Module Guide for OCRacle

Phillip Tran

March 17, 2025

1 Revision History

Date	Version	Notes
March 5, 2025	1.0	Initial Document Creation

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
OCRacle	Name of the Project
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 control-flow of the application.

AC3: The accepted image formats of the input data.

AC4: The training methods used for the model.

AC5: The testing methods used for the model.

AC6: The image preprocessing techniques used before classification.

AC7: The model used for image classification.

AC8: The output format of the model.

AC9: The specific form of the graphical user interface.

AC10: The performance metrics used to evaluate the model.

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 input data will always be in the form of an image.

UC2: The results of the model output will always be displayed to the graphical user interface.

UC3: The program will only be used for classifying a single Latin alphabet character 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 Module

M3: Input Format Module

M4: Model Training Module

M5: Model Testing Module

M6: Image Preprocessing Module

M7: Prediction Model Module

M8: Model Output Module

M9: Graphical User Interface Module

M10: Performance Metrics Module

M11: Constants Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Module Model Output Module Model Training Module Model Testing Module Prediction Model Module Application Module
Software Decision Module	Image Preprocessing Module Performance Metrics Module Graphical User Interface 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.

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. *OCRacle* means the module will be implemented by the OCRacle 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 Application Module (M2)

Secrets: The data and execution flow of the application.

Services: Orchestrates the execution of the system. This includes coordinating the input M3, preprocessing M6, model training M4, model testing M5, model classification M7, and output M8 modules. It also displays the relevant information to the user in the graphical user interface M9.

Implemented By: OCRacle

Type of Module: Abstract Data Type

7.2.2 Input Format Module (M3)

Secrets: The format and structure of the input data.

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

Implemented By: OCRacle

Type of Module: Library

7.2.3 Model Output Module (M8)

Secrets: The format and structure of the output data.

Services: Converts the output data from the model into the format required by the graphical user interface.

Implemented By: OCRacle

Type of Module: Library

7.2.4 Model Training Module (M4)

Secrets: The methods for training the model.

Services: Trains the model using the input data via a test-train split.

Implemented By: OCRacle

Type of Module: Abstract Data Type

7.2.5 Model Testing Module (M5)

Secrets: The methods for testing the model.

Services: Tests the model using both data that the model was trained on and data that the model was not trained on.

Implemented By: OCRacle

Type of Module: Abstract Data Type

7.2.6 Prediction Model Module (M7)

Secrets: The model weights and architecture.

Services: Given an input image, provides an output classification, which is a probability distribution over the possible classes.

Implemented By: OCRacle

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 Image Preprocessing Module (M6)

Secrets: The image preprocessing techniques used before classification.

Services: Used by M3 to preprocess the input image data before sending it to M7 for classification.

Implemented By: Pandas, OpenCV, NumPy

Type of Module: Library

7.3.2 Performance Metrics Module (M10)

Secrets: The performance metrics used to evaluate the model.

Services: Given the model output, ground truth, and performance of the previous OAR project, calculates the performance metrics.

Implemented By: scikit-learn, seaborn, OAR

Type of Module: Library

7.3.3 Graphical User Interface Module (M9)

Secrets: User event handling, display of input data, display of output data, error messages, and other user interface functions.

Services: Provides a Graphical User Interface to the user, allowing them to interact with the system.

Implemented By: Python Notebook

Type of Module: Library

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, M3, M9, M2
R2	M6, M3
R3	M4, M5, M7, M10
R4	M8, M7 M9
R5	M8, M7, M9, M2

Table 2: Trace Between Requirements and Modules

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

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.

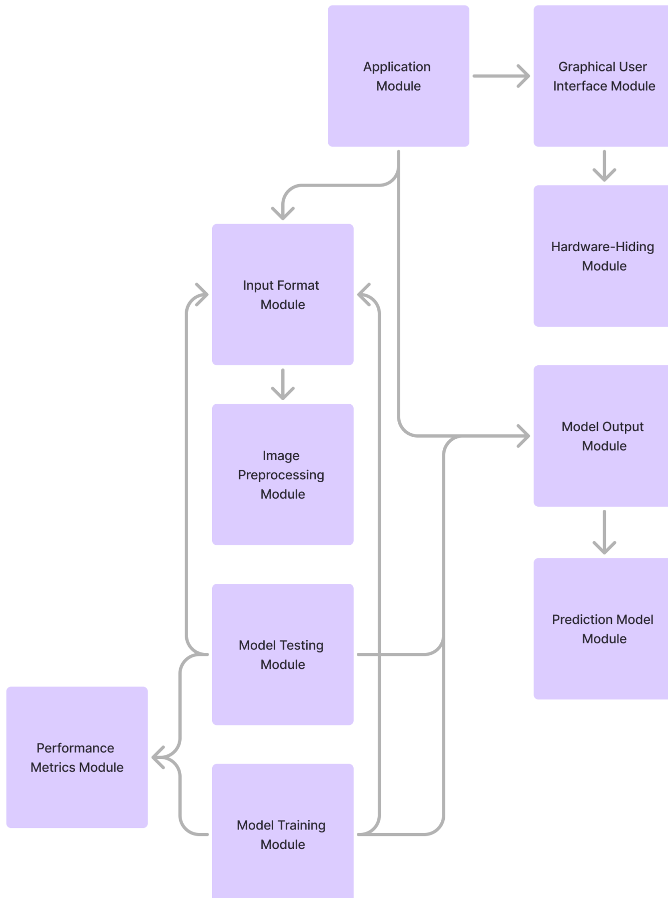

Figure 1: Use hierarchy among modules

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.