N8 SWALLEY 3/2/2022

**SE463 – Spring 2022**

**Assignment #2**

Equivalence Class Testing and Decision Table Testing

**Introduction**

This document outlines the analysis for Equivalence Class Testing and Decision Table Testing of a sorting system in C++. The sorting system is menu driven and consists of two sorting algorithms: bubble sort and insertion sort.

**Equivalence Class Testing**

Equivalence Class Testing is a black-box testing technique used in software testing that deals with partitioning the test input data into a number of different classes. The four different types of Equivalence Class Testing include Weak Normal, Strong Normal, Weak Robust, and Strong Robust. Weak Normal ECT focuses on one variable from each equivalence class and is also known as single fault assumption. Strong Normal ECT is known as multiple fault assumption, and it focuses on test cases from each element of the Cartesian product of the equivalence. Weak Robust ECT tests one variable from each equivalence class but is focused on invalid values. Strong Robust ECT produces test cases for all valid and invalid elements of the product of the equivalence class. Equivalence Class Testing is an effective form of software testing because it improves the quality of test cases by removing large amounts of redundancy and gaps.

**Equivalence Classes and Test Cases**

For this assignment, the equivalence classes include valid and invalid inputs. The user is prompted to enter input values in a menu system, “1” loads data values from a file, “2” exits the program. After the first input, the user is prompted again to enter either “1” for insertion sort, “2” for bubble sort, and “3” to exit the program. The following section illustrates the different test cases in the menu system for each of the different types of equivalent class testing. The range of valid values for the first user input is 1-2. The range of valid values for the second user input is 1-3. The data file is either valid or invalid, with valid being a file of integers separated by a comma.

**Weak Normal Equivalence Class Testing**

In Weak Normal ECT, we use a single-fault assumption. Weak Normal is concerned with valid inputs only and assumes a fault in one of the variable inputs. This variable is outlined in red.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cases: | userInput1 | userInput2 | dataFile | Outcome |
| 1 | 1 | 1 | ints | Valid |
| 2 | 1 | 2 | ints | Valid |
| 3 | 1 | 1 | ints | Valid |

Text

Description automatically generated

**Text

Description automatically generated**

In Weak Normal Equivalence Class Testing, all expected outcomes were reported successfully.

**Strong Normal Equivalence Class Testing**

In Strong Normal ECT, we use a multiple-fault assumption. In Strong Normal we select test cases from each element of the Cartesian product of the equivalence. This ensures the notion of completeness as it offers possible combinations of inputs. The cases in blue represent the different cases actually tested in the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cases: | userInput1 | userInput2 | dataFile | Outcome |
| 1 | 1 | 1 | ints | Valid |
| 2 | 2 | 1 | ints | Valid |
| 3 | 1 | 2 | ints | Valid |
| 4 | 1 | 3 | ints | Valid |

**A screenshot of a computer

Description automatically generated with low confidence**

Text

Description automatically generated

**Weak Robust Equivalence Class Testing**

Weak Robust ECT is very similar to Weak Normal ECT except now we are concerned with invalid inputs. We test one invalid variable from each equivalence class. This variable is outlined in red.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cases: | userInput1 | userInput2 | dataFile | Outcome |
| 1 | 0 | 1 | ints | Invalid |
| 2 | 1 | 4 | ints | Invalid |
| 3 | 1 | 1 | strings | Invalid |

Text

Description automatically generated

Text

Description automatically generated

\*invalid input file

In Weak Robust Equivalence Class Testing, all expected outcomes were reported successfully. The sorting menu system handles invalid inputs and prompts the user to re-enter a value but does not handle an invalid data file.

**Strong Robust Equivalence Class Testing**

In Strong Robust ECT, we produce test cases for all valid and invalid elements of the product of the equivalence class. However, it is incapable of reducing the redundancy in testing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cases: | userInput1 | userInput2 | dataFile | Outcome |
| 1 | 1 | 1 | ints | Valid |
| 2 | 1 | 2 | ints | Valid |
| 3 | 1 | 3 | ints | Valid |
| 4 | 2 | 1 | ints | Valid |
| 5 | 2 | 2 | ints | Valid |
| 6 | 2 | 3 | ints | Valid |
| 7 | 0 | 1 | ints | Invalid |
| 8 | 0 | 4 | ints | Invalid |
| 9 | 0 | 4 | strings | Invalid |
| 10 | 1 | 4 | ints | Invalid |
| 11 | 1 | 2 | strings | Invalid |
| 12 | 0 | 2 | strings | Invalid |
| 13 | 1 | 4 | strings | Invalid |
| 14 | 0 | 4 | strings | Invalid |

In Strong Robust Equivalence Class Testing, all expected outcomes were reported successfully.

**Decision Table Testing**

If the user provides a valid data file and valid menu selections, the program will execute successfully. If any of the input is invalid, the user will either be prompted to enter a new value, or an error message will possibly be displayed. Decision Table testing is used to test the system behavior for different input combinations. In this specific sorting system, we could use either limited entry decision tables or extended entry decision tables, depending on how you look at the equivalence classes. I decided to use a Limited Entry Decision Table because the equivalence classes are invalid and valid values. The following section illustrates Decision Table Testing for the sorting system.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Conditions | Set #1 | Set #2 | Set #3 | Set #4 | Set #5 | Set #6 | Set#7 |
| UserInput1 | Invalid | Invalid | Invalid | Valid | Valid | Valid | Valid |
| UserInput2 | Invalid | Valid | Invalid | Invalid | Valid | Invalid | Valid |
| DataFile | Invalid | Invalid | Valid | Invalid | Invalid | Valid | Valid |
| Output | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | SUCCESS |