System Verification and Validation Plan for Software Engineering

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

October 28, 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
Oct 27, 2025	1.0	Added Sections 4.2, 4.3, and 4.4

Contents

1 Symbols, Abbreviations, and Acronyms **iv**

2 General Information **1**
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 Challenge Level and Extras . . . 1
- 2.4 Relevant Documentation . . . 2

3 Plan **2**
- 3.1 Verification and Validation Team . . . 2
- 3.2 SRS Verification . . . 3
- 3.3 Design Verification . . . 4
- 3.4 Verification and Validation Plan Verification . . . 5
- 3.5 Implementation Verification . . . 6
- 3.6 Automated Testing and Verification Tools . . . 7
- 3.7 Software Validation . . . 8

4 System Tests **8**
- 4.1 Tests for Functional Requirements . . . 8
  - 4.1.1 Tests for FR1 — Accept Speech Audio via Microphone 9
  - 4.1.2 Tests for FR2 — Convert Impaired Speech to Text at ≥80% Accuracy . . . 10
  - 4.1.3 FR3 — Display Transcription for Verification . . . 10
  - 4.1.4 FR4 — Map Text to Arbitrary Device Commands . . . 11
  - 4.1.5 FR5 — Execute Commands on the Host Device . . . 13
- 4.2 Tests for Nonfunctional Requirements . . . 14
  - 4.2.1 Look & Feel . . . 14
  - 4.2.2 Usability & Humanity . . . 15
  - 4.2.3 Performance . . . 17
  - 4.2.4 Operational & Environmental . . . 19
  - 4.2.5 Maintainability & Support . . . 20
  - 4.2.6 Security . . . 21
  - 4.2.7 Cultural . . . 22
  - 4.2.8 Compliance . . . 23
- 4.3 Traceability Between Test Cases and Requirements . . . 23

5 Unit Test Description **25**
5.1 Unit Testing Scope . . . 25
5.2 Tests for Functional Requirements . . . 25
5.2.1 Module 1 . . . 25
5.2.2 Module 2 . . . 26
5.3 Tests for Nonfunctional Requirements . . . 26
5.3.1 Module ? . . . 27
5.3.2 Module ? . . . 27
5.4 Traceability Between Test Cases and Modules . . . 27

6 Appendix **28**
6.1 Symbolic Parameters . . . 28
6.2 Usability Survey Questions? . . . 28

List of Tables

1 SRS Verification Checklist . . . 4
2 Design Verification Checklist . . . 5
3 V&V Plan Verification Checklist . . . 6
4 Implementation Verification Checklist . . . 7
5 Automated Testing and Verification Tools . . . 7

List of Figures

1 Symbols, Abbreviations, and Acronyms

Symbol / Acronym	Description
T	Test
OOD	Out-of-Domain: input or data that falls outside the system's trained or

[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]

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

This section defines the system-level tests for *VoiceBridge*. It follows the course template: each heading includes connective text, and each test case includes the control/type, initial state, inputs/conditions, expected outputs/results, a brief test-case derivation (where applicable), and how the test will be performed. The functional tests map directly to SRS Functional Requirements (FR1–FR5). The nonfunctional tests map to SRS Nonfunctional Requirements (LFR, UHR, PRF, OER, MSR, SCR, CLR, CPR). A traceability table linking tests to requirements is provided at the end of this section.

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 Tests for FR1 — Accept Speech Audio via Microphone

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

1. test-FR1-MIC-1
   Control: Manual

   Initial State: App installed; permissions not yet granted.

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

   Output: 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).

1. test-FR1-MIC-2
   Control: Manual

   Initial State: App freshly installed on iOS and Android.

   Input: First-time launch and attempt to record.

   Output: 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).

1. test-FR1-MIC-3
   Control: Manual

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

   Input: Unplug/plug devices while capturing.

   Output: 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 at ≥80% Accuracy

test-FR2-ASR-1 Dataset Evaluation vs Baseline

1. test-FR2-ASR-1
   Control: Automated (batch evaluation)

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

   Input: Run system and baseline over full test set.

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

   Test Case Derivation: Directly tests Fit Criterion's WER reduction expectation.

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

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

1. test-FR2-ASR-2
   Control: Automated (scripted playback)

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

   Input: Play each command sample to the system.

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

   Test Case Derivation: Meets "≥80% accuracy for common commands."

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

4.1.3 FR3 — Display Transcription for Verification

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

1. test-FR3-UI-1
   Control: Automated (instrumented)

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

Input: 100 short utterances across platforms.

Output: Time-to-first-text $\leq$2 s in $\geq$95% of trials.

Test Case Derivation: Direct Fit Criterion.

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

test-FR3-UI-2 Continuous Update & Finalization

1. test-FR3-UI-2
   Control: Manual

   Initial State: Streaming UI enabled.

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

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

   Test Case Derivation: Ensures transparency/trust via visible updates (FR3).

   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 FR4 — Map Text to Arbitrary Device Commands

test-FR4-MAP-1 Canonical Command Set Accuracy

1. test-FR4-MAP-1
   Control: Automated

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

   Input: Exact text phrases.

   Output: $\geq$90% correct mapping to the expected command representation.

   Test Case Derivation: Direct Fit Criterion.

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

test-FR4-MAP-2 Paraphrase & Synonym Robustness

1. test-FR4-MAP-2
   Control: Automated

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

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

   Output: Maintains $\geq$90% accuracy across the combined set or recorded separately for robustness.

   Test Case Derivation: Realistic language coverage (FR4 rationale).

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

test-FR4-MAP-3 Ambiguity & Disambiguation

1. test-FR4-MAP-3
   Control: Manual

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

   Input: Ambiguous phrasing.

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

   Test Case Derivation: Safe mapping for independence without surprises (FR4).

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

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

1. test-FR4-MAP-4
   Control: Manual

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

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

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

Test Case Derivation: Graceful handling of arbitrary requests (FR4).

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

4.1.5 FR5 — Execute Commands on the Host Device

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

1. test-FR5-EXEC-1
   Control: Automated (instrumented E2E)

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

   Input: Trigger each command (text or speech → text).

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

   Test Case Derivation: Direct Fit Criterion.

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

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

1. test-FR5-EXEC-2
   Control: Manual/Automated

   Initial State: Primary API disabled (simulate outage).

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

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

   Test Case Derivation: Ensures robust execution loop (FR5 rationale).

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

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

1. test-FR5-EXEC-3
   Control: Manual

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

   Input: Issue destructive command.

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

   Test Case Derivation: Safe device control (FR5).

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

4.2 Tests for Nonfunctional Requirements

This section covers system tests for the Non-Functional Requirements (NFR) in the SRS. Each test ties back to one or more NFRs (see "Covers" field). NFRs not explicitly tested here are covered by functional or unit tests and are referenced in the traceability matrix.

4.2.1 Look & Feel

test-LF-1 UI Browser Overlay Coverage

1. test-LF-1
   Type: Non-Functional, Manual, Survey-based

   Covers: LFR-11.1

   Initial State: VoiceBridge extension/app active on common pages (docs, dashboards, news).

   Input/Condition: Idle state; listening state; confirmation state.

   Output/Result: Browser is full visible, browser is fully visible with indication that in listening state, interface confirms action and user confirms.

   How test will be performed: Usability survey is performed with users indicating how well they were able to see the content while running the interface on a scale from 1-5 with 5 being the clearest. Manual spot-checks confirm no critical content is masked.

test-LF-2 Visual Consistency & Minimalism Heuristic Review

1. test-LF-2
   Type: Heuristic Inspection (Design QA)

   Covers: LFR-11.1, LFR-11.2

   Initial State: Latest build with UI design with spacing/typography/colors.

   Input/Condition: Inspect primary screens & dialogs.

   Output/Result: Checklist rating and noting consistent spacing scale, typography hierarchy, color roles, icon style.

   How test will be performed: 10-item checklist scored by two reviewers done at separate times to prevent bias, ratings are review and consolidated to make necessary fixes.

4.2.2 Usability & Humanity

test-UH-1 Core Task Ease of Use

1. test-UH-1
   Type: User Testing, Instrumented

   Covers: UHR-12.1

   Initial State: Tasks prepared (start/stop listening, confirm command, view transcript).

   Input/Condition: n$\geq$10 participants (target users).

   Output/Result: $\geq$ 90% complete each task in $\leq$ 4 interactions; success feedback observed within 1s of action.

   How test will be performed: Users interact with the interface through their browser and go through the prepared tasks, observers verify timing/task success rate/feedback time.

test-UH-2 Onboarding Learnability & First-Try Success

1. test-UH-2
   Type: User Testing + Survey

   Covers: UHR-12.3

   Initial State: Fresh profile; onboarding tutorial enabled.

Input/Condition: Participants complete first-run tutorial then perform core tasks.

Output/Result: $\geq$ 85% complete all core tasks on first attempt; onboarding time $\leq$ 10 minutes on average.

How test will be performed: Time-on-task success percentage, rating documentation out of 5 with feedback on what they liked/disliked.

test-UH-3 Personalization Ability

1. test-UH-3
   Type: Manual, Dynamic

   Covers: UHR-12.2

   Initial State: Preferences (e.g., language, microphone, confirmation style) changed.

   Input/Condition: Switch browser profile, sign-out/in.

   Output/Result: Preferences persist across sessions/devices when a user has an account.

   How test will be performed: User fills out their preferences such as language, confirmation style. Then logs out and logs in on a different device and check if behavior is saved after logging in again.

test-UH-4 Message Clarity & Politeness

1. test-UH-4
   Type: Survey-Based Language Review

   Covers: UHR-12.4

   Initial State: Library of prompts/errors/confirmations.

   Input/Condition: Participants rate clarity and politeness.

   Output/Result: $\geq$ 90% rate $\geq$ 4/5 for clarity & tone; comments reviewed.

   How test will be performed: Run a survey with participants rating the confirmation/errors/prompts out of 5, with an option to leave specific feedback at the end of the survey.

test-UH-5 Assistive Navigation

1. test-UH-5
   Type: Manual, User Testing

   Covers: UHR-12.5

   Initial State: Screen reader & keyboard nav enabled; color tokens set.

   Input/Condition: Navigate all primary flows via keyboard; screen reader announces roles/states; run contrast analyzer.

   Output/Result: All interactive elements reachable and user is able to interact and fullfill their chosen task.

   How test will be performed: User turns on their stand assistive tools and performs a task through the interface, observer checks if they are able to complete their tasks and how the program interacts with their assistive tools to ensure there are no issues.

4.2.3 Performance

test-PF-1 ASR Latency Benchmark

1. test-PF-1
   Type: Automated, Dynamic

   Covers: PRF-13.1

   Initial State: ASR service with a ready test audio set.

   Input/Condition: Feed audio with marked end-of-speech.

   Output/Result: latency ≤ 5 s from end-of-speech to transcript availability.

   How test will be performed: Measure timestamps from End of Audio to getting transcript, repeat over 20 different audio samples and calculate average latency.

test-PF-2 Command Pipeline Latency

1. test-PF-2
   Type: Automated, Dynamic

   Covers: PRF-13.2

Initial State: Full pipeline (NLP → action planning → execution).

Input/Condition: Standard command set (e.g., "summarize page", "reply to this email").

Output/Result: Average ≤ 15 s from transcript ready to command result.

How test will be performed: Measure timestamps from End of Audio to command set being performed, repeat over 20 different audio samples and calculate average latency.

test-PF-3 Accuracy & Precision (Stationary Noise)

1. test-PF-3
   Type: Offline Evaluation

   Covers: PRF-13.1, PRF-13.2

   Initial State: Labeled impaired-speech with a command set with stationary background noise (AC/fan) mixed at 20/10/5 dB.

   Input/Condition: Run ASR with the intent/command classifier.

   Output/Result: WER meets ≥ 70% accuracy; command precision ≥ 80% at SNR less than or equal to 20 dB.

   How test will be performed: Run sample audios with the different levels of background noise, calculate accuracy and command recognition against the what was actually meant from the audio sample, then calculate percentages of accuracy and precision.

Unnecessary Tests and Justifications

- **PRF-13.5 Capacity (≥ 20 concurrent users) — Out of Scope (PoC)**
  Why not tested now: The PoC targets **single-user** evaluation focused on correctness and latency. It is infeasible at this stage to run a system test that scales over 20 users and adds significant load and costs.

- **PRF-13.6 Scalability/Extensibility (Horizontal scaling; modular extensions) — Out of Scope (PoC)**
  Why not tested now: Horizontal scaling is an **infrastructure** concern that isn't meaningfully exercised without real or simulated multi-tenant

traffic. Extensibility is largely an **architecture/code-structure** attribute rather than a runtime metric.
How it's handled instead: Verified via **architecture/code review** and maintainability checks (see **test-MS-1** for modularity and extension points).

- **PRF-13.7 Longevity (reliable operations $\geq$ 5 years) — Out of Scope (PoC)**
Why not tested now: Longevity cannot be validated with a short-cycle **system test**; it requires years of operation data. Running a 5-year endurance test is infeasible at this stag

4.2.4 Operational & Environmental

test-OER-1 Cross-Platform Browser Compatibility

1. test-OER-1
Type: Manual + Automated

   Covers: OER-14.1, OER-14.2, MSR-15.3

   Initial State: Windows 10+, macOS 12+, Ubuntu 20.04+; Chrome/Edge/Firefox/Safari (latest two versions).

   Input/Condition: Run flow test.

   Output/Result: All flows pass, no OS tweaks/extensions beyond product install.

   How test will be performed: Run same flow test through each browser, validate that each runs as expected regardless of browser used.

test-OER-2 Update & Backward Compatibility

1. test-OER-2
Type: Manual + Automated Regression

   Covers: OER-14.4, OER-14.5

   Initial State: Real user profile with data/preferences.

   Input/Condition: Apply minor and major updates.

   Output/Result: No data loss. preferences retained; ¡1 minute downtime perceived.

How test will be performed: Update then validate data and features are saved simulate rollback and check the same conditions.

Unnecessary Tests and Justifications

- **OER-14.3 Adjacent systems & OSS compatibility — Out of Scope (PoC)**
  Why not tested now: No concrete adjacent systems are defined for the PoC, runtime end to end integration would be speculative and low-value. Compatibility is sufficiently covered via CI dependency/license checks and mock contract tests at this stage.

4.2.5 Maintainability & Support

test-MS-1 Codebase Maintainability Review

1. test-MS-1
   Type: Static Analysis + Doc Review

   Covers: MSR-15.1

   Initial State: Final repo & docs.

   Input/Condition: Evaluate architecture, boundaries, test coverage, comments, READMEs.

   Output/Result: Checklist score $\geq$ 85%, new dev completes a small change in $\leq$ 0.5 day using docs.

   How test will be performed: Two new reviewers/devs score the documentation and the code, they attempt to make a change to the code and rate how easily they were able to perform the task.

test-MS-2 Support Assets Accessibility

1. test-MS-2
   Type: Manual + Automated Accessibility Checks

   Covers: MSR-15.2

   Initial State: Help/FAQ pages built.

   Input/Condition: Audit contrast and readability and media aids.

   Output/Result: Pass high-contrast checks with captions/alt text present.

   How test will be performed: Run a contrast tool on the page and manual spot-checks to ensure captions and alt text is present.

4.2.6 Security

test-SEC-1 Role-Based Access Control (RBAC)

1. test-SEC-1
   Type: Manual + Automated Security Tests

   Covers: SCR-16.1

   Initial State: Accounts for primary/secondary/tertiary roles.

   Input/Condition: Attempt in-role and out-of-role actions.

   Output/Result: Least-privilege enforced, unauthorized actions blocked and logged.

   How test will be performed: Test privileges of each of the 3 users, run through a checklist of what they can and cannot access and make sure they are able to do all actions they have permissions to do and are blocked out of actions they are unauthorized to do.

test-SEC-2 Command Confirmation & Intent Match

1. test-SEC-2
   Type: Manual, Dynamic

   Covers: SCR-16.2

   Initial State: Confirmation UI enabled.

   Input/Condition: Ambiguous/noisy commands; potentially disruptive actions.

   Output/Result: System requires user validation and shows warnings with no unintended execution.

   How test will be performed: Run with a script of commands and verify confirmation flow.

test-SEC-3 Consent & Data Handling

1. test-SEC-3
   Type: Static Review + Manual Flow

   Covers: SCR-16.3

   Initial State: Fresh user; data policy active.

Input/Condition: Onboarding consent and option to opt-in for model improvement.

Output/Result: Explicit consent captured and a decline path respected with data anonymized before training.

How test will be performed: Review code/config plus run onboarding with a new user, inspect data pipeline & backup config to ensure user consent is respected.

Unnecessary Tests and Justifications

- **SCR-16.4 Secure logging (audit integrity & retention) — Out of Scope (Unit tests)**
  Why not tested now: Best validated via code-level/unit tests and static review. Duplicating with system tests in the PoC adds little value.
- **SCR-16.5 Immunity to misuse (robust to noise/rate limits) — Out of Scope (Unit tests)**
  Why not tested now: Requires targeted fault injection/adversarial unit tests (e.g., noisy inputs, burst traffic) and contract tests for rate limiting. Portions are indirectly covered by existing flows; full coverage deferred to unit/integration tests.

4.2.7 Cultural

test-CUL-1 Tone, Bias & Harm Guardrails Review

1. test-CUL-1
   Type: Static and Scenario-Based Testing

   Covers: CLR-17.1

   Initial State: Prompt templates and moderation/guardrail rules configured.

   Input/Condition: Run scenarios across diverse names with edge cases.

   Output/Result: Neutral tone; harmful/biased outputs blocked with safe messaging; escalation path provided.

   How test will be performed: Run a curated scenario set and have a reviewer checklist how the output tone is and if guardrails are deployed against harmful content.

4.2.8 Compliance

test-CPL-1 PIPEDA Compliance Audit

1. test-CPL-1
   Type: Documentation & Process Audit

   Covers: CPR-18.1

   Initial State: Data inventory, consent records, retention policy, DSAR process.

   Input/Condition: Auditor reviews artifacts & simulates data access/deletion request.

   Output/Result: Pass checklist; DSAR fulfilled within policy timelines; lawful basis documented.

   How test will be performed: Map data flows and verify consent & retention.

test-CPL-2 WCAG 2.0 AA Spot-Compliance

1. test-CPL-2
   Type: Automated and Manual Accessibility Testing

   Covers: CPR-18.2

   Initial State: UI build with accessibility tooling integrated.

   Input/Condition: Run automated audit; manual keyboard/screen reader checks on critical paths.

   Output/Result: No critical AA violations on core flows; issues tracked with remediation plan.

   How test will be performed: Automated scan and manual verification on forms.

4.3 Traceability Between Test Cases and Requirements

The table below maps each test case to the SRS requirements it supports.

Test ID	Requirement(s)
test-FR1-MIC-1	FR1
test-FR1-MIC-2	FR1
test-FR1-MIC-3	FR1
test-FR2-ASR-1	FR2
test-FR2-ASR-2	FR2
test-FR3-UI-1	FR3
test-FR3-UI-2	FR3
test-FR4-MAP-1	FR4
test-FR4-MAP-2	FR4
test-FR4-MAP-3	FR4
test-FR4-MAP-4	FR4
test-FR5-EXEC-1	FR5
test-FR5-EXEC-2	FR5
test-FR5-EXEC-3	FR5
test-LF-1	LFR-11.1
test-LF-2	LFR-11.1, LFR-11.2
test-UH-1	UHR-12.1
test-UH-2	UHR-12.3
test-UH-3	UHR-12.2
test-UH-4	UHR-12.4
test-UH-5	UHR-12.5
test-PF-1	PRF-13.1
test-PF-2	PRF-13.2
test-PF-3	PRF-13.1, PRF-13.2
test-OER-1	OER-14.1, OER-14.2, MSR-15.3
test-OER-2	OER-14.4, OER-14.5
test-MS-1	MSR-15.1
test-MS-2	MSR-15.2
test-SEC-1	SCR-16.1
test-SEC-2	SCR-16.2
test-SEC-3	SCR-16.3
test-CUL-1	CLR-17.1
test-CPL-1	CPR-18.1
test-CPL-2	CPR-18.2

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.

1. **What went well while writing this deliverable?**
   The writing process went smoothly because there was strong alignment between the project requirements and the testing approach. Breaking down each functional requirement into specific, well-defined tests helped focus the effort and ensured completeness. The collaboration within the team was productive, allowing us to clarify assumptions and verify tool use early on. Being able to connect test cases directly to the requirements and think through measurable success criteria gave the plan a solid foundation.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   A major difficulty was ensuring that all planned tests were realistically feasible given our current toolset, resources, and project timeline. Some initial test ideas were technically interesting but would have required tools or integrations we didn't yet have, or would have taken more time than we could allocate. To address this, we carefully reviewed past project documentation and assessed our CI/CD and automation capabilities to see what was immediately implementable. We then adjusted the scope of certain tests, streamlining some, combining others, and prioritizing high-impact cases, so that every test included in the plan was both practical and meaningful. This approach allowed us to maintain ambitious coverage without overcommitting, ensuring that the V&V plan is actionable, realistic, and aligned with project constraints.

3. **What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project?**
   To successfully complete the verification and validation of the VoiceBridge project, the team will need to acquire a combination of technical, ethical, and user-centered skills. This includes understanding speech data quality standards to verify the clarity, consistency, and accuracy

of dysarthric speech recordings, along with proficiency in labeling and annotation tools to ensure reliable datasets. The team must also develop skills in machine learning verification techniques, such as cross-validation, confusion matrix analysis, and bias detection, to evaluate model performance objectively. Knowledge of accessibility and ethical guidelines will be essential to ensure responsible data collection, participant consent, and inclusivity for users with speech impairments. In addition, team members should strengthen their understanding of software testing practices, including unit, integration, and system testing, to verify code reliability. Finally, the ability to conduct usability testing and analyze user feedback will be crucial for validating the system's real-world effectiveness and ensuring that VoiceBridge genuinely improves accessibility and communication support for its intended users.