# Module Guide for Farming Matters

Team #14, The Farmers
Brandon Duong
Andrew Balmakund
Mihail Serafimovski
Mohammad Harun
Namit Chopra

April 5, 2023

## 1 Revision History

| Date | Developer(s) | Change |
|---|---|---|
| 18/01/2023 | Namit Chopra, Brandon Duong<br>Andrew Balmakund, Mohammad Harun<br>Mihail Serafimovski | Finished first version |
| 04/03/2023 | Namit Chopra, Brandon Duong<br>Andrew Balmakund, Mohammad Harun<br>Mihail Serafimovski | Finished second version |
| 04/05/2023 | Namit Chopra, Brandon Duong<br>Andrew Balmakund, Mohammad Harun<br>Mihail Serafimovski | Addressed GitHub issues |

## 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 |
| FR | Functional Requirement |
| NR | Non-Functional Requirement |
| LF | Look and Feel Requirement |
| UH | Usability and Humanity Requirement |
| PR | Performance Requirement |
| OE | Operations and Environmental Requirement |
| MR | Maintainability and Support Requirement |
| SR | Security Requirement |
| LR | Legal Requirement |
| SC | Scientific Computing |
| SRS | Software Requirements Specification |
| UI | User Interface |
| Farming Matters | Final Year Software Engineering Capstone Project 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 et al., 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 laid out by Parnas 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, the 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 in 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 **??** 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.

# 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 adopted here is called design for change.

**AC1:** Fluctuating prices for purchasing an item

**AC2:** Adding more variety to the type of avatars for the user to interact with

**AC3:** Adding more items in the shop

**AC4:** The effects of natural disasters on the farm grid

**AC5:** The method for determining the likelihood of future events for the consultant

**AC6:** Multiple soundtracks for different aspects of the game

**AC7:** The method to sell harvested crops

**AC8:** The method to buy insurance for a purchased item

**AC9:** The statistical technique utilized for the insurance system

**AC10:** The user interface layout of the game

**AC11:** The method of saving game state

**AC12:** The number of turns per season

## 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 this 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:** User input for the in-game controls

**UC2:** The details needed to create an account

**UC3:** The environment in which the user will be able to play the game

**UC4:** The system will only one login session per user

**UC5:** The specific game data collected for research

**UC6:** The ability to purchase and sell items

**UC7:** The ability to access a consultant

**UC8:** The authentication of a user is required before performing actions in the game

**UC9:** The ability to delete user data

**UC10:** The ability to plant and harvest a crop

**UC11:** The changing of seasons

## 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 be implemented.

**M1:** GenerateStatistic

**M2:** Avatar

**M3:** AvatarMenu

**M4:** Consultant

**M5:** OtherAvatar

**M6:** SeasonalEvents

**M7:** Item

**M8:** Inventory

**M9:** Shop

**M10:** DatabaseOperations

**M11:** MusicPlayer

**M12:** GameSettings

**M13:** AuthState

**M14:** Socket

**M15:** ClientFirebase

**M16:** User

**M17:** AuthError

**M18:** CreateAccount

**M19:** Login

**M20:** ServerFirebase

**M21:** RedisClient

**M22:** Server

**M23:** Seed

**M24:** FarmTile

**M25:** FarmGrid

**M26:** GameController

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | - |
| Behaviour-Hiding Module | GameController<br>AvatarMenu<br>Shop<br>Inventory<br>FarmGrid<br>GameSettings<br>CreateAccount<br>Login<br>SeasonalEvents<br>GenerateStatistic |
| Software-Hiding Module | Avatar<br>Consultant<br>OtherAvatars<br>FarmTile<br>Seed<br>Item<br>DatabaseOperations<br>ServerFirebase<br>ClientFirebase<br>AuthState<br>Socket<br>RedisClient<br>Server<br>MusicPlayer<br>User<br>AuthError |

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 **??**.

# 7 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. *Farming Matters* means the module will be implemented by the Farming Matters 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

**Secrets:** The data structure and the algorithm used to implement the virtual hardware.

**Services:** Serves as 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 behaviors.

**Services:** Includes programs that provide externally visible behavior 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 GameController (M26)

**Secrets:** User input handlers and in-game logic.

**Services:** Responsible for calling all related submodules throughout a session.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.2 AvatarMenu (M3)

**Secrets:** User input handling during the menu.

**Services:** Interact with the user selecting several avatars to choose to interact with.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.3 Shop (M9)

**Secrets:** The data structure of a Shop.

**Services:** Provide functionality to interact with various items in a shop, make purchases, and update the inventory.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.4 Inventory (M8)

**Secrets:** The data structure of an Inventory.

**Services:** Provide the ability to get the current state of the inventory, add items to an inventory, and remove items from an inventory.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.5 FarmGrid (M25)

**Secrets:** The data structure of a Farm Grid.

**Services:** Provide the ability to get the current state of the grid tiles and add tiles to the current grid tiles.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.6 GameSettings (M12)

**Secrets:** User adjustable setting values.

**Services:** The ability to change volume and opt out of the study.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.7 CreateAccount (M18)

**Secrets:** User account details.

**Services:** Provides functionality for registering an account to create the game and handling invalid user inputs.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.8 Login (M19)

**Secrets:** User account details.

**Services:** Provides functionality for logging into the system and handling invalid user inputs.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.9 GenerateStatistic (M1)

**Secrets:** The real probability of an event occurring.

**Services:** Provides functionality for generating consultant statements, and general avatar statements, to determine if an event is happening and what type of event is happening.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.2.10 SeasonalEvents (M6)

**Secrets:** The different statements of each event.

**Services:** Displays a visual dialog of what event had just occurred and it also will remove all crops from the current farm. It will only store crops that are insured in the inventory, all other crops that were planted are forever gone.

**Implemented By:** Farming Matters

**Type of Module:** Library

## 7.3 Software Decision Module

**Secrets:** The design decision is 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 does not provide direct interaction with the user.

**Implemented By:** –

#### 7.3.1 Avatar (M2)

**Secrets:** The data structure of an Avatar.

**Services:** Provide functionality to get and set various properties of an Avatar.

**Implemented By:** Farming Matters

**Type of Module:** Abstract Object

#### 7.3.2 Consultant (M4)

**Secrets:** Avatar statements.

**Services:** Provides functionality to generate a statement that assesses the current state of the game and provides a statistic based on a type of decision (probabilistic or deterministic).

**Implemented By:** Farming Matters

**Type of Module:** Record

#### 7.3.3 OtherAvatars (M5)

**Secrets:** Avatar statements.

**Services:** Provides functionality to generate a statement related to a specific avatar.

**Implemented By:** Farming Matters

**Type of Module:** Record

#### 7.3.4 FarmTile (M24)

**Secrets:** Data structure of a farm tile.

**Services:** Provides functionality to plant a seed, harvest a crop, determine what has been planted on a given tile, and how long a seed has been growing.

**Implemented By:** Farming Matters

**Type of Module:** Abstract Object

#### 7.3.5 Seed (M23)

**Secrets:** Data structure of a seed.

**Services:** Provides functionality to determine the seed growth length, the seasons in which the seed can grow, and the sellable price ranges.

**Implemented By:** Farming Matters

**Type of Module:** Abstract Object

#### 7.3.6 Item (M23)

**Secrets:** Data structure of an item.

**Services:** Provides functionality to get and set the price of an item, get the name of an item, get the quantity of an item, and get the type of the item.

**Implemented By:** Farming Matters

**Type of Module:** Abstract Object

#### 7.3.7 DatabaseOperations (M10)

**Secrets:** Establishing database connection.

**Services:** Provides functionality for terminating the database connection, adding and removing users from the database, logging a given data, and saving and loading the current state of the game.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.8 ServerFirebase (M20)

**Secrets:** Firebase API credentials.

**Services:** Connects the server to a Firebase API.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.9 ClientFirebase (M15)

**Secrets:** Firebase API credentials.

**Services:** Connects the client to a Firebase API.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.10 AuthState (M13)

**Secrets:** User session details.

**Services:** Validates if a user is authenticated session and handles unauthorized sessions.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.11 Socket (M14)

**Secrets:** User session details.

**Services:** Grants the user access to play the game.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.12 RedisClient (M21)

**Secrets:** API credentials needed to access Firebase instance.

**Services:** Handles invalid credentials.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.13 Server (M22)

**Secrets:** Internal routing and database setup.

**Services:** Provides communication protocols to connect the front-end and the back-end.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.14 MusicPlayer (M11)

**Secrets:** Audio files.

**Services:** Provides the ability to load, start and stop playing an audio file.

**Implemented By:** Farming Matters

**Type of Module:** Library

#### 7.3.15 User (M16)

**Secrets:** User credentials stored in Firebase system.

**Services:** Retrieve user information.

**Implemented By:** Firebase

**Type of Module:** Record

#### 7.3.16 AuthError (M17)

**Secrets:** Authentication-specific details for a user.

**Services:** Raises in error for unverified users.

**Implemented By:** Firebase

**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. Here is the Software Requirements Specification document to find specific details to the corresponding requirements listed below in Table **??**.

| **Req.** | **Modules** |
|---|---|
| FR1 | M18, M13, M19, M10 |
| FR2 | M19, M10 |
| FR3 | M9, M8 |
| FR4 | M8, M7 |
| FR5 | M13, M18, M19 |
| FR6 | M9, M8, M7 |
| FR7 | M25, M24, M23, M7, M8 |
| FR8 | M8, M7, M25, M24 |
| FR9 | M8, M7, M25, M24 |
| FR10 | M8, M9, M25, M24, M26 |
| FR11 | M8, M7, M25, M24 |
| FR12 | M25, M24,M26 |
| FR13 | M2, M3, M4, M26 |
| FR14 | M8, M9, M7, M26 |
| FR15 | M8, M7 |
| FR16 | M10, M22, M6, M1, M4 , M9, M8, M25, M12, M26 |
| FR17 | M10, M22, M6, M1, M4 , M9, M8, M25, M12, M26 |
| FR18 | M25, M24, M26 |
| FR19 | M4, M26 |
| FR20 | M1, M6, M25, M24 M26 |
| FR21 | M6, M26 |
| FR22 | M4, M26 |
| LF1 | M9, M8, M3, M12 |
| LF2 | M9, M8, M3, M12 |
| LF3 | M11 |
| LF4 | M9, M8, M3, M25, M24, M4, M5, M7 |
| LF5 | M26 |
| LF6 | M9, M8, M3, M25, M24, M4, M5, M7, M26 |
| UH1 | M9, M8, M3, M12, M26 |
| PR1 | M10, M22, M26 |
| PR2 | M10, M22, M26 |
| PR3 | M10, M22, M26 |

Table 2: Trace Between Requirements and Modules

| **Req.** | **Modules** |
|---|---|
| OE1 | M19, M26 |
| OE2 | M19, M26 |
| MR1 | M11, M7, M3 |
| SR1 | M19, M18 |
| SR2 | M19, M18, M10 |
| SR3 | M19, M18, M10 |
| LR1 | M10, M12 |
| LR2 | M7, M11 |

Table 3: Cont. Trace Between Requirements and Modules

| **AC** | **Modules** |
|---|---|
| AC1 | M9, M7 |
| AC2 | M3, M5 |
| AC3 | M7 |
| AC4 | M26, M25, M24, M4 |
| AC5 | M4, M26 |
| AC6 | M11 |
| AC7 | M9, M8 |
| AC8 | M9, M8, M26, M7 |
| AC9 | M9,M7 |
| AC10 | M9, M8, M3, M12 |
| AC11 | M10, M26 |
| AC12 | M26 |

Table 4: Trace Between Anticipated Changes and Modules

## 9 Use Hierarchy Between Modules

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

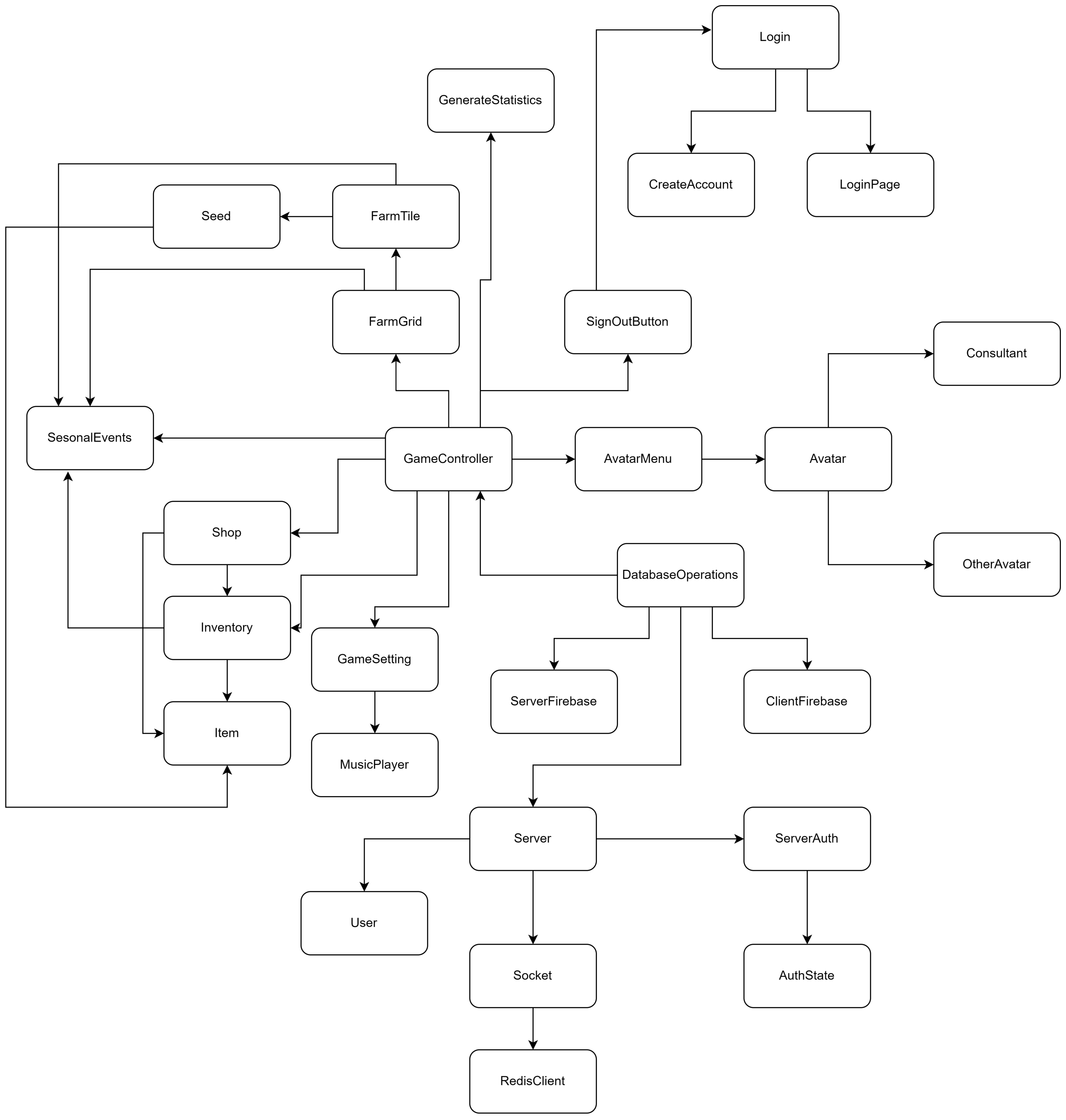

Figure 1: Use hierarchy among modules

## References

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.