Module Guide for Software Engineering

Team 21, Visionaries
Angela Zeng
Ann Shi
Ibrahim Sahi
Manan Sharma
Stanley Chen

January 19, 2026

1 Revision History

Date	Version	Notes
Nov. 13, 2025	1.0	First revision of MG Document
Jan. 19, 2026	1.1	Split privacy/infrastructure into single-secret modules; added uses DAG

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
FR	Functional Requirement
NFR	Non-Functional Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change
ET	Eye-tracking
CI/CD	Continuous Integration / Continuous Delivery
API	Application Programming Interface
RBAC	Role-based Access Control
POC	Proof-of-Concept

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 design and presentation of output data may evolve as we introduce a dashboard for post-session and real-time analytics. Visualizations and their underlying implementations are expected to change based on usability feedback and new instructor needs.

AC2: The current homography module is anticipated to be updated with lighter-weight or more efficient algorithms. These changes aim to improve alignment accuracy and reduce latency for real-time gaze projection.

AC3: Additional analytics such as engagement metrics or anomaly detection may be added as part of the dashboard.

AC4: Privacy, access-control, storage/retention, and monitoring policies may evolve (or be introduced) as we harden the system for classroom deployment and research compliance (e.g., redaction rules, RBAC roles, retention windows, audit logging).

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 existing system relies on fixed hardware (Pupil Labs Neon Eye Tracking Glasses, Android smartphone, network webcam), and changing these devices would require extensive redesign.

UC2: The pipeline depends heavily on Pupil Labs' APIs for data access and synchronization. Modifying or replacing these APIs would impact multiple system components, making such changes unlikely.

UC3: The system assumes specific input data formats. Altering these formats would require large-scale changes to parsing and processing modules.

UC4: The existing CLI supports essential homography, visualization, and analysis functions. Its role in offline processing makes major changes unlikely.

UC5: The existing GlassesRecord module handles multi-glasses recording and synchronization through a custom terminal interface will most likely not be changed.

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: Hardware-Hiding Module

M2: Data Ingestion Module

M3: Real-Time Streaming Module

M4: Dashboard Visualization Module

M5: Reporting Module

M6: Data Preprocessing Module

M7: Privacy Filtering Module

M8: Access Control Module

M9: Secure Storage & Retention Module

M10: Observability Module

M11: Engagement Analytics Module

M12: Correlation & Visual Analysis Module

Level 1	Level 2
Hardware-Hiding Module	Hardware-Hiding Module (M1)
Behaviour-Hiding Module	Data Ingestion Module (M2) Real-Time Streaming Module (M3) Dashboard Visualization Module (M4) Reporting Module (M5)
Software Decision Module	Data Preprocessing Module (M6) Privacy Filtering Module (M7) Access Control Module (M8) Secure Storage & Retention Module (M9) Observability Module (M10) Engagement Analytics Module (M11) Correlation & Visual Analysis Module (M12)

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.

- **Separation of data ingestion from preprocessing:** Requirements relating to multi-device input, raw data capture, timing synchronization, and recovery from connection loss motivated the creation of a dedicated *Data Ingestion Module*. Requirements concerning noise reduction, calibration correction, homography projection, and coordinate normalization are distinct and subject to change independently, motivating the *Data Preprocessing Module*. This reflects the SRS separation between raw data capture and analytic data preparation.

- **Explicit handling of real-time constraints:** The need for live metrics, dashboard updating, latency guarantees, and real-time streaming (as described in the SRS operational modes) led to the creation of a separate *Real-Time Streaming Module*. This encapsulates decisions about streaming frameworks, buffering policy, and delivery latency, which are likely to evolve as performance constraints change.

- **Distinguishing analytics computation from visualization:** Engagement metrics, instructor–student gaze alignment, heatmap generation, and correlation analyses are internal computations not directly described to the end user. These behaviours motivated two separate software-decision modules: the *Engagement Ana-*

lytics Module and the *Correlation & Visual Analysis Module*. Meanwhile, all visible UI behaviours—including real-time views, playback, filtering, and seat-map interactions—are grouped in the *Dashboard Visualization Module*. This follows the SRS distinction between analytic computation and user-facing presentation.

- **Privacy as a first-class design concern (split into independent secrets):** The SRS includes privacy/anonymization, RBAC access control, secure storage/retention, and logging/monitoring requirements. These concerns change independently and should not be embedded inside ingestion, analytics, or dashboard logic without creating strong coupling. Therefore, the former monolithic "Privacy & Infrastructure" responsibility is decomposed into:
  - *Privacy Filtering Module* for redaction/anonymization rules,
  - *Access Control Module* for RBAC/authz policy,
  - *Secure Storage & Retention Module* for storage schema/retention strategy,
  - *Observability Module* for audit logging/monitoring configuration.
- **Support for offline summaries and instructor reporting:** Requirements calling for post-session summaries, exportable metrics, and instructor-friendly performance reports lead to the *Reporting Module*. This module encapsulates decisions about report structure, export format, and summarization logic—decisions that are likely to change independently of both preprocessing and dashboard design.
- **Abstraction of physical devices and OS services:** The SRS assumes multiple hardware sources (eye-tracking glasses, central camera, audio, file storage, and network interfaces). To prevent downstream modules from depending on specific device drivers or OS-level behaviour, a general *Hardware-Hiding Module* is included.
- **Backend service and session-lifecycle requirements:** The SRS specifies requirements for creating, managing, and retrieving recorded sessions, as well as backend API access for live or historical data. These behaviours are encapsulated in the *Real-Time Streaming Module*, which provides client subscription and session-control interfaces.

Overall, the module structure shows the boundaries between behaviours visible to the user, performance-driven internal computations, privacy-critical infrastructure, and hardware-specific interactions. This makes sure that changes in requirements - such as improved analytics, new privacy constraints, or new hardware - affect only the module that hides the corresponding 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. *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 Module (M1)

Secrets: The device-specific interfaces, drivers, and low-level OS mechanisms used to access eye-tracking glasses (Neon), world camera input, microphones, GPU acceleration, file system APIs, local and cloud storage backends, network sockets, and system time synchronization (e.g., NTP services).

Services: Provides a virtualized hardware interface for capturing gaze streams, world camera frames, audio signals, timestamps, and device metadata, and for reading/writing data to persistent storage. Exposes uniform APIs so downstream modules do not depend on specific device models, storage engines, or operating systems.

Implemented By: OS, vendor SDKs, device drivers

Type of Module: Library

7.2 Behaviour-Hiding Modules

Secrets: The externally visible behaviours of the system, including session playback, dashboard display logic, ingestion behaviour, and reporting rules as defined in the SRS.

Services: Provides all user-facing behaviours, connecting virtualized hardware to internal algorithms. These modules must change if the SRS behaviour changes.

Implemented By: –

7.2.1 Data Ingestion Module (M2)

Secrets: The method of connecting to and receiving data from multiple hardware sources (glasses, cameras, microphones), including device protocols, buffering policy, timestamp alignment strategy (including use of system/NTP time), and fault-recovery behaviour for dropped connections or device failures.

Services:

- Accepts raw gaze, video, audio, and metadata streams from all devices.
- Performs initial timestamping and session indexing using a common time base.

- Manages session start/stop hooks that allow backend services to create sessions, register participants, and mark completed recordings.
- Outputs unified raw data packets for preprocessing and optional storage.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.2 Real-Time Streaming Module (M3)

Secrets: The design decisions for real-time transport, encrypted communication, latency buffering, session lifecycle APIs, and authenticated endpoint access.

Services:
- Exposes programmatic APIs for session creation, control, and analytics retrieval.
- Streams live and replay data using encrypted protocols.
- Enforces RBAC-protected access to all streaming and analytics endpoints.
- Maintains session state and reports health metrics to observability infrastructure.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.3 Dashboard Visualization Module (M4)

Secrets: The visualization grammar, UI layout, interaction rules, role-based presentation logic, and display of system status indicators.

Services:
- Presents real-time and historical analytics.
- Renders heatmaps, fixation charts, engagement timelines, and gaze-alignment plots.
- Displays system status indicators (latency, stream health, sync drift).
- Supports role-based views (e.g., instructor vs. researcher).
- Provides playback and filtering over recorded sessions.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.4 Reporting Module (M5)

Secrets: The structure and formatting of summaries, metrics, annotations, and export formats (CSV, JSON, PNG) and the rules for instructor reflection and research reporting.

Services:
- Generates post-session summaries and aggregated attention metrics.
- Exports analytics and visualizations in CSV, JSON, and PNG formats.
- Supports annotation, note-taking, and timeline cropping for instructional review.

Implemented By: Software Engineering

Type of Module: Library

7.3 Software Decision Modules

Secrets: Internal algorithms, mathematical models, data structures, and analytic methods that are not externally visible. These change primarily due to performance or research-driven improvements, not due to changes in behaviour.

Services: Provides internal data processing, analytics, correlations, and privacy enforcement services for behaviour-hiding modules.

Implemented By: –

7.3.1 Data Preprocessing Module (M6)

Secrets: Filtering algorithms (smoothing, denoising), calibration transformations, gaze-to-screen homography methods, fixation detection rules, and temporal alignment techniques.

Services:
- Converts raw gaze vectors into screen/world-space coordinates.
- Denoises and filters gaze data.
- Identifies fixations, saccades, attention windows.
- Standardizes multimodal streams for analytics.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

7.3.2 Privacy Filtering Module (M7)

Secrets: The anonymization and redaction policy applied to video, gaze data, stored artifacts, and exported outputs.

Services:
- Applies redaction and masking rules to raw and derived data.

- Ensures masked regions are excluded prior to storage and export.
- Removes or obfuscates identifiers from analytics and reports.

Implemented By: Software Engineering

Type of Module: Library

7.3.3 Access Control Module (M8)

Secrets: Authentication strategy and RBAC policy for dashboards, analytics, streaming APIs, and data exports.

Services:
- Authenticates users and services.
- Authorizes access to dashboards, analytics APIs, and exports based on role.

Implemented By: Software Engineering

Type of Module: Library

7.3.4 Secure Storage & Retention Module (M9)

Secrets: Storage backend selection, encryption-at-rest strategy, session metadata schema, retention windows, and traceability linkage rules.

Services:
- Stores raw, anonymized, and processed session artifacts.
- Stores session metadata (session ID, timestamps, participant mappings).
- Maintains traceability links between raw data, analytics, and reports.
- Enforces retention and deletion policies.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

7.3.5 Observability Module (M10)

Secrets: Audit logging policy and performance metric definitions.

Services:
- Records audit logs for access and export actions.
- Collects runtime metrics (latency, dropped frames, sync drift).
- Exposes metrics used by dashboard status indicators.

Implemented By: Software Engineering

Type of Module: Library

7.3.6 Engagement Analytics Module (M11)

Secrets: The mathematical models used to compute engagement metrics, moving averages, attention peaks/troughs, activity-phase detection, and time-on-task calculations.

Services:
- Computes engagement over time for individuals and aggregates.
- Detects disengagement periods and high-attention intervals.
- Classifies lecture activities (discussion, slides, group work).
- Generates metrics for reporting and dashboard visualization.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

7.3.7 Correlation & Visual Analysis Module (M12)

Secrets: The gaze-correlation model, instructor–student alignment formulas, visual material detection (text block, image region), and methods for transforming heatmaps into interpretable coordinate-space analytics.

Services:
- Computes correlations between instructor gaze and class gaze.
- Maps class attention onto slide regions or world-camera space.
- Identifies which visual elements students focus on, and how this changes over time.
- Supports seat-map and spatial-zone based metrics and filters.

Implemented By: Software Engineering

Type of Module: Abstract Data Type

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-10	M11, M12, M5, M4
FR-11	M9
FR-12	M2, M9
FR-13	M3, M8
FR-14	M8
FR-17	M4, M8
FR-18	M10
FR-21	M7, M3, M9
FR-22	M9, M5
FR-26	M4, M10, M3
FR-27	M7, M9, M5
NFR-9	M3, M9
NFR-10	M8, M3, M4
NFR-11	M7, M9

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M4, M5, M6, M11, M12
AC2	M3, M6
AC3	M11, M12, M4, M5
AC4	M7, M8, M9, M10

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 summarizes the use relation between the modules as a directed acyclic graph (DAG). The edge list below is a readable restatement of the same relations.

At a high level, the use hierarchy for Software Engineering is organized into layers:

- **Level 0 – Hardware layer**: Hardware-Hiding Module (M1) encapsulates device/OS access. No other module is used by M1.
- **Level 1 – Data acquisition**: Data Ingestion Module (M2) uses M1 to collect raw gaze and video streams.
- **Level 2 – Preparation**: Data Preprocessing Module (M6) uses M2 to filter, normalize, and structure captured data.
- **Level 2 – Cross-cutting governance**: Privacy Filtering (M7) uses M2 and M6 to enforce redaction rules. Access Control (M8) provides authorization checks to protect API and session actions. Secure Storage & Retention (M9) uses M2, M6, and M7 to store and retrieve privacy-compliant artifacts and enforce retention policies. Observability (M10) collects logs and runtime metrics from other modules.
- **Level 3 – Analytics and streaming**: Real-Time Streaming (M3) uses M2, M6, and governance modules to serve compliant streams. Engagement Analytics (M11) uses M6. Correlation & Visual Analysis (M12) uses M6 and M11.
- **Level 4 – User-facing visualization**: Dashboard Visualization (M4) uses M3, M11, M12, M8, and M10.
- **Level 5 – Reporting and exports**: Reporting (M5) uses M4, M11, M12, M9, and M8.

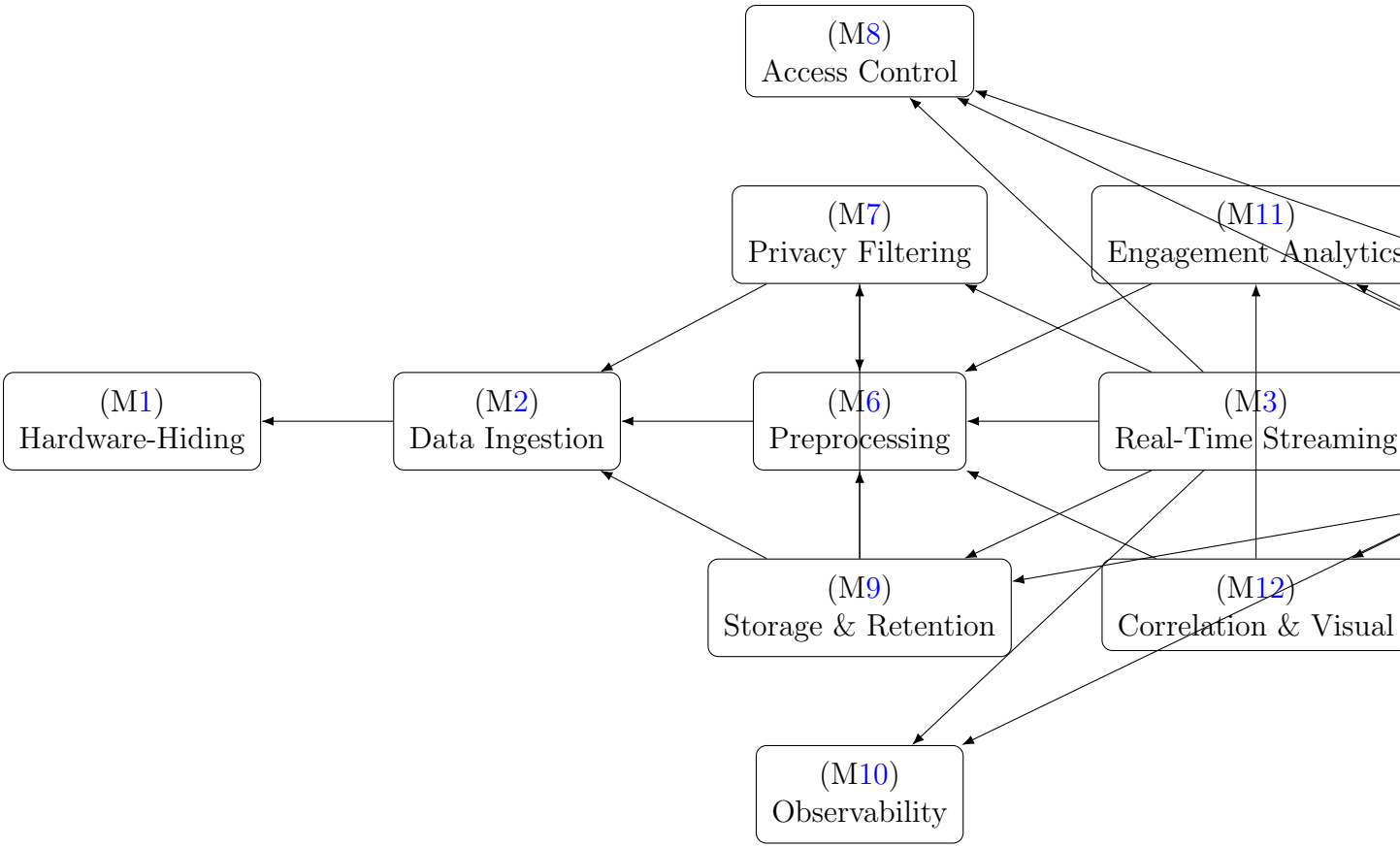

Figure 1: Use hierarchy among modules (DAG)

The main uses relations can be summarized as:

- M2 uses M1.
- M6 uses M2.
- M7 uses M2 and M6.
- M9 uses M2, M6, and M7.
- M3 uses M2, M6, M7, M8, M9, and M10.
- M11 uses M6.
- M12 uses M6 and M11.
- M4 uses M3, M8, M10, M11, and M12.
- M5 uses M4, M8, M9, M11, and M12.

10 User Interfaces

This section provides conceptual sketches of the primary user interface screens for Software Engineering. These sketches represent the core interactions and visual layout of the system. They are intended to illustrate how instructors will navigate through the platform, monitor real-time classroom attention, and review analytics after each session. The sketches are early design mockups and may evolve as the implementation progresses.

10.1 10.1 Student Overview

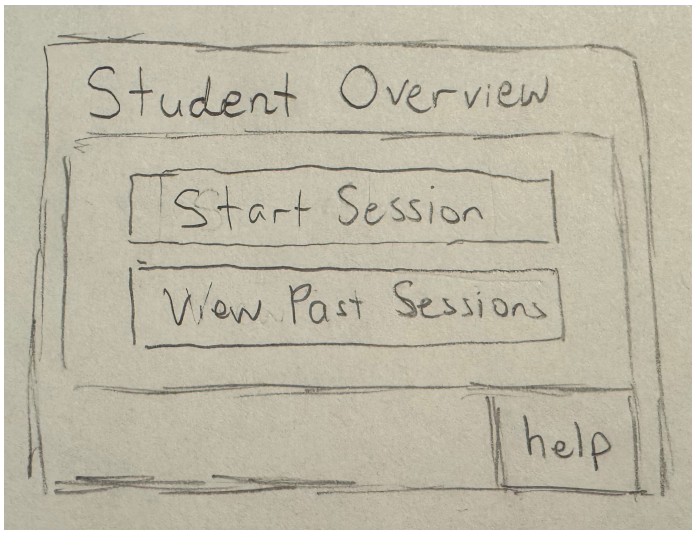

Figure 2: Student Overview Screen

This screen serves as the entry point to the platform. Instructors can begin a new live session or review summaries of previously recorded sessions.

10.2 10.2 Live Classroom Dashboard

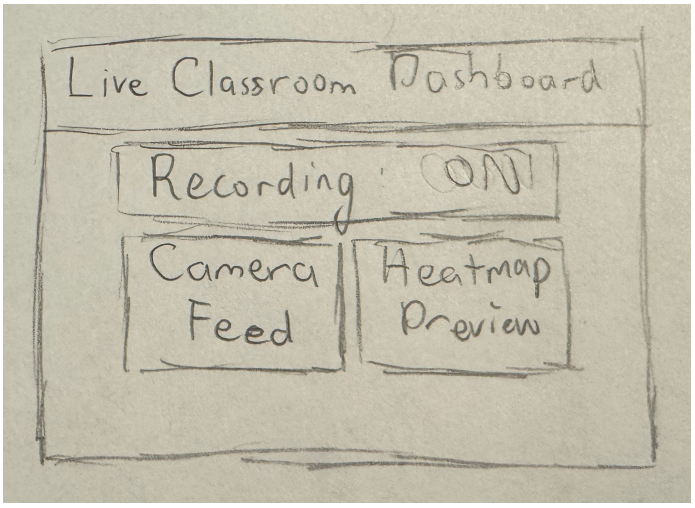

Figure 3: Live Classroom Dashboard

During an active session, instructors are presented with a real-time dashboard. It displays the recording state, a live camera feed from the capture device, and a heatmap preview showing the current distribution of gaze activity.

10.3 10.3 Student Attention Panel

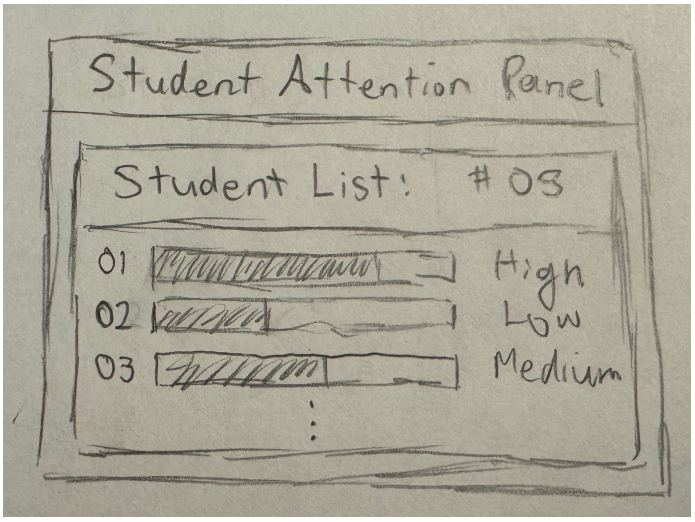

Figure 4: Student Attention Panel

This panel provides a real-time overview of individual student attention levels. Each student is listed with a visual bar indicator, allowing the instructor to quickly identify who is highly

engaged, who is disengaged, and how attention varies across learners.

10.4 10.4 Selected Student Detail View

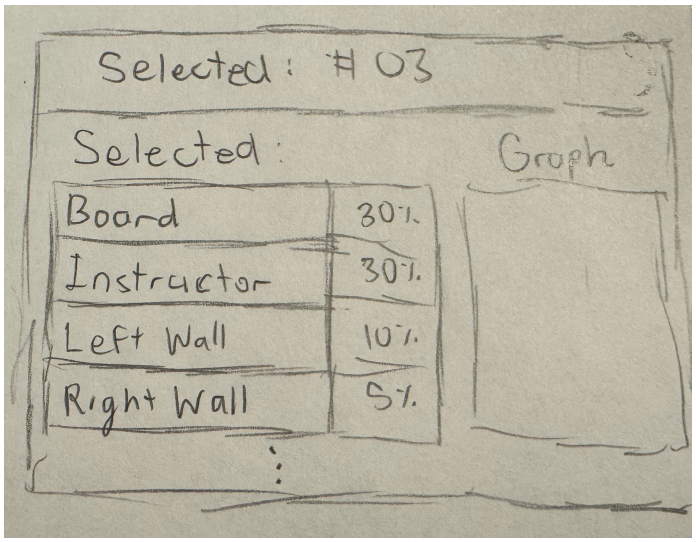

Figure 5: Selected Student Detail View

When a specific student is selected, the instructor is shown a detailed breakdown of that student's gaze distribution across key classroom regions (e.g., board, instructor, walls). A small trend graph summarizes how their engagement changes over time.

10.5 10.5 Region-of-Interest Analytics

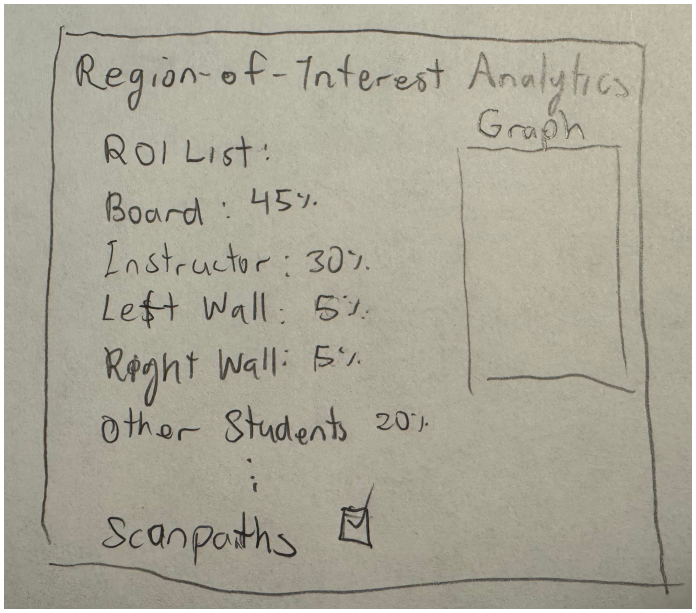

Figure 6: Region-of-Interest (ROI) Analytics

This screen provides a post-session analytical view of where attention was distributed in the classroom. Percentages for each region-of-interest are listed, and an accompanying graph displays temporal patterns. Instructors may also toggle visual elements such as scanpaths.

10.6 10.6 Session Summary

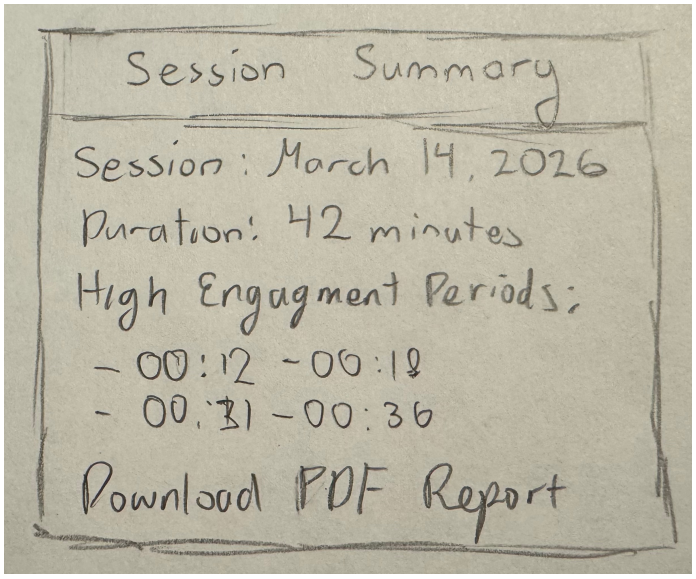

Figure 7: Session Summary Screen

After a session ends, the summary view presents key metrics, including duration, overall engagement, and the periods of highest attention. Instructors may export a report for documentation or further analysis.

10.7 10.7 Settings & Calibration

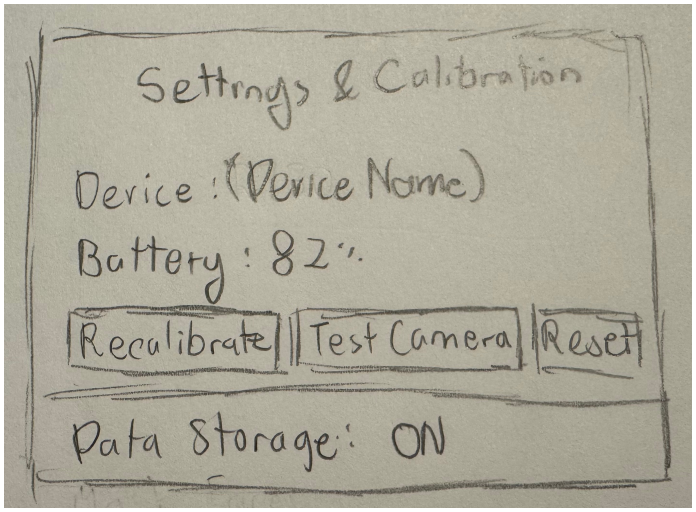

Figure 8: Settings and Calibration Screen

The settings screen allows instructors to calibrate their eye-tracking device, check battery level, test the camera feed, and adjust privacy or data storage preferences. This screen ensures that sessions begin with reliable and well-initialized hardware.

11 Design of Communication Protocols

This section describes the communication protocols used within Software Engineering. Communication occurs between the client-facing dashboard, the real-time streaming backend, the analytics components, and the system's secure storage infrastructure. All communication follows a modular service-based pattern where each module exposes a set of services that may be invoked by higher-level modules as shown in the uses hierarchy.

11.1 11.1 Overall Architecture

System communication is organized around three main channels:

- **Real-Time Data Delivery:** Raw and preprocessed gaze streams and aggregated attention metrics are transmitted from Data Ingestion / Preprocessing through Real-Time Streaming to the Dashboard Visualization module.
- **Asynchronous Analytics Pipeline:** Preprocessed gaze data is passed from the Data Preprocessing module to the Engagement Analytics and Correlation & Visual Analysis modules for metric generation.
- **Secure Storage and Retrieval:** Secure Storage & Retention manages storage of raw, anonymized, and processed artifacts. Privacy Filtering enforces redaction policy prior to exposure. Access Control governs who can request stored artifacts or exports.

All communication follows structured data formats, with components interacting through well-defined access routines specified in the MIS. Communication does not rely on shared global state; instead, modules exchange data through explicit service calls or via the streaming pipeline.

11.2 11.2 Data Flow Patterns

Communication follows two major patterns:

- **Push-Based Real-Time Streaming:** The Real-Time Streaming module pushes compliant (privacy-filtered and authorized) gaze data and attention indicators to the Dashboard Visualization module, enabling instructors to view attention trends with minimal latency.
- **Pull-Based Analysis and Reporting:** The Engagement Analytics, Correlation & Visual Analysis, and Reporting modules request preprocessed or stored data from Preprocessing and Secure Storage & Retention as needed, subject to Access Control checks.

Temporal Characteristics

- Real-time streams operate on short intervals (approximately 1–2 seconds).
- Analytics requests are event-driven rather than continuous.
- Reporting operations occur after class and may be batched.

11.3 11.3 Message Types

Modules exchange data using structured message formats. Typical messages include:

- **GazeFrame:** Raw or preprocessed gaze coordinates, timestamps, pupil size, and detection confidence.
- **AttentionSummary:** Aggregated per-region or per-student attention metrics computed by the Engagement Analytics module.
- **CorrelationMap:** Data structures produced by the Correlation & Visual Analysis module describing gaze distribution heatmaps.
- **SessionRecord:** Metadata describing a class session, stored and retrieved through Secure Storage & Retention.
- **ReportBundle:** Final compiled data provided to the Reporting module for output formatting.

11.4 11.4 Privacy and Security Constraints

All communication adheres to the privacy and security requirements outlined in the Hazard Analysis. Key protections include:

- **Privacy Filtering:** Sensitive regions and identifiers are redacted by Privacy Filtering before exposure to user-facing modules.
- **Access Control:** Only authorized roles may request stored data or exports (enforced by Access Control).
- **Data Minimization:** Real-time streaming transmits only aggregated or anonymized indicators rather than raw identifying visual data.
- **Integrity Checks:** Modules using stored datasets validate checksums or timestamps to avoid stale or corrupted data.

11.5 11.5 Design Considerations

The communication protocol design prioritizes:

- **Low Latency:** Real-time streaming ensures timely updates for instructors during live sessions.
- **Scalability:** Analytics modules operate independently of the real-time pipeline, allowing more complex computations without slowing down the dashboard.
- **Modularity:** Modules communicate through stable abstractions defined in the MIS, reducing interdependencies.
- **Reliability:** Data retrieval and reporting do not depend on the continuous availability of real-time processes, enabling fault tolerance.

This design ensures consistent, secure, and efficient communication between modules in Software Engineering, supporting real-time visualization, post-session analysis, and long-term data storage.

12 Timeline

This section outlines the implementation timeline for Software Engineering. The schedule is organized according to the module hierarchy described in Section 5, with lower-level modules developed first to enable higher-level functionality. The work is divided into phases that reflect the system's data pipeline: ingestion, preprocessing, analytics, and visualization.

Phase 1 — Foundations and Hardware Integration

- **M1: Hardware-Hiding Module** Set up and test Pupil Labs Neon glasses, central camera, and instructor workstation. Confirm data access points and ensure that device drivers and calibration tools function reliably.
- **M2: Data Ingestion Module** Implement raw gaze and video capture, device synchronization, and time-stamped buffering. Establish initial interfaces for downstream modules.

Phase 2 — Data Processing and Governance Foundations

- **M6: Data Preprocessing Module** Develop filtering, noise reduction, coordinate normalization, and outlier detection routines. Produce preprocessed frames for use by analytics modules.
- **M7: Privacy Filtering Module** Implement redaction and masking rules (faces/screens/ROIs), and ensure privacy-filtered artifacts can be produced for downstream consumption.

- **M8: Access Control Module** Implement RBAC checks for session access, exports, and administrative actions.
- **M9: Secure Storage & Retention Module** Implement storage interfaces for raw/anonymized/processed artifacts and define retention/deletion mechanisms.
- **M10: Observability Module** Implement audit logging and basic health/metrics collection for integration testing.

Phase 3 — Real-Time and Analytical Capabilities

- **M3: Real-Time Streaming Module** Construct the pipeline for sending processed, policy-compliant gaze frames and attention signals to the dashboard at 1–2 second intervals.
- **M11: Engagement Analytics Module** Build algorithms for attention scoring, fixation detection, and aggregated engagement metrics.
- **M12: Correlation & Visual Analysis Module** Implement spatial correlation calculations, heatmap generation, and visual analysis tools using preprocessed data and analytics outputs.

Phase 4 — User-Facing Interfaces

- **M4: Dashboard Visualization Module** Integrate real-time streaming, analytics, and correlation outputs into a unified instructor-facing dashboard.
- **M5: Reporting Module** Generate post-session summaries that combine analytics and visualizations. Produce exports or formatted reports based on session data (subject to RBAC and retention policies).

Phase 5 — Integration, Testing, and Refinement

- **System Integration** Combine all modules into an end-to-end pipeline from ingestion to reporting. Ensure consistent data formats and stable communication across modules.
- **Verification and Validation** Conduct unit tests, integration tests, and scenario-based validation following the VnV Plan. Address feedback based on latency, accuracy, and usability tests.
- **Performance and Privacy Review** Verify privacy constraints, retention/deletion behaviour, RBAC enforcement, and real-time performance targets.
- **Final Documentation and Deliverables** Prepare final reports, updated design documents, and demonstration artifacts.

This phased timeline ensures that core data-handling components are completed first, enabling analytics and visualization to be developed on a stable foundation. It also supports iterative testing and refinement as the pipeline becomes fully integrated.

References

BeatLab McMaster. Socialeyes: Scaling mobile eye-tracking to multi-person social settings. https://github.com/beatlab-mcmaster/SocialEyes, 2025. Accessed: 2025-11-13.

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.

Shreshth Saxena, Areez Visram, Neil Lobo, Zahid Mirza, Mehak Khan, Biranugan Pirabaharan, Alexander Nguyen, and Lauren K. Fink. Socialeyes: Scaling mobile eye-tracking to multi-person social settings. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 979-8-4007-1394-1. doi: 10.1145/3706598.3713910. URL https://doi.org/10.1145/3706598.3713910.
(Saxena et al., 2025) (BeatLab McMaster, 2025)