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

Team 17, DomainX

Awurama Nyarko
Haniye Hamidizadeh
Fei Xie
Ghena Hatoum

October 23, 2025

Revision History

Table 1: Revision History

Date	Developer(s)	Change
October 17, 2025	Awurama, Fei	Initial v0 Draft

Contents

1 Symbols, Abbreviations, and Acronyms

Table 2: Symbols, Abbreviations, and Acronyms

Symbol / Acronym	
API	Application Programming Interface – mechanism for data retri
AHP	Analytic Hierarchy Process – method for pairwise c
CSV	Comma-Separated V
DB	Database – MySQL instan
UI	User Interface – fron
PoC	Proof of Concept – initial demonstrati
VnV	Verification and Validation – process of ensuring correctn
CI/CD	Continuous Integration / Continuous Deployment – automated testing and deploymen

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 n 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 Domain Assessment Tool is a web-based application that automates evidence collection, analysis, and visualization for assessing open-source libraries of various domains.

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 various domains.

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 3: Relevant Documentation and Their Relevance to V&V Activities

Document	
Software Requirements Specification (SRS)	Defines functional and non-functi
Development Plan	Describes test environments, toolchains (PyTest, coverage.py, Git
Hazard Analysis	Identifies potential failures (e.g., data loss,
Problem Statement & Goals	Clarifies project scope, stakeholders, ar
Design Document (MG/MIS – future)	Will prov

3 Plan

This section defines the overall verification and validation (V&V) strategy for the Domain 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

	Team Member	
	Awurama Konadu Nyarko	*Research Lead / Usability & Va*
	Fei Xie	*Scrum Mast*
	Haniye Hamidizadeh	*SRS Verificati*
	Ghena Hatoum	*Q*
	Dr. Spencer Smith	

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 5: Automated Testing and Verification Tools

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

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

3.7 Software Validation

Software validation confirms that the delivered Domain 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 7.2).
- **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

Test IDs follow the pattern `T-XX-#` where `XX` denotes the subsystem (AC = Access Control, DM = Domain Management, VR = Visualization, NF = Non-Functional).

4.1 Tests for Functional Requirements

This section describes the dynamic verification activities that will confirm the implemented system meets all functional requirements in the SRS.

Functional tests are grouped by feature areas that reflect the system's major use cases. All tests will be executed after integration and will rely on automated CI pipelines (`pytest`, GitHub Actions) supplemented by manual UI verification for user-facing features.

Each test case references the corresponding **SRS requirement identifier (SR–xx)** to maintain full traceability (see Section **??**).

4.1.1 Area of Testing 1: User Management and Access Control

Objective: Verify that user account creation, login, role assignment, and authentication operate correctly and protect system integrity.

Test ID	Description / Procedure
T–AC–1	Attempt sign-up and login with valid credentials through the UI and API.
T–AC–2	Attempt login with invalid password or unregistered email.
T–AC–3	Update user role (Viewer → Contributor) and verify access rights.
T–AC–4	Attempt unauthorized API call (e.g., delete domain without privilege).
T–AC–5	Verify password reset workflow (email token, form validation).

Table 6: Functional test cases for user management and access control.

Verification Approach:
White-box unit tests (`pytest`) will validate authentication and token logic.
Black-box UI tests (`Selenium`/`Playwright`) will simulate login, logout, and password reset flows.
Manual security validation will include injection and privilege-escalation attempts, referencing hazards identified in the Hazard Analysis (e.g., credential exposure, improper access).

4.1.2 Area of Testing 2: Domain and Data Management

Objective: Verify that domains, libraries, and metrics can be created, modified, and retrieved accurately, and that automated data collection and AHP ranking behave as specified.

Test ID	Descriptio
T–DM–1	Create a new domain and confirm database
T–DM–2	Add libraries (e.g., PyTorch, TensorFlow) to a domain and retrieve metadata fro
T–DM–3	Manually edit a metric and verify changes saved and reflected i
T–DM–4	Simulate API failure (limit exceeded) and verify fallb
T–DM–5	Export domain data (JSON / Excel) and ver
T–DM–6	Bulk upload data using Excel template and v
T–DM–7	Publish a completed domain for public

Table 7: Functional test cases for domain and data management.

Verification Approach:
Dynamic system tests executed using `pytest-requests` for API and database verification.
Automated UI tests confirm front-end state updates and form validation.
Error handling (rate limits, invalid uploads) tested manually and automatically, referencing hazards from the Hazard Analysis (data loss, API outage, invalid input).
Export and publishing workflows validated through integration testing and file-integrity checks.

4.1.3 Area of Testing 3: Visualization and Reporting

Objective: Verify that visualization components and export functionalities correctly display and summarize the analyzed data, ensuring results are accurate, interactive, and consistent with underlying datasets.

	Test ID	Desc
	T–VR–1	Generate comparison dashboard for at least 3 libra
	T–VR–2	Change visualization filter (e.g., da
	T–VR–3	Export visualized data to PDF/PNG
	T–VR–4	Resize browser window or test on differ
	T–VR–5	Simulate missing metric valu

Table 8 (continued)

	Test ID	Desc
	T–VR–6	Display complete metric definitions list fo

Verification Approach

- **Dynamic Tests:** Executed using `pytest-dash` for back-end data verification and Playwright or Selenium for front-end UI rendering.
- **Static Verification:** Visual inspection by the project team and supervisor to confirm readability, legend accuracy, and chart-type selection alignment with research goals.
- **Cross-Validation:** Exported results are compared with raw dataset values to ensure visual outputs match actual AHP scores.
- **Failure Handling:** UI behaviour under missing data and network interruptions is verified against Hazard Analysis controls (e.g., safe-state notifications).

4.2 Tests for Nonfunctional Requirements

This section specifies execution-based experiments (with summaries or plots rather than simple pass/fail where appropriate) and structured static reviews for the Domain Assessment Tool. The scope aligns with the SRS's nonfunctional sections (Usability, Performance, Robustness/Fault-Tolerance, Maintainability, Security) and with hazards identified in the Hazard Analysis.

Note on accuracy: For this project, "accuracy" concerns tool computations and outputs (e.g., AHP ranking consistency), not machine-learning model accuracy (explicitly out of scope). Relative error will be reported against a manual baseline as per the fit criterion.

4.2.1 Performance & Scalability Experiments

Goal: Measure how quickly the tool responds as the number of libraries and metrics increases.

Method (Automated Test):

- Record the time to load dashboards, apply filters, recalculate AHP scores, and export results.

- Run tests on datasets of 5, 10, 20, and 40 libraries.
- Save the median and 95th-percentile response times and plot them against dataset size.

Expected Output: A table and graph showing how response time grows with dataset size; any slowdowns will be noted for optimization.

Traceability: PR-SL1 (Speed and Latency Requirement).

Development Plan Tools: Automated using `pytest` and GitHub Actions performance workflows.

4.2.2 Reliability and Fault Tolerance

Goal: Confirm the system remains stable, consistent, and capable of recovery when services fail or operations are interrupted.

Method (Automated + Manual Tests):

- Simulate API errors (HTTP 429, 5xx) and verify that the tool uses cached data and displays a warning instead of crashing.
- Interrupt a mid-update transaction to ensure rollback to the last valid state.
- Perform a backup-and-restore test to confirm recovery within one hour.
- Verify that no partial or corrupted records are written during failures.

Expected Output: Concise test log summarizing behaviour under each error type, rollback event, and restore cycle, confirming that data integrity was preserved.

Traceability: PR-RFT1–PR-RFT3, SR-INT5.

References: Hazard Analysis — data loss and API outage risks.

4.2.3 Usability Evaluation

Goal: Check that the interface is intuitive, accessible, and that charts and exports are clear and easy to interpret.

Method (Manual Test + Survey):

- Have a small group of testers perform common tasks (create domain, add libraries, apply filters, export data).

- Record time taken, errors encountered, and number of clicks required per task.
- Ask participants to complete a short usability survey (Appendix B) rating ease of use, clarity, and navigation flow on a 1–5 scale.
- A mean rating of ≥ 4.0 across questions will meet the success criterion.

Expected Output: Average task completion times, click counts, and survey results summarized in a bar chart (Appendix C).

Traceability: UH-EU1 (primary navigation ≤ 3 clicks), UH-AR1 (font scalability), LF-AR2 (responsiveness across screens), LF-SR2 (minimalist graph design), UH-UP3 (clear error messages).

4.2.4 Computational Accuracy

Goal: Ensure that automatically computed AHP ranking results and auto-filled repository metrics are correct within an acceptable tolerance.

Method (Automated + Manual Comparison):

- Prepare a small AHP matrix and compute expected pairwise weights manually.
- Run the same inputs through the tool and compare outputs.
- For automatically fetched repository data, manually cross-check key fields (creation date, last commit date) against GitHub.
- Calculate relative error (%) for each metric; flag any deviation $> 2\%$.

Expected Output: Table comparing manual vs. tool-generated values, showing relative-error percentages and confirming compliance with thresholds.

Traceability: PR-PA1 (auto-filled data accuracy) and FR12 (AHP ranking computation).

4.2.5 Security and Access Control

Goal: Validate that user data, credentials, and permissions are properly secured, and that unauthorized actions are detected and prevented.

Method (Static + Dynamic):

- Perform a code review of authentication modules to confirm proper password hashing (e.g., `bcrypt` or `Argon2`) and secure session-token handling.
- Attempt unauthorized actions (e.g., deleting data without sufficient privileges, accessing admin-only routes) and verify they are blocked and logged.
- Review audit-log entries to ensure failed attempts are timestamped and attributed to the correct user.
- Confirm session-timeout and account lockout mechanisms activate after consecutive failed attempts.

Expected Output: Completed security-verification checklist, screenshots/logs of simulated intrusions, and a concise summary of results confirming no critical vulnerabilities.

Traceability: PR-SC1–PR-SC3, SR-AC6, SR-IM1, SR-AU1.
Also aligns with Hazard-Analysis controls for unauthorized access and data integrity risks.

4.2.6 Maintainability

Goal: Ensure that the codebase remains clean, modular, and easy to update, with consistent style and adequate automated-test coverage.

Method (Static + Automated):

- Run Python linters and formatters (`flake8`, `black`) to check for violations of the team's coding standards.
- Generate test-coverage reports from CI pipeline runs and record the overall percentage (target $\geq 80\%$).
- Conduct a brief team code walkthrough to review function naming, modular structure, and documentation completeness.
- Log all findings and proposed improvements in the GitHub issue tracker.

Expected Output: Lint-violation report, CI-coverage summary, and meeting notes capturing reviewer feedback and assigned follow-ups.

Traceability: MS-MR1–MS-MR6 (maintainability standards: style compliance, modularity, testability, documentation). Supports the Development-Plan verification process for maintainability compliance.

Format and Reporting Notes

- Accuracy tests use relative-error (%) instead of pass/fail.
- Performance and usability tests produce tables and graphs (Appendix C).
- Static reviews (walkthroughs, checklists) record who attended and what was found.
- All outputs are stored in the project's GitHub repository under `/test/reports/`.

4.3 Traceability Between Test Cases and Requirements

Test ID	Require
T–AC–1	SR
T–AC–2	SI
T–AC–3	
T–AC–4	S
T–AC–5	
T–DM–1	
T–DM–2	FR7, FR10, OE
T–DM–3	FR7,
T–DM–4	PR–R
T–DM–5	
T–DM–6	
T–DM–7	
T–VR–1	FR13
T–VR–2	
T–VR–3	
T–VR–4	L
T–VR–5	
T–VR–6	
T–NF1 (Performance & Scalability)	

Table 9 (continued)

Test ID	Require
T–NF2 (Reliability)	PR–R
T–NF3 (Usability)	UH–EU1, L
T–NF4 (Computational Accuracy)	
T–NF5 (Security)	PR–SC1–PR–SC3, S
T–NF6 (Maintainability)	

5 Unit Test Description

N/a

6 References

1. **V&V Plan Template (McMaster Capstone, Dr. Spencer Smith).**
   Source template defining section structure, verification guidelines, and notes for functional/nonfunctional testing. Used as the basis for all section headings and content organization.

2. **Software Requirements Specification (SRS) – DomainX.**
   Source of all requirement identifiers (FR, LF, UH, PR, MS, SR). Referenced throughout Sections 3 and 4 for test case traceability.

3. **Development Plan – DomainX.**
   Referenced for verification workflow, CI/CD tools (`PyTest`, `GitHub Actions`), and automation approach used in performance and maintainability testing.

4. **Hazard Analysis – DomainX.**
   Used for risk-based test design in reliability and security sections (API outage, rollback, and data-loss scenarios).

5. **Problem Statement and Project Objectives – DomainX.**
   Referenced for fit criteria in nonfunctional testing (accuracy $\pm 2\%$, usability $\geq 4/5$ score threshold) and for defining project scope (tool computation accuracy $\neq$ ML model accuracy).

7 Appendix

7.1 Symbolic Parameters

The following symbolic parameters are referenced throughout this document and may be updated centrally for consistency across all tests and scripts. Values marked as *team targets* are not defined in the SRS but used for validation thresholds.

Symbolic Constant	
MAX_LOGIN_ATTEMPTS	Number of consecutive failed login attempts befor
AHP_ERROR_THRESHOLD	Maximum acceptable relative error between manual and tool-calcu
USABILITY_TARGET_SCORE	Minimum average usability rating re
PERF_TEST_DATASETS	Dataset sizes used for performar
CI_COVERAGE_TARGET	Minimum required automated test cover

Table 10: Symbolic constants used in validation and verification tests. Values marked "team target" are not fixed by the SRS.

7.2 Usability Survey Questions?

These are the survey questions referenced in Section 4.2.3 (Usability Evaluation). Each question is rated on a 1–5 Likert scale (1 = Strongly Disagree, 5 = Strongly Agree).

1. The interface layout was intuitive and easy to navigate.
2. I could complete all main tasks (create domain, add libraries, export results) without external help.
3. The information displayed in charts and tables was clear and easy to understand.
4. The tool responded quickly to my actions (no major lag or errors).
5. Error messages were clear and helped me understand how to fix issues.
6. I found the overall design (colours, font size, contrast) visually comfortable.

7. I would feel confident using this tool again for a research project.

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?