Module Guide for Sandlot

Team 29
Nicholas Fabugais-Inaba
Casra Ghazanfari
Alex Verity
Jung Woo Lee

January 17, 2025

1 Revision History

Date	Version	Notes
January 13, 2025	1.0	TA Feedback
January 15, 2025	1.1	Rev0

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
Sandlot	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

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 (?). We advocate a decomposition based on the principle of information hiding (?). 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 ?, 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 (?). 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 specific hardware on which the software is running.

AC2: The format of the log in process and data.

AC3: The algorithm used for the season scheduler.

AC4: The constraints on the season schedule.

AC5: The format of a team's season availability data.

AC6: How scheduling conflicts are to be resolved.

AC7: The format of the season schedule.

AC8: The format of the season standings.

AC9: The format of the create player/team account process and data.

AC10: The process of a player requesting to join a team.

AC11: The process of a commissioner making an admin command.

AC12: The process of creating an alert.

AC13: The process of rescheduling a game.

AC14: The method by which alerts are received.

[Anticipated changes relate to changes that would be made in requirements, design or implementation choices. They are not related to changes that are made at run-time, like the values of parameters. —SS]

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 system being a web application.

UC3: Player, team, and commissioner account structure.

UC4: The goal of the system to generate a season schedule.

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: Account Module

M3: Player Module

M4: Team Module

M5: Commissioner Module

M6: Account Structure Module

M7: Team Structure Module

M8: Schedule Structure Module

M9: Standings Structure Module

M10: Season Scheduler Module

M11: Reschedule Module

M12: Alerts Module

M13: Web Application Framework Module

M14: Database Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Account Module
	Player Module
	Team Module
	Commissioner Module
	Account Structure Module
	Team Structure Module
	Schedule Structure Module
	Standings Structure Module
	Reschedule Module
	Alerts Module
	Database Module
Software Decision Module	Season Scheduler Module
	Web Application Framework 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.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. —SS]

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. *Sandlot* means the module will be implemented by the Sandlot 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 Account Module (M2)

Secrets: The general functionality of all accounts.

Services: Contains functions to allow the user to create, delete, login, and edit their Sandlot account.

Implemented By: Sandlot

Type of Module: Abstract Object

7.2.2 Player Module (M3)

Secrets: The functionality of a player account.

Services: Contains functions to allow the user to join and leave a team.

Implemented By: Sandlot

Type of Module: Abstract Object

7.2.3 Team Module (M4)

Secrets: The functionality of a team account.

Services: Contains functions to allow the user to submit a game score and access to the Reschedule Module functionality.

Implemented By: Sandlot

Type of Module: Abstract Object

7.2.4 Commissioner Module (M5)

Secrets: The functionality of a commissioner account.

Services: Contains functions to allow the user to perform defined admin commands and send alerts.

Implemented By: Sandlot

Type of Module: Abstract Object

7.2.5 Account Structure Module (M6)

Secrets: The format of the account data.

Services: Defines a general account data structure for all accounts, inluding player, team, and commissioner accounts.

Implemented By: Sandlot

Type of Module: Abstract Data Type

7.2.6 Team Structure Module (M7)

Secrets: The format of the team account data.

Services: Defines a team account data structure that inheirits from the account structure module.

Implemented By: Sandlot

Type of Module: Abstract Data Type

7.2.7 Schedule Structure Module (M8)

Secrets: The format of the season schedule data.

Services: Defines a season schedule data structure.

Implemented By: Sandlot

Type of Module: Abstract Data Type

7.2.8 Standings Structure Module (M9)

Secrets: The format of the season standings data.

Services: Defines a season standings data structure.

Implemented By: Sandlot

Type of Module: Abstract Data Type

7.2.9 Reschedule Module (M11)

Secrets: The function of a team account to request and accept a game reschedule.

Services: Contains functions to allow the user to request and accept a game reschedule.

Implemented By: Sandlot

Type of Module: Library

7.2.10 Alerts Module (M12)

Secrets: The function of a commissioner account to create and send alerts.

Services: Contains functions to allow the user to create and send alerts.

Implemented By: Sandlot

Type of Module: Library

7.2.11 Database Module (M14)

Secrets: The function of storing player, team, commissioner, schedule and standings data.

Services: Defines and maintains a database that stores all relevant data to the Sandlot system. This includes account data for players, teams, and commissioners such as contact information and team player lists. It also includes full game lists for the schedule and game scores. The maintenance includes regular backups and data integrity checks.

Implemented By: Sandlot

Type of Module: Library

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 Season Scheduler Module (M10)

Secrets: The algorithm used to generate a season schedule.

Services: The algorithm generates a season schedule using each teams availability and a set of constraints as inputs and outputs a schedule of the form of the data structure defined in the schedule structure module.

Implemented By: Sandlot

7.3.2 Web Application Framework Module (M13)

Secrets: The framework needed to host the web application on which Sandlot will run.

Services: Defines a framework that will allow all other modules to be hosted on a web application including any other UI libraries or tools needed.

Implemented By: Sandlot

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**
FR-1	M9, M10,
FR-2	M4, M14
FR-3	M2, M6
FR-4	M2, M6
FR-5	M2, M6
FR-6	M5, M12
FR-7	M4, M5
FR-8	M3,
FR-9	M10
FR-10	M3, M11
FR-11	M3, M11
FR-12	M3, M11
FR-13	M3, M4
FR-14	M3, M4
FR-15	M3, M4
FR-16	M2
FR-17	M3, M4
FR-18	M5, M10
FR-19	M3, M5, M7
FR-20	M8, M10
FR-21	M7, M8, M10
FR-22	M5, M8, M10
FR-23	M7
FR-24	M2, M13

Table 2: Trace Between Functional Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M2, M6, M13, M14
AC3	M10
AC4	M8
AC5	M3, M13, M14
AC6	M8, M13
AC7	M8, M13
AC8	M9, M13
AC9	M3, M4, M13, M14
AC10	M3, M4, M13
AC11	M5, M13
AC12	M12, M13
AC13	M3, M11, M13
AC14	M12, M13

Table 3: Trace Between Anticipated Changes and Modules

9 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 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.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

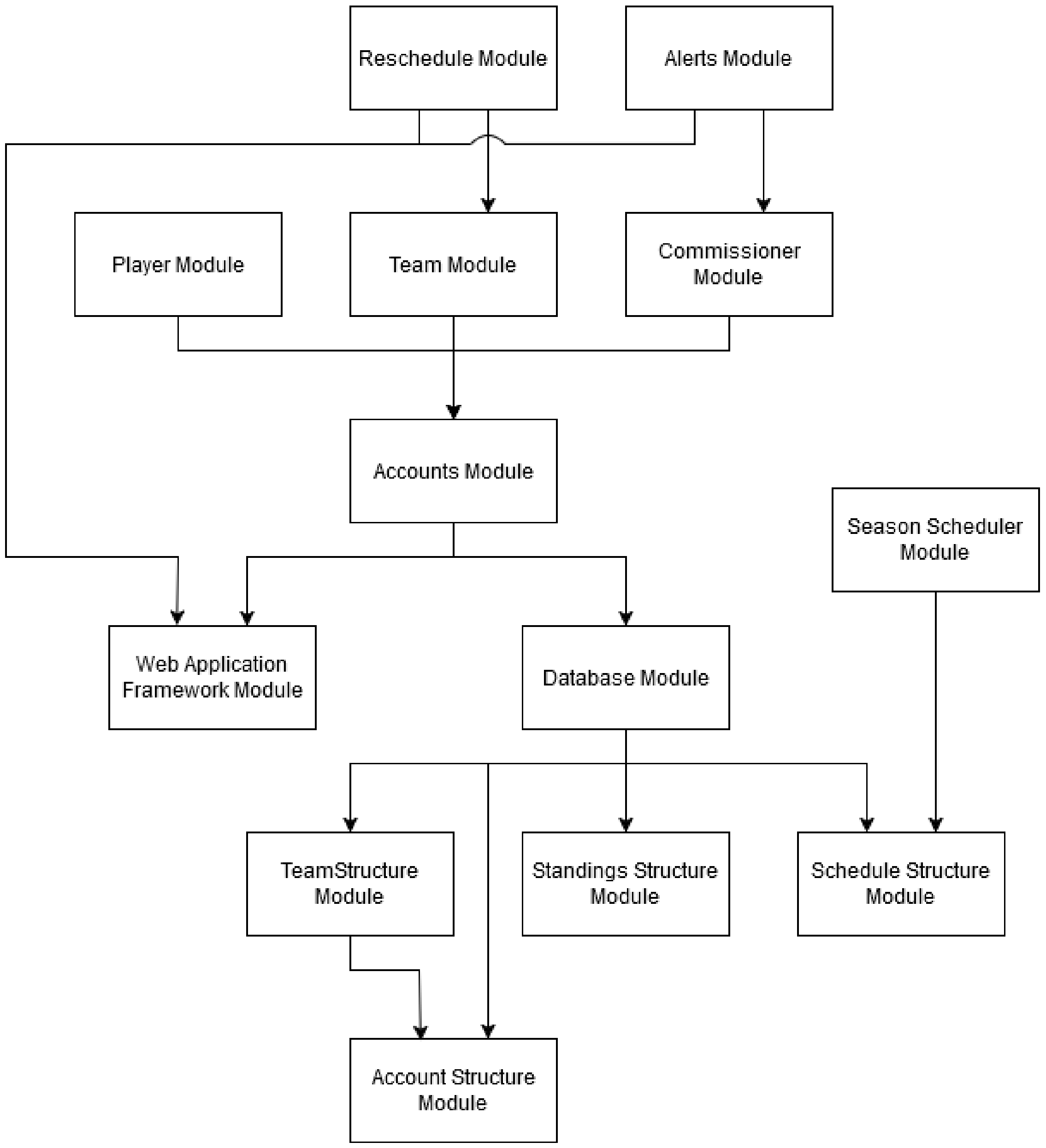

Figure 1: Use hierarchy among modules

10 User Interfaces

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]

11 Design of Communication Protocols

[If appropriate —SS]

12 Timeline

The implementation of modules for Rev 0 will follow the detailed timeline below. Daily stand-up meetings will be held to track each team member's progress and address any issues they may have run into, prior to the meeting. Each module will have unit tests to ensure successful functionality, and integration tests will be performed after integrating each module. Final testing will then validate the core functionality, making sure the performance of the system is able to accomplish each module's functions.

- **Week 1 (January 13-19)**: Initial Setup and Planning
  - Create Github issues that team members will assign themselves (All team members)
- **Week 2 (January 20-26)**: Core Module Implementation
  - Implement the account module (Name)
  - Implement the player, team, and commissioner modules (Name)
  - Setup database module for storing different data (Name)
  - Create the schedule and standings structure modules (Name)
  - Update season scheduler module to generate an optimal schedule (Alex Verity)
- **Week 3 (January 27-February 2)**: Remaining Modules and Integration
  - Implement alerts module (Name)
  - Implement web application framework module (Name)
  - Implement reschedule module (Name)
  - Initial integration testing (All team members)
- **Week 4 (February 3)**: Final Integration and Testing
  - Design and implement the user interface (Name)
  - Final integration and comprehensive testing (All team members)

[Schedule of tasks and who is responsible —SS]
[You can point to GitHub if this information is included there —SS]