Module Guide for AortaGeomRecon

Jingyi Lin

June 22, 2023

1 Revision History

Date	Version	Notes
2022-10-18	1.0	First draft of Module Guide
2023-04-21	1.1	Second draft of Module Guide

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
NFR	Non-Functional requirements
SC	Scientific Computing
SRS	Software Requirements Specification
AortaGeomRecon	Aorta Geometry Reconstruction
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 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 algorithm to segment the aorta.

AC4: The data structures to store the input parameters required to execute the algorithm.

AC5: The methods to create a user interface.

AC6: The methods to retrieve a region of interest.

AC7: The methods to visualize a volume.

AC8: How the overall control of the calculations is orchestrated.

AC9: The format of the final output data.

...

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 input volume data's dimensionality 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 Format Module

M3: Input Parameter Module

M4: Control Module

M5: GUI Module

M6: Volume Visualization Module

M7: Crop Volume Module

M8: Aorta Segmentation Module

M9: Image Processing Module

M10: Multidimensional Array Processing Module

...

Level 1	**Level 2**	**Level 3**
Hardware-Hiding Module		
Behaviour-Hiding Module	Input Format Module	
	Input Parameter Module	
	Control Module	
	Volume Visualization Module	
	Crop Module	
	Aorta Segmentation Module	
Software Decision Module	GUI Module	
	Image Processing Module	
	Multi-Dimensional Array Processing 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. *AortaGeomRecon* means the module will be implemented by the AortaGeomRecon 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 behaviors.

Services: Includes programs that provide externally visible behavior 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: 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: AortaGeomRecon

Source: AortaGeomReconDisplayModuleLogic module's process function

7.2.2 Input Parameter Module (M3)

Secrets: The data structure for input parameters, how the values are input and how the values are verified. The load and verify secrets are isolated to their own access programs.

Services: Gets input from user, stores input and verifies that the input parameters comply with physical and software constraints. Throws an error if a parameter violates a physical constraint. Throws a warning if a parameter violates a software constraint. Stored parameters can be read individually, but write access is only to redefine the entire set of inputs.

Implemented By: AortaGeomRecon

Source: AortaSegmenter class' atributes

7.2.3 Control Module (M4)

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

Services: Provides the main program's entry point, the ability to jump from a program state to another.

Implemented By: AortaGeomRecon

Source: AortaGeomReconDisplayModuleWidget module

7.2.4 Volume Visualization Module (M6)

Secrets: The methods which allow users to visualize a 3D Volume.

Services: Display the aorta images and vtk 3D geometry.

Implemented By: 3D Slicer

7.2.5 Crop Module (M7)

Secrets: The parameters, libraries to retrieve a region of interest from a volume.

Services: Import the necessary libraries, store the input parameter, and coordinate the uses of these data and libraries to retrieve a region of interest.

Implemented By: 3D Slicer

7.2.6 Aorta Segmentation Module (M8)

Secrets: The parameters, libraries to perform segmentation.

Services: Import the necessary libraries, store the input parameter, and coordinate the uses of these data and libraries to perform segmentation.

Implemented By: AortaGeomRecon

Source: AortaSegmenter module

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 GUI Module (M5)

Secrets: The necessary library/framework to build a GUI software.

Services: Povide Graphical User Interface for user to write/read inputs, and send commands to the program. It could include the rendering a windows, inputs, keyboard and mouse interaction with the GUI elements.

Implemented By: 3D Slicer

7.3.2 Image Processing Module (M9)

Secrets: The libraries and the APIs to perform image analysis, and image segmentation.

Services: Provides useful APIs such as ThresholdSegmentationLevelSetsImageFilter, LabelStatisticImageFilter to perform aorta segmentation.

Implemented By: SITK

7.3.3 Multi-Dimensional Array Processing Module(M10)

Secrets: The libraries and the APIs to perform element-wise mathematic operations on multidimensional array.

Services: Provides useful APIs such as calculating the max, min, average of multidimensional array. NumPy.where function provides the search in the multidimensional array functionality which will return any element's indexes if it satisfies the given condition.

Implemented By: NumPy

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes. The list of requirements can refer to the AortaGeomRecon's SRS. Lin (2023)

Req.	**Modules**
R1	M1, M2, M3, M4
R2	M3, M7
R3	M8, M9, M10
R4	M6
NFR1	M3, M4, M5
NFR2	M4, M8, M9, M10
NFR3	M3, M4, M5 M6, M7, M8
NFR4	M7, M8, M9, M10
NFR5	M3

Table 2: Trace Between Requirements and Modules

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

Table 3: Trace Between Anticipated Changes and Modules

Modules	Code
M1	-
M2 Input Format Module	

```
# AortaGeomReconDisplayModule.py
# AortaGeomReconDisplayModuleWidget.onApplyButton
stop_limit = self._parameterNode.GetParameter("stop_limit")
threshold_coef = self._parameterNode.GetParameter("threshold_coef")
rms_error = self._parameterNode.GetParameter("rms_error")
no_ite = self._parameterNode.GetParameter("no_ite")
curv_scaling = self._parameterNode.GetParameter("curv_scaling")
prop_scaling = self._parameterNode.GetParameter("prop_scaling")
kernel_size = self._parameterNode.GetParameter("kernel_size")

# AortaGeomReconDisplayModuleLogic.process
def process(self, des_seed, asc_seed, stop_limit, threshold_coef,
            kernel_size, rms_error, no_ite, curvature_scaling,
            propagation_scaling, debug):
    """Convert the parameters to the correct format and
    call begin_segmentation from AortaSegmenter.

    Returns:
        SITK::image: The processing image, or the segmentation label
            image
    """
    des_seed = des_seed.split(",")
    asc_seed = asc_seed.split(",")
    asc_seed = [int(i) for i in asc_seed]
    des_seed = [int(i) for i in des_seed]
    now = datetime.now()
    if not self._cropped_image:
        volume = slicer.mrmlScene.GetFirstNode(
                "cropped", None, None, False)
        self.transform_image(volume)
    logging.info(f"{now} processing")
    segmenter = AortaSegmenter(
        cropped_image=self._cropped_image, des_seed=des_seed,
        asc_seed=asc_seed, stop_limit=float(stop_limit),
        threshold_coef=float(threshold_coef),
        kernel_size=int(float(kernel_size)),
        rms_error=float(rms_error), no_ite=int(no_ite.split(".")[0]),
        curvature_scaling=float(curvature_scaling),
        propagation_scaling=float(propagation_scaling), debug=debug
    )
    segmenter.begin_segmentation()
    now = datetime.now()
    logging.info(f"{now} Finished processing")
    return segmenter.processing_image
```

M3 Input Parameter Module

```
# AortaSegmenter.py
# AortaSegmenter.__init__ (Constructor)
def __init__(
            self, cropped_image, des_seed, asc_seed, stop_limit=10,
            threshold_coef=3, kernel_size=6, rms_error=0.02, no_ite=600,
            curvature_scaling=2, propagation_scaling=0.5, debug=False
    ):
    self._des_seed = des_seed
    self._des_prev_centre = des_seed[:2]
    self._asc_seed = asc_seed
    self._asc_prev_centre = asc_seed[:2]
    self._stop_limit = stop_limit
    self._threshold_coef = threshold_coef
    self._cropped_image = cropped_image
    self._kernel_size = kernel_size
    self._debug_mod = debug
    self._stats_filter = sitk.LabelStatisticsImageFilter()
    self._segment_filter = sitk.ThresholdSegmentationLevelSetImageFilter
        ()
    self._segment_filter.SetMaximumRMSError(rms_error)
    self._segment_filter.SetNumberOfIterations(no_ite)
    self._segment_filter.SetCurvatureScaling(curvature_scaling)
    self._segment_filter.SetPropagationScaling(propagation_scaling)
    self._segment_filter.ReverseExpansionDirectionOn()
    self._k = 2
```

M4 Control Module

```
# AortaGeomReconDisplayModule.py
# AortaGeomReconDisplayModuleWidget.onApplyButton
def onApplyButton(self):
    """
    Go to next phase if on phase 1 crop aorta or perform segmentation if
        on phase 2 aorta segmentation.
    """

    with slicer.util.tryWithErrorDisplay(errorMessage, waitCursor=True):
        if self._parameterNode.GetParameter("phase") == "1":
            size = len(slicer.util.getNodes("*cropped*", useLists=True))
            if not size:
                logging.info("Cannot find cropped volume")
            else:
                self.showPhaseAS()
        elif self._parameterNode.GetParameter("phase") == "2":
            descAortaSeed = self._parameterNode.GetParameter(
                "descAortaSeed")
            ascAortaSeed = self._parameterNode.GetParameter(
                "ascAortaSeed")
            volume = sceneObj.GetFirstNode("cropped", None, None, False)
            self.logic.transform_image(volume)
            image = self.logic.process(
                descAortaSeed, ascAortaSeed, stop_limit,
                threshold_coef, kernel_size, rms_error, no_ite,
                curv_scaling, prop_scaling, self.ui.debugBox.checked
            )
            sitkUtils.PushVolumeToSlicer(
                image,
                name="Seg_th{}_k{}_c{}_p{}".format(
                    threshold_coef,
                    kernel_size,
                    curv_scaling,
                    prop_scaling),
                className="vtkMRMLScalarVolumeNode"
            )
```

M5 GUI Module

```
# AortaGeomReconDisplayModule.AortaGeomReconDisplayModuleWidget class
# Set up code
class AortaGeomReconDisplayModuleWidget(ScriptedLoadableModuleWidget,
    VTKObservationMixin):
    def setup(self):
        ScriptedLoadableModuleWidget.setup(self)
        uiWidget = slicer.util.loadUI(self.resourcePath('UI/
            AortaGeomReconDisplayModule.ui'))  # noqa: E501
        self.layout.addWidget(uiWidget)
        self.ui = slicer.util.childWidgetVariables(uiWidget)
        uiWidget.setMRMLScene(slicer.mrmlScene)
        self.logic = AortaGeomReconDisplayModuleLogic()
        scene = slicer.mrmlScene
        self.crosshairNode = slicer.util.getNode("Crosshair")
        self.crosshairNode.AddObserver(
            slicer.vtkMRMLCrosshairNode.CursorPositionModifiedEvent,
            self.onMouseMoved
        )

        self.addObserver(scene, scene.StartCloseEvent, self.
            onSceneStartClose)
        self.addObserver(scene, scene.EndCloseEvent, self.onSceneEndClose
            )
        self.ui.ascAortaSeed.connect(
            "coordinatesChanged(double*)", self.
                updateParameterNodeFromGUI)
        self.ui.descAortaSeed.connect(
            "coordinatesChanged(double*)", self.
                updateParameterNodeFromGUI)
        self.ui.stopLimit.connect(
            "valueChanged(double)", self.updateParameterNodeFromGUI)
        self.ui.kernelSize.connect(
            "valueChanged(double)", self.updateParameterNodeFromGUI)
        self.ui.thresholdCoefficient.connect(
            "valueChanged(double)", self.updateParameterNodeFromGUI)
        self.ui.rmsError.connect(
            "valueChanged(double)", self.updateParameterNodeFromGUI)
        self.ui.noIteration.connect(
            "valueChanged(double)", self.updateParameterNodeFromGUI)
        self.ui.curvatureScaling.connect(
            "valueChanged(double)", self.updateParameterNodeFromGUI)
        self.ui.propagationScaling.connect(
            "valueChanged(double)", self.updateParameterNodeFromGUI)

        # Buttons
        self.ui.applyButton.connect('clicked(bool)', self.onApplyButton)
        self.ui.revertButton.connect('clicked(bool)', self.onRevertButton
            )
        self.ui.resetButton.connect('clicked(bool)', self.onResetButton)
        self.ui.skipButton.connect('clicked(bool)', self.onSkipButton)
        self.ui.getVTKButton.connect('clicked(bool)', self.onGetVTKButton
            )

        sliceDisplayNodes = slicer.util.getNodesByClass(
            "vtkMRMLSliceDisplayNode")
        for sliceDisplayNode in sliceDisplayNodes:
            sliceDisplayNode.SetIntersectingSlicesVisibility(1)

        sliceNodes = slicer.util.getNodesByClass('vtkMRMLSliceNode')
        for sliceNode in sliceNodes:
            sliceNode.Modified()
        self.initializeParameterNode()
```

M6 Volume Visualization Module	3D Slicer's Volume Rendering Module
M7 Crop Volume Module	3D Slicer's Crop Volume Module
M8 Aorta Segmentation Module	AortaSegmenter class
M9 Image Processing Module	SimpleITK
M10 Multi-Dimensional Array Processing Module	NumPy

Table 4: Trace Between Modules and Code

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

10 References

References

Jingyi Lin. System requirements specification. https://github.com/smiths/aorta/blob/main/docs/SRS/SRS.pdf, 2023.

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.