Module Guide for GMM-GM

Kim Ying WONG

March 19, 2024

1 Revision History

Date	Version	Notes
13 Mar 2024	1.0	Notes
Date 2	1.1	Notes

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
	Explanation of program name
UC	Unlikely Change
etc.	...

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 the input parameters.

AC4: The constraints on the input parameter.

AC5: The format of the final output data.

AC6: How the overall control of the calculations is orchestrated.

AC7: Algorithm on how to initiate the model hyper-parameters

AC8: The specification parameters may be implemented as input parameters instead of fixed parameters.

AC9: The implementation for the sequence (array) data structure.

AC10: The implementation of plotting data.

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: Algorithm on training the model

UC3: The goal of the model is to do clustering on the dataset.

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: Input Module

M3: Loading Data and Parameters Module

M4: Model Hyper-parameters Initiation Module

M5: Model Training Module

M6: Model Predict Module

M7: Control Module

M8: Parameters Specification Module

M9: Sequence Data Structure Module

M10: GMM Model Module

M11: Plotting Module

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.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module
	Loading Data and Parameters Module
	Model Hyper-parameters Initiation Module
	Model Training Module
	Model Predict Module
	Control Module
	Parameters Specification Module
Software Decision Module	Sequence Data Structure Module
	GMM Model Module
	Plotting Module

Table 1: Module Hierarchy

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. means the module will be implemented by 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.

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 Input Module (M2)

Secrets: Implementation on reading CSV file and store the dataset in the program and wait for converting into the data structure inside our class GMM model.

Services: Gets dataset from user and stores input and verifies that the CSV is available for our program.

Implemented By: C++

Type of Module: Record

7.2.2 Loading Input Data and Parameters Module (M3)

Secrets: The data structure for Dataset and input parameters, how the values are input and how the values are verified. The load and verify secrets are isolated to their own access programs (like submodules).

Services: Gets input from user (including material properties, processing conditions, and numerical parameters),and transform the dataset to our model input and verifies that the input parameters comply with physical and software constraints. Throws an error if a parameter violates a physical constraint. Throws a warning if a parameter violates a software constraint. Stored parameters can be read individually, but write access is only to redefine the entire set of inputs.

Implemented By: GMM-EM

Type of Module: ADT

7.2.3 Model Hyper-parameters Initiation Module (M4)

Secrets: The algorithm to initiate the hyper-parameters of the model. How to initiate and verify the hyper-paramaters in our model.

Services: With the dataset and number of cluster we needed, the module generate the internal hyper-parameters that will be used to training and predict module. The module adopt either random Initiation or K-mean algorithm Initiation.

Implemented By: GMM-EM

7.2.4 Model Training Module (M5)

Secrets: The algorithm on updating the model's hyper-parameter using the initiated hyper-parameters and data stored in the model.

Services: Define how the hyper-parameters will be updated and when will the update terminates (whether it reaches maximum update iteration or the convergence threshold).

Implemented By: GMM-EM

7.2.5 Model Predict Module (M6)

Secrets: Predict and verify the clustering result.

Services: Output the result in given environment variables (csv file or terminal).

Implemented By: GMM-EM

7.2.6 Control Module (M7)

Secrets: The algorithm for coordinating the running of the program.

Services: Provide the main program

Implemented By: GMM-EM

7.2.7 Parameters Specification Module (M8)

Secrets: The values of the specification parameters.

Services: Define the specific parameters (convergence threshold and maximum iteration) for training module.

Implemented By: GMM-EM

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, statistical fact, 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 Sequence Data Structure Module (M9)

Secrets: The data structure for a sequence data type.

Services: Provide multi-dimensional array manipulation, including building , slicing , accessing an array. Provide mathematical manipulation on multi-dimensional array including inverse, transpose , determinant , etc.

Implemented By: C++

7.3.2 GMM Model Module (M10)

Secrets: The abstract object define our GMM model.

Services: Provide an class template for the GMM model.

Type: ADT

Implemented By: GMM-EM

7.3.3 Plotting Module (M11)

Secrets: The data structure and algorithm for plotting data graphically

Services: Provide plot function

Implemented By: C++

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes. The requirement should refer to the SRS document.

Req.	Modules
R1	M2, M3
R2	M4, M5
R3	M10 , M7
R4	M6, M11

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M3
AC5	M6
AC7	M4
AC9	M9
AC6	M7
AC10	M11

Table 3: 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.

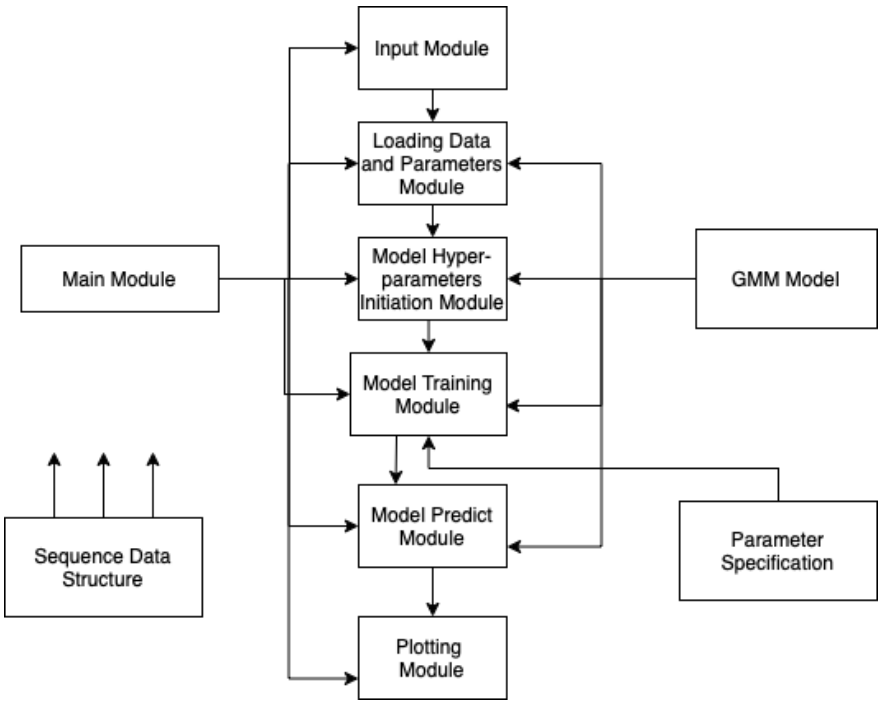

Figure 1: Use hierarchy among modules

10 User Interfaces

Since our software is a library. The user will run their program with terminal or command line.

References

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.