SE 3XA3: Test Report
Finite State Machine Simulator

Team #16, NonDeterministic Key
Anhao Jiao (jiaoa3)
Kehao Huang (huangk53)
Xunzhou Ye (yex33)

April 12, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
12 April 2022	1.0	Initial Draft
12 April 2022	1.1	Revision

1 Functional Requirements Evaluation

1. Test Name: UIT-1

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Modification of transitions between states in an existing FSM

   Result: The user is able to able to define a valid FSM

2. Test Name: UIT-2

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Modification of transitions between states in an existing FSM

   Result: The user is able to modify transition in an existing FSM

3. Test Name: UIT-3

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Modification of states in an existing FSM

   Result: The user is able to modify states in an existing FSM

4. Test Name: UIT-4

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: The specified transitions

   Result: The user is able to switch states with a valid transition

5. Test Name: UIT-5

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

Input: a non-existing trigger event

Result: The system is able to notify the user after an invalid input

6. Test Name: OT-1

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Require current state

   Result: The user is able to get the current state of the FSM

7. Test Name: OT-2

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: A valid FSM

   Result: The user is able to get the LaTeX snippets of the FSM

8. Test Name: OT-3

   Type: Functional, Dynamic, Manual

   State: FSM simulator that is used to define a FSM

   Input: A valid FSM

   Result: The user is able to get a .png image representing the pre-defined FSM

9. Test Name: OT-4

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: A valid FSM with only states.

   Result: The user is able to get a .png image representing the pre-defined FSM with no transition

2 Nonfunctional Requirements Evaluation

2.1 Performance

1. Test Name: UT-1

   Type: Functional, Dynamic, Manual

   Initial State: a well-defined FSM with states, initial state, transitions

   Input/Condition: user asked the program to export the defined FSM into LaTeXsnippets

   Result: The image rendered using the exported snippet is visually appealing, with minimal overlapping of arrows, circles, or other visualization elements.

2. Test Name: POC-1

   Type: Functional, Dynamic, Manual

   Initial State: user have defined a FSM

   Input: FSM with 5 states and 4 transitions

   Result: The LaTeX snippets can be compiled in a .tex file. The output image is as described as the FSM defined by the user.

2.2 Usability

1. Test Name: UT-2

   Type: Non-Functional, Dynamic, Manual

   Initial State: a computer with only Python preinstalled

   Input: user asked to install the program without assistance

   Result: The majority of users(over 90%) are able to install the program independently within 5 minutes.

2. Test Name: UT-3

Type: Non-Functional, Dynamic, Manual

Initial State: a computer with the program installed

Input: user asked to incorporate the program in their academic work flow

Result: The majority of users(over 90%) are benefited from the use of our tool

2.3 Robustness

1. Test Name: POC-4_1

   Type: Functional, Dynamic, Manual

   Initial State: user have defined a FSM

   Input: FSM with 100 states and 400 transitions

   Result: A FSM with 100 states and 400 transitions can be defined by the software.

2. Test Name: POC-4_2

   Type: Functional, Dynamic, Manual

   Initial State: user have defined a FSM

   Input: FSM with 100 states and 400 transitions

   Result: The system is able to generate the corresponding LaTeX snippets of a FSM with 100 states and 400.

2.4 Maintainability and Support

1. Test Name: MS-1

   Type: Non-Functional, Dynamic, Manual

   Initial State: a computer with the program installed

   Input: update the software with new features

   Result: The overall maintenance is finished within 4 hours

2. Test Name: MS-2

   Type: Non-Functional, Dynamic, Manual

   Initial State: a computer with the program installed

   Input: Install and use the software

   Result: 98% of students are able to run the system on their pcs or laptops.

3 Comparison to Existing Implementation

N/A

4 Unit Testing

Each method in the software is considered as a unit. For each method, at least one test case that execute the method was conducted and passed.

5 Changes Due to Testing

6 Automated Testing

N/A

7 Trace to Requirements

Test Case	Requirement
UIT-1	FR1
UIT-2	FR2
UIT-3	FR3
UIT-4	FR4
UIT-5	FR4
OT-1	FR5
OT-2	FR6
OT-3	FR5
OT-4	FR5
UT-1	NF1–2
UT-2	NF3, NF10, NF13
UT-3	NF2, NF4–7, NF9, NF11–12
MS-1	NF12
MS-2	NF13

8 Trace to Modules

Test Case	Modules
UIT-1	M2-M8
UIT-2	M2-M8
UIT-3	M2-M8
UIT-4	M2-M8
UIT-5	M2-M8
OT-1	M2-M8
OT-2	M2-M8
OT-3	M2-M8
OT-4	M2-M8
UT-1	M1
UT-2	M1-M8
UT-3	M1-M8
MS-1	M1
MS-2	M1

9 Code Coverage Metrics

After applying all the tests we designed in the test plan document, we managed to achieved a 98% statement coverage of entire project. To achieve the code coverage goal, we conducted unit testing with pytest together with manual testing with visual inspection. This number is supported by the fact that all the modules are covered in the testing phase and are tested multiple times.