Module Guide for Utrition

Team 16, Durum Wheat Semolina
Alexander Moica
Yasmine Jolly
Jeffrey Wang
Jack Theriault
Catherine Chen
Justina Srebrnjak

April 5, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	Initial Document
March 7, 2023	1.1	Added New Modules - VNV Report
April 5, 2023	1.2	Final Document Revision

2 Reference Material

This section records information for easy reference.

2.1 Relevant Documentation

This document references multiple other documents that are listed below:

- SRS, Semolina (2022)

2.2 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
Utrition	Application name
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 laid 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 identified below are small modifications to already existing modules. Ideally, implementing one of the anticipated changes will only require changing a single module. The Utrition team has specifically taken designing for change into account when creating modules.

AC1: The specific hardware on which the software is running.

AC2: The format of the initial input data.

AC3: The graphics being displayed on the user interface.

AC4: The nutritional data details of a food to be displayed.

AC5: The dataset used to train the image classification model.

4.2 Unlikely Changes

Unlikely changes to Utrition are listed below. These modifications are not expected to be implemented based on their unnecessary nature and difficulty of implementation. If any of these changes are implemented, 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: There will always be a source of input data external to the software.

UC3: Local storage of the user's past nutritional history.

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: Application Path Module

M2: Home Page Module

M3: Upload Page Module

M4: Profile Page Module

M5: BMI Page Module

M6: Settings Page Module

M7: Upload Container Module

M8: Image Upload Module

M9: Text Upload Module

M10: Voice Upload Module

M11: Navigation Bar Module

M12: Input Pre-Processing Module

M13: Training Dataset Module

M14: Image Classification Module

M15: Nutritional Data Retriever Module

M16: Profile Data Calculation Module

Level 1	Level 2
Hardware-Hiding Module	N/A
Behaviour-Hiding Module	Application Path Module
	Home Page Module
	Upload Page Module
	Profile Page Module
	BMI Page Module
	Settings Page Module
	Upload Container Module
	Image Upload Module
	Text Upload Module
	Voice Upload Module
	Navigation Bar Module
Software Decision Module	Input Pre-Processing Module
	Training Dataset Module
	Image Classification Module
	Nutritional Data Retriever Module
	Profile Data Calculation 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. *Utrition* means the module will be implemented by the Utrition 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

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: Utrition

7.2.1 Application Path Module (M1)

Secrets: The structure of the application.

Services: Imports all used React component to structure the basic interface of each page based on the path of the website.

Implemented By: Utrition

Type of Module: Record

7.2.2 Home Page Module (M2)

Secrets: Displays all information needed on the Home Page.

Services: Imports components relevant to the Home Page to organize all components on the page.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.3 Upload Page Module (M6)

Secrets: Displays all objects needed on the Upload Page.

Services: Imports components required to upload data, and organizes all components within the page.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.4 Profile Page Module (M3)

Secrets: Displays all information needed on the Profile Page.

Services: Imports components relevant to the user's profile statistics to organize all components on the Profile Page.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.5 BMI Page Module (M??)

Secrets: Displays all information needed on the BMI Page.

Services: Provides interface for the user to manage their settings, and as well relevant information figures.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.6 Settings Page Module (M??)

Secrets: Displays all information needed on the Settings Page.

Services: Imports components relevant to user parameters and provides a view for the user's settings.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.7 Upload Container Module (M??)

Secrets: Provides interface to allow the user to log their meal by uploading an image.

Services: Imports components relevant to image upload, text upload, and voice upload to organize all components on the page.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.8 Image Upload Module (M7)

Secrets: Provides interface to allow the user to log their meal by uploading an image.

Services: Formats input and queries backend regarding the nutritional contents of the classified food.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.9 Text Upload Module (M??)

Secrets: Provides interface to allow the user to log their meal by manual, textual entry of their meals.

Services: Formats input and queries backend regarding the nutritional contents of the specified food.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.10 Voice Upload Module (M9)

Secrets: Provides interface to allow the user to log their meal by verbal communication of their meals.

Services: Formats input and queries backend regarding the nutritional contents of the classified food.

Implemented By: Utrition

Type of Module: Abstract Object

7.2.11 Navigation Bar Module (M10)

Secrets: Provides interface to allow the user to navigate through various pages of the application.

Services: Changes the path of the website to render a new page based on a user action.

Implemented By: Utrition

Type of Module: Abstract Object

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: Utrition

7.3.1 Input Pre-Processing Module (M12)

Secrets: Converts an image path into a pixel array that can be used for machine learning processes in the Image Classification Module.

Services: Converts the input file path to a pixel array formatted as a multidimensional array of integers through accessing the image at the file path and concatenating its pixel values.

Implemented By: Utrition

Type of Module: Library

7.3.2 Training Dataset Module (M13)

Secrets: Creates a dictionary of known food identifiers for use by machine learning processes in the Image Classification Module.

Services: Produces a dictionary of image labels and classes for use in the machine learning code contained in the Image Classification Module. This is done by parsing the training image, test image, and metadata files and appending the pixel array of the input image.

Implemented By: Utrition

Type of Module: Library

7.3.3 Image Classification Module (M14)

Secrets: Identifies the food present in a user image.

Services: Converts the input image pixel array to a string classification of the food in the image through a machine learning model.

Implemented By: Utrition

Type of Module: Library

7.3.4 Nutritional Data Retriever Module (M15)

Secrets: Retrieves nutritional information for an inputted food item.

Services: Using the input food item, a request is made to NutritionixAPI to fetch the nutritional data of this item.

Implemented By: Utrition

Type of Module: Library

7.3.5 Profile Data Calculation Module (M??)

Secrets: Stores all past user inputted food items and their nutritional facts.

Services: Every user inputted food item and their corresponding nutritional facts are stored in a sequence of FoodEntry objects.

Implemented By: Utrition

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. The requirements include functional and non-functional requirements that are outlined in the SRS.

Req.	**Modules**
FR1	M8
FR2	M12, M13, M14
FR3	M18, M9, M10
FR4	M15
FR5	M15, M16
FR6	M4, M16
FR7	M8, M9, M10
FR8	M9
FR9	M9
FR10	M10
FR11	M10
FR12	M10
FR13	M10
FR14	M16
FR15	M7
FR16	M4, M16
FR17	M4, M16
FR18	M4, M16
FR19	M4, M16
FR20	M1, M11
FR21	M4, M16
FR22	M6
FR23	M16
FR24	M5, M16
FR25	M16
FR26	M4, M16
FR27	M4, M16
FR28	M4

LF1	M2, M3, M4, M5, M6, M7
UH1	M11
UH2	M4, M16
UH5	M3, M4
PR2	M12, M13, M14
PR3	M15
MS3	M8
SR3	M16

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC2	M7, M8, M9, M10, M12, M15
AC3	M2, M3, M4, M5, M6, M7
AC4	M15
AC5	M13

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.

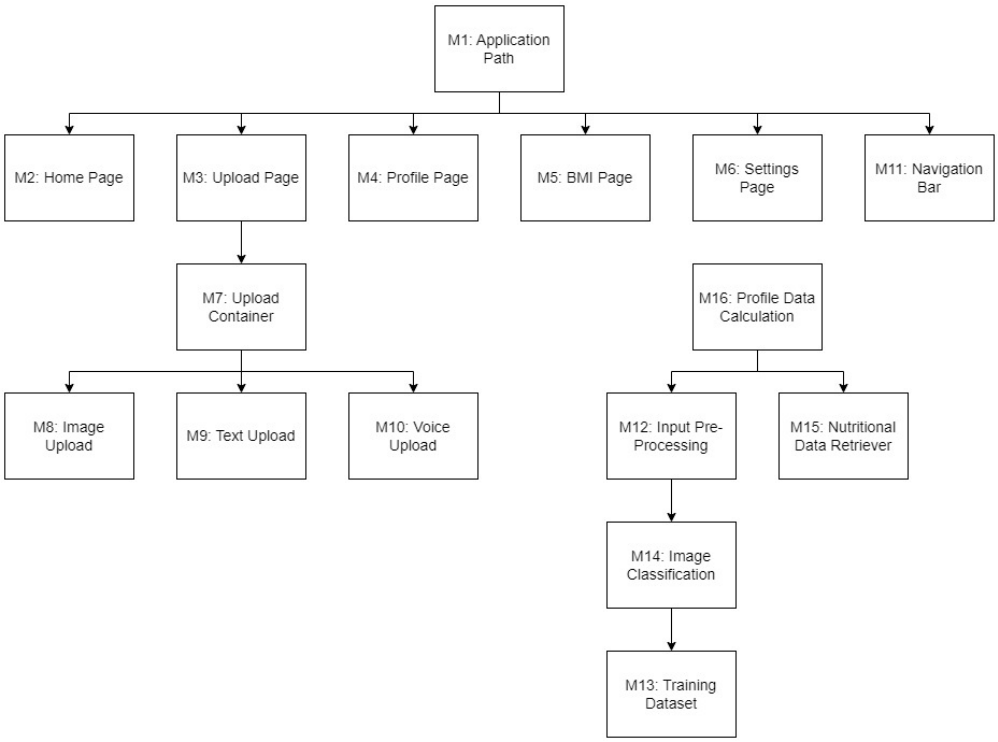

Figure 1: Use Hierarchy Among Modules

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.

Durum Wheat Semolina. System requirements specification. https://github.com/jeff-rey-wang/utrition/blob/39b1a9bda7ec90db43221e17b035e57b7fa29650/docs/SRS/SRS.pdf, 2022.