

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

Contents

1	Symbols, Abbreviations, and Acronyms	iv
2	General Information	1
2.1	Summary	1
2.2	Objectives	1
2.3	Extras	1
2.4	Relevant Documentation	2
3	Plan	2
3.1	Verification and Validation Team	2
3.2	SRS Verification	2
3.3	Design Verification	3
3.4	Verification and Validation Plan Verification	4
3.5	Implementation Verification	5
3.6	Automated Testing and Verification Tools	5
3.7	Software Validation	7
4	System Tests	7
4.1	Tests for Functional Requirements	7
4.2	Tests for Functional Requirements	7
4.2.1	Area of Testing 1 - Data Acquisition and Ingestion	8
4.2.2	Area of Testing 2 - Data Processing and Analytics	11
4.2.3	Area of Testing 3 - Data Management and Backend Services	14
4.2.4	Area of Testing 4 - Visualization and Dashboard Interface	16
4.2.5	Area of Testing 5 - Supporting Infrastructure and Stretch (Optional)	19
4.3	Tests for Nonfunctional Requirements	21
4.3.1	Area of Testing1	22
4.3.2	Area of Testing2	22
4.4	Traceability Between Test Cases and Requirements	22
5	Unit Test Description	24
5.1	Unit Testing Scope	24
5.2	Tests for Functional Requirements	24
5.2.1	Module 1	24

5.2.2	Module 2	25
5.3	Tests for Nonfunctional Requirements	25
5.3.1	Module ?	26
5.3.2	Module ?	26
5.4	Traceability Between Test Cases and Modules	26
6	Appendix	27
6.1	Symbolic Parameters	27
6.2	Usability Survey Questions?	27

List of Tables

1	Automated Testing and Verification Tools	6
	[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] ex [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

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 pre-

pared 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.

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

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

[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

4.2 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]

This subsection provides concrete, executable tests for all Functional Requirements (FRs) in the SRS. Tests are grouped by subsystem: Data Acquisition and Ingestion; Data Processing and Analytics; Data Management and Backend Services; Visualization and Dashboard Interface; and Supporting Infrastructure (and Stretch). Each test specifies Control, Initial State, Input, Output (expected result), Test Case Derivation, Test Data / Artefacts, and How the test will be performed.

4.2.1 Area of Testing 1 - Data Acquisition and Ingestion

Note: This area covers FR-1, FR-2, FR-3, FR-5, FR-6, and FR-19 from the SRS (operational modes, instructor session control, device capture, synchronization, recovery, and pre-session calibration).

Mode Switching and Session Control

1. FR-1-T1

Control: Automatic

Initial State: System services running; no active session; device simulator available.

Input: (1) From dashboard, start a Recording session and run for 10 s. (2) Switch to Streaming mode via a UI toggle while the session is active.

Output: Session state transitions succeed; no crash; data continuity preserved with no inter-packet gap greater than 100 ms around the switch.

Test Case Derivation: FR-1 requires support for Recording and Streaming operational modes to enable both offline analysis and real-time visualization.

Test Data / Artefacts: Synthetic gaze stream (JSON/Parquet), egoview video (MP4/MKV), session state logs (LOG/CSV).

How test will be performed: A UI test script triggers the state changes. A log parser computes inter-packet gaps from ingest timestamps and asserts the maximum gap is less than or equal to 100 ms.

2. **FR-2-T1 - Instructor Session Controls (Start/Pause/Resume/Stop)**

Control: Manual + Automatic

Initial State: Instructor role authenticated; dashboard open on Sessions page.

Input: Click Start, then Pause after 5 s, then Resume after 5 s, then Stop.

Output: UI status changes Active → Paused → Active → Stopped. No data are stored during the pause interval.

Test Case Derivation: FR-2 mandates instructor control to ensure intentional and ethical recording.

Test Data / Artefacts: Test user accounts (CSV/YAML), RBAC policy (YAML), session metadata export (CSV), event logs (LOG).

How test will be performed: Tester performs the sequence in the UI while a script queries session metadata and verifies there are no gaze or frame records timestamped within the pause window.

3. **FR-3-T1 - Device Data Acquisition via External API**

Control: Integration (Automatic)

Initial State: Eye-tracking device or simulator connected and reachable via API.

Input: Issue Start Recording.

Output: Synchronized gaze samples and egoview frames are received at expected rates and timestamped.

Test Case Derivation: FR-3 specifies interfacing with supported eye-tracking devices to collect synchronized gaze and video.

Test Data / Artefacts: Device API mock responses (JSON), captured video (MP4/MKV), gaze samples (JSON/Parquet), device/config settings (YAML).

How test will be performed: An integration test hits the device API simulator, records 30 s, and checks that sample and frame counts and timestamps are within expected tolerances.

4. **FR-5-T1 - Multi-Device Time Synchronization**

Control: Automatic

Initial State: Two device simulators streaming; NTP service active.

Input: Emit a visible synchronization cue at time t_0 .

Output: Detected cue timestamps differ by no more than 20 ms across the two devices.

Test Case Derivation: FR-5 requires time-aligned data streams for valid group analysis.

Test Data / Artefacts: Dual-stream video with sync cue (MP4), NTP/sync logs (LOG), cue-detection script output (CSV).

How test will be performed: A detector finds the cue frame in both streams and compares timestamps after offset correction.

5. **FR-6-T1 - Recovery After Transient Network Loss**

Control: Automatic

Initial State: Active streaming session (video and gaze).

Input: Introduce a 2 s network drop and then restore the link.

Output: Stream resumes within 5 s of link restoration; the session remains a single continuous session.

Test Case Derivation: FR-6 requires resumption within 5 s for short connectivity issues.

Test Data / Artefacts: Network shaping profile (SH/TC config), ingest/application logs (LOG), optional packet capture (PCAP), timing summary (CSV).

How test will be performed: A traffic shaper introduces the drop. The harness measures restoration and resume times and asserts resume within 5 s.

6. **FR-19-T1 - Automatic Calibration Check Before Recording**

Control: Automatic

Initial State: Device connected; no active session.

Input: Click Start Recording.

Output: If calibration score is below threshold, recording is blocked and the user is prompted to recalibrate. If the score meets threshold, recording starts and a CalibrationPassed event is logged.

Test Case Derivation: FR-19 prevents poor data due to miscalibration.

Test Data / Artefacts: Calibration result payloads (JSON), UI evidence (PNG screenshot), calibration/decision logs (LOG).

How test will be performed: Mock the device API to return failing and passing calibration states; verify gating and log entries.

4.2.2 Area of Testing 2 - Data Processing and Analytics

Note: This area covers FR-7, FR-8, FR-9, FR-10, FR-20, and FR-24 from the SRS (filtering and correction, projection to a shared space, real-time metrics, post-session analytics, off-screen filtering, and replay).

Analytics Correctness and Latency

1. FR-7-T1 - Noise and Blink Filtering

Control: Automatic

Initial State: Offline dataset with labeled blink and noise segments.

Input: Run processing pipeline on the dataset.

Output: At least 95% of labeled blink/noise samples are removed and no more than 2% of valid samples are removed incorrectly.

Test Case Derivation: FR-7 requires filtering and correction to ensure reliable metrics.

Test Data / Artefacts: Labeled blink/noise dataset (CSV), raw gaze stream (Parquet/CSV), filter pipeline configuration (YAML), accuracy report (CSV).

How test will be performed: Compare algorithm output masks to labels; compute precision and recall; assert thresholds.

2. FR-8-T1 - Gaze Projection to Shared Coordinate Space

Control: Automatic

Initial State: Egoview frames and central-view frames with visible fiducial markers.

Input: Run the projection to map egocentric gaze to central coordinates.

Output: Mean projection error is less than 3% of the central-view width; 95th percentile error is less than 10%.

Test Case Derivation: FR-8 requires accurate projection for group alignment.

Test Data / Artefacts: Egoview frames (PNG/JPG), central-view frames (PNG), fiducial/scene map (JSON), projected gaze coordinates (CSV).

How test will be performed: Compute 2D error against fiducial ground truth and assert thresholds.

3. FR-9-T1 - Real-Time Metric Latency

Control: Automatic

Initial State: Live stream active; dashboard metrics panel visible.

Input: Stream a scripted gaze trajectory for 60 s.

Output: Metric widgets (velocity, entropy, normalized contour area) update with end-to-end delay less than or equal to 1 s and an effective refresh rate near 20 Hz.

Test Case Derivation: FR-9 specifies real-time computation to support instructional feedback.

Test Data / Artefacts: Synthetic trajectory specification (JSON), backend metric logs (LOG), frontend render timestamps (CSV), browser/version note (TXT).

How test will be performed: Backend stamps metric times; frontend logs render times; a script computes latency and effective rate.

4. FR-20-T1 - Off-Screen and Invalid Gaze Exclusion

Control: Automatic

Initial State: Dataset with labeled off-screen intervals.

Input: Run real-time and post-session analysis modules.

Output: At least 98% of off-screen points are flagged and excluded from aggregates; no flagged points appear in heatmaps.

Test Case Derivation: FR-20 requires excluding invalid gaze to prevent misinterpretation.

Test Data / Artefacts: Off-screen ground-truth labels (CSV), raw gaze stream (Parquet/CSV), exclusion flags (CSV), generated heatmap (PNG).

How test will be performed: Compute confusion matrix against labels; verify heatmap generator omits flagged points.

5. **FR-10-T1 - Post-Session Summary Generation**

Control: Automatic

Initial State: One completed recording with known scripted attention shifts.

Input: Invoke Generate Summary.

Output: Exported CSV, JSON, and PNG include: session heatmap, time-series of engagement, and participant-level statistics that match an independently recomputed reference within plus or minus 2%.

Test Case Derivation: FR-10 requires generation of post-session analytics.

Test Data / Artefacts: Completed session data (CSV/Parquet), system exports (CSV/JSON/PNG), independent recomputation outputs (CSV/JSON).

How test will be performed: An independent script recomputes metrics from raw data; differences are compared and asserted.

6. **FR-24-T1 - Replay and Re-Analysis Consistency**

Control: Manual + Automatic

Initial State: Stored session available in the replay view.

Input: Play the session from start to finish and trigger Re-run Analytics.

Output: Event timing during replay matches original within less than 50 ms; recomputed metrics differ by less than 1% from the original.

Test Case Derivation: FR-24 enables verification and debugging using replay.

Test Data / Artefacts: Replay media (MP4/MKV), original metric series (CSV), recomputed metric series (CSV), event markers (CSV).

How test will be performed: Compare timestamped events and metric series to the original outputs; assert thresholds.

4.2.3 Area of Testing 3 - Data Management and Backend Services

Note: This area covers FR-11, FR-12, FR-13, FR-22, and FR-23 from the SRS (persistence, metadata, APIs, traceability, and configurability).

Persistence, Metadata, APIs, and Configurability

1. FR-11-T1 - Data Persistence Across Restarts

Control: Automatic

Initial State: Clean database; services running.

Input: Record 2 minutes; stop; restart services; load the session.

Output: Raw frames, gaze samples, and derived analytics are present and readable after restart.

Test Case Derivation: FR-11 requires local storage of raw and processed data.

Test Data / Artefacts: Database schema (SQL), database snapshot (SQL dump), session directory (mixed files as produced).

How test will be performed: An integration script records, restarts the stack, and then queries by session ID to validate counts and checksums.

2. FR-12-T1 - Session Metadata Completeness

Control: Automatic

Initial State: None.

Input: Start and stop a session with two participants.

Output: The session record contains timestamp, session ID, and participant IDs; each data row references the session ID.

Test Case Derivation: FR-12 mandates metadata for traceability.

Test Data / Artefacts: Session and participant schemas (DDL/CSV), example record export (CSV), integrity check script (SQL).

How test will be performed: SQL assertions check required columns and foreign key presence.

3. **FR-13-T1 - Programmatic API for Analytics Retrieval**

Control: Automatic

Initial State: One completed session with analytics.

Input: GET /api/sessions/{id}/analytics and POST /api/session/start.

Output: HTTP 200 with schema-valid JSON for analytics; start endpoint returns a new session ID.

Test Case Derivation: FR-13 requires APIs for analytics retrieval and session management.

Test Data / Artefacts: API specification (OpenAPI YAML), API test collection (JSON), example analytics response (JSON).

How test will be performed: An automated API test validates response codes and JSON schema.

4. **FR-22-T1 - Traceability From Raw to Reports**

Control: Automatic

Initial State: Completed session.

Input: Trace query joining raw, processed, and report tables on session ID.

Output: Complete lineage from raw to processed to export; no orphan rows.

Test Case Derivation: FR-22 requires maintained traceability via unique session identifiers.

Test Data / Artefacts: Traceability query (SQL), table extracts for raw/processed/reports (CSV), lineage visualization (PNG/PDF).

How test will be performed: SQL join plus referential integrity checks; fail on nulls or orphans.

5. **FR-23-T1 - Configuration Changes Take Effect**

Control: Automatic

Initial State: Default configuration applied.

Input: Change data directory, export format, and device ID in configuration; restart.

Output: Next session writes to the new directory; exports match the configured formats; device routing uses the new ID.

Test Case Derivation: FR-23 enables flexible deployment without code changes.

Test Data / Artefacts: Configuration file (YAML/TOML), directory listings (TXT), export manifest (JSON).

How test will be performed: Start a session and verify filesystem paths, export set, and device mapping.

4.2.4 **Area of Testing 4 - Visualization and Dashboard Interface**

Note: This area covers FR-14, FR-15, FR-16, FR-26, and FR-27 from the SRS (RBAC, real-time visual analytics, exports, status indicators, and spatial masking/anonymization).

Access, Status, Exports, and Privacy

1. **FR-14-T1 - Role-Based Access Control**

Control: Automatic

Initial State: Two users exist: instructor and unauthorized.

Input: Both users attempt to access Live Analytics and Export views.

Output: Instructor gains access; unauthorized user receives 403 Forbidden; no data leakage occurs.

Test Case Derivation: FR-14 enforces authentication and authorization for sensitive data.

Test Data / Artefacts: Test user list (CSV), RBAC policy (YAML), audit log (LOG), denied-access summary (CSV).

How test will be performed: API and UI tests simulate both roles; backend audit logs are checked for denied attempts.

2. **FR-15-T1 - Real-Time Visual Analytics Render**

Control: Automatic

Initial State: Live stream; dashboard analytics tab open.

Input: 60 s of synthetic gaze with known hotspots.

Output: Heatmap tiles and engagement widgets update within 1 s after data arrival and reflect hotspots in the correct regions.

Test Case Derivation: FR-15 requires real-time visual analytics on the dashboard.

Test Data / Artefacts: Synthetic hotspot definition (JSON), UI render timestamps (CSV), reference heatmap (PNG).

How test will be performed: DOM probes read widget timestamps; a reference heatmap is computed independently and compared with the rendered result.

3. **FR-16-T1 - Export Formats and Fidelity**

Control: Manual + Automatic

Initial State: Completed session with analytics.

Input: User exports CSV, JSON, and PNG.

Output: Files exist; CSV and JSON schemas are valid; the PNG heatmap visually matches the dashboard with structural similarity index greater than 0.95.

Test Case Derivation: FR-16 requires export of post-session analytics in common formats.

Test Data / Artefacts: CSV schema (CSV schema file), JSON schema (JSON schema file), exported outputs (CSV/JSON/PNG), dashboard render reference (PNG).

How test will be performed: A schema validator checks CSV and JSON; an SSIM comparison is run on the PNG versus a server-side render.

4. **FR-26-T1 - Status Indicators (Latency, Jitter, Drift)**

Control: Automatic

Initial State: Live session; status panel visible.

Input: Inject synthetic 150 ms network jitter and 10% packet loss for 20 s.

Output: Indicators reflect increased latency and loss within two refresh cycles; synchronization drift increases accordingly.

Test Case Derivation: FR-26 requires real-time system status indicators.

Test Data / Artefacts: Network impairment profile (SH/TC config), status endpoint samples (JSON), metrics time-series (CSV).

How test will be performed: Traffic shaping is applied; displayed metrics are sampled via test hooks and compared to injected conditions.

5. **FR-27-T1 - Spatial Masking and Anonymization**

Control: Manual + Automatic

Initial State: Privacy mask polygon configured to exclude a region.

Input: Record 30 s with participants present in the masked area; export heatmap and a representative frame.

Output: No identifiable faces or eyes appear in the masked region; exported data for masked pixels are absent or blurred.

Test Case Derivation: FR-27 requires definable exclusion zones for privacy in storage and export.

Test Data / Artefacts: Mask configuration (JSON), exported frame (PNG), heatmap (PNG), face-detector output (CSV).

How test will be performed: An automated face detector confirms zero detections inside the mask; a manual spot-check verifies exports.

4.2.5 Area of Testing 5 - Supporting Infrastructure and Stretch (Optional)

Note: This area covers FR-18 (logging/monitoring) and Stretch FR-4, FR-17, FR-21, FR-25 (Rev 1+ goals).

Observability, Roles, Privacy Pipeline, and Scalability

1. FR-18-T1 - Runtime Logging and Metrics

Control: Automatic

Initial State: Logging level = INFO; metrics collector enabled.

Input: Run a 2 min session with one induced warning (device reconnect).

Output: Log contains timestamped events for start/stop, warnings, and metrics snapshots; metrics series include FPS, jitter, and dropped frames.

Test Case Derivation: FR-18 requires logging diagnostics and performance metrics for monitoring.

Test Data / Artefacts: Logging configuration (YAML), runtime logs (LOG), metrics export (CSV), event list (CSV).

How test will be performed: Parse logs and metrics endpoint; assert presence and cadence.

2. FR-4-T1 - Central Camera Integration (Stretch)

Control: Integration (Automatic)

Initial State: Central-camera RTSP source available.

Input: Start session with central-view enabled; two egoviews active.

Output: Central view frames are timestamped and aligned with egoviews; the projection module accepts the central feed without error.

Test Case Derivation: FR-4 optionally interfaces with a central camera for shared-scene alignment.

Test Data / Artefacts: RTSP stream configuration (TXT), central-view video (MP4), alignment results (CSV), error logs (LOG).

How test will be performed: RTSP ingest with timestamp alignment check; verify projection module operates with the central frames present.

3. **FR-17-T1 - Role-Based View Separation (Stretch)**

Control: Manual + Automatic

Initial State: Two roles exist: Instructor and Researcher.

Input: Log in as each role and open the dashboard.

Output: Instructor view hides researcher-only analytics (and vice versa); access attempts outside role return 403 Forbidden.

Test Case Derivation: FR-17 provides separate role-based views to enhance usability and compliance.

Test Data / Artefacts: Role/view configuration (YAML), UI screenshots (PNG), endpoint access matrix (CSV), audit logs (LOG).

How test will be performed: UI inspection combined with API role assertions and audit log checks.

4. **FR-21-T1 - Privacy Anonymization Pipeline (Stretch)**

Control: Manual + Automatic

Initial State: Example frames containing identifiable faces and eyes; anonymization enabled.

Input: Export video frame and heatmap with anonymization on.

Output: No identifiable faces or eye images remain; face detector confidence below threshold throughout; heatmap contains no pixel data from masked regions.

Test Case Derivation: FR-21 ensures ethical compliance and participant anonymity in exports.

Test Data / Artefacts: Example frames for anonymization (PNG), anonymized outputs (PNG), detector results (CSV), privacy compliance checklist (PDF/MD).

How test will be performed: Automated face-blur verification plus manual review of a sample of exports.

5. **FR-25-T1 - Multi-Device Scaling (Stretch)**

Control: Automatic

Initial State: Ten device simulators configured.

Input: Start streaming all 10 concurrently for 5 min.

Output: Inter-device synchronization drift stays within ± 20 ms; no ingest backpressure beyond configured queue thresholds; no service crash.

Test Case Derivation: FR-25 targets scaling to ≥ 10 devices while maintaining synchronization accuracy.

Test Data / Artefacts: Multi-device simulator configuration (YAML), system metrics (CSV), synchronization drift summary (CSV), queue depth logs (CSV).

How test will be performed: Synthetic load plus synchronization measurement as in FR-5-T1; report drift statistics and queue depths.

4.3 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]

4.3.1 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:

4.3.2 Area of Testing2

...

4.4 Traceability Between Test Cases and Requirements

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

Functional Requirements Traceability Matrix:

FR (SRS)	Test Case ID(s)	Area
FR-1	FR-1-T1	A1: Acquisition & Ingestion
FR-2	FR-2-T1	A1: Acquisition & Ingestion
FR-3	FR-3-T1	A1: Acquisition & Ingestion
FR-4	FR-4-T1	A5: Support & Stretch (Stretch)
FR-5	FR-5-T1	A1: Acquisition & Ingestion
FR-6	FR-6-T1	A1: Acquisition & Ingestion
FR-7	FR-7-T1	A2: Processing & Analytics
FR-8	FR-8-T1	A2: Processing & Analytics
FR-9	FR-9-T1	A2: Processing & Analytics
FR-10	FR-10-T1	A2: Processing & Analytics
FR-11	FR-11-T1	A3: Data Mgmt & Backend
FR-12	FR-12-T1	A3: Data Mgmt & Backend
FR-13	FR-13-T1	A3: Data Mgmt & Backend
FR-14	FR-14-T1	A4: Viz & Dashboard
FR-15	FR-15-T1	A4: Viz & Dashboard
FR-16	FR-16-T1	A4: Viz & Dashboard
FR-17	FR-17-T1	A5: Support & Stretch (Stretch)
FR-18	FR-18-T1	A5: Support & Stretch
FR-19	FR-19-T1	A1: Acquisition & Ingestion
FR-20	FR-20-T1	A2: Processing & Analytics
FR-21	FR-21-T1	A5: Support & Stretch (Stretch)
FR-22	FR-22-T1	A3: Data Mgmt & Backend
FR-23	FR-23-T1	A3: Data Mgmt & Backend
FR-24	FR-24-T1	A2: Processing & Analytics
FR-25	FR-25-T1	A5: Support & Stretch (Stretch)
FR-26	FR-26-T1	A4: Viz & Dashboard
FR-27	FR-27-T1	A4: Viz & Dashboard

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, your 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** Writing this deliverable went fine since I focused on mapping the FR tests to requirements. Having the SRS finalized helped make the traceability and test design process straightforward.
 - **Ann** ...
 - **Ibrahim** ...
 - **Manan** ...
 - **Stanley** The SRS was settled, so this went quickly. I mapped FR tests to modules and kept the write-up tight to match the template.
2. What pain points did you experience during this deliverable, and how did you resolve them?
 - **Angela** The main challenge was ensuring the test cases aligned correctly with the requirements, especially when some FRs were broad. I resolved this by reviewing our SRS revision and confirming test coverage with the team.

- **Ann** ...
 - **Ibrahim** ...
 - **Manan** ...
 - **Stanley** Hardest part was choosing the right level of detail. I cut jargon and reorganized 3.3–3.6 so they read cleanly.
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** I will need to strengthen my understanding of test automation and CI workflows to help verify FR-level functionality efficiently.
 - **Ann** ...
 - **Ibrahim** ...
 - **Manan** ...
 - **Stanley** I need stronger CI skills to tie PyTest and Jest into one pipeline, and practice writing small, maintainable tests.
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** I plan to review CI documentation for GitHub Actions and explore automated testing frameworks like PyTest as it supports the test workflows we will implement later in the project.
 - **Ann** ...
 - **Ibrahim** ...
 - **Manan** ...
 - **Stanley** Two approaches: (1) study a minimal GitHub Actions template and adapt it; (2) build tiny test jobs in our repo and iterate. I'm starting with (2).