Module Guide for OAR

Hunter Ceranic

March 8, 2024

1 Revision History

Date	Version	Notes
March 8, 2024	1.0	Initial Revision

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
GUI	Graphical User Interface
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
OAR	Optical Alphabet Recognition
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, 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 constraints on the initial input data.

AC4: The constraints on the initial input parameters.

AC5: How the input data is pre-processed.

AC6: How the input classification model is generated.

AC7: How the input classification is calculated.

AC8: How the overall flow and control of the OAR program is organized.

AC9: The format of the output data.

AC10: The constraints on the output results.

AC11: The implementation of the GUI.

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 source of the input is always external to the software.

UC3: The outputs are displayed on the output device.

UC4: The goal of the system is to classify characters in an image.

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: Application Control Module

M3: Output Module

M4: Input Data Read Module

M5: Input Classifier Module

M6: OAR Model Data Module

M7: OAR Model Equations Module

M8: OAR Model Training Module

M9: OAR Model Testing Module

M10: Confusion Matrix Module

M11: Input Processing Module

M12: Graphical User Interface

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.

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

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Application Control Output Module Input Data Read Module Input Classifier Module OAR Model Data Module OAR Model Equations Module OAR Model Training Module OAR Model Testing Module
Software Decision Module	Confusion Matrix Module Input Processing Module Graphical User Interface

Table 1: Module Hierarchy

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 Application Control Module (M2)

Secrets: The algorithm for the overall flow of the program.

Services: Provides the main program

Implemented By: OAR

Type of Module: Abstract Data Type

7.2.2 Output Module (M3)

Secrets: The format and structure of the output data.

Services: Outputs the results of the classification, including the predicted label and the confidence level.

Implemented By: OAR

Type of Module: Abstract Data Type

7.2.3 Input Data Read Module (M4)

Secrets: The format and structure of the input data.

Services: Prompts user for input image. Converts the input image data into the matrix data structure used OAR.

Implemented By: OAR

Type of Module: Abstract Data Type

7.2.4 Input Classifier Module (M5)

Secrets: The algorithm for classification given the input image and the OAR classification model.

Services: Defines the algorithm used to determine the predicted label of an input image.

Implemented By: OAR

Type of Module: Abstract Data Type

7.2.5 OAR Model Data Module (M6)

Secrets: The weights and biases that make up the OAR classification model for the 26 labels.

Services: Stores the data of the highest performance model trained by OAR. The values can be read from the file and are used by the input classifier module M5. This module knows how many parameters it stores.

Implemented By: OAR

Type of Module: Record

7.2.6 OAR Model Equations Module (M7)

Secrets: The logistic regression equations used to train and test the classification model.

Services: Provides the methods to train the weights and biases of a classification model and to use said classification model. These are the equations referred to as Instance Models in the SRS.

Implemented By: OAR

Type of Module: Library

7.2.7 OAR Model Training Module (M8)

Secrets: The algorithm for training the classification model.

Services: Defines the algorithm used to train the model's weights and biases using images from the EMNIST dataset for every label.

Implemented By: OAR

Type of Module: Abstract Data Type

7.2.8 OAR Model Testing Module (M9)

Secrets: The algorithm for testing the performance of the classification model.

Services: Defines the algorithm used to test the model's performance based on images from the EMNIST dataset. The output of this module is stored in the OAR model data module M6.

Implemented By: OAR

Type of Module: Abstract Data Type

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 Confusion Matrix Module (M10)

Secrets: The algorithms to generate a confusion matrix of a classifier model.

Services: Provides the methods to take the outputs of testing the classification model and return the metrics that describe that models performance, in both graphical and numerical forms.

Implemented By: scikit-learn

Type of Module: Library

7.3.2 Input Processing Module (M11)

Secrets: The algorithms to process the input image to transform it into the desired data structure.

Services: Provides the methods to perform pre-proccessing operations such as normalization, and converting an image to grayscale, used by the input data read module M4.

Implemented By: OpenCV and scikit-learn

Type of Module: Library

7.3.3 Graphical User Interface (M12)

Secrets: User interaction event handling, user input controls, states, data formats (such as text-boxes, buttons, file dialogs), visual layout and styling (eg. colours or sizing)

Services: Sets up a visual interface for the user to see and interact with, handles user and GUI control events, updates the visual elements representing the state of the application, and sends messages back to the application control module M2 to update the application state accordingly.

Implemented By: OAR

Type of Module: Abstract Data Type

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	M1, M4, M12
R2	M11
R3	M6, M7, M8, M9, M10
R4	M3, M5, M12
R5	M3, M5, M12

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M4, M11
AC3	M4, M11
AC4	M4, M11
AC5	M11
AC6	M6, M7, M8, M9, M10
AC7	M5
AC8	M2
AC9	M3
AC10	M3, M5
AC11	M12

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.

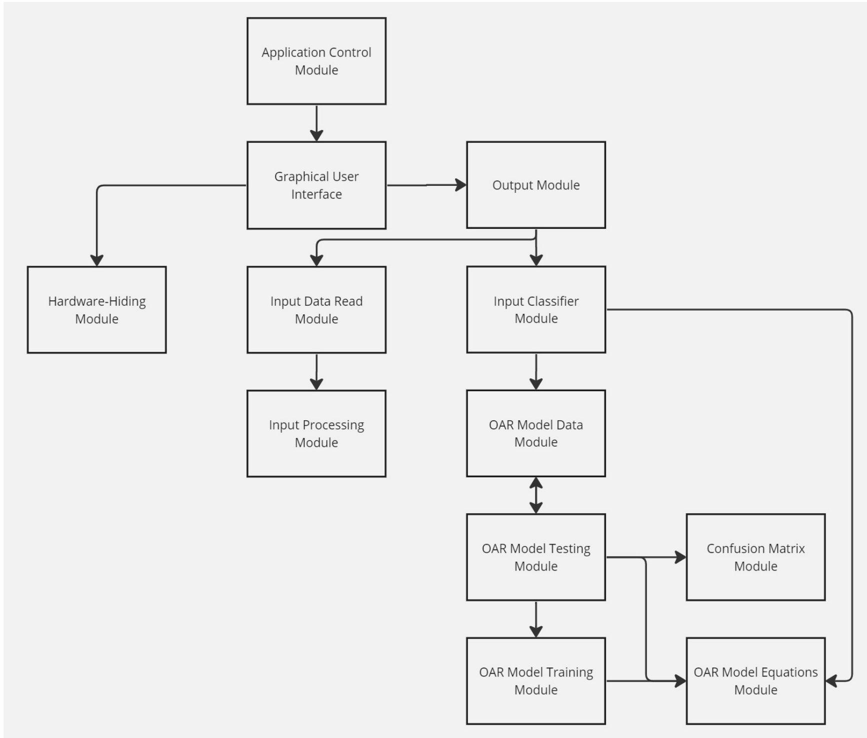

Figure 1: Use hierarchy among modules

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.