Module Guide for BrainInsight3D: 3D fMRI Visualization & Segmentation

Nada Elmasry

March 20, 2024

1 Revision History

Date	Version	Notes
Date 1	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
ProgName	Explanation of program 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 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 segmentation algorithm can change after experimentation and evaluation of accuracy and speed of inference.

AC2: The image processing steps can can change and some steps may be omitted after evaluation of performance and speed to reach the best performance at the fastest speed.

AC3: Additional input formats as DICOM maybe supported based on time limitations.

AC4: User Interface Design and Functionality can change if better design and functionality were discovered while in development process.

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 application is unlikely to support other types of fMRI scans as neck and bones as they require different segmentation algorithms and substantial effort.

UC2: The addition of functional segmentation.

UC3: The system main architecture and modules interaction and flow steps are unlikely to change as they are the foundational steps in image processing and segmentation.

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 Format Module

M3: Input Verification Module

M4: Data Preprocessing Module

M5: 2D Slice Module

M6: 3D Volume Module

M7: Segmentation Module

M8: User Interface Module

M9: Main Program Module

M10: 2D Data Visualization Module

M11: 3D Data Visualization Module

M12: User Interface Rendering Module

M13: Segmentation Algorithm Processing Module

M14: Optimization 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.

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. *ProgName* means the module will be implemented by the ProgName 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	Input Format Module
	Input Verification Module
	Data Preprocessing Module
	2D Slice Module
	3D Volume Module
	Segmentation Module
	User Interface Module
	Main Program Module
Software Decision Module	2D Data Visualization Module
	3D Data Visualization Module
	User Interface Rendering Module
	Segmentation Algorithm Processing Module
	Optimization Module

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 Input Format Module (M2)

Secrets: Provides the data structure for input parameters and the algorithm used to read and verify the format the input scans.

Services: Reads, stores, and verifies the user input satisfies software requirements (input is a NifTi format with suitable metadata and is not corrupted).

Implemented By: Python Libraries (nibabel and nilearn)

Type of Module: Library

7.2.2 Input Verification Module (M3)

Secrets: Provides application specific validation checks.

Services: Verifies the user input satisfies software requirements and redirects to the suitable actions or error messages depending in the test results.

Implemented By: BrainInsight3D

Type of Module: ADT

7.2.3 Data Preprocessing Module (M4)

Secrets: The data structure for input scans, the algorithms and data structures required for denoising and filtering, and the processed output scans.

Services: Preforms denoising, filtering, correction, retargeting and cropping of input scans.

Implemented By: BrainInsight3D

Type of Module: Abstract Data Type(ADT)

7.2.4 2D Slice Module (M5)

Secrets: The data structure for the input scan volume, the data structure for the output slices in the coronal,axial, and sagittal planes.

Services: Slices the input slice volume into three visualization planes; coronal,axial, and sagittal planes.

Implemented By: Slices the input slice volume into three visualization planes; coronal,axial, and sagittal planes. Implemented By: BrainInsight3D

Type of Module: Abstract Data Type(ADT)

7.2.5 3D Volume Module (M6)

Secrets: The data structure for the input scan volume, the data structure for the output validated volume for visualization.

Services: Performs mapping, compression, and format conversion, if necessary, of input volumes to visualize in the output interface.

Implemented By: BrainInsight3D

Type of Module: Abstract Data Type(ADT)

7.2.6 Segmentation Module (M7)

Secrets: The data structure for the input scan volume, the data structure for the segmented volume

Services: Performs anatomical segmentation on the input volume and returns the output volume with segmentation labels.

Implemented By: BrainInsight3D

Type of Module: Abstract Data Type(ADT)

7.2.7 User Interface Module (M8)

Secrets: The design and functionality of the web user interface module

Services: Provides an interface for the users to upload their scans and interact with the displayed 2D and 3D slices.

Implemented By: BrainInsight3D

Type of Module: Abstract Data Type(ADT)

7.2.8 Main Program Module (M9)

Secrets: The functionality to manage the program and coordinate between the application modules.

Services: Contains the main function that manages the program flow, visualization pipeline, and segmentation pipeline. Acts as the link between all modules.

Implemented By: BrainInsight3D

Type of Module: Abstract Data Type(ADT)

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 2D Data Visualization Module (M10)

Secrets: Displays 2D slices with and without segmentation results.

Services: Takes 2D slices and segmentation masks as input and displays them inside the user interface. Also Enables interactivity with displayed results for better user experience.

Implemented By: Papaya

Type of Module: Library

7.3.2 3D Data Visualization Module (M11)

Secrets: Displays 3D volumes with and without segmentation results.

Services: Takes 3D volumes and segmentation masks as input and displays them inside the user interface. Also Enables interactivity with displayed results for better user experience.

Implemented By: Three.js

Type of Module: Library

7.3.3 User Interface Rendering Module (M12)

Secrets: The functionality for rendering the application code inside the browser.

Services: Renders the user interface inside the browser.

Implemented By: React

Type of Module: Library

7.3.4 Segmentation Algorithm Processing Module (M13)

Secrets: The functionality for reading and loading the pre-trained segmentation model.

Services: Reads the segmentation model file with the model architecture and weights then loads the model to be available for inference inside the application.

Implemented By: PyTorch

Type of Module: Library

7.3.5 Optimization Module (M14)

Secrets: Functionality and Algorithms to enhance image preprocessing and model performance.

Services: provides functions for optimized image processing and augmentation, and optimizing model training to improve model performance.

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	M1, M2, M3
R2	M2, M3, M4, M5, M6, M8, M9, M10, M11, M12
R3	M2, M3, M4, M5, M6, M8, M9, M10, M11, M12
R4	M4, M7, M8, M9, M10, M11, M12, M13, M14
R5	M4
R6	M7, M13, M14

Table 2: Trace Between Requirements and Modules

AC	Modules
AC??	M7, M13, M14
AC??	M4
AC??	M2, M3
AC??	M8, M12, M9

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.

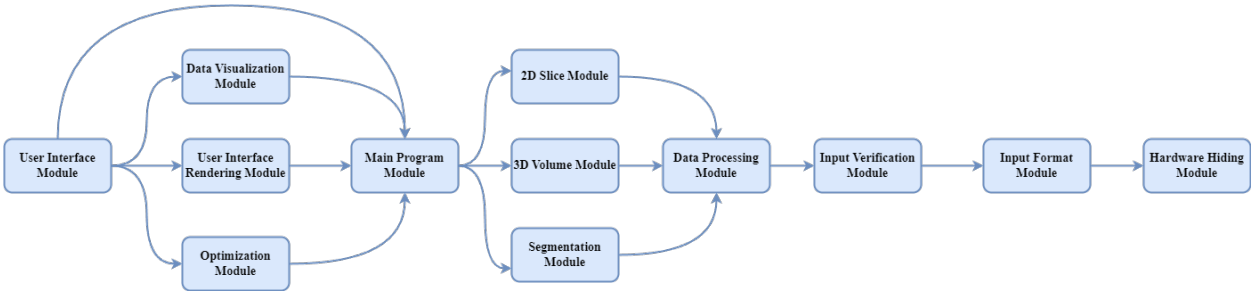

Figure 1: Use hierarchy among modules

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.