Module Guide for Software Engineering

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

November 13, 2025

1 Revision History

Date	Version	Notes
Nov 11, 2025	1.0	Added modules and Traceability Matrix
Nov 12, 2025	1.0	Added Modules M6-M11
Nov 12, 2025	1.0	Added Intro, Hierarchy Table, AC and UC

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change
FR	Functional Requirement
LF	Look and Feel
PF	Performance
OER	Operational & Environmental
MS	Maintainability & Support
SEC	Security
CUL	Cultural
CPL	Compliance

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). For VoiceBridge, we adopt this decomposition based fundamentally on Parnas's principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. This approach supports design for change, a critical concern in assistive technologies where adaptation to evolving user needs and cutting-edge speech recognition improvements frequently occur, especially in early development phases.

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 8 specifies the connections between the software requirements and the modules. Section 9 gives a detailed description of the modules. Section 10 includes two traceability matrices. One checks the completeness

of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 11 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: Hardware platform and input devices.
Advancements in microphone technology, such as arrays or new form factors, alongside evolving processing hardware, may improve audio quality, reduce costs, and enable integration with wearable devices. These changes influence audio acquisition and preprocessing modules.

AC2: Speech-to-text engine and language support.
The system might expand to support additional languages and dialects beyond English requiring adaptations in ASR models and intent interpretation logic. Continuous improvements in speech recognition accuracy and noise filtering will also necessitate regular updates.

AC3: User profile and personalization management.
Personalization features may evolve to accommodate changing user speech patterns and preferences, demanding updates in model fine-tuning, adaptive prompting based on confidence, and profile management strategies.

AC4: Command mapping and browser automation protocols.
Updates may introduce new commands, integrate with emerging browser APIs, or improve error detection and recovery in command execution modules.

AC5: User interface and accessibility compliance.
Responsive adjustments to UI designs and accessibility layers will be required to stay compliant with evolving WCAG guidelines and to address user feedback for improved usability.

AC6: Error handling and recovery policies.
Classification schemes, messaging protocols, retry/backoff strategies, and compensation mechanisms may be enhanced to increase robustness and user experience quality.

AC7: Session lifecycle and interaction flow.
Changes may extend session duration limits, improve state persistence, and better handle asynchronous or interrupted user interactions.

AC8: Data privacy, audit logging, and security.
Adaptations will be necessary to comply with evolving data privacy laws and security best practices, affecting encryption, audit trail formats, and consent mechanisms.

AC9: Personalization and prompting enhancements.
Enhancements aiming at more context-aware and confidence-driven user assistance will continually refine prompting and instruction modules.

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: Switching to specialized or high-end microphones and audio hardware.
VoiceBridge targets consumer-grade devices using standard microphones, ensuring affordability and accessibility. Moving to uncommon or costly hardware is outside the project scope and is therefore unlikely.

UC2: Offloading speech processing or command execution to external or cloud systems.
To protect user privacy and maintain responsiveness, VoiceBridge performs all key processing locally. Changing this would require major architecture redesign and is not planned.

UC3: Fundamental changes to the speech recognition approach.
The system uses specialized ASR models tuned for speech impairments based on proven models. Major shifts in ASR methodology would be disruptive and are therefore improbable.

UC4: Complete redesign of command mapping and browser control modules.
These modules are tightly coupled with browser APIs and OS functions for stability and usability. Major rewrites would cause extensive system impact and are unlikely.

UC5: Abandoning modular design for a monolithic or radically different architecture.
Modularity was chosen for maintainability and scalability. Changing to a less modular approach would increase complexity and reduce flexibility.

5 Module Hierarchy

This section provides an overview of the module design. The system is organized into architectural layers, each containing modules that encapsulate specific responsibilities. Table 1 presents the hierarchy of modules by their layers, including their primary purposes and responsibilities. The modules listed below, which are leaves in the hierarchy tree, are the modules that will be implemented.

M1: User Interface Module

M2: Accessibility Layer

M3: Feedback Display Module

M4: Speech-to-Text Engine

M5: Intent Interpreter

M6: Command Mapping Module

M7: Command Execution Layer

M8: Error Feedback

M9: Browser Controller

M10: Session Manager

M11: Error Handling & Recovery Module

M12: Data Management Layer

M13: User Profile Manager

M14: Audit Logger

M15: Credential Manager

M16: Encryption Manager

M17: Out-of-Scope Handler

M18: Microphone Manager

M19: VAD Noise Filter

M20: Prompting Module

M21: Model Tuner

M22: Instruction Registry

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.

7 Module Decomposition

Application (Processing) Modules

M7: Command Execution Layer

Secrets: How commands are validated, mapped to backend requests, dispatched, tracked, cancelled, and timed-out. Internal queueing and audit format.
Services: Execute/cancel/status for commands; translate to backend format; rollback hooks.
Implemented By: Core runtime (browser bridge + OS process APIs).

M8: Error Feedback

Secrets: Message templating, localization hooks, mapping of internal error classes to user-facing copy and recovery options.
Services: Show error; show recovery prompt with options; dismiss item; log event.
Implemented By: UI notification layer.

Control (Orchestration) Modules

M9: BrowserController

Secrets: Transport protocol to the browser controller, timeout wrappers, normalization of responses, and session plumbing.
Services: Send request; get status; cancel; open/close controller session.
Implemented By: Automation bridge client.

M10: Session Manager

Secrets: Session lifecycle rules, storage schema, and TTL/expiry handling.
Services: Start/stop session; get session state; attach command; set state.
Implemented By: In-memory cache + persistent store.

Architectural Layer	Included Modules	Purpose / Responsibilities
Presentation Layer	User Interface Module Accessibility Layer Feedback Display Module	Manages direct user interaction and feedback presentation. Displays transcribed text, confirmations, feedback messages, and status indicators. Ensures compliance with WCAG accessibility standards including contrast, font, and color.
Application (Processing) Layer	Speech-to-Text Engine Intent Interpreter Command Mapping Module Command Execution Layer Error Feedback Module	Performs speech recognition and intent processing. Converts speech audio to text, interprets user intent, maps intent to browser or OS commands, executes commands, and handles error feedback.
Control (Orchestration) Layer	Browser Controller Session Manager Error Handling and Recovery Module	Coordinates and manages flow between modules. Tracks session states such as capture, transcribe, confirm, and execute. Manages events, exceptions, retries, and cancellations to ensure smooth operation.
Data Management Layer	Data Storage Manager User Profile Manager Audit Logger	Maintains persistent data and user personalization. Stores user speech samples, preferences, command mappings, transcripts, and logs. Supports diagnostics and evaluation.
Security Layer	Credential Manager Encryption Manager Out-of-Scope Handler	Manages authentication, encryption, and system boundaries. Ensures secure integration with external APIs and operating system interfaces. Handles login confirmations.
Input Processing Layer	Microphone Manager VAD and Noise Filter	Controls microphone hardware and audio capture. Filters noise to improve voice input quality.
Personalization Layer	Prompting Module Model Tuner Instruction Registry	Provides user prompting, model fine-tuning, and instruction management. Supports optional personalization based on confidence scores and usage history.

Table 1: Module Hierarchy

M11: Error Handling & Recovery Module

Secrets: Error classification policy, retry/backoff strategy, and compensation catalog.
Services: Handle error; retry; compensate; classify; record event.
Implemented By: Orchestration runtime with policy store.

8 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 **??**.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

9 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.

9.1 Hardware Hiding Modules (M??)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

9.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

9.2.1 Input Format Module (M??)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

9.2.2 Etc.

9.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

9.3.1 Etc.

10 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR-1 10.1	M18, M19, M4, M10
FR-2 10.2	M4, M19, M21, M12
FR-3 10.3	M1, M2, M3, M8
FR-4 10.4	M5, M6, M22
FR-5 10.5	M7, M9, M11, M8

Table 2: Trace Between Functional Requirements and Modules

Req.	Modules
LF-1 11.1	M1, M2, M3, M8
LF-2 11.2	M1, M2, M20, M3

Table 3: Trace Between Look & Feel Requirements and Modules

Req.	Modules
UH-1 12.1	M1, M2, M3, M8
UH-2 12.2	M13, M12, M5, M6
UH-3 12.3	M1, M2, M20, M10
UH-4 12.4	M8, M3, M2, M20
UH-5 12.5	M2, M1, M3, M8

Table 4: Trace Between Usability & Humanity Requirements and Modules

Req.	Modules
PF-1 13.1	M4, M5, M6, M7, M9
PF-2 13.2	M5, M6, M7, M8, M11
PF-3 13.3	M4, M19, M21, M5
PF-4 13.4	M11, M10, M8, M9
PF-5 13.5	M10, M12, M13
PF-6 13.6	M10, M12, M7, M9
PF-7 13.7	M12, M13, M14, M15

Table 5: Trace Between Performance Requirements and Modules

Req.	Modules
OER-1 14.1	M18, M19, M4, M11
OER-2 14.2	M1, M2, M9, M10
OER-3 14.3	M5, M6, M9, M22
OER-4 14.4	M12, M13, M14, M15, M10
OER-5 14.5	M10, M12, M14, M15

Table 6: Trace Between Operational & Environmental Requirements and Modules

Req.	Modules
MS-1 15.1	M10, M12, M13, M14, M15, M22
MS-2 15.2	M1, M2, M3, M8, M20
MS-3 15.3	M1, M2, M9, M10, M11

Table 7: Trace Between Maintainability & Support Requirements and Modules

Req.	Modules
SEC-1 16.1	M15, M16, M13, M12, M10
SEC-2 16.2	M5, M6, M7, M8, M11
SEC-3 16.3	M12, M13, M14, M15, M16
SEC-4 16.4	M14, M12, M10, M15
SEC-5 16.5	M11, M8, M19, M7, M9

Table 8: Trace Between Security Requirements and Modules

Req.	Modules
CUL-1 17.1	M2, M3, M8, M20, M5

Table 9: Trace Between Cultural Requirements and Modules

Req.	Modules
CPL-1 18.1	M12, M13, M14, M15, M16
CPL-2 18.2	M1, M2, M3, M8

Table 10: Trace Between Compliance Requirements and Modules

11 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.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

Figure 1: Use hierarchy among modules

12 User Interfaces

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]

13 Design of Communication Protocols

[If appropriate —SS]

14 Timeline

[Schedule of tasks and who is responsible —SS]

[You can point to GitHub if this information is included there —SS]

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.