Module Guide for Software Eng 4G06

Team 2, Parnas' Pals
William Lee
Jared Bentvelsen
Bassel Rezkalla
Yuvraj Randhawa
Dimitri Tsampiras
Matthew McCracken

April 5, 2023

1 Revision History

Date	Developer(s)	Change
January 10 2023	Matthew McCracken, Jared Bentvelsen, Bassel Rezkalla, Yuvraj Randhawa, William Lee, and Dimitri Tsampiras	Initial Draft
March 31 2023	Matthew McCracken	Updated modules to align with end of term codebase

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 Eng 4G06	Explanation of program name
UC	Unlikely Change

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 process by which users discover new content / other users.

AC2: Visual styling and color design choices.

AC3: Connect goals feature up to database.

AC4: Further multimedia options including custom profile pictures and the option to embed images and videos into workouts.

AC5: Track exercise progress with more metrics than just reps and sets. Include quantifiers such as time.

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: Security requirements and legal compliance requirements are unlikely to be changed: NFR15, NFR16, NFR17, NFR18, NFR22.

UC2: To maintain accessibility for all users, the following Usability and Humanity requirements are likely to remain unchanged: NFR2, NFR3, NFR4, NFR5, NFR6, NFR7.

UC3: The performance requirement specified by NFR8 encapsulates the mission of this application, which is to allow users to quickly update / interact with programs. As such, it is unlikely to change.

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: Exercise Module

M3: Workout Module

M4: Program Module

M5: User Login Module

M6: User Sign Up Module

M7: User Profile Module

M8: User Fitness Goal Module

M9: User Workout Browser Module

M10: Creation Module

M11: Workout Creation Module

M12: Program Creation Module

M13: Exercise Creation Module

M14: Quantifier Module

M15: Database Communicator Module

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	exercise workout program user login user sign up user profile user fitness goal workout browser creation workout creation program creation exercise creation database communicator
Software Decision Module	quantifier

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 3.

Requirement	Design
The product shall appear minimal and straightforward	The application will contain an interface that only presents information that is necessary as to prevent cluttering and reduce minimality
The product shall use fonts of readable size to the target user group	Only black and dark colored fonts will be used to display information. Font sizes of 12 and larger will be standard. All industry best practices for font style will be followed.
The product shall be able to be used by untrained fitness enthusiasts and amateurs alike, who receive no training before using it	UI will be simplistic and consistent throughout, reducing the learning curve for users
The product shall be usable by users with hearing loss or partial blindness	The application will rely on both audio and visual cues to indicate correct inputs or application events
The application must inform users when maintenance is taking place and must warn them at least 1 day in advance	The application will utilize pop up screens to display important notifications to the user
The applicaiton will allow users to report offensive content and remove it from their feed	There will be a button on each post that gives user the option to report offensive content

Table 2: Requirements to Design Mapping

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 Eng 4G06* means the module will be implemented by the Software Eng 4G06 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 Exercise Module (M2)

Secrets: The values of quantifiers and descriptions for each exercise.

Services: Provides read and write (get and set) capability for individual exercises.

Implemented By: Exercise.tsx

Type of Module: Abstract Object

7.2.2 Workout Module (M3)

Secrets: The exercises that make up each workout.

Services: Manage (read and write) which exercises comprise a workout.

Implemented By: Workout.tsx

Type of Module: Abstract Object

7.2.3 Program Module (M4)

Secrets: The workouts and scheduling to make up a program.

Services: Manage the scheduling and contents of a program.

Implemented By: Program.tsx

Type of Module: Abstract Object

7.2.4 User Login Module (M5)

Secrets: The functions needed to verify a users identity.

Services: Accepts user credentials and, if correct, provides a token to log user in.

Implemented By: Login.tsx

7.2.5 User Sign Up Module (M6)

Secrets: The functions needed to create a new user in the application database.

Services: Accepts user credential data and, if valid, creates a new user in the database with the provided data.

Implemented By: SignUp.tsx

7.2.6 User Profile Module (M7)

Secrets: The personal data for a user.

Services: Manage all personal user data.

Implemented By: Profile.tsx

7.2.7 User Fitness Goal Module (M8)

Secrets: The quantifiers that users set for their fitness goals.

Services: Manage fitness goals for a specific user.

Implemented By: Goals.tsx

7.2.8 Workout Browser Module (M9)

Secrets: The data structure for filtering and displaying public workouts.

Services: Display public workouts and allow user to search and filter options.

Implemented By: Discover.tsx

7.2.9 Workout Creation Module (M11)

Secrets: The verification of new workout data.

Services: Uses creation module to add new workout to database.

Implemented By: WorkoutEdit.tsx

7.2.10 Program Creation Module (M12)

Secrets: The verification of new program data.

Services: Uses creation module to add new program to database.

Implemented By: CreateProgram.tsx

7.2.11 Exercise Creation Module (M13)

Secrets: The verification of new exercise data.

Services: Uses creation module to add new exercise to database.

Implemented By: Exercise.tsx

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 Quantifier Module (M14)

Secrets: The data types used to quantify workout exercises.

Services: Measures the exercise inputted by users as a unit of measurement.

Implemented By: quantifiers.ts

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	M12
FR2	M2, M4, M13
FR3	M2, M14
FR4	M4, M7
FR5	M4
FR6	M4, M9
FR7	M7
FR8	M7
FR9	M2, M8, M13, M14
FR10	M2, M8, M14
FR11	M8, M9
NFR1	M7
NFR2	M7
NFR3	M7
NFR4	M2, M3, M4, M5, M6, M7, M8, M9, M10
NFR5	M2, M3, M4, M7, M8, M9
NFR6	M2, M3, M4, M7, M8, M9
NFR7	M2, M3, M4, M7, M8, M9
NFR8	M2, M3, M4, M11, M12, M13
NFR9	M9
NFR10	M15
NFR11	M15
NFR12	M15
NFR13	M1
NFR14	M7
NFR15	M7
NFR16	M1
NFR17	M1
NFR18	M1
NFR19	M7, M9
NFR20	M7, M9
NFR21	M7
NFR22	M15

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M9
AC2	M3
AC3	M8
AC4	M3
AC5	M13

Table 4: 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.

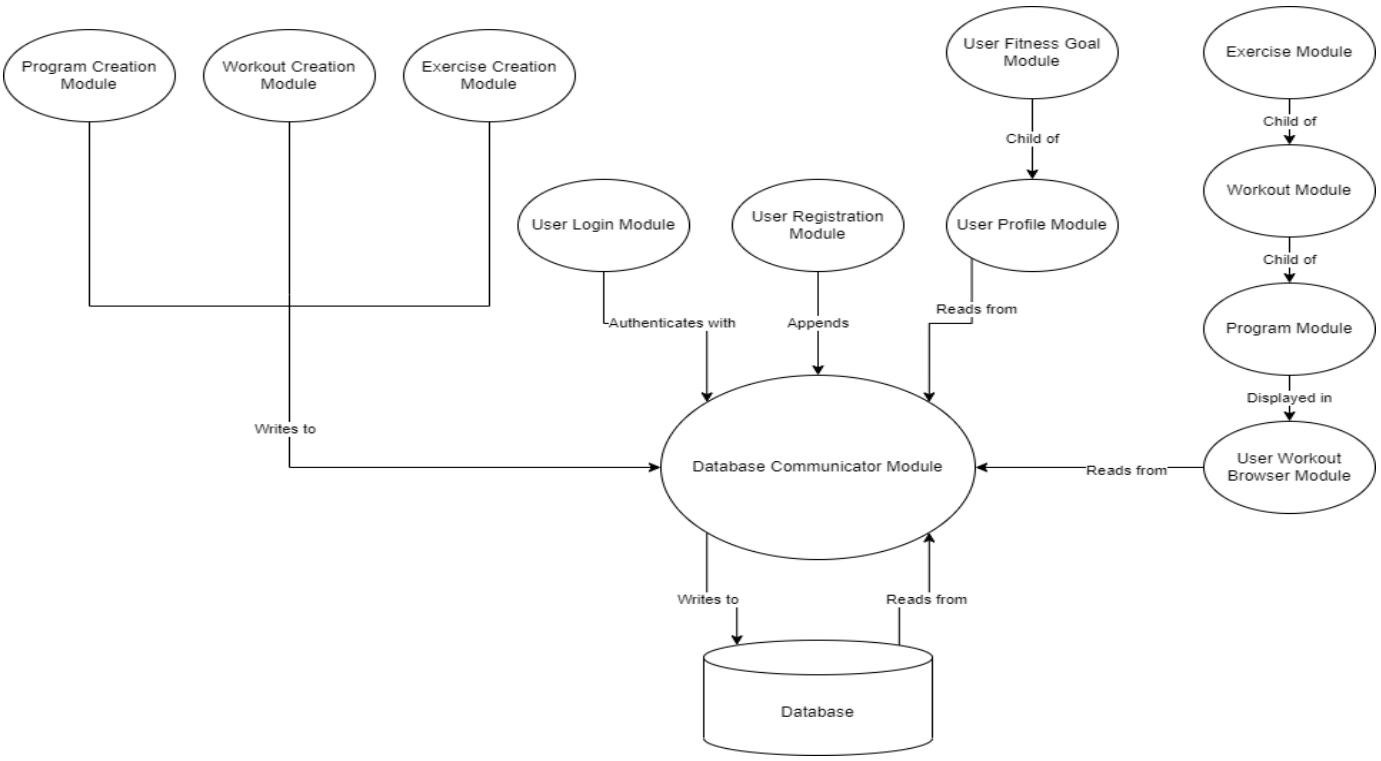

Figure 1: Use hierarchy among modules

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.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. Q: What other designs were considered?
   a) Currently, the users and workout/routine creation are considered separate from each other but during one of the design meetings there was a conversion regarding creation being a user's action. This design involved the user module creating workouts and routines rather than having a separate interface.

   b) An alternative design for the flow of exercise, workout and routines that was considered was time-based. This means that each (workout, routine, exercise) would have a time and their sub-parts would schedule off a portion of the total duration.

2. Q: Why was this one selected?
   a) As a group, we decided to keep these as separate modules because creation is its own process and does not directly interface with the user module. By keeping these as separate processes, it helps exercise the idea of 'separation of concerns'.

   b) We decided not to use time-sectioning for the workouts / routines because it would cause limitations and be much more difficult to implement compared to having them as a sequence of events. If we wanted to collect the total duration of the workout / routine, a simple summation of parts would suffice.