System Verification and Validation Plan for Software Engineering

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

October 30, 2025

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

The 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.

2.2 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.

2.3 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.

2.4 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.

3 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.

3.1 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

3.2 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.

Tools Used:

- GitHub Issues and Projects for tracking comments and progress.
- Review checklists for verifying requirement structure and numbering.
- Simple scripts to detect missing or duplicate requirement IDs.
- Revision history documenting updates and approvals.

3.3 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.

3.4 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.

3.5 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.

3.6 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.

3.7 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.

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   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:

4.1.2 Area of Testing2

...

4.2 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.

4.2.1 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

4.2.2 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

4.2.3 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.

4.2.4 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

4.2.5 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

4.2.6 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

4.2.7 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

4.2.8 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

4.3 Traceability Between Test Cases and Requirements

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

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

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

References

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.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

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

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

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

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?