Software Testing Report

NSW Traffic Penalty Tool

Brianne Byer s5175100

Wonwoo Choi s5145987

Marco Querzola s5264979

Table of Contents

[1.0 Unit Tests 3](#_Toc116207242)

[2.0 Coverage Report (HTML) 5](#_Toc116207243)

[3.0 Requirements Acceptance Testing 5](#_Toc116207244)

# Unit Tests

Software testing is an important stage in any software project. By making software fail, bugs are easier to identify and allow failures to be fixed. In the NSW Traffic Penalty Data Tool (NTPT) project, the waterfall model has been applied. The waterfall model means that testing is performed at the end and was used as a buffer to compensate for unexpected project delay. Each component of NTPT has been tested individually, this is known as unit testing (white-box style). Table 1 displays the NTPT unit tests, allowing stakeholders to view the overall quality of the software program.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Expected Results | Actual Results |
| **1.0** | **Check that text file exists** | | |
| 1.1 | Test file existence | Print “data.csv exists”. | Prints “data.csv exists” on console. |
| 1.2 | Test wrong file name | Print “file not found or file name error”. Stops code and exits. | Prints “file not found or file name error”. Stops code and exits. |
| 1.2 | Check | Error message appears. | Print “file not found or file name error”. Stops code and exits. |
| 2.0 | **Check Start Month** | | |
| 2.1 | Testing that start\_month can be converted into integer. | Print “Start Month is correct type” on console. | Prints “Start Month is correct type” on console. |
| 2.2 | Put string value(‘a’) to start\_month and run. | Print “Start Month is wrong type” on console. | Prints “Start Month is wrong type” on console. |
| **3.0** | **Check Start Year** | | |
| 3.1 | Testing that start year can be converted into integer. | Print “Start year is correct type” on console. | Prints “Start year is correct type” on console. |
| 3.2 | Put string value(‘a’) to start year and run. | Print “Start year is wrong type” on console. | Prints “Start year is wrong type” on console. |
| **4.0** | **Check End Month** | | |
| 4.1 | Testing that end month can be converted into integer. | Print “end month is correct type” on console. | Prints “end month is correct type” on console. |
| 4.2 | Put string value(‘a’) to end month and run | Print “end month is wrong type” on console. | Prints “end month is wrong type” on console. |
| **5.0** | **Check End Year** | | |
| 5.1 | Testing that end year can be converted into integer. | Print “end year is correct type” on console. | Prints “end year is correct type” on console. |
| 5.2 | Put string value(‘a’) to end year and run. | Print “end year is wrong type” on console. | Prints “end year is wrong type” on console. |
| **6.0** | **Check that school\_zone\_bool is a Boolean** | | |
| 6.1 | Use isinstance function to check whether the school\_zone\_bool is a bool type. | Print “school\_zone\_bool is correct type” on console. | Prints “school\_zone\_bool is correct type” on console. |
| 6.2 | Put “True” in school\_zone\_bool. | Print “school\_zone\_bool is not correct type” on console. | Prints “school\_zone\_bool is not correct type” on console. |
| **7.0** | **Check that the range\_date\_date\_format is a date type by using isinstance** | | |
| 7.1 | Use isinstance function to check whether the range\_date\_date\_format is a bool type. | Print “range\_date\_date\_format is correct type” on console. | Prints “range\_date\_date\_format is correct type” on console. |
| 7.2 | Insert “2012-03-01” to range\_date\_date\_format. | Print “range\_date\_date\_format doesn't convert into date format” on console. | Prints “range\_date\_date\_format doesn't convert into date format” on console. |

Table - Testing

# Coverage Report (HTML)

The idea of code coverage is to create a list of tasks and check that each task is covered during the testing process. Functions, statements, branches, and conditions must be evaluated. Function coverage refers to how many functions defined in code have been called. Statement coverage refers to how many statements have been executed. Branches coverage refers to how many branches of the control structures have been executed. Condition coverage refers to how many of the Boolean sub-expressions have been tested (True or False value).

As shown in Table 2, NTPT has a test coverage of 55.4%. The higher the code coverage, the higher the defect detection. While it may be assumed that coverage should be 100%, to do so is costly and does not mean the system is defect free. Furthermore, 100% coverage may not provide much benefit, as the domain of possible inputs for a program can be too large to be used in a testing system. The coverage of NTPT is measured by the percentage of program statements executed by the test suite.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Statements | Missing | Excluded | Coverage |
| draw\_graph\_all.py | 55 | 39 | 0 | 58.5% |
| oop\_main.py | 146 | 123 | 0 | 54.3% |
| Total | **201** | **162** | **36** | **55.4%** |

Table 2 - Test Coverage

# Requirements Acceptance Testing

(You will need to fill out the column on the left with the requirements listed in software design documents and the columns on the right with the results of your own testing)

| **Software  Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
| 1 | Accept multiple file names as arguments from the command line |  |  |  |
| 2 | Display the details of all valid files |  |  |  |
| 3 | Display an appropriate message if a file does not exist or if a file name is invalid |  |  |  |
| 4 | Display a message if an argument is a directory instead of a file |  |  |  |
| 5 | File name can be a simple file name or include the full path of the file with one or more levels |  |  |  |
| 6 | file names must start with an alphabetical character |  |  |  |
| 7 | Valid file name extensions must be 3 or 4 alphabetical characters preceded by a dot) |  |  |  |
| 8 | Directory/level names must start with an alphabetical character to be considered valid |  |  |  |
| 9 | The program should be able to accept as many levels for each file name as the user wants to input. This is limited only by the number of levels allowed in Windows (approximately 120) |  |  |  |