System Verification and Validation Plan for SFWRENG 4G06 - Capstone Design Process

Team 17, DomainX

Awurama Nyarko
Haniye Hamidizadeh
Fei Xie
Ghena Hatoum

October 17, 2025

Revision History

Date	Version	Notes
Oct 17 2025	1.0	Initial Draft

Contents

1 Symbols, Abbreviations, and Acronyms

Table 1: Symbols, Abbreviations, and Acronyms

Symbol / Acronym	**Description**
API	Application Programming Interface – mechanism for data retrieval (e.g., GitHub API, PyPI API).
AHP	Analytic Hierarchy Process – method for pairwise comparison and ranking of libraries.
CSV	Comma-Separated Values – export format for datasets.
DB	Database – MySQL instance used for persistent data storage.
UI	User Interface – front-end component built with React.
NNL	Neural Network Libraries – the domain being analyzed (e.g., PyTorch, TensorFlow).
PoC	Proof of Concept – initial demonstration validating workflow integration.
VnV	Verification and Validation – process of ensuring correctness and meeting stakeholder needs.
CI/CD	Continuous Integration / Continuous Deployment – automated testing and deployment pipeline used in GitHub Actions.

All additional terms conform to those defined in the SRS Glossary (Section 4.1).

This document outlines the Verification and Validation (V&V) strategy for the Neural Network Libraries (NNL) Assessment Tool capstone project. It defines how the team will confirm that the implemented system satisfies its specified requirements, performs reliably, and aligns with the objectives stated in the Software Requirements Specification (SRS), Development Plan, and Hazard Analysis.

The V&V Plan provides a structured roadmap covering requirement reviews, system and nonfunctional testing, and traceability between test cases and requirements. It also establishes the methods, responsibilities, tools, and metrics that ensure all deliverables are verified for correctness and validated against user and research expectations.

2 General Information

2.1 Summary

The Neural Network Libraries (NNL) Assessment Tool is a web-based application that automates evidence collection, analysis, and visualization for assessing open-source neural-network libraries such as PyTorch and TensorFlow.

The tool replaces spreadsheet-based scoring with a traceable, auditable system that integrates the following automated features:

- **Automated Data Collection:** retrieval of repository metrics (commits, issues, languages, and lines of code) through the GitHub API and PyPI metadata.
- **Interactive Data Table:** centralized, editable interface for entering and verifying measurements.
- **Automated Analytic Hierarchy Process (AHP):** automated computation of pairwise comparisons for reproducible rankings.
- **Visualization and Export:** generation of graphs and reports in PNG and LaTeX formats for research use.

The V&V Plan defines how the software's correctness, reliability, and usability will be verified and validated throughout its lifecycle.

2.2 Objectives

The objectives of this V&V Plan are to:

1. Build confidence in software correctness by verifying that each functional requirement in the SRS (e.g., automated AHP calculation, data retrieval, visualization, and multi-user collaboration) is fully implemented and traceable to its tests.

2. Ensure system reliability and data integrity, particularly for automated data collection and storage operations identified as high-risk in the Hazard Analysis.

3. Validate usability and accessibility through internal surveys and pilot-user feedback from the Research Sub-team and Dr. Smith.

4. Confirm performance and security non-functional requirements, such as load handling and access-control validation.

Out of Scope:

- Third-party API correctness (GitHub, PyPI); assumed verified by their respective providers.

- Formal proof techniques and hardware-level testing; outside academic scope.

- Multi-browser performance optimization beyond the core compatibility tests specified in the SRS (Chrome, Firefox, Safari, Edge).

2.3 Challenge Level and Extras

Challenge Level: Advanced. Per the Problem Statement, the project integrates a custom software tool that automates data gathering, analysis, and visualization for neural-network libraries.

Although the broader capstone includes a research study on software-quality assessment, the scope of this V&V Plan pertains only to the verification and validation of the software tool itself, not to the accompanying research methodology or paper.

Extras (Approved by Supervisor): Usability Testing and Peer Code Reviews, conducted through structured user surveys and GitHub-based peer inspection checklists as defined in the Development Plan workflow.

2.4 Relevant Documentation

Table 2: Relevant Documentation and Their Relevance to V&V Activities

Document	**Relevance to V&V Activities**
Software Requirements Specification (SRS)	Defines functional and non-functional requirements that form the basis for test derivation and traceability.
Development Plan	Describes test environments, toolchains (PyTest, coverage.py, GitHub Actions), and team responsibilities used for verification automation.
Hazard Analysis	Identifies potential failures (e.g., data loss, access control errors, API failures) that inform stress and reliability tests.
Problem Statement & Goals	Clarifies project scope, stakeholders, and intended outputs to align validation activities with research objectives.
Design Document (MG/MIS – future)	Will provide module interfaces and algorithms for unit test mapping in Section **??**.

3 Plan

This section defines the overall verification and validation (V&V) strategy for the Neural Network Libraries (NNL) Assessment Tool.

It outlines the V&V team structure, review methods for each lifecycle artifact, verification tools, and validation activities that will be conducted to ensure functional and non-functional compliance with the SRS.

The plan follows both static reviews and dynamic execution-based testing, combining automated pipelines with peer-driven inspections.

3.1 Verification and Validation Team

Table 3: Verification and Validation Team Responsibilities

Team Member	**Role**	**Verification and Validation Responsibilities**
Awurama Nyarko	*Research Lead / Usability & Validation Analyst / Project Coordinator*	Coordinates all V&V activities, including scheduling artifact reviews and maintaining traceability between requirements, hazards, and test cases. Oversees usabil-

3.2 SRS Verification

Verification of the Software Requirements Specification (SRS) ensures that all requirements are consistent, unambiguous, complete, and testable.

The team will use a combination of peer inspection, supervisor walkthrough, and structured checklist verification to confirm requirement quality and traceability.

Each requirement identified in the SRS (e.g., SR-AC1, SR-INT2, SR-IM4) will be reviewed using a checklist based on the IEEE 29148 standard, evaluating criteria such as correctness, completeness, consistency, verifiability, and traceability.

During the scheduled peer inspection session, team members will mark any ambiguous or unverifiable requirements for revision through the GitHub issue tracker under the label "SRS Review."

Feedback from the assigned primary reviewer team will also be incorporated to ensure external validation of requirement clarity.

Following the peer inspection, a structured 30-minute walkthrough will be conducted with the supervisor.

In this meeting, the team will present the SRS structure, explain traceability between requirement identifiers and their planned test cases, and clarify any areas of uncertainty.

All decisions, clarifications, and follow-up actions from the session will be logged in the repository's issue tracker to maintain a transparent audit trail of SRS verification outcomes.

3.3 Design Verification

The design verification process ensures that the software architecture and module interfaces correctly realize all functional and non-functional requirements defined in the SRS.

A structured design review will be conducted after completion of the Module Guide (MG) and Module Interface Specification (MIS) documents.

The team will:

- Perform checklist-based inspections of class diagrams, API schemas, and data-flow diagrams to confirm interface correctness, modularity, and traceability to SRS requirements.

- Use the issue tracker to document any inconsistencies or missing data-flow connections.
- Review database and API integration points to verify that the design addresses identified hazards such as data loss and synchronization errors.
- Hold a design walkthrough with the supervisor, during which reviewers will ask task-based questions (e.g., "trace the path of a data update request") to confirm the design supports required behaviour.

The checklist will evaluate design completeness, consistency, interface clarity, and maintainability.

3.4 Verification and Validation Plan Verification

The V&V Plan itself will be verified to ensure internal consistency and feasibility.

Planned activities include:

- **Peer review session:** each team member will inspect the document against the V&V Plan template and rubric, checking for section completeness, alignment, and internal traceability.
- **Supervisor feedback:** Dr. Smith will review the plan draft and provide comments prior to submission.
- **Version control:** changes and responses will be tracked in GitHub using pull-request comments and a "VnV Plan Review" label.
- **Mutation check:** minor edits (for example, changing requirement identifiers) will be tested to confirm that traceability references remain correct.

3.5 Implementation Verification

Implementation verification ensures that each software component is correctly realized and adheres to design and coding standards.

- **Static verification:** code reviews will be conducted for all pull requests. Linters (`flake8`, `black` for Python, `ESLint` for JavaScript) will enforce syntax and style compliance (see *Development Plan*, Section 2).
- **Dynamic verification:** unit and integration tests written in PyTest will confirm correct module behaviour and interface compatibility.
- **Continuous integration:** GitHub Actions will automatically execute all tests on commit and generate coverage reports (target $\geq 80\%$).
- **Regression testing:** test suites will rerun automatically whenever code is merged to detect unexpected changes in functionality.
- **Code walkthroughs:** conducted at major milestones to compare implementation decisions with design diagrams and hazard mitigations.

3.6 Automated Testing and Verification Tools

Automated verification will rely on the toolchain defined in the Development Plan.

Table 4: Automated Testing and Verification Tools

Tool / Framework	**Purpose**
PyTest + `coverage.py`	Unit and integration testing with code-coverage metrics.
GitHub Actions CI/CD	Executes automated tests and generates build reports for every push or pull request.
`flake8` / `black` / `ESLint`	Static analysis and style enforcement for Python and JavaScript.
Selenium or Playwright	Automated front-end UI testing of key user workflows (domain creation and visualization).
Postman or `pytest-requests`	API endpoint validation and response-code checking.

Coverage summaries and logs will be archived in `/test/reports/` within the repository.

3.7 Software Validation

Software validation confirms that the delivered NNL Assessment Tool satisfies stakeholder expectations and performs reliably under normal operating conditions.

Validation will combine functional scenario testing, usability evaluation, and non-functional performance checks:

- **End-to-end validation:** execute representative workflows, from data collection through AHP ranking and visualization, to confirm system integrity.
- **Usability assessment:** conducted with internal research users using short post-demo surveys (questions provided in Appendix 6.2).
- **Performance validation:** stress-test database operations and API response times to verify thresholds stated in SRS Section 4.3.
- **Security validation:** attempt invalid inputs and authentication edge cases to ensure protection against hazards such as data leakage and unauthorized access.
- **Supervisor review:** a final demonstration and feedback session with Dr. Smith will confirm that the implemented system meets project objectives and research goals.

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 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

Output:

How test will be performed:

4.2.2 Area of Testing2

...

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 should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   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:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   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:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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?

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

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.

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?