Module Guide for Retinal Vessel Segmentation System (RVSS)

Xinyu Ma

March 18, 2024

1 Revision History

Date	**Version**	**Notes**
03/13/2024	1.0	Initial Release

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
CNN	Convolution Neural Network
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
RVSS	Retinal Vessel Segmentation System
SRS	Software Requirements Specification
UC	Unlikely Change

For complete symbols, abbreviations and acronyms used within the system, please refer the section 1.3 in SRS document.

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 (3). We advocate a decomposition based on the principle of information hiding (1). 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 **(author?)** (3), 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 (3). 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. This "design for change" approach ensures that when updates or modifications are needed, they can be managed within the scope of a single module, minimizing the impact on the overall system. Below are the anticipated changes that have been considered in the development of RVSS, along with the modules that would encapsulate these change.

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

- Encapsulated in: Hardware-Hiding Module
- Rationale: Adds portability to the software since not everyone has the same hardware to run it on and allows for hardware upgrades or changes without affecting the rest of the system..

AC2: The image preprocessing algorithms.New or improved algorithms for image normalization, denoising, and contrast enhancement may become available.

- Encapsulated in: Image Preprocessing Module
- Rationale: By isolating image preprocessing algorithms within this module, updates algorithms can be made without affecting the segmentation process or other system components.

AC3: The image segmentation algorithms. Advances in image segmentation algorithms, including machine learning models, could offer enhanced accuracy or efficiency.

- Encapsulated in: Image Segmentation Module
- Rationale: Encapsulating segmentation algorithms within its module allows for straightforward integration of new algorithms or updates to existing ones, facilitating continuous improvement in segmentation performance.

AC4: Reporting Formats and Contents

- Encapsulated in: Report Generation Module

- Rationale: Encapsulating report generation within its module allows for modifications to report formats, contents, or additional data visualization techniques without affecting other system components.

AC5: Image data storage methods.

- Encapsulated in: Image Management Module
- Rationale: Encapsulating data storage allows for changes in underlying storage solutions or adaptation to new data management practices without impacting other system functionalities.

AC6: Updates the user interface design principles or the introduction of new user interaction models.

- Encapsulated in: User Interface Module
- Rationale: User Interface trends change over time and encapsulating user interface within a dedicated module allows for updates and redesigns to be carried out with minimal disruption.

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: Fundamental image input formats: The decision to support standard fundus image formats (e.g., JPEG, PNG) is unlikely to change. Expanding to different or proprietary image formats would necessitate considerable adjustments in preprocessing algorithms and data management methods.

UC2: Primary programming language: The choice of programming language (i.e., Python) for the RVSS development has implications for performance optimization, library availability and compatibility. A change in the primary programming language is considered unlikely due to the extensive effort required to rewrite and retest the system.

UC3: The core system architecture: The modular architecture of RVSS, including the division of functionalities into specific modules, is a foundational design choice unlikely to undergo radical changes.

UC4: The goal of RVSS is to generate vessel segmentation maps of fundus images, which is unlikely to change.

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 Upload and Validate Module

M3: Image Management Module

M4: Image Preprocessing Module

M5: Image Segmentation Module

M6: Output Format Module

M7: Report Generation Module

M8: User Interface Module

M9: Main Function Module

M10: Algorithm Optimization Module

M11: Plotting Result Module

M12: Logging Module

M13: Image patching Module

M14: Image Dataset Reader Module

M15: Network Model Reader 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.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Upload and Validate Module Image Management Module Image Preprocessing Module Image Segmentation Module Output Format Module Report Generation Module User Interface Module Main Function Module
Software Decision Module	Algorithm Optimization Module Plotting Result Module Logging Module Image patching Module Image Dataset Reader Module Network Model Reader Module

Table 1: Module Hierarchy

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **(author?)** (3). 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. *RVSS* means the module will be implemented by the RVSS 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, algorithm and underlying hardware 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 Upload and Validate Module (M2)

Secrets: The processes and validations involved in uploading images within the system.

Services: Provides functionalities for users to upload fundus images and ensures that uploaded images meet specified format and quality criteria.

Implemented By: RVSS

Type of Module: Abstract Object

7.2.2 Image Management Module (M3)

Secrets: The processes and validations involved in storing and managing fundus images within the system.

Services: Provides functionalities for users to manage stored images, and retrieve them for processing.

Implemented By: RVSS

Type of Module: Record

7.2.3 Image Preprocessing Module (M4)

Secrets: The algorithms for image normalization, denoising, and contrast enhancement.

Services: Prepares images for segmentation by improving their quality and standardizing their format.

Implemented By: RVSS

Type of Module: Library

7.2.4 Image Segmentation Module (M5)

Secrets: The specific machine learning algorithms and techniques used to identify and delineate retinal vessels from the preprocessed fundus images.

Services: Segments retinal vessels from the preprocessed fundus images.

Implemented By: RVSS

Type of Module: Library

7.2.5 Output Format Module (M6)

Secrets: The format and structure of the output.

Services: Outputs the results of the segmentation, including the input image and the corresponding image with only blood vessels (white pixels) and background (black pixels).

Implemented By: RVSS

Type of Module: Abstract Object

7.2.6 Report Generation Module (M7)

Secrets: The format, layout, and content inclusion criteria for generating reports.

Services: Compiles the input image, the segmentation results and analyses into structured reports.

Implemented By: RVSS

Type of Module: Abstract Object

7.2.7 User Interface Module (M8)

Secrets: The design and workflow of the graphical user interface.

Services: Provides an interface for users to interact with the system, from uploading fundus images to viewing reports.

Implemented By: RVSS

Type of Module: Abstract Object

7.2.8 Main Function Module (M9)

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

Services: Provides the main function of the system which uses the services that are provided by other modules to upload fundus images, preprocess the original fundus images, segment retinal vessels from the preprocessed fundus images, and generate output images and report.

Implemented By: RVSS

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 Algorithm Optimization Module(M10)

Secrets: Specific optimization techniques and algorithms used to enhance the performance of image processing and segmentation tasks.

Services: Provides optimized processing for image preprocessing and segmentation, reducing computation time and resource usage.

Implemented By: PyTorch

Type of Module: Library

7.3.2 Plotting Result Module (M11)

Secrets: Plots the data and algorithms graphically.

Services: Provides function that can accept the output of the segmentation algorithm and plot the results using in the output format module and the report generation module.

Implemented By: torchvision,tensorboard

Type of Module: Library

7.3.3 Logging Module (M12)

Secrets: The specific algorithms and configurations used for log message formatting, storage, categorization by severity levels, and handling of log data persistence and retrieval without impacting system performance or resource utilization.

Services: Delivers essential system-wide services that enhance the maintainability and operability of the system, including logging services for debugging and audit trails.

Implemented By: RVSS or OS (for standard functionalities provided by the operating system or python libraries)

Type of Module: Library

7.3.4 Image patching Module (M13)

Secrets: The methods to divide the image into multiple small patches.

Services: Provides data for the process of neural network learning and training.

Implemented By: PyTorch

Type of Module: Library

7.3.5 Image Dataset Reader Module(M14)

Secrets: The algorithm to read fundus training dataset and test dataset.

Services: Provides images that can be used for model training and testing.

Implemented By: PyTorch

Type of Module: Library

7.3.6 Network Model Reader Module(M15)

Secrets: Load the model parameters of the pre-trained neural network.

Services: Provide model parameters for the testing phase

Implemented By: PyTorch

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	M2, M3, M8, M9
R3	M3, M4, M9, M12, M14
R4	M1, M5, M9, M10, M12, M13, M14, M15
R5	M6, M7, M8, M9,M11, M12

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M4
AC3	M5
AC4	M7
AC5	M3
AC6	M8

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (2) 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.

Assume that the model is mainly used for application rather than development (training a own model to use), so before using the model, we will pre-train a model, and then call the main function for user use.

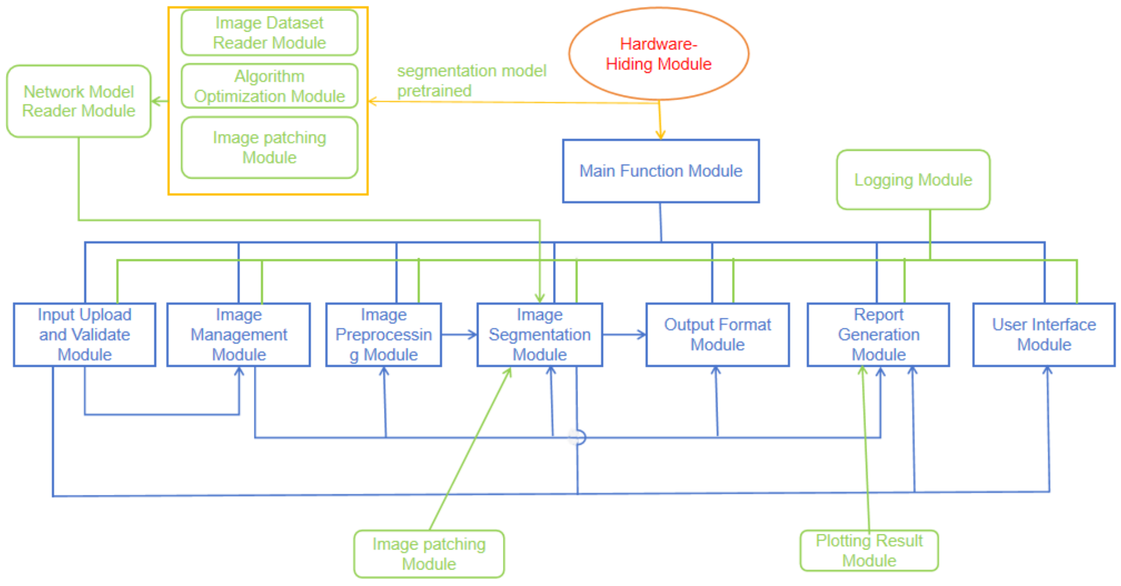

Figure 1: Use Hierarchy Among Modules

References

[1] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[2] David Lorge Parnas. Designing software for ease of extension and contraction. *IEEE transactions on software engineering*, (2):128–138, 1979.

[3] David Lorge Parnas, Paul C Clements, and David M Weiss. The modular structure of complex systems. *IEEE Transactions on software Engineering*, (3):259–266, 1985.