Module Guide for Sayyara

Team 3, Tiny Coders
Arkin Modi
Joy Xiao
Leon So
Timothy Choy

January 15, 2023

1 Revision History

Table 1: Revision History

Date	Developer(s)	Change
December 28, 2022	Arkin Modi	Create Revision History
January 3, 2023	Arkin Modi	Add Module Hierarchy
January 8, 2023	Arkin Modi	Add Use Hierarchy Between Modules Diagram
January 10, 2023	Arkin Modi	Add Module Decompositions
January 12, 2023	Leon So	Anticipated Changes & Unlikely Changes
January 15, 2023	Arkin Modi	Merge Shop Profile & Shop Lookup Module

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
ORM	Object-relational Mapping
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Sayyara	Explanation of program name
UC	Unlikely Change
CRUD	Create, Read, Update, and Delete
[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 (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 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 layed out by Parnas et al. (1984), 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, 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 initial input data.

AC3: The format of server-side responses

AC4: The styling of UI forms and components

AC5: The formatting of UI forms and components

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 application will be built-upon the Next.js framework

UC2: The application will use Prisma for ORM

UC3: There will always be a source of input data external to the system

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: Database Driver Module

M3: Users Module

M4: Quotes Module

M5: Appointments Module

M6: Work Orders Module

M7: Employee Management Module

M8: Services Module

M9: Shop Module

Note that M1 is a commonly used module and is already implemented by the operating system. It will not be reimplemented.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Users Module Quotes Module Appointments Module Work Orders Module Employee Management Module Services Module Shop Module
Software Decision Module	Database Driver Module

Table 2: 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 3.

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. *Sayyara* means the module will be implemented by the Sayyara 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

7.2.1 User Module (M3)

Secrets: The format and structure of user data objects.

Services: Provide CRUD operations for user objects, and the ability to verify a user's authorization.

Implemented By: Sayyara

7.2.2 Quotes Module (M4)

Secrets: The format and structure of quote and chat message data objects.

Services: Provide CRUD operations for quote and chat message objects.

Implemented By: Sayyara

7.2.3 Appointments Module (M5)

Secrets: The format and structure of appointment data objects.

Services: Provide CRUD operations for appointment objects, the ability to verify if an appointment is acceptable (i.e., is possible to fulfill), and the ability to accept an appointment and reject all conflicting appointments.

Implemented By: Sayyara

7.2.4 Work Orders Module (M6)

Secrets: The format and structure of work order data objects.

Services: Provide CRUD operations for work order objects.

Implemented By: Sayyara

7.2.5 Employee Management Module (M7)

Secrets: –

Services: Provide CRUD operations for managing employee's relation with shop.

Implemented By: Sayyara

7.2.6 Services Module (M8)

Secrets: The format and structure of service data objects.

Services: Provide CRUD operations for service objects.

Implemented By: Sayyara

7.2.7 Shop Module (M9)

Secrets: The format and structure of shop data objects.

Services: Provides CRUD operations for shop objects and the ability to query for a shop by distance, postal code and/or shop name.

Implemented By: Sayyara

7.3 Software Decision Module

7.3.1 Database Driver Module (M2)

Secrets: The data structures and algorithms for representing the database's schema and communicating with the database.

Services: Provides a driver to interface with the database

Implemented By: Prisma

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	MM1, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	**Modules**
ACAC1	MM1
AC??	M??
AC??	M??
AC??	M??
AC??	M??
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.

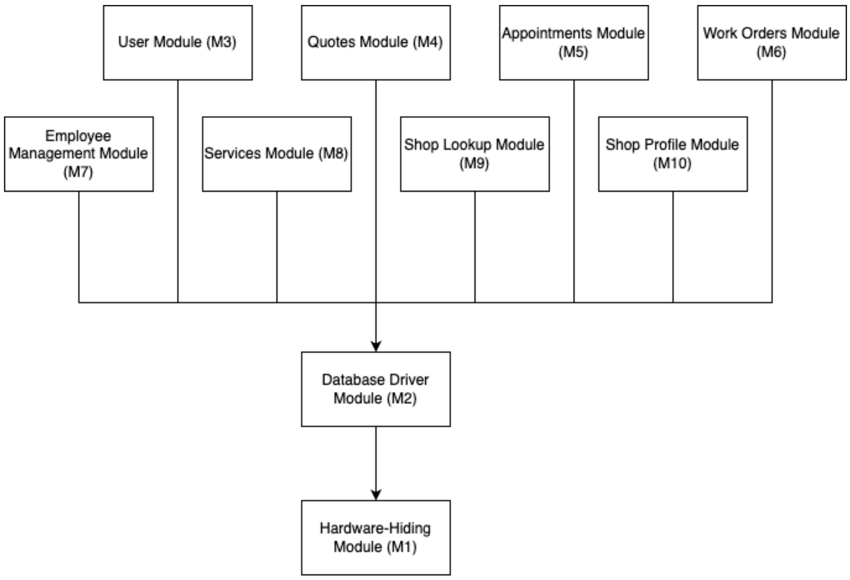

Figure 1: Use hierarchy among modules

10 Bibliography

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

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.

11 Appendix

[Extra information if required —SS]