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
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

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

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 (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: The communication pattern between modules through the MCP agent interface.

UC4: The fundamental decomposition into Hardware-Hiding, Behaviour-Hiding, and Software-Decision modules.

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.

- **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-STT/TTS Config** ensuring proper speech configuration.

- **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 data 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-STT/TTS Config** 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. The 200ms loading requirement is met through efficient registry design and minimal initialization overhead.

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 three traceability matrices: between the modules and the functional requirements plus nonfunctional requirements, and between the modules and the anticipated changes.

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 functional 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 2: Traceability between nonfunctional requirements and modules

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

Table 3: Traceability between nonfunctional requirements and modules continued

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

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

Table 4: 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.

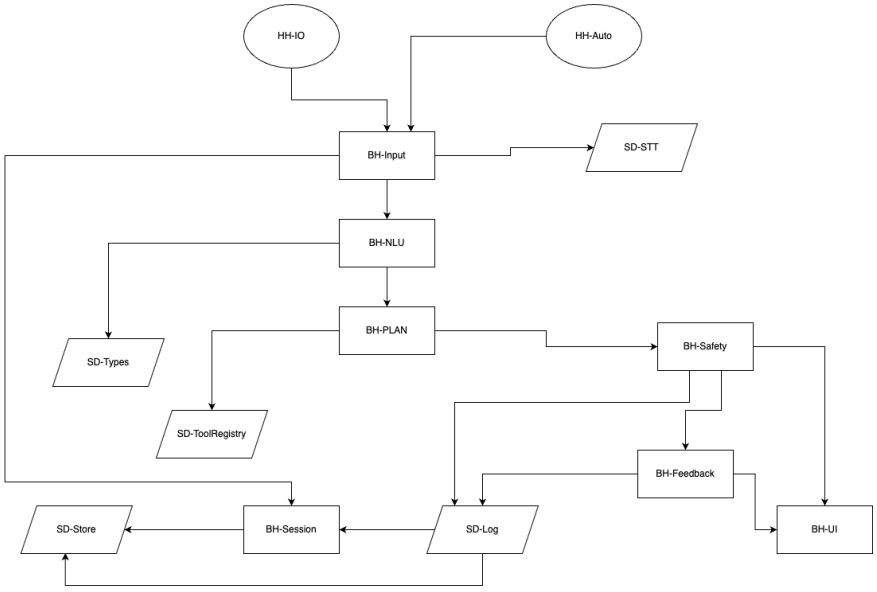

Figure 1: Use hierarchy among modules

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.

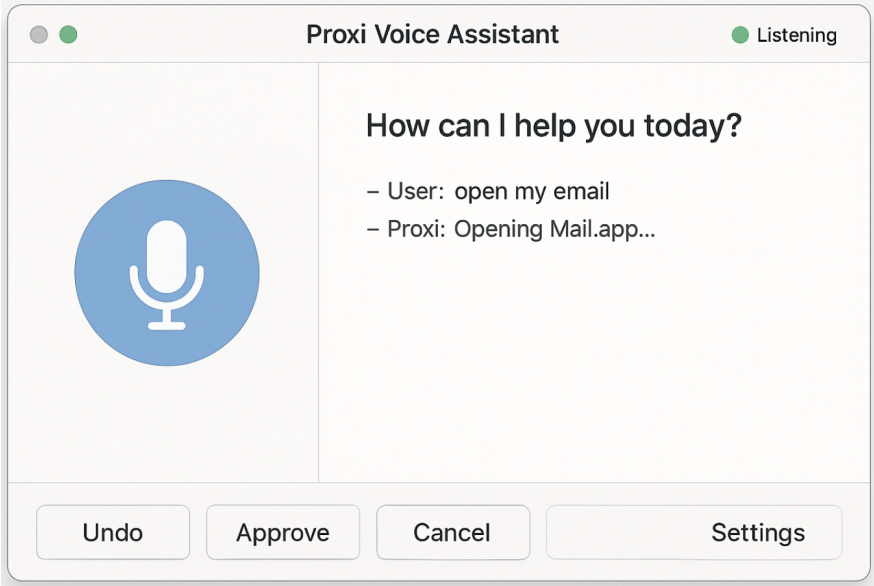

Figure 2: Mockup of the main Proxi interface

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**: M3 and M8 call external AI services over standard HTTPS with JSON payloads. Provider details are in M13.
- **MCP tools and automation**: M5 uses MCP tools and desktop automation using the Model Context Protocol, which exchanges JSON requests and responses through a local and reliable channel. Tool mapping is maintained by M11.

12 Timeline

GitHub Projects is used for project management. Tasks, milestones and assignee information can be found in the Kanban Board.

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.