Module Guide for Software Engineering

Team 4, EvENGage
Virochaan Ravichandran Gowri
Omar Al-Asfar
Rayyan Suhail
Ibrahim Quraishi
Mohammad Mahdi Mahboob

November 6, 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 (**?**). We advocate a decomposition based on the principle of information hiding (**?**). 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 **?**, 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 (**?**). 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: Virtual Hardware Interface Module

M2: User Authentication Module

M3: User Authorization Module

M4: Form Template Module

M5: Form Submission Module

M6: Event Management Module

M7: Event Notification Module

M8: Registration Module

M9: Attendance Tracking Module

M10: Survey Management Module

M11: Survey Response Module

M12: Analytics Module

M13: Report Generation Module

M14: Database Access Module

M15: Data Validation Module

M16: Encryption Module

M17: Communication Module

M18: Logging Module

M19: Audit Module

Level 1	Level 2
Hardware-Hiding Module	M1: Virtual Hardware Interface Module
Behaviour-Hiding Modules	M7.2.1: User Authentication Module M7.2.2: User Authorization Module M7.2.3: Form Template Module M7.2.4: Form Submission Module M7.2.5: Event Management Module M7.2.6: Event Notification Module M7.2.7: Registration Module M7.2.8: Attendance Tracking Module M7.2.9: Survey Management Module M7.2.10: Survey Response Module M7.2.11: Analytics Module M7.2.12: Report Generation Module
Software-Decision Modules	M14: Database Access Module M15: Data Validation Module M16: Encryption Module M17: Communication Module M18: Logging Module M19: Audit Module

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 **?**. 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 User Authentication Module (M7.2.1)

Secrets: The internal methods used for verifying user identities (e.g., token generation, password hashing, OAuth, McMaster SSO).
Services: Authenticates users by validating credentials and issuing secure access tokens.
Implemented By: Backend authentication service.

7.2.2 User Authorization Module (M7.2.2)

Secrets: Rules defining access levels and user roles.
Services: Determines permissions for authenticated users, controlling access to admin or

attendee features.
Implemented By: Backend access control middleware.

7.2.3 Form Template Module (M7.2.3)

Secrets: The structure and layout of form templates used in surveys and event creation.
Services: Provides reusable blueprints for constructing event or survey forms.
Implemented By: Frontend form builder and backend template schema.

7.2.4 Form Submission Module (M7.2.4)

Secrets: Validation rules and submission workflows for user input data.
Services: Receives and validates user-submitted forms, then stores them in persistent storage.
Implemented By: Backend form processing API.

7.2.5 Event Management Module (M7.2.5)

Secrets: Event data structures and scheduling rules.
Services: Enables creation, editing, and cancellation of events by administrators.
Implemented By: Backend event management service.

7.2.6 Event Notification Module (M7.2.6)

Secrets: Notification delivery logic and timing rules.
Services: Sends alerts to users regarding new events, updates, or cancellations.
Implemented By: Backend notification handler and third-party messaging APIs.

7.2.7 Registration Module (M7.2.7)

Secrets: Mapping between users, events, and registration states.
Services: Allows users to register, modify, or cancel event participation.
Implemented By: Backend registration handler.

7.2.8 Attendance Tracking Module (M7.2.8)

Secrets: Methods for recording attendance and validating entry codes.
Services: Tracks event attendance and verifies participant access.
Implemented By: Backend attendance subsystem.

7.2.9 Survey Management Module (M7.2.9)

Secrets: Survey metadata and organizational logic.
Services: Creates, stores, and manages surveys linked to events.
Implemented By: Backend survey management service.

7.2.10 Survey Response Module (M7.2.10)

Secrets: Logic for storing and associating survey responses to users and events.
Services: Collects and stores completed survey responses for later analysis.
Implemented By: Frontend survey interface and backend API.

7.2.11 Analytics Module (M7.2.11)

Secrets: Algorithms for aggregating and calculating statistics.
Services: Computes summaries, attendance rates, and survey trends.
Implemented By: Backend analytics processor.

7.2.12 Report Generation Module (M7.2.12)

Secrets: Formatting logic for report generation (PDF, CSV, charts).
Services: Converts analytics data into exportable, human-readable reports.
Implemented By: Backend report generation utility.

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

Implemented By: –

7.3.1 Database Access Module (M14)

Secrets: Database schema design and query optimization strategies.
Services: Provides CRUD operations through a unified data access layer.
Implemented By: Drizzle ORM and PostgreSQL.

7.3.2 Data Validation Module (M15)

Secrets: Input validation and sanitization logic.
Services: Ensures user and system inputs conform to expected formats.
Implemented By: Backend validation middleware.

7.3.3 Encryption Module (M16)

Secrets: Encryption algorithms and key management strategies.
Services: Encrypts and decrypts sensitive data to ensure secure storage and communication.
Implemented By: Backend security subsystem.

7.3.4 Communication Module (M17)

Secrets: Message routing, formatting, and delivery protocols.
Services: Sends messages, alerts, and confirmations to users or admins.
Implemented By: Backend communication service and third-party APIs.

7.3.5 Logging Module (M18)

Secrets: Log format, storage mechanisms, and retention policy.
Services: Records system events and errors for debugging and monitoring.
Implemented By: Backend logging framework.

7.3.6 Audit Module (M19)

Secrets: Policy and data structure for recording administrative actions.
Services: Tracks user and admin activities to ensure traceability and compliance.
Implemented By: Backend audit service.

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	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M??
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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]