System Verification and Validation Plan for Software Engineering

Team #23, Project Proxi
Savinay Chhabra
Amanbeer Singh Minhas
Gourob Podder
Ajay Singh Grewal

October 31, 2025

Revision History

Date	Version	Notes
30 October 2025	1.0	Added VnV plan

Contents

This document outlines the Verification and Validation plan for Project Proxi. It explains how our team plans to check the system we're building against our requirements. The plan goes over what tests will be run, how documents and designs will be validated and how everything connects back to the original SRS document. The main goal is to make sure Project Proxi is reliable, easy to use, and truly helps people who will use it for accessibility.

1 General Information

1.1 Summary

The aim of this document is to layout the verification and validation plan for Project Proxi. Project Proxi is an AI powered voice assistant which allows it's users to use their voice to interface with their computer. The general target demographic for Project Proxi is seniors and individuals with disabilities who wouldn't otherwise be able to use their computers for varying tasks.

1.2 Objectives

The main objective of the Verification and Validation (V&V) plan is to confirm that Project Proxi satisfies it's stated functional and non functional requirements; and fulfuills its primary purpose of enhancing computer accessibility for seniors and individuals with disabilities.

The principal objectives that this VnV plan aims to meet are:

- **Verify Functional Correctness:** Ensure that the features described in the SRS are implemented and behave as expected through Unit, Integration and System Level testing.
- **Validate Usability and Accessibility:** demonstrate that users within the target demographic can effectively operate the system using natural language commands while meeting the relevant accessibility standards like WCAG 2.
- **Assess Performance and reliability:** Confirm the application meets performance, stability and error goals as defined in the SRS document.
- **Hardware Compatibility Testing:** Meet verficiation goals on a variety of consumer hardware devices. Speacialized or adaptive devices will

not be included; only commercially available Windows/MacOS Computers.

Out of scope objectives include Extensive long-term field usability studies, comprehensive privacy compliance, and Speacialized Hardware Testing. The project will assume that any external libraries have already been verfied by their respective development team. These exclusions are justified given the project's limited time, budget and academic context. Verification efforts are therefore directed towards objectives that help meet core software requirements.

1.3 Challenge Level and Extras

This project does not include a designated challenge level, as its primary focus is on the development and validation of a functional, accessible, and user-friendly AI-powered desktop assistant. The project scope aligns with the general expectations of the capstone course and emphasizes reliability, usability, and compliance with accessibility standards rather than advanced AI research or complex algorithmic innovation.

This project includes the following approved extras:

- **User Manual:** A user manual that explains how to install, setup, and use the software on a day-to-day basis.
- **Usability Report:** A usability report will be developed that will assess the systems core objectives and requirements. It will summarize the results of testing as specfied in the SRS doc.

1.4 Relevant Documentation

The following documentation is directly relevant to the Verification and Validation activities for Project Proxi:

- **Development Plan** [Chhabra et al. (2025a)]**:** This document defines the overall project workflow, timelines and milestones for the different phases of the project.
- **Problem Statement and Goals** [Chhabra et al. (2025c)]:: This document defines the problems, goals and stakeholders of the project.

- **Software Requirements Specification** [Chhabra et al. (2025d)]**:** Describes the functional and non functional requirements of the project.
- **Hazard Analysis** [Chhabra et al. (2025b)]**:** This document identifies any potenital sources of harm or risk that may be associated with the project. Potential risks like unauthorized access, misuse or data breaches and their mitigation strategies are documented here.
- **Usability Report**[Chhabra et al. (2025e)]**:** Includes results of user evaluation and beta testing during validation stage. It summarizes key findings related to the system meeting it's design and usability goals.
- **User Manual** [Chhabra et al. (2025f)]**:** Provides documentation that explains how to install, setup and use the software. Includes helpful instruction along with walkthroughs and examples to help users with varying techincal proficiency.

2 Plan

This section outlines the strategy for Verification and Validation (V&V) across all phases of the Project Proxi lifecycle. The plan defines the roles and responsibilities of the V&V team and specifies structured approaches for verifying the Software Requirements Specification (SRS), Design, and Implementation, as well as the approach for Software Validation. This systematic process increases confidence in the correctness, accessibility, and reliability of the final system.

Verification ensures that the system is built correctly and meets its stated requirements, while validation confirms that the right system has been built for its intended users. Both static techniques (reviews, inspections, and walkthroughs) and dynamic techniques (testing and user evaluations) are used. All artifacts, results, and issues are tracked through GitHub for full traceability between requirements, test cases, and evidence.

2.1 Verification and Validation Team

The V&V activities are shared among all members of the Proxi team. Each member's role leverages their technical strengths while maintaining overlap

to ensure redundancy and consistent quality assurance. All verification tasks and results are peer-reviewed, documented, and tracked in GitHub.

Name	**Role**	**V&V Responsibilities**
Savinay Chhabra	Team-Lead/ Project-Manager	Coordinates all V&V activities, schedules document and design reviews, ensures traceability matrices are complete, and validates plan feasibility.
Amanbeer Singh Minhas	Verification Engineer / Documentation Lead	Authors test specifications, manages accessibility validation (WCAG 2.1 AA), executes FR and NFR tests, and maintains V&V evidence and metrics.
Gourob Podder	AI/Backend Developer	Verifies accuracy and latency of voice-recognition and NLP modules, conducts integration and performance tests, and performs code inspections.
Ajay Singh Grewal	Front-End / Integration Engineer	Implements automated UI tests, validates responsiveness and error handling, and ensures cross-platform consistency and workflow reliability.
Team Proxi	Quality-Assurance and Review Team	Participate in peer reviews of all documents (SRS, MIS, V&V, and Hazard Analysis). Execute assigned system and unit tests, record results, file issues on GitHub, and contribute to final validation reports

Table 1: Verification and Validation Team Roles

2.2 SRS Verification

The Software Requirements Specification (SRS) will be verified using a structured review process involving internal checks, peer feedback, and automated validation. The goal is to confirm that all requirements are clear, consistent, and ready for design and testing.

Planned Verification Stages:

- **Internal Team Review:** Each team member reviews assigned SRS

sections to ensure requirements are clearly written, uniquely numbered, and testable. Feedback and questions are added as GitHub issues under the "SRS-review" tag for tracking.

- **Peer Team Feedback:** Another project team will review our SRS to identify unclear or missing requirements. Their comments will be discussed in a short review meeting, and agreed changes will be included in the next SRS revision.

- **Automated Consistency and Traceability Check:** Scripts and manual scans confirm that every requirement label (FR, NFR, etc.) appears in the traceability table and links to a planned test in the V&V Plan. Any missing or duplicate IDs will be flagged for correction.

- **Team Validation Meeting:** After all updates, the team meets to confirm that project goals such as accessibility, reliability, and usability are fully captured in the SRS. The final version is approved as the baseline for design and testing.

2.3 Design Verification

The design of Project Proxi will be verified through focused peer reviews and checklists to confirm that the architecture, interfaces, and data flows align with the approved SRS. Each review will help identify inconsistencies or missing details before implementation begins.

Planned Verification Activities:

- **Internal Design Review:** Team members will examine their assigned modules to ensure the design is modular, consistent with SRS requirements, and feasible for implementation. Each reviewer will complete a checklist covering naming conventions, interface clarity, and data flow accuracy.

- **Cross-Team Design Review:** Another project team will review our design document and provide feedback on readability, logic, and completeness. Their comments will be logged as GitHub issues tagged "Design-review" and resolved in the next revision.

- **Walkthrough Meeting:** The team will conduct a collaborative walkthrough where each member explains their subsystem's role and connections. This helps verify that all modules integrate correctly and that there are no overlapping or missing elements.
- **Traceability Check:** A simple matrix will confirm that every major design component maps to at least one SRS requirement. Any unmatched modules or requirements will be flagged for revision before coding starts.

Design Review Checklist Items:

- Module responsibilities are clear and non-overlapping.
- Inputs, outputs, and interfaces are well defined and consistent.
- Data flow diagrams reflect correct information movement.
- Error handling and edge cases are addressed.
- Design supports accessibility and responsiveness goals.
- Traceability to the SRS is complete for all components.

2.4 Verification and Validation Plan Verification

The Verification and Validation (V&V) Plan itself will also be verified to ensure it is complete, consistent, and realistic. This review confirms that the plan defines clear responsibilities, test coverage, and measurable evidence of software quality.

Planned Verification Activities:

- **Internal Review:** Team members will inspect the V&V Plan for completeness, logical flow, and consistency with other project documents such as the SRS, MIS, and Hazard Analysis. Issues will be logged on GitHub under the tag "VnV-review".
- **Peer Team Review:** A partner project team will review the plan to provide external feedback on clarity, organization, and feasibility. Comments will be discussed in a team meeting, and accepted suggestions will be incorporated into the next version.

- **Cross-Document Consistency Check:** The team will confirm that all V&V activities listed here appear in other relevant documents and that requirement identifiers match across artifacts.
- **Review Process Validation:** Small, intentional edits (such as removing a trace link or changing a test description) will be introduced to confirm that the team's review process can detect inconsistencies. This validates the effectiveness of our review workflow.

V&V Plan Review Checklist:

- All documents are correctly referenced and consistent.
- Roles and responsibilities match Table 1.
- Each section clearly states its purpose and expected outcomes.
- Traceability between requirements and tests is complete.
- Internal and peer review processes are clearly defined.
- Plan is feasible and aligned with the project timeline and tools.

2.5 Implementation Verification

The implementation of Project Proxi will be verified through a combination of dynamic and static techniques to ensure the system performs correctly, reliably, and according to its specified design. This section summarizes how unit, integration, and system testing activities will be supported by automated tools and structured reviews.

Dynamic Verification:

- **Unit Testing:** Each software module will have dedicated unit tests as outlined in the Unit Testing Plan. Tests will confirm correct outputs, data flow, and error handling for voice-processing, NLP, and UI modules. Coverage results will be monitored through automated reports to ensure that all major functions are exercised.
- **Integration and System Testing:** Combined module testing will validate interactions between the speech engine, user interface, and backend components. These tests correspond directly to the functional and non-functional test cases listed in this document.

- **Mutation Testing:** To evaluate the quality of our test suite, small artificial faults (mutants) will be introduced into the codebase. Existing unit and system tests will be rerun to confirm that they detect the injected faults. A high mutation score will increase confidence in the strength and completeness of our test suite.

Static Verification:

- **Code Walkthroughs and Inspections:** Scheduled walkthrough meetings will allow each developer to explain their code and receive peer feedback on structure, readability, and consistency. The final class presentation will also serve as a partial code walkthrough and an opportunity to discuss implementation decisions.
- **Static Analysis Tools:** Linters and analyzers (such as ESLint, Pylint, or similar) will be used to detect common issues, unused variables, and potential logic errors. Results will be documented as part of the verification evidence.
- **Code Review via Pull Requests:** All implementation changes will go through GitHub pull requests with at least one reviewer. Reviewers will verify coding standards, style, and alignment with design diagrams.

2.6 Automated Testing and Verification Tools

Automated tools will be used throughout the implementation phase to support testing, continuous integration, and quality assurance. These tools help reduce manual effort, detect regressions early, and provide measurable evidence of code reliability and maintainability.

Unit Testing Framework:

- The project will use the built-in `pytest` framework for Python to create and execute unit tests. Each module will have its own test file and fixtures to verify inputs, outputs, and edge cases. Test results will be automatically generated after every commit.

Continuous Integration (CI):

- GitHub Actions will be configured to automatically run unit, integration, and system tests whenever new code is pushed. The CI workflow will include static analysis, linting, and coverage checks. Test outcomes and code coverage metrics will be displayed in each pull request.

Code Coverage and Reporting:

- Tools like `coverage.py` will be used to measure how much of the source code is executed during tests. Reports will include statement, branch, and function coverage. The goal is to maintain at least 85% overall coverage. Coverage summaries will be included in milestone reports.

Static Analysis and Linters:

- Static analyzers such as `pylint` and `flake8` will check for syntax errors, unused variables, and potential logic flaws. These tools also verify that code follows project style guidelines and maintainability rules. Results will be logged and tracked as part of the verification evidence.

Build and Automation Utilities:

- The project will use `make` commands and scripts to automate environment setup, dependency installation, and test execution. This ensures consistent testing results across all development machines.

2.7 Software Validation

Software validation confirms that Project Proxi satisfies user needs and performs as intended in realistic scenarios. The focus is on demonstrating that the system achieves its goals of accessibility, responsiveness, and ease of use for all users, including those with limited mobility or vision.

Planned Validation Activities:

- **User Testing Sessions:** A small group of peers and volunteers will perform task-based evaluations, such as launching applications, dictating text, and managing files through voice commands. Their feedback will measure ease of learning, efficiency, and satisfaction.

- **Accessibility Validation:** Validation will focus on confirming compliance with WCAG 2.1 AA accessibility principles. Voice feedback clarity, contrast, and response times will be recorded and compared with expected usability metrics defined in the SRS.
- **Stakeholder Review Session:** The team will present the system in a demonstration that acts as a Rev 0 validation milestone. Feedback from course staff and classmates will be collected through short surveys and used to improve the user experience.
- **External Data and Comparison:** Where available, open-source voice datasets will be used to validate recognition accuracy against baseline results. This ensures that the system behaves consistently with established standards.

3 System Tests

Each test is specific and measurable, providing sufficient detail for replication once the system is built. The tests are grouped into logical areas covering input handling, natural language understanding, task execution, feedback mechanisms, memory management, logging, and system safety.

3.1 Tests for Functional Requirements

The following tests verify that each functional requirement defined in the SRS has been properly implemented and behaves as intended. Each test specifies the type of testing, initial system state, inputs and expected results, and how the test will be performed.

3.1.1 Input and Recognition Requirements

1. Test FUNC.IR.1 - Speech and Text Input Acceptance
   Type: Functional / Manual Testing

   Initial State: System initialized and idle at the main interface.

   Input/Condition: Users provide 20 valid commands, 10 by text input (keyboard) and 10 by speech through the microphone interface. Commands include examples like “Open the calendar,” “What’s the weather today?”, and “Send an email to John.”

Output/Result: System correctly recognizes and processes at least 95% of both speech and text inputs, generating the corresponding interpreted command in the console or display window.

How test will be performed: Each input is logged automatically. The test supervisor will compare recognized commands to expected ones and calculate the success rate. Failures and misinterpretations will be noted for analysis.

2. Test FUNC.IR.2 - Speech-to-Text Accuracy (ASR Module)
   Type: Functional / Automated Evaluation

   Initial State: System with ASR module enabled and configured in a controlled acoustic environment.

   Input/Condition: 50 pre-recorded audio samples containing common computer commands spoken by 3 different speakers with varied accents.

   Output/Result: Transcribed text matches the ground-truth transcript with at least 90% word accuracy.

   How test will be performed: Audio samples will be batch-fed into the ASR module. Output transcripts will be compared to reference text using a word-error-rate (WER) metric to compute accuracy.

Functional Requirements Covered: FUNC.R.1 - FUNC.R.2

3.1.2 Intent Understanding and Task Execution

1. Test FUNC.IT.1 - Intent Recognition Accuracy
   Type: Functional / Automated

   Initial State: Natural language understanding (NLU) component initialized with trained intent classification model.

   Input/Condition: 100 labeled user commands in natural language, each mapped to a known ground-truth intent (e.g., send email, open browser, play music).

   Output/Result: In at least 90% of cases, the intent predicted by the system matches the labeled ground truth.

   How test will be performed: A test script feeds the commands into the NLU module. Predicted intents are logged and compared to human-labeled references for accuracy scoring.

2. Test FUNC.IT.2 - Task Planning and Execution Success Rate
   Type: Functional / Integration

   Initial State: System active with available software agents and network connectivity.

   Input/Condition: 20 representative high-level commands (e.g., "Schedule a meeting," "Search the web for news," "Summarize this document").

   Output/Result: Agents execute actions successfully and complete tasks in at least 85% of test cases.

   How test will be performed: Commands are executed sequentially. Testers verify that each resulting task outcome matches the intended user goal. Failures are logged with error cause (planning vs. execution failure).

Functional Requirements Covered: FUNC.R.3 - FUNC.R.4

3.1.3 Feedback and Responsiveness

1. Test FUNC.FR.1 - Response Feedback Timing
   Type: Performance / Manual Timing

   Initial State: System ready to execute user commands.

   Input/Condition: 10 varied user tasks executed (e.g., "Create a note," "Check calendar," "Send a message").

   Output/Result: Each command generates a feedback or confirmation message within 2 seconds of task completion.

   How test will be performed: A timer will start at the moment of task completion. Feedback appearance time is recorded for each trial. Average and max response times will be compared to the 2-second threshold.

Functional Requirements Covered: FUNC.R.5

3.1.4 Contextual Memory and Logging

1. Test FUNC.ML.1 - Contextual Follow-Up Command Interpretation
   Type: Functional / Manual Sequence Test

Initial State: System with active short-term conversational memory.

Input/Condition: 20 conversational command sequences (e.g., "Open my calendar" then "Add meeting at 2 PM" then "Move it to tomorrow").

Output/Result: Follow-up commands correctly interpreted in at least 90% of cases based on prior context.

How test will be performed: Each conversation sequence is entered by testers. The interpreted action and memory context will be reviewed to ensure proper resolution of pronouns and implicit references.

2. Test FUNC.ML.2 - Interaction and Task Logging Verification
   Type: Functional / Inspection

   Initial State: Logging module enabled with write access to local or cloud storage.

   Input/Condition: 10 user sessions including a mix of successful and failed tasks performed via speech and text.

   Output/Result: Each session generates a timestamped log containing input, interpreted intent, actions performed, and results.

   How test will be performed: After each session, reviewers inspect the log file for completeness and proper formatting (timestamp, session ID, action outcome).

Functional Requirements Covered: FUNC.R.6 - FUNC.R.7

3.1.5 Usability and Safety

1. Test FUNC.US.1 - High-Level Command Usability
   Type: Usability / Observational

   Initial State: Deployed system interface with speech and text enabled.

   Input/Condition: 10 participants attempt to complete 10 standard tasks (e.g., opening documents, browsing the web, checking system info) using only natural language.

   Output/Result: 100% of users complete tasks without needing to use low-level system interfaces (e.g., file paths, terminal commands).

How test will be performed: Observers record each participant's interactions. Any instance where a user needs to access a system-level interface is noted as a failure.

2. Test FUNC.US.2 - Privileged Action Confirmation
   Type: Functional / Security Validation

   Initial State: System running with permission-sensitive agents enabled.

   Input/Condition: Privileged commands such as "Delete file," "Shut down computer," or "Access settings."

   Output/Result: Each privileged command triggers a confirmation prompt and executes only after explicit user approval.

   How test will be performed: Testers issue privileged commands via both text and speech. Observers confirm that the system requests confirmation before any irreversible or high-privilege action.

Functional Requirements Covered: FUNC.R.8 - FUNC.R.9

3.2 Tests for Nonfunctional Requirements

In here specific tests are defined to help determine if Proxi's nonfunctional requirements have been met. Every test in this will look at the requirements from the SRS and determine how every result will be verified.

3.2.1 Look and Feels Requirements

1. Test NF.LF.1 - UI Conformance and Readability
   Type: Static inspection and Manual

   Initial State: The latest UI build

   Input/Condition: Walkthrough the man screens and all of the status indicators (listening, thinking, and ready) all with a checklist.

   Output/Result: Presence and visibility of main controls such as listen, stop, and undo; minimal clutter and advanced options hidden; large targets and a theme toggle being present.

   How test will be performed: In this two reviewers will go through the UI and check off the items on the checklist.

Nonfunctional Requirements Covered: APP.1 - APP.5

2. Test NF.LF.2 - Test Style and Iconography Check
   Type: Static inspection and Manual

   Initial State: Build with speech output enabled and the UI icon set implemented.

   Input/Condition: It will be a trigger system speech, with review labels, captions, and icons.

   Output/Result: Plain, short labels; synchronized on screen captions for spoken output; consistent iconography with textual labelling.

   How test will be performed: Inspecting the strings and visual elements and flows; record any inconsistencies found with videos and screenshots for evidence.

   Nonfunctional Requirements Covered: STY.1 - STY.3

3.2.2 Usability and Humanity Requirements

1. NF.US.1 - Task Based Usability and Learnability Study
   Type: Dynamic with manual usability test that will have timed tasks and a survey.

   Initial State: New users will get a 5 minute orientation to Proxi.

   Input/Condition: Doing the core tasks such as opening the app, reading files, browsing, and saving files/actions.

   Output/Result: A simple command button works for simple tasks; main actions are obvious; the participants do every task without assistance; time to do a repeat task decreases; examples on what to say are easily found; messages are simple, risky actions are confirmed, and error messages suggest next steps.

   How test will be performed: Observing timed tasks, collecting completion rates and any errors.

   Nonfunctional Requirements Covered: EOU.R.1 - EOU.R.3, LEA.R.1 - LEA.R.2, UAP.R.1 - UAP.R.3

2. NF.US.2 - Accessibility, personalization, and AODA/WCAG conformance
   Type: Dynamic test and accessibility inspection

Initial State: Build with voice only pathway enabled; captions, text scaling, and color contrast tools implemented.

Input/Condition: There is voice only interactive execution of setup and exit; repeated dictation sessions with captions enabled and locale set to English.

Output/Result: All actions are possible with voice only; captions, scaling, and contrast verified; speech model adaption over user session are noticeable; Canadian english formats are used.

How test will be performed: Run some WCAG checks, verify locale outputs, and have users test the voice only pathway.

Nonfunctional Requirements Covered: ACC.R.1 - ACC.R.2, PER.R.1 - PER.R.2

3.2.3 Performance Requirements

1. NF.PF.1 - Performance Study
   Type: Dynamic, mostly automatic performance testing

   Initial State: Standard hardware environment with typical system load

   Input/Condition: Speak a small scripted set of normal commands and a few malformed ones; keep one run in a quiet room and one with moderate indoor noise. Let the app run for a few hours

   Output/Result: Respond in less than 2 seconds; actions should be done with 80 to 90 percent accuracy; features load fast; settings are kept; rollbacks fast.

   How test will be performed: Time commands, tally accuracy, log resources, and perform updates and rollbacks.

   Nonfunctional Requirements Covered: SAL.R.1-R.2, SAF.R.1-R.2, POA.R.1-R.2, CAP.R.1-R.2, SOE.R.1-R.2, LON.R.1-R.2.

3.2.4 Operational and Enviromental Requirements

1. NF.OP.1 - Environment, ecosystem fit, interfaces, packaging, and release
   Type: Dynamic and checklist based testing

Initial State: Fresh install on Windows/macOS with integrations turned on.

Input/Condition: Use the app indoors, with normal environment, including noise, temperature, and distance; clean install; check release info

Output/Result: Works within env bounds; does not disrupt other apps; uses secure interfaces with logs; installs easily; versioned and documented releases.

How test will be performed: Spot checking tasks and env, glancing at traffic + logs, running clean installs, and reviewing the releasing page.

Nonfunctional Requirements Covered: EPE.R.1-R.4; WER.R.1-R.2; IAS.R.1-R.4; PRD.R.1-R.3; REL.R.1-R.2

3.2.5 Maintability and Support Requirements

1. NF.MS.1 - Onboarding, support, and adaptability
   Type: Light task trial and checklist based testing

   Initial State: Fresh clone; Report Problem presentable and the config editable.

   Input/Condition: A new person builds and runs using the README; they add one small tool through the registry; generate support bundle; and change default apps via the config and retry.

   Output/Result: The build run succeeds; the tool added without touching others; the support bundle has logs and info, it still works after the default app change.

   How test will be performed: Time the setup, add and register the tool, click report to inspect the ZIP, and switch default apps and retest.

   Nonfunctional Requirements Covered: 14.1-1, 14.1-2; 14.2-1, 14.2-2; 14.3-1, 14.3-2

3.2.6 Security Requirements

1. NF.SEC.1 - Access, integrity, privacy, audit stance, immunity
   Type: Permission checks and quick scans

Initial State: Fresh install; privacy policy; dependency scanner.

Input/Condition: Trigger sensitive actions; look for stored and sent data; run dependency scans.

Output/Result: No account needed; sensitive actions require explicit grants; data is protected; minimal third party data accessed with consent; no audit trails are left; no known vulnerabilities in dependencies.

How test will be performed: Start flows that require permissions, look for traffic and logs, and run dependency scans. Run SCA scans.

Nonfunctional Requirements Covered: ACS-01, ACS-02; INT-01; PRIV-01; IMM-01, IMM-02

3.2.7 Cultural Requirements

1. NF.CUL.1 - Language and Tone
   Type: Quick review of content

   Initial State: Final strings and language selector.

   Input/Condition: Scan common screens and messages. Switch languages and check key screens.

   Output/Result: Polite neutral wording; multiple languages will be usable for many basic flows.

   How test will be performed: Two people will go through the common screens and messages; one person will switch languages, complete tasks, and check key screens.

   Nonfunctional Requirements Covered: CULR-01, CULR-02

3.2.8 Compliance Requirements

1. NF.CUL.1 - Legal and standards
   Type: Checklist

   Initial State: Licenses, privacy policy, accessibility notes in repo and release.

   Input/Condition: Map behaviors to legal and standards checklist. This includes PIPEDA, MIT.

   Output/Result: PIPEDA IS followed; all licenses are compatible and included. The given standards are followed correctly.

How test will be performed: Check policies vs behaviors, open the LICENSE and release notes, do quick scan and spot check, and confirm protocols are secure.

Nonfunctional Requirements Covered: CULR-01, CULR-02

3.3 Traceability Between Test Cases and Requirements

Table 2: Requirements and Test Traceability Matrix

Test ID	Requirement ID	Comment
FUNC.IR.1	FUNC.R.1	Manual Test that checks if system successfully accepted user input
FUNC.IR.2	FUNC.R.2	Automated Test that checks Speech to Text Accuracy
FUNC.IT.1	FUNC.R.3	Automated Test that checks if system corrected interpreted user intent
FUNC.IT.2	FUNC.R.4	Integration Test that checks the system end-to-end
FUNC.FR.1	FUNC.R.5	Manual Test that checks if system is ready to execute user commands
FUNC.ML.1	FUNC.R.6	Manual Test that checks the short term memory of the model
FUNC.ML.2	FUNC.R.7	Automated Functional that checks if system logs are working as expected
FUNC.US.1	FUNC.R.8	Manual Observational test to check if users can complete task without needing low-level system interface
FUNC.US.2	FUNC.R.9	Automated test to make sure system doesn't use any unauthorized commands without prompting
NF.LF.1	APP.1 - APP.5	Manual Test that covers the Appearance Requirements
NF.LF.2	STY.1 - STY.3	Manual Test that covers the Style Requirements

Continued on next page

Table 3 – *Continued from previous page*

Test ID	Requirement ID	Comment
NF.US.1	EOU.R.1 - EOU.R.3, LEA.R.1, LEA.R.2, UAP.R.1 - UAP.R.3	Manual Test that covers the Usability, Learning, Understanding, and Politeness Requirements
NF.PF.1	SAL.R.1-R.2, SAF.R.1-R.2, POA.R.1-R.2, CAP.R.1-R.2, SOE.R.1-R.2, LON.R.1-R.2	Automatic Test that covers the Performance Requirements
NF.OP.1	EPE.R.1-R.4; WER.R.1-R.2; IAS.R.1-R.4; PRD.R.1-R.3; REL.R.1-R.2	Automatic Test that covers the Operation and Environmental Requirements
NF.MS.1	14.1-1, 14.1-2; 14.2-1, 14.2-2; 14.3-1, 14.3-2	Manual Test that covers the Maintainability and Support Requirements
NF.SEC.1	CS-01, ACS-02, INT-01, PRIV-01, IMM-01, IMM-02	Automatic Test that covers the Security Requirements
NF.CUL.1	CULR-01, CULR-02	Manual Test that covers the Cultural Requirements
NF.COM.1	LGL-01, LGL-02	Automated Test that covers the Compliance Requirements

4 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]

4.1 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]

4.2 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]

4.2.1 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. ...

4.2.2 Module 2

...

4.3 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]

4.3.1 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:

4.3.2 Module ?

...

4.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Savinay Chhabra, Gourob Podder, Amanbeer Singh Minhas, and Ajay Singh Grewal. System requirements specification. https://github.com/gkpodder/Capstone/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf, 2025a.

Savinay Chhabra, Gourob Podder, Amanbeer Singh Minhas, and Ajay Singh Grewal. System requirements specification. https://github.com/gkpodder/Capstone/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf, 2025b.

Savinay Chhabra, Gourob Podder, Amanbeer Singh Minhas, and Ajay Singh Grewal. System requirements specification. https://github.com/gkpodder/Capstone/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf, 2025c.

Savinay Chhabra, Gourob Podder, Amanbeer Singh Minhas, and Ajay Singh Grewal. System requirements specification. https://github.com/gkpodder/Capstone/blob/main/docs/SRS-Volere/SRS.pdf, 2025d.

Savinay Chhabra, Gourob Podder, Amanbeer Singh Minhas, and Ajay Singh Grewal. System requirements specification. https://github.com/gkpodder/Capstone/tree/main/docs/Extras, 2025e.

Savinay Chhabra, Gourob Podder, Amanbeer Singh Minhas, and Ajay Singh Grewal. System requirements specification. https://github.com/gkpodder/Capstone/tree/main/docs/Extras/UserManual, 2025f.

5 Appendix

This is where you can place additional information.

5.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

5.2 Usability Survey Questions?

The purpose of this survey is to validate that Project Proxi meets its usability, accessibility, and satisfaction goals from the SRS. Responses will be collected after user testing sessions to evaluate clarity, efficiency, and inclusiveness. Each question below lists how the response will be recorded and the intent behind it.

1. **How easy was it to perform tasks using voice commands?**
   Response Type: 1–5 scale (Very Hard → Very Easy).
   Intent: Measure how intuitive and learnable the system is for new users.

2. **Were system responses clear and understandable?**
   Response Type: 1–5 scale (Very Unclear → Very Clear).
   Intent: Assess clarity of feedback and correctness of system output.

3. **Did the system react quickly to your commands?**
   Response Type: 1–5 scale (Very Slow → Very Fast).
   Intent: Validate perceived responsiveness and latency satisfaction.

4. **How intuitive did the interface feel while completing tasks?**
   Response Type: 1–5 scale (Confusing → Very Intuitive).
   Intent: Evaluate the overall user interface flow and predictability.

5. **Was visual or audio feedback easy to notice and understand?**
   Response Type: 1–5 scale (Not Accessible → Highly Accessible).
   Intent: Verify accessibility and multimodal feedback effectiveness.

6. **Did you feel confident that Proxi understood your intent?**
   Response Type: 1–5 scale (Never → Always).
   Intent: Measure trust in speech recognition and NLP accuracy.

7. **How satisfied are you with the overall experience?**
   Response Type: 1–5 scale (Very Unsatisfied → Very Satisfied).
   Intent: Determine overall satisfaction and user acceptance level.

8. **Did you encounter confusion or frustration while using Proxi?**
   Response Type: Open text.
   Intent: Identify specific usability or interaction issues.

9. **Would you consider using Proxi for daily tasks?**
   Response Type: Yes / No.
   Intent: Gauge adoption potential and long-term usability perception.

10. **Do you have any comments or suggestions for improvement?**
    Response Type: Open text.
    Intent: Gather qualitative feedback for future iterations.

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?

Reflection — Amanbeer Singh Minhas

1. What went well while writing this deliverable? For this deliverable, I worked mainly on Section 3 (Plan) and Section 6.2 (Usability Survey Questions). These sections went well because they helped connect the technical and human sides of our project. In Section 3, I focused on creating a clear roadmap for verification that was realistic for our team. It felt good to see how each part linked to the SRS and our earlier work. Section 6.2 was also rewarding because it allowed me to design usability questions that measure user experience in a meaningful way.

2. What pain points did you experience during this deliverable, and how did you resolve them? The main challenge was figuring out how much detail to include in each section. At first, some parts of our plan were too general, while others were too detailed. Finding the right balance took a few rounds of review and feedback. Another difficulty was maintaining consistency in tone and wording across sections written by different team members. I helped fix this by revising and merging ideas so the final version felt smooth and unified.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? As a team, we realized we need stronger testing and validation skills. I plan to improve my understanding of automated documentation, coverage metrics, and survey analysis. Savinay aims to deepen his knowledge of integration testing. Gourob wants to explore performance and stress testing for our AI modules, and Ajay is focusing on automated UI and accessibility verification.

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? To build the skills we identified, our team plans to rely on both formal and self-directed learning. We will start by reviewing course material from SFWRENG 3S03, which covered testing concepts such as unit testing, coverage, and static analysis. This will help refresh the theoretical foundation before moving into practical work. We also plan to read online documentation and tutorials on dynamic testing and usability validation to better understand industry practices. Each member will then apply this knowledge directly to our project tasks for example, by creating small prototype tests

or running sample validation sessions. This approach keeps learning active and helps us improve through real application instead of only theory.

Reflection — Ajay Singh Grewal

1. What went well while writing this deliverable? For this deliverable, I worked mainly on section 4.2 to figure out the tests for non-functional requirements. This section went pretty good for me because it allowed me to think critically about how to measure different aspects for our software and how to make sure it essentially meets the needs of our users.
2. What pain points did you experience during this deliverable, and how did you resolve them? The main challenge I had faced during this deliverable was making sure that the test cases for the nonfunctional requirements made sense and were realistic to do. To properly do this, I had to really think about what our software was going to be used for and how we could truly measure the success of these nonfunctional requirements.
3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? The knowledge and skills that our team will need in order to ensure success in the verification and validation of our project include dynamic testing knowledge, static testing knowledge, and certain tool usage. I hope to improve my knowledge in various testing methods that can helpy verify and validate our projecet.
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? For each of the knowledge areas and skills we indentified, our team plans to use a combo of formal and self directed learning. We can start by reviewing course material and by reading online articles and tutorials on relevant topics. Each member will then apply this learnt knowledge directly into the successful creation of our project. We can also do small prototype tests or run sample validation sessions to help us learn better.

Reflection — Gourob Podder

1. What went well while writing this deliverable?

Writing this deliverable went smoothly once I established a clear mapping between each functional requirement and its corresponding tests. The structure of the SRS helped guide our testing logic, and I was able to design specific, measurable, and realistic test cases for all major system components from input handling to agent-based task execution. Collaboratively, we divided responsibilities efficiently, which allowed us to maintain a consistent level of technical depth and clarity throughout the VnV plan.

2. What pain points did you experience during this deliverable, and how did you resolve them?

One major challenge we faced was time management. Our group had to apply for an MSAF (McMaster Student Absence Form) to obtain academic relief for a late submission due to unexpected health issues amongst our group members. Getting approval for the MSAF was a bit of a hassle — it required communication with prof and the relevant university department. This delay temporarily disrupted our workflow and added stress to the deliverable timeline. We resolved it by setting up a stricter internal schedule after the approval to get the work completed efficiently.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? We had to learn a lot on how testing and verification worked for black boxes like LLMs:

- Testing and evaluation of AI/NLP models (e.g., measuring intent classification accuracy and ASR performance).
- Usability testing with diverse user groups, particularly those with accessibility needs.

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?

- **AI/NLP Testing and Evaluation:** Approaches include (1) completing online courses or tutorials on NLP evaluation (e.g., Coursera,

Hugging Face documentation) and (2) implementing small-scale evaluation scripts for our ASR and intent recognition modules. I will focus on the implementation approach since I already have a strong technical background and prefer hands-on experimentation with model outputs.

- **Usability and Accessibility Testing:** Approaches include (1) conducting supervised user testing sessions with accessibility-focused participants One of our teammates with a human–computer interaction background will lead user testing, while I will focus on analyzing the qualitative data collected from these sessions to guide improvements.

Reflection — Savinay Chhabra

1. What went well while writing this deliverable? The team did a much better job communicating what each team member was doing well in advance. This made working on dependent sections a lot easier than previous deliverables. This prevented unnecessary back and forth between team members which saved significant time.
2. What pain points did you experience during this deliverable, and how did you resolve them? The biggest challenge with this deliverable was that the entire team was absent for the deadline and several days before that as well. We were unclear on how to request academic relief which made it even more challenging. In the end, we were able to work with the Office of the Associate Dean and the course professor to get an extension. All this would have been a lot simpler and easier for everyone if we requested academic relief well before the deadline.
3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Some basic verficiation and validation skills including coming up with test plans, scenarios, user groups and correctness are required. This is important as the objectives we wish to validate would require comprehensive test cases which cover the vast majority of scenarios we would expect to see in the real world. For that, knowledge about Unit Testing, Integration Testing, Stability Testing and End to End Integration Testing would be very useful to have.
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? A good start would be to review prior course material that taught us all of these concepts. SFWRENG 3S03 was a very useful course that taught about testing. This would help build a strong theoretical foundation. For practical learning, going through open source software projects and their tests (both automated and manual) would be very helpful. Reading their test plans and contributing to their testing when feasible would be a great way to get some practical experience.