SE 3XA3: Module Guide
ChessAce

Team 18, Team MIF
Jerry Ke, kex1
Harry Fu, fuh6
Morgan Cui, cuim2

December 5, 2018

Contents

List of Tables

List of Figures

1 Introduction

1.1 Overview

The purpose of this project is to re-implement the open source package ChessOOP which allows two players to play a chess game on one device offline.

1.2 Design Principles

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 (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represents 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 **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used 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.

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 9th, 2018	1.0	Starting create the Module Guide
Nov 30th, 2018	2.0	Reversion 1

1.3 context

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (?). 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 MG also documents how the system meets both functional and non-functional requirements that we were specified in the SRS. 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. (?)

  After the creation of the MG, the Module Interface Specification(MIS) is created. The MIS explains the semantics and syntax of exported functions (input, output and exceptions) for each module, essentially providing further detail on each module that is specified in the MG. (?)

The rest of the document is organized as follows. Section **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** 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 **??** describes the use relation between modules.

2 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 **??**, and unlikely changes are listed in Section **??**.

2.1 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.

AC3: The parameter of the input data.

AC4: The format of the final output/display mechanisum.

AC5: How long for each turn is determined from input data.

AC6: The players' name and color is determined using the input data.

AC7: The graphical user interface element such as the chess board style and the pictures of each chess pieces.

AC8: Default settings for input.

2.2 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: There will always be a source of input data external to the software.

UC3: The basic rule of the chess game could not be changed(the basic algorithm of this program intended not to be changed).

UC4: Basic movements for each pieces could not be changed.

UC5: The numbers of rows and columns of chess board could not be changed.

UC6: The initial position of each pieces could not be changed.

UC7: The number of players could not be changed.

3 Module Hierarchy

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

~~Hardware-Hiding Module~~

M1: Parameter Module

M2: Time Module

M3: Player Module

M4: GUI display Module

M5: Board Module

M6: Output Module

M7: Cell Module

M8: Pieces Module

Level 1	**Level 2**	**Level 3**
~~Hardware-Hiding Module~~		
Behaviour-Hiding Module	Input module Default module	Parameter module Time module Player module
	GUI display module Board module Output Module	
Software Decision Module	Cell module Pieces module	

Table 2: Module Hierarchy

4 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 **??**.

The system was designed to satisfy the requirements. The Main class connects all components together and ensures the program can be executed. Performance, Operational and Environment requirements are satisfied through the Hardware-hiding Module as it serves as a virtual hardware used by the rest of the system and provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs. The GUI display Module shows all the operations using graphic interface which can satisfy Look and Feel, Usability and Humanity requirements. Maintainablity requirement is satisfied by Board Module and Cell Module. The Board Module stores and shares data needed by other modules by using cell module, while the Cell Module initializes position of each piece of the game.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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. Also indicate if the module will be implemented specifically for the 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. Whether or not this module is implemented depends on the programming language selected.

~~Hardware Hiding Modules~~

~~[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~~

5.1 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: –

5.1.1 Input Module

Secrets: The inputs of data.

Services: Converts the input data into the data structure used by the input module.

Implemented By: –

- Parameter Module (M**??**)

  Secrets: The parameters of the input data.

  Services: Converts the input parameters data into the data structure.

  Implemented By: Java Libraries

5.1.2 Default Module

Secrets: The default setting of the game.

Services: Provides users the default setting to the game used by the default module.

Implemented By: –

- Time Module (M**??**)

  Secrets: The time setting of the game.

  Services: Allows users to set the timing before the game starting by using parameter module.

  Implemented By: Java Libraries

- Player Module (M**??**)

  Secrets: The player setting of the game.

  Services: Allows users to set players before the game starting by using parameter module.

  Implemented By: Java Libraries

5.1.3 GUI display Module(M??)

Secrets: All the operations.

Services: Shows all the operations using graphic interface by using pieces module.

Implemented By: Java Libraries

5.1.4 Board Module (M??)

Secrets: Data

Services: Stores and shares data needed by other modules by using cell module.

Implemented By: Java Libraries

5.1.5 Output Module (M??)

Secrets: End game and checkmate.

Services: Provides the result of the game by using board module.

Implemented By: Java Libraries

5.2 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: N/A

5.2.1 Cell Module (M??)

Secrets: Position of pieces.

Services: Initializes position of each piece of the game by using pieces module.

Implemented By: Java Libraries

5.2.2 Pieces Module (M??)

Secrets: Possible movement of each piece.

Services: Highlights possible movenment each turn by using parameter module.

Implemented By: Java Libraries

6 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	M??, M??
FR2	M??, M??
FR3	M??, M??
FR4	M??, M??, M??
FR5	M??
FR6	M??, M??
FR7	M??, M??
FR8	M??, M??
FR9	
FR10	M??,M??
FR11	M??, M??
FR12	M??, M??
FR13	M??
FR14	M??, M??
FR15	M??, M??
FR16	M??, M??
FR17	M??
NF3.1	M??, M??
NF3.2	M??, M??
NF3.3.1	M??
NF3.3.2	automated
NF3.3.3	M??
NF3.3.4	automated
NF3.4.1	automated
NF3.4.2	automated
NF3.4.3	automated
NF3.5.1	M??, M??
NF3.5.2	automated
NF3.6	automated
NF3.7	automated
NF3.8	automated
NF3.9	automated

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC??	NULL
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M?? M??

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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 **??** 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.

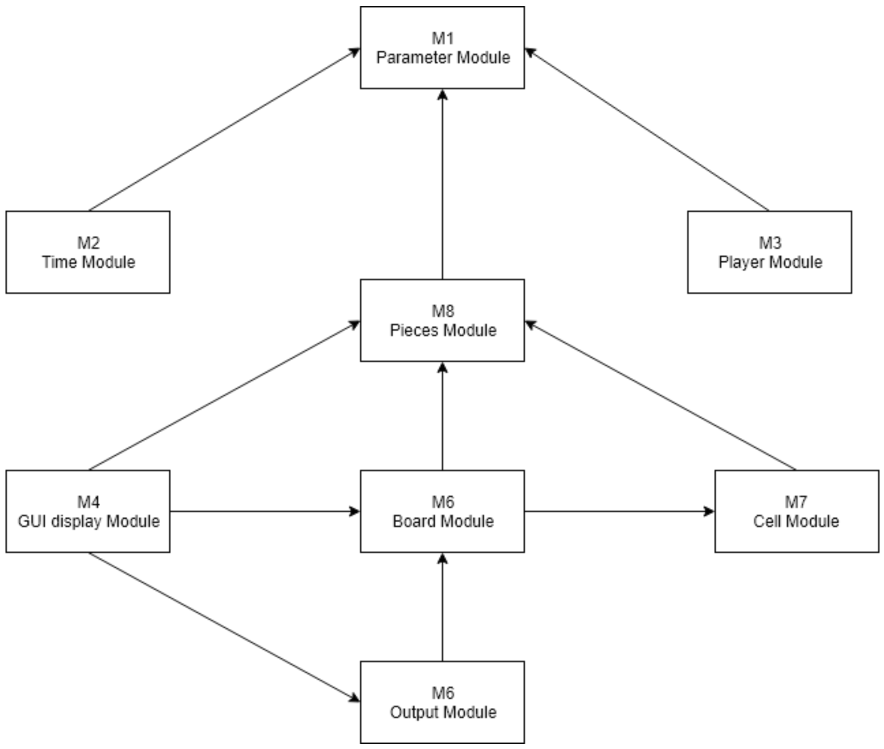

Figure 1: Use hierarchy among modules