**Software Test Plan**

# **1.0 Test Plan Identifier**

Turing Machine (TM) in C# 1.3.

1.11 Team 2 (Lansdon Page, Jason Wong, Ryan Wilson, Jason Stidham)

# 

**Table of Contents**

**2.0 Testing Approach** 3

2.1 Unit Testing and Coverage test Strategy 3

2.2 Combinational Test Strategy 3

2.3 UML Test Strategy 3

**3.0 Unit Test Description and Implementation** 4

3.1 Unit Test Cases 5

3.2 Unit Test Results 10

3.3 Unit Test Coverage 12

**4.0 Combinational Tests** 14

4.1 Combinational Test Models 15

4.2 Combinational Test Case 15

4.3 Combinational Test Results 15

**5.0 UML Diagram** 16

5.1 UML Test Case 17

# 

# **2.0 Testing Approach**

**2.1 Unit Testing and Coverage**

We will be performing a series of unit tests using NUnit test suite verifying validity of the Parse function of the Turing Machine. In addition code coverage will identify the percentage of the application our tests will address.

**2.2 Combinational Testing**

We will be performing combinational testing on the run command using a black box approach. The Turing Machine has three variables we are able to manipulate using a manual approach. s(E)t allows changing the number of transitions per run, (T)runcate allows changing the output formatting, and we can change the input string when calling (R)un to be valid/invalid. In the combinational test table, figure 1, the values T and F represent True and False respectively. True represents a valid entry from the command line/menus for the corresponding variable. Likewise False refers to an invalid entry made from the menu system.

**2.3 UML Test Cases**

We are identifying test cases for the Turing machine from the UML Diagram; see figure 2, and using the aggregations from each of the necessary Turing machine objects as focal points for our tests.

# 

# **3.0 Unit Test**

The following pages detail the unit tests run, the results we derived from those unit tests, and the coverage that those tests achieved.

The implementation for our unit tests are located on our Github page at <https://github.com/lansdon/cpts422>.

**3.1 Unit Tests Cases**

|  |  |
| --- | --- |
| **Test ID** | 3.1.1 - ParseDefinition\_CheckForDuplicateStates |
| **Test Description** | Verify Conformance of Requirement 4.2.1.4a of Requirements Document that State names must be unique. This is accomplished by feeding a definition file with two duplicate states to the load method of the States Class, and verifying a false boolean is returned. |
| **Component** | Type: Class method  Name: States.load()  bool States.load( List<string> definition\_file ) |
| **Input Condition** | List<string> invalid\_definition = {“S1 S1”, “INPUT\_ALPHABET:”}  States test\_state = new States() |
| **Input State** | Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:” |
| **Expected Results** | A false boolean should be returned from the load class method of the States class. |

|  |  |
| --- | --- |
| **Test ID** | 3.1.2 - ParseDefinition\_CheckCaseSensitivityForStates |
| **Test Description** | Verify Conformance of Requirement 4.2.1.4b of Requirements Document that State names must be case sensitive. This is accomplished by feeding a definition file with two similarly named, but differingly cased states to the load method of the States class, and verifying that case is maintained by States class. |
| **Component** | Type: Class method  Name: States.load()  bool States.load( List<string> definition\_file ) |
| **Input Condition** | List<string> invalid\_definition = {“state1 STATE1”, “INPUT\_ALPHABET:”}  States test\_state = new States() |
| **Input State** | Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:” |
| **Expected Results** | State class should contain two states, and maintain case sensitivity. |

**3.1 Unit Test Cases (Continued)**

|  |  |
| --- | --- |
| **Test ID** | 3.1.3 - ParseDefinition\_CheckForAtLeastOneState |
| **Test Description** | Verify Conformance of Requirement 4.2.1.2 of Requirements Document that there must be at least one state. This is accomplished by feeding a definition file with no states, and verifying that an exception was thrown. |
| **Component** | Type: Class method  Name: States.load()  bool States.load( List<string> definition\_file ) |
| **Input Condition** | List<string> invalid\_definition = (“”, “INPUT\_ALPHABET:”)  States test\_state = new States() |
| **Input State** | Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:” |
| **Expected Results** | A false boolean should be returned from the load class method of the States class. |

|  |  |
| --- | --- |
| **Test ID** | 3.1.4 - ParseDefinition\_CheckForValidStateCharacters |
| **Test Description** | Verify Conformance of Requirement 4.2.1.4c of Requirements Document that only alphanumeric characters and the underscore character are allowed to be used in the naming of states. This is accomplished by feeding an invalid definition file using invalid characters, and verifying that an exception was thrown. |
| **Component** | Type: Class method  Name: States.load()  bool States.load( List<string> definition\_file) |
| **Input Condition** | List<string> invalid\_definition = (“$”, “INPUT\_ALPHABET:”)  States test\_state = new States() |
| **Input State** | Turing Machine program has just finished processing both the Turing Machine description and the keyword “STATES:” |
| **Expected Results** | A false boolean should be returned from the load method of the States class. |

**3.1 Unit Test Cases (Continued)**

|  |  |
| --- | --- |
| **Test ID** | 3.1.5 - ParseDefinition\_CheckThatElementsAreLengthOne |
| **Test Description** | Verify Conformance of Requirement 4.2.2.2 of Requirements Document that input alphabet consists of elements of only length one. This is accomplished by feeding an invalid definition file that contains input alphabet elements of length greater than one. |
| **Component** | Type: Class Method  Name: InputAlphabet.load()  bool InputAlphabet.load( List<string> definition\_file ) |
| **Input Condition** | List<string> invalid\_definition = (“ab”, “TAPE\_ALPHABET:”)  InputAlphabet test\_inputalphabet = new InputAlphabet() |
| **Input State** | Turing Machine program has just finished processing both the Turing Machine description, the keyword “STATES:” and its elements, and the keyword “INPUT\_ALPHABET”. |
| **Expected Results** | A false boolean should be returned from the load class method of the InputAlphabet class. |

|  |  |
| --- | --- |
| **Test ID** | 3.1.6 - ParseDefinition\_CheckForDuplicateInputAlphabetCharacters |
| **Test Description** | Verify Conformance of Requirement 4.2.2.3 of Requirements Document that input alphabet consists of unique elements. This is accomplished by feeding an invalid definition file that contains duplicate input alphabet elements. |
| **Component** | Type: Class Method  Name: InputAlphabet.load()  bool InputAlphabet.load( List<string> definition\_file ) |
| **Input Condition** | List<string> invalid\_definition = (“ab”, “TAPE\_ALPHABET:”)  InputAlphabet test\_inputalphabet = new InputAlphabet() |
| **Input State** | Turing Machine program has just finished processing both the Turing Machine description, the keyword “STATES:” and its elements, and the keyword “INPUT\_ALPHABET”. |
| **Expected Results** | A false boolean should be returned from the load class method of the InputAlphabet class. |

**3.1 Unit Test Cases (Continued)**

|  |  |
| --- | --- |
| **Test ID** | 3.1.7 - ParseDefinition\_TransFunct\_Valid |
| **Test Description** | Test that a transition function containing proper 5 components is parsed properly. |
| **Component** | transition\_function |
| **Input Condition** | Definition input containing a transition function with 5 valid fields. |
| **Input State** | “TRANSITION\_FUNCTION:” keyword has already been parsed. |
| **Expected Results** | all 5 fields are correctly parsed and stored in transition\_function class. |

|  |  |
| --- | --- |
| **Test ID** | 3.1.8 - ParseDefinition\_TransFunct\_InvalidFieldCount |
| **Test Description** | Test that a transition function NOT containing proper 5 components is produces an error. |
| **Component** | transition\_function |
| **Input Condition** | Definition input containing a transition function without 5 valid fields. |
| **Input State** | “TRANSITION\_FUNCTION:” keyword has already been parsed. |
| **Expected Results** | Should produce an error and return false. |

|  |  |
| --- | --- |
| **Test ID** | 3.1.9 - ParseDefinition\_TransFunct\_InvalidChar |
| **Test Description** | Verify that accepted transitions have valid characters from the tape alphabet. |
| **Component** | transition\_function |
| **Input Condition** | Definition containing a state that doesn’t exist in the tape alphabet |
| **Input State** | “TRANSITION\_FUNCTION:” keyword has already been parsed. |
| **Expected Results** | Should have error |

|  |  |
| --- | --- |
| **Test ID** | 3.1.10 - ParseDefinition\_InitState\_TooManyStates |
| **Test Description** | Validate only 1 initial state is accepted |
| **Component** | turing\_machine |
| **Input Condition** | Give two initial states. |
| **Input State** | TRANSITION\_FUNCTION: already parsed |
| **Expected Results** | Should have an error due to too many initial states. |

**3.1 Unit Test Cases (Continued)**

|  |  |
| --- | --- |
| **Test ID** | 3.1.11 - ParseDefinition\_BlankChar\_NotInAlphabet |
| **Test Description** | Blank char must be member of tape\_alphabet. |
| **Component** | turing\_machine |
| **Input Condition** | blank char used that is not part of tape alphabet. |
| **Input State** | INITIAL\_STATE: keyword has been parsed. |
| **Expected Results** | Should have an error. |

|  |  |
| --- | --- |
| **Test ID** | 3.1.15 - ParseDefinition\_FinalStates\_NoStates |
| **Test Description** | Final states must have at least one final state. |
| **Component** | final\_states |
| **Input Condition** | Pass 0 final states |
| **Input State** | INPUT\_STATE: keyword has already been parsed. |
| **Expected Results** | Must have one or more final states |

|  |  |
| --- | --- |
| **Test ID** | 3.1.16 - ParseDefinition\_FinalStates\_NotInStates |
| **Test Description** | Final State must be a member of states |
| **Component** | final\_states |
| **Input Condition** | final\_state defined that is not in states |
| **Input State** | FINAL\_STATES: already parsed. |
| **Expected Results** | Should have error due to invalid final state. |

**3.2 Unit Test Results**

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.1 - ParseDefinition\_CheckForDuplicateStates |
| **Result** | Test Failed - False was returned by method. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.2 - ParseDefinition\_CheckCaseSensitivityForStates |
| **Result** | Test Passed - Indicating that the State class does maintain case. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.3 - ParseDefinition\_CheckForAtLeastOneState |
| **Result** | Test Failed - False was returned by the method. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.4 - ParseDefinition\_CheckForValidStateCharacters |
| **Result** | Test Failed - False was returned by the method. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.5 - ParseDefinition\_CheckThatElementsAreLengthOne |
| **Result** | Test Failed - False was returned by the method. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.6 - ParseDefinition\_CheckForDuplicateInputAlphabetCharacters |
| **Result** | Test Failed - Class method failed to detect duplicate Input Alphabet characters. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.10 - ParseDefinition\_TransFunct\_Valid |
| **Result** | PASS |

**3.2 Unit Test Results (Continued)**

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.11 - ParseDefinition\_TransFunct\_InvalidFieldCount |
| **Result** | FAILED - Invalid transition accepted. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.12 - ParseDefinition\_TransFunct\_InvalidChar |
| **Result** | FAILED - Invalid char accepted. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.13 - ParseDefinition\_InitState\_TooManyStates |
| **Result** | PASS |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.14 - ParseDefinition\_BlankChar\_NotInAlphabet |
| **Result** | FAIL - Invalid blank char accepted. |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.15 - ParseDefinition\_FinalStates\_NoStates |
| **Result** | PASS |

|  |  |
| --- | --- |
| **Unique Test Identifier** | 3.1.16 - ParseDefinition\_FinalStates\_NotInStates |
| **Result** | FAIL - Invalid final state accepted. |

**3.3 Unit Test Coverage**

Using NUnit as our testing frame work we were able to provide 34.68% code coverage for the Turing Machine application. The chart below provides detailed information on the methods covered.



# 

**3.3 Unit Test Coverage (Continued)**



# **4.0 Combinational Tests**

For our combinational testing, it was decided that the decision table would be the most appropriate test model because it served as our list of test cases as well as our test results. It allowed for a clear representation of the variables to be tested against, the inputs used for said variables, the results we expected, and the actual results we received –all in one concise table.

**4.1 Combinational Test Models**

Combination tests will be ran using a manual black box approach. The Turing Machine has two variables we are allowed to manipulate using a manual approach, s(E)t and (T)runcate.

**4.2 Combinational Test Case**

Reference columns Run/Expected in figure one for a description of the test to be conducted for the corresponding TestID.

**4.3 Combinational Test Results**

Reference column Result in figure one for a description of the test to be conducted for the corresponding TestID.



Figure 1

# 

# **5.0 UML Diagram**

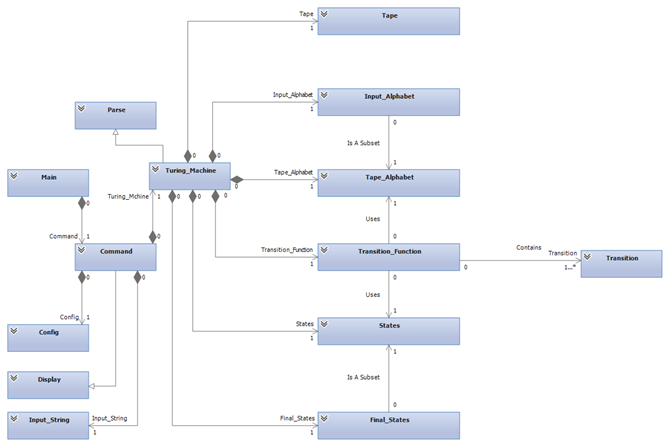


Figure 2

# 

**5.1 UML Test Case**

|  |  |
| --- | --- |
| Test ID: | UML tc1 - Check for Tape Object |
| Description: | Verifies that a Tape object is instantiated for the Turing Machine. |
| Expected Result: | Upon instantiation of the Turing machine the object Tape will be generated. |

|  |  |
| --- | --- |
| Test ID: | UML tc2 - Check for Input Alphabet Object |
| Description: | Verifies that a Input\_Aphabet is instantiated for the Turing Machine. |
| Expected Result: | Upon instantiation of the Turing machine the object Input\_Alphabet will be generated. |

|  |  |
| --- | --- |
| Test ID: | UML tc3 - Check for Tape Alphabet Object |
| Description: | Verifies that a Tape\_Alphabet object is instantiated for the Turing Machine. |
| Expected Result: | Upon instantiation of the Turing machine the object Tape\_Alphabet will be generated. |

|  |  |
| --- | --- |
| Test ID: | UML tc4 - Check for Final States Object |
| Description: | Verifies that a Transition\_Function object is instantiated for the Turing Machine. |
| Expected Result: | Upon instantiation of the Turing machine the object Transition\_Function will be generated. |

|  |  |
| --- | --- |
| Test ID: | UML tc5 - Check for States Object Instantiation |
| Description: | Verifies that a States object is instantiated the Turing Machine. |
| Expected Result: | Upon instantiation of the Turing machine, the object States will be generated. |

|  |  |
| --- | --- |
| Test ID: | UML tc6 - Check for Final States Object |
| Description: | Verifies that the Final\_States object is an instance of the Turing Machine. |
| Expected Result: | Upon instantiation of the Turing machine the objectFinal\_ States will be generated. |

**5.1 UML Test Case (continued)**

|  |  |
| --- | --- |
| Test ID: | UML tc7 - Check for Tape Alphabet Subset |
| Description: | Verifies that the Transition object is an instance of the Turning Machine. |
| Expected Result: | Upon instantiation of the Turing machine the object Transition will be generated. |

|  |  |
| --- | --- |
| Test ID: | UML tc8 - Check the Tape Alphabet Subset |
| Description: | Verifies that Input\_alphabet is a subset of tape\_alphabet |
| Expected Result: | The content of the input alphabet is infact a subset of the tape alphabet. |

|  |  |
| --- | --- |
| Test ID: | UML tc9 - Check the Final States Subset |
| Description: | Verifies that each final state is an element of the set of States. |
| Expected Result: | All elements of the set Final States are also elements of the set States. |