Module Guide for Sandlot

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

January 11, 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 Login Module

M3: Account Creation Module

M4: Team Join Request Module

M5: Season Scheduler Module

M6: Schedule Display Module

M7: Standings Display Module

M8: Rescheduler Module

M9: Alerts Module

M10: Score Submission Module

M11: Commissioner Admin Module

M12: Web Application Framework Module

M13: User Interface Module

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Account Login Module Account Creation Module Team Join Request Module Schedule Display Module Standings Display Module Rescheduler Module Alerts Module Commissioner Admin Module User Interface 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 Login Module (M2)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.2 Account Creation Module (M3)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.3 Team Join Request Module (M4)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.4 Schedule Display Module (M6)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.5 Standings Display Module (M7)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.6 Rescheduler Module (M8)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.7 Alerts Module (M9)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.8 Commissioner Admin Module (M11)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.9 User Interface Module (M13)

Secrets: The format and structure of the account data.

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

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

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 (M5)

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: –

7.3.2 Web Application Framework Module (M12)

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: –

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	M1, M??, M??, M??
FR-2	M??, M??
FR-3	M??
FR-4	M??, M??
FR-5	M??, M??, M??, M??, M??, M??
FR-6	M??, M??, M??, M??, M??, M??
FR-7	M??, M??, M??, M??, M??
FR-8	M??, M??, M??, M??, M??
FR-9	M??
FR-10	M??, M??, M??
FR-11	M??, M??, M??, M??
FR-12	
FR-13	
FR-14	
FR-15	
FR-16	
FR-17	
FR-18	
FR-19	
FR-20	
FR-21	
FR-22	
FR-23	
FR-24	

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

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

[Schedule of tasks and who is responsible —SS]

[You can point to GitHub if this information is included there —SS]