System Verification and Validation Plan for Software Engineering

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

November 3, 2024

Revision History

Date	Version	Notes
November 1	1.0	Initial documentation

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Khan et al., 2024d) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document outlines the strategies and processes used to ensure that the plagiarism detection system developed by the SyntaxSentinals 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:

- A user manual for instructors and administrators.
- Benchmarking of the tool's effectiveness compared to MOSS.

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 1: 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 2: 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.

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.

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

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

Control: Manual versus Automatic

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.2.1 Look and Feel Requirements

4.2.2 Usability and Humanity Requirements

4.2.3 Performance Requirements

Batch processing time is less than 10 minutes

1. Capacity-Test-1

   NFR: PR-C1

   Type: Dynamic

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

   Input:Submit a batch of inputs (code submissions) to the system for processing.

   Output: The system completes processing the batch within 600 seconds (10 minutes).

   How test will be performed:

   (a) Submit a batch of code submissions to the system.

   (b) Measure the time taken for the system to process the batch.

   (c) Verify that the processing time is less than 600 seconds.

Notification is sent when Processing Time Exceeds Ten Minutes

1. Speed-Latency-Test-1

   NFR: PR-SL2

   Type: Performance, Manual

   Initial State: System is idle; no processes are currently running.

   Input/Condition: Start a task that is expected to run for more than ten minutes.

   Output/Result: An email is sent to the user once processing exceeds ten minutes.

   How test will be performed:

   (a) Start a process that is known to take more than ten minutes to complete. For example, a large number of files.

   (b) Wait for 10 minutes while the process is running.

   (c) Check the user's email inbox for the notification and verify it was received exactly 10 minutes after the process started.

(d) Check that the notification for correctness and that it is clear and understandable for the user.

2. Speed-Latency-Test-2 NFR: PR-SL2

   Type: Performance, Automated

   Initial State: System is idle; no processes are currently running.

   Input: Initiate a new task that is simulated to take over ten minutes.

   Output: Automated system log entry indicating that a user notification was triggered after ten minutes.

   How test will be performed:

   (a) Setup a mock process to run for over 10 minutes.

   (b) Verify after 10 minutes were up wether a function call was made to send the notification.

   (c) Automatically check system logs or notifications queue to verify a notification was generated after ten minutes.

System remains operational after malformed input

1. Robustness-Test-1

   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 is able to interface with cloud computing services

1. Adjacent-Systems-Test-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. Adjacent-Systems-Test-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. Adjacent-Systems-Test-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.

4.2.4 Operational and Environmental Requirements

4.2.5 Maintainability and Support Requirements

Model release is accompanied by a report of metrics

1. Maintaince-Test-2

   NFR: MS-M2

   Type: Manual

   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. Maintaince-Test-3

   NFR: MS-S3

   Type: Manual

   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.

4.2.6 Security Requirements

User can access UI with valid login credentials

1. Security-Test-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. Security-Test-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. Security-Test-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.

4.2.7 Cultural Requirements

4.2.8 Compliance Requirements

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

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/lilweege/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/lilweege/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/lilweege/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/lilweege/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/lilweege/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.

6.2 Usability Survey Questions?

Currently, there are no usability survey questions 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.

4. **Lucas Chen:** Security Testing Knowledge, Performance Testing Knowledge
5. **Dennis Fong:**
6. **Julian Cecchini:**

7. **Mohammad Mohsin Khan:** Dynamic Testing Knowledge, Performance Testing Knowledge

8. 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?

    - **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:**
    - **Julian Cecchini:**
    - **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:**