Module Guide for Software Engineering

Team #2, Campus Connections
Waseef Nayeem
Zihao Du
Matthew Miller
Firas Elayan
Abhiram Neelamraju
Michael Kim

April 4, 2024

1 Revision History

Date	Version	Notes
Jan 11	1.0	Add Section Timeline and Reflection
Jan 15	1.0	Revision 0

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
AR	Augmented Reality
CRUD	Create, Read, Update and Delete
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
Software Engineering	Explanation of program name
SQL	Structured Query Language
SRS	Software Requirements Specification
UC	Unlikely Change
UI	User Interface

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The algorithm for paginating

AC3: The implementation of lecture/event filter

AC4: The protocol of server request and response

AC5: The format of the server data packets

AC6: The layout and styling of lecture/event list components UI

AC7: The layout and styling of lecture/event detail view components UI

AC8: Interaction with real-time map markers

AC9: The look and feel of map layout

AC10: The layout and styling of User Profile component UI

AC11: The layout and styling of User Login component UI

AC12: The layout and styling of friend-related components UI

AC13: The look and feel of AR Interface elements

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: User input for in-app control

UC2: OS on which the application is running

UC3: The information needed to create an account

UC4: The Firebase real-time database implementation

UC5: The ASP.NET server implementation for real-time chatting and location sharing

UC6: The Vuforia AR Camera implementation

UC7: The three access levels of users

UC8: The application will be implemented with Unity Engine

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: Database Module

M3: Server Module

M4: Authentication Module

M5: AR Camera Module

M6: AR Interface Module

M7: Mapbox Module

M8: RealTimeMap Module

M9: User Module

M10: Lecture Module

M11: Event Module

M12: Account Module

M13: Permission Module

M14: User Profile Module

M15: User Login Module

M16: Friend Manager Module

M17: Friend Request Module

M18: Friend Chat Module

M19: Activity Detail View Module

M20: Lecture Detail View Module

M21: Event Detail View Module

M22: Pagination and Filter Module

M23: Lecture List Manager Module

M24: Event List Manager Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	AR Interface Module Real-time Map Module User Module Lecture Module Event Module User Profile Module User Login Module Friend Manager Module Friend Request Module Friend Chat Module Lecture Detail View Module Event Detail View Module Lecture List Manager Module Event List Manager Module Account Module Permission Module
Software Decision Module	Database Module Server Module Authentication Module AR Camera Module Mapbox Module Activity Detail View Module Pagination and Filter 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, 3, 4.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

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

7.2.1 AR Interface Module (M6)

Secrets: How AR User Interface elements are created and displayed.

Services: Creates 3D AR elements when notified by the AR Camera of a new valid target. Handles user input relating to AR UI elements.

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.2.2 Real-time Map Module (M8)

Secrets: How users, events and buildings are organized and displayed on the virtual map provided by the MapBox Module.

Services: Displays points of interest such as buildings or events. Handles user interaction with points of interest. Displays user's avatar at their current location. Displays nearby friend locations on the map.

Implemented By: CampusConnections

Type of Module: Abstract Object

7.2.3 User Module (M9)

Secrets: Data structure of a User.

Services: Provides functionalities to interact with user data for other modules to use.

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.2.4 Lecture Module (M10)

Secrets: Data structure of a Lecture.

Services: Provides CRUD operations for users and administrators to interact with lecture data.

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.2.5 Event Module (M11)

Secrets: Data structure of an Event.

Services: Provides CRUD operations for users and administrators to interact with event data.

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.2.6 Account Module (M12)

Secrets: How user information is stored and used among modules.

Services: Provides functionalities to get and mutate state variables of the User representing current user.

Implemented By: CampusConnections

Type of Module: Abstract Object

7.2.7 Permission Module (M13)

Secrets: How current user authentication information is stored and used among modules.

Services: Provides functionalities to get current user permission information and methods inherited from Authentication Module.

Implemented By: CampusConnections

Type of Module: Abstract Object

7.2.8 User Profile Module (M14)

Secrets: How to display user profile and edit profile handlers

Services: Allows users to view and edit their own data, and allows them to view other user profile to limited degree.

Implemented By: CampusConnections

Type of Module: Library

7.2.9 User Login Module (M15)

Secrets: The layout of account login/creation handlers and input fields

Services: Allows users to enter account information to login to the system or create an account. Provide functionality to reset password as well.

Implemented By: CampusConnections

Type of Module: Library

7.2.10 Friend Manager Module (M16)

Secrets: How to display friends as a list with corresponding handlers including adding, deleting, messaging and viewing.

Services: Displays friends and as as a list and handle user input to add, message, delete a friend.

Implemented By: CampusConnections

Type of Module: Abstract Object

7.2.11 Friend Request Module (M17)

Secrets: How to display friend requests as a list with corresponding handlers to accept or ignore requests.

Services: Displays friend requests and as as a list and handle user input to accept or ignore a request.

Implemented By: CampusConnections

Type of Module: Abstract Object

7.2.12 Friend Chat Module (M18)

Secrets: Procedures related to how chat messages are sent, received and handled on the client side.

Services: Provides a link between the user interface and the backend server to allow users to chat in real time. Converts user input in the form of messages to data packets that are sent to the backend server.

Implemented By: CampusConnections

Type of Module: Abstract Object

7.2.13 Event Detail View Module (M21)

Secrets: How to inherit (M19) with activity type Event.

Services: All services inherited from (M19) with activity type Event

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.2.14 Lecture Detail View Module (M20)

Secrets: How to inherit (M19) with activity type lecture.

Services: All services inherited from (M19) with activity type Event

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.2.15 Lecture List Manager Module (M23)

Secrets: Implementation details of how lectures are displayed as a list with services inherited from (M22).

Services: Displays a sequence of lectures as a list with pagination and filtering functionalities inherited from (M22).

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.2.16 Event List Manager Module (M24)

Secrets: Implementation details of how events are displayed as a list with services inherited from (M22).

Services: Displays a sequence of events as a list with pagination and filtering functionalities inherited from (M22).

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.3 Software Decision Module

7.3.1 Database Module (M2)

Secrets: Data structures of stored data and how database connection is established.

Services: Provides CURD operations to database tables.

Implemented By: Firebase

Type of Module: Library

7.3.2 Server Module (M3)

Secrets: How data is sent to and received from clients.

Services: Provides real-time communication endpoints for chatting and location sharing.

Implemented By: Microsoft

Type of Module: Library

7.3.3 Authentication Module (M4)

Secrets: Authentication token and user data related to authentication

Services: Provides authentication token to existing users and prevent unauthorized users from accessing any accounts.

Implemented By: Firebase

Type of Module: Library

7.3.4 AR Camera Module (M5)

Secrets: How AR targets are detected.

Services: Detects valid AR targets in the device's camera view and notifies the AR Interface module of the newly detected target.

Implemented By: Vuforia

Type of Module: Library

7.3.5 Mapbox Module (M7)

Secrets: How to build and display customizable maps.

Services: Provides different styles of interactive and customizable maps with basic functionalities (e.g. pan, center).

Implemented By: Mapbox

Type of Module: Library

7.3.6 Activity Detail View Module (M19)

Secrets: How details of activities are displayed and changed.

Services: Provides functionality for users and administrators to view, pin, unpin; edit, and delete activities respectively.

Implemented By: CampusConnections

Type of Module: Abstract Data Type

7.3.7 Pagination and Filter Module (M22)

Secrets: Algorithm used for paginating and filter activities.

Services: Displays activities as a paginated list which can be filtered by certain keyword.

Implemented By: CampusConnections

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**
FR1-1	M15
FR2-1	M13, M2, M15
FR2-2	M13, M14, M2
FR2-3	M13, M15
FR2-4	M13, M15
FR2-5	M13, M15
FR2-6	M9, M14, M2
FR2-7	M13, M14
FR2-8	M9, M14, M2
FR3-1	M16, M12
FR3-2	M16, M12
FR3-3	M18, M3
FR3-4	M18, M3, M11
FR3-5	M18, M3, M7
FR3-6	M17, M12
FR4-1	M14, M2, M11, M21, M19
FR4-2	M14, M2, M10, M20, M19
FR4-3	M11, M21, M19, M2, M4
FR4-4	M10, M20, M19, M2, M4
FR4-5	M11
FR4-6	M10
FR4-7	M11, M24, M22
FR4-8	M10, M23, M22
FR5-1	M5
FR5-2	M6
FR6-1	M7, M8, M2
FR6-2	M7, M8, M2
FR6-3	M7, M8, M3

Table 2: Trace Between Functional Requirements and Modules

Req.	**Modules**
LF-A1	M6 M16, M14, M20, M21, M23, M24, M15, M8
LF-A2	M6 M16, M14, M20, M21, M23, M24, M15, M8
LF-S1	M6 M16, M14, M20, M21, M23, M24, M15, M8
LF-S2	M6 M16, M14, M20, M21, M23, M24, M15, M8
UH-EOU1	M6 M16, M14, M20, M21, M23, M24, M15, M8
UH-L1	NA since it is a learning requirement related to document only
UH-UP1	M6 M16, M14, M20, M21, M23, M24, M15, M8
UH-A1	M6 M16, M14, M20, M21, M23, M24, M15, M8
P-SL1	M5, M6
P-SL2	M3, M7
P-SC1	M2, M3
P-SC2	M3
P-SC3	M8
P-SC4	M3, M8
P-SC5	M4, M14, M2
P-SC6	M4, M2
P-PA1	M5
P-RF1	M6 M16, M14, M20, M21, M23, M24, M15, M8
P-RF2	M2, M3, M4
P-RF3	M3
P-RF4	M6
P-SE1	M3
P-SE2	M2, M4
P-SE3	M2, M3, M4
P-L1	All modules (requirement related to coding style)
P-L2	All modules (requirement related to coding style)
OE-EPE1	NA since it is a device related requirement
OE-P1	NA since it is a productization requirement

Table 3: Trace Between Non-Functional Requirements and Modules

Req.	Modules
MS-M1	NA since it is a requirement for maintainers
MS-S1	NA since it is a requirement for maintainers
S-A1	M4, M9
S-A2	M4, M9
S-A3	M4, M9
S-P1	M2, M3, M9, M11, M10
S-P2	M2, M4
CUL-C1	M6, M8, M14, M15, M16, M20, M21, M23, M24
COM-L1	M2, M9, M11, M10

Table 4: Trace Between Non-Functional Requirements and Modules Cont.

AC	Modules
AC1	M1
AC2	M22
AC3	M22
AC4	M3
AC5	M3
AC6	M23, M24
AC7	M19, M20, M21
AC8	M8, M7
AC9	M8
AC10	M14
AC11	M15
AC12	M16, M17, M18
AC13	M6

Table 5: 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.

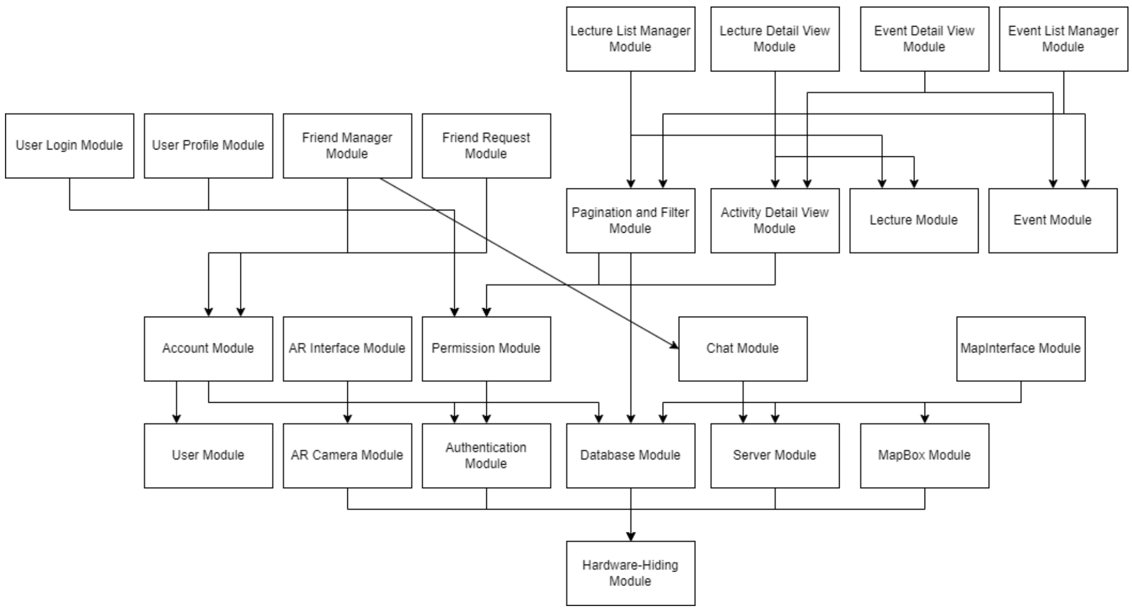

Figure 1: Use hierarchy among modules

10 User Interfaces

User interfaces can be found here.

11 Timeline

Module Name	Team Member	Due Date
User	Zihao Du	Nov 15, 2023
Account	Zihao Du	Dec 4, 2023
Database	Zihao Du	Dec 4, 2023
Friend Request	Zihao Du	Jan 10, 2024
Friend Manager	Zihao Du	Jan 15, 2024
Chat	Waseef Nayeem, Zihao Du	Jan 15, 2024
Mapbox	Waseef Nayeem	Jan 15, 2024
Server	Waseef Nayeem	Jan 15, 2024
Authentication	Michael Kim	Jan 15, 2024
Permission	Michael Kim	Jan 15, 2024
User Profile	Michael Kim	Jan 15, 2024
Lecture	Abhiram Neelamraju	Jan 15, 2024
Lecture List Manager	Abhiram Neelamraju	Jan 15, 2024
Pagination and Filter	Abhiram Neelamraju, Firas Elayan	Jan 15, 2024
Activity Detail View	Matthew Miller	Jan 17, 2024
Event Detail View	Matthew Miller	Jan 17, 2024
Lecture Detail View	Matthew Miller	Jan 17, 2024
Event	Firas Elayan	Jan 17, 2024
Event List Manager	Firas Elayan	Jan 17, 2024
User Login	Michael Kim	Jan 17, 2024
RealTimeMap	Waseef Nayeem	Jan 17, 2024
Mapbox Manual Test	Waseef Nayeem	Jan 26, 2024
Server Manual Test	Waseef Nayeem	Jan 26, 2024

Table 6: CampusConnections Module Completion Timeline

Test Name	Team Member	Due Date
Database Manual Test	Zihao Du	Jan 26, 2024
User Unit Test	Zihao Du	Jan 26, 2024
Authentication Manual Test	Michael Kim	Jan 26, 2024
Event Unit Test	Firas Elayan	Jan 26, 2024
Lecture Unit Test	Abhiram Neelamraju	Jan 26, 2024
Map Scene Manual Test	Waseef Nayeem	Feb 2, 2024
ARCamera	Waseef Nayeem, Zihao Du	Feb 2, 2024
ARCamera Manual Test	Waseef Nayeem, Zihao Du	Feb 2, 2024
ARInterface	Waseef Nayeem, Zihao Du	Feb 2, 2024
AR Scene Manual Test	Waseef Nayeem, Zihao Du	Feb 2, 2024
Friend Scene Manual Test	Zihao Du	Feb 2, 2024
User Login Scene Manual Test	Michael Kim	Feb 2, 2024
User Profile Scene Manual Test	Michael Kim	Feb 2, 2024
List View Manual Test	Abhiram Neelamraju, Firas Elayan	Feb 2, 2024
Detail View Manual Test	Matthew Miller	Feb 2, 2024
Integration Test for Rev 0	All	Feb 5, 2024

Table 7: CampusConnections Test Timeline

A Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

   One of the main limitations of the solution is the time allotted for this project. This is an ambitious project that has various impressive features, but due to the time limitation, we have to limit the number of features and leave some good designs in the waiting room. One way to improve the project is to iterate over the project for several rounds when implementing and ranking all features by significance. For the very first round, we only import the fundamental features, and in the later round, we will work on other less important features and UI. Currently here is a list of features to implement in this round:

   - Real-time system server
   - Multi-user location-sharing map
   - List view of events/lectures with pagination and filtering
   - Detail View of events/lectures
   - User Profile Page
   - Friend Chatting System

   And there are also some features already on the list for the next round like Event sharing, Heat map, etc.

   Another limitation is the budget. Since we are using some third-party library (e.g. AR Engine) and the budget is limited, we can only use the free plan for now. The free plan has limited capacity and watermark, which affects the user experience and limits the features we can implement. To improve this, we need to limit the number of target buildings on the map and provide only a portion of indoor AR features for the first release.

   For this project, data-sharing restrictions limit the features as well. Since the administrator needs to feed the community up-to-date information about campus activities. If the administrator cannot provide good resources, the application is much less helpful for its user. One way to solve the problem is to invite club leaders or even McMaster staff as our administrators so that the community can be always active.

2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select documented design? (LO_Explores)

   We are implementing this application with Unity, using Firebase Database and Authentications. We also designed an ASP.NET server for multi-user chatting and location sharing. The benefit of using Unity Engine is it has great support for AR features. Our AR Engine, Vuforia works perfectly fine with Unity. Also, we can make use of lots of 3D/2D object from Unity Asset Store when implementing maps and our interfaces, which provides an immersive user experience. Unity also has a very active community and detailed documentation as well. Compared to some other front-end frameworks like React, Unity doesn't have good support for UI components as a tradeoff. However, the team thinks map and AR components are much more important features than layout and UI. We are also using Firebase Real-time database, a non-relational database for this project. Compared to relational databases like Microsoft SQL, it is flexible and updates in almost real-time. As a product of Firebase, it collaborates with Firebase Authentication System very well. It provides protection rules for reading and writing on the database side which prevent users from bypassing the front end and attacking the database directly. As a tradeoff, the data in the database are not well organized by schemas like relational databases, and it may be quite difficult to extend and maintain in the future.

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.