Module Guide for FitCoachAR

Seyedmohamad Mirhosseininejad

April 21, 2026

1 Revision History

Date	Version	Notes
2026-03-23	1.0	Initial MG/MIS presentation draft
2026-04-02	1.1	Formalized 8-module hierarchy
2026-04-07	1.2	Reviewer's edit (Yibing Mei and Dr. Smith)
2026-04-15	1.3	Final-doc consistency pass: added Hardware/Behaviour-Hiding subsection headers in Section 6; corrected module-to-requirement traceability (Table 2) for R1–R5
2026-04-15	1.4	Final Documentation release for CAS 741
2026-04-20	1.5	Moved M8 (Signal Smoothing) from Behaviour-Hiding to Software Decision; Kalman-filter algorithm is not in the SRS (AC4).
2026-04-20	1.6	Aligned §6 prose and Table 2 with M8's Software-Decision classification (removed M8 from the R2 trace; M8 traces to AC4 only).

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
FitCoachAR	Explanation of program name
UC	Unlikely Change

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 (CPU/GPU vs. NPU).

AC2: The specific machine learning model used for pose estimation (MediaPipe, MoveNet, HRNet).

AC3: The logic and thresholds for different exercise types (Squats, Bicep Curls).

AC4: The algorithm used for signal noise reduction (Kalman Filter vs. Savitzky-Golay).

AC5: The format of the initial video data (Base64 vs. Binary Buffers).

AC6: The frequency and refresh rate of the real-time feedback displayed to the user.

AC7: The visual styling and branding of the user interface (colors, fonts, badge icons).

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 mathematical basis for biomechanical analysis (dot-product joint angle computation) is assumed to be a fixed, physically grounded approach that will not change.

UC2: The number of tracked body landmarks (33 MediaPipe Pose landmarks) is treated as fixed.

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: Video Input Module

M2: Display Output Module

M3: Video Formatting Module

M4: Pose Tracking Module

M5: Kinematic Engine Module

M6: Exercise State Module

M7: UI Rendering Module

M8: Signal Smoothing Module

Level 1	**Level 2**
Hardware-Hiding	M1
Behaviour-Hiding	M2 M3 M4 M5 M6 M7
Software Decision	M8

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.

Figure 1: System Architecture Overview

Most modules' design decisions trace directly to an SRS requirement; those modules are classified as Behaviour-Hiding. For example, the decision to use MediaPipe (M4) was made to satisfy the keypoint-tracking requirement (R2) without requiring specialised hardware such as depth cameras. A few modules hide decisions that are *not* in the SRS — for instance, the choice of smoothing algorithm (M8) is a pure software-level decision (see AC4); M8 therefore traces to an Anticipated Change rather than to a functional requirement, and is classified as Software Decision.

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. *FitCoachAR* means the module will be implemented by the FitCoachAR 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 Module

Secrets: The data structure and algorithm used to access hardware-level video capture.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the behaviour of the system.

Implemented By: Browser/OS

7.1.1 Video Input Module (M1)

Secrets: The data structure and implementation used to capture raw video frames from the webcam.

Services: Provides a stream of image data to the system.

Implemented By: Browser/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 SRS. This module serves as a communication layer between the hardware-hiding module and the software decision module.

Implemented By: FitCoachAR

7.2.1 Display Output Module (M2)

Secrets: The logic for rendering the AR skeleton and badge overlays on the screen.

Services: Displays the processed video with visual feedback to the user.

Implemented By: Browser/three.js APIs

7.2.2 Video Formatting Module (M3)

Secrets: The encoding and decoding algorithms used for network transmission.

Services: Converts raw image frames into transportable formats (e.g., Base64/JPEG).

Implemented By: FitCoachAR

7.2.3 Exercise State Module (M6)

Secrets: The state machine logic and repetition counting thresholds.

Services: Tracks exercise progress and increments repetition count.

Implemented By: FitCoachAR

7.2.4 UI Rendering Module (M7)

Secrets: The JSON schema and communication protocols for real-time feedback.

Services: Provides textual and numerical feedback (rep count, stability) to the user interface.

Implemented By: FitCoachAR

7.2.5 Pose Tracking Module (M4)

Secrets: The specific machine learning models and inferencing logic for landmark extraction.

Services: Extracts 3D/2D coordinates of human joints from video frames.

Implemented By: MediaPipe (Library)

7.2.6 Kinematic Engine Module (M5)

Secrets: The mathematical formulas for joint angle computation and coordinate frame transforms.

Services: Computes biomechanical metrics like joint flexion and limb alignment.

Implemented By: FitCoachAR

7.3 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 structures and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Signal Smoothing Module (M8)

Secrets: The filtering algorithms used to remove noise from landmark coordinates (e.g., Kalman filter vs. Savitzky–Golay). The choice of algorithm is not dictated by the SRS — it is a software-decision-level choice made to help satisfy the NFR1 accuracy bound. This corresponds to anticipated change AC4.

Services: Provides a stable, non-jittery signal for downstream modules.

Implemented By: FitCoachAR

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
R1 (R_Input)	M1, M3, M7
R2 (R_Process)	M4
R3 (R_Calc)	M5
R4 (R_Verify)	M6
R5 (R_Output)	M2, M7

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M4
AC3	M6, M5
AC4	M8
AC5	M3

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. The use relation corresponds to Python `import` statements in the implemented code.

The uses hierarchy is a directed acyclic graph (DAG) as illustrated below: The uses hierarchy is a directed acyclic graph (DAG) as illustrated in Figure 2:

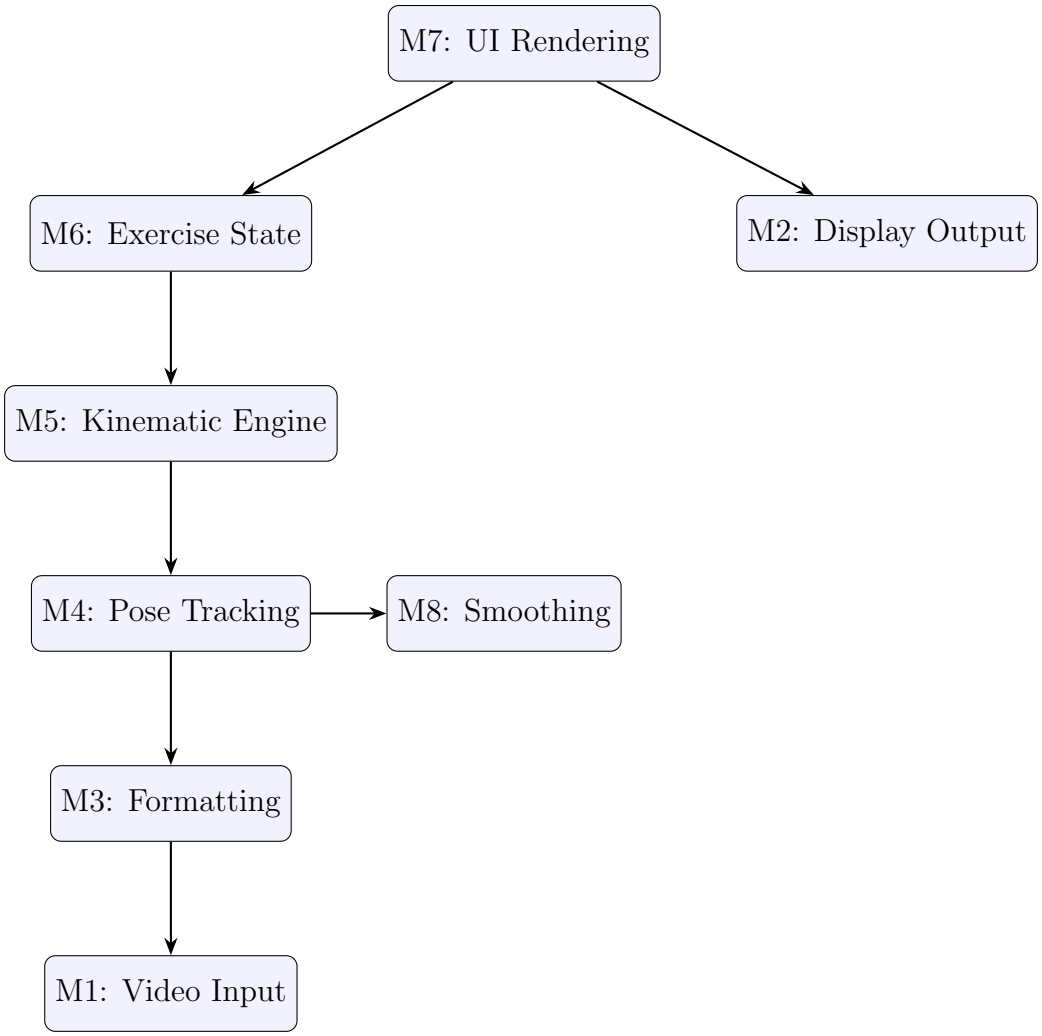

Figure 2: Uses Hierarchy (DAG)

- M7 (UI Rendering) *uses* M6 (Exercise State), M2 (Display Output)
- M6 (Exercise State) *uses* M5 (Kinematic Engine)
- M5 (Kinematic Engine) *uses* M4 (Pose Tracking)
- M4 (Pose Tracking) *uses* M8 (Signal Smoothing), M3 (Video Formatting)
- M3 (Video Formatting) *uses* M1 (Video Input)

10 User Interfaces

FitCoachAR provides two user-facing interfaces:

- **Webcam Feed with AR Overlay (M2: Display Output)**: The primary interface is a live video feed rendered in the browser. The system draws a skeleton overlay on the user's body using detected landmarks, and displays real-time feedback badges showing the current repetition count and form quality scores (e.g., knee stability index, torso lean angle).
- **Exercise Control Panel (M7: UI Rendering)**: A sidebar panel allows the user to select an exercise type (Squat or Bicep Curl), start and stop a session, and view a summary of completed repetitions. The panel updates in real-time via a WebSocket connection to the backend.

11 Database Design

FitCoachAR does not use a persistent database in the current implementation. All session state (rep count, current phase, smoothed angles) is maintained in memory on the backend for the duration of a session and discarded afterward. This is an intentional design decision to simplify the system and avoid privacy concerns with storing biometric workout data.

12 Timeline

Implementation was completed as part of the CAS 741 course project. The development schedule and task assignments are tracked via GitHub Issues and milestones at https://github.com/mmdmirh/cas741.

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.