Module Guide for Software Engineering

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

November 13, 2025

1 Revision History

Date	Version	Notes
Nov. 13, 2025	1.0	First revision of MG Document

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: A privacy module may be added to the SocialEyes framework used to censor sensitive information that may be captured in gaze data (e.g: phone screen, laptop screen, human faces) if time permits.

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 & Infrastructure Module

M8: Engagement Analytics Module

M9: Correlation & Visual Analysis Module

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.

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 & Infrastructure Module (M7) Engagement Analytics Module (M8) Correlation & Visual Analysis Module (M9)

Table 1: Module Hierarchy

- **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 Analytics 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 and anonymization as a first-class design concern:** The SRS explicitly lists privacy, anonymization, secure handling of raw video, and compliance with data retention constraints. These requirements cannot be embedded inside ingestion, analytics, or dashboard logic without creating strong coupling. They therefore motivate the independent *Privacy & Infrastructure Module*, which encapsulates the design decisions related to face blurring, spatial masking, secure storage, authorization, and logging.

- **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. This encapsulates the design decision to virtualize camera and sensor access, a key anticipated change identified in the hazards and SRS (e.g., replacing Neon with another device).

- **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 (SRS: Service & API requirements; Session Lifecycle Requirements). These behaviours are not purely real-time streaming nor visualization concerns. This motivated folding backend session-control logic into the *Real-Time Streaming Module*, which now encapsulates API endpoints for session creation, playback control, and client subscriptions.

- **Secure storage, retention, and access-control requirements:** The SRS defines requirements for data retention policies, authenticated access, role-based controls (e.g., instructor vs. researcher), and logging of data access. Embedding these into ingestion or analytics would create tight coupling, so these decisions are isolated in the *Privacy & Infrastructure Module*. This module therefore owns secure storage, retention strategy, authorization, and audit logging as design decisions hidden from the rest of the system.

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 (Kafka, WebSockets), latency buffering, stream synchronization, retry/timeout strategies, throttling, and the external API endpoints through which clients subscribe to streams and control session state (e.g., REST/GraphQL/WebSocket interfaces).

Services:

- Exposes backend APIs for creating, listing, and controlling live or replay sessions (start, pause, stop) used by the dashboard and other clients.
- Delivers live preprocessed data and analytics to the dashboard and other authorized consumers with bounded latency.
- Maintains real-time session state (live/offline toggle, paused state, health status) and reports it to monitoring/logging infrastructure.
- Ensures smooth playback for recorded sessions by streaming stored data through the same interfaces used for live sessions.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.3 Dashboard Visualization Module (M4)

Secrets: The visualization grammar, UI layout, interaction rules, playback controls, and display logic for heatmaps, timelines, seat maps, and instructor–student alignment graphs.

Services:

- Presents real-time and historical analytics.
- Renders heatmaps, fixation charts, engagement over time, and gaze-alignment plots.
- Supports filters (student, seat zone, activity type, time window).
- Provides playback and zoom-in on individual student streams.

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 (PDF/HTML) and the rules for instructor reflection.

Services:

- Generates session summaries, attention-duration metrics, and engagement timelines.
- Produces exportable reports for instructors and teaching portfolios.

- Supports annotation, note-taking, screengrabs, and timeline cropping.

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 & Infrastructure Module (M7)

Secrets: The anonymization pipeline, face-blurring method, device obfuscation, consent rules, access-control and role-based authorization policies, logging and monitoring configuration, database schema and storage layout for raw and processed data, and data retention and deletion strategy.

Services:

- Performs face blurring, device masking, and area-of-interest redaction on video and gaze data before it is exposed to user-facing modules.
- Ensures audio and video privacy filtering before data reaches the dashboard.
- Manages authentication and authorization for backend APIs and the dashboard (e.g., instructor vs. researcher roles).

- Provides secure storage and retrieval of raw and processed data in the underlying database or file system according to defined retention rules.
- Provides logging and monitoring hooks (e.g., audit logs, API access logs, health checks) used by other modules.

Implemented By: Software Engineering

Type of Module: Library

7.3.3 Engagement Analytics Module (M8)

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

7.3.4 Correlation & Visual Analysis Module (M9)

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-3	M1, M2
FR-7	M6
FR-8	M6
FR-9	M8
FR-15	M4, M3
FR-16	M5
FR-21	M7
NFR-1	M6, M4
NFR-3	M1, M2
NFR-11	M7

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M4, M5, M6 M8, M9
AC2	M3, M6
AC3	M6,
AC4	M7

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.

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

- **Level 0 – Hardware layer**: Hardware-Hiding Module (M1) encapsulates the details of the Pupil Labs Neon glasses, central camera, and instructor workstation. No other module is used by M1.
- **Level 1 – Data acquisition**: Data Ingestion Module (M2) uses M1 to collect raw gaze and video streams from the eye-tracking hardware and central camera.
- **Level 2 – Preparation and infrastructure**: Data Preprocessing Module (M6) uses M2 to filter, normalize, and structure the captured data. Privacy & Infrastructure Module (M7) uses M1, M2, and M6 to apply anonymization, access control, and secure storage over the ingested and preprocessed data.
- **Level 3 – Analytics and streaming**: Real-Time Streaming Module (M3) uses M2, M6, and M7 to serve live, privacy- compliant streams to the dashboard. Engagement Analytics Module (M8) uses M6 to compute engagement metrics over preprocessed data. Correlation & Visual Analysis Module (M9) uses M6 and M8 to compute gaze-correlation and spatial visual analytics.
- **Level 4 – User-facing visualization**: Dashboard Visualization Module (M4) uses M3, M7, M8, and M9 to display real-time and historical analytics while respecting privacy constraints.
- **Level 5 – Reporting and exports**: Reporting Module (M5) uses M4, M8, and M9 to generate summaries and exported reports based on the visualizations and analytics.

The main uses relations can be summarized as:

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

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

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 gaze streams and aggregated attention metrics are transmitted from the Data Ingestion and Real-Time Streaming modules 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:** Privacy & Infrastructure manages anonymized storage of both raw and processed data, making it accessible to authorized modules such as Reporting.

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 (M3) pushes processed gaze data and attention indicators to the Dashboard Visualization module (M4). This enables instructors to view attention trends with minimal latency.

- **Pull-Based Analysis and Reporting:** The Engagement Analytics module (M8), Correlation & Visual Analysis module (M9), and Reporting module (M5) request preprocessed or stored data from the Privacy & Infrastructure module (M7) and Data Preprocessing module (M6) as needed to compute summary statistics or generate class reports.

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 the Privacy & Infrastructure module.
- **ReportBundle:** Final compiled data provided to the Reporting module for output formatting.

These message formats allow the system to maintain a separation of concerns: modules only require knowledge of the structure of the data they consume, not how it was generated.

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:

- **Anonymization:** No personal identifiers are stored or transmitted with gaze data.
- **Access Control:** Only authorized modules may request stored data from M7 (Privacy & Infrastructure).

- **Data Minimization:** Real-time streaming transmits only aggregated or anonymized indicators rather than raw identifying visual data.
- **Integrity Checks:** Modules using stored datasets perform checksum or timestamp validation 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 Infrastructure

- **M6: Data Preprocessing Module** Develop filtering, noise reduction, coordinate normalization, and outlier detection routines. Produce preprocessed frames for use by analytics modules.
- **M7: Privacy & Infrastructure Module** Implement anonymization, secure data storage, access control, and infrastructure services (session management, integrity checks).

Phase 3 — Real-Time and Analytical Capabilities

- **M3: Real-Time Streaming Module** Construct the pipeline for sending processed gaze frames and attention signals to the dashboard at 1–2 second intervals.
- **M8: Engagement Analytics Module** Build algorithms for attention scoring, fixation detection, and aggregated engagement metrics.
- **M9: 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. Design displays for class-level and region-level attention indicators.
- **M5: Reporting Module** Generate post-session summaries that combine analytics and visualizations. Produce exports or formatted reports based on session data.

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 that modules meet privacy constraints, maintain real-time performance targets, and operate reliably under varying classroom conditions.

- **Final Documentation and Deliverables** Prepare final reports, updated design documents, and demonstration artifacts for Rev 1 and project completion.

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)