FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.0

Prepared by: Student 4
SoftFlux Engineer

SoftFlux

May 5, 2024

Contents

Revision History

Name	Date	Reason for Changes	Version
Student 4	22/04/24	Completed Section 1	0.5
Student 4	23/04/24	Completed Blackbox Section	0.6
Student 4	24/04/24	Completed Whitebox section	0.7
Student 4	25/04/24	Completed Mutant Section	0.8
Student 4	26/04/24	Completed Justification section	0.9

1 Introduction/Purpose

1.1 Purpose

The purpose of this document is to provide a functional software test plan for the Encost Smart Graph Project (ESGP). The plan will outline the techniques, testing strategies and test cases, and what will be used to verify all the correctness and implementations of the ESGP System according to what the Software Requirements Specification (SRS) and the Software Design Documentation provided by student 3 (SDS) are.

1.2 Document Conventions

The following conventions and acronyms are used throughout this document:

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

1.3 Intended Audience and Reading Suggestions

The document is intended for the following:

- Software developers: will utilise this test plan for the ESGP system and will formulate system, with specification requirements are that are met.
- Quality Insurance Engineers: will formulate test plans and make tests cases to further verify the system's functionality, and follow to system requirements.
- Project Managers: They will employ this test plan to monitor testing operations while making informed choices regarding risk management and project planning

The requirements and design of the project, it is advised that you read the SRS and SDS documents that Student 3 delivered , before looking over this test plan.

1.4 Project Scope

The Encost Smart Graph Project (ESGP) is a software application provides summary statistics on device distribution, location, and connectivity. It has visualizes Encost's smart devices using a graph data structure. In addition to processing data from the Encost Smart Homes Dataset, the system will also enable, to load custom datasets, graph visualizations, and summary statistics access for Encost Users who have verified their identity. Community Users will only be able to view the graph visualizations and will have restricted access.

2 Specialized Requirements Specification

1. It is stated in the SDS document: "The User Credentials will be encrypted and decrypted only during run time." However, no information is given regarding the encryption algorithm or technique that should be applied to secure user credentials. Re clarifying and incorporating this idea within the Specialized Requirements Specification can be required.
2. The SRS doesn't explicitly state that custom datasets need to be stored within the persistent storage, according to the SDS document. Therefore, any provided custom data set will only be used in memory and won't be saved for when the application launches again." Although a design choice, even though it's not mentioned in the SRS, it could be helpful to find out if the client (Encost) has a requirement to keep the datasets permanently.
3. To ensure Brute force cannot be used to log in as an Encost user," the SDS paper states. Before a timeout, the user is only allowed ten attempts to log in. Neither the length of the timeout nor the procedure for resetting the login attempts when it ends are specified. For reiteration, these specifics might be incorporated in the Specialized Requirements Specification.

4. The structure and format of the custom datasets that Encost Users can upload are not covered in detail in the SDS paper. Giving details about the data structures, file formats, and validation procedures that are acceptable for use with custom datasets may be useful.

3 Black-box Testing

3.1 Categorizing Users

3.1.1 Description

Users should be able to state 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

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

3.2 User Authentication (Encost User)

3.2.1 Description

The SDS, Encost Users should be able to verify their identity by supplying legitimate login credentials. These actions are required to be taken during the authentication process

1. The user is prompted by the system to enter their password and username.
2. The given password and username are checked by the system against the encrypted user credentials kept on permanent storage.
3. The system gives the verified Encost User access and gives them access to more features if the all the credentials are valid.
4. The system should prompt the user to re-enter their passwords and usernames and display an appropriate error message if the credentials are invalid.
5. A brute-force protection mechanism will be required to be incorporated into the system, permitting a limited amount of unsuccessful login attempts (ten attempts) prior to enforcing a timeout.
6. The system should reactivate the failed login attempt counter and provide the user another chance after the timeout period has passed.

3.2.2 Functional Requirements Tested

- SRS 4.1 REQ-3: After successful authentication, the system must limit Encost Users' access to extra features.
- SRS 4.1 REQ-4: User credentials should be safely stored by the system.
- SRS 4.2 REQ-1: When an input or action is deemed invalid, the system should to indicate the error appropriately.

3.2.3 Test Type

Level of test: Black-box testing, integration testing (with the User Interface and Data Components) Test Technique: Equivalence partitioning, boundary value analysis

3.2.4 Test cases

Test Case	**User Type**	**Steps**	**Expected Outcome**

User Type Selection	Any User	• Launch the application • Observe the user interface • Select the appropriate user type ("1" for Community or "2" for Encost)	The interface should prompt the user to select their user type and provide instructions.
Authentication	Encost User	• Launch app • Select Encost User • Enter valid credentials	Guide user through authentication and display features for Encost Users.
Load Custom Dataset	Encost User	• Launch app • Authenticate as Encost User • Load custom dataset	Provide clear instructions for loading datasets.
Graph Visualization	Any User	• Launch app • Select user type • Visualize the graph	Present graph visualization clearly.

Summary Statistics	Encost User	• Launch app • Authenticate as Encost User • View summary statistics	Display summary statistics in a clear format.

Table 3.1: Test Case Details

3.3 Graph Visualisation

3.3.1 Description

The Encost Smart Hones Dataset or a custom dataset (for Encost Users with verified identities) are able to be seen by the system as a graph data structure, these are a list of the following requirements that should be met by the graph visualisation:

1. Nodes: Every device in the dataset needs to be shown in the graph as a node. The device category should be visibly by the node's size, shape, or color.
2. Edges: Lines or arrows connecting the respective nodes should be used to direct connectivity between devices. The ability to send and receive data between the linked devices should be represented by the edge's orientation and direction.
3. structure: To provide clear visibility and reading, the graph structure should be set up to minimize node and edge overlap.
4. Zooming and Panning: To enable users to easily traverse across and study the graph, particularly for huge datasets, the visualization will include zooming and panning.
5. Interactive Features: Users should be able to interact with the graph by dragging their cursor over nodes or edges to see more details (such the name, category, and connectivity details of the device).
6. Performance: To guarantee smooth rendering and interaction, the graph visualization should be quick to respond, responsive, and capable of managing large datasets.
7. Error Handling: If an incorrect or unsupported dataset format, an empty dataset, or any other problem occurs during visualization, an error messages should be shown.

3.3.2 Functional Requirements Tested

- SRS 4.1 REQ-1: The supplied Encost Smart Homes Dataset should be able to be seen by the system as a graph data structure.
- SRS 4.1 REQ-6: Custom datasets for visualization should be able to be loaded into the system by authenticated Encost Users.
- SRS 4.2 REQ-1: When an input or action is deemed invalid, the system ought to indicate the problem appropriately.
- SRS 4.2 REQ-2: An intuitive user interface is required for the system.

3.3.3 Test type

- Level of test: Black-box testing, integration testing (with the User Interface, Data, and Visualization Components) - Test technique: Equivalence partitioning, boundary value analysis, use case testing

3.3.4 Test cases

Test Case	**User Type**	**Dataset**	**Output**
Default Dataset (Community User)	Community User	Encost Smart Homes Dataset	Graph visualization of the default dataset with nodes representing devices, edges depicting connectivity, and visual cues for device categories
Default Dataset (Encost User)	Encost User	Encost Smart Homes Dataset	Graph visualization of the default dataset with nodes representing devices, edges depicting connectivity, and visual cues for device categories
Valid Custom Dataset (Encost User)	Encost User	Valid custom dataset file	Graph visualization of the custom dataset with nodes representing devices, edges depicting connectivity, and visual cues for device categories

Invalid Custom Dataset (Encost User)	Encost User	Invalid file format or data structure	Error message: "Invalid dataset format"
Empty Custom Dataset (Encost User)	Encost User	Empty file or no data	Error message: "Empty dataset"
Large Custom Dataset (Encost User)	Encost User	Very large dataset file (e.g., 1 GB)	Graph visualization of the large dataset, with appropriate handling of performance and memory constraints

Table 3.2: Dataset Test Case Details for graph visualization

3.4 Load Custom Dataset (Encost User)

3.4.1 Description

Verified Encost Custom datasets need to be allowed to be loaded by users for display in the graph structure. A user-friendly interface for choosing and uploading the custom dataset file is provided by the system. To guarantee that the file format and data structure are compatible with the visualization component, the system should additionally validate them.

3.4.2 Functional Requirements Tested

- SRS 4.1 REQ-6: Custom datasets for visualization should be able to be loaded into the system by authenticated Encost Users.
- SRS 4.2 REQ-1: When an input or action is deemed invalid, the system ought to indicate the problem appropriately.
- SRS 4.2 REQ-2: An intuitive user interface is required for the system.

3.4.3 Test Type

- Level of test: Black-box testing, integration testing (with the User Interface, Data, and Visualization Components) - Test technique: Equivalence partitioning, boundary value analysis, use case testing

3.4.4 Test case

Test case 1: Valid Custom Dataset

Prerequisites: User has been verified as an Encost User
- Actions: 1. The user chooses to a custom dataset.
2. The user submits a legitimate dataset file (in CSV, JSON, or another format that is supported).
3. The user can verify the choice of file
Anticipated Outcome: The customized dataset is loaded and processed by the system with success, and it displays the updated graph visualization.

Test case 2: Invalid File Format

Prerequisites: User has been verified as an Encost User - Actions: 1. The user chooses to load a custom dataset.
2. The user submits a file in an invalid or unsupported format.
3. The user verifies the choice of file
Anticipated Outcome: An error message stating that the file format is invalid or not supported is shown by the system.

Test Case 3: Empty Dataset

Prerequisites: The user has successfully authenticated as an Encost User.
- Actions: 1. The user chooses to import a custom dataset.
2. Uploads a data-free or empty file.
3. The user verifies the choice of file
Anticipated Outcome: An error notice stating that the dataset is empty or devoid of data is displayed by the system.

Test Case 4: Boundary Case - Large Dataset

Prerequisites: User has been verified as an Encost User - Actions:
1. The user chooses to load a custom dataset.
2. The user submits an acceptable but very large dataset file (e.g., 1 GB or more)
3. The user verifies the choice of file
Anticipated Outcome: The system loads and processes the enormous dataset successfully, managing memory and performance restrictions appropriately. The custom dataset is updated in the graph visualization.

Test Case 5: User-friendly Interface

Prerequisites: The user has successfully authenticated as an Encost User.
- Actions: 1. The user selects the option to import a custom dataset.
2. The user views the UI where they can choose and upload the dataset file.

Anticipated Outcome: An easily navigable interface featuring visual signals, clear instructions, and a dataset may be loaded.

3.5 Summary Statistics

3.5.1 Description

Based on the loaded dataset (either the default Encost Smart Homes Dataset or a custom dataset loaded by an authenticated Encost User), the system should be able to compute and show a variety of summary statistics. Insights into the device distribution, classifications, connection, and other data obtained from the graph data structure, this all should be provided by the summary statistics.

3.5.2 Functional Requirements Tested

- SRS 4.1 REQ-5: The system will compute summary statistics using the data kept in the graph data structure.
 - SRS 4.1 REQ-7: To authenticated Encost Users, the system has to show summary statistics.

3.5.3 Test Type

- Level of test: Black-box testing, integration testing (with the Data and Visualization Components) - Test technique: Equivalence partitioning, boundary value analysis, use case testing

3.5.4 Test Cases

Test Case	**User Type**	**Dataset**	**Expected Summary Statistics**
Default Dataset (Community User)	Community User	Encost Smart Homes Dataset	• Total number of devices • Number of devices per category • Connectivity statistics (e.g., average number of connections per device, devices with maximum/minimum connections)

Default Dataset (Encost User)	Encost User	Encost Smart Homes Dataset	• Total number of devices • Number of devices per category • Connectivity statistics (e.g., average number of connections per device, devices with maximum/minimum connections) • Additional statistics based on the SRS and SDS (e.g., devices by location, device categories with highest connectivity)
Valid Custom Dataset (Encost User)	Encost User	Valid custom dataset file	• Total number of devices • Number of devices per category • Connectivity statistics (e.g., average number of connections per device, devices with maximum/minimum connections) • Additional statistics based on the SRS and SDS (e.g., devices by location, device categories with highest connectivity)
Empty Dataset (Encost User)	Encost User	Empty file or no data	Error message: "Empty dataset". No summary statistics displayed.

Large Dataset (Encost User)	Encost User	Very large dataset file	Summary statistics calculated and displayed efficiently, with appropriate handling of performance and memory constraints.
Boundary Case - Single Device (Encost User)	Encost User	Dataset with a single device	Summary statistics correctly calculated and displayed for a single device (e.g., total devices = 1, no connectivity statistics).
Boundary Case - No Connectivity (Encost User)	Encost User	Dataset with devices but no connectivity information	Summary statistics correctly calculated and displayed, excluding connectivity-related statistics.

Table 3.3: Summary Statistics Test Case Details

3.6 Error Handling

3.6.1 Description

When an input or action is invalid, the system should address it and show the relevant error messages. Error messages should be brief, easy to understand, and give the user instructions on how to fix the problem or take correct action solving it.

3.6.2 Functional Requirements Tested

- SRS 4.2 REQ-1: The system should display appropriate error messages for invalid inputs or actions.

3.6.3 Test Type

- Level of test: Black-box testing, integration testing (with all Components) - Test technique: Equivalence partitioning, boundary value analysis, use case testing.

3.6.4 Test Cases

Test Case	**User Type**	**Input/Action**	**Expected Error Message**
Invalid User Type Selection	Any User	User enters a value other than '1' or '2' for user type selection	"Invalid user type selection. Please enter '1' for Community User or '2' for Encost User."

Invalid Username (Encost User)	Encost User	User enters an invalid or non-existent username during authentication	"Invalid username or password. Please try again."
Invalid Password (Encost User)	Encost User	User enters an incorrect password during authentication	"Invalid username or password. Please try again."
Brute-force Protection (Encost User)	Encost User	User exceeds the maximum allowed failed login attempts	"Too many failed login attempts. Please try again after [X] minutes."
Invalid Custom Dataset Format (Encost User)	Encost User	User tries to load a custom dataset with an unsupported or invalid file format	"Invalid dataset format. Please upload a file in the supported format(s)."
Empty Custom Dataset (Encost User)	Encost User	User tries to load an empty dataset file or a file with no data	"Empty dataset. Please upload a valid dataset file."
Invalid Input (Any User)	Any User	User enters an invalid input (e.g., non-numeric value when a number is expected)	"Invalid input. Please enter a valid value."
System Error	Any User	An unexpected system error occurs	"An unexpected error occurred. Please try again later or contact support."

Table 3.4: Error Handling Test Case Details

3.7 User Interface (UI)

3.7.1 Description

An intuitive user interface should be offered by the system so that users may easily engage with its features and options. Users should be guided through all of the features by the interface, including choosing their user type, authenticating (for Encost Users), importing datasets, displaying graphs, and gaining access to summary statistics. The program should have user-friendly navigation, clear instructions, and visual clues.

3.7.2 Functional Requirements Tested

- SRS 4.2 REQ-2: An intuitive user interface is required for the system.
  - SRS 4.2 REQ -3: Users should be guided by the system as they explore the various features and options.

3.7.3 Test Type

- Level of test: Black-box testing, integration testing (with the User Interface Component) - Test technique: Use case testing, usability testing

3.7.4 Test Cases

Table 3.5: Detailed Test Cases and Expected Outcomes for UI

Test Case	**User Type**	**Steps**	**Expected Outcome**
User Type Selection	Any User	1. Launch the application 2. Observe the UI prompting for user type selection 3. Select appropriate user type (”1” for Community or ”2” for Encost)	The interface should prompt the user to select their user type and provide instructions on how to proceed. The selected user type should be correctly processed, and the user should be directed accordingly.
Authentication (Encost User)	Encost User	1. Launch the application 2. Select the Encost User option 3. Observe authentication interface 4. Enter valid login credentials 5. Observe interface after successful authentication	The interface should guide the user through the authentication process and display features for Encost Users upon successful authentication.

(Table 3.5 continued from previous page)

Test Case	**User Type**	**Steps**	**Expected Outcome**
Loading Custom Dataset (Encost User)	Encost User	1. Launch the application 2. Authenticate as an Encost User 3. Select option to load custom dataset 4. Provide dataset file path or browse for the file 5. Observe interface after successful dataset loading	The interface should provide clear instructions for loading a custom dataset and indicate completion with options for visualization or accessing statistics.
Graph Visualization	Any User	1. Launch the application 2. Select appropriate user type 3. Choose option to visualize the graph 4. Observe graph visualization interface	The interface should clearly present graph visualization with necessary instructions and user-friendly features.

(Table 3.5 continued from previous page)

Test Case	User Type	Steps	Expected Outcome
Summary Statistics	Encost User	1. Launch the application 2. Authenticate as an Encost User 3. Select option to view summary statistics 4. Observe interface displaying summary statistics	The interface should provide a clear option for Encost Users to access and view readable summary statistics.

3.8 Categorizing Smart Home Devices

3.8.1 Description

Based on the name and type of the device information provided in the dataset, the system should be able to classify each Encost Smart Device into one of five categories. Encost WiFi Routers, Encost Hubs/Controllers, Encost Smart Lighting, Encost Smart Appliances, and Encost Smart White ware are the categories.

3.8.2 Functional Requirements Tested:

The Encost Smart Homes Dataset (or custom dataset) should be used as a source of information for the system to identify the device type for each individual device.

Every device should have an Object created by the system. All of the device's data needs to be stored in this object.

3.8.3 Test Type:

- Level of test: Black-box testing, unit testing
- Test technique: Equivalence partitioning, boundary value analysis

3.8.4 Test Cases

1. Devices from all five categories are included in the sample dataset for valid devices. Devices that are appropriately classified and stored as objects are the expected output.

2. Device Category Error: An example dataset containing an invalid device name or type. Error messages or uncategorized devices are expected outputs.
3. A dataset that is empty. An error message is the result.
4. Single Device: A dataset consisting of just one device. The device should be accurately classified and saved as an item.

3.9 Building a Graph Data Type

3.9.1 Description

All of the Encost Smart Device Objects need to be stored in a graph data structure that the system generates. The Encost Smart Devices will be shown on this graph, which will also be used to produce summary data.

3.9.2 Functional Requirements Tested

The graph data structure should hold each Encost Smart Device Object. The nodes in the graph should be the objects.

The edges should be the point of connection between items.

The graph should contain each distinct data point.

The graph needs to include representations of every home.

3.9.3 Test Type:

- Level of test: Black-box testing, integration testing
- Test technique: Equivalence partitioning, boundary value analysis

3.9.4 Test Cases:

1. A valid dataset for the creation of graphs. A graph data structure with all devices as nodes and all connections as edges is the expected result.
2. A dataset that is empty. An error notice or an empty graph is what is anticipated.
3. One Device: A dataset consisting of just one device. A single-node graph is the anticipated result.
4. Devices from a single household make up the Single Household Dataset.
5. A graph with nodes representing each device in that home is the anticipated outcome.
6. Device Duplicates: A dataset containing duplicate entries for devices.
7. The graph should only contain unique nodes or data points.

4 White-box testing

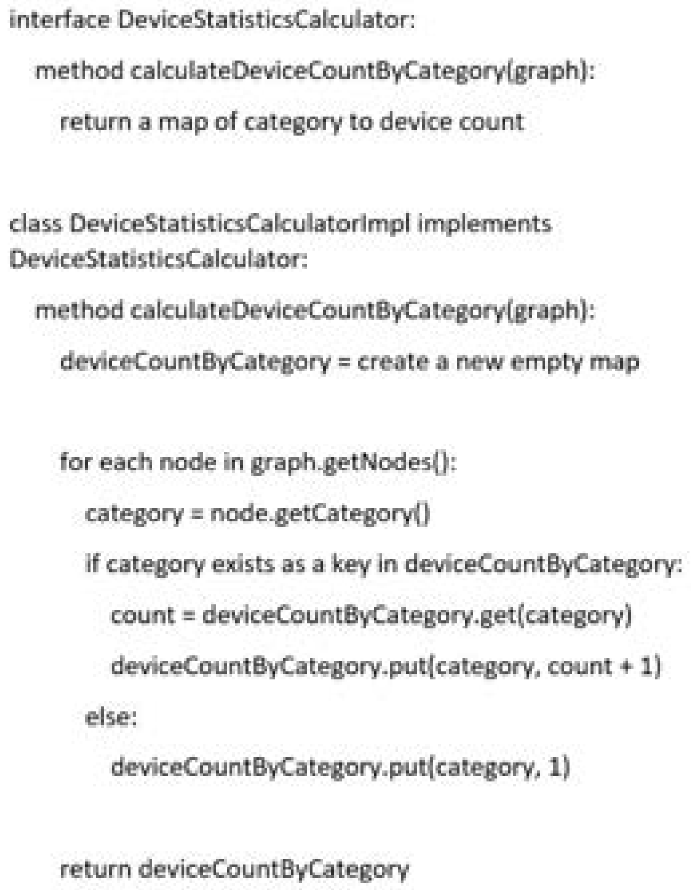
4.1 <Calculating the number of devices in each device category > Pseudo code

4.1.1 Description

The purpose of this test is to confirm that the system counts the devices for each category of devices that are present in the loaded dataset (graph data structure) accurately. A set of test cases that guarantee 100% branch coverage for the pseudocode implementation will be provided by the test.

4.1.2 Pseudocode

Figure 4.1: psuedocode



```
interface DeviceStatisticsCalculator:
    method calculateDeviceCountByCategory(graph):
        return a map of category to device count

class DeviceStatisticsCalculatorImpl implements
DeviceStatisticsCalculator:
    method calculateDeviceCountByCategory(graph):
        deviceCountByCategory = create a new empty map

        for each node in graph.getNodes():
            category = node.getCategory()
            if category exists as a key in deviceCountByCategory:
                count = deviceCountByCategory.get(category)
                deviceCountByCategory.put(category, count + 1)
            else:
                deviceCountByCategory.put(category, 1)

        return deviceCountByCategory
```

In this pseudocode:

1. A method called calculateDeviceCountByCategory is declared in the DeviceStatisticsCalculator interface. It accepts a graph object as input and outputs a map of categories to device counts.
2. The calculateDeviceCountByCategory method's implementation and the DeviceStatisticsCalculator interface are both provided by the DeviceStatisticsCalculatorImpl class.
3. The calculateDeviceCountByCategory method first creates an empty map called deviceCountByCategory.
4. It iterates over all the nodes in the input `graph` object.
5. For each node, the category of the node gets retrieved.
6. It increases the count for that category if it already appears as a key in the deviceCountByCategory map.

7. If the category does not exist in the map, it adds a new entry with the category as the key and a count of 1.
8. Once every node has been iterated over, the deviceCountByCategory map is returned.

4.1.3 Test Cases

Test Case	Input Graph	Output
1	No Nodes	An empty map
2	A single Node ”A”	A map with one entry: {”A”: 1}
3	Multiple nodes of the same category (”A”)	A map with one entry: {”A”: 3}
4	Nodes of different categories (”A”, ”B”, ”C”)	Map with three entries: {”A”: 2, ”B”: 3, ”C”: 1}
5	Nodes of different categories with duplicates (”A”, ”B”, ”A”, ”C”, ”B”)	Three entries: {”A”: 2, ”B”: 3, ”C”: 1}

Table 4.1: Graph Input and Output Test Cases

4.2 Branch Coverage Testing

Level of test

Unit Test

Test Technique

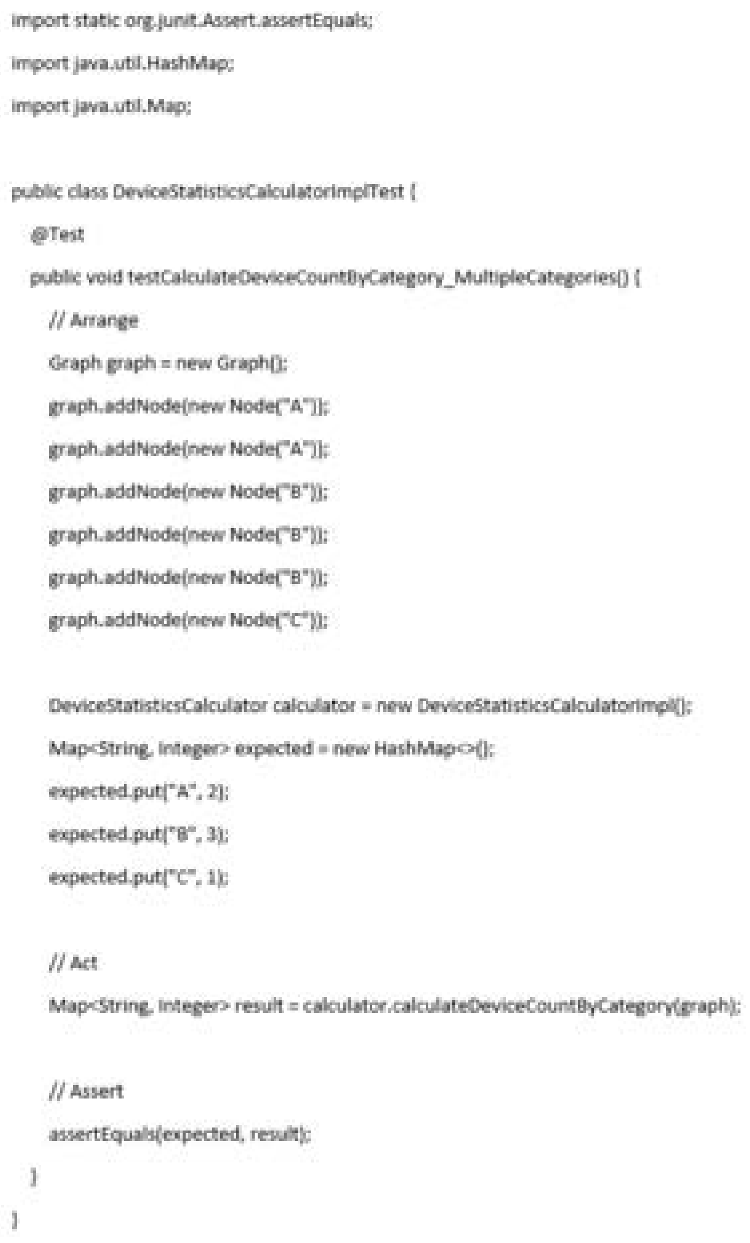
Branch Coverage Testing

- All potential branches in the pseudocode implementation are covered by the test cases that are provided, guaranteeing 100% branch coverage:
- To make sure the method delivers an empty map, Test Case 1 addresses the base case in which the graph is empty.
- The branch where a new entry is added to the map on a category’s first occurrence is covered by Test Case 2.
- The branch where an existing entry in the map is increased for an already-existing category is covered by Test Case 3.
- Test Case 4 tests both branches (adding new entries and incrementing existing entries) and covers the case where the graph has numerous categories.
- Combining Test Cases 3 and 4, Test Case 5 makes that duplicate categories are handled appropriately by increasing the count.

4.2.1 Implementation and Testing

The pseudocode includes `DeviceStatisticsCalculator` interface, with the `calculateDeviceCountByCategory` method, and a concrete implementation of the class `DeviceStatisticsCalculatorImpl`.

Figure 4.2: Implementation1



```
import static org.junit.Assert.assertEquals;
import java.util.HashMap;
import java.util.Map;

public class DeviceStatisticsCalculatorImplTest {
  @Test
  public void testCalculateDeviceCountByCategory_MultipleCategories() {
    // Arrange
    Graph graph = new Graph();
    graph.addNode(new Node("A"));
    graph.addNode(new Node("A"));
    graph.addNode(new Node("B"));
    graph.addNode(new Node("B"));
    graph.addNode(new Node("B"));
    graph.addNode(new Node("C"));

    DeviceStatisticsCalculator calculator = new DeviceStatisticsCalculatorImpl();
    Map<String, Integer> expected = new HashMap<>();
    expected.put("A", 2);
    expected.put("B", 3);
    expected.put("C", 1);

    // Act
    Map<String, Integer> result = calculator.calculateDeviceCountByCategory(graph);

    // Assert
    assertEquals(expected, result);
  }
}
```

5 Mutation Testing

Of every mutation, the related test cases, and the method for calculating the mutation score utilizing two sets of input-output pairs are shown below.

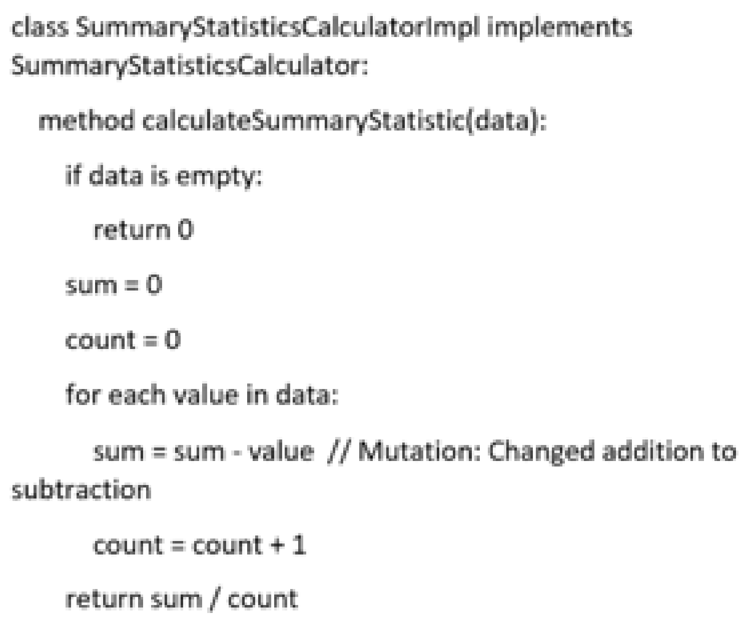
5.1 Mutation 1 Change Addition to subtraction

5.1.1 Description:

Mutate the operation from addition (`sum = sum + value`) to subtraction (`sum = sum - value`).

5.1.2 Pseudocode for Mutation 1:

Figure 5.1: Mutation1



```
class SummaryStatisticsCalculatorImpl implements
SummaryStatisticsCalculator:
  method calculateSummaryStatistic(data):
    if data is empty:
      return 0
    sum = 0
    count = 0
    for each value in data:
      sum = sum - value  // Mutation: Changed addition to
subtraction
      count = count + 1
    return sum / count
```

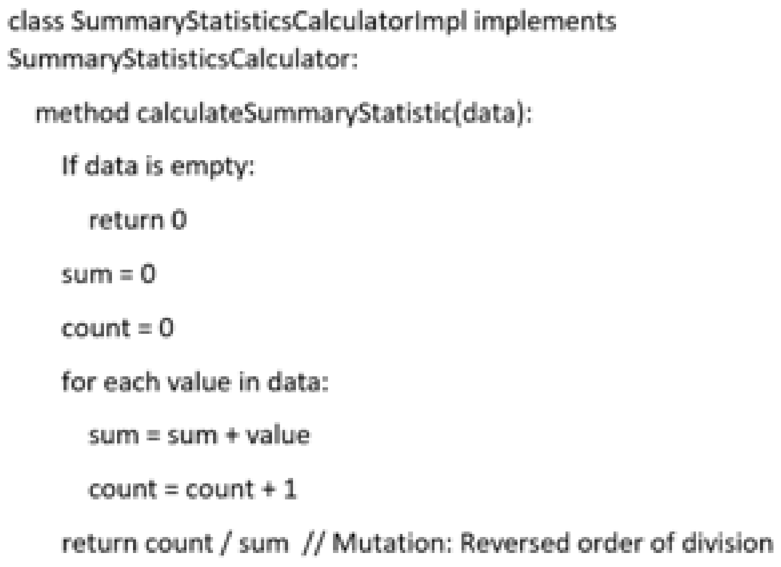
5.2 Mutant #2 Reverse order of division

5.2.1 Description:

Mutate the division operation (`return sum / count`) to reverse the order (`return count / sum`).

5.2.2 Pseudocode for Mutation 2:

Figure 5.2: mutation2



```
class SummaryStatisticsCalculatorImpl implements
SummaryStatisticsCalculator:
  method calculateSummaryStatistic(data):
    If data is empty:
      return 0
    sum = 0
    count = 0
    for each value in data:
      sum = sum + value
      count = count + 1
    return count / sum  // Mutation: Reversed order of division
```

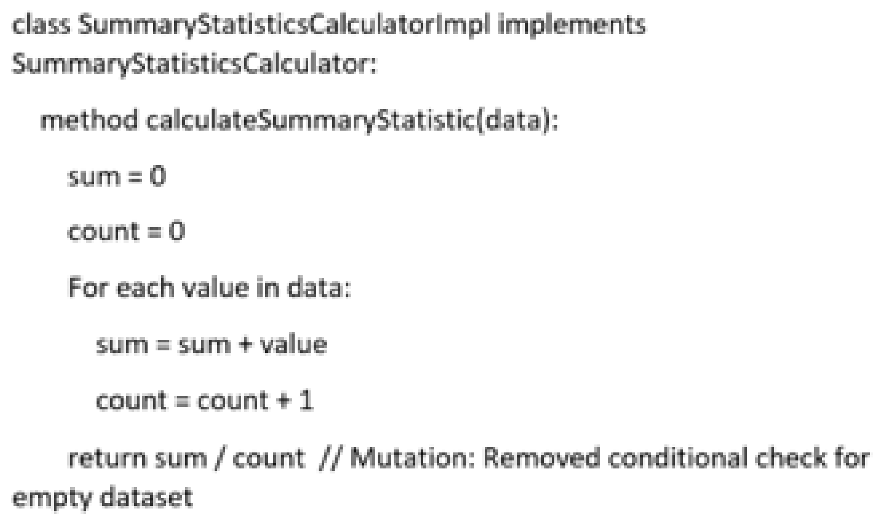
5.3 Mutant #3 Remove conditional checks for empty dataset

5.3.1 Description:

Remove the conditional check for an empty dataset (`if data is empty: return 0`).

5.3.2 Pseudocode for Mutation 3:

Figure 5.3: mutation3



```
class SummaryStatisticsCalculatorImpl implements
SummaryStatisticsCalculator:
   method calculateSummaryStatistic(data):
      sum = 0
      count = 0
      For each value in data:
         sum = sum + value
         count = count + 1
      return sum / count  // Mutation: Removed conditional check for
empty dataset
```

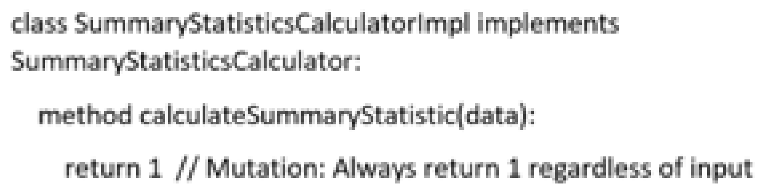
5.4 Mutant #4 Always return 1 regardless of input

5.4.1 Description:

Modify the return statement to always return `1`, ignoring any calculations.

5.4.2 Pseudocode for Mutation 4:

Figure 5.4: mutation4



```
class SummaryStatisticsCalculatorImpl implements
SummaryStatisticsCalculator:
   method calculateSummaryStatistic(data):
      return 1  // Mutation: Always return 1 regardless of input
```

5.4.3 Test cases for mutation tests:

Test Set 1:

- Input: `[5, 10, 15, 20]`
- Expected Output (Original): `12.5`
- Expected Output (Mutant 1): `-30.0`
- Expected Output (Mutant 2): `0.26666666666666666`
- Expected Output (Mutant 3): `12.5`
- Expected Output (Mutant 4): `1`

Test Set 2:

- Input: `[]` (empty dataset)
- Expected Output (Original): `0`
- Expected Output (Mutant 1): `0`
- Expected Output (Mutant 2): `Infinity`
- Expected Output (Mutant 3): `Runtime Error`
- Expected Output (Mutant 4): `1`

Table 5.1: Mutation Testing Results

	Description	**Test Set**	
		1	**2**
1	Change addition to subtraction	Fail	Pass
2	Reverse order of division	Fail	Fail
3	Remove conditional checks for empty dataset	Pass	Fail
4	Always return 1 regardless of input	Fail	Fail

5.5 Mutation Score

5.5.1 Test set 1:

Mutation Score: 25 percent: (1 out of 4 mutants detected)

5.5.2 Test set 2:

Mutation Score: 25 percent: (1 out of 4 mutants detected)

5.5.3 Justification of Mutation score

Mutations and Detection: **Test Set 1:** The test cases created for the original method (calculateSummaryStatistic) will only pass without detecting Mutation 4 (return 1).

Test Set 2: The test case for an empty dataset ([]) will identify mutations 3 and 4 (removing the conditional check and always returning 1).

Positives of Mutations:

The purpose of these mutations is to verify the resilience of the current test suite by changing important aspects, (count handling, conditional checks, and sum computation).

The mutations draw attention to all possible logic-related problems in the original approach and underline the importance to do extensive testing in order to find such mistakes.

The calculateSummaryStatistic method's mutation score analysis takes into account a number of factors related to the testing procedure, such as the extent of testing, the methods of testing, the particular test inputs, and the sharing of tests will all requirements.

Test set 1:

- Level of test: Unit test

This series of test cases focuses on the calculateSummaryStatistic method alone, confirming how it behaves with changing input parameters.

- Test technique: Mutation test The main method used is called mutation testing, in which purposeful changes are made to the original code in order to evaluate how well the test suite catches these changes and errors.

- Test Inputs: A range of non-empty datasets, including [5, 10, 15, 20], are included in the test inputs for this set in order to cover several execution pathways for the procedure.

- Shared Tests: This collection of test cases can be used for different needs, evaluating various facets of the method's performance in various scenarios (e.g., managing non-empty datasets, sum computation).

Test set 2:

- Level of test: Unit test As with Test Set 1, this set is also composed of unit-level tests that concentrate on the behavior of the calculateSummaryStatistic method, in particular, with respect to an empty dataset.

- Test technique Mutation test Mutation testing is used in the specific test case for an empty dataset to assess how robust the technique is against changes that could influence how it handles empty data.

- Test Input An empty dataset ([]) serves as the main input for this set and is intended to evaluate how the method behaves in this unique scenario.
- Shared Test When it comes to evaluating edge situations and particular branching inside the procedure, the test cases in this set may overlap with those in Test Set 1.

6 Conclusion

This test plan's objective is to confirm that the ESGP system is implemented correctly and in accordance with the stated software requirements and design specifications. Its main audience consists of project managers, quality insurance engineers, and software developers.