Senior Design Test Plan

Improved Visualization for Formal Languages

Group Members:

Chris Pinto-Font (cpintofont2021@my.fit.edu)

Vincent Borrelli (vborrelli2022@my.fit.edu)

Andrew Bastien (abastien2021@my.fit.edu)

Keegan McNear (kmcnear2022@my.fit.edu)

Faculty Advisor:

Dr. Luginbuhl (dluginbuhl@fit.edu)

Client:

Dr. Luginbuhl

Florida Institute of Technology

September 4, 2025

Table of Contents

1. Introduction

1.1 Purpose

This Test Plan describes the strategy and test cases for verifying the **Improved Visualization for Formal Languages** application. The goal is to validate that the system reliably enables users to create, animate, minimize, learn about, and interact with DFAs as specified in the Requirements Document.

1.2 Scope

Testing will encompass the functional, performance, and non-functional requirements of the application. Focus areas include DFA creation, animation, minimization, multi-symbol transitions, and overall usability. The scope of our tests will also allow the DFA logic to be as accurate as possible.

1.3 References

- IEEE Std 829-1998, Software Test Documentation
- Improved Visualization for Formal Languages Requirements Document (v1.0)

2. Test Items

- DFA visualization desktop application
- Core modules for DFA creation, animation, and minimization
- GUI components and built-in documentation

3. Features to be Tested

- DFA creation and editing ([FR-1])
- String execution/animation ([FR-2])
- State designation ([FR-3])
- DFA minimization and completion ([FR-4], [FR-5])
- Multi-symbol transitions ([FR-6])
- In-program documentation ([FR-7])
- Performance under large DFA load ([PR-1], [PR-2])
- Security, reliability, and usability ([NFR-1]–[NFR-6])

4. Features Not to be Tested

- Online collaboration/sharing features (not in current release)
- Integration with external DFA libraries

5. Test Approach

A mix of black-box testing, usability testing, and stress testing will be employed. Test data will include both normal inputs (valid DFA definitions, strings) and unusual inputs (invalid transitions, incomplete DFAs) to uncover edge cases.

6. Test Cases

6.1 Functional Requirements

- **[FR-1] Test Case – DFA Creation**
  Input: User creates a DFA graphically in the editor
  Method: Black-box testing of GUI drawing and backend storage
  Usual Output: States and transitions appear correctly and can be saved
  Unusual Output: Invalid state names trigger error messages

- **[FR-2] Test Case – DFA Execution Animation**
  Input: Enter string to run on a DFA
  Method: Observe animation step-by-step
  Usual Output: Animation highlights current state and transitions correctly
  Unusual Output: If string invalid, system stops and shows error

- **[FR-3] Test Case – State Designation**
  Input: Mark states as initial/final/dead
  Method: GUI toggle test
  Usual Output: States visually update (icons/labels)
  Unusual Output: Conflicting designations prevented by warning

- **[FR-4] Test Case – DFA Minimization**
  Input: Provide DFA with redundant states
  Method: Run minimization algorithm and compare to expected minimal DFA
  Usual Output: Output DFA has fewer states and is equivalent
  Unusual Output: Algorithm logs error if input not a valid DFA

- **[FR-5] Test Case – DFA Completion**
  Input: Provide incomplete DFA
  Method: Run “complete DFA” function
  Usual Output: Missing transitions auto-generated or user prompted
  Unusual Output: If completion impossible, clear warning displayed

- **[FR-6] Test Case – Multi-Symbol Transitions**
  Input: Add multiple symbols to one transition
  Method: GUI input test
  Usual Output: Transition label shows all symbols clearly
  Unusual Output: Overflow or unreadable text triggers formatting adjustment

- **[FR-7] Test Case – In-Program Documentation**
  Input: Click help icon
  Method: Usability check
  Usual Output: Documentation opens in-app without external files

Unusual Output: Missing topics logged for update

6.2 Performance Requirements

- **[PR-1] Test Case – Large DFA Load**
  Input: Load DFA with 100+ states
  Method: Time loading and rendering
  Usual Output: Diagram appears in ≤3 seconds
  Unusual Output: If slower, system displays “loading” indicator

- **[PR-2] Test Case – Animation Speed**
  Input: Run DFA execution with adjustable speed
  Method: Observe FPS and smoothness
  Usual Output: Animation runs smoothly at 30 FPS or more
  Unusual Output: If lag detected, logs performance metrics

6.3 Non-Functional Requirements

- **[NFR-1] Test Case – Reliability (Autosave)**
  Input: Edit DFA then force-close app
  Method: Reopen app and check recovery
  Usual Output: Last state auto-saved within 5 minutes
  Unusual Output: If lost, error reported in logs

- **[NFR-5] Test Case – Usability for New Users**
  Input: First-time user creates DFA without tutorial
  Method: Observe task completion time
  Usual Output: User completes simple DFA within 5 minutes
  Unusual Output: If >5 minutes, usability redesign required

7. Pass/Fail Criteria

- **Pass:** Test case meets requirement thresholds (time, correctness, usability)
- **Fail:** System does not meet requirements, crashes, or misbehaves with invalid input

8. Test Deliverables

- Test Plan
- Test Case Specifications
- Test Execution Report
- Bug Reports
- Final Test Summary

9. Testing Schedule

Each feature will follow: Initial Test → Fix bugs → Retest → Client review → Incorporate feedback.

10. Risks and Contingencies

- **Risk:** Performance issues with very large DFAs
  Mitigation: Optimize drawing algorithms, consider Cython/C++ offloading
- **Risk:** Complexity of implementing animations
  Mitigation: Build prototype animations early; use existing GUI libraries