Software Requirements Specification for Software Engineering: subtitle describing software

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

October 10, 2025

Contents

Revision History

Date	Version	Notes
10/10/2025	1.0	Added Initial SRS

1 Purpose of the Project

1.1 User Business

Proxi is an AI-powered desktop assistant that lets people operate a computer entirely through natural speech. It targets users who face barriers with traditional input devices (keyboard, mouse, complex UIs) and organizations that want to provide inclusive access to essential digital tasks (communication, learning, work). Proxi augments independence and reduces the digital divide by turning voice into safe, precise computer actions. While accessibility is the primary driver, Proxi is equally intended for general users who want faster, lower-friction workflows so it benefits both disabled and non-disabled users.

1.2 Goals of the Project

G-1 (Latency) Spoken system responses for common commands shall begin within ≤ 2.0 s from end-of-speech in a quiet environment.

G-2 (Recognition Accuracy) Command recognition accuracy for supported language(s) in quiet environment shall be $\geq 90\%$ intent-level accuracy.

G-3 (Task Coverage) Users from the primary user group shall complete $\geq 80\%$ of a predefined core task suite (open app/file, browse, compose, save, schedule) using voice or keyboard.

G-4 (Effectiveness) Compared to baseline (traditional input for same users), Proxi shall improve task completion rate by $\geq 20\%$.

G-5 (Satisfaction) Accessibility-focused usability tests shall yield 4.0/5.0 satisfaction score.

G-6 (Stretch Goals) Voice recognition improvements, offline capabilities, multimodal interaction support, personalized profiles, enhanced accessibility

2 Stakeholders

2.1 Client

The clients for this project are the SFWRENG 4G06A Capstone teaching team at McMaster University (course Instructor and assigned Teaching Assistants), serving as the product owners on behalf of the department. Their mandate is to ensure the solution meets accessibility, usability, and engineering quality standards appropriate for a capstone deliverable and potential real-world piloting within academic environments. They provide domain expectations (accessibility best practices, privacy/compliance constraints in academic settings) and approve scope and milestones.

2.2 Customer

Our customers are the end-users and organizations that will deploy Proxi to enable inclusive and more efficient computer use:

1. **Educational institutions** (libraries, computer labs, accessibility services) seeking hands-free or low-friction access to standard desktops and web apps.
2. **Healthcare and community organizations** supporting users with motor, vision, or hearing challenges.
3. **General consumers and power users** who prefer faster, voice-first or mixed-modality workflows.

2.3 Other Stakeholders

Other stakeholders include any person or group with interest beyond the client and the customer:

1. **Team Proxi (development team):** responsible for requirements, design, implementation, testing, and deployment artefacts.
2. **Course Staff (Instructor & TA):** guidance, assessment, feedback, and approvals.

3. **Accessibility Advisors (if engaged):** best practices for WCAG/AT compatibility.

4. **Pilot participants:** individuals who will use the system during user studies and provide feedback.

2.4 Hands-On Users of the Project

Primary hands-on users who will directly interact with Proxi:

1. **Accessibility-focused users:** People with motor impairments or temporary/situational limitations needing hands-free or simplified control.

2. **General users/power users:** Users seeking faster workflows via voice/text commands with optional keyboard/mouse confirmation.

2.5 Personas

1. **P1: Amrita (72) - Elderly User:** Amrita is a retired teacher who uses her desktop to check emails, pay bills, and video call her family. She struggles with small buttons, complicated menus, and remembering multi-step actions, which makes her anxious about using technology. She needs a way to perform common tasks more easily and with clear guidance to feel confident online.

2. **P2: Leo (21) - User with Motor Disability:** Leo is a computer science student who finds it difficult to use a keyboard or mouse due to a motor impairment. He needs to read PDFs, take notes, and switch between different apps for his coursework. He requires a way to interact with his computer hands-free and complete his work without relying on physical input.

3. **P3: Ari (28) - Power User:** Ari works with spreadsheets, emails, and web tools throughout the day and often repeats the same steps over and over. Switching between programs slows him down, and remembering different commands and shortcuts is frustrating. He needs a more efficient way to complete multi-step tasks and manage his work without constant interruptions.

2.6 Priorities Assigned to Users

The **highest priority** users for this project are **accessibility-focused** users, including elderly users like Amrita and users with motor impairments like Leo, as the main goal is to improve computer access and usability for people who face physical challenges or find current systems difficult to use. Their needs guide the core features and design decisions of the project. **Power users** like Ari are the **secondary priority** because, while they do not face accessibility barriers, their focus on speed and efficiency helps shape advanced features that make the system useful to a wider audience. Prioritizing these groups ensures the solution is both inclusive for those who need accessibility support and valuable for everyday users seeking more efficient workflows.

2.7 User Participation

Estimated participation during the project will primarily involve end users and the development team. Accessibility-focused users are expected to participate for about 1 hour per week through remote or in-lab sessions focused on usability testing, feedback, and formative evaluations. General users and power users will contribute roughly 1 hour per week by taking part in efficiency and performance testing of new features. Additionally, the development team will dedicate around 8–10 hours per week to designing, building, testing, and refining the system based on user feedback and project milestones.

2.8 Maintenance Users and Service Technicians

The primary maintenance users for this project will be the development team. During the capstone project, they will be responsible for identifying and fixing issues, releasing updates and patches, and ensuring that the system continues to function as expected throughout development. They will also create and maintain documentation such as installation guides and user manuals to support future use and potential handoff of the project after completion.

3 Mandated Constraints

3.1 Solution Constraints

1. The application must be lightweight and run without requiring any speacial hardware

2. The application will operate without requiring user accounts or signups.

3.2 Implementation Environment of the Current System

1. The application is designed to run on the personal computers of the users running mainstream operating systems like Windows and MacOS.

2. The application uses standard I/O devices like mice, keyboards, mics, and screens to interact with the user.

3. The software bundle includes JavaScript support, the MCP (modular control platform) and any other required libraries.

3.3 Partner or Collaborative Applications

1. The application may use AI services via secure APIs for natural language processing and logic.

2. Integration with external accessibility tools like screen readers may be supported to enhance usability for older adults and users with disabilities.

3.4 Off-the-Shelf Software

The application will use off-the-shelf software components to reduce devlopement time:

1. MCP (Modular Control Platform) will be used for accessibility and interface management.

2. JavaScript libraries and frameworks will be used to implement interactive elements and handle input/output operations.

3. LLM services will be used to provide natural language responses and reasoning capabilities. The application does not require a specialized model, so no additional training will be necessary. An off-the-shelf LLM is sufficient to deliver accurate and reliable results for the intended functionality.

3.5 Anticipated Workplace Environment

1. The application is expected to run on the personal computers of the users at their homes or assited-living facilities.

2. The application will be used primarily on Windows and MacOS operating systems.

3. The system is designed for users with limited technical experience, including older adults and people with disabilities, requiring a simple, intuitive interface.

4. Devices may vary in screen size, input methods, and processing power, so the app must function on standard consumer computers without specialized hardware.

5. Environmental factors such as background noise or poor acoustics may affect voice interactions. Therefore, the application shall provide alternative input methods, such as text or on-screen controls, to ensure usability when voice input is infeasible.

3.6 Schedule Constraints

Development will follow the planned project timeline ensuring that deliverables are completed before the deadlines.

3.7 Budget Constraints

The project budget is limited to $500 in total expenditure. Of this $500, $125 will be provided by the CAS department for approved expenses.

Table 1: Project Timeline

Deliverable	**Deadline**
Req. Doc. and Hazard Analysis Revision 0	October 6, 2025
V&V Plan Revision 0	October 27, 2025
Design Document Revision -1	November 10, 2025
Design Document Revision 0	January 19, 2026
Revision 0 Demonstration	February 13, 2026
V&V Report and Extras Revision 0	March 09, 2026
Final Demonstration (Revision 1)	March 29, 2026
EXPO Demonstration	TBD
Final Documentation (Revision 1)	April 06, 2026

3.8 Enterprise Constraints

There are no enterprise constraints for this application. The application will be allowed to use any service or LLM as required as long as it adheres to the budget constraints. This ensures flexibility and future adaptability.

4 Naming Conventions and Terminology

4.1 Glossary of All Terms, Including Acronyms, Used by Stakeholders involved in the Project

MCP (Model Context Protocol) A contract with which an AI agent can find and invoke a toolset to accomplish specific tasks.

MCP Server A local server that implements the MCP to allow AI agents to discover and use the available tools.

MCP Client An AI agent that connects to the MCP server to use the provided tools.

AI Agent A software (LLM) that uses artificial intelligence to perform tasks.

LLM (Large Language Model) A type of AI model that can understand and generate human language.

Token A unit of text used in LLMs to process and generate language.

Sandbox A secure environment in which code can be run and tested without affecting the rest of the system.

Action A single execution of a tool with validated parameters.

Plan An ordered sequence of actions to fulfill a user intent.

Command An instruction given by the user to the AI agent.

Permission Scope Level of access that is given to the AI agent for completing actions.

Confirmation Gate A approval prompt given to the user by the AI before cpmpleting risky actions.

TTS (Text-to-Speech) Tool that converts written text into spoken words.

STT (Speech-to-Text) Tool that converts spoken words into written text.

Audit Log A record of all actions taken by the AI agent.

Accessibility It is usable by people with a wide range of abilities/disabilities.

User Interface (UI) The visual elements of the software (frontend) in which a user will engage with the system.

PII (Personally Identifiable Information) Any data that could identify a specific individual.

POLP (Principle of Least Privilege) A security concept in which the AI agent will have minimum levels of access necessary to perform certain tasks.

OS Automation Tools Software on one's computer that gives the AI agent the ability to control and engage with their OS.

5 Relevant Facts And Assumptions

5.1 Relevant Facts

1: The Proxi's main goal is to enable hands free computer control through natural language commands for disabled users, while also providing users with a more efficient workflow. Tasks can include opening applications, reading and writing files, browsing the web, and scheduling events. Interaction with the proxi is voice command, while also having a textual input option.

2: The system will run on desktop operating systems (Windows, macOS) and will engage with many common applications (web browsers, email clients, and documents/file browsers).

3: Since certain actions can be considered high risk, such as deleting files or changing system settings, the system will confirm with the user before performing such risky actions that can potentially have destructive or sensitive outcomes.

5.2 Business Rules

The proxi must get consent before getting or dealing with any personal data. It must also decrease the amount of data shared with any external source, to only what is necessary to fulfill the user's command. Any command or action that deals with a high risk scenario (delete data, change system settings, install any software, etc.) must be first confirmed by the user to do so. Any actions done should be recorded in an audit log, allowing a track of what the AI has done. This way any incorrect actions can be traced back and reversed if necessary. Tools the AI use can only run in a user permission scope, meaning it can only access and do actions on what the user has access to and allowed.

5.3 Assumptions

Hardware The user has a working computer with a microphone and some sort of audio output, that is capable of running the software on a modern OS (Windows, MacOS).

Permissions The user can give the permissions the AI agent needs (microphone, screen access, and automation APIs) and can run the MCP server effectively.

Internet The user has a strong internet connection to use the AI agent as the LLM will run online and will require online tools (browsers, web search, etc.).

Environment and Language The user will be in a quiet environment in which the mic can comfortably pick up sound. The user will also talk in a supported language for interaction (English).

6 The Scope of the Work

6.1 The Current Situation

In the current computing landscape, many individuals face barriers when interacting with traditional computer interfaces such as keyboards, mice, and complex graphical user interfaces. People with motor disabilities, visual impairments, elderly users, or those lacking technical proficiency often struggle to perform basic computer tasks efficiently. Existing digital assistants (e.g., Siri, Alexa, or Google Assistant) provide limited task execution and lack deep integration with desktop environments or productivity applications. As a result, accessibility remains a significant challenge, and there is a growing demand for more adaptive, intelligent, and inclusive systems that can understand user intent through natural language and perform meaningful actions across software environments.

6.2 The Context of the Work

The proposed project aims to develop an AI-powered assistive technology platform designed to improve computer accessibility by leveraging natural language processing, speech recognition, and intelligent task automation. The system will act as an intermediary between the user and the computer, interpreting spoken or textual commands and executing them through modular software agents. These agents will be capable of reasoning, planning, and performing actions such as sending emails, browsing the web, or managing

files. The work will combine accessibility design principles with AI technologies to create an inclusive experience that empowers users who cannot easily interact with conventional input devices.

6.3 Work Partitioning

The work will be divided into the following major components:

1. **Input and Recognition Layer:** Responsible for capturing and processing user input through speech recognition or text interfaces.
2. **Natural Language Understanding (NLU):** Converts the user's intent into structured commands that can be acted upon by the agent system.
3. **Agent Planning and Execution Layer:** Deploys intelligent agents capable of reasoning, planning, and interacting with tools or APIs to achieve user goals.
4. **Feedback and Output Layer:** Returns the results of executed actions to the user through text or synthesized speech, ensuring clarity and accessibility.
5. **Accessibility and Usability Design:** Focused on interface simplicity, adaptive behavior, and inclusive design for users with various needs.
6. **Evaluation and Testing:** Involves user studies and performance evaluation to ensure reliability, accuracy, and user satisfaction.

6.4 Specifying a Business Use Case (BUC)

BUC–01: Intelligent Accessibility Assistant
Primary Actor: User (individual with limited mobility or low computer literacy)
Goal: To perform computer-based tasks through natural speech or text commands without relying on traditional input devices.
Preconditions:

- The user has access to a device with a microphone, speaker, and internet connectivity.

- The assistive system is installed and running in the background.

Main Flow:

1. The user issues a command such as "Open my email and send a message to my doctor."
2. The system processes the speech or text input and identifies the intended task.
3. The planning agent determines which actions or tools are required to complete the task.
4. The system executes the corresponding steps (e.g., opens an email client, composes a message, and sends it).
5. The system provides feedback to the user confirming the task completion.

Alternative Flows:

- If the system cannot interpret the user's request, it will ask clarifying questions.
- If a required tool or permission is missing, the system will notify the user and suggest corrective actions.

Postconditions:

- The requested action is completed successfully or an appropriate message is returned.
- The system logs the interaction for future learning and performance improvement.

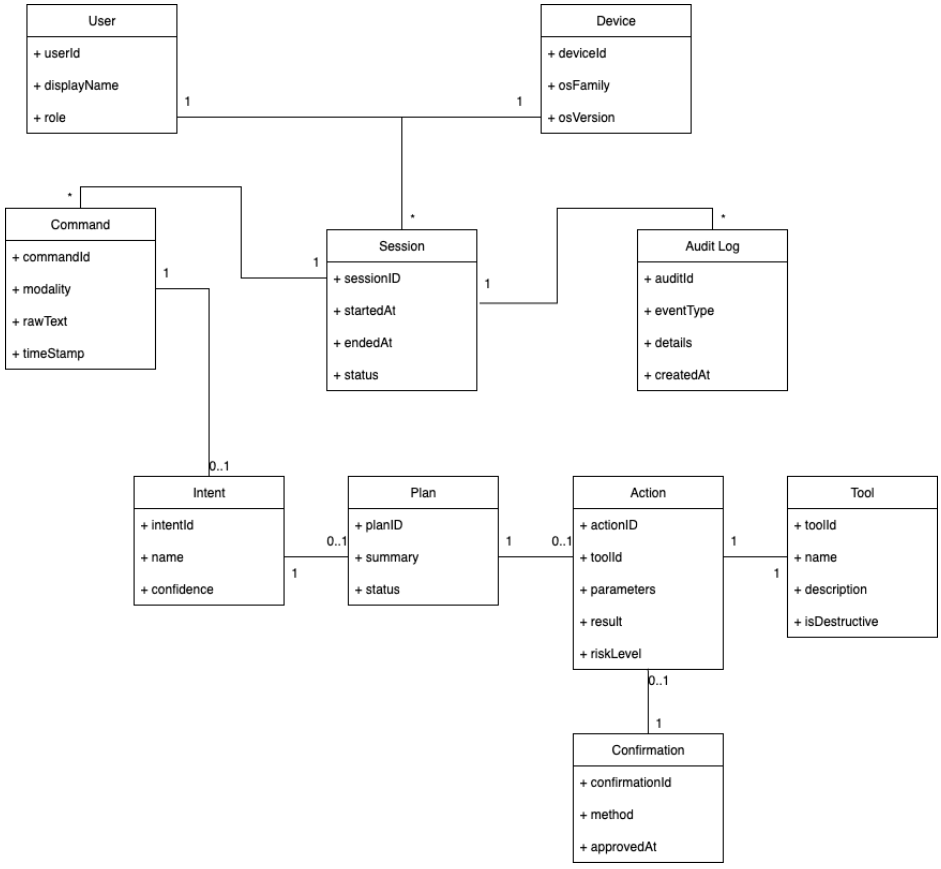

Figure 1: Business Data Model

7 Business Data Model and Data Dictionary

7.1 Business Data Model

7.2 Data Dictionary

User

Attribute	Type	Description
userId	UUID	Unique identifier for a user
displayName	String	Name shown in confirmations/UI
role	Enum	Authorization role (*Standard*, *Admin*)

Device

Attribute	Type	Description
deviceId	UUID	Logical device identity
osFamily	Enum	OS family (*Windows*/*macOS*)
osVersion	String	OS version string

Session

Attribute	Type	Description
sessionId	UUID	Logical dialog/control session
startedAt / endedAt	Timestamp	Session start/end
status	Enum	*Active*, *Ended*, or *Error*

Command

Attribute	Type	Description
commandId	UUID	Captured user request
modality	Enum	*Voice*, *Text*, or *Mixed*
rawText	String	Transcribed/typed command text
timestamp	Timestamp	Capture time

Intent

Attribute	Type	Description
intentId	UUID	Normalized goal
name	Enum/String	Label
confidence	Float	Classifier confidence (0.0–1.0)

Plan

Attribute	Type	Description
planId	UUID	Ordered set of actions
summary	String	One-line description
status	Enum	*Proposed*, *Running*, *Succeeded*, *Failed*

Action

Attribute	Type	Description
actionId	UUID	Atomic operation
toolId	UUID	Tool to invoke
parameters	JSON	Tool inputs (schema-validated)
result	JSON	Tool outputs/return
riskLevel	Enum	*Low*, *Medium*, *High*

Tool

Attribute	Type	Description
toolId	UUID	Unique tool capability
name	String	Human-friendly tool name
description	String	Purpose/constraints
isDestructive	Boolean	Writes/installs/deletes

Confirmation

Attribute	Type	Description
confirmationId	UUID	Approval record
method	Enum	*GUI* or *Voice*
approvedAt	Timestamp	Time of approval

AuditLog

Attribute	Type	Description
auditId	UUID	Append-only log entry
eventType	Enum	example: *CommandCaptured*, *ActionSucceeded*
details	String/JSON	Minimal facts about the event
createdAt	Timestamp	When the entry was written

8 The Scope of the Product

8.1 Product Boundary

Proxi is a desktop helper that lets people run their computer with short voice or text commands. In this project, Proxi listens, figures out what the user means, asks for a quick confirmation when needed, and then carries out the task on the local machine through tools (MCP).

In-Scope Features:

- **Input and understanding:** Listen for a request, interpret the intent, and ask a brief follow-up if there are multiple matches. Learn about the user's system and common tasks as time goes on to make Proxi more efficient.
- **Desktop actions via MCP:** Perform common tasks such as: open an app or file, learn and interact with user apps, browse/search, save/move/rename files using local tools with least privilege access.
- **Safety and privacy:** Ask before anything destructive, offer a simple undo when possible, and keep a short audit trail of what happened.

- **Accessible, plain language:** Show captions for spoken output, use readable defaults (contrast/size), and support keyboard or voice for every step.

Out-of-Scope Features:

- Replacing full third-party apps (complete email client, calendar, or browser).
- Unattended background automation, remote administration, or multiple users controlling the same desktop session.
- Offline language understanding/generation, enterprise policy management, or default long-term cloud storage of user data.

8.2 Product Use Case Table

Use Case	Actors	Description	Preconditions	Outcome
PUC-01: Open Application	User	Open a named app.	App installed; automation allowed.	App launched or focused.
PUC-02: Open and Read File	User	Open a file; preview/read.	File exists; within scope.	File opened; content visible.
PUC-03: Browse and Search	User	Go to a URL or search.	Browser available; internet on.	Page or results shown.
PUC-04: Compose Draft	User	Create a short email/message draft.	Default mail/editor set.	Draft opens prefilled.
PUC-05: Save/Organize File	User	Save As, move, or rename.	Paths allowed; confirm overwrite.	Operation completed; confirmed.
PUC-06: Interact with User Apps	User	Perform simple actions in an open app (click, type, select).	Target app is open; within allowed scope.	Action done or clear reason shown.

Table 2: Product Use Case Table

8.3 Individual Product Use Cases (PUC's)

PUC-01: Open Application The user asks Proxi to open an application by name. If there are several matches, Proxi lists them and the user picks one. Proxi checks permissions, opens or focuses the app, and confirms success. If the app isn't available, Proxi explains why and suggests the next step.

PUC-02: Open and Read File The user asks to open or read a file. Proxi resolves the path or name within the allowed scope, opens it in the default viewer, and can read a section aloud or give a short summary on request. If the file can't be accessed, Proxi says why and offers alternatives.

PUC-03: Browse and Search The user says to go to a certain URL or to search for something, Proxi validates the URL or forms a search, opens the default browser, and lands on the right page. If the link looks unsafe, Proxi warns the user and asks for confirmation.

PUC-04: Compose Draft The user dictates a short message or email. Proxi extracts recipients and content, opens a new draft in the default app, and fills in the fields. The user reviews and sends. If a recipient is unknown, Proxi asks for the address.

PUC-05: Save/Organize File The user asks to Save as, Move, or Rename. Proxi checks the target path, handles name collisions, and asks before any overwrite. After the operation, Proxi confirms what changed; if it couldn't proceed, it explains the reason and options.

PUC-06: Interact with User Apps The user asks Proxi to do a simple action inside an app that's already open (for example: click a menu item, type a short phrase, press a button, or toggle a setting). If there's any ambiguity, Proxi asks a brief follow-up ("Which button?") and then performs the action. If the app can't be controlled, Proxi explains why and suggests the next step.

9 Functional Requirements

9.1 Functional Requirements

Req#	Requirement	Fit Criterion
FUNC.R.1	The system shall accept user input through both speech and text interfaces.	Successful recognition and processing of at least 95% of valid user speech or text commands during testing.
FUNC.R.2	The system shall convert speech input into text using an automatic speech recognition (ASR) module.	The transcribed text shall match user speech with at least 90% word accuracy in controlled test conditions.
FUNC.R.3	The system shall interpret user intent from natural language input to determine the desired task.	In 90% of test cases, the interpreted intent shall match the human-labeled ground truth task.
FUNC.R.4	The system shall plan and execute tasks by invoking one or more software agents that perform actions on behalf of the user.	The system shall successfully complete end-to-end task execution in at least 85% of test scenarios.
FUNC.R.5	The system shall provide textual or spoken feedback describing the status or results of the requested task.	Each executed command shall generate a confirmation message or output within 2 seconds of task completion.
FUNC.R.6	The system shall maintain a short-term conversational memory to handle contextual follow-up commands.	In at least 90% of test cases, follow-up commands dependent on prior context shall be correctly interpreted.
FUNC.R.7	The system shall log all user interactions and task results for analysis and traceability purposes.	Each user session shall generate a verifiable log entry containing timestamped input and outcome data.

Table 3: Functional Requirements and Fit Criteria

Req#	Requirement	Fit Criterion
FUNC.R.8	The system shall operate without requiring the user to interact with low-level system interfaces (e.g., OS file paths or command-line tools).	In usability tests, 100% of user operations shall be achievable through high-level natural language commands only.
FUNC.R.9	The system shall ensure that all agent actions requiring system-level access request user confirmation before execution.	Each privileged action (e.g., deleting files) shall trigger a confirmation request and require explicit user approval.

Table 4: Functional Requirements and Fit Criteria - Continued

10 Look and Feel Requirements

10.1 Appearance Requirements

APP.1 The interface should look simple and familiar, like a normal desktop tool with a small status bar that shows when the system is listening, thinking, or ready.

APP.2 Main actions such as listen, stop, undo, and confirm should be clearly visible as buttons on the main screen.

APP.3 The screen should not feel crowded; only the most important options should be shown, and advanced options hidden in a simple menu.

APP.4 Text and buttons should be large and easy to read or click, especially for new or elderly users.

APP.5 A light and dark theme toggle should be available so users can choose what is most comfortable for them.

10.2 Style Requirements

STY.1 Labels and messages should use short, plain language with no technical hard language.

STY.2 When the system speaks, the text should also appear on screen as captions.

STY.3 Icons should stay consistent and have a short text label underneath to avoid confusion.

11 Usability and Humanity Requirements

11.1 Ease of Use Requirements

- **EOU.R.1** The system shall be easy for new users to start using. Basic tasks like opening an application or reading a file should require only one simple command or button press.
- **EOU.R.2** The system shall have an intuitive interface that does not overwhelm users. Main actions such as listen, stop, undo, and confirm must be clearly visible and easy to access.
- **EOU.R.3** After a short introduction, users should be able to complete most everyday tasks without assistance, and tasks should become faster with practice.

11.2 Personalization and Internationalization Requirements

- **PER.R.1** The system shall adapt to a user's speech patterns over time to improve recognition accuracy.
- **PER.R.2** The system shall support Canadian English standards for language, date, and time formatting and allow future translation if needed.

11.3 Learning Requirements

- **LEA.R.1** The system shall have a short learning curve, allowing new users to understand and use basic commands within 30 minutes.
- **LEA.R.2** The system shall provide a list of example commands or a "what can I say" option to help users learn available features quickly.

11.4 Understandability and Politeness Requirements

- **UAP.R.1** The system shall use simple and clear language for all text and messages, avoiding technical jargon.
- **UAP.R.2** The system shall confirm actions that could change or delete important files or settings before executing them.
- **UAP.R.3** Error messages shall be polite, explain the issue in plain language, and suggest a next step to fix it.

11.5 Accessibility Requirements

- **ACC.R.1** The system shall allow all actions, including setup and exit, to be performed using voice commands without requiring a keyboard or mouse.
- **ACC.R.2** The system shall comply with recognized accessibility standards such as **WCAG 2.2** and the **Accessibility for Ontarians with Disabilities Act (AODA)**. All interface elements must provide captions for spoken responses, support text scaling, and ensure sufficient color contrast to assist users with visual impairments.

12 Performance Requirements

12.1 Speed and Latency Requirements

- **SAL.R.1** The system shall begin responding to a voice command within 2 seconds under normal conditions.
- **SAL.R.2** Common actions, such as opening an application or navigating files, shall complete within 3 seconds on standard hardware.

12.2 Safety-Critical Requirements

- **SAF.R.1** Any action that changes files or system settings shall require user confirmation before execution.
- **SAF.R.2** The system shall allow users to undo or roll back critical actions in a single step to prevent accidental changes.

12.3 Precision or Accuracy Requirements

- **POA.R.1** The system shall correctly recognize at least 90% of supported voice commands in a quiet environment.
- **POA.R.2** In a moderately noisy environment, the system shall maintain at least 80% command recognition accuracy.

12.4 Robustness or Fault-Tolerance Requirements

- **ROFT.R.1** The system shall log and report input errors without crashing or losing user data.
- **ROFT.R.2** The system shall continue running even if it receives an invalid or incomplete command.

12.5 Capacity Requirements

- **CAP.R.1** The system shall support continuous operation for at least 4 hours without performance degradation.
- **CAP.R.2** CPU usage shall remain under 30% during normal operation on standard hardware.

12.6 Scalability or Extensibility Requirements

- **SOE.R.1** The system shall allow new features or modules to be added without major changes to existing functionality.
- **SOE.R.2** Any new feature or skill shall load and become available for use within 200 milliseconds.

12.7 Longevity Requirements

- **LON.R.1** The system shall retain all user settings and personalization data after updates.
- **LON.R.2** The system shall allow rollback to a previous version within 60 seconds if an update fails.

13 Operational and Environmental Requirements

13.1 Expected Physical Environment

- **EPE.R.1** The system shall operate on personal computing devices such as laptops, desktop computers, equipped with standard hardware components, including a microphone, speakers, and a stable internet connection.
- **EPE.R.2** The system shall be designed to work optimally for indoor environments such as homes, offices where noise levels are moderate(under 65dB).
- **EPE.R.3** The speech recognition capability shall achieve reliable accuracy when the user speaks within approximately one meter of the microphone input.
- **EPE.R.4** The system shall be able to operate effectively under normal indoor environmental conditions, including temperatures between 15°C and 30°C.

13.2 Wider Environment Requirements

- **WER.R.1** The system shall integrate safely within digital ecosystems, ensuring that its automation does not interfere with other running applications.
- **WER.R.2** The system shall be designed to minimize environmental impact by reducing unnecessary computational load and optimizing resource usage.

13.3 Requirements for Interfacing with Adjacent Systems

- **IAS.R.1** The system shall provide secure APIs or middleware for integration with third-party tools and applications (e.g., email, web browsers, file systems).

- **IAS.R.2** The system shall support standard communication protocols such as HTTPS, MCP and RESTful API calls.
- **IAS.R.3** The system shall interact with speech recognition engines (e.g., Whisper, Google ASR) and text-to-speech systems through well-defined interfaces.
- **IAS.R.4** The system shall log interactions with external systems for traceability and debugging.

13.4 Productization Requirements

- **PRD.R.1** The system shall be packaged as a deployable desktop or web-based application with minimal setup steps.
- **PRD.R.2** The product shall include installation documentation and user onboarding instructions in the github repo
- **PRD.R.3** The design shall allow modular upgrades, enabling future integration of improved AI models or additional tools.

13.5 Release Requirements

- **REL.R.1** The releases will follow the MAJOR.MINOR.PATCH versioning scheme.
- **REL.R.2** The release package shall be version-controlled and archived for reproducibility.

14 Maintainability and Support Requirements

14.1 Maintenance Requirements

- **1:** The project shall include a short README and setup steps so new contributors can build and run it without help.
- **2:** Adding a new MCP tool shall only require a new module and a registry entry; existing tools do not need to be changed.

14.2 Supportability Requirements

- **1:** The release shall include a simple user guide (quick start, common commands, basic troubleshooting).
- **2:** The app shall offer a Report Problem button that bundles recent logs and basic system info into a ZIP with the user's consent.

14.3 Adaptability Requirements

- **1:** The system shall run on current versions of Windows and macOS on standard hardware.
- **2:** The system shall keep working if the user switches default apps by updating configuration only, no code changes.

15 Security Requirements

15.1 Access Requirements

1. **Open Access:** The application will be available publicly and no user registration or account is required.
2. **Permission Access:** Any necessary permissions, such as microphone or screen access, must be explicitly granted by the user.

15.2 Integrity Requirements

1. **Data Integrity:** The system shall ensure that any stored or transmitted data is protected from unauthorized alteration or corruption.

15.3 Privacy Requirements

1. **User Data Storage:** Any stored user data will be stored acording to PIPEDA.
2. **Third Party Data Storage:** If any external APIs are used, only minimal, non identifiable data may be transmitted. These services must adhere to GDPR. The user must also explicitly accept the privacy policies of these third party services.

15.4 Audit Requirements

There are no audit requirements for the application. The application does not store user data, maintain user accounts or perform any actions that require tracking or verification. There is nothing that requires audits or security reviews. The privacy focused design eliminates the need for audit trails.

15.5 Immunity Requirements

1. **Secure Practices:** The application must be to prevent DDos Attacks, SQL injection, and unauthorized access.

2. **Third Party Safety:** All third party libraries and dependencies must be up to date and scanned for vulnerabilities.

16 Cultural Requirements

16.1 Cultural Requirements

1. **Respectful and Neutral Interaction:** The application shall use polite, culurally neutral language, avoid slangs and any references that may confuse or offend users.

2. **Language Support:** The application shall support multiple languages and will allow users to interact in their preferred language.

17 Compliance Requirements

17.1 Legal Requirements

1. **Adherence to PIPEDA:** The application will adhere to PIPEDA for any stored user data.

2. **Copyright and IP:** The application shall be distributed under the MIT license; allowing for free use, modification and distribution as long as the original license and copyright notice is also included.

17.2 Standards Compliance Requirements

1. **Adherence to Accessibility Standards:** The application will follow WCAG 2 to ensure accessibility for users with disabilities.
2. **Adherence to Coding Standards:** The application should follow ISO 25010 and ISO 27001 to ensure delivery of high quality and secure software.

18 Open Issues

- **OI.1** The integration of the MCP (Model Context Protocol) for controlling system-level functions is still in progress, and further testing is needed to ensure reliable and safe execution of commands across different applications.
- **OI.2** The system's compatibility and consistent performance across different operating systems (such as Windows, macOS, and Linux) have not yet been fully tested or confirmed.
- **OI.3** Ensuring that user commands do not unintentionally trigger harmful or unauthorized actions on the device is still under investigation and requires additional safety checks and permission handling.

19 Off-the-Shelf Solutions

19.1 Ready-Made Products

Several existing tools can cover key parts of Proxi so we do not reinvent them:

- **Speech-to-Text (STT):** Many STT engines can perform microphone input and return text with timestamps. A lot of computers have a built in STT software that is free to use.
- **Text-to-Speech (TTS):** Open AI has its own TTS model that can be used to convert text to speech, such as GPT-4o mini TTS.
- **Desktop Automation:** OS-level automation, such as lightweight automation libraries, can open apps, switch windows, and engage with controls.

- **Language Understanding:** An API to a LLM such as GPT-4o Mini can be the head AI agent for understanding user commands and generating plans.
- **Storage and Logging:** A lightweight database (SQLite) or a log file can have audit logs and MCP settings with very minimal setup.

19.2 Reusable Components

We can reuse or lightly adapt components that are common across desktop assistants:

- **MCP tool skeletons:** A small template for tools and a registry storer so new tools are added by convention.
- **OS adapters:** Adapters for Windows/macOS that present the same functions (open app, focus on window, file operations) behind one single interface.
- **Prompt templates:** We can reuse existing templates for short, reusable prompts to open apps, open files, browse/search, and save/move/rename.

19.3 Products That Can Be Copied

We shall adopt proven patterns from familiar tools to speed delivery:

- **Command palette pattern:** A single search box that finds actions and recent items, having arrow-key selection and enter to run it.
- **Clear confirmation modal:** A dialog that states the action in simple language, showing what will change, and offers options for approve/cancel.
- **Pre-run checklist:** A simple setup that asks for microphone permission, automation permission, and desktop control on first use. Whenever app access is needed, permission will be simply requested from the user once again for its access before executing any further commands.

20 New Problems

20.1 Effects on the Current Environment

The introduction of the AI-powered assistive technology platform may alter how users interact with their existing computing environment. Potential effects include:

- Users may become dependent on natural language commands, reducing direct interaction with traditional interfaces.
- Additional system resources (CPU, memory) will be used to run AI agents, which could impact performance of other applications.
- There may be changes in user workflows, requiring adaptation to the system's operational model.
- The presence of AI-driven agents could introduce unexpected interactions with other software if not properly sandboxed.

20.2 Effects on the Installed Systems

Integrating the system with existing hardware and software may create compatibility challenges:

- Existing applications may require updated permissions or APIs to allow agent interactions.
- System updates (OS or applications) may temporarily disrupt AI agent functionality.
- The system may necessitate installation of additional software libraries or frameworks, which could conflict with pre-existing software.
- Increased network usage for cloud-based AI services could affect other applications relying on the same connectivity.

20.3 Potential User Problems

While the system is designed to enhance accessibility, users may encounter challenges:

- Misinterpretation of voice commands may lead to incorrect task execution.
- Users may feel discomfort or distrust if the system performs unexpected actions.
- Cognitive overload may occur if the system provides too much feedback or prompts.
- Users with atypical speech patterns or accents may experience reduced recognition accuracy.

20.4 Limitations in the Anticipated Implementation Environment That May Inhibit the New Product

Certain environmental and technical limitations may restrict the effectiveness of the platform:

- Limited or unstable internet connectivity could hinder cloud-based speech recognition or AI planning.
- Hardware constraints (low-end processors, insufficient RAM) may slow down task execution.
- Applications with restrictive APIs/security measures may not allow full agent interaction.

20.5 Follow-Up Problems

After deployment, additional challenges may emerge that require monitoring and mitigation:

- Continuous maintenance will be needed to ensure compatibility with updated applications and OS versions.
- Security and privacy risks from storing or processing user data must be actively managed.

21 Tasks

21.1 Project Planning

1. The project will follow a iterative and incremental development plan. Early prototyping along with continuous feedback from beta users will help address defects early.

2. Core stages will include requirements, system design, development, testing, and deployment.

3. Developers will be responsible for coming up with test cases (unit and functional tests), so sufficient time should also be allocated to writing test cases.

21.2 Planning of the Development Phases

Since development will be interative and incremental, the phases are not strictly linear. Development and testing, in particular, will be repeated iteratively as part of an incremental cycle. The Development phases will be broadly divided into these phases:

- **Requirements Phase**
  - Identify what the system must do and it's constraints.
  - Come up with Functional, Non-Functional requirements.
  - Gather input from target users.
- **System Design Phase**
  - Create overall system architecture, high level component interactions and integrations.
  - Select appropriate off-the-shelf LLM model for natural language processing and logic
  - Brainstrom and pick appropriate technology stack.
  - Create diagrams, mockups and data flow diagrams to aide development.
- **Development Phase**

  - Implement system according to design and feedback from previous iterations.
  - Build the user interface using selected technology stack and integrate partner applications and technologies.
  - Come up with automated and manual test cases for any new features added or any change in functionality. These can be added to the automated testing pipelines.

- **Testing Phase**
  - Create and conduct comprehensive test suites that include testing objectives, failures, successes and stability.
  - Test the system with representative users, including older adults and users with disabilities, to confirm the interface is intuitive and meets accessibility standards.
  - Evaluate responsiveness and stability across different hardware setups, including low-spec devices.
  - Ensure regressions aren't introduced between successive iteractions and that new changes don't break existing features.
  - Document all testing results and come up with possible improvements for next development iteration.

- **Deployment Phase**
  - Ensure that instruction manuals are up to date with latest features.
  - Ensure installation and initial setup are simple and do not require technical expertise.
  - Deploy the application using the selected channel. Continuously monitor system performance and user feedback for maintenance and updates.
  - Maintain version control and update logs so improvements and bug fixes can be tracked systematically

22 Migration to the New Product

22.1 Requirements for Migration to the New Product

There are no special migration steps beyond just installing Proxi and granting the usual permissions on the first run (microphone, automation, etc.). After that, for access to any apps and folders, Proxi will request permission again. Existing workflows and applications will remain the same; users only need to familiarize themselves with the new voice/text command interface.

22.2 Data That Has to be Modified or Translated for the New System

No data conversion is needed. This tool works with the user's existing files and applications as they are. The only new data created is potentially a local configuration file generated. Nothing else needs to be imported or translated from previous tools.

23 Costs

To minimize costs, the project will aim to use open source and off-the-shelf solutions whenever possible. Since this project is a student project as part of a course, development costs will not be taken into consideration. The only costs associated with development are related to any third-party libraries or any APIs used by the application. The application itself will be free to use and publicly available.

24 User Documentation and Training

24.1 User Documentation Requirements

- The system shall include a user manual explaining installation, setup, and configuration procedures.
- The documentation shall provide step-by-step instructions for performing common tasks via speech and text commands.

- The manual shall include troubleshooting guidance for common issues such as misinterpreted commands, failed task execution, or connectivity problems.
- The documentation shall provide an explanation of accessibility features, including options for speech rate, font size, and interface contrast.
- The system shall include inline help or tooltips within the interface for immediate guidance during use.
- Documentation shall be available in multiple formats, including PDF, HTML, and context-sensitive help within the software.

24.2 Training Requirements

There isn't a dedicated training program for the system but the user will be provided with example scenarios that they can follow to build intuition on how to direct the agentic system.

- Users shall be able to complete an introductory tutorial that demonstrates how to issue commands and interact with the AI agents.
- The system shall provide interactive training sessions or guided exercises to familiarize users with core functionalities such as sending emails, managing files, and performing web searches.
- Training materials shall include examples of natural language commands, including variations in phrasing to account for diverse user speech patterns.

25 Waiting Room

These are good ideas we're keeping on the side for now, as they are not the most urgent.

- **Offline mode:** Basic voice commands without internet using a small local model.
- **App plugins:** Simple SDK so other people can add new MCP tools.

- **Interactions with multiple devices:** Start a task on one computer and finish on another. For example, computer to computer connection.
- **Custom voice:** Let users pick a voice for Text-to-Speech. A lot of TTS engines have many voices to choose from. For now we shall keep it simple with one voice.
- **More languages:** Support more languages for voice commands and responses.

26 Ideas for Solution

Proxi should stay simple and safe. The main idea is a small desktop app that listens for a request, confirms anything risky in plain language, and then runs a scoped action through a set of MCP tools.

- **Local MCP server:** A set of processes expose a bunch of safe tools (open app, open file, browse, save/move/rename, click buttons on screen, etc.). Each tool runs with low privilege and only within user approved scopes
- **Lightweight UI:** A simple AI with options for users click, type, or speak. The UI shows status (listening, thinking, ready), captions for spoken output, and a few big buttons (listen, stop, undo, approve).
- **Voice in, text out (with captions):** Use a STT engine for microphone input and TTS for spoken output. Always show captions to the text that is coming out/in.
- **Simple intent handling:** Map common phrases to a small set of intents using prompts/templates (Open app, Open file, browse, save/move/rename). If unclear, ask one short follow up.
- **Safety by default:** Permission scopes (folders/apps) are chosen at setup and can be edited or approved by the user later. Destructive operations always require an explicit approval from the user.
- **Traceability:** Keep a short, local audit log. Any sensitive information should be kept out of the audit logs.

Appendix — Reflection

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. How many of your requirements were inspired by speaking to your client(s) or their proxies (e.g. your peers, stakeholders, potential users)?

4. Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project.

5. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

6. 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?

27 Reflections

27.1 Amanbeer Minhas Reflection

1. **What went well while writing this deliverable?**
   One thing that went well while writing this deliverable was that I had a much clearer understanding of what was expected compared to our previous submissions. The Volere template helped a lot because it gave a clear structure and description of what needed to be included in each section, so I was able to focus on writing instead of figuring out the format. As a team, we also communicated better and stayed consistent in how we wrote different sections, which made the final document more cohesive.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   The biggest challenge we faced was dividing up the work evenly across the team. At first, some team members started earlier and took on more tasks, while others joined later, leading to an imbalance in workload. This caused some sections to be rushed near the end. To resolve this, we agreed to plan task assignments ahead of time for future deliverables so that everyone has clear responsibilities and the work is distributed more fairly. This will help us stay more organized and avoid last-minute issues.

3. **How many of your requirements were inspired by speaking to your client(s) or their proxies?**
   It is hard to put an exact number on how many requirements were inspired by client conversations, but several key ones were shaped by those discussions. Our focus on accessibility and ease of use came directly from client feedback about making the system usable for people with motor or visual impairments. Additionally, requirements around safety, such as confirming system-level actions before execution, were influenced by stakeholder input about avoiding accidental or harmful commands.

4. **Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project?**

Several courses I have taken will directly support our work on this project. Software Requirements (SFWRENG 3RA3) taught me how to gather and define clear requirements, which is essential for this deliverable. Software Architecture and Design courses will help with structuring the system and designing modular components. Human-Computer Interfaces will be useful for designing an accessible and user-friendly interface. Finally, our programming and systems courses like Object-Oriented Programming and Concurrent Systems Design provide the technical foundation for building and connecting different parts of the system.

5. **What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project?**
   As a team, we will need to gain deeper knowledge of how to integrate the MCP (Model Context Protocol) effectively to control system-level actions. We also need to improve our understanding of operating system differences to make sure the solution works across Windows, macOS, and Linux. On the skills side, we will need to strengthen our testing practices, improve our technical writing for documentation, and enhance our team coordination and time management skills. Individually, I want to focus on improving my testing and integration skills, as that will be important for the reliability of our system.

6. **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? Which will each team member pursue, and why did they make this choice?**
   To improve our knowledge of MCP and system-level integration, we can study official documentation and build small prototype projects to experiment with its capabilities. We can also seek help from online forums and developer communities to learn from real-world use cases. For cross-platform knowledge, we can test our software on multiple operating systems throughout development and review OS-specific guidelines. To improve testing skills, we can practice writing unit and integration tests for smaller modules early in the project and use tools like automated testing frameworks. For writing and documentation, we plan to review examples of high-quality documentation and get feedback from instructors and TAs. I will personally focus on building prototypes to

learn MCP integration and writing more automated tests, as these are most relevant to the parts of the project I am contributing to.

27.2 Gourob Podder Reflection

1. **What went well while writing this deliverable?**
   While writing this deliverable, we were able to clearly define the functional and operational requirements of our AI-powered assistive technology platform. Structuring requirements as black-box and verifiable statements ensured clarity and made it easier to anticipate testing criteria. Breaking down the system into modular components such as input handling, agent planning, and feedback mechanisms allowed us to comprehensively cover potential user interactions and system behaviors. Additionally, using a structured LaTeX format helped maintain consistency across sections.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   A key challenge was ensuring that all requirements were verifiable and free of design decisions. Initially, some requirements were too abstract (e.g., "the system shall be user-friendly"), which was not testable. We resolved this by reframing each requirement to include measurable criteria, such as task completion rates, recognition accuracy, or response times. Another pain point was anticipating all potential user problems and environmental constraints without making assumptions. We addressed this by reviewing accessibility guidelines, examining similar assistive technologies, and considering diverse user scenarios, including elderly users and people with disabilities.

3. **How many of your requirements were inspired by speaking to your client(s) or their proxies?**
   We don't have an exact measure of client impact, but the core requirements were either constructed or revised based on talking to potential clients. Discussions with accessibility-focused community members provided insight into what tasks are most important, potential user errors, and expectations for feedback mechanisms. This input was crucial for shaping functional requirements such as conversational memory and clarification prompt features.

4. **Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project?**
   Several courses will directly support our work on this project:

   - **Human-Computer Interaction (4HC3):** Understanding usability principles and accessibility guidelines.
   - **Software Engineering Requirements (3RA3):** Writing SRS, functional and non-functional requirements, and system design.
   - **Operating Systems (3SH3):** Managing agent execution, concurrency, and integration with host systems.
   - **Databases and Networking (4C03):** Handling logging, cloud-based APIs, and data persistence for the system.

5. **What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project?**
   To successfully complete this project, the team will need:

   - **Natural Language Processing (NLP) knowledge:** Understanding speech-to-text, intent recognition, and dialogue management.
   - **Agent-based system design:** Planning, execution, and tool integration for autonomous agents.
   - **Accessibility and usability skills:** Designing for diverse users, including those with disabilities or low technical proficiency.

6. **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? Which will each team member pursue, and why did they make this choice?**
   To acquire these skills:

   - **NLP knowledge:**
     (a) Complete online tutorials and courses on speech recognition and intent detection.
     (b) Implement small prototypes of text and speech processing modules for hands-on experience.

- **Agent-based system design:**
  (a) Study academic papers and open-source projects on multi-agent systems.
  (b) Develop proof-of-concept agents in a controlled environment to understand task planning and execution.
- **Accessibility and usability skills:**
  (a) Review accessibility guidelines such as WCAG 2.1 and analyze existing assistive software.
  (b) Conduct user testing sessions with peers or volunteers to identify usability issues.

Each team member will focus on specific areas while gaining a breadth-first understanding of other areas. In my case, I have prior knowledge with NLP and basic agentic systems, so I will focus on reviewing accessibility studies related to software design.

27.3 Ajay Grewal Reflection

1. **What went well while writing this deliverable?**
   We agreed early on what Proxi should do, which is to turn prompts from a user into safe actions. That made the writing easier and overall requirements easier to understand and write down. This resulted in the sections coming together quickly. We also had good team communication, with everyone contributing ideas.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   We kept slipping into how this would be built instead of what Proxi would do. We fixed this by asking ourselves if this can be ideated properly without going too deep into the technology side. We also had many LaTeX issues when writing this regarding formatting. This was resolved through heavy trial and error.

3. **How many of your requirements were inspired by speaking to your client(s) or their proxies?**
   It is difficult to put an exact number on this, but many important ones were from those discussions. Making Proxi accessible and simple

came from obvious client needs. Safety requirements, such as approving system level actions, were also heavily influenced by stakeholders.

4. **Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project?**
   Data structures and algorithms will help us in having efficient code and data processing. Software requirements will help in defining structured requirements. Human computer interactions will help in making an effective and simple to use UI design for Proxi.

5. **What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project?**
   Better understanding of MCP tools, how to do OS automation, and how to build reliable tests/logs for our software. Personally, I really want to get better at MCP and OS automation as this will define how well Proxi works. We will also need to get better at writing tests and logs to ensure Proxi is reliable and safe. We also need to better work as a group to ensure the code work is separated well and everyone is contributing equally.

6. **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? Which will each team member pursue, and why did they make this choice?**
   Firstly, for this to work, a lot of trial and error will need to happen. Multiple prototypes will have to be built to see what works best, furthering our understanding of how MCP, and OS automation works, as well as writing effective test cases. I and each team member will also watch and read tutorials on MCP and OS automation to further understanding of these topics. This way we can all equally contribute to the technical design of Proxi.

27.4 Savinay Chhabra Reflection

1. **What went well while writing this deliverable?**
   The templates helped a lot as these explained what was expected of us. The weekly team meetings also helped us with coordinate with other team members for inspiration and ideas.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   Dividing the work proved challenging once again. The previous deliverable also suffered from an imbalanced workload. The problem is that we don't know the time commitment each section requires until we actually do that section. We split out the workload based on what we thougth were our individual strengths however sections are not necessarily even in their workload. Another challenge was staying solution neutral while coming up with requirements.

3. **How many of your requirements were inspired by speaking to your client(s) or their proxies?**
   I don't recall excatly how many requirements were inspired by the clients, but the core requirements were made after understanding the needs of older adults. Other requirements were inferred from general best practices and what we thought would make sense.

4. **Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project?**
   SFWRENG 2AA4 and SFWRENG 3RA3 are two course in particular that will help our team to be successful. These courses include requirements and software delivery which are particularly important for this course.

5. **What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project?**
   As a team, we will need to learn more about MCP and how to use it to control the user's system. We will also need to learn about LLM integration and voice to text features. We will also need to upskill in the testing domain as none of us have exhaustive knowledge on end-to-end system testing.

6. **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? Which will each team member pursue, and why did they make this choice?**
   For MCP, documentation and smaller projects are good approaches to familiarize ourselves with the technology. For LLM and Voice to speech

integrations, online courses and exmaple projects on github are good real world examples we can learn from. Projects are always a good learning method as they provide hands on learning.