Module Guide for SFWRENG 4G06

Team #12, Team 12.0
Kanugalawattage, Anton
Subedi, Dipendra
Rizkalla, Youssef
Leung, Tamas
Zhao, Zhiming

January 19, 2023

1 Revision History

Date	Version	Notes
01/17/2023	1.0	Inital Draft
01/18/2023	1.1	Small fixes

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
SFWRENG 4G06	The Software Engineering Capstone Project Course at McMaster University
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 statistics collected in matches for each user.

AC2: The container engine used to compile and run code.

AC3: The database engine used to store and retrieve data.

AC4: The algorithm used to select a problem for a round.

AC5: The number of players which get to qualify for a given round.

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).

UC2: The client will be accessed through a browser.

UC3: Persistent data will always be stored in a database.

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: ClientT Module

M3: GameT Module

M4: MatchT Module

M5: UserT Module

M6: UserStatsT Module

M7: ProblemT Module

M8: Difficulty Module

M9: Difficulty Module

M10: TestCaseT Module

M11: SubmissionT Module

M12: JudgeResultT Module

M13: JudgeVerdict Module

M14: TestCaseVerdictT Module

M15: Home Page Module

M16: Profile Page Module

M17: Leaderboard Page Module

M18: Lobby Page Module

M19: Game Page Module

M20: Login Page Module

M21: SubmissionService Module

M22: ProblemsService Module

M23: UserService Module

M24: AuthService Module

M25: LobbyService Module

M26: WebSocketService Module

M27: GameHandler Module

M28: Judge Module

M29: Auth Module

M30: Problems Module

M31: User Module

M32: CodeRunner Interface Module

M33: Database Module

M34: Router Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	ClientT Module
	GameT Module
	MatchT Module
	UserT Module
	UserStatsT Module
	ProblemT Module
	Difficulty Module
	Language Module
	TestCaseT Module
	SubmissionT Module
	JudgeResultT Module
	JudgeVerdict Module
	TestCaseVerdictT Module
	HomePage Module
	ProfilePage Module
	LeaderboardPage Module
	LobbyPage Module
	GamePage Module
	LoginPage Moudle
	SubmissionService Module
	ProblemsService Module
	UserService Module
	AuthService Module
	LobbyService Module
	WebSocketService Module
	GameHandler Module
	Judge Module
	Auth Module
	Problems Module
	User Module
Software Decision Module	CodeRunner Module
	Database Module
	Router 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.

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. *SFWRENG 4G06* means the module will be implemented by the SFWRENG 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

7.2.1 ClientT Module (M2)

Secrets: The data structures used to represent a connected browser client.

Services: Holds the data of a browser client.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.2 GameT Module (M3)

Secrets: The data structures used to represent a game instance.

Services: Holds the data of a game instance.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.3 MatchT Module (M4)

Secrets: The data structures used to represent a completed match.

Services: Holds the data of a completed match for a specific user.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.4 UserT Module (M5)

Secrets: The data structures used to represent a user.

Services: Holds the data of a user.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.5 UserStatsT Module (M6)

Secrets: The data structures used to represent the game stats of a user.

Services: Holds the data of a user's stats.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.6 ProblemT Module (M7)

Secrets: The data structures used to represent a problem.

Services: Stores the data of a problem.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.7 Difficulty Module (M8)

Secrets: None

Services: Defines the set of possible difficulties for a problem.

Implemented By: CodeChamp

Type of Module: Record

7.2.8 Language Module (M9)

Secrets: None

Services: Defines the set of supported languages.

Implemented By: CodeChamp

Type of Module: Record

7.2.9 TestCaseT Module (M10)

Secrets: The data structures used to hold the values for a problem's test case.

Services: Stores values for a test case's inputs and outputs.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.10 SubmissionT Module (M11)

Secrets: The data structures used to hold the values for a user's code submission.

Services: Stores values for a user's code submission and ensures that the user selected a language supported by the system.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.11 JudgeResultT Module (M12)

Secrets: The algorithm used to calculate the overall verdict for a submission.

Services: Outputs a verdict given the user's outputs for a set of test cases to inform them of their submission's result.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.12 JudgeVerdict Module (M13)

Secrets: None

Services: Defines the set of possible judging verdicts for a code submission.

Implemented By: CodeChamp

Type of Module: Record

7.2.13 TestCaseVerdictT Module (M14)

Secrets: The data structures used to hold the values for a test case's result.

Services: Stores values for a test case's results.

Implemented By: CodeChamp

Type of Module: Abstract Data Type

7.2.14 Home Page Module (M15)

Secrets: The algorithms used to handle inputs and display interactions on the Home Page.

Services: Host buttons that let user navigate to each page,

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.15 Profile Page Module (M16)

Secrets: The algorithms used to retrieve player statistics and display the Profile Page.

Services: Displays user statistics.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.16 Leaderboard Page Module (M17)

Secrets: The algorithms used to retrieve Leaderboard data and display the Leaderboard Page.

Services: Displays Leaderboard data.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.17 Lobby Page Module (M18)

Secrets: The algorithms used to retrieve Lobby data and display the Lobby Page.

Services: Displays the lobby of the current game, shows start button and lobby code

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.18 Game Page Module (M19)

Secrets: The algorithms used to handle inputs and display interactions on the Game Page.

Services: Displays the game page, takes in input allowing use to input code, select submit button or leave.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.19 Login Page Module (M20)

Secrets: The algorithms used to allow user to login and display on the Login Page.

Services: Displays the login page and allows users to sign in.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.20 SubmissionService Module (M21)

Secrets: The algorithm used to send code submissions and return submission results from a server module.

Services: Accepts code inputs, sends network requests, receives network responses and returns appropriate responses based on the user's submission.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.21 ProblemsService Module (M22)

Secrets: The algorithms used to send and retrieve problem data from a server module.

Services: Get problems by problem id from the server

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.22 UserService Module (M23)

Secrets: The algorithms used to send and retrieve user data from a server module.

Services: Get leaderboard data and user stats from the server

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.23 AuthService Module (M24)

Secrets: The algorithm used to authorize users to the server

Services: Login, login and get the current user logged in

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.24 LobbyService Module (M25)

Secrets: The algorithm used to connect clients to a lobby of a game instance.

Services: Adds, removes and send lobby state updates to clients from a game instance

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.25 WebSocketService Module (M26)

Secrets: The algorithm used to send and receive bi-directional messages between clients and servers for a client.

Services: Sends, receives and processes messages/events between clients and servers on the client.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.26 GameHandler Module (M27)

Secrets: The algorithm used to handle game events on the server.

Services: Sends, receives and processes messages/events between from the server to clients. Handles logic of the game events such as a client completing a round and moving the game to the next round.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.27 Judge Module (M28)

Secrets: The algorithm used to compute verdicts for individual test cases.

Services: Judges submissions and returns their result. Informs the GameHandler (M27) of correct submissions.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.28 Auth Module (M29)

Secrets: The algorithm used to authorize user login.

Services: Authorizes user login attempts for the game.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.29 Problems Module (M30)

Secrets: The algorithm used to handle problem creation and retrieval.

Services: Get Problems, Add problems, Get random problem, and get problem from id from the database

Implemented By: CodeChamp

Type of Module: Abstract Object

7.2.30 User Module (M31)

Secrets: The algorithm used to handle users and matches.

Services: Save user matches, creating user, getting users, getting leaderboard and getting user stats.

Implemented By: CodeChamp

Type of Module: Abstract Object

7.3 Software Decision Module

7.3.1 CodeRunner Module (M32)

Secrets: The data structures and algorithms used to compile and run code.

Services: Provides code compilation, a containerized run-time environment and interactability with the run-time's standard input and standard output streams.

Implemented By: Docker

7.3.2 Database Module (M33)

Secrets: The data structures and algorithms used to store and retrieve data.

Services: Allows concurrent storage and retrieval of abstract data types.

Implemented By: MongoDB

7.3.3 Router Module (M34)

Secrets: The network protocols used to navigate web pages.

Services: Provides a routing function to navigate between page modules.

Implemented By: Browser

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	M27, M18, M25 M26
FR2	M27, M18, M25 M26
FR3	M27
FR4	M19, M22, M26
FR5	M19, M27
FR6	M28, M19
FR7	M13
FR8	M19, M10, M12
FR9	M27, M26, M25
FR10	M27, M26, M25
FR11	M27, M26, M25
FR12	M30
FR13	M10, M14
FR14	M32
FR15	M32
FR16	M19, M28
FR17	M29, M31
FR18	M20, M29, M31
FR19	M20, M29
FR20	M28, M32
FR21	M28, M32
FR22	M30
FR23	M27, M18, M25
FR24	M27, M15, M18, M25
FR25	M31, M16
FR26	M31, M16
FR27	M31, M16
FR28	M31, M16
FR29	M17, M31, M16

Table 2: Trace Between Functional Requirements and Modules

Req.	Modules
NFR1	M15, M16, M17, M20, M18, M19
NFR2	M15, M16, M17, M20, M18, M19
NFR3	M15, M16, M17, M20, M18
NFR4	M15, M16, M17, M20, M18, M19
NFR5	M15, M16, M17, M20, M18, M19
NFR6	M15, M16, M17, M20, M18, M19
NFR7	M19, M12
NFR8	All
NFR9	M15, M16, M17, M20, M18, M19
NFR10	M30, M22, M33
NFR11	M29, M24, M20
NFR12	M29, M24, M20
NFR13	M15, M16, M17, M20, M18, M19
NFR14	All
NFR15	M15, M16, M17, M20, M18, M19

Table 3: Trace Between Non-functional Requirements and Modules

AC	Modules
AC1	M6
AC2	M32
AC3	M33
AC4	M30
AC5	M27

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.

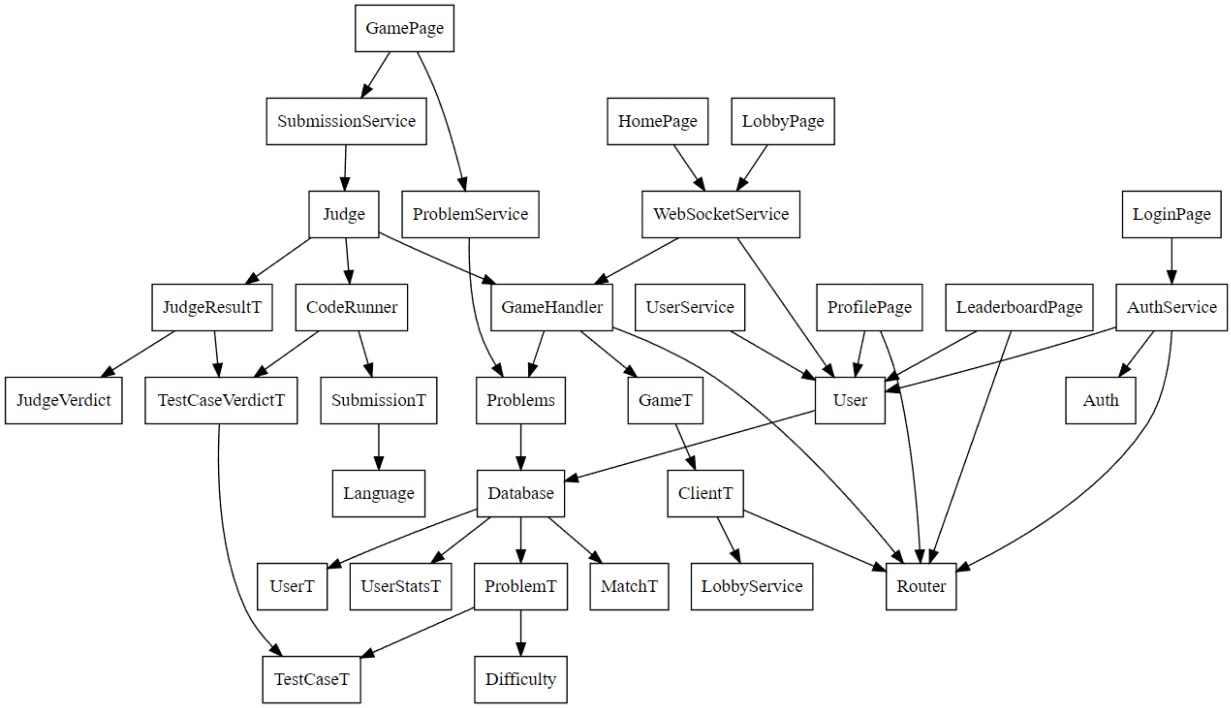

Figure 1: Use hierarchy among modules

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.