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
Date 1	1.0	Notes
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

1 Symbols, Abbreviations, and Acronyms

Acronym	Description
ET	Eye-tracking: the process of measuring where a person is looking (the gaze point) using specialized hardware, such as wearable glasses.
NTP	Network Time Protocol: used to align timestamps across multiple devices, ensuring synchronized data streams.
CI/CD	Continuous Integration / Continuous Delivery: a software engineering practice involving automated building, testing, and deployment pipelines.
API	Application Programming Interface: defined methods and data formats that allow system components or external applications to communicate.
RBAC	Role-based Access Control: a security model that restricts system access based on user roles and permissions.
POC	Proof-of-Concept: the initial implementation phase (Rev 0) focusing on single-device operation, local data processing, and dashboard visualization.

This document outlines the Verification and Validation (VnV) plan for the group eye-tracking learning platform developed in this capstone project. The purpose of this plan is to establish a structured approach for verifying that the system has been built according to its specifications and validating that it meets its intended purpose. The plan will detail the methods, tools, and responsibilities involved in increasing the confidence of the developed software. It includes documentation of general information, team roles, verification procedures across the system's lifecycle stages (requirements, design, and implementation), and both system plus the eventual addition of unit-level test specifications. The document also defines the testing framework for functional and non-functional requirements, along with traceability between requirements, modules, and test cases.

2 General Information

2.1 Summary

The software being tested is an enhanced version of the **SocialEyes** framework, a system originally designed for recording and analyzing group gaze data. This project extends the framework to support **large-group eye-tracking in classroom environments**, implementing both post-session and real-time analytics for instructors to gain insights into their students and teaching.

The system consists of three primary components:

- **Instructor Dashboard with RBAC:** Presents gaze and engagement data in a clear, interpretable format.
- **Post-Session Analytics Module:** Provides aggregated insights to evaluate attention patterns and group interactions after class.
- **Real-Time Analytics Module:** Delivers live feedback to instructors during lectures to support adaptive teaching.

Together, these components aim to enhance instructors' understanding of student engagement and contribute to research on attention and collaboration in learning environments.

2.2 Objectives

This project aims to extend the existing SocialEyes system to support real-time session analytics through the integration of a lightweight computer vision algorithm that can efficiently map gaze points from egocentric to central views without compromising accuracy. Verification tests will focus on ensuring that the implemented models will perform correctly and consistently under real-time conditions.

Secondary objectives include the creation of the instructor dashboard for visualizing gaze data and providing analytics. Together, these components aim for easy interpretation of gaze-based feedback for instructors to provide useful feedback.

Certain objectives will intentionally be excluded due to project constraints. For instance, full verification of the underlying SuperPoint and SuperGlue models and other third-party libraries will not be performed, as these components are assumed to have been previously validated by their original developers. Additionally, large-scale usability testing with diverse participant groups is beyond the project's current timeframe and resources.

2.3 Extras

This project will include the following extra deliverables that were confirmed during the initial meeting with the project supervisors.

1. **Code Walkthrough Report**
   This will be a thorough documentation of the files and folders in the GitHub repository associated with the project.

2. **User Instruction Video**
   This will be an instructional video demonstrating how to interact with the dashboard deliverable of the project for each of the user roles (e.g., instructor).

2.4 Relevant Documentation

The Verification and Validation (V&V) Plan is dependent on and supported by several key project documents that define the foundation, scope, and success criteria for the group eye-tracking learning platform. These documents collectively guide how requirements are verified and validated throughout the

development process.

Software Requirements Specification (SRS)

The *Software Requirements Specification (SRS)* defines the functional and non-functional requirements of the system, serving as the primary reference for the verification process. It specifies what the system must do, how it should perform, and the constraints under which it must operate. Each verification test will be mapped to one or more requirements defined in the SRS.

Development Plan

The *Development Plan* outlines the project's implementation roadmap, including project decomposition, proof-of-concept demonstrations, and the technologies used to conduct it. It informs the scheduling and prioritization of verification and validation activities, ensuring that testing aligns with the overall development timeline and that each module is verified at appropriate stages in the lifecycle.

Problem Statement and Goals Document

The *Problem Statement and Goals Document* defines the motivation, context, and intended impact of the project. It provides the conceptual foundation for validation activities, ensuring that the developed system not only meets its technical specifications but also fulfills its educational and research goals. This document reinforces the relevance of the V&V Plan by providing a basis for why the tests being verified are crucial for achieving the project's goals.

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

To ensure that the system design aligns with the functional and non-functional requirements described in the SRS, the team will conduct a structured design review process composed of three complementary activities.

First, internal peer review sessions will be held to inspect the Modular Interface Specification (MIS) and system architecture diagrams using a prepared checklist. The checklist will focus on completeness, traceability, modularity, and adherence to design principles such as separation of concerns and single responsibility. Each module (for example, the data ingestion service, analytics engine, and instructor dashboard) will be verified to ensure its interface definitions are consistent with its described purpose and expected data flows.

Second, a cross-team review will be conducted with another capstone team to identify design ambiguities and potential integration issues from an external perspective. Feedback from this review will be recorded in the GitHub issue tracker for traceability.

Finally, the design documents, including component diagrams and data-flow models, will be presented to the project supervisors for validation of

research-specific considerations such as synchronization accuracy between multiple gaze streams and the interpretability of dashboard metrics.

Identified defects or improvement items will be categorized as documentation, logic, or scope issues, prioritized, and resolved before implementation. Evidence of completion will be stored as annotated PDFs and issue links within the project repository.

3.4 Verification and Validation Plan Verification

The verification and validation plan will also undergo verification to ensure that it is internally consistent, complete, and feasible given the project timeline. This will be achieved through a combination of peer inspection, supervisor validation, and cross-artifact consistency checks.

Classmates will conduct a checklist-based inspection to evaluate the clarity of objectives, completeness of planned tests, and consistency of terminology with the SRS. The supervisors will review the document to confirm that the chosen verification and validation techniques, such as unit testing, usability testing, and performance evaluation, are appropriate for both the engineering and research goals of the project.

In addition, the VnV Plan will be cross-checked against the Development Plan and Hazard Analysis documents to ensure that testing activities address identified risks such as privacy concerns or latency issues. Any inconsistencies found will be logged in the issue tracker under the label `type:docs` and corrected before the next revision. This approach ensures that the VnV Plan remains an up-to-date reflection of the team's testing progress.

3.5 Implementation Verification

Implementation verification ensures that the developed system conforms to the verified design and meets the requirements specified in the SRS. Verification will be performed through a combination of automated and manual activities.

Automated unit and integration tests will be created for each module of the system, including the backend API, real-time analytics processor, and

React dashboard. These will be implemented using PyTest for Python and Jest for JavaScript. Test coverage percentage and pass/fail status will be tracked through GitHub Actions to maintain continuous integration.

Static analysis and linting will be performed using ESLint, Prettier, and flake8 to enforce coding standards and detect syntax or style violations. Before Revision 0, the team will conduct a supervised code walkthrough with the project supervisor, where each developer will explain their module's implementation and trace its connection to the system requirements.

Each new feature merged into the main branch will automatically trigger a CI/CD pipeline that runs all tests to prevent regressions. In addition, static verification of documentation will be performed to detect outdated comments or broken references between source code and documentation.

3.6 Automated Testing and Verification Tools

Automation is a central part of the verification strategy, ensuring that every commit is tested consistently. The following tools and frameworks will be used throughout development.

At the end of each development iteration, the CI dashboard will export a summary of test coverage, lint violations, and failed cases. These results will be included in the final VnV Report as objective evidence of verification and overall system reliability.

3.7 Software Validation

The validation of the enhanced SocialEyes framework will primarily be conducted through regular review sessions and iterative demonstrations with project supervisors Dr. Lauren Fink and Dr. Irene Yuan. Weekly meetings will serve as checkpoints to assess progress against the requirements outlined in the Software Requirements Specification (SRS), ensuring that the evolving system aligns with both the technical and research goals of the project.

A formal Rev 0 demonstration will be presented to the supervisors once completed. This demo will be used to validate that the core system features—real-time gaze analytics, post-session data visualization, and the instructor dashboard—accurately reflect the intended requirements and provide value within

Table 1: Automated Testing and Verification Tools

Tool	**Purpose**	**Verification Activity**
GitHub Actions	Continuous integration pipeline that runs automatically on each pull request	Executes all test suites, linting, and build checks. Prevents merges if tests fail.
PyTest	Python testing framework for backend and data processing modules	Unit tests for data parsing, computation accuracy, and real-time gaze mapping
Jest and React Testing Library	Front-end testing for user interface components and data visualizations	Verifies correct rendering and state management on the instructor dashboard
ESLint, Prettier, and flake8	Static code analysis and style enforcement tools	Ensures consistent syntax, formatting, and code quality
Valgrind or cProfile	Profiling and memory analysis tools	Identifies performance bottlenecks and inefficiencies in real-time data handling
Coverage.py and Codecov	Code coverage tracking utilities	Measures test completeness and identifies untested portions of the codebase
Docker	Reproducible environment for testing and deployment	Verifies installability and portability of the full system across different machines

a classroom context. Feedback from this session will guide subsequent iterations and refinement of the system. In addition to supervisor reviews, user validation will be obtained through feedback from a postdoctoral researcher affiliated with the SocialEyes project who has experience conducting lectures and an interest in applying this technology to teaching contexts. This will help validate the system's usability and relevance from an instructor's perspective.

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?