# Verification and Validation Report: Software Engineering

Team 2, SyntaxSentinals
Mohammad Mohsin Khan
Lucas Chen
Dennis Fong
Julian Cecchini
Luigi Quattrociocchi

March 11, 2025

# 1 Revision History

| Date | Version | Notes |
|---|---|---|
| Date 1 | 1.0 | Notes |
| Date 2 | 1.1 | Notes |

# 2 Symbols, Abbreviations and Acronyms

| symbol | description |
|---|---|
| T | Test |

[symbols, abbreviations or acronyms – you can reference the SRS tables if needed —SS]

# Contents

# List of Tables



# List of Figures

# 3 Functional Requirements Evaluation

### 3.0.1 Plagiarism Analysis Input Upload Tests

Applies to all FRs involving uploading code snippets to the system. This refers to test-FR-1 from the VnV Plan.

1. **test-FR-1**

   Expected result: The script should run and successfully upload the files, without encountering any errors due to the upload functionality

   Actual Result: No errors from uplading

   Evaluation: Pass

### 3.0.2 Plagiarism Analysis Result Tests

Applies to all FRs involving immediate output of plagiarism analysis. This refer to test-FR-2 from the VnV Plan.

1. **test-FR-2**

   Expected result: The analysis should not throw any errors while the script runs, up until the analysis provides a response

   Actual result: No errors were observed during script execution

   Evaluation: Pass

### 3.0.3 Guide Documentation Generation Tests

Applies to all FRs involving guide documentation generation for user. This refers to test-FR-3 from the VnV plan.

1. **test-FR-3**

   Expected result: Generation of documentation successful

   Actual result: TBD when feature is implemented

   Evaluation: TBD

### 3.0.4 Report Documentation Generation Tests

Applies to all FRs involving analysis report documentation generation for user. This refers to test-FR-4 from the VnV plan.

1. **test-FR-4**

   Expected result: Generation of report is successful, with no errors encountered

   Actual result: Report was successfully generated, no errors were found

   Evaluation: Pass

### 3.0.5 Account Creation Tests

Applies to all FRs involving creating an account within the system. This refers to test-FR-5 and test-FR-6 from the VnV plan

1. **test-FR-5**

   Expected result: An account should be successfully created, while the system avids any failure modes.

   Actual result: Account successfully created, and no failure modes were exhibited

   Evaluation: Pass

2. **test-FR-6**

   Expected result: Account creation should fail and the system will await further input from the user

   Actual result: Account creation fails, and the system does not proceed

   Evaluation: Pass

### 3.0.6 Account Login Tests

Applies to all FRs involving logging into account within the system. This refers to test-FR-7 and test-FR-8 in the VnV plan

1. **test-FR-7**

   Expected result: Logging into system with correct existing credentials should succeed, and no errors should arise from logging in

Actual result: Logging in with existing credentials succeeds with no errors

Evaluation: Pass

2. **test-FR-8**

   Expected result: Logging in with invalid credentials should fail

   Actual result: Logging in with invalid credentials fails

   Evaluation: Pass

#### 3.0.7 Result Email Tests

Applies to all FRs involving emailing results of plagiarism analysis to users. This refers to test-FR-9 from the VnV plan.

1. **test-FR-9**

   Expected result: The system will send an e-mail to address associated with the logged in account, containing a .zip file with plagiarism results corresponding to the files uploaded

   Actual result: Correct and corresponding .zip file was sent to the expected e-mail

   Evaluation: Pass

#### 3.0.8 Result Visualization Tests

Applies to all FRs involving visualizing plagiarism analysis results provided by user in a .zip file. This refers to test-FR-10 from the VnV plan.

1. **test-FR-10**

   Expected result: A visualization is successfully created after uplading a .zip file, with no errors observed during the process.

   Actual result: A visualization was successfully created with no errors

   Evaluation: Pass

# 4 Nonfunctional Requirements Evaluation

## 4.1 Usability

## 4.2 Performance

## 4.3 etc.

# 5 Comparison to Existing Implementation

## 5.1 Overview of Existing Solutions

### 5.1.1 MOSS (Measure of Software Similarity)

MOSS is currently the standard tool for detecting code plagiarism in academic settings. Developed at Stanford University, it works by:

- Tokenizing code into a sequence of symbols
- Using document fingerprinting to detect similar code segments
- Generating a similarity score based on matching sequences
- Providing a web interface to view overlapping code segments

### 5.1.2 Other Notable Solutions

- **JPlag**: Uses a token-based approach similar to MOSS, but with different tokenization strategies
- **PLAGGIE**: Java-specific plagiarism detection tool used in academic environments
- **CodeMatch**: Commercial solution that uses both tokenization and metric-based methods

## 5.2 Our Implementation vs. Existing Solutions

### 5.2.1 Current Approach

Our implementation leverages CodeBERT, a pre-trained model for programming language understanding, to perform code pair comparison. Key aspects include:

- Utilization of transformer-based neural networks to understand code semantics
- Direct comparison of code pairs to determine similarity
- A modern, user-friendly interface for result visualization and analysis

### 5.2.2 Functional Comparison

| **Feature** | **MOSS** | **Our CodeBERT Implementation** |
|---|---|---|
| Detection Method | Token-based fingerprinting | Neural representation and semantic understanding |
| Language Support | Multiple languages | Multiple languages (supported by CodeBERT) |
| Semantic Understanding | Limited (syntax-focused) | Enhanced (captures semantic meaning) |
| Resistance to Obfuscation | Low-moderate | Potentially higher |
| Speed | Fast for large submissions | Currently slower for large-scale comparisons |
| Visualization | Basic web interface | Modern, interactive UI |

Table 1: Comparison between MOSS and our CodeBERT implementation

## 5.3 Advantages of Our Approach

1. **Semantic Understanding**: Unlike MOSS's purely syntactic approach, CodeBERT can potentially understand the meaning behind code, making it more difficult to fool with non-functional additions.
2. **Context Awareness**: Our implementation considers the context in which code appears, potentially reducing false positives from common solutions to standard problems.
3. **Modern Interface**: Our user interface provides a more intuitive experience for instructors reviewing potential plagiarism cases.

## 5.4 Current Limitations and Trade-offs

1. **Computational Requirements**: Neural models like CodeBERT require more computational resources than traditional approaches like MOSS.

2. **Development Maturity**: MOSS has been refined over decades, while our solution is still in the early stages of development.

3. **Validation**: Our approach needs more extensive testing across different programming languages and assignments to prove its effectiveness.

4. **Explainability**: Neural network decisions can be less transparent than token-matching approaches, making it potentially harder to explain why certain code pairs were flagged.

## 5.5 Design Decision Justification

1. **Choice of CodeBERT**: We selected CodeBERT over other models because it was specifically pre-trained on code from multiple programming languages, making it well-suited for understanding code semantics across different languages used in academic settings.

2. **Focus on Pair Comparison**: While MOSS compares each submission against all others, our current approach focuses on direct pair comparison, allowing for more detailed analysis of potentially plagiarized code.

3. **UI Priority**: We invested in a high-quality UI early in development to facilitate easier testing and validation of our detection results.

## 5.6 Future Improvements

1. **Hybrid Approach**: Combining neural understanding with traditional token-based methods to leverage strengths of both approaches.

2. **Performance Optimization**: Improving the computational efficiency to handle large class submissions more effectively.

3. **Tunable Sensitivity**: Implementing adjustable thresholds for different assignment types and programming languages.

# 6 Unit Testing

## 6.1 AnalyzeFiles Component

### 6.1.1 Renders the component

- **Description**: Ensures the `AnalyzeFiles` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Check for the presence of specific text.
- **Expected result**: The component renders with the text "Analyze a Dataset" and "Upload a dataset to analyze for potential code plagiarism. Use a ZIP file containing your project files."

### 6.1.2 Displays an alert when no file is uploaded

- **Description**: Ensures an alert is displayed when no file is uploaded.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Click the "Analyze Dataset" button.
- **Expected result**: An alert with the message "Please upload a dataset file first." is displayed.

#### 6.1.3 Displays an alert when no analysis name is entered

- **Description**: Ensures an alert is displayed when no analysis name is entered.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Upload a valid ZIP file.
  3. Click the "Analyze Dataset" button.
- **Expected result**: An alert with the message "Please enter an analysis name." is displayed.

#### 6.1.4 Calls uploadFiles and displays success message when form is submitted

- **Description**: Ensures the `uploadFiles` function is called and a success message is displayed when the form is submitted.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Upload a valid ZIP file.
  3. Enter an analysis name.
  4. Click the "Analyze Dataset" button.
- **Expected result**: The `uploadFiles` function is called with the file and analysis name, and a success message "Analysis started successfully. You'll receive an email with the results once it's done!" is displayed.

#### 6.1.5 Handles file drop event

- **Description**: Ensures the file drop event is handled correctly when a file is dropped on the component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Simulate a file drop event with a valid ZIP file.
- **Expected result**: The file is uploaded successfully.

### 6.2 UploadResults Component

#### 6.2.1 Renders the component

- **Description**: Ensures the `UploadResults` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadResults` component.
  2. Check for the presence of specific text.
- **Expected result**: The component renders with the text "View Results" and "Upload your results ZIP file to view the analysis."

#### 6.2.2 Displays an error message when a non-ZIP file is uploaded

- **Description**: Ensures an error message is displayed when a non-ZIP file is uploaded.
- **Type**: Automatic, Functional

- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadResults` component.
  2. Upload a non-ZIP file.
- **Expected result**: An error message "Only ZIP files are allowed." is displayed.

#### 6.2.3 Displays an error message when no JSON file is found in the ZIP

- **Description**: Ensures an error message is displayed when no JSON file is found in the ZIP file.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadResults` component.
  2. Upload a ZIP file without a JSON file.
- **Expected result**: An error message "No JSON file found in the ZIP." is displayed.

#### 6.2.4 Displays a success message when a valid ZIP file is uploaded

- **Description**: Ensures a success message is displayed when a valid ZIP file is uploaded.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadResults` component.
  2. Upload a valid ZIP file.

- **Expected result**: A success message "Results file uploaded successfully." is displayed.

#### 6.2.5 Displays a warning message when trying to view results without uploading a file

- **Description**: Ensures a warning message is displayed when trying to view results without uploading a file.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadResults` component.
  2. Click the "View Results" button.
- **Expected result**: A warning message "Please upload a results file first." is displayed.

#### 6.2.6 Handles errors during ZIP file processing

- **Description**: Ensures errors during ZIP file processing are handled correctly and an error message is displayed to the user.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadResults` component.
  2. Upload a ZIP file that cannot be processed.
- **Expected result**: An error message "Invalid ZIP file or corrupt JSON." is logged into the console.

#### 6.2.7 Handles file drop event

- **Description**: Ensures the file drop event is handled correctly when a file is dropped on the component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadResults` component.
  2. Simulate a file drop event with a valid ZIP file.
- **Expected result**: The file is uploaded successfully.

### 6.3 Index (Landing Page) Component

#### 6.3.1 Renders the component

- **Description**: Ensures the `Index` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.
  2. Check for the presence of specific text.
- **Expected result**: The component renders with the text "Verify Academic and Contest Submissions" and "Empower educators and contest administrators with advanced AI-powered plagiarism detection. Ensure the integrity of student work and competition submissions."

#### 6.3.2 Displays loading spinner when loading

- **Description**: Ensures the loading spinner is displayed when the component is in the loading state.
- **Type**: Automatic, Functional

- **Initial state**: `isLoading` is true
- **Test case steps**:
  1. Mock the `useAuth0` hook to return `isLoading` as true.
  2. Render the `Index` component.
- **Expected result**: The text "Preparing your experience..." is displayed.

#### 6.3.3 Calls loginWithRedirect when Get Started button is clicked

- **Description**: Ensures the `loginWithRedirect` function is called when the "Get Started" button is clicked.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.
  2. Click the "Get Started" button.
- **Expected result**: The `loginWithRedirect` function is called.

#### 6.3.4 Navigates to /home when authenticated

- **Description**: Ensures the user is redirected to the `Home` page when authenticated.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Mock the `useAuth0` hook to return `isAuthenticated` as true.
  2. Render the `Index` component.
- **Expected result**: The user is redirected to the `Home` page.

#### 6.3.5 Displays features section

- **Description**: Ensures the features section is displayed on the `Index` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.
- **Expected result**: The features section is displayed with the text "Why Choose SyntaxSentinals?" along with the features.

#### 6.3.6 Displays how it works section

- **Description**: Ensures the "How It Works" section is displayed on the `Index` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.
- **Expected result**: The "How It Works" section is displayed along with the steps.

#### 6.3.7 Displays footer

- **Description**: Ensures the footer is displayed on the `Index` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.

- **Expected result**: The footer is displayed with the text "SyntaxSentinals" and the links.

## 6.4 Upload File Box Component

### 6.4.1 Renders the component

- **Description**: Ensures the `UploadFileBox` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadFileBox` component.
- **Expected result**: The component renders with the text "Upload a ZIP file" and the file input.

### 6.4.2 Uploads a valid file

- **Description**: Ensures a valid file is uploaded using the `UploadFileBox` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadFileBox` component.
  2. Upload a valid ZIP file.
- **Expected result**: The file is uploaded successfully.

### 6.4.3 Removes a file

- **Description**: Ensures a file is removed using the `UploadFileBox` component.
- **Type**: Automatic, Functional
- **Initial state**: A file is uploaded
- **Test case steps**:
  1. Render the `UploadFileBox` component.
  2. Click the "Remove" button.
- **Expected result**: The file is removed.

### 6.4.4 Clears the file list when clearFiles is true

- **Description**: Ensures the file list is cleared when the `clearFiles` prop is true.
- **Type**: Automatic, Functional
- **Initial state**: A file is uploaded
- **Test case steps**:
  1. Render the `UploadFileBox` component with the `clearFiles` prop set to true.
- **Expected result**: The file list is cleared.

## 6.5 NavBar Component

### 6.5.1 Renders the component

- **Description**: Ensures the `NavBar` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:

1. Render the `NavBar` component.

- **Expected result**: The component renders with the logo and navigation links.

#### 6.5.2 Displays the login button when not authenticated

- **Description**: Ensures the login button is displayed when the user is not authenticated.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Mock the `useAuth0` hook to return `isAuthenticated` as false.
  2. Render the `NavBar` component.
- **Expected result**: The login button is displayed.

#### 6.5.3 Displays the logout button when authenticated

- **Description**: Ensures the logout button is displayed when the user is authenticated.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Mock the `useAuth0` hook to return `isAuthenticated` as true.
  2. Render the `NavBar` component.
- **Expected result**: The logout button is displayed.

#### 6.5.4 Calls loginWithRedirect when Log In button is clicked

- **Description**: Ensures the `loginWithRedirect` function is called when the "Log In" button is clicked.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
- 1. Mock the `useAuth0` hook to return `isAuthenticated` as false.
  2. Render the `NavBar` component.
  3. Click the "Log In" button.
- **Expected result**: The `loginWithRedirect` function is called.

# 7 Changes Due to Testing

During the testing process, several issues were identified and addressed, leading to improvements in both functionality and reliability:

- **Upload File Handling**
  - An issue was detected in `UploadBox.tsx`, where files were not properly validated before being uploaded. Additional validation logic was added to prevent incorrect file types from being processed.
  - A missing error message for unsupported file types was implemented to improve user feedback.
- **Error Handling in File Processing**
  - In `UploadResults.tsx`, tests revealed that errors during ZIP file processing were not handled gracefully. Additional error handling was introduced to catch and display meaningful error messages when invalid or corrupted ZIP files were uploaded.
- **Form Validation Enhancements**

  - The `AnalyzeFiles.tsx` component initially allowed form submission without an analysis name, causing backend errors. A required field check was added to prevent submissions without a name.

- **Test Coverage Improvements**
  - Some uncovered lines in `button.tsx` and `UploadBox.tsx` were identified through test coverage reports. Additional unit tests were written to cover these missing scenarios, increasing overall test coverage.
  - Server side code was not tested with a testing framework. This will be implemented in the future.

- **Emailing Results**
  - We noticed we had not yet implemented the feature where users are emailed after 10 minutes that their request is still processing. Thhis feature will be implemented and tested in the future.

- **Cloud**
  - Cloud deployment is a future feature that will be implemented once the model is certified per priorities

# 8 Automated Testing

The tests mentioned in the `Unit Testing` section are automated using the Jest testing framework. The tests are run using the command `npm test`.

To ensure continuous integration and automated testing, we have set up a GitHub Actions workflow. The workflow is triggered on every push and pull request to the repository. It runs the tests on an Ubuntu environment with Node.js version 18.x.

## 8.1 Testing Workflow Configuration

The following is the configuration of the GitHub Actions workflow used for automated testing:

```
name: Run Frontend Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-20.04

    strategy:
      matrix:
        node-version: [18.x]

    steps:
    - name: Checkout repository
      uses: actions/checkout@v4

    - name: Set up Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node-version }}

    - name: Install dependencies
      run: npm install
      working-directory: ./src/frontend

    - name: Run tests
      run: npm test
      working-directory: ./src/frontend
```

## 8.2 Testing Workflow Steps

- **Checkout repository**: This step uses the `actions/checkout@v4` action to check out the repository code.
- **Set up Node.js**: This step uses the `actions/setup-node@v4` action to set up Node.js with the specified version (18.x).
- **Install dependencies**: This step runs `npm install` to install the necessary dependencies for the frontend.

- **Run tests**: This step runs `npm test` to execute the Jest tests for the frontend.

By using this automated testing workflow, we ensure that our code is continuously tested and validated, providing immediate feedback on any issues that may arise from changes in the codebase.

## 8.3 Test Coverage Workflow Configuration

In addition to running unit tests, we have configured a separate GitHub Actions workflow to generate a test coverage report using Jest. This helps us monitor the extent to which our code is covered by automated tests.

The following is the configuration for the test coverage workflow:

```
name: Run Frontend Unit Tests

on: [push, pull_request]

jobs:
coverage:
runs-on: ubuntu-20.04

steps:
  - name: Checkout repository
    uses: actions/checkout@v3

  - name: Run Jest Coverage Report
    uses: ArtiomTr/jest-coverage-report-action@v2
    with:
      working-directory: ./src/frontend
      github-token: ${{ secrets.GITHUB_TOKEN }}
```

## 8.4 Test Coverage Workflow Steps

- **Checkout repository**: This step uses the `actions/checkout@v3` action to check out the repository code.

- **Run Jest Coverage Report**: This step uses the `ArtiomTr/jest-coverage-report-action@v2` action to generate a coverage report for the Jest tests in the frontend directory.

By incorporating test coverage reporting into our workflow, we can ensure that our automated tests provide sufficient coverage of the codebase, identifying untested areas and improving software reliability.

# 9 Trace to Requirements

# 10 Trace to Modules

# 11 Code Coverage Metrics

The below report is for the frontend codebase of SyntaxSentinals. The backend has not been tested with a testing framework yet. Instead we ensure the backend is working as expected by manually testing the API endpoints using Postman and ensuring the server is sent the expected request via the frontend testing suite.

The test coverage report provides the following key metrics:

- **Statements**: 94.35% (234/248)
- **Branches**: 81.03% (47/58)
- **Functions**: 86.79% (46/53)
- **Lines**: 96.18% (227/236)

Below is a breakdown of coverage by individual files:

| Directory | Statements | Branches | Functions | Lines |
|---|---|---|---|---|
| Features.tsx | 100% | 100% | 100% | 100% |
| Navbar.tsx | 100% | 100% | 100% | 100% |
| UploadBox.tsx | 96.42% | 84.21% | 93.33% | 98.14% |
| GetStarted.tsx | 100% | 100% | 100% | 100% |
| Home.tsx | 90.32% | 69.56% | 73.91% | 93.16% |
| Button.tsx | 100% | 66.66% | 100% | 100% |
| utils.ts | 100% | 100% | 100% | 100% |
| AnalyzeFiles.tsx | 91.52% | 75% | 75% | 94.54% |
| UploadResults.tsx | 89.23% | 63.63% | 72.72% | 91.93% |

Table 2: Test Coverage Breakdown by Directory

Some uncovered lines include:

- **UploadBox.tsx**: Line 26.
- **button.tsx**: Line 44.
- **AnalyzeFiles.tsx**: Lines 22, 25, 76.
- **UploadResults.tsx**: Lines 20, 23, 50-51, 89.

These uncovered areas indicate potential gaps in test cases that should be addressed to achieve full coverage.

# References

# Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)