System Verification and Validation Plan for Software Engineering

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

April 4, 2025

Revision History

Date	Version	Notes
November 1	1.0	Initial documentation

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

Symbol/Abbreviation/Acronym	**Description**
V&V	Verification and Validation
NLP	Natural Language Processing
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Deployment
API	Application Programming Interface
SRS	Software Requirements Specification
UI	User Interface
MG	Module Guide
MIS	Module Interface Specification
PIPEDA	Personal Information Protection and Electronic Documents Act
FIPPA	Freedom of Information and Protection of Privacy Act

Table 1: Symbols, Abbreviations, and Acronyms

This document outlines the strategies and processes used to ensure that the plagiarism detection system developed by the SyntaxSentinels team meets all functional and non-functional requirements. The primary goal of this plan is to build confidence in the correctness, usability, and performance of the system. It also focuses on identifying and mitigating potential risks, ensuring that the final product aligns with academic and competition standards. This document is organized as follows. The general information section provides an overview of the objectives, challenges, and relevant project documents used throughout the V&V process. The plan section describes the roles and responsibilities of the team members and the tools used for automated testing and verification. The system tests section lists the tests performed for both functional and non-functional requirements, with traceability to the SRS. The unit test description section details the unit testing scope, the modules tested, and the strategies for covering edge cases. Finally, the appendix contains symbolic parameters, survey questions (if applicable), and any other relevant information to support the V&V process. This plan will evolve as the project progresses, with updates following the completion of the detailed design and implementation phases.

2 General Information

2.1 Summary

The software being tested is a plagiarism detection system designed to identify similarities in Python code submissions, called SyntaxSentinels. This system utilizes natural language processing (NLP) techniques to analyze code semantics, preventing common circumvention methods like adding benign lines or altering variable names. Its core function is to allow users to input code snippets and receive a plagiarism report containing similarity scores, which, when compared to a threshold, indicate the likelihood of plagiarism. This tool is primarily intended for use in academic and competitive environments to promote fairness and integrity in code submissions.

2.2 Objectives

The primary objectives of this V&V plan are to:

- Build confidence in the programs correctness by ensuring alignment with the SRS requirements.
- Show that we have met the documented safety and security requirements (SR-SAF1- SR-SAF5) in the Hazard Analysis document.
- Demonstrate adequate usability in the program by conducting functional and non-functional tests mentioned in this document.

Out of Scope:

- Validation of any external libraries will be assumed to be handled by their maintainers.

2.3 Challenge Level and Extras

This project has been classified as having a General difficulty level, as agreed with the course instructor. Planned extras include:

- **User Manual for Instructors and Administrators**: The user manual will provide detailed instructions on how to use the plagiarism detection system. It will include:
  - Step by step instructions for setting up the system.
  - Instructions on how to run the servers
  - Step-by-step instructions for uploading code snippets and generating reports.
  - Troubleshooting tips for common issues.
- **Benchmarking Against MOSS**: The system's performance will be benchmarked against MOSS (Measure of Software Similarity) to evaluate its effectiveness. This will involve:
  - Running the same set of code snippets through both systems.
  - Comparing similarity scores and false positive rates.
  - Documenting the results in a performance report.

2.4 Relevant Documentation

The following documents are critical to the development and V&V efforts for this project.

- **Software Requirements Specification (SRS)**: Defines the project's requirements, guiding both verification and validation. (Khan et al., 2024d).
- **User Guide**: Provides operational instructions, relevant for usability testing. (Khan et al., 2024e).
- **Module Guide (MG)**: Outlines the system's architecture, essential for design verification. (Khan et al., 2024b).
- **Module Interface Specification (MIS)**: Details the internal modules and interfaces, critical for unit testing. (Khan et al., 2024c).
- **Hazard Analysis**: Identifies potential risks, guiding validation efforts for safety and security. (Khan et al., 2024a).

3 Plan

This section outlines the structured plan for verification and validation of the project. It defines team roles, describes approaches for verifying different phases of the development process, and specifies tools to be used in ensuring the project meets quality standards. The subsections cover SRS verification, design verification, implementation verification, automated testing tools, and software validation.

3.1 Verification and Validation Team

The following table lists team members and their respective roles in the verification and validation process. Each member will be responsible for writing test cases, executing tests, and documenting results for their assigned areas. The team will meet regularly to discuss progress, address issues, and ensure alignment with project goals.

Team Member	Role in Verification and Validation
Mohammad Mohsin Khan	Oversees system architecture verification and leads checklist creation.
Lucas Chen	Security verification, focusing on access control and data flow assessments.
Dennis Fong	Responsible for interface and compatibility reviews between components.
Julian Cecchini	Ensuring that individual modules or components integrate smoothly with one another.
Luigi Quattrociocchi	Tracks checklist items and maintains verification documentation.

Table 2: Verification and Validation Team Roles

3.2 SRS Verification Plan

The SRS (Software Requirements Specification) verification will ensure that all functional and non-functional requirements are accurately documented and align with the project goals. The verification plan includes:

- **Structured Reviews**: Team members will perform a structured review of the SRS document, verifying that all requirements are feasible and testable.
- **Checklist-Based Verification**: An SRS checklist will be used to ensure all critical elements, such as functional completeness and clarity, are covered.
- **Reviewer Feedback Sessions**: We will gather feedback from peer reviewers and our project supervisor, meeting to discuss any discrepancies or ambiguous requirements. These meetings will include task-based inspections, where reviewers are asked to analyze requirements based on specific scenarios.

The checklist for SRS review will cover:

Item	Description
1. Completeness	
1.1 Purpose and Scope	Document states the purpose and scope of the project.
1.2 Stakeholders	Key stakeholders (clients, users) are defined and are relevant.
1.3 Functional Requirements	All primary functions (e.g., plagiarism detection, reporting) are covered and are descriptive.
1.4 Non-Functional Requirements	Includes performance, usability, and security requirements.
2. Clarity	
2.1 Unambiguous Terminology	Each requirement is clearly stated, terms are defined (e.g., MOSS, NLP).
2.2 Glossary Completeness	All acronyms and terms are included in the glossary.
3. Consistency	
3.1 Consistent Terminology	Terminology and references are consistent throughout.
3.2 No Conflicting Requirements	No contradictory requirements (e.g., conflicting performance vs. security).
4. Verifiability	
4.1 Testable Requirements	Each 'requirement' is testable and verifiable (e.g., accuracy metrics, response times).
4.2 Acceptance Criteria	Clear acceptance criteria for each requirement.
5. Traceability	
5.1 Unique Identifiers	Each requirement has a unique identifier.
5.2 Source of Requirements	Requirements link to stakeholder needs or project goals.
6. Feasibility	
6.1 Technical Feasibility	Requirements are achievable within project constraints.

Item	Description
6.2 Practical Constraints	Constraints such as budget and timeline are realistic.
7. Security and Privacy	
7.1 Data Retention Policy	Compliance with data privacy laws (e.g., PIPEDA).
7.2 Access Control Requirements	Requirements for user authentication and authorization are clear.
8. Modifiability	
8.1 Organized Structure	Requirements are logically organized for easy updates.
8.2 No Redundancies	No duplicate requirements to avoid confusion.
9. Compliance and Ethics	
9.1 Legal and Ethical Standards	Legal (e.g., Copyright) and ethical considerations are addressed.

Table 3: Checklist for SRS Verification

3.3 Design Verification Plan

The design verification plan aims to ensure that the design accurately implements the requirements and adheres to best practices. This plan includes:

- **Checklist-Based Design Review**: A checklist will be used to guide reviews, focusing on key aspects such as modularity, scalability, and security.
- **Peer Design Reviews**: Peer reviews by classmates and team members will provide feedback on the design, highlighting potential areas of improvement.
- **Regular Team Reviews**: Scheduled meetings will allow the team to discuss any design modifications, verify alignment with the SRS, and

ensure consistency across components.

The checklist for design review will cover:

Item	Description
1. Functional Verification	
1.1 Requirement Mapping	Each design element corresponds to at least one requirement in the SRS.
1.2 Functionality Coverage	The design covers all specified functionalities, including error handling and edge cases.
1.3 Interface Definition	All interfaces between components are clearly defined and consistent with requirements.
2. Structural Verification	
2.1 Modular Design	The design is divided into logical, independent modules with well-defined interfaces.
2.2 Dependency Analysis	Dependencies between components are minimized, and unnecessary couplings are avoided.
2.3 Hierarchical Structure	The design follows a clear hierarchy, with higher-level components orchestrating lower-level ones.
3. Usability and Accessibility	
3.1 User Interface Design	UI elements are consistent, intuitive, and meet accessibility standards (if applicable).
3.2 Navigation and Flow	User navigation and workflow are logical, efficient, and follow a coherent path.
3.3 Accessibility Standards	The design adheres to relevant accessibility guidelines, such as WCAG, to ensure usability for all users.
4. Performance and Optimization	
4.1 Performance Criteria	The design incorporates mechanisms to meet performance requirements (e.g., response time, resource usage).

Item	Description
4.2 Scalability	Design supports scalability to handle expected load increases without significant degradation.
5. Security and Privacy	
5.1 Data Flow Security	The design ensures secure data handling, storage, and transmission between components.
5.2 Access Control Mechanisms	Roles and permissions are implemented to restrict unauthorized access to sensitive components.
5.3 Compliance	Design complies with security and privacy standards as outlined in the SRS.
6. Traceability and Documentation	
6.2 Documentation Completeness	Documentation is complete, with descriptions of components, workflows, and data flow.
6.3 Version Control	Design documentation is version-controlled to track changes and updates.

Table 4: Checklist for Design Verification

3.4 Verification and Validation Plan Verification Plan

The verification and validation plan itself will also undergo verification to ensure its effectiveness. This will be achieved by:

- **Peer Reviews**: Classmates will review the plan to provide feedback on its clarity, feasibility, and alignment with project requirements.
- **Mutation Testing**: We will apply mutation testing to validate that the plan can effectively catch errors and discrepancies in project requirements and implementation.
- **Checklist-Based Verification**: A checklist specific to the verification and validation plan will guide reviewers in assessing all critical aspects of the plan.

Item	Description
1. Verification Plan Completeness	Verify that the Verification and Validation Plan includes all necessary sections, such as objectives, scope, and methodologies.
2. Clarity of Objectives	Ensure that the objectives of the verification and validation activities are clearly stated and aligned with project goals.
3. Methodology Definition	Check that each verification method (e.g., reviews, inspections, testing) is well-defined with clear procedures.
4. Team Roles and Responsibilities	Confirm that each team member's role in the verification and validation activities is documented and clear.
5. Review and Inspection Procedures	Validate that there are structured procedures for design reviews and inspections, with criteria for passing/failing.
6. Integration of Automated Tools	Verify that automated tools for testing and validation (e.g., CI/CD, linters, static analyzers) are specified and included in the plan.
7. Traceability of Requirements	Confirm that the verification and validation activities trace back to specific project requirements to ensure coverage.
8. Acceptance Criteria	Ensure there are clear acceptance criteria defined for each verification and validation task.
9. Documentation of Test Cases	Verify that each planned test case has clear documentation, including expected outcomes, inputs, and procedures.
10. Risk Management in Verification	Confirm that potential risks in the verification and validation process are identified and mitigation strategies are documented.
11. Feedback Loop	Ensure that there is a mechanism for capturing feedback and iterating on the verification and validation process as needed.
12. Reporting and Tracking of Issues	Check that there is a process for documenting, tracking, and addressing issues found during verification and validation.

Item	Description
13. Schedule and Milestones	Verify that there is a realistic schedule and milestones for completing verification and validation activities.

Table 5: Checklist for V&V Plan Verification

3.5 Implementation Verification Plan

The implementation verification plan focuses on ensuring the correctness and quality of the implementation phase. The plan includes:

- **Unit Testing**: Each function and module will undergo unit testing using the frameworks mentioned in section 3.6 to ensure they meet functional requirements.
- **Static Code Analysis**: We will use static analysis tools mentioned in section 3.6 to verify code quality, adherence to coding standards, and security practices.
- **Code Walkthroughs**: Code walkthroughs will be held in team meetings and during the final presentation, allowing team members to inspect each other's code for issues in logic, structure, and readability.

3.6 Automated Testing and Verification Tools

To streamline the verification process, the following automated testing and verification tools will be used:

- **Testing Framework**: A testing framework (Pytest for Python) and (Playwright for ReactJs) will be used to automate testing of individual functions, modules, and E2E.
- **Continuous Integration (CI) Tool**: GitHub Actions will be set up for continuous integration to ensure that tests are automatically run on new code submissions.

- **Code Coverage Tool**: Code coverage tools (e.g Pytest with Coverage.py) will track the extent to which the codebase has been tested, ensuring that all critical paths are covered.
- **Linters/Static Analysis Tools**: Linters appropriate to the project's programming language will enforce coding standards, improving code readability and maintainability.

3.7 Software Validation Plan

The software validation plan outlines the strategies for validating that the software meets the intended requirements. This includes:

- **User Review Sessions**: Review sessions with stakeholders and user representatives will be conducted to validate that the system meets user needs and expectations.
- **Rev 0 Demonstration**: Shortly after the scheduled Rev 0 demo, we will seek feedback from stakeholders and supervisors, if applicable, to confirm that the design and initial implementation align with project goals.
- **End to End Testing**: We will plan E2E sessions to ensure the software works correctly from start to end and meets all functional requirements.
- **Performance Testing**: Once functional requirements have been implemented, non-functional requirements will be implemented then performance testing will be conducted to validate that the software meets non-functional requirements, such as response times and resource usage.

The validation process will involve gathering external data, where possible, to test the system's accuracy and performance under realistic scenarios.

Summary

This comprehensive plan for verification and validation addresses each phase of the project lifecycle, from requirements to implementation. By following a structured approach with well-defined roles, checklists, and automated tools,

we aim to ensure a high level of quality, accuracy, and security in the final product.

4 System Tests

This section outlines the system tests to be performed for both functional and non-functional requirements. The tests are categorized based on the area of testing, such as look and feel requirements, usability, and versioning. Each test includes a title, test ID, control, initial state, input, output, test case derivation, and how the test will be performed.

4.1 Tests for Functional Requirements

Each test section addresses a particular initial condition and input set. Therefore, each test has a particular set of outputs that the system should produce for it. A functional requirement of section 9 in the SRS (Khan et al., 2024d) is only associated with a test if the output set of the test corresponds to the fit criterion of that functional requirement, and the fit criterion should be an observable component of a system state, such as a display of a return item or notification presented. Through this association, every functional requirement is provided a test that is capable of verifying it, thereby ensuring the test areas fully cover the functional requirements.

4.1.1 Plagiarism Analysis Input Upload Tests

Applies to all FRs involving uploading code snippets to the system. This currently covers FR-1 in subsection 9.1 of the SRS (Khan et al., 2024d).

1. test-FR-1

   Control: Automatic.

   Initial State: system is active and set up in spot to receive code snippet uploads, zero or more existing snippets are currently uploaded.

   Input: code snippet file(s).

   Output: System signifies input was received and continues activity without error, awaiting further action directives (such as uploading more files or initiating analysis).

Test Case Derivation: The system will look to continually receive code snippets from user as part of a necessary step to initiate plagiarism analysis.

How test will be performed: A script will open the system in the appropriate spot for receiving code and will upload files in varying amounts, one batch at a time, all the while inspecting for failure during the process. Script will be integrated into CI/CD pipeline to ensure continued system functionality (types: dynamic, functional).

4.1.2 Plagiarism Analysis Result Tests

Applies to all FRs involving immediate output of plagiarism analysis. This currently covers FR-2, FR-4, and FR-5 in subsection 9.1 of the SRS (Khan et al., 2024d).

1. test-FR-2

   Control: Automatic.

   Initial State: system is active and set up in spot for initiating analysis, 2 or more code snippets have been uploaded.

   Input: command to initiate analysis (button, enter, etc.).

   Output: System presents list of similarity scores for all code snippet pairings along with corresponding threshold scores and any code snippet pairing that has exceeded its threshold has been flagged. System remains active, awaiting for action directives on what to do.

   Test Case Derivation: The system should possess an algorithm that analyzes code snippets and produces similarity scores and threshold scores in return. These scores should be produced between every pairing of code snippets as the plagiarism is a relative assessment between one code piece and another. The system must able to pass this algorithm received code snippets and use its results to flag any pairing that exceeded its threshold before returning from its analysis state to the user for further interaction.

   How test will be performed: A script will open the system's spot to initiate analysis with 2 or more code snippets already inserted into the system, and command for starting analysis will be entered. Errors will

be inspected for up until similarity scores and thresholds have been presented. By this point, code flaggings should also be available to ascertain. (type: dynamic, functional).

4.1.3 Guide Documentation Generation Tests

Applies to all FRs involving guide documentation generation for user. This currently covers FR-3 in subsection 9.1 of the SRS (Khan et al., 2024d).

1. test-FR-3

   Control: Automatic.

   Initial State: System is active and set up in spot for generating guide documentation.

   Input: command for documentation presentation (button, enter, etc.).

   Output: System generates documentation in a viewable medium, such as text or a PDF file, and remains active while awaiting for further action directives.

   Test Case Derivation: system possesses guide documentation which it should be capable of transferring to user to provide guidance on use cases of system when prompted.

   How test will be performed: A script will open the system's spot for generating guide documentation and command for guide generating documentation will be entered, all the while inspecting for error. It will check for missing documentation at the end. (type: dynamic, functional).

4.1.4 Report Documentation Generation Tests

Applies to all FRs involving analysis report documentation generation for user. This currently covers FR-6 of subsection 9.1 of the SRS (Khan et al., 2024d).

1. test-FR-4

   Control: Automatic.

Initial State: System is active and set up in spot for report generation, most recent plagiarism analysis has been completed since after system activation and its results are available.

Input: command for report documentation generation (button, enter, etc.).

Output: System provides report documentation in a viewable medium, such as text or a PDF file, and remains active awaiting for further action directives.

Test Case Derivation: The system should possess the ability to aggregate the results of the plagiarism analysis which has most recently occurred and insert them into a template that can summarize all findings to a user, which will become the report document given.

How test will be performed: A script will have the system conduct a plagiarism analysis to provide inputs for the report generation as this simulates what must happen before a user can unlock report generation. If successful, it will proceed to open the spot for generating a report where the command for generating a report will be entered, all the while inspecting for errors during the process. It will check for missing documentation at the end (type: dynamic, functional).

4.1.5 Account Creation Tests

Applies to all FRs involving creating an account within the system. This currently covers FR-7 of subsection 9.1 of the SRS (Khan et al., 2024d).

1. test-FR-5

   Control: Automatic.

   Initial State: system is active and set up in spot for account creation.

   Input: command for account creation (button, enter, etc.) alongside account user email and password, and the email is not associated with any existing account.

   Output: System indicates account creation was successfully created for logging in with (evidenced by existence of email within created account list) and remains active, awaiting further action directives.

Test Case Derivation: The system should be able to assess a set of account credentials is not yet within the system and proceed to add this set to the set of accounts that can be logged in with.

How test will be performed: A script will open the system's spot for account creation, email and password not associated with any existing account will be given in the appropriate area, and command for account creation will be entered; all the while inspecting for any failure mode within the process. Created account will be inspected for at the end. (type: dynamic, functional).

2. test-FR-6

   Control: Automatic.

   Initial State: system is active and set up in spot for account creation.

   Input: command for account creation (button, enter, etc.) alongside account user email and password, and the email is associated with an existing account.

   Output: System notifies account was not possible to create for logging in with and remains active, awaiting further action directives.

   Test Case Derivation: A set of pre-existing account credentials should not be possible to create an account with. Otherwise, account creation is arbitrary and not truly provided by the system as any set of account credentials are not tied to any particular account.

   How test will be performed: A script will open the system's spot for account creation, email and password asssociated with an existing account will be given in the appropriate area, and command for account creation will be entered; all the while inspecting for any failure mode within the process (type: dynamic, functional).

4.1.6 Account Login Tests

Applies to all FRs involving logging into account within the system. This currently covers FR-8 of subsection 9.1 of the SRS (Khan et al., 2024d).

1. test-FR-7

Control: Automatic.

Initial State: system is active and set up in spot for account login.

Input: command for account login (button, enter, etc.) alongside account user email and password, and the email is associated with an existing account.

Output: System notifies account login was successful and and remains active, awaiting further action directives.

Test Case Derivation: The system should be able to validate a set of pre-existing account credentials to facilitate login.

How test will be performed: A script will open the system's spot for account login, email and password asssociated with existing account will be given in the appropriate area, and command for account login will be passed; all the while inspecting for any failure mode within the process (type: dynamic, functional).

2. test-FR-8

Control: Automatic.

Initial State: system is active and set up in spot for account login.

Input: command for account login (button, enter, etc.) alongside account user email and password, and the email is not associated with any existing account.

Output: System notifies account login was not successful and remains active, awaiting further action directives.

Test Case Derivation: A set of account credentials not associated an existing account will fail to allow login as the system should determine there is no account to validate against.

How test will be performed: A script will open system's spot for account login, email and password not asssociated with existing account with will be given in the appropriate area, and command for account login will be passed; all the while inspecting for any failure mode within the process (type: dynamic, functional).

4.1.7 Result Visualization Tests

Applies to all FRs involving visualizing plagiarism analysis results provided by user in a .zip file. This currently covers FR-9 of subsection 9.1 of the SRS (Khan et al., 2024d).

1. test-FR-9

   Control: Automatic

   Initial State: System is active and a job has been completed and is ready to be visualized.

   Input: None

   Output: System provides visualization in viewable medium, such as image or file, that corresponds to the results.

   Test Case Derivation: System should possess ability to pull all relevant data from a completed job and insert it into a visualization template that can be presented to the user.

   How test will be performed: A script will open the system's spot for result visualization and will inspect for any failure mode within the process

4.2 Tests for Nonfunctional Requirements

This section outlines the system tests to be performed for non-functional requirements, including accuracy, performance, and usability.

4.2.1 Look and Feel

These tests are to ensure that the way the front end is designed adheres to the defined requirements with respect to layout, typography, component alignment, etc.

Front End Component Review: We will engage a developer to look through the code and interface and ensure components are implemented as they are defined in the SRS.

Accessibility Check: A developer will review if accessibility tools exist or not within the application. Important tools are screen readers and alt text for images. Furthermore, the developer must check for sufficient colour contrast between components.

Validate Look and Feel User Interface Tests

1. test-LF-1

   NFR: LF-AR1, LF-AR2, LF-AR3, LF-SR1, LF-SR2, LF-SR3, LF-SR4

   Type: Manual, Front End Review

   Initial State: Application on home page, ready for use

   Input/Condition: Tester engagement

   Output/Result: List of all identified inconsistencies that are found that do not adhere to SRS defined requirements.

   How test will be performed:

   (a) A developer will review the code to make sure the look and feels requirements were implemented:

      i. They will check that the application has a a uniform colour palette.
      ii. They will check that the application has tooltips on buttons with informative text.
      iii. They will check that the application is responsive and scales well for different screen sizes and resolutions.
      iv. They will check that the application uses the Roboto font family.
      v. They will check that the application has uniform spacing between elements.
      vi. They will check that the application has buttons with rounded corners and change colour upon hover.

   (b) They will then check that the application to ensure that it adheres to all look and feel requirements.

2. test-LF-2

   NFR: CR-SC3

Type: Manual, Accessibility Check

Initial State: Application on home page, ready for use

Input: Tester engagement

Output: List of missing accessibility considerations that have been overlooked

How test will be performed:

(a) The developer will review the code and application to ensure that accessibility considerations have been implemented.

(b) They will verify that all images have alt text.

(c) They will verify that all text has sufficient contrast and sizing using Google Chrome devtools.

(d) The developer will note down any missing implementations of accessibility requirements.

4.2.2 Usability and Humanity

These test cases are intended to build a foundation for the user's experience on the application. Thus, the only real way to get feedback the user experience is to have someone use the application and provide feedback.

User Test Demo: We will engage a user to test the application, bringing them through a new user process, and ask for feedback on specific areas relevant to the requirements that were set. The feedback will be a result of a survey, with questions structured as "On a scale of 1 to 10, how would you rate X?" with X being some aspect of usability.

Validate Usability and Humanity tests

1. test-UH-1

   NFR: UH-E1, UH-E3, UH-L1, UH-UP1, UH-UP2, OE-P1

   Type: Manual, User Test Demo

   Initial State: Application on home page, ready for use

   Input/Condition: User engagement

Output/Result: Subjective feedback for how well application scores on different requirements relating to user experience

How test will be performed:

(a) A user will use the application, going through the onboarding process and doing a demo of the application.

(b) After a full demo, the user will be asked to provide feedback on different aspects of usability, such as ease of use, app clarity, and prompt politeness.

2. test-UH-2

   NFR: UH-E2

   Type: Manual

   Initial State: Application on home page, ready for use

   Input: Tester engagement

   Output: A count for the number of clicks required to navigate to different functionality

   How test will be performed:

   (a) The user will navigate to each of the following functions in the user interface from the home page:

      i. The user will make a code submission to be analyzed.

      ii. The user will upload a generated report file to be viewed.

   (b) The user will note how many clicks were required to accomplish each function.

3. test-UH-3

   NFR: UH-PI2

   Type: Manual

   Initial State: Application on home page, ready for use

   Input: Tester engagement

   Output: All identified instances of foreign languages used

   How test will be performed:

(a) The tester will navigate to each page and submenu in the application.

(b) They will verify that all pages and prompts are in the correct language (English (US)).

4.2.3 Versioning

These testcases are for versioning requirements that are key in order to roll back to old iterations of the code or old iterations of the model when needed. These will mainly be tested by checking if old versions are saved. Note that this check will be done after each iteration of the model.

Validate old versions of project

1. test-V-1

   NFR: MS-M1, OR-R2

   Type: Document Review

   Initial State: N/A

   Input/Condition: Tester engagement

   Output/Result: Historical versions of machine learning models with a changelog should be stored and accessible on GitHub

   How test will be performed:

   (a) A developer will navigate to the project GitHub repository and find the folder containing all past and current versions of models, each with an associated performance report, as well as a changelog including dates of revisions

   (b) This will be done with each iteration of the model code.

4.2.4 Privacy/Security

These testcases ensure that the system is secure, and does not violate any privacy rights. The tests for these requirements will revolve around doing audits to ensure system integrity.

Data Storage Audit: We will manually check the databases, file system and any logs to look for any user-generated data. If found, this data must be determined to be secure and only accessible by the user who uploaded it. It must also be in a state that it is ready to be deleted upon a user request at any time. This audit may also be done in a test environment to check live logs of the system.

Endpoint Security Check: A tester will invoke backend endpoints and ensure that they are secure. They must also check that the website uses https encryption (padlock symbol in address bar)

Legal Audit: We will manually check the system to ensure that it adheres to legal policies defined in the SRS, such as PIPEDA and FIPPA. This audit will be done regularly, and with each new version release, to the best of our abilities.

Validate Secure Data Storage and Data Encryption

1. test-PS-1

   NFR: SR-P1

   Type: Manual, Data Retention Audit

   Initial State: Application ready for use

   Input/Condition: Tester engagement

   Output/Result: A list of all identified user-generated data that is stored insecurely or not ready for deletion

   How test will be performed:

   (a) A tester will enable logging in the components of the application.

   (b) They will do a run through of the different functionalities, including making a submission and analyzing a generated report file.

   (c) With each function, the tester will check the databases, filesystem, and logs, and then search for any user-generated or personally identifiable data.

   (d) For each instance of identified user-data, the tester must determine if the data is stored securely, is only accessible by the user who uploaded it, and that it is ready to be deleted upon a user request at any time.

2. test-PS-2

   NFR: SR-P2

   Type: Dynamic

   Initial State: Application ready for use

   Input/Condition: Backend endpoint invocation

   Output/Result: A list of exposed endpoints that do not support in-transit encryption

   How test will be performed:

   (a) Script will invoke backend end points using HTTPS protocol.

   (b) They will ensure that the response provides relevant security headers.

3. test-PS-3

   NFR: CR-L1, CR-L2, CR-L3, CR-S2, CR-S5

   Type: Manual, Legal Audit

   Initial State: Application ready for use

   Input/Condition: Tester engagement

   Output/Result: List of components of the system that breach any regulations, with an accompanying explanation of the how the breach could manifest itself

   How test will be performed:

   (a) A third party tester will inspect the code repository and system application against laws and regulations, noting any component of the system that may potentially infringe upon any data or privacy laws, intellectual property, or academic integrity.

4.2.5 Development Standards

These requirements ensure that the system is built in a way that adheres to standard software development principles. This ensures that the code is maintainble, scalable, and reliable.

Code Inspection: The code will be inspected to make sure the system is built adhering to software development standards.

Respository Inspection: The Github repository for this project will be checked to make sure bugs are listed as issues to keep track of what needs to be done in the system.

Validate Development Standards

1. test-DS-1

   NFR: MS-M3

   Type: Manual, Repository Inspection

   Initial State: Development temporarily paused

   Output/Result: Indication of bugs being tracked on Github

   How test will be performed:

   (a) A tester will look at the 'issues' section of the github repository.

   (b) They will check that there are bugs that exist as issues.

   (c) If there aren't, they will consult the team and ask if developers are issuing bugs or if there are simply no bugs.

2. test-DS-2

   NFR: MS-S1, CR-SC1

   Type: Manual, Code Inspection

   Initial State: Development temporarily paused

   Output/Result: Indication of components not adhering to software development standards

   How test will be performed:

   (a) Team members will regularly inspect the code and ensure all components follow software development standards, including SOLID principles and the existence of documentation and tests.

   (b) If there are components that don't they will consult the other team members to discuss issues with that component.

4.2.6 Documentation

This section is to ensure that documentation exists to guide users and help them troubleshoot issues they are having.

Documentation Review: A tester will look through the webpage, and ensure that documentation exists. Then, they must check to make sure the documentation is accurate, and not misleading.

Validate Program, help, and training documentation exists and is accurate

1. test-D-1

   NFR: UH-L2, OE-P2

   Type: Manual, Documentation Review

   Initial State: Application ready for use

   Output/Result: Certification that documentation is valid

   How test will be performed:

   (a) A developer will navigate to the user documentation and help pages within the application. They must first ensure that documentation exists, and is easy to find.

   (b) They must then verify that the documentation is correct, complete, uses clear non-technical language, and is helpful for users.

4.2.7 Performance

These testcases ensure that the system is able to handle a certain amount of load, and that it is able to process data in a timely manner.

Batch processing time is less than 4 hours for assumed upper bound of classroom size

1. test-PR-1

   NFR: PR-SL1, PR-C1

   Type: Performance, Dynamic

Initial State: The system is idle and ready to process a batch of code submissions

Input: 500 code snippets of 200 lines or less

Output: The system completes processing the batch within 4 hours.

How test will be performed: Script will open system's spot for plagiarism analysis and pass system 500 code snippets of 200 lines or less alongside command to initiate analysis. It will use a computer timer through a library, such as pyperf, to time the interval between the analysis start and the analysis return. If the interval is under 4 hours, the test will return a pass. Will be integrated into a CI/CD pipeline and be run periodically to affirm system health and performance over time.

System remains operational after malformed input

1. test-PR-2

   NFR: PR-RFT1

   Type: Robustness, Dynamic

   Initial State: System is operational, awaiting user input.

   Input: Malformed input (e.g., corrupted file) is submitted to the system.

   Output: System displays an error message to the user indicating invalid input, without crashing or becoming unresponsive.

   How test will be performed:

   (a) navigate to the home page.

   (b) Submit a file that is known to be corrupted or invalid.

   (c) Verify that an error message is displayed to the user by checking if the function call was made.

   (d) Submit a file that is known to be valid.

   (e) Verify that the system accepts the valid input and processes it correctly.

System meets accuracy expectations

1. test-PR-3

   NFR: PR-PA1, PR-PA2

   Type: Performance, Dynamic

   Initial State: System is operational and awaiting user input for code submission

   Input: Code snippet test set (on the scale of 500 snippets) and command to initiate analysis

   Output: All system analysis results have greater than 90% accuracy and less than 5% false positives compared against the ground truths when using default threshold

   How test will be performed: Script will pass current test set of code snippets and give command-line argument to initiate analysis from detector. Upon return of analysis, it will compare results against ground truths of test set using default threshold. If accuracy rate is 90% and false positives are lower than 5%, the test will return a pass. This will be integrated into a CI/CD pipeline and be run periodically to affirm system health and performance over time. Also, the test set will potentially change over time to keep any possible bias in check.

Feature Updates Do Not Degrade Performance

1. test-PR-4

   NFR: PR-SE1

   Type: Functional, Dynamic, Regression

   Initial State: N/A

   Input: Code for new feature is added

   Output: All base features still function as expected with new feature code in place

   How test will be performed: When a new feature is added to the system (as denoted in a pull request or otherwise), all system tests for functional requirements are re-ran to ensure no previous functionality has been lost. This will be facilitated by a CI/CD pipeline (using GitHub actions) for all functional requirement tests.

4.2.8 Operational and Environmental

These testcases ensure that the system is able to operate in different environments and under different conditions.

System is able to interface with cloud computing services

1. test-OE-1

   NFR: OE-IAS1

   Type: Manual

   Initial State: System setup with no cloud service connected.

   Input: User attempts to configure and connect the system to a cloud service (e.g. AWS or Azure).

   Output: System successfully connects to the selected cloud service and displays a confirmation message.

   How test will be performed:

   (a) navigate to the cloud configuration settings.

   (b) Select a cloud provider (AWS or Azure) and enter necessary credentials.

   (c) Verify that the system displays a confirmation message upon successful connection.

System is deployable on Free Hosting Service

1. test-OE-2

   NFR: OE-IAS2

   Type: Manual

   Initial State: User clones the system repository from GitHub.

   Input: Deploy the system on a free hosting service (e.g. Heroku, Netlify).

   Output: System is accessible via a public URL and functions as expected.

   How test will be performed:

(a) Clone the system repository from GitHub.

(b) Deploy the system on a free hosting service.

(c) Verify that the system is accessible via a public URL and functions as expected without any issues.

System is able to authenticate the user via external services

1. test-OE-3

   NFR: OE-IAS3

   Type: Manual

   Initial State: User is not authenticated and does not have a valid auth token.

   Input: User attempts to log in using an external service (e.g. Google, GitHub).

   Output: System authenticates the user and grants access to the UI.

   How test will be performed:

   (a) Navigate to the login page.

   (b) Select an external service (e.g. Google, GitHub) to log in with.

   (c) Verify that the system authenticates the user and is redirected to the home page.

4.2.9 Maintainability and Support

These testcases ensure that the system is maintainable and supportable. This includes ensuring that the system is easy to maintain and that support is available to users.

Model release is accompanied by a report of metrics

1. test-M-1

   NFR: MS-M2

   Type: Inspection

   Initial State: The model has been trained and is ready for release

Input: Release the model

Output: A report is generated with metrics such as accuracy, precision, etc.

How test will be performed: Release the model and verify that a report is generated with relevant metrics. A verifiable metric is that the report exists.This test will be done manually by the team members of SyntaxSentinels.

A pathway for users to post or vote for requests/issues

1. test-M-2

   NFR: MS-S3

   Type: Inspection

   Initial State: The user has a GitHub account and is logged in

   Input: User posts a request or issue

   Output: The request or issue is posted and visible to other users

   How test will be performed: The user will post a request or issue and verify that it is visible to other users. A verifiable metric is that the request or issue is visible to other users. This test will be done manually by the team members of SyntaxSentinels.

Model Code follows Template

1. test-M-3

   NFR: MS-A2

   Type: Inspection, Static

   Initial State: N/A

   Input: Source code pertaining to machine learning model is passed to a parsing script

   Output: The output of the parsing script states model components all follow templates

   How test will be performed: Source code pertaining to machine learning model will be statically checked (through a parsing script looking

for reserved class words or ordering potentially) to see if model contains components/layers that inherit an interface/class. If all components/layers are found to inherit an interface/class, it is assumed it will be simple to create or exchange components/layers of the model in code to keep it up with research without major overhaul to the model code in its entirety (although there may be significant changes within individual components/layers). This will be integrated into a CI/CD pipeline and run periodically to ensure model code remains in adherence to templates.

Model Code follows Template

1. test-M-4

   NFR: MS-A1

   Type: Dynamic

   Initial State: System is inactive (cannot modify model during activity)

   Input: None into system directly, training script receives two formats of training sets

   Output: Training script successfully completes epoch

   How test will be performed: Training script for model provided will receive two different formats of training set data from a test script, and test script will verify training occurred either through inspecting a generated checkpoint file or a return value from the training script. This will be integrated into a CI/CD pipeline and run periodically to ensure compatibility for model training remains.

Model Code follows Template

1. test-M-5

   NFR: MS-A3

   Type: Manual, Unit

   Initial State: N/A

   Input: A set of appropriately dimensioned vectors into model component/layer

Output: Expected modification of vector according to component/layer architecture

How test will be performed: Every unique model layer/component will attempt to be separated from any preceding and proceeding layer/components, and will be copied into a test function where it will be given an input vector of appropriate size. The output function should apply transformations strictly associated with that layer and no other, which should be evident through output vector size and contents of the vector. Must be manually done as assessing how to separate layers may not be possible to be done in a consistent manner, even if layers/components are ultimately modular. The difference of having embedded layers that are not modular versus separable layers is akin to having z=f(g(x)) versus y=g(x) and z=g(y). Test must only be done every time a new type of model component or layer is added.

4.2.10 Security

These testcases ensure that the system is secure and that user data is protected.

User can access UI with valid login credentials

1. test-S-1

   NFR: SR-A1

   Type: Dynamic

   Initial State: The user is not authenticated and does not have a valid auth token.

   Input/Condition: User enters valid login credentials (username and password).

   Output/Result: The user is authenticated and gains access to the UI.

   How test will be performed: This test will be done via UI automated test suite using Playwright. The test will simulate a user entering valid login credentials and verify that the user is authenticated and gains access to the UI.

User can access API with valid auth token

1. test-S-2

   NFR: SR-A1

   Type: Dynamic

   Initial State: The user is not authenticated and does not have a valid auth token

   Input: User attempts to access the API with a valid auth token

   Output: The API allows the user to upload code for comparison

   How test will be performed: Pass a valid auth token to the API and verify that the system allows code upload and returns a success response.

2. test-S-3

   NFR: SR-A1

   Type: Dynamic

   Initial State: The user is not authenticated and does not have a valid auth token

   Input: User attempts to access the API with an invalid or missing auth token

   Output: The API denies access and returns an unauthorized response

   How test will be performed: Pass an invalid or missing auth token to the API and verify that the system denies access and returns an unauthorized response.

4.3 Traceability Between Test Cases and Requirements

Test ID	Requirements
test-FR-1	FR-1
test-FR-2	FR-2, FR-4, FR-5
test-FR-3	FR-3
test-FR-4	FR-6

Test ID	Requirements
test-FR-5	FR-7
test-FR-6	FR-7
test-FR-7	FR-8
test-FR-8	FR-8
test-FR-9	FR-9
test-LF-1	LF-AR1, LF-AR2, LF-AR3, LF-SR1, LF-SR2, LF-SR3, LF-SR4
test-LF-2	CR-SC3
test-UH-1	UH-E1, UH-E3, UH-L1, UH-UP1, UH-UP2, OE-P1
test-UH-2	UH-E2
test-UH-3	UH-PI2
test-V-1	MS-M1, OR-R2
test-PS-1	SR-P1
test-PS-2	SR-P2
test-PS-3	CR-L1, CR-L2, CR-L3, CR-S2, CR-S5
test-DS-1	MS-M3
test-DS-2	MS-S1, CR-SC1
test-D-1	UH-L2, OE-P2
test-PR-1	PR-SL1, PR-C1
test-PR-2	PR-SL2
test-PR-3	PR-RFT1
test-PR-4	PR-PA1, PR-PA2
test-PR-5	PR-SE1
test-OE-1	OE-IAS1
test-OE-2	OE-IAS2
test-OE-3	OE-IAS3
test-M-1	MS-M2
test-M-2	MS-S3
test-M-3	MS-A2

Test ID	Requirements
test-M-4	MS-A1
test-M-5	MS-A3
test-S-1	SR-A1
test-S-2	SR-A1
test-S-3	SR-A1

Table 6: Traceability Matrix

5 Unit Test Description

This section will not be filled in until after the MIS document has been completed.

References

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. Hazard analysis. https://github.com/SyntaxSentinels/SyntaxSentinels/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf, November 2024a.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. Module guide. https://github.com/SyntaxSentinels/SyntaxSentinels/blob/main/docs/Design/SoftArchitecture/MG.pdf, November 2024b.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. Module interface system. https://github.com/SyntaxSentinels/SyntaxSentinels/blob/main/docs/Design/SoftDetailedDes/MIS.pdf, November 2024c.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. System requirements specification. https://github.com/SyntaxSentinels/SyntaxSentinels/blob/main/docs/SRS-Volere/SRS.pdf, November 2024d.

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. User guide. https://github.com/SyntaxSentinels/SyntaxSentinels/blob/main/docs/UserGuide/UserGuide.pdf, November 2024e.

6 Appendix

6.1 Symbolic Parameters

Currently, there are no symbolic parameters in the document. It is possible that some will be added in the future, however, at this point in time, it is too early to determine what they will be.

Appendix — Reflection

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

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?

   While writing this deliverable, our team was able to collaboratively clarify and solidify our understanding of both the functional and non-functional requirements. This process helped us align on specific goals and create comprehensive test plans, which ensures that our testing will effectively verify that all key project requirements are met.

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

   A significant challenge we faced was the extensive amount of non-functional requirements (NFRs), with over 30 NFRs to address. Each NFR required us to develop a detailed test plan, specifying factors like type (e.g., functional, dynamic, manual), initial state, input/conditions, expected output/results, and test method. While the template helped streamline our process, the sheer volume of NFRs meant the documentation grew quickly, and managing this without sacrificing detail was challenging. We prioritized efficiency by dividing NFRs among team members and holding review sessions to ensure consistent quality and adherence to our testing criteria. This allowed us to maintain clarity

without being overwhelmed by the documentation demands. One of the main challenges we faced was the large amount of

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

    - **Dynamic Testing Knowledge**
        - **Online Courses and Tutorials**: We can learn dynamic testing techniques through structured courses on platforms like Udemy, Coursera, or LinkedIn Learning, which cover both functional and non-functional testing strategies.
        - **Hands-on Practice with Sample Projects**: Practicing dynamic testing on sample projects or real scenarios allows us to directly apply the techniques, reinforcing our understanding through application.
    - **Static Testing Knowledge**
        - **Tool-Specific Documentation and Tutorials**: Reading the documentation and tutorials for static testing tools (e.g. ESLint) helps us understand how to perform effective static code analysis.
        - **Webinars and Workshops**: Many companies offer webinars or workshops on static analysis, often with interactive demos that can help us learn the nuances of static testing.
    - **Automated Testing Tools**
        - **Training and Certification Programs**: Courses that cover automation tools (like Pytest) and their integration with CI/CD pipelines provide a solid foundation for us in automated testing.
        - **Building a Project with CI/CD Integration**: By implementing automated testing on a project using CI/CD tools like Jenkins or GitHub Actions, we can apply automation skills in real scenarios, enhancing both our tool knowledge and integration capabilities.

- **Security Testing Knowledge**
  - **Cybersecurity and Penetration Testing Courses**: Taking courses on platforms like Cybrary or Coursera offers insights into security testing practices, covering areas like penetration testing, vulnerability assessment, and secure code practices.
  - **Practice with Security Testing Tools**: We can use tools like OWASP ZAP or Burp Suite for hands-on security testing, which helps us identify vulnerabilities and ensure code security.
- **Performance Testing Knowledge**
  - **Tool-Specific Training (e.g., JMeter, Locust)**: Learning a performance testing tool through its official documentation, tutorials, or community guides helps us understand load testing, scalability testing, and performance profiling.
  - **Performance Testing Workshops or Certifications**: Many organizations provide certifications or workshops focused on performance testing methodologies and tool usage, which provide both theoretical knowledge and practical applications for us.
- **Test Case Management**
  - **Exploring Test Management Tools**: Familiarizing ourselves with tools like TestRail, Zephyr, or Jira for creating, organizing, and tracking test cases helps us manage our testing process efficiently.
  - **Learning Best Practices in Test Documentation**: By reading resources or taking tutorials on effective test case design and management, we can develop a structured, comprehensive approach to test case documentation.

- **Lucas Chen:** Security Testing Knowledge, Performance Testing Knowledge
- **Dennis Fong:** Automated Testing Tools, Security Testing Knowledge
- **Julian Cecchini:** Automated Testing Tools, Static Testing Knowledge

- **Mohammad Mohsin Khan:** Dynamic Testing Knowledge, Performance Testing Knowledge
- **Luigi Quattrociocchi:** Static Testing Knowledge, Test Case Management

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   Approaches for acquiring knowledge or mastering the skill has been identified above.

   - **Lucas Chen:**
     - **Security Testing Knowledge**
       - **Chosen Approach**: YouTube Tutorials and online courses
       - **Reason for Choice**: YouTube tutorials and online courses provide a flexible and accessible way to learn security testing concepts and tools. By watching tutorials and taking courses, I can gain a solid understanding of security testing practices and apply them effectively in our project.
     - **Performance Testing Knowledge**
       - **Chosen Approach**: Tool-Specific Training (e.g., JMeter, PostMan)
       - **Reason for Choice**: Tool-specific training allows me to focus on the performance testing tools that are most relevant to our project. By learning tools like JMeter or PostMan, I can gain practical skills that directly apply to our performance testing requirements.
   - **Dennis Fong:**
     - **Automated Testing Tools**
       - **Chosen Approach**: Online courses and textbooks
       - **Reason for Choice**: This approach was chosen because of the flexibility and convenience where I can learn at my own pace. Furthermore, online resources are usually more up to date than textbooks that may be old

  - **Security Testing Knowledge**
    * **Chosen Approach**: Youtube videos (theory) and online courses
    * **Reason for Choice**: For security testing, the theory behind the foundations of security are very important. Furthermore, online courses also provide the knowledge, and application of these concepts to learn more about security testing.

- **Julian Cecchini:**
  - **Automated Testing Tools**
    * **Chosen Approach**: Hands-on practice with sample projects
    * **Reason for Choice**: I previously examined online tutorials and articles which taught the use of jenkins and github actions. I feel the material gave me more comfort with the area but not everything completely snapped in place within my mind. Therefore, I think it would be very beneficial to make some arbitrary source code which I can make a minor CI/CD pipeline for to gain a sense for all the nitty-gritty details involved within the setup of automated testing.
  - **Static Testing Knowledge**
    * **Chosen Approach**: Tool-Specific Documentation and Tutorials
    * **Reason for Choice**: I have used linters and the like before, but have not explored them in-depth. I would enjoy to not just learn superficial/basic stuff that can be found in webinars or youtube videos but to really sink my teeth into niche capabilities of linters which could possibly benefit my capstone or personal projects. Therefore, I feel documentation would be the best route seeing that it should contain all the functionality of static analysis tools, including experimental or less commonly known/discussed aspects.

- **Mohammad Mohsin Khan:**
  - **Dynamic Testing Knowledge**

    * **Chosen Approach**: Hands-on practice with sample projects
    * **Reason for Choice**: Practicing dynamic testing directly on projects will allow me to apply testing techniques in real-world scenarios, strengthening my ability to identify issues as they arise. This hands-on approach will provide practical insights that go beyond theoretical learning and help me become proficient in testing under realistic conditions.
  - **Performance Testing Knowledge**
    * **Chosen Approach**: Tool-specific training
    * **Reason for Choice**: Tool-specific training allows me to gain direct experience with the performance testing tools that I will use in the project. This approach is ideal because it provides both a solid understanding of tool features and the technical skills needed to set up and execute performance tests effectively.

- **Luigi Quattrociocchi:**
  - **Static Testing Knowledge**
    * **Chosen Approach**: Tool-Specific Documentation and Tutorials
    * **Reason for Choice**: Reading and following documentation and tutorials specific to static testing tools, such as ESLint, should allow me to gain a more detailed understanding of how these tools work. This approach will also make it easier for me to learn best practices for effective code analysis within the actual tool environment we'll be using.
  - **Test Case Management**
    * **Chosen Approach**: Exploring Test Management Tools
    * **Reason for Choice**: Familiarizing myself with test management tools should provide me with direct experience in organizing, creating, and tracking test cases. This is especially useful for our project because having a structured approach to documentation should streamline collaboration with other team members as we conduct testing.