Module Guide for Software Engineering

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

January 16, 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
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
CRUD	Create, Read, Update and Delete

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 format of the initial input data.

...

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: Hardware-Hiding Module

M2: Database Module

M3: Server Module

M4: Authentication Module

M5: AR Camera Module

M6: AR Interface Module

M7: Map Module

M8: User Module

M9: Lecture Module

M10: Event Module

M11: User Profile Module

M12: Friend Manager Module

M13: Activity Detail View Module

M14: Lecture Detail View Module

M15: Event Detail View Module

M16: Pagination and Filter Module

M17: Lecture List Manager Module

M18: Event List Manager Module

M19: Notification 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, 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.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	AR Interface
	User Module
	Lecture Module
	Event Module
	User Profile Module
	Friend Manager Module
	Lecture Detail View Module
	Event Detail View Module
	Lecture List Manager Module
	Event List Manager Module
	Notification Module
Software Decision Module	Database Module
	Server Module
	Authentication Module
	AR Camera Module
	Map Module
	Activity Detail View Module
	Pagination and Filter Module

Table 1: Module Hierarchy

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

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 Friend Manager Module (M12)

Secrets: Friend related handlers including adding, deleting, messaging and viewing.

Services: Provides adding friend, deleting friend, message friend, view friend profile functionalities for the current user and displays friends and friend requests as lists.

Implemented By: CampusConnections

Type of Module: Library

7.2.2 Notification Module (M19)

Secrets: Implementation detail and settings of toast notifications and alerts

Services: Provides functionalities to display all notifications, alerts and consents.

Implemented By: CampusConnections

Type of Module: Library

7.2.3 User Module (M8)

Secrets: Implementation details of user data and database structure for user data.

Services: Contains and structures user data for database communication and for other modules to use.

Implemented By: CampusConnections

Type of Module: Library

7.2.4 User Profile Module (M11)

Secrets: Handles database calls and update regarding user information.

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

7.3 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, CampusConnections

Type of Module: Library

7.3.2 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 Authentication, CampusConnections

Type of Module: Library

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

Table 2: Trace Between Functional Requirements and Modules

Req.	Modules
LF-A1	M6 M12, M11, M14, M15, M17, M18, M19
LF-A2	M6 M12, M11, M14, M15, M17, M18, M19
LF-S1	M6 M12, M11, M14, M15, M17, M18, M19
LF-S2	M6 M12, M11, M14, M15, M17, M18, M19
UH-EOU1	M6 M12, M11, M14, M15, M17, M18, M19
UH-L1	
UH-UP1	M6 M12, M11, M14, M15, M17, M18, M19
UH-A1	M6 M12, M11, M14, M15, M17, M18, M19
P-SL1	M5, M6
P-SL2	M3, M7
P-SC1	M2, M3
P-SC2	M3
P-SC3	M19
P-SC4	M3, M7
P-SC5	M4, M11, M2
P-SC6	M4, M2
P-PA1	M5
P-RF1	M19
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	
P-L2	
OE-EPE1	
OE-P1	

Table 3: Trace Between Non-Functional Requirements and Modules

Req.	Modules
MS-M1	
MS-S1	
S-A1	M4, M8
S-A2	M4, M8
S-A3	M4, M8
S-P1	M2, M3, M8, M10, M9
S-P2	M2, M4
CUL-C1	M6, M11, M12, M14, M15, M17, M18
COM-L1	M2, M8, M10, M9

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

AC	Modules
AC1	M1
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

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 Timeline

Module Name	Team Member	Due Date
Database	Zihao Du	Dec. 4, 2023
User	Zihao Du	Nov 15, 2023
Friend Manager	Zihao Du	Jan 15, 2024
Notification	Zihao Du	Feb 5th, 2024
User Profile	Michael Kim	Jan 15, 2024
Authentication	Michael Kim	Jan 15, 2024

Table 6: CampusConnections Module Completion 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)

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)

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.