System Verification and Validation Plan for Software Engineering

Team 21, Visionaries
Angela Zeng
Ann Shi
Ibrahim Sahi
Manan Sharma
Stanley Chen

October 28, 2025

Revision History

Date	Version	Notes
2025-10-27	1.0	Add initial draft
Date 2	1.1	Notes

[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 (Author, 2019) tables, if appropriate —SS]

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

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Extras

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

2.4 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

The Verification and Validation process for this project will involve both the planning and testing teams, as well as the project supervisor and members of the research group.

Planning team: Stanley Chen, Ann Shi, and Ibrahim Sahi will be responsible for developing review plans for SRS, VnV, and other documentation, as well as plans for design, implementation and software verification.

Testing team: Manan Sharma and Angela Zang will focus on developing and executing system tests that confirm the implementation satisfies the functional and non-functional requirements defined in the SRS.

The supervisor and research collaborators will provide support through reviewing and validating the planned systems.

3.2 SRS Verification

To ensure that the Software Requirements Specification is accurate, comprehensive, and aligned with the project's goals, a two-stage review process will be used: a preliminary peer review followed by a structured team review meeting.

Peer Review

An initial peer review will be conducted by classmates to identify issues in readability, organization, and completeness. This step helps ensure that the SRS communicates effectively to a general audience and that the flow, terminology, and formatting are clear.

Research Team Review Meeting

A review meeting will be held with the research team during the routine meeting time. This session will combine the technical, instructional, and managerial perspectives required to validate the SRS in a single coordinated effort. During the meeting, a condensed version of the SRS will be presented, highlighting key sections such as system scope, functional and non-functional requirements, datasets, technology dependencies and goals. The review will be guided using a task-based inspection, structured around each participant's perspective.

Instructors will verify that the system goals, use cases, and requirements align well with their needs as the primary users. Researchers will assess the logical soundness of the proposed workflows and ensure that data collection and analysis described in the SRS are technically consistent. The Supervisor will evaluate the overall feasibility of the system, ensuring that the listed technologies, environments, and resources are accessible and that the project remains within defined scope and constraints.

Feedback will be collected in real time and logged into the project's issue tracker for traceability.

3.3 Design Verification

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.5 Implementation Verification

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walk-throughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walk-through. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.7 Software Validation

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

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. TC-NFR-1-Latency
   Type: Dynamic, Automated

   Initial State: Both backend and frontend are running locally and the system is connected to mock eye tracking data source.

   Input/Condition: Conduct a 30 min, test session with simulated eye tracking and video data. Then record the timestamps of when data is recorded, processed and displayed.

   Output/Result: Record the average latency between capturing and visualizing data. Pass if latency is less than 1.5 s.

   How test will be performed: Log the timestamp when data is inputted, and when the dashboard is updated. Then compute the delay from the logs of timestamps.

2. TC-NFR-2-UpdateHz
   Type: Dynamic, Automated

   Initial State: Dashboard is rendering heatmaps in browser.

Input/Condition: For a 10 min session, with browser performance logs recording the render times.

Output/Result: Average and 95th percentile render frequency. Test will pass if the mean is greater than 20 Hz and the 95th percentile is greater than 18 Hz. This will include FPS histogram and timeline graph.

How test will be performed: The performance at each render will be recorded by using a Javascript probe, then the data will be summarized automatically in the command line.

3. TC-NFR-3-SyncDrift
   Type: Dynamic, Automated

   Initial State: 3 to 10 devices synchronized via Network Time Protocol (NTP), all streaming gaze and egoview data under standard classroom conditions.

   Input/Condition: Each device emits synchronization markers once per second during a session.

   Output/Result: Time-difference series between devices with maximum drift per minute recorded. Pass if drift stays within 20 ms.

   How test will be performed: Match sync markers by frame index across devices and compute drift statistics to verify requirement.

4. TC-NFR-4-JitterLoss
   Type: Dynamic, Automated

   Initial State: Live data streaming between devices and analytics back-end.

Input/Condition: Browser script sends WebSocket heartbeat every 100 ms while measuring round-trip delay.

Output/Result: Logs containing average, P95, and maximum jitter and a chart of packet loss vs jitter.

How test will be performed: Browser records timestamps for sent/received messages, computes latency variation, and summarizes results in dashboard or CI logs.

5. TC-NFR-16-Scale10
   Type: Load

   Initial State: Backend services running, synthetic gaze stream generators configured.

   Input/Condition: Simulate 1, 3, 5, and 10 devices concurrently for up to 20 minutes each, streaming gaze data at normal rates.

   Output/Result: CPU and memory usage, latency percentiles, and sync error recorded. Pass if sync error ¡ 20 ms at 10 devices. Produce plots of latency vs device count and sync error vs device count.

   How test will be performed: Use synthetic streams to simulate multiple devices while collecting system metrics; visualize results.

6. TC-NFR-5-AutoRecover
   Type: Dynamic, Automated

   Initial State: Multiple devices actively streaming gaze data to backend.

   Input/Condition: Intentionally disconnect each device stream for 10 seconds at randomized intervals, repeated 20 times.

Output/Result: Reconnection time for each disconnection; pass if all streams automatically reconnect without data loss or corruption.

How test will be performed: Scripted disconnections; backend verifies reconnection by checking frame sequence continuity and comparing checksums before/after reconnection.

7. TC-NFR-6-Uptime
   Type: Long-run observation

   Initial State: Full classroom analytics system deployed and running under normal conditions.

   Input/Condition: Conduct three scheduled 90-minute sessions across one week while tracking uptime metrics.

   Output/Result: Summary report with total uptime percentage, number of downtime incidents, and average recovery time.

   How test will be performed: Send a signal every 10 seconds; missed signals logged as downtime and recovery duration measured.

8. TC-NFR-17-TimeACID
   Type: Dynamic, Static

   Initial State: Database running and handling necessary operations.

   Input/Condition: Insert 10,000 events with known timestamps while performing multiple transactions.

   Output/Result: Timestamp drift within 33 ms and ACID properties maintained.

   How test will be performed: Compare database timestamps to ingestion clock and verify transactions commit/rollback correctly using transactional test suite.

9. TC-NFR-18-BigSession
   Type: Load test

   Initial State: System ready for large-scale data intake.

   Input/Condition: Process 50 x 200 GB of generated session data while running ingestion and analytics queries concurrently.

   Output/Result: Throughput and query latency remain stable; record any performance degradation or curves.

   How test will be performed: Batch load generated data, monitor CPU/memory (Prometheus), and measure query response times under increasing load.

10. TC-NFR-26-BackupRestore
    Type: Static, Dynamic

    Initial State: Backup configuration active and scheduled.

    Input/Condition: Trigger full system backup and restore into clean database instance.

    Output/Result: Verify encryption during backup and exact match of restored data; pass if checksums and verification logs match.

    How test will be performed: Activate backup workflow, restore data, and perform row-by-row hash comparison to confirm data integrity.

4.2.2 Usability and Accessibility

11. TC-NFR-7-Onboarding
    Type: Usability study, task-based inspection

    Initial State: Dashboard deployed and accessible to first-time users.

Input/Condition: New instructors complete key tasks (start session, view heatmaps, export data).

Output/Result: Users complete tasks within expected onboarding time; collect ease-of-use notes and feedback.

How test will be performed: Observe task completion times and collect short post-task surveys.

12. TC-NFR-8-Contrast
    Type: Static, Dynamic

    Initial State: Dashboard functional in web browser.

    Input/Condition: Run automated accessibility audits and manual checks under various lighting.

    Output/Result: UI elements meet WCAG AA contrast standards and are legible under classroom lighting.

    How test will be performed: Use Lighthouse or axe and verify results visually with contrast checkers.

13. TC-NFR-21-UCD
    Type: Iterative usability study

    Initial State: Dashboard prototype ready for user testing.

    Input/Condition: Conduct multiple usability sessions with instructors and TAs using realistic scenarios.

    Output/Result: Improvements in task completion rates, reduced errors, and higher satisfaction across iterations.

    How test will be performed: Run structured sessions, analyze feedback, and compare results across iterations.

4.2.3 Security and Privacy

14. TC-NFR-9-TLSOnly

    Type: Static, Dynamic

    Initial State: Application deployed with HTTPS proxy enabled.

    Input/Condition: Attempt requests using both HTTP and HTTPS.

    Output/Result: HTTP requests blocked or redirected; HTTPS succeeds with valid certificate.

    How test will be performed: Use curl and browser devtools to confirm redirects and certificate validity.

15. TC-NFR-10-RBAC
    Type: Dynamic

    Initial State: Backend roles configured for instructor and researcher accounts.

    Input/Condition: Send API requests under different roles to verify access control.

    Output/Result: Only authorized roles access protected endpoints; unauthorized requests rejected.

    How test will be performed: Use Postman or automated scripts with varied tokens to test role enforcement.

16. TC-NFR-11-AnonStore
    Type: Static, Dynamic

Initial State: Database populated with session data.

Input/Condition: Inspect stored session records and IDs.

Output/Result: No personal identifiers stored; only pseudonyms or session IDs used.

How test will be performed: Run SQL queries and manually review samples to ensure anonymization.

17. TC-NFR-24-Consent
    Type: Static, Dynamic

    Initial State: Application ready for a new recording session.

    Input/Condition: Attempt to start recording without completing consent process.

    Output/Result: Recording cannot start until consent explicitly confirmed.

    How test will be performed: Observe UI behavior and check consent status recorded in database.

4.2.4 Portability, Maintainability, and Process

18. TC-NFR-12-CrossPlat
    Type: Dynamic

    Initial State: System containerized and ready for deployment.

    Input/Condition: Run setup on Windows, macOS, and Ubuntu.

Output/Result: Application runs successfully with no critical functional issues on all platforms.

How test will be performed: Use Docker Compose or equivalent to deploy on each OS and verify key flows.

19. TC-NFR-13-Linters
    Type: Static

    Initial State: Source code repository available.

    Input/Condition: Run ESLint for JS/TS and flake8/ruff for Python.

    Output/Result: No linting errors or style violations reported.

    How test will be performed: Run linting tools in CI and fix or document any findings.

20. TC-NFR-14-CI
    Type: Process verification

    Initial State: Continuous integration (CI) pipeline configured.

    Input/Condition: Push a new pull request to trigger CI.

    Output/Result: The build, lint, and test stages run automatically and pass without errors.

    How test will be performed: Review CI logs and confirm successful artifact generation.

21. TC-NFR-15-Config
    Type: Static, dynamic

    Initial State: Configuration files and environment variables defined.

    Input/Condition: Modify configuration values without changing the code.

    Output/Result: Application updates behavior correctly based on new configurations.

How test will be performed: Edit .env values or config files and restart the service to verify changes take effect.

22. TC-NFR-19-License
    Type: Static

    Initial State: Dependency list available.

    Input/Condition: Run a license scan across all dependencies.

    Output/Result: All dependencies comply with NCRL-1.0 or compatible licenses.

    How test will be performed: Use tools like license-checker to scan and validate license information.

23. TC-NFR-20-NonComm
    Type: Static

    Initial State: Documentation and UI ready for review.

    Input/Condition: Review license text and all visible legal disclaimers.

    Output/Result: Academic use only or equivalent notice is clearly displayed in the documentation and UI.

    How test will be performed: Manually inspect the README, EULA, and app splash screen.

24. TC-NFR-23-EnvProtocol
    Type: Checklist, observation

    Initial State: Classroom setup complete.

    Input/Condition: Run the environmental calibration checklist.

    Output/Result: All requirements lighting, distance, calibration, and setup are met and documented.

    How test will be performed: Complete the checklist, take supporting photos, and attach them to the report.

25. TC-NFR-25-Comfort
    Type: Observation, survey

    Initial State: Session underway with active participants.

    Input/Condition: Gather short post-session feedback from participants.

Output/Result: Record comfort ratings using a Likert scale and collect optional comments.

How test will be performed: Conduct an anonymous feedback survey followed by a brief debrief session.

4.2.5 Research Validity

26. TC-NFR-22-Model
    Type: Dynamic, review

    Initial State: Analytics model implemented and trained.

    Input/Condition: Run the model on a labeled dataset with known ground truth.

    Output/Result: Model predictions closely match reference results; report scalar relative error and vector norm differences.

    How test will be performed: Compare model outputs with the reference dataset, calculate error metrics, and document findings, including any observed bias.

4.2.6 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

Author Author. System requirements specification. https://github.com/..., 2019.

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?
   - **Angela** ...
   - **Ann** ...
   - **Ibrahim** ...
   - **Manan** ...
   - **Stanley** ...
2. What pain points did you experience during this deliverable, and how did you resolve them?
   - **Angela** ...
   - **Ann** ...
   - **Ibrahim** ...
   - **Manan** ...
   - **Stanley** ...

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.

   - **Angela** ...
   - **Ann** ...
   - **Ibrahim** ...
   - **Manan** ...
   - **Stanley** ...

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?

   - **Angela** ...
   - **Ann** ...
   - **Ibrahim** ...
   - **Manan** ...
   - **Stanley** ...