count = 0  # New variable

  FOR EACH device in graph
    category = device.category

    IF category == "EncostWifiRouter":
      INCREMENT categoryCounts["EncostWifiRouter"]

    ELSE IF category == "EncostHubOrController":
      INCREMENT categoryCounts["EncostHubOrController"]

    ELSE IF category == "EncostSmartLighting":
      INCREMENT misc_count  # Mutated line (wrong variable)

    # Rest of the code remains unchanged
```

Figure 5.4: Incorrect Increment Variable Mutant Pseudocode

This mutation introduces a data error by incrementing a newly introduced variable instead of the intended category count. This could lead to missing data for ”EncostSmartLighting” devices.

5.5 Mutation Score

Two test sets have been provided below:

- **Test Set 1:**
  - **Input:** Graph with 3 devices - 1 "EncostWifiRouter", 1 "EncostHubOrController", 1 unknown category
  - **Expected Output:** {"EncostWifiRouter": 1, "EncostHubOrController": 1, "EncostSmartLighting": 0, "EncostSmartAppliance": 0, "EncostSmartWhiteware": 0}
- **Test Set 2:**
  - **Input:** Graph with 5 devices - 2 "EncostSmartLighting", 2 "EncostSmartWhiteware", 1 "EncostSmartAppliance"
  - **Expected Output:** {"EncostWifiRouter": 0, "EncostHubOrController": 0, "EncostSmartLighting": 2, "EncostSmartAppliance": 1, "EncostSmartWhiteware": 2}

5.5.1 Mutant Output

- **Mutant 1: Swapping "EncostWifiRouter" With Another Category**
  - **Test Set 1 Output:** {"EncostWifiRouter": 1, "EncostHubOrController": 1, "EncostSmartLighting": 0, "EncostSmartAppliance": 0, "EncostSmartWhiteware": 0}
  - **Test Set 2 Output:** {"EncostWifiRouter": 0, "EncostHubOrController": 0, "EncostSmartLighting": 2, "EncostSmartAppliance": 1, "EncostSmartWhiteware": 2}
  - **Killed?:** No
- **Mutant 2: Incorrect Category Name**
  - **Test Set 1 Output:** {"EncostWifiRouter": 1, "EncostHubOrController": 1, "EncostSmartLighting": 0, "EncostSmartAppliance": 0, "EncostSmartWhiteware": 0}
  - **Test Set 2 Output:** {"EncostWifiRouter": 0, "EncostHubOrController": 0, "EncostSmartLighting": 2, "EncostSmartAppliance": 0, "EncostSmartWhiteware": 2}
  - **Killed?:** Yes
- **Mutant 3: Modified IF Conditions**
  - **Test Set 1 Output:** {"EncostWifiRouter": 1, "EncostHubOrController": 0, "EncostSmartLighting": 0, "EncostSmartAppliance": 0, "EncostSmartWhiteware": 0}

- **Test Set 2 Output:** {“EncostWifiRouter”: 0, “EncostHubOrController”: 0, “EncostSmartLighting”: 0, “EncostSmartAppliance”: 0, “EncostSmartWhiteware”: 0}
- **Killed?:** Yes

- **Mutant 4: Incorrect Increment Variable**
  - **Test Set 1 Output:** {“EncostWifiRouter”: 1, “EncostHubOrController”: 1, “EncostSmartLighting”: 0, “EncostSmartAppliance”: 0, “EncostSmartWhiteware”: 0}
  - **Test Set 2 Output:** {“EncostWifiRouter”: 0, “EncostHubOrController”: 0, “EncostSmartLighting”: 0, “EncostSmartAppliance”: 1, “EncostSmartWhiteware”: 2}
  - **Killed?:** Yes

Based on Test Sets 1 and 2, the mutation score for the original code is **3/4 = 0.75**. This means that 75% of the mutants were killed by the test sets, indicating that the tests can detect some potential errors.

6 Conclusion

In conclusion, this functional test plan has outlined a comprehensive strategy for testing the Encost Smart Graph Project. As this project progresses through development, this functional test plan will serve as a guiding document for the testing team, helping validate the software’s functionality.