Module Guide for Software Engineering

Team #22, TeleHealth Insights
Mitchell Weingust
Parisha Nizam
Promish Kandel
Jasmine Sun-Hu

January 15, 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 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Clinician GUI Module

M2: Parent GUI Module

M3: App Controller Module

M4: API Gateway Module

M5: Authentication Module

M6: Result Storage Module

M7: Media Processing Module

M8: Logging Module

M9: Question Bank Module

M10: Real-Time Feedback Module

M11: Report Generation Module

M12: Video Processing Module

M13: Audio Processing Module

M14: English Question Bank Module

M15: Mandarin Question Bank Module

M16: Matching Question Bank Module

M17: Repetition Question Bank Module

Level 1	**Level 2**
Hardware-Hiding	N/A
Behaviour-Hiding	Clinician GUI Parent GUI Authentication Module Result Storage Module Real-Time Feedback Module Report Generation Module Media Processing Module Video Processing Module Audio Processing Module Logging Module Question Bank Module Mandarian Question Bank English Question Bank Repetition Question Bank Module Matching Question Bacnk Module
Software Decision	APP Controller API Gateway

Table 1: Module Hierarchy

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 2.

[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 (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

7.2 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: –

7.2.1 Input Format Module (M??)

Secrets: The data pipeline and algorithms used to process live media streams and generate immediate feedback.

Services: Provides real-time analysis of audio and video streams during an assessment session. Flags issues or disturbances and delivers immediate feedback to guide session adjustments.

Implemented By: RealTimeFeedbackService

Type of Module: Library

Secrets: The methods and algorithms used to process video data, including frame extraction, format handling, and metadata processing.

Services: Handles all video-related data processing tasks, such as analyzing video frames, ensuring quality, and extracting relevant details. This module communicates with the Media Processing Module.

Implemented By: Media-processing-service

Type of Module: Abstract Object

7.2.2 Report Generation Module (M11)

Secrets: The methods and templates used to compile data into user-friendly reports summarizing session results and flagged disturbances.

Services: Generates comprehensive session reports for review by parents and clinicians. Summarizes processed media results, flagged disturbances, and overall session feedback.

Implemented By: ReportGenerationService

Type of Module: Library

7.3 Software Decision Module

7.3.1 Audio Processing Module (M??)

Secrets: The methods and algorithms used to process audio data, such as format conversions, noise filtering, and speech analysis.

Services: Handles all audio-related data processing tasks, including speech detection, sound quality analysis, and extracting key audio features. This module communicates with the Media Processing Module.

Implemented By: Media-processing-service

Type of Module: Abstract Object

7.3.2 Logging Module (M8)

Secrets: The design and implementation of log storage and retrieval mechanisms, as well as the structure of the logs (e.g., event logs, error logs, debug logs).

Services: Tracks system activity and errors through event logging. Supports audit logging for user actions and system changes. Provides interfaces to query and clear logs for maintenance.

Implemented By: logging-monitoring-service

Type of Module: List of Records

7.4 Software Decision Module

8 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-A1	M1, M2, M3, M5, M8
FR-A2	M2, M3, M5, M8
FR-A3	M1,M3, M5, M8
FR-A4	M5, M8
FR-A5	M5, M8
FR-SS1	M2, M3
FR-SS2	M2, M3
FR-SS3	M2, M3
FR-SS4	M2, M3
FR-SS5	M2, M3, M9
FR-AI1	M2, M3, M7, M9, M12, M13, M14, M15, M16
FR-AI2	M2, M3, M4, M7, M9, M12, M13, M14, M15, M16
FR-AI3	M2, M3, M4, M7, M9, M12, M13, M14, M15, M16
FR-AI4	M2, M3, M4, M7, M9, M12, M13, M14, M15, M16
FR-AI5	M2, M3, M4, M7, M9, M12, M13, M14, M15, M16
FR-AI6	M2, M3, M4, M6, M7, M9 M12, M13, M14, M15, M16
FR-AI7	M2, M3, M4, M7, M9 M12, M13, M14, M15, M16, M17
FR-DSC1	M4, M6, M8,
FR-DSC2	M4, M6, M7, M10, M12, M13
FR-DSC3	M4, M5
FR-DSC4	M4, M5
FR-DSC5	M4, M8, M10, M11
FR-VADA1	M4, M7, M12, M13
FR-VADA2	M4, M7,M8 M12, M13
FR-VADA3	M4, M7, M8, M12, M13
FR-DPD1	M4, M6, M8, M10, M11
FR-DPD2	M4, M6, M8, M10, M11
FR-DPD3	M1, M4, M6, M8, M10, M11
FR-DPD4	M1, M4, M6, M8, M10, M11

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M9, M17
AC2	M9, M16, M17

Table 3: Trace 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]

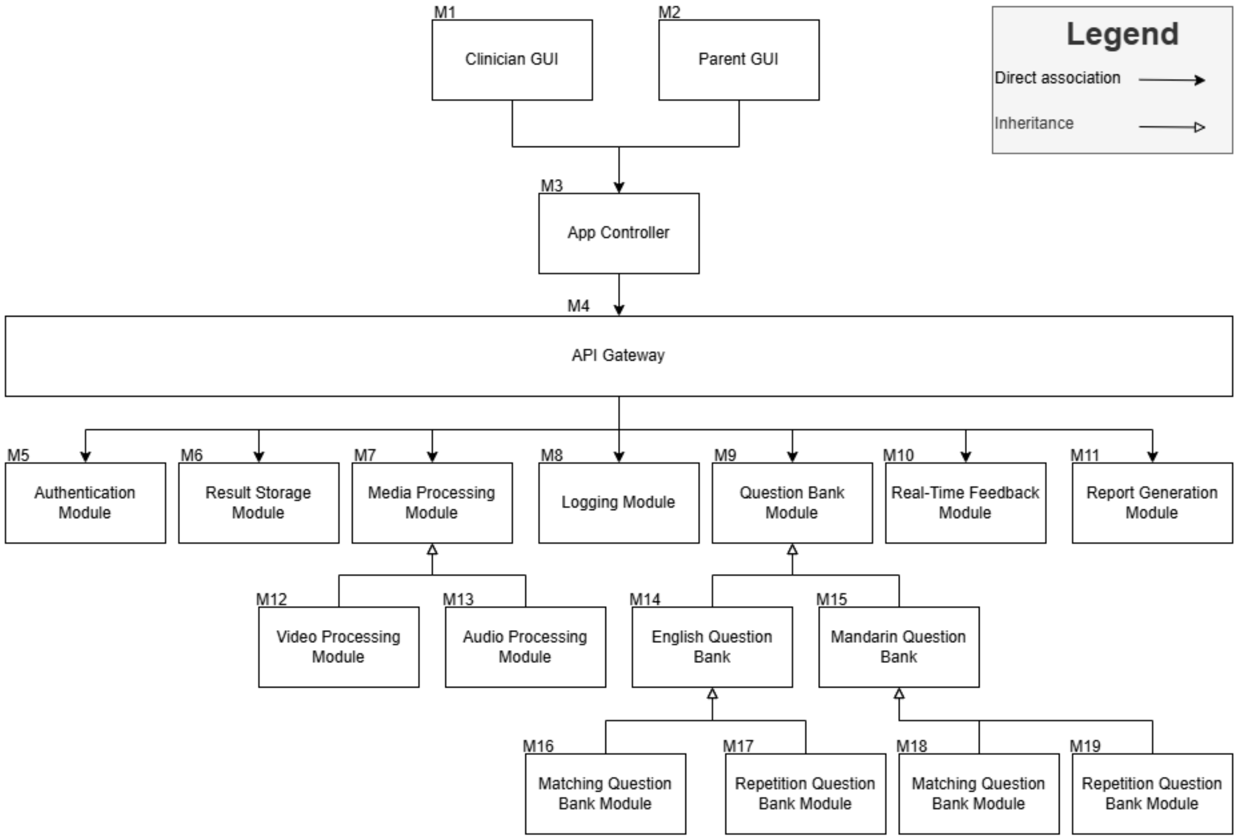

Figure 1: Use hierarchy among modules

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.