FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.0

Prepared by: Student 3
SoftFlux Engineer

SoftFlux

May 3, 2024

Contents

Revision History

Name	Date	Reason for Changes	Version
Student 3	26/04/24	Introduction finished	0.1
Student 3	2/04/24	Black Box testing finish for doc	0.2
Student 3	2/05/24	White Box testing finish for doc	0.3
Student 3	2/05/24	Mutation testing finish for doc	0.4
Student 3	2/06/24	Finishing touches	1.0

1 Introduction/Purpose

1.1 Purpose

The purpose of this document is to specify the test plan based on the Encost Smart Graph Project (ESGP) software requirements and design specification. This document will define and describe the tests need for a complete and functional program. Black-box, white-box and mutation testing will be used to create a thorough test plan that covers the functional requirements of this project that were elaborated on in the design document. This document is to used to test with during development to determine if this software is being successfully implemented.

1.2 Document Conventions

This document uses the following conventions:

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

1.3 Intended Audience and Reading Suggestions

This document is intended for any, developer, tester, and project manager. Here are the potential uses for each of the reader types:

- Developer: The developer who wants to read, change, modify or add new test into the existing program, should firstly consult this document and update the test plan in an appropriate manner so as to not destroy the actual meaning of this plan and pass the information correctly to the next phases of the development process.
- Tester: The tester needs this document in order to develop tests which follow the test plan for this system, given in this document, checking if any program which some developer claims to pass these test is actually true.
- Project Manager: The Project Manager can use this document to get an idea of how this software is meant to be tested so that they can better manage the project as a whole.

.

1.4 Project Scope

Encost is a Smart Home company manufacturing WiFi Routers, Smart Hubs, Light Bulbs, Appliances, and Whiteware. They're studying how these devices are used and connected in New Zealand homes. Collaborating with energy companies and users, Encost collected data from 100 New Zealand homes (April 2020 to April 2022) to create the Encost Smart Homes Dataset.

The Encost Smart Graph Project (ESGP) is software that visualizes device connections using graph data. With the dataset, users can explore device relationships and access summary statistics on distribution, location, and connectivity.

2 Specialized Requirements Specification

- For each requirement there should be some output saved in the system which can be accessed by a method for testing.
- When the user goes back in any user interaction it goes back to the previous user interaction. For example, if in the Encost version of feature options you go back to login but, for community users you go back to choosing the type of user.
- GraphStream in the SDS class diagram (fig 3.18) should have a way to store a graph.
- System should hold preset passwords and usernames in a way that doesn't compromise security.
- SDS in class diagram is missing a function for ESGP feature options. It should take a input and output the feature to go to.

- SDS in class diagram in EncostData class there should be a method that is passed a string and returns if it's valid and saves an array, it's called transformData turning data from a file into an array. There should also be a method the takes this array and turns it into a device to store in the dataset called createDevice returns device category. So the dataset should be made up of devices. Therefore, there should be a class for devices that holds their information.
- All data stored in system should be anonymised for security reasons.

3 Black-box Testing

3.1 Categorizing Users

3.1.1 Description

Users should be able to indicate whether they are a Community User or an Encost User. As described in the SDS, users will enter '1' for a community user and '2' for an Encost User. An internal method will be created which takes this input and categorises users as either 'community' or 'encost-unverified'. The method will accept one char parameter which represents the user-type entered by the user (1 or 2). Any char value other than 1 or 2 will result in the return of 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

3.2 ESGP Account Login

Rename the section header to relate to the requirement you are writing a test for

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

3.2.1 Description

The user starts as an ’encost-unverified’ user, as specified in the SRS (Software Requirements Specification). Upon selecting the login option, the user should be prompted with a login window, as shown in the SDS (System Design Specification). The user can then log in using a predetermined username and password, outlined in the SRS. If the user enters a valid login, the program should output ’encost-verified’. However, if the user enters an invalid login, the system should display an error message and restart the login process. In the case of ten consecutive login failures, the system should display an error message, timeout the user as per the SDS. Either way if it fails user type should still be “encost-unverified”.

3.2.2 Functional Requirements Tested

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 usertype to be “encost-verified” and should provide the user with the ESGP Feature Options;

3.2.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Decision table testing

3.2.4 Test Cases

I/O	Rule 1	rule 2	rule 3	rule 4
Username	‘invalid’	‘invalid’	‘valid’	‘valid’
Password	‘invalid’	‘valid’	‘invalid’	‘valid’
Output	“encost-unverified”	“encost-unverified”	“encost-unverified”	“encost-verified”

3.3 ESGP Feature Options

3.3.1 Description

For an Encost user five options appear and for the community users three options appear as shown in the SDS. Then users should be able to indicate which option they would like to use. As mentioned before community users should be able to enter 1 (Visualise graph), 2 (Back) or 3 (Exit). Whereas, for Encost users they should be able to enter 1 (Load a custom dataset), 2 (Visualise graph), 3 (View summary statistics), 4 (Back), 5 (End). The output of these inputs should be a corresponding feature which then can be used somewhere in the program to determine which feature the program should use.

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

Community user

Input	Expected Output
‘1’	“visualiseGraph”
‘3’	“exit”
‘4’	“invalid”
‘a’	“invalid”
‘’	“invalid”

Encost user

Input	Expected Output
‘1’	“loadDataset”
‘5’	“exit”
‘6’	“invalid”
‘a’	“invalid”
‘’	“invalid”

3.4 Loading the Encost Smart Homes Dataset

3.4.1 Description

Each line of the dataset should be passed in as inputs. The line passed in should be checked for correct data length and value types which should be the same format as the example in the SRS(4.4.3). This includes checking if the deviceType is a valid device. Then the data is saved into an array holding the relevant information which is the output as per the SRS.

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.

3.4.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Decision table testing

3.4.4 Test Cases

I/O	Rule 1	rule 2	rule 3	rule 4
dataLength	‘invalid’	‘invalid’	‘valid’	‘valid’
dataTypes	‘invalid’	‘valid’	‘invalid’	‘valid’
Output	“invalid”	“invalid”	“invalid”	“valid”

3.5 Categorizing Smart Home Devices

3.5.1 Description

The input should be an array passed in that should have correct data in it but, invalid data and incorrect deviceType gives error and continues. The array should be checked in the load dataset requirement. Therefore, a device should be able to be created from the input. Then using the device information the device should be able to calculate it’s category as per the SRS(4.6) which it returns after device is saved to dataset. Which the system should be able to get it if the device is not null. Use try catch that returns ”Error: invalid data” if it doesn’t work.

3.5.2 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). Device categories are shown in the table in SRS(4.7.1);
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 inputs, edge cases, boundary cases

3.5.4 Test Cases

Input is the device type section of array.

Input	Expected Output
‘invalid array’	“Error: invalid data”
‘Router’	“Encost Wifi Routers”
‘router’	“Encost Wifi Routers”
‘Light bulb’	“Encost Smart Lighting”
‘’	“Error: invalid data”
‘a’	“Error: invalid data”

3.6 Building a Graph Data Type

3.6.1 Description

The system should add devices to the graph data type Encost data per the SDS. These devices should have the information should have device type, category, send, receive, connection, household and other unique identifiers. Then it should add it to the dataset which will have the devices which include info about the connections between them as per the SRS and SDS.

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

3.6.4 Test Cases

Input	Expected Output
'invalid device' 'valid device with no connection' 'valid device with connection'	"Error device not in dataset" "device in dataset, no connection" "device in dataset, connected to ____"

3.7 Graph Visualisation

3.7.1 Description

The system should then add the graph data structure into the graph stream. Then the graph stream should display the nodes, connections (edges), category and illustrates if it sends and/or receives as per the SRS. The output should be a graph stream with the correct UI window display.

3.7.2 Functional Requirements Tested

SRS 4.8 REQ-2	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 4.8 REQ-3	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;

3.7.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Decision table testing

3.7.4 Test Cases

Decision table when two nodes are added to graph stream trying to connect to each other

I/O	Rule 1	rule 2	rule 3	rule 4
node one node two Output	'invalid' 'invalid' "Error node not in GraphStream"	'valid, node 2' 'invalid' "Error: can't connect to node 2"	'invalid' 'valid, node 1' "Error node not in GraphStream"	'valid, node 2' 'valid, node 1' "node in GraphStream, connected to node2"

4 White-box testing

4.1 Calculating Device Distribution

4.1.1 Description

The system uses the dataset from the EncostData to get the set of devices and passes it into this method. If this dataset has devices then it uses these devices to calulate the number of devices in each category and the number of device types.

4.1.2 Pseudocode

```
method calculateDistribution passed dataset
  category hashmap
  deviceType hashmap
  If dataset is not null
    for each data in dataset
      if new category
        set new category in hashmap
        set category value to one
      else
        add one to category
      end
      if new deviceType
        set new deviceType in hashmap
        set deviceType value to one
      else
        add one to deviceType
      end
    end
    for each category in hashmap
      print category and it's value
    end
    for each deviceType in hashmap
      print deviceType and it's value
    end
  else
  Error no data
  end
end method
```

4.2 Branch Coverage Testing

4.2.1 Test Type

- **Level of test:** White-box testing, unit test
- **Test technique:** Branch coverage testing

4.2.2 Test Cases

Input is the device type for the device added to EncostData dataset.

Input	Expected Output
‘Router’, ‘Light bulb’	“Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1”
‘Router’, ‘Extender’	“Encost Wifi Routers = 2, Router = 1, Extender = 1”
‘Router’, ‘Router’, ‘Light bulb’	“Encost Wifi Routers = 2, Encost Smart Lighting = 1, Router = 2, Light bulb = 1”
‘Extender’, ‘Extender’, ’Extender’, ’Light bulb’	“Encost Wifi Routers = 3, Encost Smart Lighting = 1, Extender = 3, Light bulb = 1”

4.2.3 Justification

Branch testing is testing when breaches are true. This is technically covered by the ‘Router’, ‘Light bulb’ set of inputs as it goes into every truth part of the branches. However, there is a for loop so we have to add tests that use the for loop so that we can estimate 100% coverage stopping any mutations slipping through. This is why the three other tests are added to test to a reasonable point that we can estimate that 100% coverage is achieved.

5 Mutation Testing

The mutation in pseudocode is in bold

5.1 Mutant #1

5.1.1 Description

This mutation checks if the data set is null instead of not null. This should be caught by any test that test has an expected out that gives a proper output when dataset is not null.

5.1.2 Pseudocode

```
method calculateDistribution passed EncostData dataset
  category hashmap
  deviceType hashmap
  If dataset is null
    for each data in dataset
      if new category
        set new category in hashmap
        set category value to one
      else
        add one to category
      end
      if new deviceType
        set new deviceType in hashmap
        set deviceType value to one
      else
        add one to deviceType
      end
    end
    for each category in hashmap
      print category and it's value
    end
    for each deviceType in hashmap
      print deviceType and it's value
    end
  else
  Error no data
  end
end method
```

5.2 Mutant #2

5.2.1 Description

This mutation adds two to a category instead of one when a repeat category is seen. This should be caught by any test that has two input devices with the same category.

5.2.2 Pseudocode

```
method calculateDistribution passed EncostData dataset
  category hashmap
  deviceType hashmap
  If dataset is not null
    for each data in dataset
      if new category
        set new category in hashmap
        set category value to one
      else
        **add two to category**
      end
      if new deviceType
        set new deviceType in hashmap
        set deviceType value to one
      else
        add one to deviceType
      end
    end
    for each category in hashmap
      print category and it's value
    end
    for each deviceType in hashmap
      print deviceType and it's value
    end
  else
  Error no data
  end
end method
```

5.3 Mutant #3

5.3.1 Description

This mutation sets the start value for a category two. This mutation should be caught by any test that doesn't have three devices in only one category.

5.3.2 Pseudocode

```
method calculateDistribution passed EncostData dataset
  category hashmap
  deviceType hashmap
  If dataset is not null
```

```
    for each data in dataset
      if new category
        set new category in hashmap
        **set category value to two**
      else
        add one to category
      end
      if new deviceType
        set new deviceType in hashmap
        set deviceType value to one
      else
        add one to deviceType
      end
    end
    for each category in hashmap
      print category and it's value
    end
    for each deviceType in hashmap
      print deviceType and it's value
    end
  else
  Error no data
  end
end method
```

5.4 Mutant #4

5.4.1 Description

This mutation is instead of adding one to a device type when you see it repeated it sets it to two. This is only caught when there are three or more inputs with the same device type in the test.

5.4.2 Pseudocode

```
method calculateDistribution passed EncostData dataset
  category hashmap
  deviceType hashmap
  If dataset is not null
    for each data in dataset
      if new category
        set new category in hashmap
        set category value to one
      else
```

```
        add one to category
      end
      if new deviceType
        set new deviceType in hashmap
        set deviceType value to one
      else
        **set value two to deviceType**
      end
    end
    for each category in hashmap
      print category and it's value
    end
    for each deviceType in hashmap
      print deviceType and it's value
    end
  else
  Error no data
  end
end method
```

5.5 Mutation Score

5.5.1 Test Type

- **Level of test:** White-box testing, unit test
- **Test technique:** Branch coverage testing

5.5.2 Test Cases

Two selected test sets:

Input	Expected Output
‘Router’, ‘Light bulb’	“Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1”
‘Extender’, ‘Extender’, ’Extender’, ’Light bulb’	“Encost Wifi Routers = 3, Encost Smart Lighting = 1, Extender = 3, Light bulb = 1”

5.5.3 Mutant #1

Original: If dataset is not null
Mutation: If dataset is null

Expected output for both test case inputs:

1. "Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1"
2. "Encost Wifi Routers = 3, Encost Smart Lighting = 1, Extender = 3, Light bulb = 1"

Actual output for both test case inputs:

- Error no data

This shows that both sets of inputs catch mutant 1.

5.5.4 Mutant #2

Original: add one to category
Mutation: add two to category
Expected output for both test case inputs:

1. "Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1"
2. "Encost Wifi Routers = 3, Encost Smart Lighting = 1, Extender = 3, Light bulb = 1"

Actual output for both test case inputs:

1. "Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1"
2. "Encost Wifi Routers = 5, Encost Smart Lighting = 1, Extender = 3, Light bulb = 1"

This shows that the output of the second test case catches mutant 2.

5.5.5 Mutant #3

Original: set category value to one
   add one to category
Mutation: set category value to two
   add one to category
Expected output for both test case inputs:

1. "Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1"
2. "Encost Wifi Routers = 3, Encost Smart Lighting = 1, Extender = 3, Light bulb = 1"

Actual output for both test case inputs:

1. "Encost Wifi Routers = 2, Encost Smart Lighting = 2, Router = 1, Light bulb = 1"
2. "Encost Wifi Routers = 3, Encost Smart Lighting = 2, Extender = 3, Light bulb = 1"

As you can see mutant 3 is also caught by both test case but partially matches the second test case.

5.5.6 Mutant #4

Original: add one to deviceType
Mutation: set value two to deviceType
Expected output for both test case inputs:

1. "Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1"
2. "Encost Wifi Routers = 3, Encost Smart Lighting = 1, Extender = 3, Light bulb = 1"

Actual output for both test case inputs:

1. "Encost Wifi Routers = 1, Encost Smart Lighting = 1, Router = 1, Light bulb = 1"
2. "Encost Wifi Routers = 3, Encost Smart Lighting = 1, Extender = 2, Light bulb = 1"

The fourth mutant is caught by the second test case but for example if it was set to three this mutation would of passed. Which is why in in the full set of test cases (4.2.2) it's important to have the test case with inputs 'Router', 'Router', 'Light bulb' which would catch that variation of mutant four.

5.5.7 Score

The mutant score is 100% but could easily be less with different mutations showing the need for all test cases.

6 References

- Student 3 SDS
- ESGP SRS