Module Guide for Software Engineering

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

November 13, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

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
[etc. —SS]	[... —SS]

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 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: The specific hardware on which the software is running.

AC2: The format of the initial input data.

...

[Anticipated changes relate to changes that would be made in requirements, design or implementation choices. They are not related to changes that are made at run-time, like the values of parameters. —SS]

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: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

...

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 Adapter) – manages microphone input and audio output across platforms.

M2: HH-Auto (System Control) – 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 Executor) – selects MCP tools and executes user intents.

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

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

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

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

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

M11: SD-ToolRegistry (Action Map) – maps intents to MCP tools/actions.

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

M13: SD-STT/TTS Config – defines Whisper + TTS model configuration and language settings.

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.

[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]

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)

7.1.1 HH-IO: Audio Adapter (M1)

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

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

Implemented By: OpenAI Whisper

Type Of Module: Library

7.1.2 HH-Auto: System Control (M2)

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

Services: Move Cursor, click, type, open and close applications.

Implemented By: PyAutoGUI

Type Of Module: Library

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

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

Secrets: Speech To Text model configuration, text normalization rules and buffering strategies.

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

Implemented By: Project Proxi

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: Coverts normalized text into intents with metadata fields.

Implemented By: Project Proxi

7.2.3 BH-Plan: Task Executor (M5)

Secrets: Decision-making logic for selecting correct MCP agent based on intent. Execution plan and task table. Error handlers and request handlers.

Services: Create an execution plan based on intent. Execute plan using MCP agent and track task to completion.

Implemented By: Project Proxi

7.2.4 BH-Safety: Confirmation Gate (M6)

Secrets: Risk Analysis PolicyTable, Irreversible Detection algorithm, timeout actions.

Services: Implements accidental command protections. Classifies action risks, decides appropriate policy and prompts user for confirmation when action is deemed high risk.

Implemented By: Project Proxi

7.2.5 BH-Session: Context Manager (M7)

Secrets: Session History Table, short-term conversational context for AI model, previous action tree.

Services: Tracks all previous sessions and previous actions completed for each session. Maintains continuity across user requests.

Implemented By: Project Proxi

7.2.6 BBH-Feedback: Response Manager (M8)

Secrets: Output action logic, Text-to-speech configuration, message delivery monitor.

Services: Manages applications to open and actions to do based on user input. Maintains messages and prompts to be shown to the user. Converts messages to speech and outputs through speakers if required.

Implemented By: Project Proxi

7.2.7 BH-UI: Proxi Interface (M9)

Secrets: View Groups, Information Presentation Rules and prompt timing rules.

Services: Updates UI State, displays messages, presents user prompts and show voice feedback prompts.

Implemented By: Project Proxi

7.3 Software Decision Module (M10)-(M14)

7.3.1 SD-Types: Core Structures (M10)

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

Services: Defines shared Classes and Objects used system-wide.

Implemented By: Project Proxi

7.3.2 SD-ToolRegistry: Action Map (M11)

Secrets: Maps to link action intents to MCP tools, MCP Agents and system automation routines.

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

Implemented By: Project Proxi

7.3.3 SD-Store: Local Storage (M12)

Secrets: User Settings, Accessibility Settings, Session History, Action Tree, AI model context.

Services: Saves and loads user settings and previously saved states from memory.

Implemented By: Project Proxi

7.3.4 SD-STT/TTS Config (M13)

Secrets: Stores API keys, configurations and fallback options for OpenAI Whisper model.

Services: Passes Whisper/TTS parameters to BH-Input and parses API output to BH-Feedback.

Implemented By: Project Proxi

7.3.5 SD-Log: Event Logger (M14)

Secrets: Diagnostic log formatter, parser and storage algorithm.

Services: Stores and sends diagnostic logs in the event of failure.

Implemented By: Project Proxi

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements, and between the modules and the anticipated changes. Functional requirements are referenced using their identifiers from the SRS (e.g. FUNC.R.1–FUNC.R.9).

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 1: Traceability between SRS functional requirements and modules

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

Anticipated Change	Modules
AC1	M??
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 2: Traceability between anticipated changes 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.

[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

10 User Interfaces

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

11 Design of Communication Protocols

[If appropriate —SS]

12 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.