

Module Guide for Software Engineering

Team 13, Speech Buddies

Mazen Youssef

Rawan Mahdi

Luna Aljammal

Kelvin Yu

January 19, 2026

1 Revision History

Date	Version	Notes
Nov 11, 2025	1.0	Added modules and Traceability Matrix
Nov 12, 2025	1.1	Added Modules M6-M11
Nov 12, 2025	1.2	Added Modules M18-M22
Nov 26, 2025	1.3	Added Modules M1-M6
Jan 10, 2026	1.4	Implemented TA Feedback for M7-M11
Jan 10, 2026	1.4	Implemented TA and Peer Feedback for AC/UC distinctions

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
UC	Unlikely Change
FR	Functional Requirement
LF	Look and Feel
PF	Performance
OER	Operational & Environmental
MS	Maintainability & Support
SEC	Security
CUL	Cultural
CPL	Compliance
TLS	Transport Layer Security

Contents

1	Revision History	1
2	Reference Material	2
2.1	Abbreviations and Acronyms	2
3	Introduction	1
4	Anticipated and Unlikely Changes	2
4.1	Anticipated Changes	2
4.2	Unlikely Changes	3
5	Module Hierarchy	4
6	Connection Between Requirements and Design	5
7	Module Decomposition	7
8	Traceability Matrix	17
9	Use Hierarchy Between Modules	19
10	User Interfaces	20
11	Design of Communication Protocols	20
12	Timeline	20
12.1	Development Phases	21
12.2	Task Responsibilities	21

List of Tables

1	Module Hierarchy	6
2	Trace Between Functional Requirements and Modules	17
3	Trace Between Look & Feel Requirements and Modules	17
4	Trace Between Usability & Humanity Requirements and Modules	18
5	Trace Between Performance Requirements and Modules	18
6	Trace Between Operational & Environmental Requirements and Modules	18
7	Trace Between Maintainability & Support Requirements and Modules	19
8	Trace Between Security Requirements and Modules	19
9	Trace Between Cultural Requirements and Modules	19
10	Trace Between Compliance Requirements and Modules	19

List of Figures

1	Use hierarchy among modules	20
---	---------------------------------------	----

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 ?? specifies the connections between the software requirements and the modules. Section ?? gives a detailed description of the modules. Section 8 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 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: Consumer-grade hardware platform and input devices.

VoiceBridge is expected to run on a range of consumer-grade devices (desktops, laptops, tablets) with standard microphones and evolving OS audio stacks. Variations in microphone quality, form factor, and host processing capabilities, as well as future improvements in consumer audio hardware, should be handled by the audio acquisition and preprocessing modules (e.g., Microphone Manager and VAD/NoiseFilter) without impacting higher layers.

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.

Browser capabilities, permission models, and automation interfaces (such as extension manifests and scripting APIs) are expected to evolve over time. New commands, revised command schemas, and updated browser/OS integration will require changes localized to the Command Mapping, Command Execution, and Browser Controller modules, which are designed to isolate such API-level changes from the rest of the system.

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 non-consumer deployment platforms.

VoiceBridge is designed to run on consumer-grade devices using standard microphones and common operating systems. Supporting specialized assistive hardware would require major changes to hardware assumptions, certification, and system integration, and is out of scope for this project.

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 speech recognition models adapted for users with speech impairments. Major changes to the recognition method—such as replacing local processing with a fully cloud-based approach—would be disruptive and are not planned.

UC4: Complete redesign of command mapping and browser control modules.

The system uses a browser-bridge automation pattern, with dedicated modules (Command Mapping, Command Execution, Browser Controller) that encapsulate browser APIs and OS integration. Updates to support new browser APIs or extensions are expected, but replacing the underlying automation architecture is not planned.

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](#).

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

7 Module Decomposition

Presentation Layer Modules

M1: User Interface Module

Secrets:

- Representation of UI state, layout, and visible components
- Mapping of raw browser events into UiEvents
- Management of keyboard and screen-reader focus

Services:

- Initialize and render UI
- Receive and normalize input events into UiEvents
- Route feedback items
- Update keyboard and screen-reader focus

Implemented By: UserInterface module using the browser rendering engine and DOM/event APIs.

M2: Accessibility Layer

Secrets:

- Encoding of accessibility configuration (AccessConfig)
- ARIA attribute mappings
- Validation rules for UI elements against WCAG standards

Services:

- Apply accessibility options
- Announce messages
- Check UI elements and report problems using ValidationReports

Implemented By: AccessibilityLayer module operating on UI elements exposed by UserInterface.

M3: Feedback Display Module

Secrets:

- Internal representation of FeedbackItems keyed by UUID
- Policies for creation, clearance, and recovery options of feedback messages

Services:

- Attach to parent UI
- Show messages of given types
- Clear messages
- Create recovery options for feedback items

Implemented By: FeedbackDisplay module rendering via UserInterface and coordinating with AccessibilityLayer.

Application (Processing) Modules

M4: Speech-to-Text Engine

Secrets:

- Configuration (AsrConfig) and internal processing details of the speech-to-text engine
- Methods for converting audio input into accurate text transcripts

Services:

- Initialize engine
- Process audio to produce transcripts with confidence metadata
- Reset internal state

Implemented By: SpeechToTextEngine module wrapping acoustic and language models.

M5: Command Orchestrator

Secrets:

- Prompt engineering strategies and system instructions for direct command synthesis.
- Session-based conversation context and short-term memory management.
- Heuristics for mapping natural language to browser-specific schemas without a fixed registry.
- Validation logic for ensuring LLM outputs conform to executable security guardrails.

Services:

- Initialize the orchestrator with system prompts and API configurations.
- Synthesize raw speech transcripts directly into structured, validated browser commands.
- Assign confidence scores based on model reasoning.
- Reset session context and clear conversation history.

Implemented By: CommandOrchestrator module using an external LLM API, dynamic context injection, and a JSON schema validation layer.

Application (Processing) Modules

M7: Command Execution Layer

Secrets:

- Command validation and mapping to backend requests
- Dispatching, tracking, cancellation, and timeout logic
- Internal queueing and audit formats

Services:

- Execute, cancel, and provide status for commands
- Translate commands to backend formats
- Provide rollback hooks

Implemented By: Core runtime (browser bridge + OS process APIs).

M8: Error Feedback

Secrets:

- Expected error types
- Message templates and localization hooks
- Mapping of internal/domain errors to user-facing message keys and recovery options
- Redaction rules for logs and UI

Services:

- Display error notifications
- Display recovery prompts with options
- Dismiss items
- Log structured error events
- Normalize and map errors to UI messages and recovery actions

Implemented By: UI Notification Layer and Shared Error Handling Module (M11).

Control (Orchestration) Modules

M9: BrowserController

Secrets:

- Transport protocol to the browser controller
- Timeout wrappers and normalization of responses
- Session plumbing

Services:

- Send request
- Get status
- Cancel operations
- Open and close controller sessions

Implemented By: Automation bridge client.

M10: Session Manager

Secrets:

- Session lifecycle rules
- Storage schema
- TTL and expiry handling

Services:

- Start and stop sessions
- Get session state
- Attach commands
- Set state

Implemented By: In-memory cache + persistent store.

M11: Error Handling & Recovery Module

Secrets:

- Error classification policy
- Retry and backoff strategies
- Compensation catalog

Services:

- Handle error
- Retry and compensate
- Classify errors
- Record events

Implemented By: Orchestration runtime with policy store.

M12: Storage Management Module

Secrets:

- Database connection credentials
- Cloud API tokens
- Encryption keys

Services:

- `DataStorageService`: Stores and retrieves transcripts, user data, and configuration files.
- `BackupService`: Performs scheduled or manual backups to secure cloud storage.
- `DataRetrievalService`: Provides indexed access to stored files for authorized modules.
- `DataRetentionService`: Enforces deletion and retention rules based on time or policy.

Implemented By: `DataStorageManager`.

M13: User Profile Management Module

Secrets:

- User tokens and refresh keys
- Profile encryption keys
- Hashed user identifiers

Services:

- `ProfileCreationService`: Initializes new user profiles with unique IDs.
- `PreferenceService`: Saves and retrieves user preferences and personalization data.
- `ConsentManagementService`: Records and verifies consent for data collection and personalization.

Implemented By: `UserProfileManagerImpl`.

M14: Audit Logging Module

Secrets:

- Log signing key
- Audit database token
- Log encryption key

Services:

- ActivityLogService: Logs system and user activities for traceability.
- LogQueryService: Provides authorized access to query and review logs.
- AnomalyDetectionService: Detects suspicious activity and security anomalies.

Implemented By: AuditLoggerImpl.

M15: Credential Management Module

Secrets:

- Vault master key
- Token signing keypair
- OAuth client secrets

Services:

- AuthenticationService: Validates user credentials during login.
- SessionTokenService: Issues and validates secure access tokens (JWT/OAuth).
- PasswordVaultService: Stores and retrieves hashed passwords securely.

Implemented By: CredentialManagerImpl.

M16: Encryption Management Module

Secrets:

- Private encryption keypair
- TLS certificates for secure communication
- Key rotation schedule metadata

Services:

- DataEncryptionService: Encrypts data at rest and in transit.
- KeyManagementService: Generates, rotates, and revokes cryptographic keys.
- AnonymizationService: Removes identifiable data prior to model training or storage.
- IntegrityVerificationService: Ensures stored or transmitted data has not been altered.

Implemented By: EncryptionManagerImpl.

M17: Out-of-Scope Handling Module

Secrets:

- Command whitelist/blacklist configuration
- Safety policy file checksum
- Secure incident reporting token

Services:

- CommandValidationService: Validates command scope and prevents unsafe actions.
- UserSafetyService: Provides safe fallback messages and cancel options.
- RecoveryService: Rolls back partially executed or invalid operations.
- IncidentReportingService: Reports anomalies and unsafe commands to the AuditLogger.

Implemented By: OutOfScopeHandlerImpl.

Input Processing Layer Modules

M18: Microphone Manager

Secrets:

- Device-level audio capture configuration and sampling parameters
- Buffering strategy and fallback device selection
- Normalization of raw audio before downstream processing

Services:

- Start and stop microphone stream
- Read audio frames
- Provide normalized PCM buffer
- Expose current device status and surface hardware errors

Implemented By: Browser audio APIs or OS audio interface.

M19: VAD Noise Filter

Secrets:

- Voice-activity-detection thresholds
- Noise-suppression heuristics and smoothing windows
- Adaptive gain rules

Services:

- Filter raw microphone frames
- Determine speech vs. silence
- Emit clean audio frames
- Send VAD events (start/stop speech)

Implemented By: DSP/VAD library.

Personalization Layer

M20: Prompting Module

Secrets:

- Prompt templates and personalization rules
- Condition-based variation (history, context)
- Confidence-based prompt selection logic

Services:

- Generate user prompts
- Provide clarifying follow-ups when intent confidence is low
- Produce culturally inclusive and accessibility-enabled copy

Implemented By: Application prompt engine.

M21: Model Tuner

Secrets:

- Adaptation strategy for user-specific speech patterns
- Tuning weights for model updates
- Online vs. offline update thresholds
- Embedding storage rules

Services:

- Update inference parameters based on user data
- Compute personalized embeddings
- Adjust thresholds for intent classification and STT confidence

Implemented By: ML runtime (Whisper fine-tuning layer or lightweight personalization pipeline).

M22: Instruction Registry

Secrets:

- Internal representation of user-custom commands
- Schema for mapping instructions to intents
- Safety constraints preventing unsafe or ambiguous command creation

Services:

- Store, update, and delete user instructions
- Resolve user-defined commands during intent parsing
- Validate new instructions
- Expose registry lookup API

Implemented By: Local or cloud command registry service.

8 Traceability Matrix

The traceability matrices show how each requirement is implemented by specific modules and how each anticipated change is isolated within a single module.

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

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.

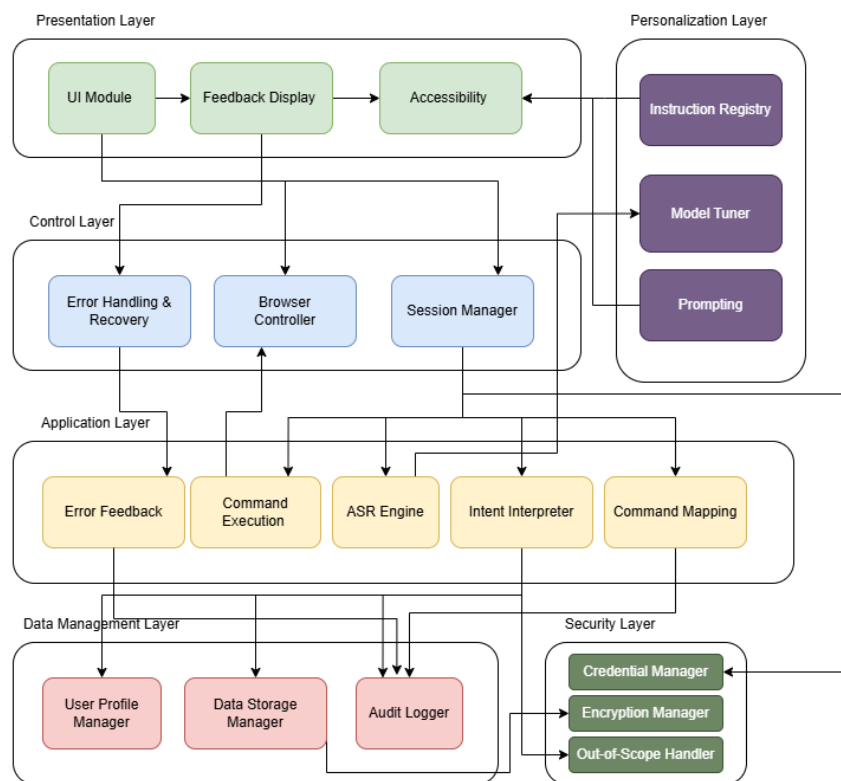


Figure 1: Use hierarchy among modules

10 User Interfaces

Will be defined in later design iterations.

11 Design of Communication Protocols

Not applicable at this time.

12 Timeline

The project is divided into phases with clear objectives, and all tasks are handled collaboratively — there is no single module ownership.

12.1 Development Phases

- **Foundational Development Phase (September – October 2025):** Core functionality such as speech-to-text, intent interpretation, and command mapping was developed collectively.
- **Integration and Testing Phase (November – December 2025):** The team works together to integrate modules, improve accessibility, and optimize performance.

12.2 Task Responsibilities

- Speech recognition and intent interpretation
- Command mapping and system integration
- User interface and accessibility
- Testing and performance optimization
- Documentation and support

All tasks are executed collaboratively, ensuring shared responsibility and smooth progress.

Note: The full timeline is outlined in Section 8 ("Project Decomposition and Scheduling") of the Development Plan document: <https://github.com/speech-buddies/VoiceBridge/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf>

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.