Module Guide for Chess Connect

Team #4,
Alexander Van Kralingen
Arshdeep Aujla
Jonathan Cels
Joshua Chapman
Rupinder Nagra

January 18, 2023

1 Revision History

Table of Revisions

Table 1: Revision History

Date	Developer(s)	Change
2023-01-16	Jonathan Cels, Rupinder Nagra	Web Application Modules
2023-01-18	Jonathan Cels, Rupinder Nagra	Finalized Web Application Modules

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
UC	Unlikely Change
FEN	Forsyth-Edwards Notation

Contents

List of Tables

List of Figures

3 Introduction

3.1 Overview

The Chess Connect project allows two users to play a game of chess on a physical board with the information being transmitted to an online web application over Bluetooth. Currently, there is no way for players to seamlessly switch between playing on a physical board and playing online, but Chess Connect intends to change this by creating a central platform that will provide flexibility and remove barriers for new players looking to learn the game.

3.2 Context

This document is the Module Guide (MG), which is created after the Software Requirements Specification (SRS). The purpose of the Software Requirements Specification document is to present a description of the software system to be developed, including the functional and non-functional requirements for the project. The following MG has a different purpose, where it is instead providing a modular decomposition of the system, showing the modular structure of the application. The MG also describes how the requirements in the SRS are met with the modular structure that is described. Along with the MG, it is also necessary to create a Module Interface Specification (MIS) explaining the semantics and syntax of each module. Examples of such semantics and syntax includes the access routines, state variables, inputs/outputs, and exceptions of the modules. This document will further expand on the information provided in the MG.

3.3 Design Principles

Information Hiding and Encapsulation are some of the design principles being used to build a modular structure of our application. The project should also assess the software metric of coupling and cohesion. Ideally, the project has high cohesion, and a low degree of coupling. Information Hiding is the process of hiding the details of an object or function. This process disassociates the calling code from the internal workings of the object or function being called. This makes it possible to change the hidden portions without having to also change the calling code. Encapsulation is a design principle that makes it easier to enforce information hiding. Encapsulation simply hides the states of an object of a class by preventing direct access to it. A high cohesion signifies that the methods and data within a module are closely related. Low coupling means that there is a low degree of interdependence between the modules of the system.

3.4 Document Structure

The document structure is organised as follows:

- Section 4 lists Anticipated and Unlikely Changes to the system's implementation. This list is used for the Traceability Matrices later in the document.

- Section 5 presents the Module Hierarchy, listing all the modules and their hierarchy by levels.
- Section 6 describes the Connection Between Requirements and Design, which details how the software requirements are related to the modules.
- Section 7 describes the Module Decomposition, detailing the module secrets, services, and implementations.
- Section 8 provides the Traceability Matrices. The first matrix connects the functional and nonfunctional requirements to the modules. The second matrix connects anticipated changes from Section 2 to the modules.
- Section 9 presents the Uses Hierarchy diagram for the application. The model shows the uses relations 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 format in which the data is passed from the hardware to the web application.

AC2: The implementation of the data structure that stores the game board state.

AC3: User options to change the font and board size.

AC4: The GUI and interface design.

AC5: The amount of time for a move to be reflected on the web application.

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 The user option to play against another user online.

UC2 The interface functionality of the system.

UC3 The rules and values related to the game itself.

UC4 The purpose of the system to allow users to play a game of chess against an opponent.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Web Application Input Module

M3: Display Module

M4: Web Application Output Module

M5: User Mode Module

M6: Board Module

M7: Web Application Game State Module

M8: Engine Module

6 Connection Between Requirements and Design

The design satisfies the system requirements developed in the SRS. The Input module will handle all interactions from the inputs provided by the user, such as the FEN String, the user mode, and whether the game has terminated. The Display module outputs the visual component of the interface, and this module will cover all requirements that have a component of the user directly interacting with the system. The Board and Game modules cover the board setup and positions along with checking the end game conditions such as Checkmate and Stalemate. The look and feel requirements are satisfied by the input and output modules as they define the core areas of user interaction. The usability and humanity requirements are again satisfied by the input and output modules which define the user interaction. Performance requirements are to be satisfied by the combination of all modules, but the key modules that impact performance are software decision hiding modules. The operational and environmental requirements, and the maintainability and support requirements

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Web Application Input Module
	Display Module
	Web Application Output Module
	?
	?
	?
	?
Software Decision Module	User Mode Module
	Board Module
	Web Application Game State Module
	Engine Module
	?
	?
	?

Table 2: Module Hierarchy

are achieved by all modules. This is because all modules are designed to be operational for the environment (the server and client) and maintained for a set time (end of semester). Security and legal requirements are covered by the entirety of the system. This is because the accessibility of the repository and the JavaScript standard in use affects all modules.

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. *Chess Connect* means the module will be implemented by the Chess Connect 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 Modules

7.2.1 Web Application Input Module (M2)

Secrets: Input data.

Services: Takes in input data of current board state to provide to other modules.

Implemented By: Chess Connect (Node.js libraries, Bluetooth)

7.2.2 Display Module M3

Secrets: Graphics output data.

Services: Allows users to view the current board configuration of the system.

Implemented By: Chess Connect (React.js framework)

7.2.3 Web Application Output Module M4

Secrets: None.

Services: Takes game state and engine moves, encodes the data, and transmits it.

Implemented By: Chess Connect (React.js framework, Bluetooth)

7.3 Software Decision Module

7.3.1 User Mode Module M5

Secrets: User mode logic and data.

Services: Handles switching between user modes and communicating with mode-specific modules.

Implemented By: Chess Connect (React.js framework)

7.3.2 Board Module M6

Secrets: Board data.

Services: Stores and modifies board state information.

Implemented By: Chess Connect (React.js framework)

7.3.3 Web Application Game State Module M7

Secrets: Game state and data.

Services: Handles checking the game state (none, check, checkmate, stalemate).

Implemented By: Chess Connect (React.js framework, Node.js libraries)

7.3.4 Engine Module M8

Secrets: Chess engine moves.

Services: Uses the board state to calculate 3 possible moves.

Implemented By: Chess Connect (Node.js libraries and Stockfish)

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	M1, M2, M??, M??
R2	M2, M??
R3	M??
R4	M4, M??
R5	M4, M??, M??, M??, M??, M??
R6	M4, M??, M??, M??, M??, M??
R7	M4, M??, M??, M??, M??
R8	M4, M??, M??, M??, M??
R9	M??
R10	M4, M??, M??
R11	M4, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	Modules
AC??	M1
AC??	M2
AC??	M??
AC??	M??
AC??	M4
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

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

FEN. Fen (forsyth-edwards notation) - chess terms. https://www.chess.com/terms/fen-chess. Accessed: 2023-01-18.

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.