System Verification and Validation Plan for FitCoachAR

Seyedmohamad Mirhosseininejad

April 21, 2026

Contents

List of Tables

Revision History

Date	Version	Notes
Feb 12, 2026	1.0	Initial Plan based on Presentation Strategy
Mar 03, 2026	1.1	Detailed validation process and systematic reviews added
Mar 12, 2026	1.2	Addressed Yibing Mei's Peer Review feedback (traceability, dataset, and formatting)
Apr 15, 2026	1.3	Final-doc pass: fixed requirement-to-test traceability matrix to match SRS R1–R5; corrected section heading to R_Verify; noted T3 numbering gap
Apr 15, 2026	1.4	Final Documentation release for CAS 741
Apr 15, 2026	1.5	Rubric pass: added TOC and List of Tables; expanded acronyms table; added SRS hyperlinks; expanded non-dynamic testing (code review, walkthroughs, traceability audit); added Research Level and Extras section; expanded Unit Test section; fixed LaTeX formatting
Apr 20, 2026	1.6	Addressed Yibing Mei feedback (issue #28): added R1–R5 system-test coverage table to align with the VnV Report.
Apr 20, 2026	1.7	Addressed Dr. Smith's NFR1 feedback: rewrote Accuracy as a-posteriori and added T10 (Dynamic Backend Benchmark).

1 Symbols, Abbreviations, and Acronyms

symbol	description
CI	Continuous Integration
DD	Data Definition (from the SRS)
E2E	End-to-End (test)
FSM	Finite State Machine
GD	General Definition (from the SRS)
IM	Instance Model (from the SRS)
MAE	Mean Absolute Error
MG	Module Guide
MIS	Module Interface Specification
NFR	Non-Functional Requirement
PoC	Proof of Concept
R1–R5	Functional requirements (R_Input, R_Process, R_Calc, R_Verify, R_Output) defined in
SRS	Software Requirements Specification
T	Test case (e.g. T1 = Test case 1)
TM	Theoretical Model (from the SRS)
VnV	Verification and Validation

This document outlines the Verification and Validation (VnV) plan for the FitCoachAR system.

2 General Information

2.1 Summary

FitCoachAR is a browser-based exercise feedback system that uses a computer-vision pose-estimation library (MediaPipe) to extract joint landmarks and an FSM (IM_RepetitionCount) to count repetitions and classify exercise form. Because the scientific core of the system is the geometric computation of joint angles and the phase-based counting logic, verification of those two concerns is the primary purpose of this document.

2.2 Objectives

The primary objectives of this VnV plan are to:

- Build confidence in the correctness of the **repetition counting logic**.
- Verify the **accuracy** of the biomechanical angle calculations.

2.3 Relevant Documentation

Mirhoseininejad (2026) provides the functional and non-functional requirements that will be verified.

2.4 Scope

In scope. Verification of the scientific-computing core of FitCoachAR: joint-angle geometry (GD_KneeAngle, GD_ElbowAngle), exercise validity instance models (IM_SquatValid, IM_BicepCurlValid), and the repetition counting state machine (IM_RepetitionCount). Validation against labelled exercise data is also in scope.

Out of scope. Usability testing is deferred to a future iteration; the PoC deliverable targets scientific-computing correctness, not end-user experience. Hardware-specific camera-driver verification is also out of scope because FitCoachAR relies on the browser's `getUserMedia` API, which is treated as a trusted external component.

3 Plan

3.1 Verification and Validation Team

- **Primary developer** — Seyedmohamad Mirhosseininejad. Responsible for writing every test case in this document, running the test suite (`pytest`) and the system-level demonstrations, maintaining the labelled validation dataset, and keeping this plan in sync with the SRS and the implementation.
- **Peer reviewer** — Yibing Mei. Responsible for the structured GitHub-issue review of the SRS, VnV Plan, MG, and MIS, and for independent walkthrough of selected test cases.
- **Instructor** — Dr. Spencer Smith. Responsible for annotated review of the plan and the VnV Report, and for the final grading of the deliverables.

3.2 SRS Verification and Validation

The SRS has been verified and validated through a systematic plan to ensure the requirements align with both software engineering standards and real-world fitness coaching needs. This plan includes:

- **Scenario-Based Walkthroughs**: Conducting structured reviews with the assigned Domain Expert (Yibing Mei) to trace specific exercise scenarios (e.g., a perfect squat vs. a shallow squat) through the Instance Models, ensuring the biomechanical constraints map to actual coaching expectations.
- **Peer Review**: Asynchronous evaluation by the assigned primary and secondary reviewers from CAS 741 via structured GitHub Issues.
- **Checklist-Based Verification**: Methodical verification of Goal Statements and Input Constraints utilizing the CAS 741 SRS Checklist.

3.3 Design Verification

The Design (MG and MIS) will be verified through:

- Peer review (GitHub Issues) by the assigned domain expert and secondary reviewers.
- Comparison against the SRS utilizing a Traceability Matrix to ensure all requirements are mapped to corresponding modules.

3.4 Implementation Verification

Implementation verification combines dynamic testing (automated and manual) with several forms of non-dynamic verification to ensure correctness beyond what test suites alone can establish.

3.4.1 Dynamic Testing

- **Automated unit and system tests.** The primary developer runs the full `pytest` suite (Section 4 and Section 6) before every merge to `main`. GitHub Actions CI re-runs the suite on every push to catch regressions.
- **Manual system-level demonstration.** For test cases that involve live camera input (e.g. PoC inspection for R1 and R5), the developer performs a hands-on walkthrough using the running application in a browser.

3.4.2 Static Verification and Non-Dynamic Evidence

- **Code Review.** Every non-trivial change to a scientific-computing module (M3–M6, M8) is reviewed via a GitHub Pull Request before merge. Reviewers check algorithm correctness, adherence to the MIS interface contracts, and defensive handling of edge inputs (e.g. collinear points in `compute_angle`).
- **Static Analysis / Linting.** Automated linting via `flake8` and `pylint` is enforced in CI; warnings must be resolved before merge. This catches unreachable code, unused imports, and type-mismatch issues that unit tests would not cover.
- **Document Review.** The SRS, MG, MIS, and VnV Plan are each reviewed by the peer reviewer (Yibing Mei) and the instructor (Dr.

Smith) via annotated PDFs or GitHub Issues. Feedback is tracked and resolved in the Reflect and Trace document.

- **Walkthrough / Inspection.** The MG/MIS presentation to the CAS 741 class serves as a group walkthrough of the module decomposition and the interface specification. Questions raised during the presentation are treated as review items and tracked as GitHub Issues.

- **Traceability Audit.** A manual cross-document consistency check is performed before the Final Documentation deadline: requirement IDs (R1–R5), module IDs (M1–M8), test case IDs (T1–T9, T_M3–T_M8), and assumption labels are verified to match across the SRS, MG, MIS, VnV Plan, and VnV Report (see Section 5).

3.5 Automated Testing and Verification Tools

The following tools are used:

- **Pytest**: Unit and system test execution for the backend modules.
- **GitHub Actions**: CI pipeline that runs the `pytest` suite and `flake8` on every push to `main`.
- **Makefile**: The project `Makefile` provides `make test` as a single-command entry point to run the full test suite locally.

3.6 Software Validation

- **Dataset**: Validation utilizes a custom recorded dataset consisting of 50 exercise videos. The dataset includes a diverse distribution of correct form, incorrect depth, incorrect angles, varying user heights, and different camera angles to ensure coverage of realistic usage scenarios. Each video has been manually labelled frame-by-frame indicating whether the user is exhibiting correct or incorrect physical form.
- **Validation Approach**: Compare system output against the labelled datasets.
- **Method**: Run FitCoachAR on the dataset and compare the expected (labelled) outputs against the actual system feedback.

- **Oracle**: The labelled dataset serves as the oracle—it provides the "known truth" for validation.

4 System Tests

4.1 Tests for Functional Requirements

The system tests defined here cover the five functional requirements R1–R5 from the SRS. For each requirement, the verification approach is named and — where a dynamic test case exists — the corresponding test identifier is listed. Detailed results for each requirement appear in Section 3 of the VnV Report.

R1, R2, and R5 are primarily architectural / UI-layer requirements. R1 is verified by dynamic boundary testing of its only software boundary (M3); R2 is verified indirectly through T8/T9 (since MediaPipe landmark extraction is a third-party dependency, its own correctness is out of scope); R5 is verified by inspection because automated browser-rendering tests require a headless-browser harness that is not in scope for this iteration. Only R4 has a dedicated block of dynamic system-test cases, detailed below.

4.1.1 Exercise Validity and Repetition Counting (R_Verify, R4)

Note: Test identifier T3 was reserved during planning but consolidated into T2 during execution; the gap in numbering is intentional.

1. T1: Perfect Squat
   Type: Functional, Dynamic, Automated Simulation
   Initial State: System Initialized
   Input: Synthetic cosine wave of knee angles ($180° \rightarrow 90° \rightarrow 180°$).
   Expected Output and Final State: `RepCount` increments by 1. Final state is Top. State transitions: Top $\rightarrow$ Down $\rightarrow$ Bottom $\rightarrow$ Up $\rightarrow$ Top.
   Test Case Derivation: Tests the happy path of a full valid repetition.
   How test will be performed: Automatically feed angle data to the state machine function using pytest.

2. T2: Failed Rep
   Type: Functional, Dynamic, Manual Simulation
   Initial State: System Initialized

Req.	Name	Verification Approach	Test(s)
R1	R_Input	Boundary-layer dynamic test of the video formatter (M3): valid Base64 frame accepted; malformed / empty / non-data-URI input rejected.	T_M3
R2	R_Process	Indirect verification: landmark extraction is exercised by geometric accuracy test T8, which consumes 3D point triples derived from MediaPipe output. Dynamic occlusion handling exercised by T9.	T8, T9
R3	R_Calc	Analytic verification of `KinematicEngine.compute_angle` against closed-form ground truth for 5 reference geometries.	T8
R4	R_Verify	State-machine dynamic tests (see §4.1.1).	T1, T2, T4, T5, T6, T7
R5	R_Output	Inspection during Proof-of-Concept demonstration: skeletal overlay and feedback text verified visually on the live AR canvas; feedback strings also asserted in T1 and T2.	PoC demo + T1, T2

Table 1: System-Test Coverage by Functional Requirement

Input: Angle dips to 100° only (never reaches 90° threshold).
Expected Output and Final State: `RepCount` does NOT increment. Feedback: "Go Deeper". Final state is Top.
Test Case Derivation: Tests the threshold logic (Boundary Value Anal-

ysis).
How test will be performed: Manually feed angle data to the state machine function.

3. T4: State Machine Cycle (IM_RepetitionCount)
   Type: Functional, Dynamic, Manual Simulation
   Initial State: System Initialized (Bottom State, $p < 0.15$), where p is the normalized scalar progress of the exercise ($0.0 \rightarrow 1.0$).
   Input: Sequence of p values: $0.1 \rightarrow 0.5 \rightarrow 0.9 \rightarrow 0.5 \rightarrow 0.1$ with $\Delta t = 2.0s$.
   Expected Output and Final State: `RepCount` increments by 1. Transitions: Start $\rightarrow$ Ascend $\rightarrow$ Peak $\rightarrow$ Descend $\rightarrow$ Complete. Final State: Complete.
   Test Case Derivation: Verification of the 5-stage state machine logic.
   How test will be performed: Feed normalized progress data to the `analyze_form` function.

4. T5: Timing Constraint (Too Fast)
   Type: Functional, Dynamic, Manual Simulation
   Initial State: System Initialized (Top State)
   Input: Sequence of progress values representing a full rep: $1.0 \rightarrow 0.0 \rightarrow 1.0$. The simulated timestamps for these frames are $t_0 = 0.0s, t_1 = 0.1s, t_2 = 0.2s$ (where $t_{min} = 0.5s$).
   Expected Output and Final State: `RepCount` does NOT increment. A warning "Too Fast" is logged. Final State: Top State.
   Test Case Derivation: Verification of usage constraints defined in IM_RepetitionCount.
   How test will be performed: Feed data with fast timestamps directly into the rep counter.

5. T6: Intermediate State Transition (Ascend)
   Type: Functional, Dynamic, Automated Simulation
   Initial State: Start State ($p = 0.1$)
   Input: Normalized progress increases to $p = 0.6$.
   Expected Output and Final State: The state machine transitions to the Ascend state. `RepCount` does not increment.
   Test Case Derivation: Verification that the intermediate ascending threshold triggers correctly.
   How test will be performed: Automatically pass p values into the state machine and assert the returned state Enum.

6. T7: State Reversal (Failed Transition)
   Type: Functional, Dynamic, Automated Simulation
   Initial State: Ascend State ($p = 0.6$)
   Input: Normalized progress decreases back to $p = 0.1$ instead of continuing to Peak.
   Expected Output and Final State: The state machine transitions back to the Start state. `RepCount` does not increment.
   Test Case Derivation: Verification that the state machine handles reversals and does not erroneously count incomplete movements.
   How test will be performed: Automatically pass p values into the state machine and assert the returned state Enum.

4.2 Tests for Nonfunctional Requirements

4.2.1 Accuracy (NFR1)

NFR1 requires calculated angles to be within $\pm 5^\circ$ of visual ground truth. The plan verifies this by measuring accuracy against ground truth rather than asserting it. One test case is not enough to justify an accuracy claim for a pose-estimation pipeline, so three tests are used at different layers of the system. T8 isolates the kinematic engine from the pose backend using analytical inputs. T10 is a dynamic cross-backend benchmark that captures real pose noise on recorded video. T9 checks behaviour under occlusion. The results from all three are carried over into the VnV Report.

1. T8: Geometric Verification (Analytical Dataset)
   Type: Non-Functional, Dynamic, Automated
   Initial State: System Initialized
   Input: A set of five reference three-point configurations spanning $[0^\circ, 180^\circ]$ (right angle, straight, acute, obtuse, 3D) with closed-form ground-truth angles.
   Expected Output and Final State: Each computed angle matches the closed-form ground truth to within $\pm 0.5^\circ$; mean absolute error reported.
   Test Case Derivation: isolates `compute_angle` from the pose backend, so that any residual error seen in T10 can be attributed to the backend and not to the math.
   How test will be performed: Automated pytest case (`test_vnv.py::test_geometric_accurac` over the five reference configurations.

2. T9: Dynamic Occlusion Verification (Video Dataset)
   Type: Non-Functional, Dynamic, Automated
   Initial State: System Initialized
   Input: A labeled video dataset containing 10 scenarios where the subject is temporarily occluded by an object.
   Expected Output and Final State: The system identifies the occluded frames via the backend's per-landmark visibility score and gracefully skips them for angle extraction without crashing; error reported only on the visible-frame subset.
   Test Case Derivation: accuracy also needs to be checked under imperfect conditions, not only on the clean set. This is the occlusion counterpart to T10's stability measurement.
   How test will be performed: Automated script processes the occluded videos and asserts both the skip behaviour and the error bound on visible frames.

3. T10: Dynamic Backend Benchmark (Recorded Video Dataset)
   Type: Non-Functional, Dynamic, Automated
   Initial State: System Initialized
   Input: A recorded video dataset of short squat and bicep-curl clips captured at multiple camera pitches (eye-level, 45°-down) and orientations (front, front-45°, side, back-45°, back), replayed through each of the three candidate pose backends (`mediapipe_2d`, `mediapipe_3d`, `movenet_3d`).
   Expected Output and Final State: For each backend, per-frame latency and the mean absolute frame-to-frame angle delta $\overline{|\Delta\theta|} = \frac{1}{N-1}\sum_{i=1}^{N-1}|\theta_i - \theta_{i-1}|$ are reported for the knee and elbow channels. The production backend is chosen to minimise $\overline{|\Delta\theta|}$ subject to the real-time latency budget. The chosen backend must satisfy the NFR1 $\pm 5°$ bound on the T8 analytical set and keep elbow $\overline{|\Delta\theta|} \leq 5°$ on the recorded dataset.
   Test Case Derivation: measures accuracy a posteriori with more than one test case, which is what the review feedback asked for. The metric $\overline{|\Delta\theta|}$ is a stability proxy. There is no per-frame ground truth on live video, so this is not an error measurement. It does put a limit on how much information the downstream Kalman smoother (M8) has to discard. Full results are in VnV Report §4.1.
   How test will be performed: Offline benchmark script replays the recorded dataset through each backend and emits a CSV of latency and $\overline{|\Delta\theta|}$ per

backend and per joint.

5 Traceability Matrices

Table 2 shows the traceability between the requirements from the SRS and the system tests outlined in this document. Each SRS requirement is covered by at least one test case or an explicit PoC-demo verification; each test case traces back to exactly one requirement to keep the mapping bi-directional.

Requirement	**Test Cases**
R1 (R_Input)	Covered by PoC demo (manual verification of webcam capture)
R2 (R_Process)	T9
R3 (R_Calc)	T8, T9, T10
R4 (R_Verify)	T1, T2, T4, T5, T6, T7
R5 (R_Output)	T1, T2 (feedback text); PoC demo (overlay)
NFR1 (Accuracy)	T8 (analytical), T9 (occlusion), T10 (dynamic benchmark)

Table 2: Traceability Matrix: Requirements to Tests

6 Unit Test Description

6.1 Unit Testing Scope

Unit testing focuses on the two modules whose secrets carry the highest scientific risk: **M5 (Kinematic Engine)** and **M6 (Exercise State Machine)**. These modules are pure Python and have no dependency on the webcam, the browser, or the WebSocket layer, which makes them straightforward to exercise from `pytest`. The hardware-hiding module M1 and the display module M2 are not unit-tested because their behaviour is defined by browser APIs outside the project's control; they are covered instead by PoC-demo inspection.

The unit-test implementation lives in `src/backend/tests/test_vnv.py` and a small interface conformance harness in `src/backend/test_interfaces.py`; both are executed by `make test`.

6.2 Unit Tests for Functional Requirements

- **T_M5 (Angle computation, R_Calc).** Feeds `KinematicEngine.compute_angle` a suite of hand-derived three-point configurations (collinear, right angle, obtuse, degenerate/near-zero vector) and asserts the output is within $\pm 0.5^\circ$ of the analytic answer. Covers the "norm $<$ 1e-6" guard branch.
- **T_M6 (State machine, R_Verify).** Drives `ExerciseStateMachine.update` with synthetic progress traces (happy path, failed dip, stuck phase, timing-too-fast) and asserts the resulting `RepState`. This is the same fixture exercised at the system level by T1, T2, T4, T5, T6, and T7.
- **T_M8 (Smoothing, supporting R_Process).** Streams a noisy signal through `KalmanLandmarkSmoother` and asserts the smoothed output is strictly closer to the ground-truth signal (MAE reduction $\geq 30\%$) than the raw input.
- **T_M3 (Video formatting, R_Input).** Encodes and decodes a small synthetic frame via the Base64 JPEG path and asserts the round trip is lossless up to JPEG quantization.

6.3 Unit Tests for Nonfunctional Requirements

Non-functional accuracy is verified at the system level by T8 and the offline benchmark in the VnV Report; no dedicated unit tests are planned for NFR1 because the metric (MAE against a goniometer-labelled dataset) is inherently end-to-end.

7 Research Level and Extras

The following items extend the core VnV activities and have been approved by the instructor as part of the Problem Statement:

- **Machine Learning Report.** A comparative benchmark of different pose-estimation backends (MediaPipe, MoveNet, HRNet) evaluating the trade-off between inference latency and geometric accuracy. The benchmark is run against the Fit3D dataset and the results are reported in the VnV Report.

- **CI Pipeline.** A GitHub Actions workflow that executes the full `pytest` test suite and linting on every push, and a second workflow that compiles all LaTeX documents and deploys PDFs to GitHub Pages.

References

Mohamad Mirhoseininejad. System requirements specification. https://mmdmirh.github.io/cas741/SRS/SRS.pdf, 2026.

8 Appendix

N/A

Appendix — Reflection

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? Planning the "Manual Unit Tests" helped clarify the logic before writing code.

2. Pain points? Determining how to verify accuracy without an expensive motion capture system. Resolved by using static goniometer measurements.

3. Skills needed? Knowledge of Python `pytest` for future automation and Pose Estimation constraints.