Module Guide for Software Engineering

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

January 21, 2026

1 Revision History

Date	Version	Notes
2025-11-13	1.0	Initial draft created and wrote the module decomposition and connections with requirements
2025-11-13	1.1	Finalized the document and created the module hierarchy diagram
2026-01-21	1.2	Rev 0, implemeneted TA feedback

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

Additional symbols, abbreviations, and acronyms specific to this document are listed below.

Symbol / Term	Definition
$:=$	Assignment operator used for state transitions.
$\geq, \leq$	Greater than or equal to, less than or equal to.
$\rightarrow$	Function mapping or transition arrow.
Action	Atomic executable system behaviour.
BH	Behaviour-Hiding module (e.g., BH-Input, BH-Plan).
ExecStatus	Execution result (Pending, Success, or Failed).
FUNC.R.#	Functional requirement number from the SRS.
Hazard ID	Identifier from Hazard Analysis (e.g., H1, H2).
HH	Hardware-Hiding module (e.g., HH-IO, HH-Auto).
InputMode	Input type (VoiceOnly, TextOnly, Mixed).
Intent	Structured interpretation of user command text.
MCP	Modular Command Protocol agent interface.
OutputMode	Output type (VoiceOnly, TextOnly, Both).
Plan	Structured action sequence from BH-Plan.
QA	Quality Assurance (software testing process).
RiskLevel	Safety classification for user actions.
SD	Software-Decision module (e.g., SD-Types, SD-Log).
SRS	System Requirements Specification.
STT	Speech-to-Text (audio input converted to text).
TTS	Text-to-Speech (text output converted to speech).
UI	User Interface.
V&V	Verification and Validation Plan.

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes traceability tables. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running (e.g., microphone or speaker models).

AC2: The format or source of input data (e.g., switching from voice-only to mixed input modes).

AC3: The speech recognition or text-to-speech library (e.g., replacing Whisper API with a local model).

AC4: The user interface layout or feedback presentation (e.g., visual vs. voice-only output).

AC5: The set of supported voice commands or MCP tools.

AC6: The thresholds for speech detection and timing between input and response.

AC7: The risk classification policy for safety confirmation (e.g., adding new risk levels or modifying approval logic).

AC8: The logging format or storage mechanism for session history and user preferences.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The main processing loop of Input $\rightarrow$ Interpret $\rightarrow$ Plan $\rightarrow$ Execute $\rightarrow$ Feedback.

UC2: The core data structures used for storing Commands, Intents, and Action Plans.

UC3: A local, reliable, request-response communication style between modules via the MCP agent interface (structured JSON requests and responses).

UC4: The product runs as a single desktop application in one process (no distributed deployment across devices).

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table **??**. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: HH-IO (Audio I/O Adapter) manages microphone input and audio output across platforms.

M2: HH-Auto (Desktop Automation) performs desktop automation tasks such as typing, clicking, and launching applications.

M3: BH-Input (Voice & Text Manager) captures user speech, uses Whisper STT, and normalizes input.

M4: BH-NLU (Intent Parser) interprets text into structured intents based on command patterns.

M5: BH-Plan (Task Planner and Executor) selects MCP tools and executes user intents.

M6: BH-Safety (Safety Confirmation Gate) validates risky actions and requests confirmation as needed.

M7: BH-Session (Session Context Manager) stores history, session context, and undo information.

M8: BH-Feedback (Feedback Output Manager) outputs responses using TTS or visual text.

M9: BH-UI (User Interface Panel) displays status, confirmation prompts, and results.

M10: SD-Types (Core Data Types) defines ADTs for Command, Intent, Plan, and related structures.

M11: SD-ToolRegistry (Tool Registry) maps intents to MCP tools and actions.

M12: SD-Store (Persistent Store) manages persistent data such as preferences, session history, and logs.

M13: SD-AIClient (AI Service Client) provides configuration and API communication for STT, TTS, and LLM services.

M14: SD-Log (Event Logger) records events for debugging and V&V traceability.

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 1.

- **Multi-modal Input Pipeline**: FUNC.R.1 and FUNC.R.2 required handling both speech and text input with accurate conversion. This led to a coordinated design where **BH-Input** manages high-level input processing, **HH-IO** handles hardware-level audio operations, and **BH-UI** provides the interface layer, with **SD-AIClient** ensuring proper speech configuration and service calls.

- **Natural Language Understanding Chain**: FUNC.R.3's intent interpretation requirement resulted in a processing chain where **BH-NLU** performs core parsing, supported by **BH-Input** for pre-processed text and **SD-Types** for structured data definitions.

- **Comprehensive Task Execution Framework**: FUNC.R.4's planning and execution requirement necessitated a multi-module approach where **BH-Plan** coordinates execution, **SD-ToolRegistry** maps intents to tools, **HH-Auto** performs low-level actions, **BH-Session** maintains context, and **BH-Safety** ensures secure operation.

- **Distributed Logging Architecture**: FUNC.R.7's comprehensive logging requirement is satisfied by making **SD-Log** a central service used by nearly all modules, with **SD-Store** handling persistent storage of log data.

- **High-Level Interaction Abstraction**: FUNC.R.8's requirement to avoid low-level OS details is achieved through a layered architecture where **BH-UI**, **BH-Input**, and **BH-Feedback** provide natural language interfaces, while **BH-Plan** and **SD-Types** maintain high-level abstractions that shield users from system complexities.

- **Safety-Critical Confirmation System**: FUNC.R.9's confirmation requirement led to the **BH-Safety** module that intercepts privileged actions, coordinated with **BH-UI** for user prompts and multiple SD modules for persistence and auditing.

- **Speed and Latency (SAL.R.1, SAL.R.2)**: To meet the 2-second response time requirement, we designed **BH-Input** with efficient audio buffering and **BH-Feedback** with immediate acknowledgment patterns. The modular architecture allows parallel processing where possible, and **HH-Auto** is optimized for quick execution of common actions.

- **Safety-Critical Operations (SAF.R.1, SAF.R.2)**: The **BH-Safety** module was specifically created to enforce confirmation requirements for file and system changes. **BH-Session** maintains action history to support single-step undo functionality, working with **SD-Store** for persistent state management.

- **Precision and Accuracy (POA.R.1, POA.R.2)**: Accuracy requirements drove the separation of **BH-Input** for robust speech processing and **SD-AIClient** for fine-tuning recognition parameters. The modular design allows swapping STT engines without affecting other system components.
- **Robustness and Fault Tolerance (ROFT.R.1, ROFT.R.2)**: **SD-Log** provides comprehensive error logging without disrupting operation, while each BH module implements graceful error handling. The use of **SD-Types** ensures data consistency even with invalid inputs.
- **Capacity Requirements (CAP.R.1, CAP.R.2)**: The modular design enables efficient resource management, with **BH-Plan** optimizing tool execution and **SD-ToolRegistry** enabling lazy loading of infrequently used components to maintain low CPU usage during extended operation.
- **Scalability and Extensibility (SOE.R.1, SOE.R.2)**: The plugin-like architecture of **SD-ToolRegistry** allows new tools to be added seamlessly, while the clear interfaces between modules ensure that new features can be integrated without major refactoring.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)-(M2)

The hardware-hiding modules isolate platform and device differences from the rest of the system. They provide stable interfaces for audio I/O and desktop automation so higher-level modules do not depend on OS or device details.

7.1.1 HH-IO: Audio I/O Adapter (M1)

Secrets: The data structure and algorithm used to interface with microphone and speaker hardware.

Services: Provides audio hardware I/O operations including opening and closing devices, recording audio, and playing audio.

Implemented By: OS audio libraries

Type Of Module: Library

7.1.2 HH-Auto: Desktop Automation (M2)

Secrets: OS specific automation (mouse and keyboard control) implementation. Hides differences between Windows, macOS, and Linux.

Services: Move cursor, click, type, and launch applications.

Implemented By: PyAutoGUI

Type Of Module: Library

7.2 Behaviour-Hiding Modules (M3)-(M9)

The behaviour-hiding modules implement the end-to-end interaction pipeline: capture input, interpret intent, plan actions, enforce safety, manage session context, and present feedback to the user.

7.2.1 BH-Input: Voice and Text Manager (M3)

Secrets: Speech-to-text configuration, text normalization rules, and buffering strategies.

Services: Handles speech variation, different speech patterns, varied accents, and filters background noise.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.2.2 BH-NLU: Intent Parser (M4)

Secrets: Rules and internal parameters for interpretation of noisy or imperfect text into structured intent objects.

Services: Converts normalized text into intents with metadata fields.

Implemented By: Software Engineering

Type Of Module: Library

7.2.3 BH-Plan: Task Planner and Executor (M5)

Secrets: Decision-making logic for selecting the correct MCP agent based on intent, execution planning, and error handling.

Services: Creates an execution plan based on intent, executes the plan using MCP tools, and tracks tasks to completion.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.2.4 BH-Safety: Safety Confirmation Gate (M6)

Secrets: Risk analysis policy table, irreversible detection rules, and timeout behaviour.

Services: Classifies action risks, decides an appropriate policy, and prompts the user for confirmation when an action is deemed high risk.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.2.5 BH-Session: Session Context Manager (M7)

Secrets: Session history table, short-term conversational context for the AI model, and previous action history for undo.

Services: Tracks sessions and actions. Maintains continuity across user requests and stores short-term model context in memory.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.2.6 BH-Feedback: Feedback Output Manager (M8)

Secrets: Output action logic, text-to-speech configuration, and message delivery monitoring.

Services: Converts messages to speech and outputs through speakers if required, and mirrors speech as text feedback.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.2.7 BH-UI: User Interface Panel (M9)

Secrets: View groups, information presentation rules, and prompt timing rules.

Services: Updates UI state, displays messages, and presents user prompts.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.3 Software Decision Modules (M10)-(M14)

The software-decision modules define shared data representations and system-wide services such as tool mapping, persistence, AI service integration, and logging.

7.3.1 SD-Types: Core Data Types (M10)

Secrets: Internal representation of commands, actions, risks, policies, and tool metadata.

Services: Defines shared ADTs used system-wide.

Implemented By: Software Engineering

Type Of Module: Abstract Data Type

7.3.2 SD-ToolRegistry: Tool Registry (M11)

Secrets: Mapping from intent types to MCP tools, agents, and automation routines.

Services: Provides the appropriate MCP tool or agent for a given intent.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.3.3 SD-Store: Persistent Store (M12)

Secrets: Persistent storage format and schema for user settings, session history, and logs.

Services: Saves and loads user settings and previously saved states from persistent storage. (Short-term model context is handled by BH-Session.)

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.3.4 SD-AIClient: AI Service Client (M13)

Secrets: API keys, configurations, timeouts, retry policy, and fallback options for external STT, TTS, and LLM services.

Services: Provides the communication layer for AI services and exposes stable routines for transcription and speech synthesis to BH modules.

Implemented By: Software Engineering

Type Of Module: Abstract Object

7.3.5 SD-Log: Event Logger (M14)

Secrets: Diagnostic log formatter and persistence policy.

Services: Records events and errors for debugging and V&V traceability.

Implemented By: Software Engineering

Type Of Module: Abstract Object

8 Traceability Tables

This section shows traceability tables between the modules and the functional requirements plus nonfunctional requirements, and between the modules and the anticipated changes. Tables 1, 2, 3, and 4 collectively provide coverage and change-impact traceability.

Table 1: Traceability between functional requirements and modules

Requirement (SRS)	Modules
FUNC.R.1 - Accept input via speech and text	M3, M1, M9, M13
FUNC.R.2 - Convert speech to text (STT)	M3, M1, M13, M14
FUNC.R.3 - Interpret user intent from NL input	M4, M3, M10, M14
FUNC.R.4 - Plan and execute tasks via agents/tools	M5, M11, M10, M2, M7, M6, M14
FUNC.R.5 - Provide textual/spoken feedback	M8, M9, M13, M14
FUNC.R.6 - Maintain short-term conversational memory	M7, M12, M3, M5, M14
FUNC.R.7 - Log user interactions and task results	M14, M12, M7, M3, M5, M8, M9, M6, M1, M2
FUNC.R.8 - High-level interaction (no low-level OS details)	M9, M3, M5, M8, M2, M10
FUNC.R.9 - Confirm privileged/system-level actions	M6, M9, M5, M10, M14, M12

Table 2: Traceability between nonfunctional requirements and modules

Nonfunctional Requirement	**Modules**
APP.1–APP.5 (Appearance)	M9, M8, M3, M13, M12
STY.1–STY.3 (Style)	M9, M8
EOU.R.1–EOU.R.3 (Ease of use)	M9, M3, M5, M8, M7
PER.R.1–PER.R.2 (Personalization)	M3, M7, M12, M13
LEA.R.1–LEA.R.2 (Learning support)	M9, M8, M3, M12
UAP.R.1–UAP.R.3 (Understandability and politeness)	M9, M8, M5, M6
ACC.R.1–ACC.R.2 (Accessibility)	M9, M3, M8, M1, M13
SAL.R.1–SAL.R.2 (Speed and latency)	M1, M3, M5, M13, M14
SAF.R.1–SAF.R.2 (Safety-critical behaviour)	M6, M5, M9, M2, M10, M12, M14
POA.R.1–POA.R.2 (Precision/accuracy)	M3, M13, M14
ROFT.R.1–ROFT.R.2 (Robustness/fault-tolerance)	M14, M12, M3, M5, M7
CAP.R.1–CAP.R.2 (Capacity)	M1, M3, M5, M7, M12, M13
SOE.R.1–SOE.R.2 (Scalability/extensibility)	M10, M11, M12, M5
LON.R.1–LON.R.2 (Longevity)	M12, M14, M7, M9
EPE.R.1–EPE.R.4 (Expected physical environment)	M1, M3, M13
WER.R.1–WER.R.2 (Wider environment)	M5, M2, M11, M13
IAS.R.1–IAS.R.4 (Interfaces with adjacent systems)	M3, M8, M5, M11, M13, M14
PRD.R.1–PRD.R.3 (Productization)	M12, M13, M10
REL.R.1–REL.R.2 (Release)	M14, M12
ACS-01–ACS-02 (Access control)	M9, M5, M6, M13, M12
INT-01 (Integrity)	M12, M14
PRIV-01 (Privacy)	M3, M5, M12, M13, M14
IMM-01–IMM-02 (Immunity)	M13, M14

Table 3: Traceability between nonfunctional requirements and modules continued

Nonfunctional Requirements Continued	Modules
CULR-01–CULR-02 (Cultural)	M9, M8, M3
LGL-01–LGL-02 (Legal)	M12, M14, M9
STDCOMP-01–STDCOMP-02 (Standards compliance)	M9, M8, M10, M12

The following table links anticipated changes (ACs) from Section 4 to the modules that hide the corresponding design decisions.

Table 4: Traceability between anticipated changes and modules

Anticipated Change	Modules
AC1 - Hardware platform (microphone, speakers, OS automation)	M1, M2
AC2 - Input format / source (voice-only vs. mixed voice + text)	M3, M9, M1
AC3 - STT/TTS/LLM provider or library	M3, M8, M1, M13
AC4 - UI layout and feedback presentation	M9, M8, M12
AC5 - Supported voice commands and MCP tools	M4, M5, M11, M10
AC6 - Speech detection and timing thresholds	M3, M1, M13
AC7 - Risk classification and safety policy	M6, M9, M10, M12, M14
AC8 - Logging and storage of history/preferences	M12, M14, M7

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

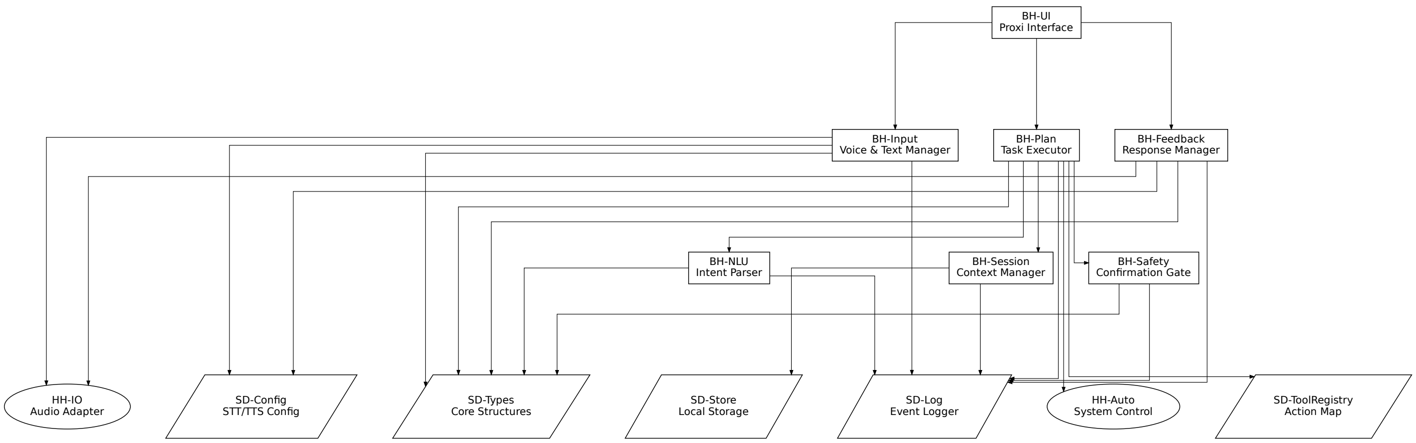

Figure 1: Use hierarchy among modules

10 User Interfaces

This section goes over the user interfaces. The UI is designed to have a voice first interaction while also having keyboard and mouse input as secondary options.

10.1 Main Desktop Panel

The primary interface is a desktop window that can be visible while the user works in other applications. It can also run in background and will listen to user. It gives:

- A clear indication of whether the program is idle, listening, processing, or waiting for any confirmation.
- A single prominent control to start/stop voice capture.
- A transcript area showing recent commands and responses (captioning of spoken feedback).
- Simple, high-level controls for undoing, approving, or cancelling actions.

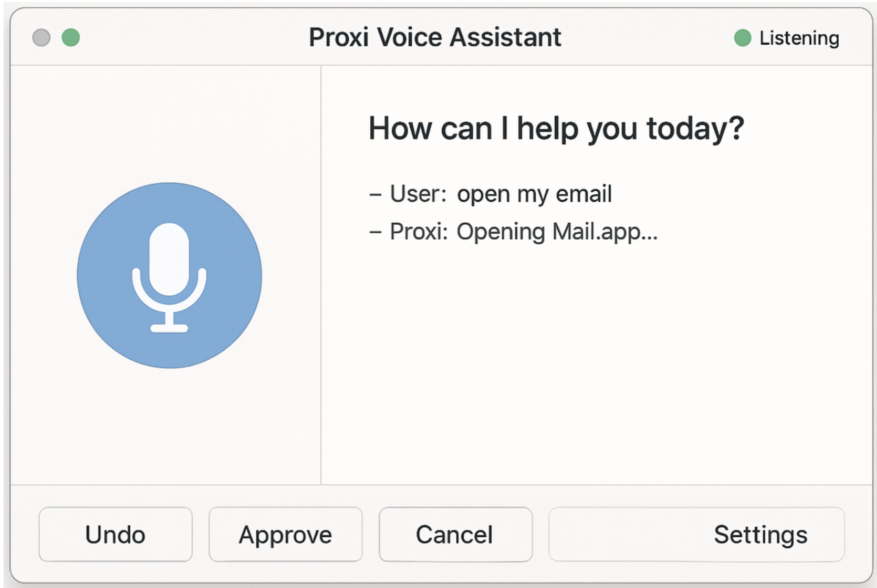

Figure 2: Mockup of the main Proxi interface

10.2 Layout and Elements

The window is divided into three regions:

Top bar: Shows the application name (Proxi Voice Assistant) and a status label (Idle, Listening, Processing, etc.)

Center: A large Mic control to start/stop capture and a transcript area that displays the most recent user command and Proxi's response.

Bottom row: Action buttons for Undo, Approve, and Cancel, plus a Settings control.

10.3 Accessibility Notes

Voice parity: Every visible action (Listen, Undo, Approve, Cancel) also has an equivalent voice command handled by the BH modules.

Keyboard access: All controls are reachable via keyboard focus and shortcuts so users do not need a mouse.

Captioning: Spoken feedback is always mirrored as text in the transcript area to satisfy captioning requirements.

11 Design of Communication Protocols

No custom low-level communication protocols are defined.

11.1 Internal Communication

All modules run in a single desktop application and communicate through normal procedure calls and shared data structures. No network protocols are used in internal modules.

11.2 External Services

- **STT/TTS/LLM services**: M13 provides the API client layer used by M3 and M8 over standard HTTPS with JSON payloads.
- **MCP tools and automation**: M5 uses MCP tools and desktop automation through a local MCP channel exchanging JSON requests and responses. Tool mapping is maintained by M11.

12 Timeline

This schedule decomposes the development of Rev 0 into four distinct phases, mapping internal procedures to the module hierarchy and assigning team members based on the Module Guide.

12.1 Phase 1: Foundations & Core Data Structures (Weeks 0-3)

Goal: Establish the shared symbolic language, hardware I/O links, and logging infrastructure.

Date	Module(s)	Lead(s)	Details
Nov 15 - Nov 29	M13, M14	Ajay	**Setup:** Environment configuration and implementation of `log_event()` for system-wide traceability (FUNC.R.7).
Nov 30 - Dec 9	M10	Gourob	**Type-Definition:** Coding of global ADTs (`Command`, `Intent`, `Plan`) to ensure type-safety across the pipeline.
Dec 10 - Dec 19	M1	Amanbeer	**Signal-Capture:** Low-level implementation of `init_audio()` and `start_capture()` using hardware-hiding drivers.
Dec 20 - Dec 31	M1	Amanbeer	**Feedback-Link:** Implementation of `play_feedback()` to route synthetic audio to system output devices.

12.2 Phase 2: Semantic Pipeline & NLU (Weeks 4-7)

Goal: Transition from raw to structured user intent and persistent storage.

Week	Module(s)	Lead(s)	Details
Jan 1 - Jan 9	M3	Savinay	**Stream-to-Text:** Integration of Whisper STT; implementation of VAD (Voice Activity Detection) logic.
Jan 10 - Jan 19	M3, M4	Savinay, Gourob	**Normalization:** Unifying Voice and Text inputs into a common `Command` object via `get_input()`.
Jan 20 - Jan 31	M4	Gourob	**Semantic-Parser:** Development of rules to map natural language strings to structured `Intent` objects.
Feb 1 - Feb 9	M11, M12	Ajay	**Action-Map:** Building the `SD-ToolRegistry` to link parsed intents to specific system automation tools.

12.3 Phase 3: Execution, Automation & Safety (Weeks 8-11)

Goal: Procedural planning, secure OS use, and session context.

Week	Module(s)	Lead(s)	Details
Feb 10 - Feb 19	M5	Gourob	**Logic-Sequence:** Implementation of recursive planning to decompose intents into atomic `Action` sequences.
Feb 20 - Feb 28	M2	Amanbeer	**OS-Abstraction:** Development of keyboard and mouse automation wrappers (click, type, launch) for Windows/macOS.
Mar 1 - Mar 9	M6	Savinay	**Gate-Keeper:** Mandatory implementation of `validate_plan()` to intercept high-risk/irreversible actions (FUNC.R.9).
Mar 10 - Mar 19	M7	Ajay	**Context-Sync:** Implementing conversational memory to track history and support "undo" operations.

12.4 Phase 4: Feedback Loop & Integration (Weeks 12-14)

Goal: Closing the user interaction loop and final V&V testing.

Week	Module(s)	Lead(s)	Details
Mar 20 - Mar 30	M9	Savinay	**View-Render:** Finalizing the Desktop Panel; rendering transcript overlays and real-time status indicators.
Mar 31 - Apr 9	M8	Amanbeer	**Voice-Synth:** Integration of TTS to convert system responses into spoken audio via the M1 interface.
Apr 10 - Apr 19	All	Full Team	**V&V:** Execution of the Confidence Protocol; testing latency (SAL.R.1) and safety-interception reliability.

12.5 Verification Milestones

To ensure implementation confidence, the following milestones must be met by Apr 20 - Apr 25:

1. **Control Loop Integrity:** Successful movement from `M1 (Audio)` to `M2 (Automation)` for a "hello world" command (Gourob to Verify).

2. **Safety Intercept:** Verification that M6 (Savinay) halts any plan involving file deletion or high risk action until M9 receives user confirmation.

3. **Performance Gate:** Total system latency from end-of-speech to start-of-action must be less than 8 seconds (Amanbeer/Ajay to verify via logs).

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.