System Verification and Validation Plan for Software Engineering

Team 13, Speech Buddies
Mazen Youssef
Rawan Mahdi
Luna Aljammal
Kelvin Yu

October 27, 2025

Revision History

Date	Version	Notes
Oct 26, 2025	1.0	Added Section 3
Oct 27, 2025	1.0	Added Sections 1,2, and 4.1
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

The V&V plan described in this document applies to the VoiceBridge application. VoiceBridge is an accessibility tool that is built around speech-to-text, designed for individuals with impaired or atypical speech caused by neurological or motor speech disorders. Using a customizable model, it converts slurred or non-standard speech into accurate text and control commands on standard consumer devices. The system integrates seamlessly with commonly used technology, including web browsers, enabling open-ended, customizable control for tasks such as online shopping, browsing, and digital communication. By allowing users to operate everyday applications with their own speech patterns, the product promotes greater autonomy, accessibility, and independence.

2.2 Objectives

The primary objective of VoiceBridge is to accurately capture and interpret the intended meaning of speech from users with impaired or atypical articulation, ensuring that their spoken input is reliably translated into the correct text or device actions. A critical focus is the safe and precise mapping of user intent, particularly when controlling web browsers and other general-purpose software environments where unintended or harmful actions could have significant consequences. Because the system is dependent on non-deterministic AI models, robust safeguards, validation layers, and command verification mechanisms will be implemented to prevent misinterpretation, accidental activation, or malicious exploitation. The project also strives to maintain usability and accessibility by operating on standard consumer hardware and providing clear, transparent feedback to the user. Ultimately, the goal is to empower users with meaningful autonomy while maintaining strict control, reliability, and safety in all system interactions.

2.3 Challenge Level and Extras

Challenges not applicable to V&V plan.

2.4 Relevant Documentation

Below is a list of documents that can be referenced for further information, as well as short descriptions highlighting their relevance to V&V

- Problem Statement & Goals: This document summarizes the core objectives of VoiceBridge and the problem it aims to address. These initial goals lay the foundation for the subsequent requirements that the V&V aims to ensure.
- Software Requirements Specification: This document goes into depth about the project specifications (user identification, business cases) as well as the functional and non-functional requirements that the V&V validates through the outlined testing plan.
- Design Document (To be written): This document will summarize the core modules of VoiceBridge, highlighting the components that needed to be directly and indirectly tested, as defined in the V&V.

3 Plan

This section describes how the VoiceBridge team will verify and validate the system throughout its lifecycle. It outlines responsibilities, reviews, and testing activities that ensure each stage, from the SRS to it's implementation. The plan emphasizes practical verification aligned with our project scope.

3.1 Verification and Validation Team

- **Kelvin Yu:** Lead Tester. Coordinates overall V&V activities and integration testing across the ASR, intent mapping, and browser components.
- **Mazen Youssef:** Functional Tester. Verifies core functional requirements (FR1–FR5) including audio capture, transcription, and command execution.

- **Rawan Mahdi:** UI and Usability Tester. Tests the front-end interface for accessibility, feedback clarity, and overall user experience.
- **Luna Aljammal:** Non-Functional Tester. Assesses performance, latency, and privacy compliance; maintains testing documentation and traceability.
- **Dr. Christian Brodbeck (Supervisor):** Oversees the V&V process, providing feedback on testing and alignment with project goals.

3.2 SRS Verification

3.2.1 Requirements Validation

Each functional and non-functional requirement will be verified for clarity, feasibility, and measurability. A formal SRS Review Checklist will be applied to Sections 10–16, focusing on fit criteria, ambiguity, and traceability. Automated verification will be performed for measurable criteria (e.g., transcription latency, confidence scores) using `PyTest`, while manual validation will confirm qualitative attributes such as user feedback clarity.

3.2.2 Supervisor Review

A structured review session will be held with Dr. Christian Brodbeck. The meeting will consist of:

1. A concise summary of all functional and safety-related requirements (IR, PRR, ACR, IMR).
2. System and use case diagrams for visual reference.
3. Specific discussion prompts on potentially ambiguous or high-risk requirements.

During the meeting, we will assess correctness, feasibility, and alignment with user needs. All comments will be logged as GitHub issues and tracked under the *SRS Verification* label for resolution.

3.2.3 Prototype-Based Validation

A low-fidelity prototype of VoiceBridge's transcription interface and feedback module will be used to validate usability-related requirements (e.g., accessibility of controls, clarity of feedback, and response timing). Test participants will perform scripted scenarios derived from core functional requirements, such as initiating live transcription, adjusting speech sensitivity, and reviewing transcript accuracy.Results will be compared against defined success metrics (e.g., *task completion within 5 seconds* or *90% accuracy threshold*).

3.2.4 Continuous Verification

To ensure ongoing alignment between the SRS and the evolving design, biweekly verification reviews will be conducted. These sessions will:

- Assess the impact of requirement modifications.
- Re-verify modified requirements using the checklist to confirm consistency and completeness.

Table 1: SRS Verification Checklist

Criteria	**Verification Activities**
Requirements Validation	☐ Apply checklist to all SRS sections ☐ Execute automated + manual verifications
Supervisor Review	☐ Prepare and distribute review materials ☐ Conduct formal walkthrough ☐ Record findings as GitHub issues
Prototype-Based Validation	☐ Develop interactive prototype ☐ Run scenario-based usability tests ☐ Compare results against success metrics
Continuous Verification	☐ Hold biweekly review meetings ☐ Update documentation and traceability records ☐ Re-inspect modified requirements

3.3 Design Verification

Design verification confirms that the VoiceBridge architecture and modules meet all verified requirements from the SRS. Planned Verification Activities:

- Design Review Meeting:
  Conduct a structured walkthrough of the Module Interface Specification with Dr. Brodbeck. Each module will be checked against SRS FR1–FR5 and non-functional requirements. Findings will be documented and tracked in GitHub under *Design Verification*.

- Checklist Inspection:
  Verify that data flows align with the SRS Data Dictionary and that control logic matches the Business Use Cases. Confirm consistency in naming, data handling, and error management.

- Interface Validation:
  Use UI mock-ups to confirm compliance with Accessibility and Safety-Critical requirements. Validate color contrast, keyboard navigation, and error feedback clarity.

Table 2: Design Verification Checklist

Criteria	Verification Activities
Design Review	☐ Cross-check each module with SRS requirements ☐ Log findings in GitHub
Checklist Inspection	☐ Validate data flows and logic consistency ☐ Check naming and error handling
Interface Validation	☐ Review mock-ups for accessibility and safety compliance

3.4 Verification and Validation Plan Verification

The Verification and Validation (V&V) Plan will undergo its own verification process to ensure that it is accurate, complete, and consistent with project standards. Planned Verification Activities:

- Peer Inspection:
  All team members will review the plan using a standardized checklist to evaluate completeness of scope, requirement traceability, and feasibility of proposed activities.
- Supervisor Approval:
  The finalized document will be submitted to Dr. Brodbeck for formal review. Feedback will confirm that verification procedures align with course and project expectations.
- Issue Logging and Revision Tracking:
  Any missing information, inconsistencies, or suggested improvements will be logged as GitHub issues. Updates will be reviewed and approved before submission.

Table 3: V&V Plan Verification Checklist

Criteria	Verification Activities
Peer Inspection	☐ Verify traceability and feasibility ☐ Review plan scope, objectives, and completeness
Supervisor Approval	☐ Submit plan for instructor feedback ☐ Incorporate required revisions
Issue Logging and Tracking	☐ Record suggested updates in GitHub ☐ Confirm all revisions are reviewed and approved

3.5 Implementation Verification

Implementation verification ensures that VoiceBridge's source code correctly implements the approved design and satisfies all verified requirements. Planned Verification Activities:

- Unit and Integration Testing:
  Automated tests will verify core functional requirements (SRS §10, FR1–FR5), covering audio input, speech-to-text, intent mapping, and command execution. End-to-end PUC flows (SRS §8.2) will be validated through CI pipelines with coverage tracking.

- Static Analysis and Peer Review:
  Code reviews will check compliance with maintainability and security requirements (SRS §15–16) and confirm mitigation of hazards listed in the Hazard Analysis (e.g., IR1, IMR1–IMR3). Issues will be tracked on GitHub for traceability.

- Automated Code Quality Tools:
  PEP 8 and ESLint will enforce code quality and consistency, supporting maintainability goals (SRS §15.1).

Table 4: Implementation Verification Checklist

Criteria	Verification Activities
Unit & Integration Testing	☐ Track coverage and test results ☐ Develop automated test suites for all modules
Static Analysis & Peer Review	☐ Conduct code inspections ☐ Log and resolve defects in GitHub
Code Quality Tools	☐ Enforce style rules with PEP 8 and ESLint ☐ Verify consistent formatting and documentation

3.6 Automated Testing and Verification Tools

To support consistent and repeatable verification, the following tools will be used.

Table 5: Automated Testing and Verification Tools

Tool	Purpose
PyTest / unittest	Automated unit tests for ASR and intent mapping modules (Python).
GitHub Actions	Continuous integration pipeline executing test suites on each commit.
ESLint / Prettier	Enforces style and code quality standards for JavaScript/TypeScript.
pytest-cov / Codecov	Collects and reports test coverage metrics; uploads reports per build.
WAVE / axe Accessibility Scanner	Evaluates UI compliance with WCAG 2.1 AA.

Coverage and Linting Summary

- `pytest-cov` will generate line and branch coverage reports.
- Coverage summaries will appear on pull requests and weekly trend reports in the repository.
- Linters (`ruff`, `eslint`, and `prettier`) will ensure consistent style and catch syntax or accessibility issues before merging.

3.7 Software Validation

Validation confirms that VoiceBridge meets user needs and operates as intended in real-world conditions. Approaches:

- Stakeholder Feedback and Demo Validation:
  Rev 0 and Rev 1 demos will be used to validate against user personas and supervisor expectations.
- User Testing:
  Collect feedback from two to three target users (simulated or actual) performing core use cases (PUC-1 to PUC-5) with success metrics on accuracy and ease of use.

- Functional Testing:
  Validate VoiceBridge's performance and behavior against comparable open-source Python speech-processing projects to confirm functional correctness.

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

The subsections below outline tests corresponding to the SRS Functional Requirements (FR1–FR5). Each test has a unique ID and clear execution steps.

4.1.1 4.1.1 Tests for FR1 — Accept Speech Audio via Microphone

Goal: Verify live microphone capture across platforms at ≥16 kHz.

1. test-FR1-MIC-1 Valid Microphone Capture (Desktop)

   Type: Manual

   Initial State: App installed; permissions not yet granted.

   Input/Condition: Speak a short phrase (*"Testing one two three"*) into the default OS microphone on a Windows OS laptop.

   Output/Result: App requests microphone permission; once granted, waveform/level indicator appears and raw audio frames are buffered at ≥16 kHz.

   How test will be performed: Launch app → start capture → verify UI level meter moves; export a short recording and inspect metadata (sample rate ≥16 kHz).

2. test-FR1-MIC-2 Mobile Permission & Capture

   Type: Manual

Initial State: App freshly installed on iOS and Android.

Input/Condition: First-time launch and attempt to record.

Output/Result: OS permission prompt shown; after approval, capture begins; denial shows friendly error and retry path.

How test will be performed: Deny once (expect error + guidance), then allow (expect capture + levels).

3. test-FR1-MIC-3 Default Device Selection & Fallback

   Type: Manual

   Initial State: Multiple audio devices present (laptop microphone + USB microphone).

   Input/Condition: Unplug/plug devices while capturing.

   Output/Result: The system selects the current default audio input device. If the device disconnects, it automatically falls back to another available microphone and notifies the user.

   How test will be performed: Start capture → unplug active microphone → observe fallback notice → confirm continued capture.

4.1.2 Tests for FR2 — Convert Impaired Speech to Text ≥80% Accuracy (MVP)

Goal: Validate transcription accuracy on impaired speech and common commands.

1. test-FR2-ASR-1 Dataset Evaluation vs Baseline

   Type: Automated *batchevaluation*

   Initial State: Curated impaired-speech test set with references; chosen baseline model.

   Input/Condition: Run system and baseline over full test set.

   Output/Result: System achieves ≥80% WER reduction vs baseline **or** meets defined WER delta per SRS; report includes per-speaker metrics.

How test will be performed: Evaluation script computes WER, CER; export CSV; assert threshold met.

2. test-FR2-ASR-2: Common Command Accuracy ≥80%

   Type: Automated *scriptedplayback*

   Initial State: Command list of 50+ common intents; audio variants (different speakers/noise).

   Input/Condition: Play each command sample to the system.

   Output/Result: Meets ≥80% correct textual outputs (exact or accepted synonyms).

   How test will be performed: Align decoded text with ground truth (string/intent match); assert ≥80% pass rate.

4.1.3 4.1.3 Tests for FR3 — Display Transcription for Verification

Goal: Real-time text display within 2 seconds for ≥95% of trials.

1. test-FR3-UI-1: Real-Time Display Latency

   Type: Automated (instrumented)

   Initial State: Logging timestamps at (a) audio frame arrival, (b) first token shown.

   Input/Condition: 100 short utterances across platforms.

   Output/Result: Time-to-first-text ≤2 s in ≥95% of trials.

   How test will be performed: Run scripted playback or live reads with a timer, store successful and unsuccessful trials and calculate percentage.

2. test-FR3-UI-2: Continuous Update & Finalization

   Type: Manual

   Initial State: Streaming UI enabled.

   Input/Condition: Speak a 10–15 s sentence; include pauses and restarts.

Output/Result: UI shows incremental text as it is transcribed; final text locks with a small visual cue; no flickering or text loss occurs.

How test will be performed: Observe display during input; record the session on video; verify that the text updates smoothly and that the finalization cue appears.

4.1.4 4.1.4 Tests for FR4 — Map Text to Arbitrary Device Commands

Goal: ≥90% correct mapping on a 50-command test set; handle ambiguity safely.

1. test-FR4-MAP-1: Canonical Command Set Accuracy

   Type: Automated

   Initial State: 50 predefined commands with canonical phrasings and expected command representations (API/CLI/accessibility).

   Input/Condition: Exact text phrases.

   Output/Result: ≥90% correct mapping to the expected command representation.

   How test will be performed: Feed text into intent mapper; compare output to expected representations for accuracy.

2. test-FR4-MAP-2: Paraphrase & Synonym Robustness

   Type: Automated

   Initial State: Same 50 intents with 2–3 paraphrases each.

   Input/Condition: Paraphrased commands (e.g., "launch mail app" for "open email").

   Output/Result: Maintains ≥90% accuracy across the combined set or recorded separately for robustness.

   How test will be performed: Evaluate precision/recall across the set and calculate percentage.

3. test-FR4-MAP-3: Ambiguity & Disambiguation

   Type: Manual

   Initial State: Commands with ambiguous target ("open Amazon" vs "open Amazon.ca").

   Input/Condition: Ambiguous phrasing.

   Output/Result: System asks a brief clarifying question before mapping; no unsafe default.

   How test will be performed: Provide ambiguous text; verify clarification prompt and final mapping.

4. test-FR4-MAP-4: Out-of-Domain Commands

   Type: Manual

   Initial State: Intent catalog fixed; user requests unsupported app/action.

   Input/Condition: "Turn on bedroom lights" (if smart-home unsupported).

   Output/Result: Clear feedback: unsupported command + suggestion (link/alternative).

   How test will be performed: Issue out-of-domain (OOD) text; verify messaging and no execution.

4.1.5 4.1.5 Tests for FR5 — Execute Commands on the Host Device

Goal: For each correctly recognized command, execute within 2 seconds, $\geq$95% reliability.

1. test-FR5-EXEC-1: End-to-End Execution Latency & Reliability

   Type: Automated (instrumented E2E)

   Initial State: Known-good mappings for a 50-command set.

   Input/Condition: Trigger each command (text or speech $\rightarrow$ text).

Output/Result: Visible system action completes within $\leq$2 s in $\geq$95% of runs (per command), across platforms.

How test will be performed: Timestamp intent and effect (e.g., window focus/app open), calculate success rate over the 50-command set.

2. test-FR5-EXEC-2: Accessibility/API Fallback Paths

   Type: Manual/Automated

   Initial State: Primary API disabled (simulate outage).

   Input/Condition: Execute commands that have both accessibility and API paths.

   Output/Result: System selects fallback path, still within target latency where feasible, or reports partial degradation.

   How test will be performed: Toggle feature flags/mocks; verify fallback invocation.

3. test-FR5-EXEC-3: Safety & Confirmation for Destructive Actions

   Type: Manual

   Initial State: Commands include potentially destructive actions (e.g., "delete email").

   Input/Condition: Issue destructive command.

   Output/Result: Confirmation prompt appears; execution only after explicit confirmation.

   How test will be performed: Attempt action, verify prompt appears and execution is blocked if canceled.

...

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

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?

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?