

BUILDING AN ASSURANCE CASE FOR
AORTA GEOMETRY RECONSTRUCTION
SOFTWARE

BUILDING AN ASSURANCE CASE FOR AORTA GEOMETRY
RECONSTRUCTION SOFTWARE

BY
JINGYI LIN, M.Eng.

A REPORT
SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE
AND THE SCHOOL OF GRADUATE STUDIES
OF MCMASTER UNIVERSITY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF ENGINEERING

© Copyright by Jingyi Lin, August 2023

All Rights Reserved

Masters of Engineering (2023) McMaster University
(Department of Computing and Software) Hamilton, Ontario, Canada

TITLE: Building an Assurance Case for Aorta Geometry Reconstruction Software

AUTHOR: Jingyi Lin
M.Eng. (Computing and Software CRP),
McMaster University, Hamilton, Canada

SUPERVISOR: Smith Spencer

NUMBER OF PAGES: **x, ??**

Abstract

Assurance cases have been used to build safe real-time system software. Like real-time systems, medical image processing software also requires a high standard of correctness, completeness, and consistency. Therefore, we have investigated adopting real-time assurance case techniques to medical image processing software. For this project, we develop medical image processing software, called AortaGeomRecon, that helps to diagnose issues related to the aorta. We build an assurance case for this software by expanding on the evidence used for previous work on assurance cases for scientific software. In this way, we reinforce our confidence in the AortaGeomRecon's correctness, completeness, and consistency. Our techniques can be generalized to other scientific software.

With a Jupyter Notebook program left by a previous student as a starting point, we improved the existing segmentation algorithm and developed a 3D Slicer extension module that includes a Graphical User Interface and a module for input parameters. We built the continuous integration infrastructure with GitHub Actions. This allows us to update the algorithm while ensuring that it is at least as good as its previous version. Additionally, a linter is set up as part of the continuous integration process to ensure the program's readability by enforcing the PEP8 standard.

Next, we built assurance cases in Goal Structuring Notation with the bottom-up

approaches. We gathered our existing evidence and explored new implementation requirements for the new evidence. We finalized our documentation on requirements, system architecture, user manual, and detailed design, which reinforces our confidence in these documentations and that the design complies with the documentation. Finally, we conduct a code review meeting and an algorithm review meeting to discuss our methodologies. These sessions provided insights into potential enhancements for our segmentation algorithm and affirmed the correctness of our remaining methods.

Acknowledgements

I would like to thank all the people who contributed in some way to the work described in this thesis.

Above all, I want to extend my heartfelt appreciation to my supervisor, Dr. Spencer Smith. His unwavering motivation, patience, and ongoing support throughout my master's studies and research have been invaluable. His guidance has been instrumental throughout the entire research and thesis writing process.

Next, I would also like to express my gratitude to Kailin Chu for her significant contribution to the research, including the development of the initial segmentation algorithm, and for dedicating her time to help in our code review.

Furthermore, I extend my appreciation to Dr. Dean Inglis for his valuable insights into an alternative segmentation algorithm and for generously dedicating his time to assist in our algorithm review.

Lastly, I want to give a special thanks to my family. They have been there for me throughout my life, offering constant love and unwavering encouragement. Without them, I wouldn't have achieved this milestone.

Contents

Abstract	iii
Acknowledgements	v
Notation, Definitions, and Abbreviations	xi
1 Introduction	1
1.1 Background	3
1.2 Methodology	5
1.3 Thesis Outline	6
2 AortaGeomRecon Research and Development	8
2.1 Existing Methods	9
2.2 Our Segmentation Algorithm	12
2.3 3D Slicer Extension Development	20
3 Assurance Cases and Selected Evidence for AortaGeomRecon	27
3.1 Assurance Case Development	27
3.2 Assurance Case for Software Specification Requirements	28
3.3 Assurance Case for the Implementation	43

3.4	The Design Documentation of the AortaGeomRecon	50
3.5	Assurance Case for Operational Assumptions	62
3.6	Assurance Case for Inputs Assumptions	65
4	Conclusion and Future Works	68
4.1	Thesis Summary	68
4.2	Challenge	70
4.3	Lessons Learned	71
4.4	Future Work	73
A	Software Requirements Specification for AortaGeomRecon	81
B	Module Guide for AortaGeomRecon	103

List of Figures

1.1	Aorta	3
1.2	Organ Segmentation	4
1.3	Simple Assurance Case Diagram	5
2.1	ITK-Snap's Bubble Segmentation UI	10
2.2	3D Slicer Built-in Segmentation UI	11
2.3	The Aorta Seeds	13
2.4	Level Sets Segmentation	15
2.5	A label image	16
2.6	A Distance Map	17
2.7	Code That Shows How To Calculate The Threshold Range	17
2.8	A Segmented Image	18
2.9	Segmentation Result	20
2.10	Jupyter Notebook Research	21
2.11	3D Slicer UI	23
2.12	AortaGeomRecon Module UI	24
2.13	AortaGeomRecon Warning message	25
3.1	AGR Assurance Cases Top Level	29
3.2	AGR Assurance Cases GR	30

3.3	AGR SRS Table Of Content	32
3.4	AGR SRS Assumptions	34
3.5	AGR SRS Data Definitions	35
3.6	AGR SRS Instance Model Region Of Interest	36
3.7	AGR SRS Instance Model Region Of Interest	37
3.8	AGR Functional Requirements	38
3.9	AGR Non- Functional Requirements	39
3.10	AGR Traceability Matrix between Data Definitions and Instance Model	40
3.11	AGR Traceability Matrix Between Assumptions and Other sections	40
3.12	AGR Traceability Matrix Between Requirements and Other sections	41
3.13	GitHub Repo Documentation Review Requests	42
3.14	AGR Assurance Cases For Implementation	44
3.15	AGR Manual Test Procedure	48
3.16	AGR Test Report	49
3.17	AGR Anticipated Changes	51
3.18	AGR Modules	52
3.19	AGR Module Decomposition Example	53
3.20	AGR Modules Traceability Matrices	54
3.21	AGR Part Of The Traceability Matrix On Modules And Code	55
3.22	AGR Design Document Website	56
3.23	AGR Design Document Glossary	57
3.24	Spyder Variable Explorer	61
3.25	AGR Assurance Case Operational Assumptions	62
3.26	AGR User Manual On GitHub README	63

3.27 AGR User Instructions on YouTube	64
3.28 AGR Assurance Case Inputs Assumptions	65
3.29 AGR Warning Message	67
4.1 Aorta Double Oblique Plane	74

Notation, Definitions, and Abbreviations

Definitions

Aorta The aorta is the largest artery of the body and carries blood from the heart to the circulatory system. It has five sections: aortic root, ascending aorta, aortic arch, descending aorta and abdominal aorta. The aortic root is the transition point where blood first exits the heart. It functions as the water main of the body. The aortic arch is the curved segment that gives the aorta its cane-like shape. It bridges the ascending and descending aorta. Throughout this documentation, aorta means ascending aorta, aortic arch and descending aorta. The abdominal aorta and aortic root are outside of the scope of the current work.

Ascending Aorta

The ascending aorta is like the starting point of the aorta. It comes out of the heart and sends blood through the aortic arch and down

into the descending aorta [21].

Descending Aorta

The descending aorta begins after a branch called the left subclavian artery separates from the main road (aortic arch) and goes all the way down into your belly [22].

Organ Segmentation

Organ segmentation refers to the process of dividing or delineating an organ's internal structures or regions from medical images or scans. This segmentation is typically done using computer algorithms or manual techniques to distinguish different parts of an organ, such as tumors, blood vessels, or healthy tissue, for the purpose of analysis, diagnosis, or treatment planning in medical applications like radiology or surgery [6].

DICOM Digital Imaging and Communications in Medicine (DICOM) is the standard for the communication and management of medical imaging information and related data.

Inferior Inferior is the direction away from the head; Inferior means the lower parts of the body (e.g., the foot is part of the inferior extremity).

Superior Superior is the direction toward the head end of the body; Superior means the upper parts of the body (e.g., the hand is part of the superior extremity).

Slice A 2-dimensional image that is retrieved from a 3-dimensional volume.

Binary Dilation

Binary dilation is a mathematical morphology operation that uses a

structuring element (kernel) for expanding the shapes in an image.

- Label Map** A labeled map or a label image is an image that labels each pixel of a source image.

Euclidean Distance Transform

The Euclidean Distance Transform (EDT) is a mathematical technique used in image processing and computer vision to compute the distance of each pixel in a binary image to the nearest boundary or edge pixel.

- Contour Line** A contour line is a continuous curve on a two-dimensional map or graph that connects points of equal value.

- Level Sets** In Level Sets, an image is represented as a function, and the evolving contour or boundary is represented as an isosurface of this function [24]. This contour evolves over time to converge toward the desired segmentation result. The key advantage of Level Sets is its ability to handle topological changes naturally, which makes it useful for tasks like segmenting objects with irregular shapes or objects that may split or merge during the segmentation process. The Figures 2.4 demonstrate an example of how Level Sets method work on finding the region of the heart. It starts with a seed contour that is within the region of interest, then by finding the gradient based on the contour line, the segmentation result will propagate towards outside of the region until the maximum difference between the neighboring pixels are reached.

Segmented slice

A 2-dimensional image with the relevant pixels labeled as 1 and other pixels as 0.

Kernel Size The size of the kernel for binary dilation.

Stop Limit This limit is used to stop the segmentation algorithm. It is used differently in segmentation in the inferior and suprior direction.

Threshold Coefficient

This coefficient is used to compute the lower and upper threshold passing through the segmentation filter SITK's ThresholdSegmentationLevelSetImageFilter. The algorithm first uses SITK's LabelStatisticsImageFilter to get the mean and the standard deviation of the intensity values of the pixels that are labeled as the white pixel. Larger values with this coefficient imply a larger range of thresholds when performing the segmentation, which leads to a larger segmented region.

RMS Error Value of RMS change below which the filter should stop. This is a convergence criterion.

Maximum Iteration

Maximum number of iterations to run

Curvature Scaling

Weight of the curvature contribution to the speed term.

Propagation Scaling

Weight of the propagation contribution to the speed term.

Abbreviations

AC	Assurance Case
AGR	AortaGeomRecon
AortaGeomRecon	
	3D Slicer's extension module, Aorta Geometry Reconstruction
CT	Computerized Tomography
DD	Design Document
DICOM	Digital Imaging and Communications in Medicine
GUI	Graphical User Interface
MG	Module Guide
NFR	Non-Functional Requirement
FR	Functional Requirement
SITK	SimpleITK
SRS	Software Requirements Specification
UI	User Interface
VTK	The Visualization Toolkit

Chapter 1

Introduction

Medical Software is a critical component of patient diagnosis and treatment. Medical software refers to computer programs, applications, or systems specifically designed for use within the healthcare and medical field. These software solutions are developed to assist healthcare professionals, researchers, administrators, and patients in various aspects of medical care, research, management, and education [27]. Our project focuses on medical software that could potentially influence a patients' well-being, particularly software that contributes to diagnosing issues related to the aorta. The aorta, a vital artery responsible for transporting blood from the heart to other bodily organs, holds immense significance. Any malfunction in its blood-carrying function could yield severe and potentially life-threatening consequences for the entire body's physiology. Specifically, we focus on the Aorta Geometry Reconstruction (AortaGeomRecon or AGR) software, which can build a 3-dimensional model of the aorta, to help the health professional diagnosing issues related to the aorta quickly and correctly.

Given the importance of medical software like AGR, we need a means to build

confidence in the software. In this report, we explore the use of assurance cases. An assurance case can be thought of as a structured argument. The main purpose of an assurance case is to establish confidence and trust in the reliability and safety of a system by presenting a well-structured argument supported by evidence [28]. Assurance cases have been applied regularly in the medical device for approval in U.S. In Europe, the assurance cases are required in systems as diverse as flight control systems, nuclear reactor shutdown systems, and railroad signaling systems, which are all critical systems [28]. Previous work [19] [25] investigating assurance cases for scientific computing software such as 3dfim+, a medical imaging software analyzing activity in the brain, has demonstrated the value of the technique in showing the software's correctness and reliability. The motivation of our project is to build on this previous work to more deeply investigate the necessary evidence. Specifically, we build an assurance case for AGR by adding more details on the evidence needed to support our correctness claims, thus building our confidence in AGR.

In this chapter, we first explain in details the contexts for the key concepts that will be discussed throughout the document, including what is organ segmentation, what is the aorta, listing the diseases that aorta segmentation could detect, and demonstrating an example of an assurance case by showing a simple example. Next, we will briefly discuss the methodology, especially how we achieve the objective of design, implementation of the software, and building confidence with the evidences in an assurance case. In the final section, we explain our report outline.

1.1 Background

In this section, we present the contexts of the key concepts within the scope of our work, including background information on the aorta (section 1.1.1), organ segmentation (section 1.1.2), and assurance cases (section 1.1.3).

1.1.1 Aorta

The aorta is the largest artery in the body. It carries blood from the heart to the circulatory system. It has a cane-like shape made up of the ascending aorta, aortic arch and descending aorta. Figure 1.1 shows the entire aorta, but the abdominal aorta is outside the scope of the current work. Our work focus on building the 3D geometry from the aortic root to descending aorta.

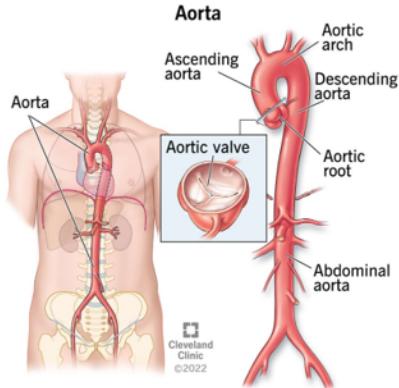


Figure 1.1: Aorta [18]

1.1.2 Organ Segmentation

The definition of the organ boundary or organ segmentation is helpful for the orientation and identification of the regions of interest inside the organ during diagnostic

or treatment procedures. Further, it allows the volume estimation of the organ, such as the aorta. A segmentation takes a medical image as input and outputs the portion of the image that corresponds to the organ of interest. Figure 1.2 demonstrates an example of abdominal organ segmentation.

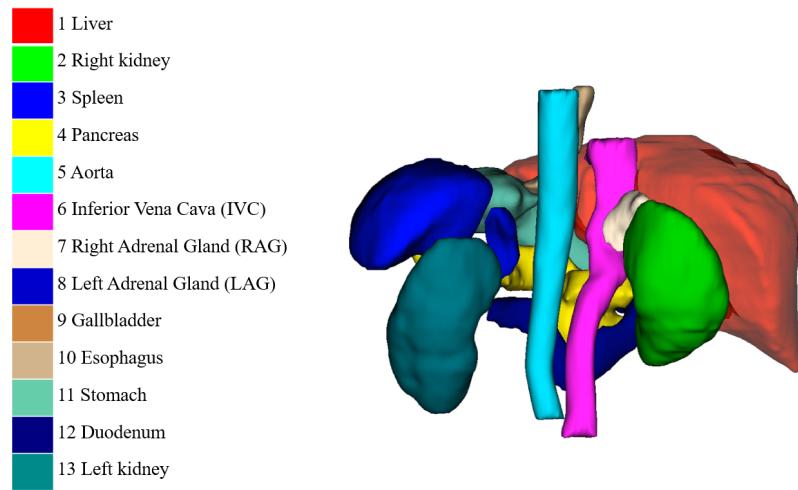


Figure 1.2: Organ Segmentation [17]

Aorta segmentation in CT (Computerized Tomography) scans is important for diagnosing or treating:

- Coarctation of the aorta (narrowing of aorta)
- Aortic aneurysm (bulge in the aorta)
- Aortic calcification quantification
- To guide the segmentation of other central vessels.

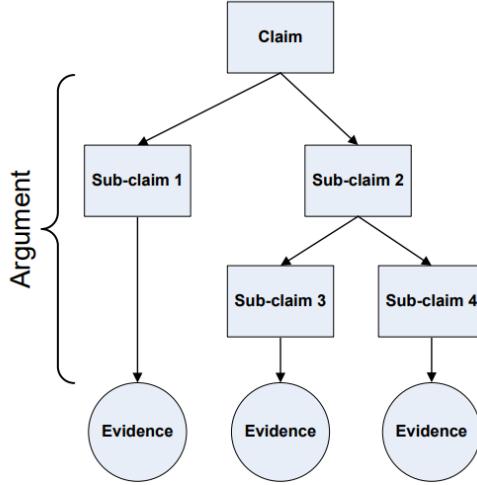


Figure 1.3: Simple Assurance Case Diagram [4]

1.1.3 Assurance Case

An Assurance Case (AC) can be thought of as a structured argument. When building an AC, you’re making a point that specific evidence backs up a particular statement. The fundamental structure is depicted in Figure 1.3. An AC essentially boils down to an organized collection of arguments, backed by a body of evidence, that helps validate the belief in a specific claim [4].

In a practical sense, creating an AC involves beginning with a main claim and then breaking it down into smaller claims through a step-by-step process. These smaller claims, at the most bottom, are supported by concrete evidence.

1.2 Methodology

In this study, we present the outcomes of integrating AC throughout the development of medical software to reinforce the stakeholders’ confidence in the software’s capabilities. The software, known as AortaGeomRecon (AGR), represents a 3D Slicer

[12] extension module designed to semi-automatically construct a 3D model of the aorta using CT scans from a patient’s chest. We started by gathering requirements for the AGR from a domain expert, drafted our requirements, and high-level design. We researched and worked on the implementation of the software, while building the infrastructure for continuous integration, version control, and project management using GitHub. When we have a functional prototype, we delved into our assurance cases, encompassing chosen arguments and supporting evidence. AC functions as a method to provide assurance for a system by presenting arguments that substantiates claims about the system. These arguments are based on evidence related to the system’s design, development, and tested behavior. By constructing the AC, we were able to follow the best practice including documentation review on the requirements and high-level design. Our goal was to finalize our documentation, and ensure the documentation’s completeness and correctness. We have built user documentation to define all operational assumptions, and guide the user to use the valid inputs with a sequence of correct operations. Finally, our assurance case evidence consists of continuous integration tests, code review, and several algorithm reviews. This increased our confidence that the implementation of the software has strictly complied with the requirements and is complete and correct.

1.3 Thesis Outline

The thesis is organized into three broad parts. In Chapter 2, we introduce our program AortaGeomRecon by mentioning the existing methods, the AGR’s algorithm overview and step-by-step workflow. We explain necessary terms and information to understand how the software functions. We also introduce the 3D Slicer [12] extension module that

the user interacts with to get the segmentation result with our algorithm. In Chapter 3, we present our AC, focusing on the evidence and including some sections of our requirements, high-level design, detailed design, algorithm review, and a test case we developed for verifying and validating the correctness of AGR. In Chapter 4, future work is proposed and conclusions are drawn based on the developed requirements, segmentation algorithm, 3D Slicer module extension, and AC.

Chapter 2

AortaGeomRecon Research and Development

This chapter will discuss the research and development of the 3D Slicer plugin AortaGeomRecon. AortaGeomRecon (AGR) stands for Aorta Geometry Reconstruction. The main objective of AGR is to semi-automatically build a 3D geometry of the aorta from the patient's chest CT scans. The existing methods often involve extensive manual work interspersed with software assistance. An experienced user, who should be a medical domain expert, typically needs to do a minimum of 10 minutes of manual work. Currently, AGR allows users who have taken the university level anatomy introductory course to set the hyperparameters within half a minute, following by a maximum 2 minutes of execution time.

In this chapter, we first introduce the existing methods for aorta segmentation with different software options, and discuss the advantages and disadvantages of these existing options. Next, we demonstrate our segmentation algorithm, with a detailed step-by-step explanation of what the algorithm is doing and the results it generates.

Finally, we discuss our experience using the platform 3D Slicer, and demonstrate our development results in the 3D Slicer plugin.

2.1 Existing Methods

There are many segmentation software options available to users. We will discuss two popular software options: [ITK-Snap](#) and [3D Slicer](#).

2.1.1 ITK-Snap bubble method

ITK-Snap provides a segmentation method that first requires the user to select multiple voxels with a custom initial size and an expanding size within the volume. This method is referred to as the “bubble method” [7].

Through many iterations, the voxels expand to fill the entire volume of the vessel. As a final step, the user will need to cut off the extra parts of the volume. Figure 2.1 shows ITK-Snap UI executing a segmentation of an aorta.

The advantage of the bubble method is that it is guaranteed to produce a correct segmentation for a valid image. A medical domain expert can manually control the desired area, and visually observe the segmentation result expanding, and shrinking. Moreover, the user can erase the unwanted parts.

The disadvantage of this method is that the operations described above are complicated. They are easier to say than do. An operator who has previous experience building the geometry with this method still needs about 20 minutes of manual work to build a new aorta geometry. Plus, ITK-Snap software can only read VTK (The Visualization Toolkit) files; therefore, the chest CT scans that are usually Digital

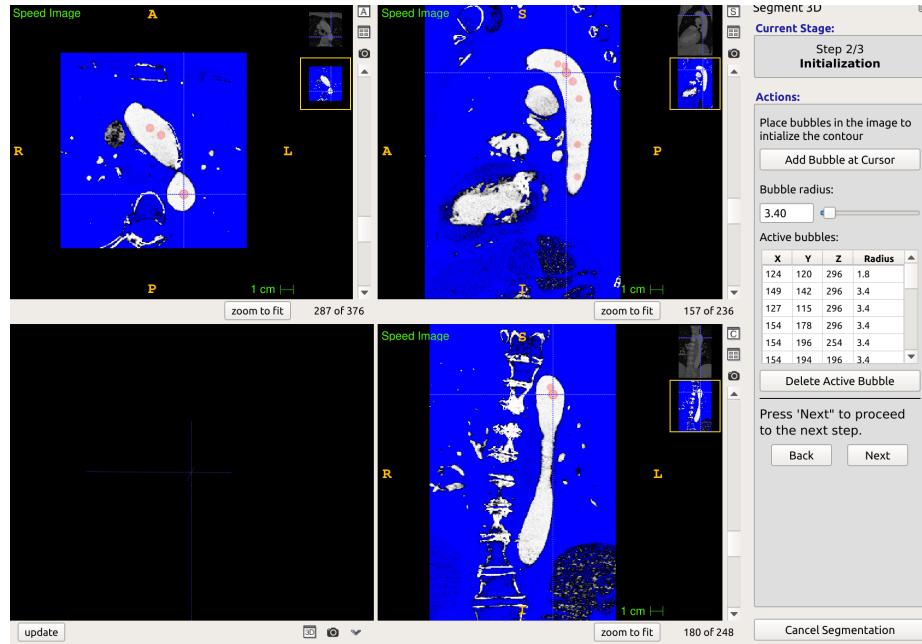


Figure 2.1: ITK-Snap’s Bubble Segmentation UI [31]

Imaging and Communications in Medicine (DICOM) format [8], needed a manual conversion step before using this software and its segmentation method.

2.1.2 3D Slicer threshold segmentation

3D Slicer is another well-known medical image processing software for academic uses. 3D Slicer provides multiple segmentation methods [1]. One of the quickest and easiest to use is the intensity based segmentation.

This method first lets users select a small area that belongs to the wanted area on a 2D plane (Axial, Sagittal, and Coronal). 3D Slicer reads the pixel intensity of the surrounding area, and segments based on the intensity threshold. Any pixel’s intensity that is within range will be segmented as the segmentation result, as demonstrated in Figure 2.2.

The advantage of this method is that the user can visualize the effect of choosing different intensity threshold in real-time. 3D Slicer can highlight the pixels that would be chosen as a part of the segmentation with an increased or decreased intensity threshold.

The disadvantage of this method is that it depends on the quality of the input data. If the input data does not have the proper intensity reading of the aorta, it could generate gaps on the wanted part. Then, this method is less efficient, because the user must use another intensity range for the missing pixels. A [YouTube video](#) shows an experienced user who gets the aorta 3D geometry with 8 minutes of manual work.

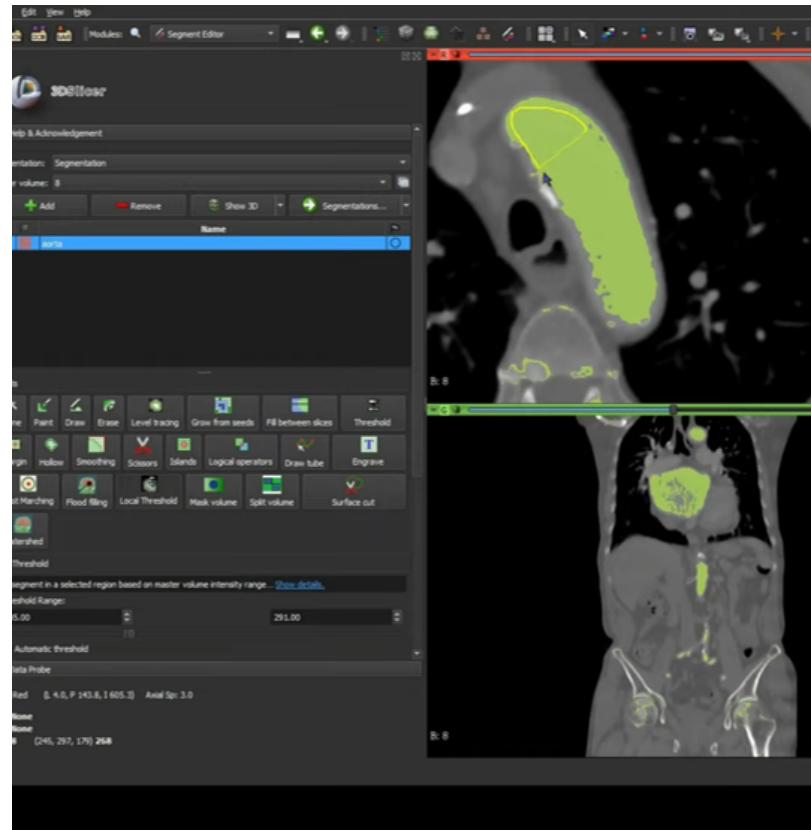


Figure 2.2: 3D Slicer Built-in Segmentation Method [12]

2.2 Our Segmentation Algorithm

This section introduces the key concepts for the implementation of our segmentation algorithm. The algorithm is developed in Python with the external libraries SimpleITK and NumPy. The algorithm builds the 3D aorta geometry by doing segmentation on each axial slice. The logic behind segmenting each slice from the axial view is that there is one or two circles that are edge bounded in each axial plane view. Using this information, and given an initial aorta center coordinate, the algorithm continuously segments the axial slice’s circles that are closest to the previous aorta center coordinates. Finally, some hyperparameters tuning can let the algorithm pick up the pixels that were missing but belongs to the aorta.

In this section, we first introduce the contexts for the external libraries used in the implementation of the algorithm. We show the parameters and hyperparameters of the algorithm. Then we explain the algorithm workflow with a step-by-step demonstration.

2.2.1 Background

SimpleITK is an open-source multidimensional image analysis library developed by the Insight Toolkit community for the biomedical sciences and beyond [3][16]. NumPy is the fundamental package for scientific computing with Python, especially for the performance of multidimensional array processing [10]. The algorithm will use functions from these two libraries for image processing and multidimensional array processing. For example, the algorithm segments each slice with ThresholdSegmentationLevelSetImageFilter from SITK.

The algorithm works best with the chest volume cropped to a rectangular prism

that contains the aorta and parts of the other organs such as the backbone, blood vessels, and the heart. This can be done with 3D Slicer and its built-in modules, Volume rendering and Crop Volume, or researched by another means familiar to the user for finding the starting point and the size to crop.

2.2.2 Parameters

At the beginning of the algorithm, the user inputs two integer coordinates indicating the position of the descending aorta and ascending aorta center on a single slice. The yellow dots in Figure 2.3 shows an example of the aorta seeds. These seeds will be updated by the algorithm after processing each axial plane.

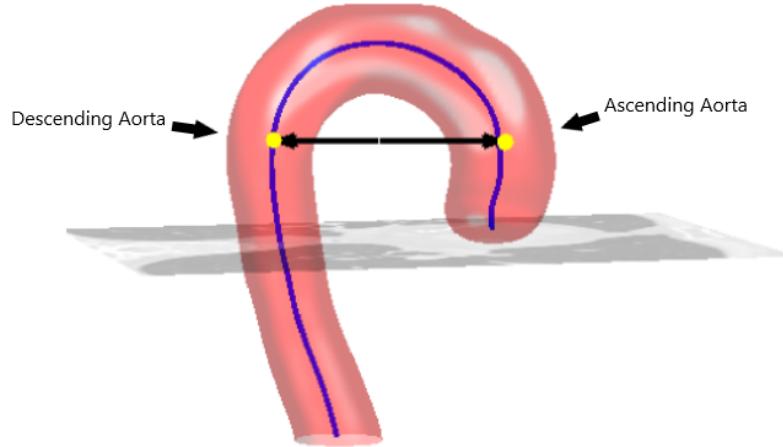


Figure 2.3: The aorta seeds [14]

The following list of hyperparameters that can be tuned to get the best segmentation result:

- The stop limit, which controls the stop condition

- The threshold coefficient, which controls the segmentation acceptable intensity range
- The kernel size, which controls the label image circle size
- The threshold Segmentation Level Sets Image Filter parameters, including:
 - The RMS error
 - The maximum iteration
 - The curvature scaling
 - The propagation scaling

One of the most important parameters is the threshold coefficient. Since the algorithm segments based on the intensity of the gray scale pixels, decreasing the threshold coefficient decreases the acceptable range of the pixels, and vice-versa.

2.2.3 Algorithm Overview

When the user has selected the aorta seeds, the plane where the aorta seeds are located is the initial plane. From this plane towards the bottom (toward the feet) is the inferior direction. The upward direction (toward the head) is the superior direction. This algorithm segments each slice with SITK::ThresholdSegmentationLevelSetImageFilter. The principles of this image filter can be explained with two terms: Level sets segmentation method, and a threshold range that defines the intensity of the acceptable pixel.

Level Sets are an important category of modern image segmentation techniques based on partial differential equations (PDE), i.e. progressive evaluation of the differences among neighboring pixels to find object boundaries. The pictures in Figure 2.4 demonstrate an example of how the Level Sets method work on finding the region

of the heart. It starts with a seed contour that is within the region of interest, then by finding the gradient based on the contour line, the segmentation result will propagate towards the outside of the region, until the maximum difference between the neighboring pixels are reached.

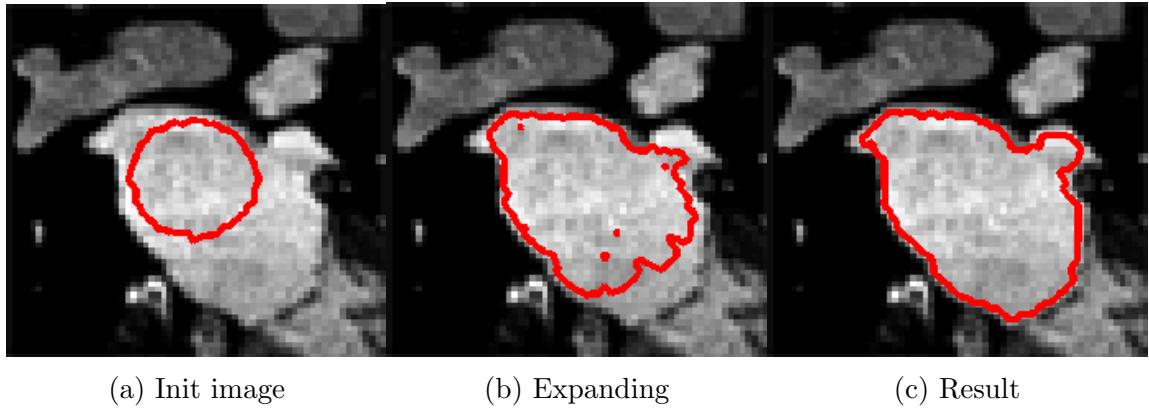


Figure 2.4: Level Sets Segmentation

2.2.4 The steps to segment a single slice

In the following section, we will present each step to segment a single slice. These steps are applied for segmentations in both the superior and inferior directions. There is a difference in the stopping condition, which we will elaborate in Section 2.2.4.6.

2.2.4.1 Create A Label Map

The algorithm uses SITK::BinaryDilateImageFilter to perform binary dilation to generate a circle-like shape around the center coordinates (user input's for the first slice and calculated by the algorithm for the rest of the slices). The binary dilation helps to fill in and smooth out gaps where the image intensity drops. Each pixel within this shape will be labeled as a white pixel (value of 1), and the rest of the pixels are

labeled as black pixels (value of 0).

The generated result is the label map image. We will use it in the next few steps. The size of the circle-like shape is determined by the kernel size (user's input). The Figure 2.5 shows an example of generated label map image (the green parts) overlay over the original slice.

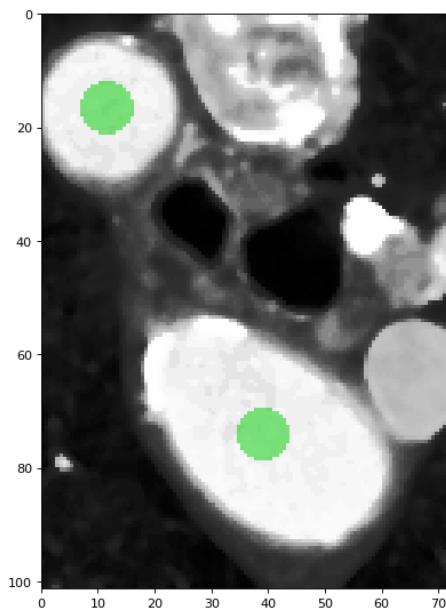


Figure 2.5: A Label Map

2.2.4.2 Create A Distance Map

With SITK::SignedMaurerDistanceMapImageFilter, the algorithm creates another image, the Euclidean distance transform of the label image from previous step. This is used as a contour line that helps build the gradient mentioned in Level sets. The Figure 2.6 shows an example of distance map.

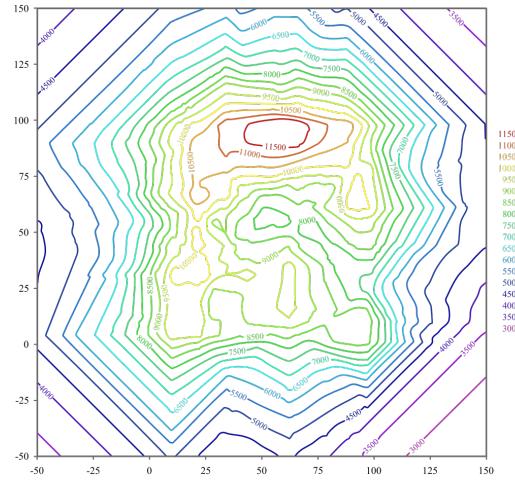


Figure 2.6: A Distance Map

2.2.4.3 Calculate A Threshold Range

By using SITK::LabelStatisticsImageFilter, the algorithm gets the mean and the standard deviation of the intensity values of the pixels that were labeled as the white pixel in the label map. The algorithm uses the threshold coefficient to calculate the lower and upper threshold to be used in the next step.

```
intensity_mean = self._stats_filter.GetMean(
    PixelValue.white_pixel.value)
std = self._stats_filter.GetSigma(PixelValue.white_pixel.value)
lower_threshold = (intensity_mean - self._threshold_coef*std)
upper_threshold = (intensity_mean + self._threshold_coef*std)
self._segment_filter.SetLowerThreshold(lower_threshold)
self._segment_filter.SetUpperThreshold(upper_threshold)
```

Figure 2.7: Code That Shows How To Calculate The Threshold Range

2.2.4.4 Segment A Single Slice

With SITK::ThresholdSegmentationLevelSetImageFilter, the seed image calculated in step 2.2.4.2, and the lower and upper threshold value calculated in step 2.2.4.3, the algorithm performs segmentation and generates a segmented slice. Figure 2.8 shows

an example of generated segmented slice (the green parts) overlay over the original slice.

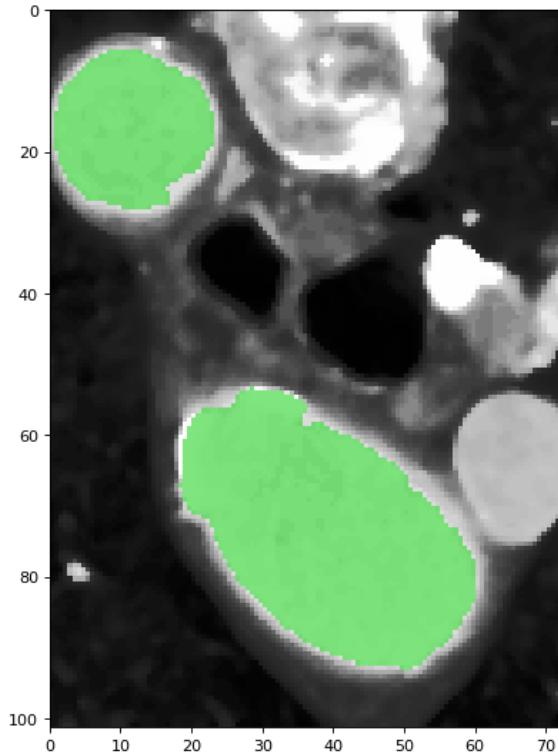


Figure 2.8: A Segmented Image On Top Of The Original Slice

2.2.4.5 Calculate New Centroids

By comparing each pixel segmented as aorta to the previous descending centroid and the previous ascending centroid, the algorithm uses the positions of the points closer to the previous descending centroid to calculate the new descending aorta centroid, and vice-versa for the ascending aorta centroid. However, at a certain point during the segmentation in the inferior direction, the slice will reach the end of the ascending aorta, the aortic root. We will explain how we detect this scenario in section 2.2.4.6. For the rest of the slices in the inferior direction, the algorithm will stop using,

and calculating the ascending aorta centroid as it only requires the descending aorta centroid.

2.2.4.6 Verify Segmentation Result

There are two main stopping conditions for verifying segmentation result, one condition for the segmentation in the inferior direction and the other one for the segmentation in the superior direction. The stopping limit is a user defined parameter to control the algorithm, that would affect the result in this step.

In the inferior direction, if the new ascending aorta centroid that is closest to the previous ascending aorta center is reaching the distance limit, then the algorithm will stop and consider taking the new centroid closer to the ascending aorta. In other words, only one centroid will be used for the descending aorta segmentation.

In the superior direction, if the standard deviation of the initial label map and the segmentation result label map have larger differences than the limit, the algorithm will stop processing segmentation for the rest of the slices. For example, if the standard deviation of the initial label image is 20, and the standard deviation of the segmentation label image is 40, with stop limit of 10, the program will halt immediately.

2.2.5 Algorithm Summary

Given two integer coordinates, ascending aorta centroid and descending aorta centroid, the algorithm set the inputted plane as the initial plane. From the initial plane to the bottom (toward the feet) plane, the algorithm calculates a label map with two centroid coordinates and kernel size, calculates a distance map with the label map,

calculates a threshold range with the label map's selected pixels, performs segmentation, calculates new centroid coordinates, and verifies the segmentation result in case that it reaches the stop condition. Once the algorithm finishes the segmentation in the inferior direction, the algorithm works from the initial plane to the top (toward the head) plane, repeating the similar steps. Each segmentation result slice is stored in a SITK's image, which supports the conversion to VTK file or DICOM file. Figure 2.9 demonstrates a segmentation result, rendered in 3D Slicer.

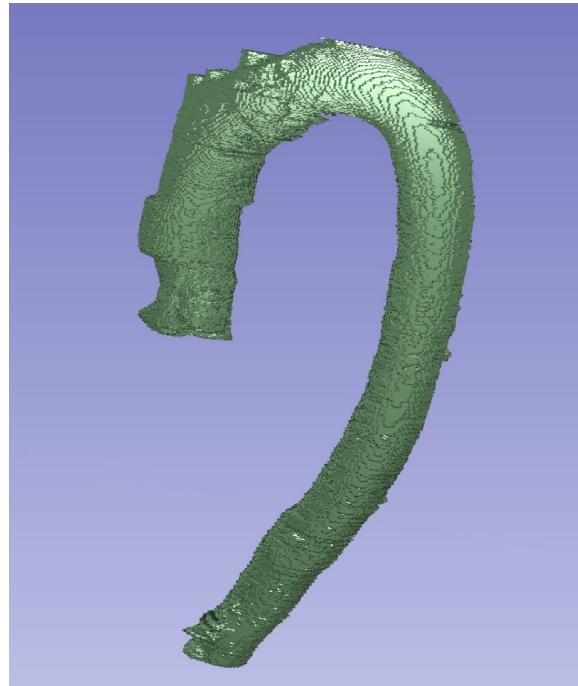


Figure 2.9: Segmentation Result

2.3 3D Slicer Extension Development

The project has started from a simple segmentation program build in Jupyter Notebook [13], inherited from a previous developer (Kailin Chu). With this program, the

user needs to investigate the chest CT scans using another software (like 3D Slicer or ITK-Snap), to get the necessary input readings, such as coordinates and size to crop (the coordinates of the yellow dots shown in Figure 2.3).

```

In [ ]: 1 # assign folder paths for all the dicom images (these are the ones provided by Vahid and Dr. Motamed)
2
3 folder_paths = ["../sample-dicom/43681283", "../sample-dicom/05937785", "../sample-dicom/07323651",
4     "../sample-dicom/75962810", "../sample-dicom/62023082", "../sample-dicom/22429388"]
5
6 # List of the number of slices in the z direction for each dicom image
7 # these are manually entered as each dicom folder has far more .dcm files than slices
8 # If you do not manually enter these values, the images will have many repeats of a singular CT
9 num_slices = [376, 423, 195, 188, 225, 447]

In [ ]: 1 # stores all of the .dcm files within each folder
2 dcm_series = []
3
4 # assign files to dcm_series
5 for i in range(len(folder_paths)):
6     # make list and sort alphanumerically
7     list_dcm = os.listdir(folder_paths[i])
8     list_dcm.sort()
9
10    # start at one to ignore .DS file
11    list_dcm = list_dcm[1:num_slices[i]:]
12
13    # change list to include path of .dcm files instead of just their names
14    s=folder_paths[i]+"\{0\}"
15    list_dcm = [s.format(dcm) for dcm in list_dcm]
16
17    dcm_series.append(list_dcm)

In [ ]: 1 # convert the .dcm files to 3D images
2 images = [sitk.ReadImage(i) for i in dcm_series]

```

Figure 2.10: Jupyter Notebook Researched by Kailin Chu

Originally, the parameters entered by the users, and many other values were hard-coded in the Jupyter Notebook. To improve the usability of the AortaGeomRecon (reduce the amount of time for user inputs and execution), we implemented an extension module on 3D Slicer.

3D Slicer is an open-sourced medical image processing software for research. 3D Slicer provides useful modules such as Crop Volume module and Volume Rendering module that easily crop any volume. 3D Slicer is highly modular with Python scripting

to control the extension module sequence, and QT designer to generate Graphical User Interfaces.

3D Slicer supports modularization with an extension. An extension can compose multiple modules, where each module is dedicated to solve a sub-problem.

2.3.1 3D Slicer’s data structure

3D Slicer’s Data Structure can be divided into two categories. The Node data structure stores large data such as DICOM, with a Volume Node, Volume rendering Region of Interest Node, and Label Map Volume Node. The parameters are stored as strings from the UI component of the module. Every data stored in 3D Slicer can be accessed by the 3D Slicer’s Widget Class and Logic Class for further processing.

3D Slicer stores all the above data in a scene object, which is also referred to as a MRMLscene file, on the higher level data format. 3D Slicer can load any MRMLscene file, this allows the user to retrieve all the data nodes and parameters. In addition, 3D Slicer has a special input module for the DICOM database that allows users to store DICOM metadata.

2.3.2 3D Slicer’s scripted module

Every ScriptLoadableModule in 3D Slicer have a Widget Class and a Logic Class. The Widget Class is used to initialize the extension module’s UI component, and the parameters tied to the UI component. The module’s Logic Class is used to perform the processing of the data. In the Logic Class, we initialize an AortaGeomRecon Segmente object with the attributes set to the parameters reading from the UI component, which are inputs by the user. After completing the segmentation with

the Segmenter object, we convert the SimpleITK image object to a volume node corresponding in 3D Slicer, which allow the user to visualize the segmentation result.

2.3.3 AortaGeomReconDisplayModule

In this section, we demonstrate the implementation details of the 3D Slicer plugin AGR. We first demonstrate the module's Graphical User Interface, then discuss the module's logic and workflow.

2.3.3.1 Graphical User Interface

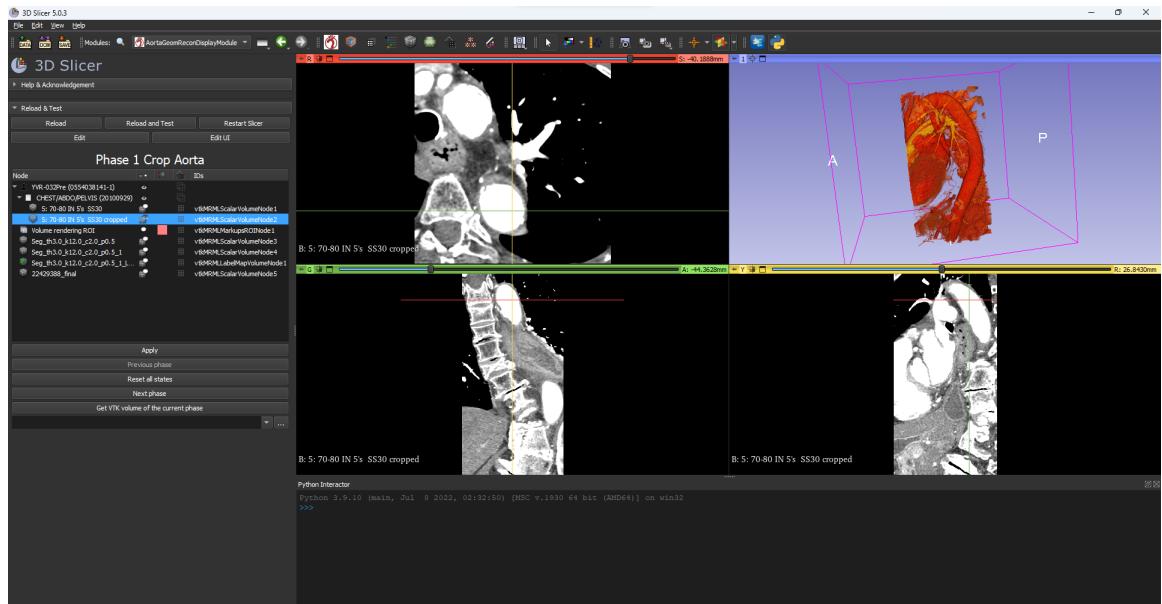


Figure 2.11: 3D Slicer UI

3D Slicer separate the UI into two parts. From Figure 2.11, we can see that the four windows on the right side of the UI are used to visualize a volume. The left side shows the SubjectHierarchyTreeView where the module stores many data nodes. The first node is the DICOM patient data with the chest CT scans stored as a volume,

and the cropped volume as generated with Crop Volume Module. There is a Volume rendering Region of Interest (ROI) node and several ScalarVolumeNode, which are the generated segmentation volume with different parameters.

The left side is the module UI. This part is designed and implemented differently based on the requirements of the modules. The Figure 2.12 shows the module UI that the AGR implements, where each parameter is stored to be passed to the algorithm class.

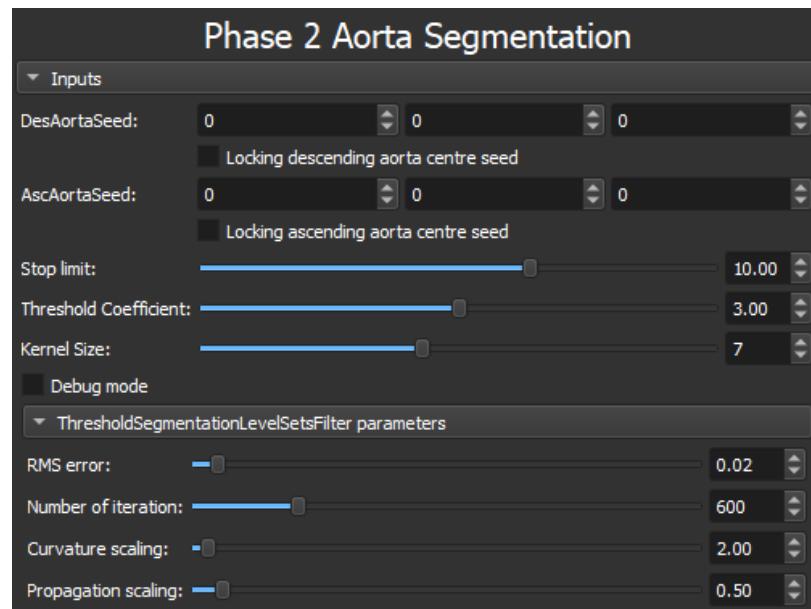


Figure 2.12: AortaGeomRecon Module UI

2.3.3.2 Module's Workflow

When the user first starts 3D Slicer and clicks on the AGR module, the warning message and tips shown in Figure 2.13 appear in the module's UI. The user must click on the confirm button below to proceed into the next steps. This warning message is a part of the evidence for the assurance case, as explained in the next

chapter.

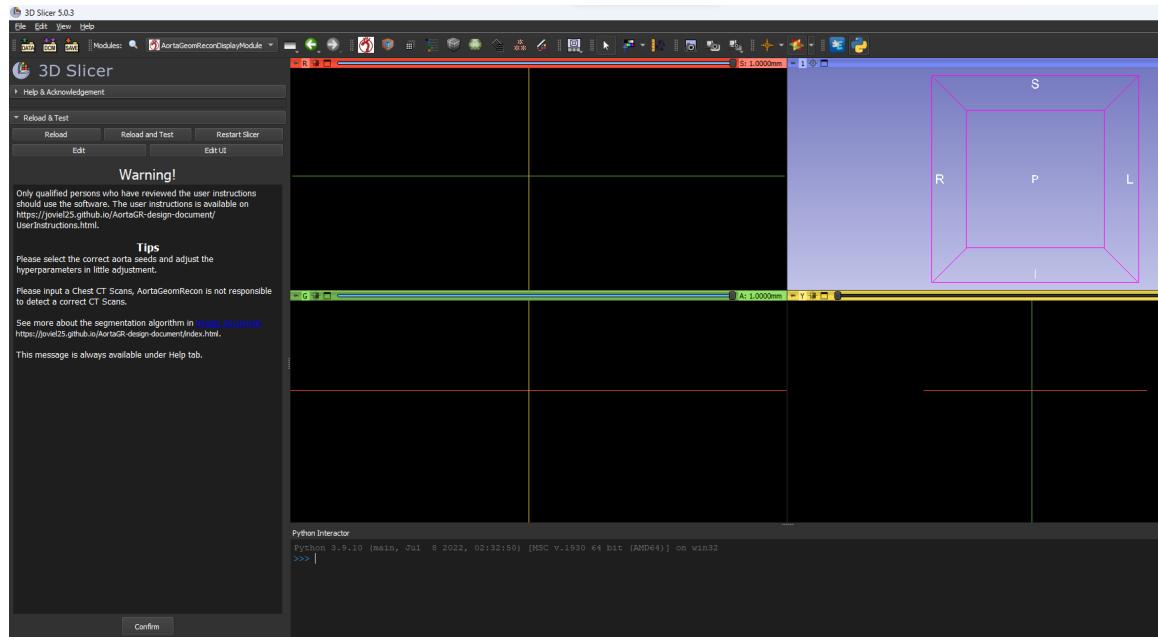


Figure 2.13: AortaGeomRecon Warning message

In the next step, assuming that the user has already read a DICOM image of the patient’s chest, the user is asked to generate a cropped volume using the 3D Slicer’s Volume Rendering module and the Crop Volume module. In this phase, the module UI displays only a SubjectHierarchyTreeView where the large data node are shown in this view. After generating a cropped volume, the apply button is enabled and the user can proceed to the next step.

In phase 2 aorta segmentation, the user is asked to input the parameters to perform the segmentation. The module UI is same as Figure 2.12. The necessary inputs are the two aorta seeds. Without any value for these two inputs, the module will not allow the user to generate a segmentation result. One of the advantages of using 3D

Slicer is the interactive UI that supports reading coordinates on the volume interactively. On the right side of the Figure 2.11, we see a crossed intersection pointing to parts of the aorta, this intersection point allows the developer to read the coordinates. Moreover, we were able to automatically pick up the coordinates in real-time in the coordinate widget. As the user moving the intersection, we get the coordinate readings. Figure 2.12 also demonstrates that the user can lock a seed, so the program stops picking up newest coordinate from the intersection point.

Chapter 3

Assurance Cases and Selected Evidence for AortaGeomRecon

This chapter discusses building the evidence to support each claim of our AC for AGR (AortaGeomRecon). We start from the top level claims of the AC developed in previous work [25] and drill down to find the evidence that can support the arguments and the top level claims. The following material is presented in this chapter as part of the evidence: the Software Requirements Specification (SRS) of AGR, the Design Document, the Module Guide, the Test Plan, the Algorithm Review, the User Manual, the User Instruction Video, AGR’s control sequence, and a Warning Message implemented in AGR.

3.1 Assurance Case Development

Assurance Cases are build with claims, subclaims, contexts and evidence. The parts of the AC add up to an argument that justifies that top level claim. By using [Astah](#)

[System Safety](#) software to present the Goal Structuring Notation (GSN) arguments [2][11], we show that our software delivers correct outputs when used for its intended use/purpose in its intended environment, and within its assumed operating assumptions. Figure 3.1 shows the top level of the assurance cases. With the goal of arguing that the software delivers correct outputs, we decompose the goal into four sub goals: GR, GI, GA and GBA, where GR stands for the goal of correct requirements of the software, GI stands for the goal of the implementation matching the requirements, GBA is the goal of that all operational assumptions have been defined, and GA is the goal that all operational assumptions are met.

3.2 Assurance Case for Software Specification Requirements

The first goal for getting trusted software is having a complete, unambiguous, correct, consistent, verifiable, modifiable and traceable SRS that shows the complete breakdown of the requirements with mathematical notation, data models and instance models. The SRS is the foundation of the software development, as the design and the implementation are created to satisfy the requirements document.

Figure 3.2 demonstrates the claims on the goal of correct requirements for AGR. On the left branch, GR_3C states the goal of complete, correct, and consistent documentation. Under this claim, the goal is separated based on each characteristic, and the corresponding evidence is presented as the leaf nodes.

As shown in the assurance case GR, one of most important pieces of evidence for the SRS having these characteristics is that it follows a standard template. Using a

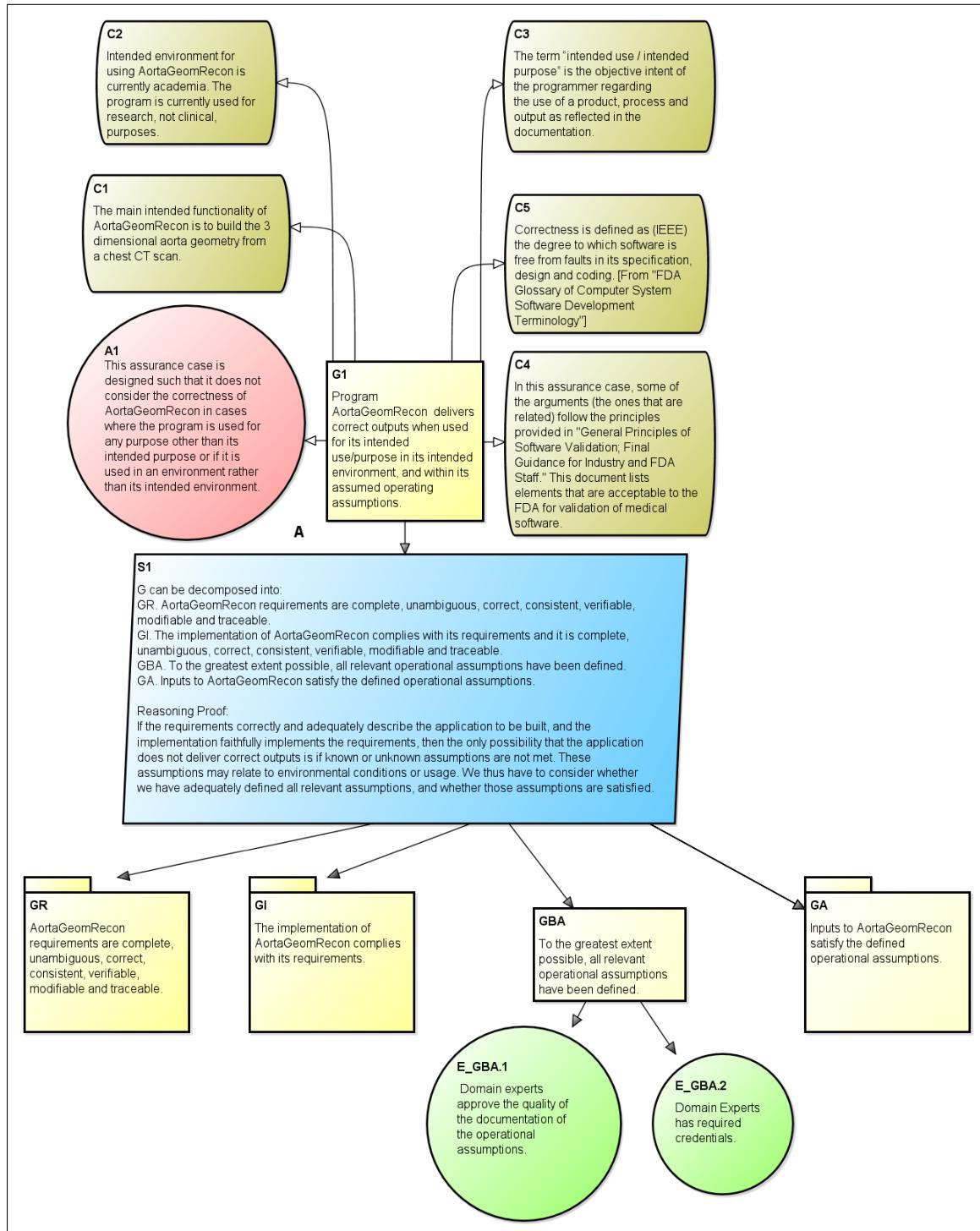


Figure 3.1: AGR Assurance Cases Top Level

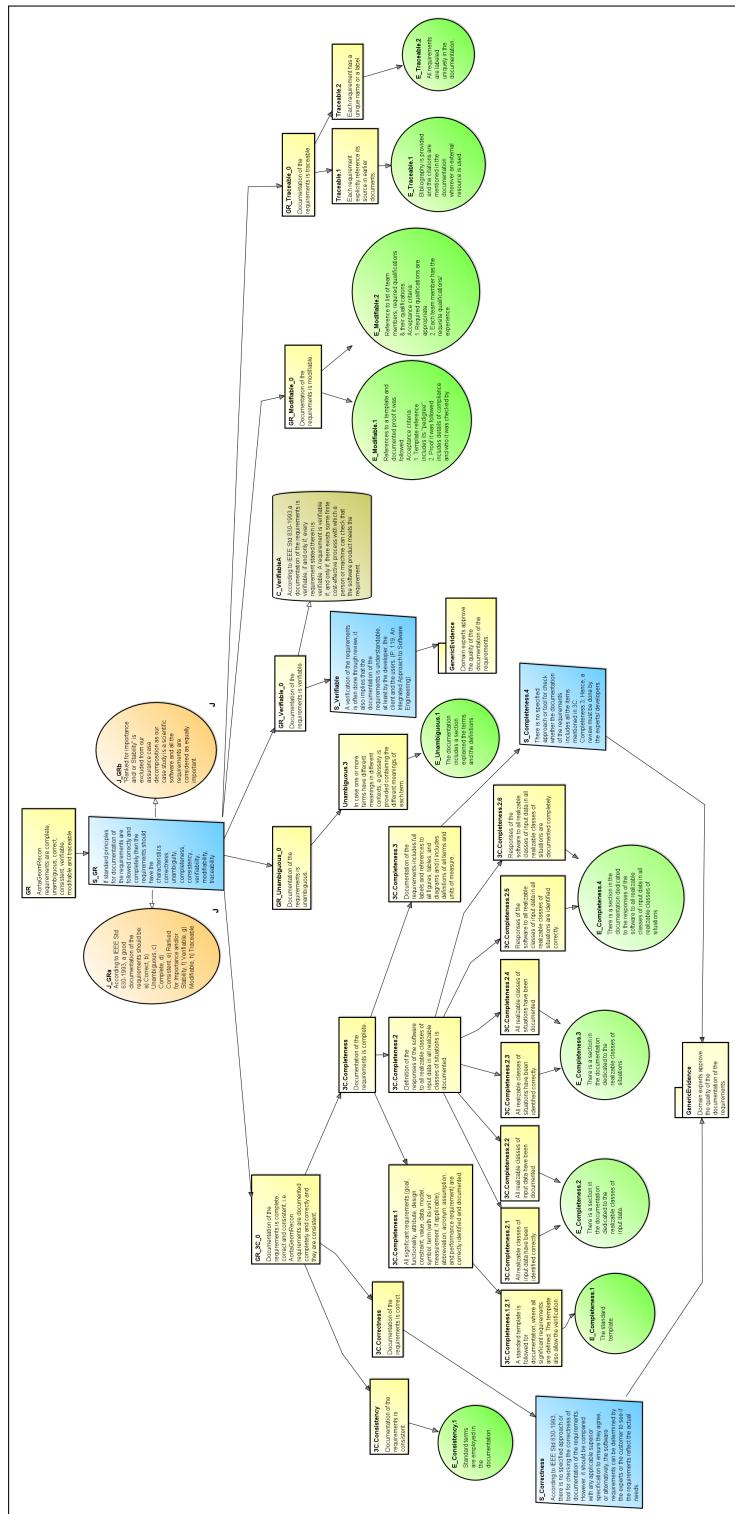


Figure 3.2: AGR Assurance Cases GR

standard template builds confidence that all necessary requirements have been defined. Moreover, standardization allows a domain expert who has used this template to verify the quality of the document. The current work uses a template tailored for research software [26], which is the standard template in the evidence E_Completeness.1 (Figure 3.2), and all other evidences where a template is included.

3.2.1 The sections of the SRS

The full SRS for AGR is reproduced in Appendix A. Excerpts from the SRS are used in the body of this report to show the document and illustrate the evidence it provides for the AC. The table of content of the SRS is provided in Figure 3.3. Some of the most important sections are explained below.

- Reference Material

In this section, a Table of Symbols and an abbreviations and acronyms table are used to explain every symbol and Abbreviations used in the document. These tables ensure the consistency and the unambiguous characteristics of the document, providing evidence for E_Consistency.1 and E_Unambiguous.1 shown in Figure 3.2. They are located at the very beginning of the document, so that the reader can first look at these tables before reading the entire document.

- Introduction

The introduction section defines the problem and the scope of the document. There are subsections for explaining the purpose of document, abstracting the scope of the requirements and defining the characteristics of the intended reader. This section is provided for the reader to reduce the chance of an unqualified reader finding the document ambiguous.

1. Reference Material
 - (a) Table of Units
 - (b) Table of Notations
 - (c) Table of Symbols
 - (d) Abbreviations and Acronyms
2. Introduction
 - (a) Purpose of Document
 - (b) Scope of Requirements
 - (c) Organization of Document
3. General System Description
 - (a) System Context
 - (b) User Characteristics
 - (c) System Constraints
4. Specific System Description
 - (a) Problem Description
 - i. Background
 - ii. Terminology Definition
 - iii. Coordinate Systems
 - iv. Physical System Description
 - v. Goal Statements
 - (b) Solution Characteristics Specification
 - i. Assumptions
 - ii. Theoretical Models
 - iii. Data Definitions
 - iv. Instance Models
 - v. Data Constraints
 - vi. Properties of a Correct Solution
5. Requirements
 - (a) Functional Requirements
 - (b) Non-functional Requirements
6. Other System Issues
7. Traceability Matrix
8. Likely Changes

Figure 3.3: AGR SRS Table Of Content

- General System Description

The general system description includes a system context diagram that explains the relationship between the users, the inputs given by the user and the outputs of the AGR program. Along with the figure, user responsibilities and AGR responsibilities are also defined in this section.

- Specific System Description

This section presents more detail about the problem and the specific system to solve the problem. The first subsection, Problem Description, discusses the definition of Organ Segmentation, the Coordinate Systems used in medical image problems, and the Goal Statements. The goal for AGR is to extract the three-dimensional segmentation of the aorta.

The next subsection, Solution Characteristics Specification, starts with the assumptions, as shown in Figure 3.4, to clearly define the scope of the requirements. The subsection Data Definitions, defines Voxel, Image/Slice, and Volume with mathematical notation so that the developer can interpret unambiguously the definitions, as shown in Figure 3.22. The Data Definitions subsection provides evidence for E_Unambiguous.1 (Figure 3.2).

Finally, the subsection Instance Models, shows the mathematical meaning of Region of Interest, as reproduced in Figure 3.6, and Segmentation in Figure 3.7. These are two essential models that the developer must know for developing their solution. The Assumptions, Data Definitions, and Instance Model sections provide evidence for E_Completeness.2, E_Completeness.3 and E_Completeness.4, which states that there is a section in the documentation dedicated to the realizable classes of input data, realizable classes of situation

and the response of the software to all realizable classes of input data in all realizable classes of situation. The Assumptions and the Data Definition sections define the realizable classes of input data. The Instance Model section defines the realizable classes of situation and the response of the software.

4.2 Solution Characteristics Specification

4.2.1 Assumptions

This section simplifies the original problem and helps in developing the theoretical model by filling in the missing information for the physical system. The numbers given in the square brackets refer to the theoretical model [T], general definition [GD], data definition [DD], instance model [IM], or likely change [LC], in which the respective assumption is used.

- A1: The 3D image provided by the user must contain a visually distinguishable aorta volume [IM1].
- A2: User should select a valid region of interest [IM2].
- A3: User should input a singular volume (3 dimensional image) even if the data format supports the 4th dimension (time) [IM1].

Figure 3.4: AGR SRS Assumptions

Number DD1	
Label	Voxel
Symbol	$v : \mathbb{R}$
SI Units	-
Equation	-
Description	A slice (DD2) consists of $n \times n$ voxels. A real number is assigned to each voxel to reports the intensity on a grey-scale image.
Sources	Nejad (2017)
Ref. By	DD2
Number DD2	
Label	Image/Slice
Symbol	$slice : \mathbb{R}^{m \times n}$
SI Units	-
Equation	-
Description	A visual representation that is using only two spatial dimensions with a sequence of arrays where a voxel (DD1) represents the color or intensity. Each move in the transverse plane (Figure 4) is considered as one slice
Sources	Nejad (2017)
Ref. By	DD3
Number DD3	
Label	Volume
Symbol	$V : \mathbb{R}^{m \times n \times p}$
SI Units	-
Equation	-
Description	A three-dimensional image is a sequence of some images/slices (DD2).
Sources	-
Ref. By	IM1

Figure 3.5: AGR SRS Data Definitions

The goals GS1 are solved by finding IM1 and perform IM2 on the aorta.

Number	IM1
Label	Region of interest
Inputs	$V_{\text{in}} : \mathbb{R}^{m_i \times n_i \times p_i}$, $Start : \mathbb{N}^3$, $m_o, n_o, p_o : \mathbb{N}$, with the following constraints: $\begin{aligned} 0 &\leq Start[0] < (m_i - 1) \\ 0 &\leq Start[1] < (n_i - 1) \\ 0 &\leq Start[2] < (p_i - 1) \\ 0 &< m_o \leq (m_i - Start[0]) \\ 0 &< n_o \leq (n_i - Start[1]) \\ 0 &< p_o \leq (p_i - Start[2]) \end{aligned}$
Output	$V_{\text{out}} : \mathbb{R}^{m_o \times n_o \times p_o}$ such that $\begin{aligned} \forall(i, j, k : \mathbb{N} \mid \\ i \in [Start[0]..Start[0] + m_o] \wedge \\ j \in [Start[1]..Start[1] + n_o] \wedge \\ k \in [Start[2]..Start[2] + p_o] : \\ V_{\text{out}}[i][j][k] = V_{\text{in}}[i][j][k]) \end{aligned}$
Description	The regions of interest is a subset (shaped like a box) of the 3D V_{out} . This subset contains the anatomical structure that the users wants to read, process or extract.
Sources	
Ref. By	IM2

Figure 3.6: AGR SRS Instance Model Region Of Interest

Number	IM2
Label	Segmentation
Input	$V_{in} : \mathbb{R}^{m \times n \times p}$, $Seed_a : \mathbb{N}^3$, $Seed_d : \mathbb{N}^3$
Output	$V_{out} : \mathbb{R}^{m \times n \times p}$ such that $\forall(i, j, k : \mathbb{N} \mid i \in [0..m - 1] \wedge j \in [0..n - 1] \wedge k \in [0..p - 1] : (V_{in}[i, j, k] \in \text{structure} \implies V_{out}[i, j, k] = HIGH \mid V_{in}[i, j, k] \notin \text{structure} \implies V_{out}[i, j, k] = LOW))$ <p>The inputs $Seed_a$ and $Seed_d$ are used to determine whether a given element of V_{in} is in structure or not.</p>
Description	The process of extract an anatomical structure from the original 3D volume. The extracted anatomical structure is represented with high intensity pixel value. The rest of the image should have a lower intensity pixel value. The segmentation needs the region of interset from IM1 to process less noise data. A seed is what the algorithm needed as the inputs to perform segmentation, the type of seed is different among different algorithm. The seeds in this section are the centre coordinate of the descending aorta and the ascending aorta. The yellow dots shown in Figure 5 are the example of the seed.
Sources	
Ref. By	R3, LC1

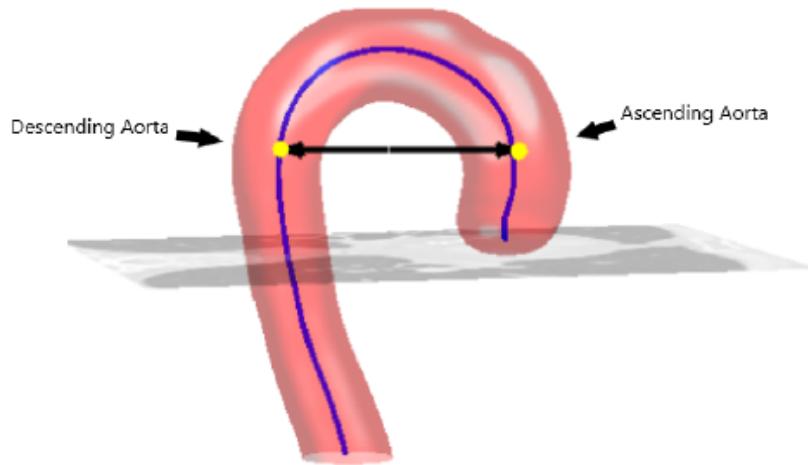
Figure 5: Aorta Seeds [Kurugol et al. \(2012\)](#)

Figure 3.7: AGR SRS Instance Model Region Of Interest

- Requirements

With all the background information provided, the Functional Requirements and the Non-Functional Requirements can be presented for AGR. The Functional Requirements are defined by using the terms given in the Data Definitions, Instance Model, and based on the other Functional Requirements, as shown in Figure 3.8. To be verifiable, the Non-Functional Requirements usually have a measurement such as execution time, or the amount of manual effort. Figure 3.9 demonstrates the Non-Functional Requirements of Usability, Safety, Learnability, Accuracy and Consistency. The Functional Requirements and Non-Functional Requirements are labeled, which provides the evidence for E_Traceable.2 (Figure 3.2).

5.1 Functional Requirements

R1: Input the following functions, data and parameters:

symbol	description
V	CT Scans volume (DD3)
$Seed_a$	The seed of ascending aorta centre coordinate (IM2)
$Seed_d$	The seed of descending aorta centre coordinate (IM2)

R2: Use the volume in R1 to create a second volume, the region of interest (IM1) that contains all voxels of the aorta.

R3: Perform segmentation (IM2) on the volume created in R2.

R4: Visualize a volume (DD3).

Figure 3.8: AGR Functional Requirements

5.2 Nonfunctional Requirements

NFR1: **Usability** AortaGeomRecon allows a user that meets the user characteristics (Section 3.2) to import any DICOM files, input the required parameters, and begin the segmentation effortlessly. The number of steps it takes using AortaGeomRecon should be at least 30% less than the number of steps it takes by using ITK-Snap (bubble method mentioned in Section 2).

11

NFR2: **Safety** For a valid image, the AortaGeomRecon provides a correct solution, or no answer.

NFR3: **Learnability** The user interface and documentation should allow a user that meets the user characteristics (Section 3.2) to learn how to do an aorta segmentation in at least 30% of the time it takes to learn and use ITK-Snap (bubble method mentioned in Section 2).

NFR4: **Accuracy** For a given image the segmentation found by AortaGeomRecon should match that found by an expert using ITK-Snap. Whether two segmentations match is something that would be judged by a medical imaging expert.

NFR5: **Consistency** The coordinate system may be modified through the calculations, but any transformations will not alter the meaning of the data.

Other NFRs that might be discussed in the future include verifiability, and reusability.

Figure 3.9: AGR Non- Functional Requirements

- Likely Changes and Unlikely Changes

This section discusses the likely changes that the developer might expect in the future, and the unlikely changes along with their justification. The only likely change discussed in the AGR’s SRS is regarding the segmentation method. For different segmentation methods, the inputs varies, since the segmentation method is a likely change, the inputs variables are also likely changes. The only unlikely change documented is the method of retrieving a region of interest. Currently, the standard approach is used of taking a starting point and sizes in different dimensions to get the region of interest.

- Traceability Matrix and Graphs

The traceability matrices provide easy references on what has to be modified if a certain component is changed. Figure 3.10 is the traceability matrix of the Data Definitions and Instance Models. Figure 3.12 shows the relationship between the requirements and other sections, and Figure 3.11 shows the relationship between the assumptions and other sections.

	DD1	DD2	DD3	IM1	IM2
DD1					
DD2	X				
DD3		X			
IM1			X		
IM2				X	

Table 2: Traceability Matrix Showing the Connections Between Items of Different Sections

Figure 3.10: AGR Traceability Matrix between Data Definitions and Instance Model

	A1	A2	A3
DD1			
DD2			
DD3			X
IM1	X		X
IM2		X	X
LC1	X	X	X
UC1			X

Table 4: Traceability Matrix Showing the Connections Between Assumptions and Other Items

Figure 3.11: AGR Traceability Matrix Between Assumptions and Other sections

	DD1	DD2	DD3	IM1	IM2	R1	R2	R3	R4	NFR1	NFR2	NFR3	NFR4	NFR5
DD1	X													
DD2	X	X												
DD3		X	X											
IM1		X	X											
IM2			X	X										
R1		X	X	X	X									
R2		X	X			X								
R3		X		X			X							
R4		X						X		X	X			
NFR1	X	X	X	X	X	X	X	X	X	X	X	X	X	
NFR2		X	X	X	X	X	X	X			X			
NFR3			X	X	X	X	X	X			X	X	X	
NFR4	X	X	X	X	X							X		
NFR5	X	X	X	X	X	X	X	X				X		X

Figure 3.12: AGR Traceability Matrix Between Requirements and Other sections

Table 4: Traceability Matrix Showing the Connections Between Requirements and Instance Models

- **Bibliography**

A bibliography is provided at the end of SRS, the bibliography points to the template [26] used for this documentation, and a list of citations for the external resources. This information is related to the evidence for E_Traceable.1, which states that a bibliography is provided to include the citations of the external resource. The template is also related to E_Modifiable.1, which states that the documentation references a template and provides proof it was followed.

3.2.2 Documentation Review

Documentation review is necessary to ensure that documentation is correct and complete. For the current project, when there is an update in the documentation, GitHub issues were used to post a documentation review request, as shown in Figure 3.13. All documents and changes were reviewed by Dr. Spencer Smith, who has the requisite qualifications/experience to review the completeness and the correctness of the documentation. Additionally, the goal of the requirements being verifiable is also reviewed by Dr. Spencer Smith. For example, he reviewed the mathematical expression and helped define the terms in Data Definitions and Instance Model correctly and unambiguously. Therefore, the documentation review provides evidence for S_Verifiable.

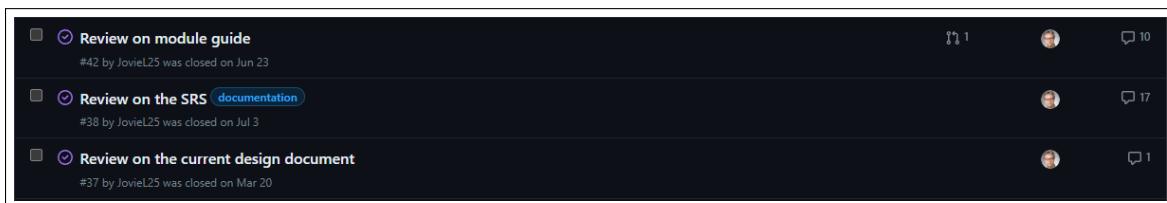


Figure 3.13: GitHub Repo Documentation Review Requests

3.3 Assurance Case for the Implementation

The goal of implementation is that it fully complies with the SRS. As Figure 3.14 shows, we argue this in two ways:

1. We argue that the implementation of the requirements has been verified.
2. We argue the design matches the requirement and that the implementation complies with the design, which together implies that the implementation matches the requirements.

This section focuses on matching the developed artifacts to the evidence shown in Figure 3.14. The first subsection discusses the test plan of the AGR, particularly the continuous integration test infrastructure, test cases and the test procedure for testing all requirements of the software. Next, this section shows the design documents, including the Module Guide, which documents system architecture, and a design document for a detailed design explanation. Finally, we present the Code Walkthrough and the Algorithm Review, which helped us to eliminate bugs, errors, and increase our confidence in the implementation’s completeness, and correctness.

3.3.1 Test Plan

GI.1 states the goal of the implementation of every requirement has been verified, so we need a test plan that is approved by a Domain Expert who has the required credentials, and the test cases cover all the Functional Requirement and Non-Functional Requirements. Unlike ITK-Snap’s bubble method and 3D Slicer’s threshold segmentation method that can interactively control the wanted pixels, our algorithm functions

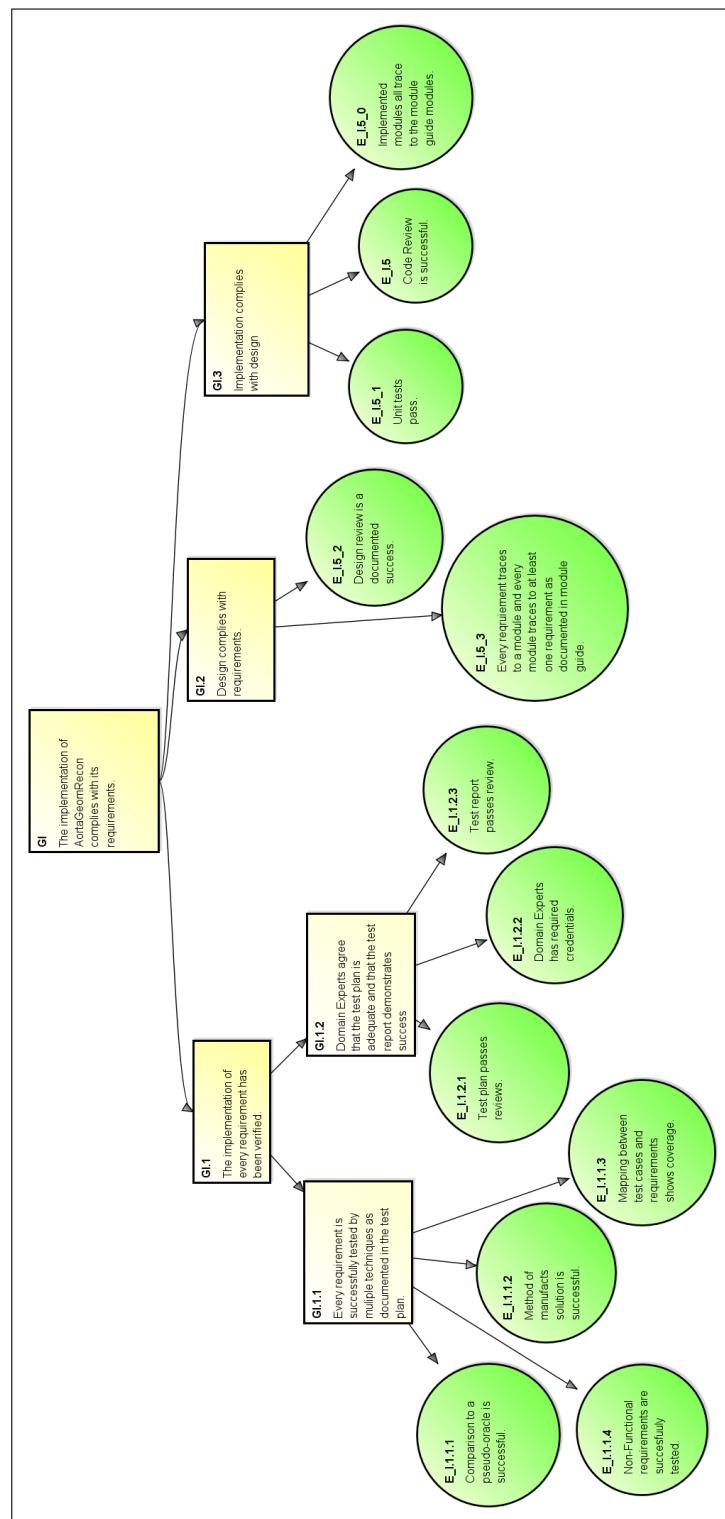


Figure 3.14: AGR Assurance Cases For Implementation

like a black box. With a given set of inputs and hyperparameters, we can only verify the result after the segmentation.

This section discusses on how “Ground Truth” was built with a verified version of the algorithm, and then compared it using Dice Similarity. Next, GitHub Actions are shown for managing the workflows, for the continuous integration infrastructure that performs static code analysis and continuous integration tests. Then, a test procedure to cover the Functional Requirements and Non-Functional Requirements is provided, as the evidences for GI.1 in Figure 3.14. Finally, the test plan approval and the test report are discussed as artifacts related to the evidences for GI.1.2.

3.3.1.1 Build “Ground Truth” Data

For the current project, we did not have the time and resource to manually build our “Ground Truth” test cases using ITK-Snap; therefore, we built the “Ground Truth” test cases using the previously verified version of the algorithm (written by Kailin) with a set of tuned hyperparameters. The process of building the test case data is described as follows:

1. Generate test case data with the previous version of the algorithm.
2. Generate test case data with the new version of the algorithm.
3. Calculate the Dice Similarity Coefficient (DSC) of the two test case data.
4. Determine whether there is a strong difference in the DSC value, and use a visualization tool, such as 3D Slicer, to see the actual difference. As long as the difference is small, the test case is assumed to pass.

The Dice Similarity Coefficient (DSC) was used as a statistical validation metric to evaluate the performance of both the reproducibility of manual segmentations and

the spatial overlap accuracy of automated probabilistic fractional segmentation of MR images [32]. The equation of DSC with boolean data, using the definition of true positive (TP), false positive (FP), and false negative (FN), is written as [30]:

$$DSC = \frac{2TP}{2TP + FP + FN} \quad (3.3.1)$$

The value of a DSC ranges from 0, indicating no spatial overlap between two sets of binary segmentation results, to 1, indicating complete overlap [32]. With a large DSC value (0.95), the evidence E_1.1.1 is achieved by comparing a new generated result with a pseudo-oracle. The statement of the evidence E_1.1.2 is also correct, which implies that a chosen approach or methodology has led to the creation of software that functions as intended, meets user requirements, and adheres to quality standards.

3.3.1.2 GitHub Actions workflows

Our Continuous Integration infrastructure was implemented with a GitHub Actions workflow. A workflow is a configurable and automated process that will run one or more jobs on the software projects when there is a commit or a pull request. GitHub Actions uses a YAML file to define the events and the commands to be executed on its temporary container [9].

For AGR there are two automated process that happens on each “push” event and “pull” event. A “push” event implies that something is changed in one or multiple commits, therefore there is a need to verify whether the commits have introduced bugs. A “pull” event happens when a feature branch is merged with the main branch. Since our main branch is protected, any update to the main branch must be merged by using a pull-request. Before a pull-request can be approved, the continuous integration

tests are examined.

The first automated process is a linter. A linter is a tool for static code analysis to flag programming errors, bugs, stylistic errors and suspicious constructs [29]. We used Python [Flake8](#) as our linter to find bugs and errors, and ensure the program's readability by enforcing the PEP8 standard.

The second automated process is our continuous integration tests. By setting up [Git Large File Storage](#) (LFS) and uploading the pre-build six ground truth tests data in the repository, the algorithm can be verified whenever a change is made. The DSC value for the comparison of the six pair of images is set to a limit of larger than 0.95. The test failed if any of the six DSC value is lower than this limit. These tests provide our evidence E_I.5_1 in Figure [3.14](#).

3.3.1.3 Test Procedure

This section introduces our test procedure for the Functional Requirements (FR) and Non-Functional Requirements (NFR).

The Functional Requirements can be tested by generating a segmentation result from scratch in 3D Slicer. Without loading a MRMLscene file on purpose, 3D Slicer is in its default state. The manual test procedure for testing the Functional Requirements is described as follows:

1. Open 3D Slicer.
2. Load a DICOM file using 3D Slicer’s DICOM database.
3. Generate a ROI object using 3D Slicer’s Volume Rendering module, as described in the user manual.
4. Generate a cropped volume using 3D Slicer’s Crop Volume module, as described in user manual.
5. Load AortaGeomReconDisplayModule, click on the “Apply” button to proceed to next step.
6. Input the aorta seeds.
7. Input the hyperparameters.
8. Click on the “Apply” button.
9. Visualize the segmentation result.

Figure 3.15: AGR Manual Test Procedure

If step 1-4 did not proceed correctly, there is an error with 3D Slicer, which is out of the scope of this project. If an error happens in step 5-7, there might be an error with the plugin as a scripted module. Otherwise, step 8 generates a result based only on the hyperparameters and the algorithm’s implementation. Step 9 is affected by 3D Slicer and the result generated from step 8. This test procedure has all Functional Requirements shown in Figure 3.8 match to a step, where FR1 matches to step 2, FR2 matches to step 3, FR3 matches to step 8, and FR4 matches to step 9. Thus, we have our evidence for E_1.1.3. The Functional Requirement, R3, is also automatically tested when a change is made by our continuous integration process described in Section 3.3.1.2.

The Non-Functional Requirements are difficult to test, because each user with different experience could result in a different learning time, for accomplishing the test of taking an image from scratch to generating a segmentation result. The test procedure used in Functional Requirement test (Figure 3.15), can be used for Non-Functional Requirements test. Both NFR1 Usability and NFR3 Learnability requires a measurement in time comparing to another software. NFR2 Safety, NFR4 Accuracy, and NFR5 Consistency requires a verification in the segmentation result. NFR4 Accuracy also requires a segmentation result generated by ITK-Snap. Although completing the NFR tests is out of the scope of the current work, once these tests are performed all Non-Functional Requirements will be tested, which is evidence for E_1.1.4.

3.3.1.4 Test Plan Approval and Test Report

GI.1.2 states that a domain expert with required credentials must review the test plan, and the test report. The test plan is reviewed by Dr. Spencer Smith, and our test report is partially automatically generated by GitHub Actions workflows, as shown in Figure 3.16.

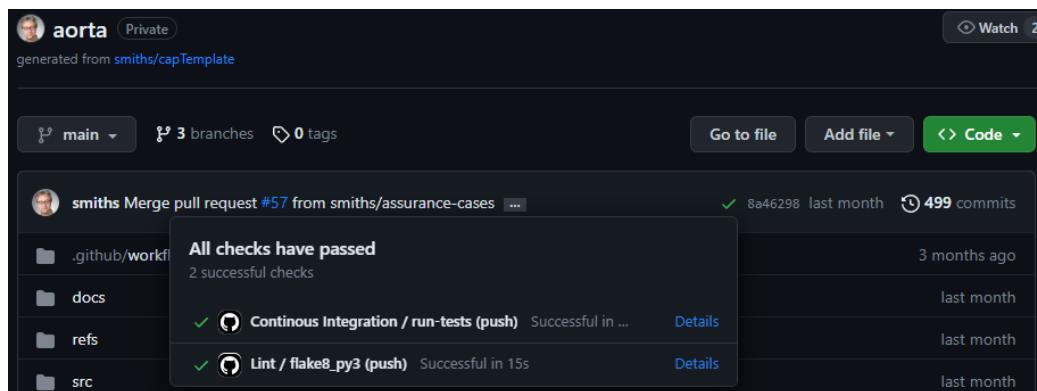


Figure 3.16: AGR Test Report

GitHub will email all the contributors when checks are not successful, so that the problem can be quickly addressed.

3.4 The Design Documentation of the AortaGeom-Recon

This section discusses the design documents of AortaGeomRecon. There are multiple aspects of the design document to discuss:

1. System architecture, or high levels design.
2. Detailed design explaining how the algorithm works.

The first item is presented with a Module Guide, and the second item is presented with a source code documentation generator, which generates the HTML documentation hosted on the web server.

3.4.1 Module Guide

An important document to show that the design is complete, correct, and consistent is the Module Guide (MG), which is attached as Appendix B. As explained previously, MG demonstrates the system architecture, or the high level design of the AortaGeom-Recon. MG is a hierarchically-structured document, intended to allow both designers and maintainers to easily identify the parts of the software that they must understand, without reading irrelevant details about other parts of the software [20].

The designer typically keeps these anticipated changes isolated to a single module so that if the change happens, only one module is impacted. The anticipated changes for AGR are listed in Figure 3.17 below.

- 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.

Figure 3.17: AGR Anticipated Changes

Modules are decomposed according to the principle of “information hiding” proposed by Parnas et al. (1984) [20]. 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. The module hierarchy and a part of the module decomposition is shown in Figure 3.18 and Figure 3.19. 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. AGR means the module will be implemented by the AGR software.

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
- M11:** Digital Enhancement Module

Figure 3.18: AGR Modules

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' attributes](#)

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

Figure 3.19: AGR Module Decomposition Example

Traceability matrices show the relationships between the modules and the anticipated changes, and between the modules and the requirements, as shown in Figure 3.20. The matrix shows all requirements are mapped to module, which is part of the evidence for E_I.5_3, under the GI.2 in Figure 3.14.

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

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

Table 2: Trace Between Requirements and Modules

Table 3: Trace Between Anticipated Changes and Modules

Figure 3.20: AGR Modules Traceability Matrices

On top of relating the modules to the requirements, the actual source code is also mapped to the modules, which is strong evidence of our implementation complying with the requirements, as shown in Figure 3.21. Module 11, Digital Enhancement Module is an example of Module mapping to a piece of the source code. The comments on top indicate the source file where this piece of the source code is located, as well as the exact places within GitHub with the exact lines highlighted. This table demonstrates that the implemented modules all trace to the module guide modules, as stated in evidence E_I.5_0.

M6 Volume Visualization Module	3D Slicer's Volume Rendering Module 3D Slicer's Volume Rendering Source Code
M7 Crop Volume Module	3D Slicer's Crop Volume Module 3D Slicer's Crop Volume Module Source Code
M8 Aorta Segmentation Module	AortaSegmenter class
M9 Image Processing Module	SimpleITK
M10 Multi-Dimensional Array Processing Module	NumPy
M11 Digital Enhancement Module	<pre># AortaGeomReconDisplayModule.py # https://github.com/smiths/aorta/blob/main/src/SlicerExtension/ # AortaGeometryReconstructor/AortaGeomReconDisplayModule/ # AortaGeomReconDisplayModule.py#L739-L769 def transform_image(self, cropped_volume): """ Histogram Equalization for Digital Image Enhancement. https://levelup.gitconnected.com/introduction-to-histogram- equalization-for-digital-image-enhancement-420696db9e43 """ cropped_image = sitkUtils.PullVolumeFromSlicer(cropped_volume) img_array = sitk.GetArrayFromImage((sitk.Cast(sitk.RescaleIntensity(cropped_image), sitk. sitkUInt8))) histogram_array = np.bincount(img_array.flatten(), minlength=256) num_pixels = np.sum(histogram_array) histogram_array = histogram_array/num_pixels chistogram_array = np.cumsum(histogram_array) transform_map = np.floor(255 * chistogram_array).astype(np.uint8) img_list = list(img_array.flatten()) eq_img_list = [transform_map[p] for p in img_list] eq_img_array = np.reshape(np.asarray(eq_img_list), img_array. shape) eq_img = sitk.GetImageFromArray(eq_img_array) eq_img.CopyInformation(cropped_image) median = sitk.MedianImageFilter() median_img = sitk.Cast(median.Execute(eq_img), sitk.sitkUInt8) self._cropped_image = median_img</pre>

Table 4: Trace Between Modules and Code

Figure 3.21: AGR Part Of The Traceability Matrix On Modules And Code

3.4.2 Detailed Design Document

The purpose of the Design Document [15] is to explain in detail how the algorithm works, and why it works. Similar to Section 2.2.3, the design document explains in plain text the workflow of the algorithm. The design document is a piece of evidences that demonstrates unambiguity. Moreover, this document can let the domain expert do a design review without reading the source codes directly, which helps to build evidence E_I.5_2.

To show and automate the detailed design, we used Sphinx, a Python Documentation Generator that can build a module's documentation from the comments in the source code. Moreover, using reStructuredText to write the Algorithm Overview, we can build HTML code which can be published on a web server, as shown in Figure 3.22, which shows the index page of the [website](#).

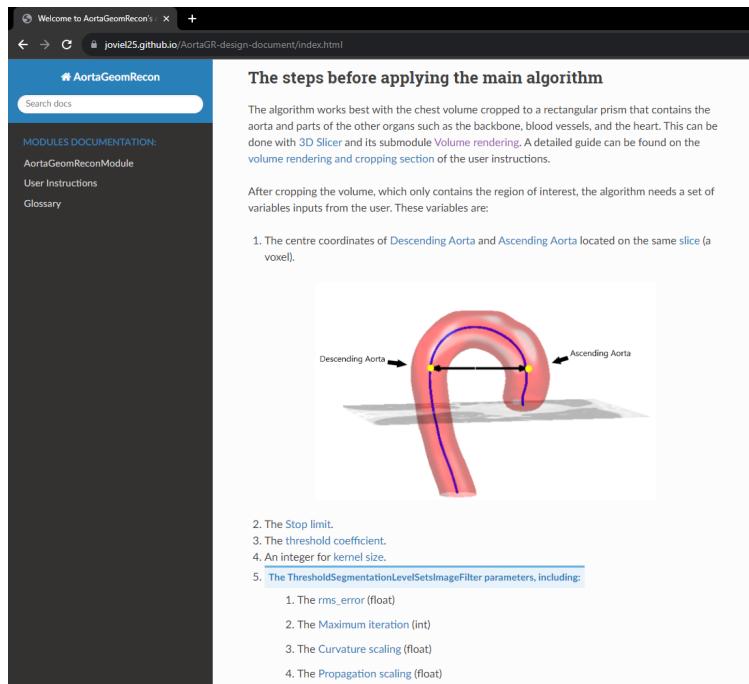


Figure 3.22: AGR Design Document Website

Another important section in the detailed design document is the Glossary. It provides a rich vocabulary of explanations, images, and links to the outside source to let the reader understand as much as possible. An excerpt from the Glossary is shown in Figure 3.23.

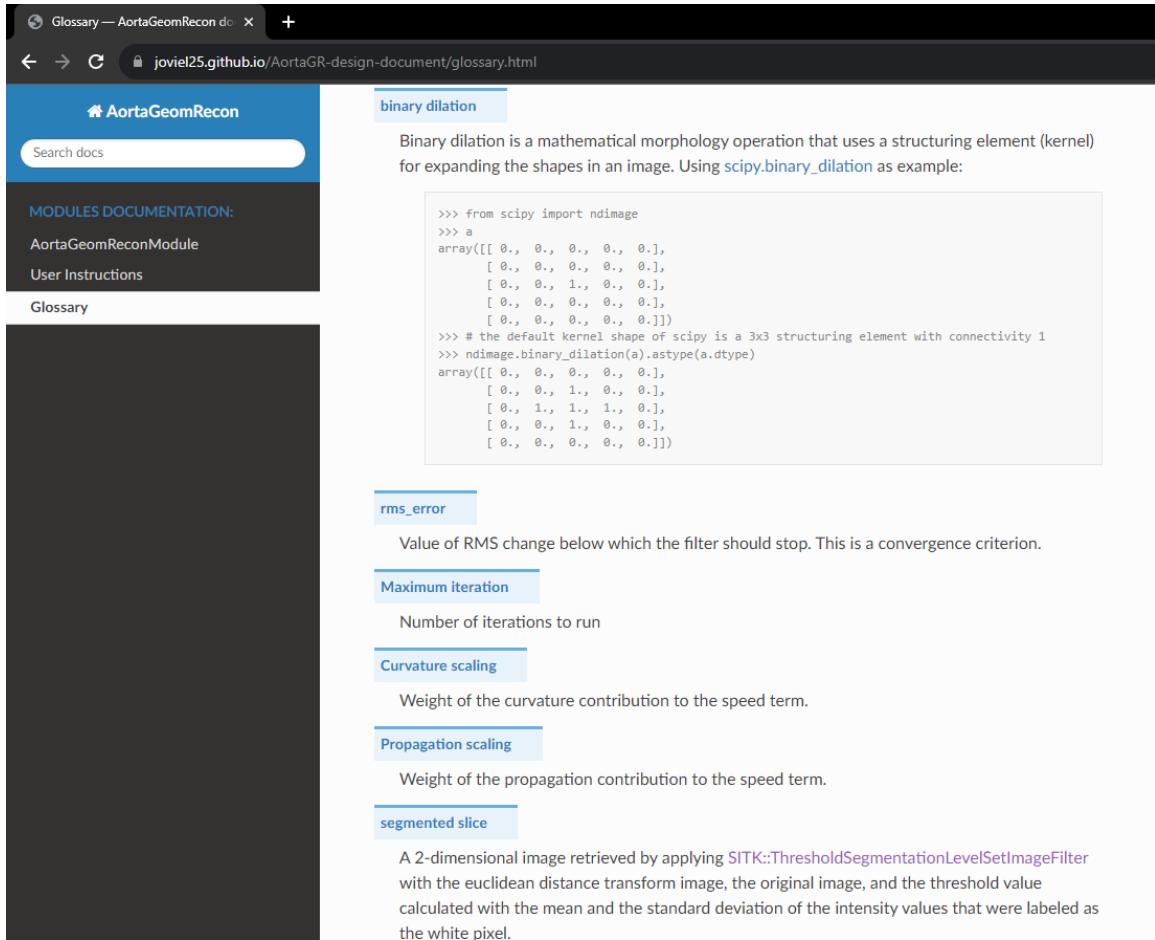


Figure 3.23: AGR Design Document Glossary

3.4.3 Algorithm Review

The Algorithm Review started with a Code Walkthrough. A Code Walkthrough is a systematic and collaborative process in software development where a team of developers, designers, and stakeholders review and analyze a piece of code, typically with the aim of identifying defects, potential issues, and improvements [5]. During a Code Walkthrough, participants examine the code line by line, discussing its design, functionality, readability, maintainability, and adherence to coding standards. The process involves both the author of the code and other team members, fostering knowledge sharing and collective learning. The goal is to catch issues and enhance the codebase through collective expertise.

We contrast a Code Review with an Algorithm Review. In an Algorithm Review, we present the algorithm to the domain expert and asking them if the detailed design fulfill the implementation objectives. In a Code Review, we are inspecting the implementation and verifying if the implementation has followed the design.

In this section, we will discuss the Code Review with Kailin Chu, and the key takeaways from this meeting. Then, we will discuss the Algorithm Review with Dr. Dean Inglis, which was a meeting to reinforce our confidence in the design. Finally, we will briefly introduce the tools that we have used in both meetings.

3.4.3.1 Code Review with Kailin Chu

The Code Review was done with Kailin Chu, who is a biomedical engineering student at McMaster University. She produced the first version of the semi-automatic aorta segmentation algorithm as a summer researcher in 2021. The code review meeting took place on Thursday, April 20, 2023. The duration was about an hour. Along with

Dr. Smith Spencer, we were aiming to increase our confidence in the code via a Code Walkthrough. The code walkthrough did not increase our confidence in the software because we quickly realized that an hour was not enough time to step through even a representative section of the code line by line. We quickly switched the purpose of the meeting to be a code review, so that we could understand the rationale behind Kailin’s original code. This part of the meeting was only partially successful, because the code was developed by Kailin two years prior, so some details and design decisions had been forgotten. We also learned that some parameters were not set by theoretical considerations, but rather by trial and error. Despite that the meeting did not achieve its original goal, this meeting was still very helpful, because it enabled algorithm improvements. Learning that parts of the algorithm were based on trial-and-error freed me to consider changing it.

Originally, my method to generate a label image (Section 2.2.4.1) was not flexible to adjust its range based on the test case. This meant the segmentation on the aortic arch requires the algorithm to generate a label image to cover part of the descending aorta. However, this coverage is missing when executing on certain test cases, because the label image is generated using fixed index. The code review with Kailin provided the revelation that this part of her algorithm was partially based on trial and error. Learning that the original algorithm are partially arbitrary gave me the confidence to improve it using the idea of centroids. By using two centroids, one centroid on the ascending aorta and another on the descending aorta, the centroids can keep track of the most centered position of ascending aorta and descending aorta even when the aortic arch region is reached. Thus, it generates a better label image, and generates a more accurate segmentation result. Moreover, the execution time is reduced with the

improved algorithm, because the improved algorithm only needs to perform one scan of the input volume. The original segmentation algorithm requires two scans of the input volume, each scan segments the descending and ascending aorta separately.

3.4.3.2 Algorithm Review with Dr. Dean Inglis

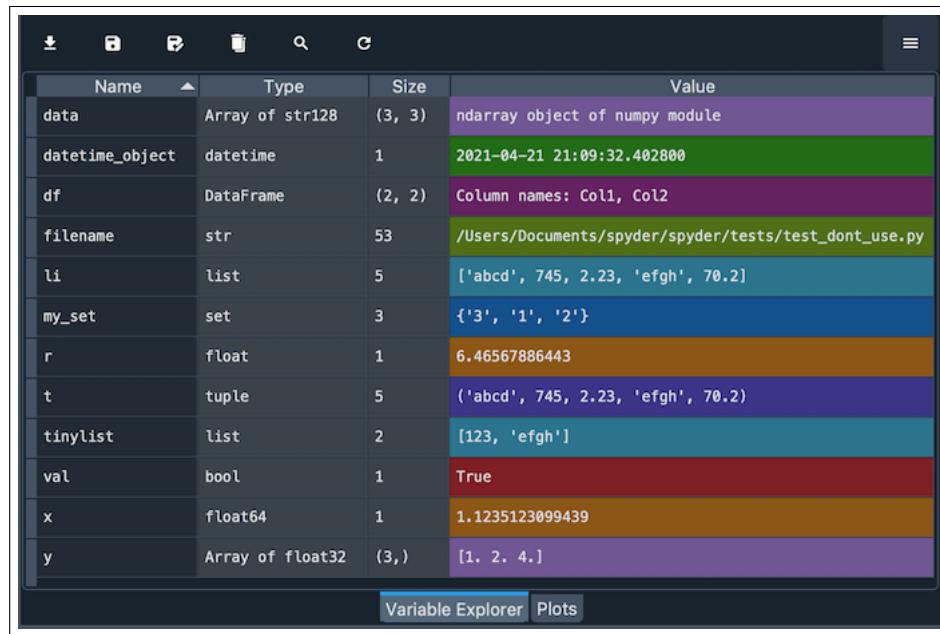
Since the experiment with Kailin Chu was only partially successful, we met with another reviewer. Rather than a code walkthrough or code review, we focused on the big picture of reviewing our algorithm. An algorithm review was conducted with Dr. Dean Inglis, an experienced professor, Medical Image Analyst, and Software Developer. This meeting happened on May 17, 2023, with a duration of about one and half hours. We presented our segmentation algorithm to him and requested validation of our approach, or suggestions for a potentially superior algorithm. Dr. Inglis provided his insights on the algorithm. These insights were meticulously recorded on the GitHub issue tracker. These insights will guide the developer responsible for enhancing the program.

This meeting significantly reinforced our confidence in both the software and our endeavors, because the methods discussed in the Section 2.2.4.1, 2.2.4.2, and 2.2.4.3 were very common in image analysis. We also learned that the level sets segmentation is also an often used technique for image segmentation. This indicates that the evidence E.I.5 is accomplished.

3.4.3.3 Tool used in Algorithm Review

Spyder is a free and open source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. The Variable Explorer

shown in Figure 3.24 allows the user to interactively browse the variables and the objects in debugging mode [23].



Name	Type	Size	Value
data	Array of str128	(3, 3)	ndarray object of numpy module
datetime_object	datetime	1	2021-04-21 21:09:32.402800
df	DataFrame	(2, 2)	Column names: Col1, Col2
filename	str	53	/Users/Documents/spyder/spyder/tests/test_dont_use.py
li	list	5	['abcd', 745, 2.23, 'efgh', 70.2]
my_set	set	3	{'3', '1', '2'}
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
tinylist	list	2	[123, 'efgh']
val	bool	1	True
x	float64	1	1.1235123099439
y	Array of float32	(3,)	[1. 2. 4.]

Figure 3.24: Spyder Variable Explorer [23]

This feature allows us to execute the program step by step, and see what happens to the variable (segmentation result) when executing the segmentation algorithm. This was very valuable for our code walkthrough/review meeting.

3.5 Assurance Case for Operational Assumptions

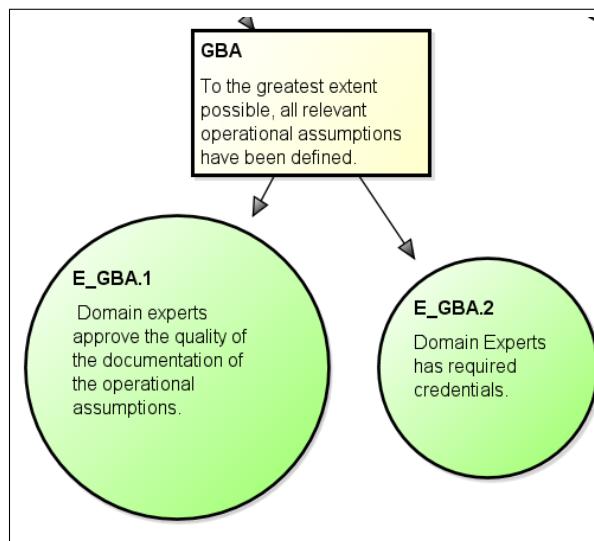


Figure 3.25: AGR Assurance Case Operational Assumptions

The evidence for the statement “To the greatest extent possible, all relevant operational assumptions have been defined” is quite simple as all that is required is a qualified Domain expert to approve the quality of the documentation. However, finalizing this evidence takes significant effort from the beginning of the project to the end of the project, because of the need to continuously improve the quality of the content matching the most recent updates of the software.

In this section, we will present two methods to define all relevant operational assumptions. The first method is a User Manual, which is written in plain text and multiple screenshots. The second method is a User Instructional Video, which includes voice over to guide the user step by step.

3.5.1 User Manual

A user manual serves the purpose of documenting all operational assumptions. When the user gets unexpected results by using this software, they should be able to refer to the user manual to see what pieces are different. Our user manual is initially located in GitHub repo's README, as shown in Figure 3.26, which is only available to the developers invited as the GitHub project contributors. The content includes the installation of the software, importing the extension modules, import inputs data, and perform segmentation. The user manual is also available publicly on the [design document website](#), for users that are not the repository contributors.

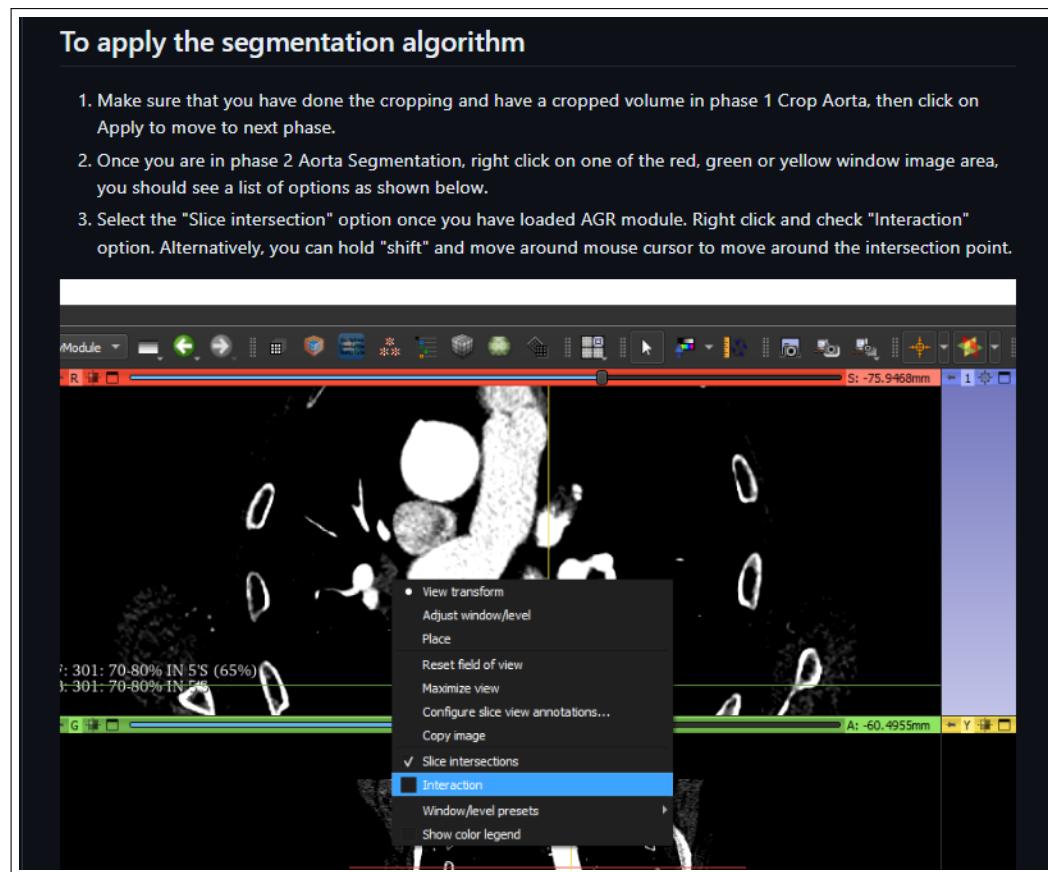


Figure 3.26: AGR User Manual On GitHub README

3.5.2 User Instruction Video

Videos are an effective way to engage your audience and deliver information in a way that's easy to follow along and understand. A better instructional content is a YouTube Video where we make step-by-step instruction with voice over to instruct the user. Figure 3.27 shows a snapshot of video on YouTube. The video is not listed publicly on YouTube, but the users who have access to the GitHub repository or Design Document website can access this video by the [URL link](#).

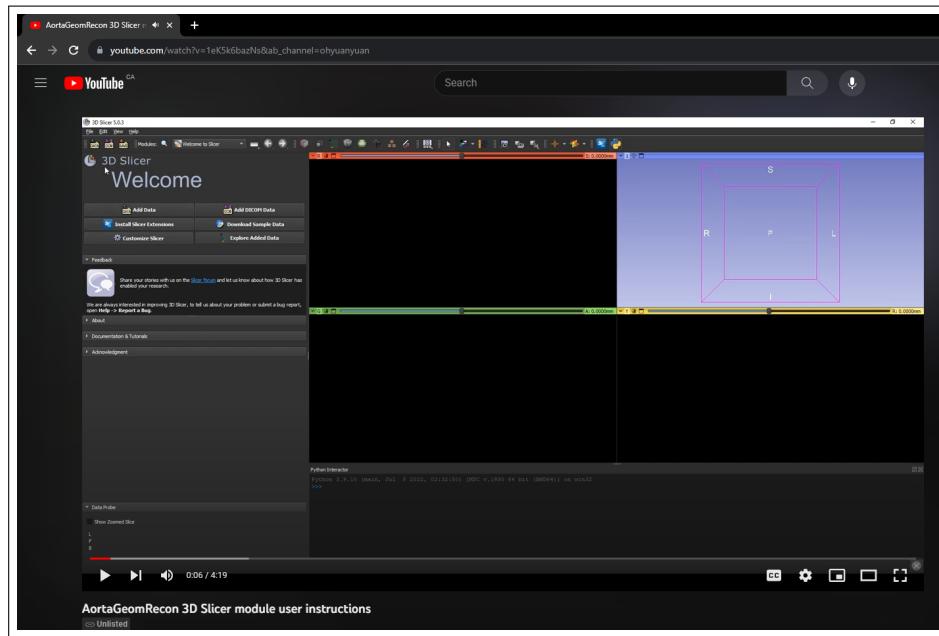


Figure 3.27: AGR User Instructions on YouTube

3.6 Assurance Case for Inputs Assumptions

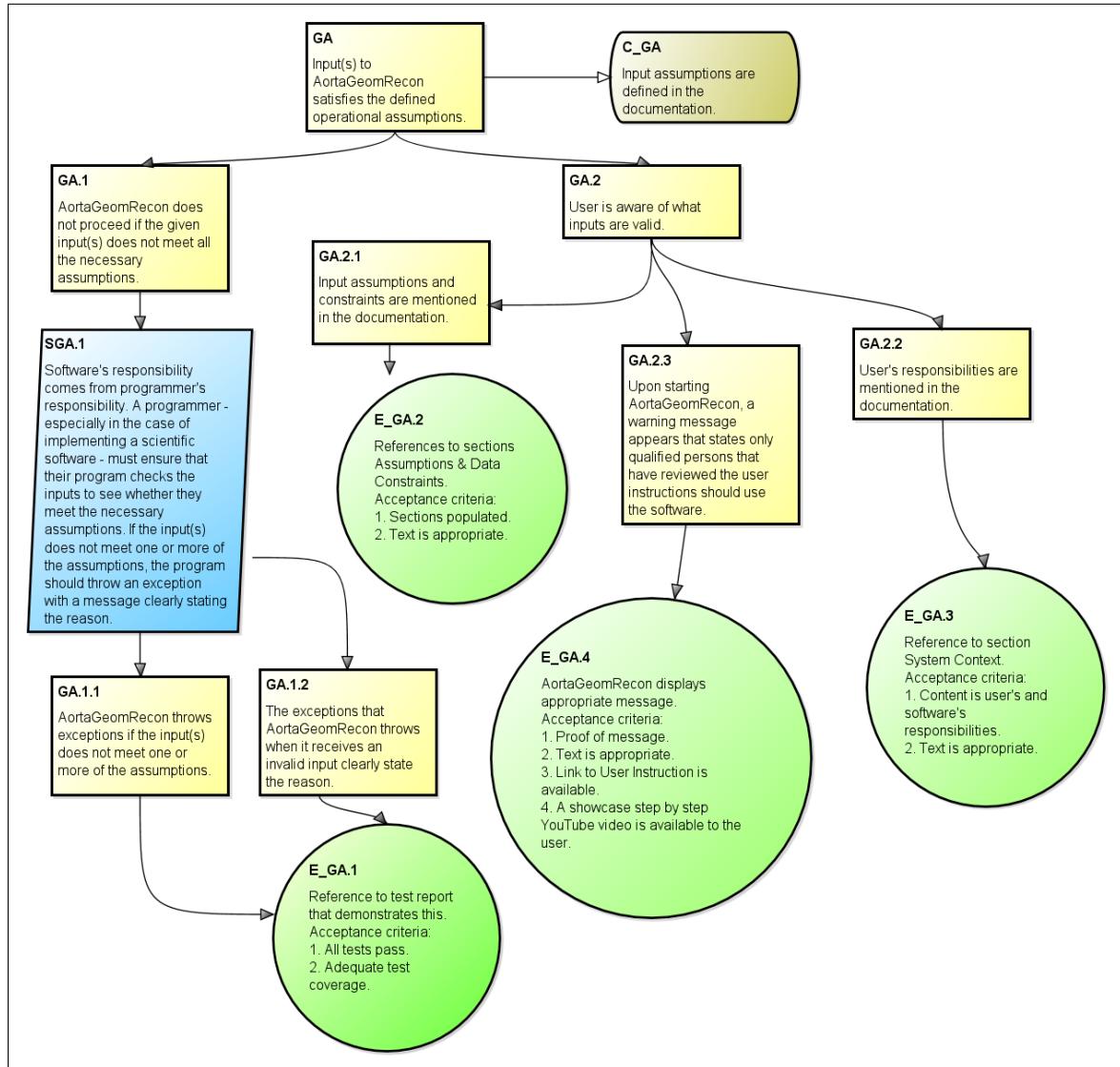


Figure 3.28: AGR Assurance Case Inputs Assumptions

Figure 3.28 shows our last assurance case, GA. This statement requires the user know what inputs are valid, and only uses the valid inputs in the software. The user documentation described in sections 3.5.1 and 3.5.2 communicate this information to

the user. When the software gets unexpected inputs, it should not proceed to the next step, which could result in unexpected results.

3.6.1 AortaGeomRecon’s Control Sequence

In the logic of the control sequence implemented as the 3D Slicer scripted module, the appropriate inputs must meet the necessary assumptions before proceeding to the next step. In phase one, a cropped volume with a name that includes the string “cropped” must be present in the node storage, where a cropped volume created by Crop Volume module is automatically named with the string “cropped” as part of the volume’s name. Otherwise, the user cannot go to the next phase through normal operation. In phase two, the aorta seeds must be provided to continue to the segmentation. Although it does not directly provide evidence, it will help passing the test mentioned in the test report, as discussed in E_GA.1 in Figure 3.28.

3.6.2 AortaGeomRecon’s Graphical User Interface

AortaGeomRecon’s Graphical User Interface, discussed in Section 2.3.3.1, reduces the problems with user input by limiting the user’s ability to enter any value possible. The hyperparameters shown in Figure 2.12 have a minimum and maximum value limitation. This is an improvement to the prior Jupyter Notebook where the users have as much freedom as they want to modify any hard-coded parameter in the code. Similarly, the implementation of GUI does not provide evidence, but it will help to build the evidence for E_GA.1 discussed in Figure 3.28.

3.6.3 Warning Message

As we initially planned, the references to sections Assumptions, Data Constraints, and System Context is available in the User Manual and User Instruction Video, where we showed the user how to import DICOM patient's data, and operate on the inputs' data to get a segmentation result. This implies that the requirements of the evidences E_GA.2, E_GA.3 and E_GA.4 are met. A user who has read the User Manual and watched the instruction video should know what inputs are valid. Therefore, in the AGR module, we need to effectively guide the user to the User Manual, whether the user has used this software before or is a first time user.

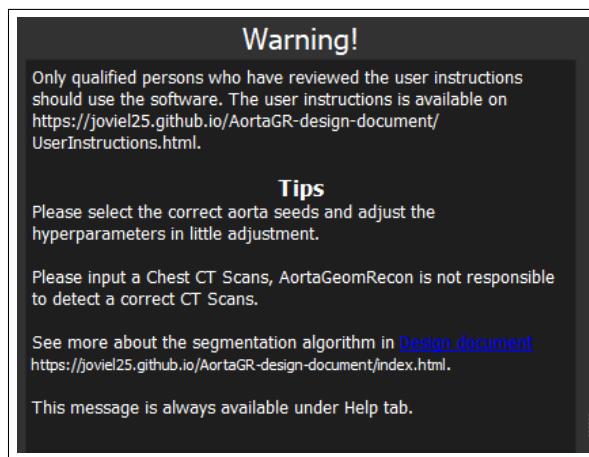


Figure 3.29: AGR Warning Message

As mentioned in Section 2.3.3.2, when the user first starts 3D Slicer and click on the AGR module, the warning message in Figure 3.29 appears. This is the appropriate message as stated in E_GA.4. The user must clicks on the Confirm button to continue to the next steps. With the warning message shown to the user, it is now the user's responsibility to use the valid inputs for AGR, so that the program will deliver the correct outputs if the other operations are performed correctly.

Chapter 4

Conclusion and Future Works

This chapter provides a summary of the thesis (Section 4.1), the challenges (Section 4.2), and the future work (Section 4.4).

4.1 Thesis Summary

This project developed software as a 3D Slicer extension to semi-automatically extract a 3D geometry of the aorta. To build confidence in the software, we applied assurance case arguments. The project started from a Jupyter Notebook program as left by a previous student. With this as a starting point, we explored what changes to the documentation, design, implementation and verification activities are necessary for the assurance case. We did the following tasks in the chronological order for the evidence supporting our assurance case:

1. Build the continuous integration infrastructure with GitHub Actions for the algorithm. This allows us to update the algorithm, and making sure that the valid update is at least as good as the previous version.

2. A linter is set up as part of the continuous integration test to ensure the program's readability by enforcing the PEP8 standard.
3. Draft our Software Requirements Specification (SRS).
4. Draft our high-level design Module Guide (MG).
5. Build a graphical user interface (GUI) because the existing approach had poor usability since it required using other software to determine the necessary parameters and then editing the code.
6. Use 3D Slicer as the platform to implement our GUI because it is modular, and it provides useful features such as Volume Rendering, volume visualization, Crop Volume and reading coordinate on a volume interactively.
7. Build assurance cases in Goal Structuring Notation with the bottom-up approaches. We gather our existing evidence, and explore new implementation requirement for the new evidence.
8. Write user instructions on GitHub README to instruct the user on how to use AGR correctly.
9. Film and post a user instructions video with voice over to provide a clear and direct user instructions.
10. Build a detailed design document and hosted on a web server.
11. Scheduled Code Review with Kailin Chu.
12. Scheduled Algorithm Review with Dr. Dean Inglis.
13. Finalize our SRS, MG, and assurance case.

It is worth mentioning that GitHub Issues, Discussions, and Pull requests are used throughout the development of the software for the project management. Dr. Spencer Smith and I were able to keep up good communication through the use of

the GitHub features. Finally, weekly and bi-weekly meetings were scheduled to help us communicate efficiently.

4.2 Challenge

In the course of this project, we have summarized a list of challenges. The first challenge was looking for an ideal platform to develop AortaGeomRecon software. At the beginning of the project, we wanted to develop our own software without any external platform. Until the point where we saw that it is not feasible to build from scratch a volume visualization system like the one provided by 3D Slicer, time and efforts were wasted in designing a UI, finding the right tool to build the UI, etc.

The second challenge is that 3D Slicer itself is a very complex software, the development resource is limited and difficult to understand. Some features provided by 3D Slicer are not obvious until you have searched through its built-in module list, which has approximately 30 modules.

Another obstacle that we had is having a domain expert to examine the quality of our segmentation result. This medical software's intended user is a university student studying in medical science or medicine, who needs to get an aorta's image or quantified volume. Throughout the development of the AortaGeomRecon, we did not have an intended user or a domain expert to review our software. However, Dr. Spencer Smith and I were also lacking the knowledge and do not know the expectation of the intended user. This causes ambiguity to specify the true requirements of AGR, until we were able to spend time with Dr. Inglis.

Finally, it was very challenging to understand assurance cases within a limited amount of time. Gathering the evidences and supporting our arguments was not in

my imagination at the beginning of the project. Without truly understanding our goals for the project, I was not certain what I was really doing. Once we have several pieces linking together, I was able to finally understand and make my efforts in the right direction.

4.3 Lessons Learned

After reviewing all available evidence and constructing the assurance case for Aorta-GeomRecon, we have compiled a list of lessons learned that could prove valuable to those seeking to develop assurance cases for research software.

When crafting the Software Requirement Specification document, it is advisable to describe our data definitions and instance models using commonly accepted mathematical notation. Since our target audience includes domain experts who may not be software developers, it is crucial not to represent instance models using pseudocode. Utilizing mathematical notation also facilitates the review process for domain experts, ensuring completeness, correctness, and consistency within the document.

The traceability matrix offers a transparent means for illustrating the relationships between various levels of abstractions. First and foremost, the traceability matrix enables other developers to verify whether the modules outlined in the Module Guide genuinely serve a purpose in fulfilling specific requirements. Furthermore, by establishing a correspondence between the modules and the source code, it allows for the verification of whether the implementation indeed provides the functionality described in the modules within the Module Guide.

Despite all the operational assumptions detailed in the documentation, it is essential to include a warning message to caution users about the proper use of this

software. This is a requirement for all critical software, including medical image processing software, as diagnoses could rely on the generated results, potentially impacting a patient’s well-being. When the software can only guarantee expected output under specific assumptions, we trust users not to violate these assumptions, and we have a responsibility to remind them of their duties.

Finally, we have learned how to conduct a code walkthrough, a code review, and an algorithm review to verify the implementation’s correctness. To start this process effectively, it’s advisable to conduct a code walkthrough alongside a software developer who recently worked on the same segment of the implementation. This ensures that the collaborator still holds a fresh memory of significant software decisions and expected output, allowing for meaningful discussions about how the implementation achieved these results.

Code and algorithm reviews have quickly provided insights into which portions of the implementation can remain untouched and where improvements can be made. During our code review with Kailin Chu, we identified a step in the segmentation algorithm implemented through trial and error. Collaboratively, we brainstormed ways to enhance this step, resulting in a significant improvement in our segmentation algorithm. In our algorithm review with Dr. Dean Inglis, he shared industry-standard methodologies for image segmentation and introduced a new segmentation method, which can reduce the execution time and eliminate some hyperparameters when applied to our algorithm.

4.4 Future Work

In this section, we discuss some possible future work that can make AortaGeomRecon better. The first improvement can be done in the segmentation algorithm, and the second improvement involve completing our assurance case.

4.4.1 Segmentation Algorithm

In this section, we discuss the possible future work related to the segmentation algorithm. Section 4.4.1.1 discusses a similar segmentation algorithm that requires fewer hyperparameters. Section 4.4.1.2 discusses a method that eliminates the use of a verification step described in our segmentation algorithm.

4.4.1.1 3D Level Set Approach

An improved segmentation algorithm [14] is available. Like our segmentation algorithm, this algorithm also needs a cropped volume and the aorta seeds to perform segmentation. However, it requires fewer hyperparameters, such as the parameters for SimpleITK’s ThresholdSegmentationLevelSetsFilter. Using this algorithm will reduce the problem of user inputs, which will help to build the evidence for E_GA.1 discussed in Figure 3.28.

4.4.1.2 Oblique Plane Segmentation

This algorithm is proposed by Dr. Dean Inglis discussed during our Algorithm Review meeting. This algorithm is based on the assumption that a healthy aorta should have reasonably uniform diameter along its length. With an initial center coordinate seed and the corresponding diameter, the algorithm can segment the aortic arch through

an Oblique Plane. Figure 4.1 shows the double oblique plane in the ascending aorta, and aortic arch.

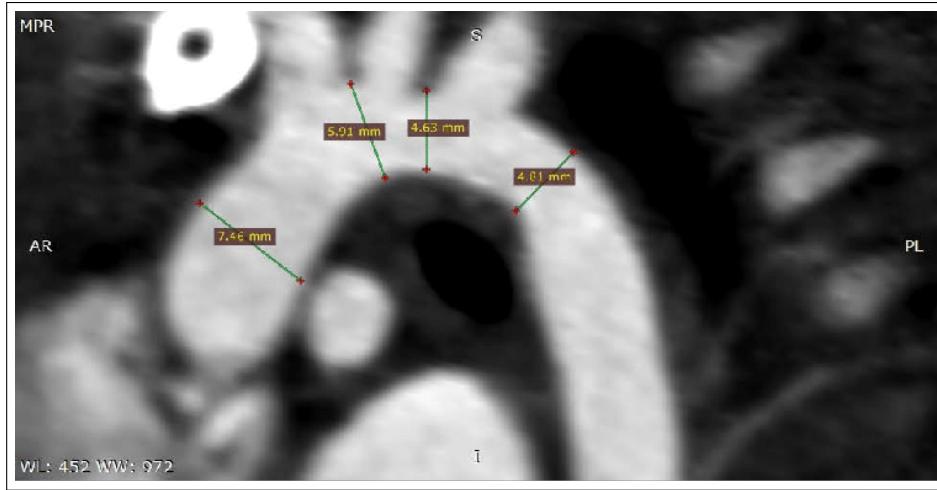


Figure 4.1: Aorta Double Oblique Plane

Successfully applying this algorithm will eliminate the need of verifying the segmentation result, discussed in Section 2.2.4.6.

4.4.2 Assurance Case

There is room for improvements on the arguments of the requirements of AortaGeomRecon. The scope of our project was to explore the breadth of the work needed for an AC. The next step would be to go into the depth where necessary. The correctness of the document is reviewed and approved by a domain expert, where there should be evidences that can support the argument. The evidence for E_GA.1 discussed in Figure 3.28 is missing because we did not have the time to complete our Verification and Validation Plan (VnV). The VnV should include the test report, an evidence for E_GA.1, that will indicate if AortaGeomRecon throws an exception when the inputs

do not meet one or more of the assumptions, with a message that clearly states the reason.

Bibliography

- [1] Introduction: Slicer tutorials#segmentation. <https://www.slicer.org/wiki/Documentation/Nightly/Training#Segmentation>.
- [2] Astah system safety. <https://astah.net/products/astah-system-safety/>, May 2023.
- [3] BEARE, R., LOWEKAMP, B., AND YANIV, Z. Image segmentation, registration and characterization in r with simpleitk. *Journal of Statistical Software* 86, 8 (2018), 1–35.
- [4] BLANCHETTE, S. *Assurance Cases for Design Analysis of Complex System of Systems Software*.
- [5] CORPORATE, A. Code review vs. code walkthrough vs. code inspection, Jan 2023.
- [6] EMINAGA, O., AND LIAO, J. C. Chapter 16 - prospect and adversity of artificial intelligence in urology. In *Artificial Intelligence in Medicine*, L. Xing, M. L. Giger, and J. K. Min, Eds. Academic Press, 2021, pp. 309–337.
- [7] GERIG, G. Snap tutorial and user's manual. <http://www.itksnap.org/docs/>

- [fullmanual.php](#). [Online; accessed 2023; Section 6. Automatic Segmentation using Region Competition Snakes].
- [8] GIBAUD, B. The dicom standard: A brief overview. In *Molecular Imaging: Computer Reconstruction and Practice* (Dordrecht, 2008), Y. Lemoigne and A. Caner, Eds., Springer Netherlands, pp. 229–238.
- [9] GITHUB. Github actions documentation. <https://docs.github.com/en/actions/using-workflows/about-workflows>, 2023.
- [10] HARRIS, C. R., MILLMAN, K. J., VAN DER WALT, S. J., GOMMERS, R., VIRTANEN, P., COURNAPEAU, D., WIESER, E., TAYLOR, J., BERG, S., SMITH, N. J., KERN, R., PICUS, M., HOYER, S., VAN KERKWIJK, M. H., BRETT, M., HALDANE, A., DEL RÍO, J. F., WIEBE, M., PETERSON, P., GÉRARD-MARCHANT, P., SHEPPARD, K., REDDY, T., WECKESSER, W., ABBASI, H., GOHLKE, C., AND OLIPHANT, T. E. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362.
- [11] KELLY, T., AND WEAVER, R. The goal structuring notation—a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases* (2004), vol. 6, Citeseer.
- [12] KIKINIS, R., PIEPER, S. D., AND VOSBURGH, K. G. *3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support*. Springer New York, New York, NY, 2014, pp. 277–289.
- [13] KLUYVER, T., RAGAN-KELLEY, B., PÉREZ, F., GRANGER, B., BUSSONNIER, M., FREDERIC, J., KELLEY, K., HAMRICK, J., GROUT, J., CORLAY, S.,

IVANOV, P., AVILA, D., ABDALLA, S., AND WILLING, C. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), F. Loizides and B. Schmidt, Eds., IOS Press, pp. 87 – 90.

- [14] KURUGOL, S., SAN JOSE ESTEPAR, R., ROSS, J., AND WASHKO, G. R. Aorta segmentation with a 3d level set approach and quantification of aortic calcifications in non-contrast chest ct. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (2012), pp. 2343–2346.
- [15] LIN, J. Design document. <https://joviel25.github.io/AortaGR-design-document/index.html>, 2023.
- [16] LOWEKAMP, B., CHEN, D., IBANEZ, L., AND BLEZEK, D. The design of simpleitk. *Frontiers in Neuroinformatics* 7 (2013).
- [17] MA, J., ZHANG, Y., GU, S., ZHU, C., GE, C., ZHANG, Y., AN, X., WANG, C., WANG, Q., LIU, X., CAO, S., ZHANG, Q., LIU, S., WANG, Y., LI, Y., HE, J., AND YANG, X. Abdomenct-1k: Is abdominal organ segmentation a solved problem? *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 10 (2022), 6695–6714.
- [18] MEDICAL PROFESSIONAL, C. C. Aorta: Anatomy and function, 2021.
- [19] NEJAD, M. S. A case study in assurance case development for scientific software. Master’s thesis, McMaster University, Hamilton, ON, Canada, July 2017. <http://hdl.handle.net/11375/23075>.

- [20] PARNAS, D., CLEMENT, P., AND WEISS, D. M. The modular structure of complex systems. In *International Conference on Software Engineering* (1984), pp. 408–419.
- [21] PROFESSIONAL, C. C. M. Ascending aorta: Anatomy and function, 2023.
- [22] PROFESSIONAL, C. C. M. What is the descending aorta?, 2023.
- [23] RAYBAUT, P. Spyder-documentation. Available online at: pythonhosted.org (2009).
- [24] RUEDEN, C. Level sets, 2021.
- [25] SMITH, S., NEJAD, M. S., AND WASSYNG, A. Building confidence in scientific computing software via assurance cases. *CoRR abs/1912.13308* (2019).
- [26] SMITH, W. S. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006* (Minneapolis / St. Paul, Minnesota, 2006), pp. 209–218.
- [27] VENTOLA, C. Mobile devices and apps for health care professionals: Uses and benefits. *P & T : a peer-reviewed journal for formulary management* 39 (05 2014), 356–64.
- [28] WEINSTOCK, C. Assurance cases and confidence. <https://insights.sei.cmu.edu/blog/assurance-cases-and-confidence/>, Aug 2013.
- [29] WIKIPEDIA. Lint (software). [https://en.wikipedia.org/wiki/Lint_\(software\)](https://en.wikipedia.org/wiki/Lint_(software)), 2023.

- [30] WIKIPEDIA. Sørensen–dice coefficient. https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient, Jul 2023.
- [31] YUSHKEVICH, P. A., PIVEN, J., CODY HAZLETT, H., GIMPEL SMITH, R., HO, S., GEE, J. C., AND GERIG, G. User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. *Neuroimage* 31, 3 (2006), 1116–1128.
- [32] ZOU, K. H., WARFIELD, S. K., BHARATHA, A., TEMPANY, C. M., KAUS, M. R., HAKER, S. J., WELLS, W. M., JOLESZ, F. A., AND KIKINIS, R. Statistical validation of image segmentation quality based on a spatial overlap index1: scientific reports. *Academic Radiology* 11, 2 (2004), 178–189.

Appendix A

Software Requirements

Specification for AortaGeomRecon

Software Requirements Specification for AortaGeomRecon

Jingyi Lin

September 15, 2023

Contents

1	Reference Material	iv
1.1	Table of Units	iv
1.2	Table of Symbols	iv
1.3	Abbreviations and Acronyms	v
2	Introduction	1
2.1	Purpose of Document	1
2.2	Scope of Requirements	1
2.3	Characteristics of Intended Reader	1
2.4	Organization of Document	1
3	General System Description	2
3.1	System Context	2
3.2	User Characteristics	3
3.3	System Constraints	3
4	Specific System Description	3
4.1	Problem Description	3
4.1.1	Organ Segmentation	3
4.1.2	Coordinate Systems	4
4.1.3	Physical System Description	6
4.1.4	Goal Statements	7
4.2	Solution Characteristics Specification	7
4.2.1	Assumptions	7
4.2.2	Theoretical Models	7
4.2.3	General Definitions	7
4.2.4	Data Definitions	7
4.2.5	Data Types	9
4.2.6	Instance Models	9
4.2.7	Input Data Constraints	11
4.2.8	Properties of a Correct Solution	11
5	Requirements	11
5.1	Functional Requirements	11
5.2	Nonfunctional Requirements	11
6	Likely Changes	12
7	Unlikely Changes	12
8	Traceability Matrices and Graphs	12

Revision History

Date	Version	Notes
2023-02-12	1.0	Notes
2023-03-01	1.01	Modified system context image, coordinate systems, and goal statements.
2023-04-29	1.02	Added requirements, instance models, data definitions
2023-06-05	1.03	Added Traceability Matrices
2023-06-18	1.04	Included the missing sections, modified the equations of DD, and IM. Added 4 new NFRs.
2023-09-15	1.05	Updated Traceability Matrix.

1 Reference Material

This section records information for easy reference.

1.1 Table of Units

Throughout this document SI (Système International d'Unités) is employed as the unit system. In addition to the basic units, several derived units are used as described below. For each unit, the symbol is given followed by a description of the unit and the SI name.

1.2 Table of Symbols

The table that follows summarizes the symbols used in this document along with their units. The choice of symbols was made to be consistent with existing documentation for 3D Slicer program. The symbols are listed in alphabetical order.

symbol	type	description
m	\mathbb{N}	The first dimension of the segmentation volume.
m_i	\mathbb{N}	The first dimension of V_{in} .
m_o	\mathbb{N}	The first dimension of V_{out} .
n	\mathbb{N}	The second dimension of the segmentation volume.
n_i	\mathbb{N}	The second dimension of V_{in} .
n_o	\mathbb{N}	The second dimension of V_{out} .
p	\mathbb{N}	The third dimension of the segmentation volume.
p_i	\mathbb{N}	The third dimension of V_{in} .
p_o	\mathbb{N}	The third dimension of V_{out} .
slice	$\mathbb{R}^{m \times n}$	A slice is a 2 dimensional image view from the superior to inferior direction.
v	\mathbb{R}	A voxel reports the intensity of a single point on a grey-scale three-dimensional images.
$HIGH$	\mathbb{N}	A high intensity values means 1 on a scale of 0 and 1, or 255 on a scale of 0 to 255.
LOW	\mathbb{N}	A low intensity values means 0 on a scale of 0 and 1, or 0 on a scale of 0 to 255.
Seed_a	\mathbb{N}^3	The initial ascending aorta centre coordinates.
Seed_d	\mathbb{N}^3	The initial descending aorta centre coordinates.
Start	\mathbb{N}^3	A coordinate indicates the indexes of a starting voxel.
V	$\mathbb{R}^{m \times n \times p}$	Volume formed by a sequence of slice

1.3 Abbreviations and Acronyms

symbol	description
A	Assumption
AortaGeomRecon	Aorta Geometry Reconstructor
DD	Data Definition
DICOM	Digital Imaging and Communications in Medicine
GD	General Definition
GS	Goal Statement
IM	Instance Model
LC	Likely Change
PS	Physical System Description
R	Requirement
SRS	Software Requirements Specification
T	Theoretical Model

2 Introduction

This document provides an overview of the Software Requirements Specification (SRS) for the AortaGeomRecon. AortaGeomRecon provides a semi-automatically aorta segmentation method, a highly customizable aorta segmentation module, and an interactive user interface to apply the segmentation workflow.

One of the existing methods involves the use ITK-Snap software and its segmentation module. First, the user needs to convert the DICOM data files (or any other file type) to VTK file. Then, the user can load the VTK file to ITK-Snap, and use its segmentation module to perform aorta segmentation. This segmentation method lets user initiate several voxels within the aorta volume and expand with a user's given size in each iteration. After the aorta volume has been filled by the "bubble", the user needs to cut the parts that are not within the aorta.

2.1 Purpose of Document

The main purpose of this document is to provide sufficient information to understand what AortaGeomRecon module does. The goals and theoretical models used in the AortaGeomRecon segmentation module implementation are provided, with an emphasis on explicitly identifying assumptions and unambiguous definitions.

2.2 Scope of Requirements

The scope of requirements only covers for the segmentation of the organ, more specifically the ascending aorta, the aortic curvature and the descending aorta. The requirements assume that the source of the data is accurate, and the user can manipulate (read, change dimensions) the data.

2.3 Characteristics of Intended Reader

The readers of the SRS should have taken the university level introduction to computational mathematic course, and be capable of understand the mathematical notation in the instance model section. The readers might have taken the university level introduction to software engineering course, have learned at least the waterfall software development model, and understands the purpose of the software specification requirement document, and other documents.

2.4 Organization of Document

The organization of this document follows the template for an SRS for scientific computing software proposed by [Koothoor \(2013\)](#) and [Smith and Lai \(2005\)](#). The presentation follows the standard pattern of presenting goals, theories, definitions and assumptions. The goal

statements are refined to the theoretical models, and theoretical models to the instance models. For readers that would like a more bottom-up approach, they can start reading the instance models in Section 4.2.6 and trace back to find any additional information they require.

3 General System Description

This section provides general information about the system. It identifies the interfaces between the system and its environment, describes the user characteristics and lists the system constraints.

3.1 System Context

Figure 1 shows the system context. A circle represents an external entity outside the software, the user in this case. A rectangle represents the software system itself. Arrows are used to show the data flow between the system and its environment.

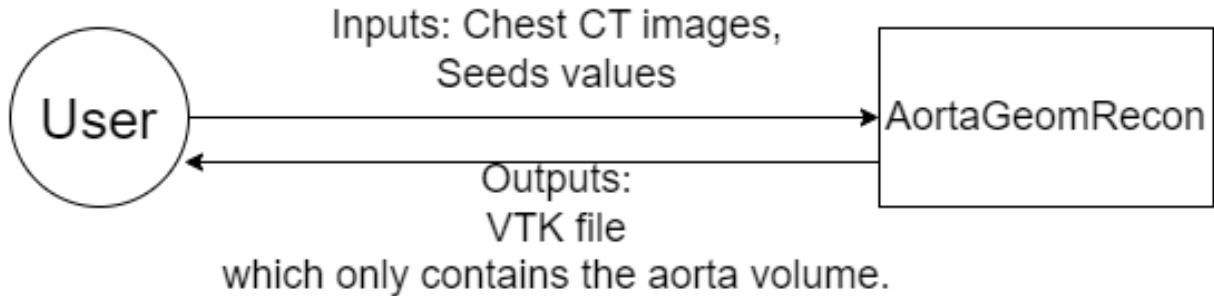


Figure 1: System Context

- User Responsibilities:
 - Provide the input data to the system
 - Ensure the input meets the necessary assumptions
 - Verify the result meets their requirements, otherwise repeat the process with a different seed values.
- AortaGeomRecon Responsibilities:
 - Provide DICOM data reader which can take a path to a folder containing DICOM files.

- Provide crop functionality to easily select a region of interest.
- Provide simple interactions to obtain and store the users' inputs. This includes a data probe to read voxel location which stored as a coordinate, and text inputs for real numbers.
- Provide visualization on the result data.

3.2 User Characteristics

The end user of AortaGeomRecon should have taken the university level anatomy introduction course, and be capable of finding the center of the descending aorta and the ascending aorta.

3.3 System Constraints

There are no system constraints of AortaGeomRecon.

4 Specific System Description

This section first presents the problem description, which gives a high-level view of the problem to be solved. This is followed by the solution characteristics specification, which presents the assumptions, theories, definitions and finally the instance models.

4.1 Problem Description

The main purpose of AortaGeomRecon is to semi-automatically segment a 3D aorta geometry from a chest CT scan.

4.1.1 Organ Segmentation

Organs are the body's recognizable structures (for example, the heart, lungs, liver, eyes, and stomach) that perform specific functions. Figure 2 below shows all the organs within a human body. The organ segmentation or the organ boundary segmentation is useful for orientation and identification of the regions of interests inside the organ during the diagnostic or treatment procedure. The aorta segmentation is important for aortic calcification quantification and to guide the segmentation of other central vessels. [Villa-Forte \(2022\)](#)

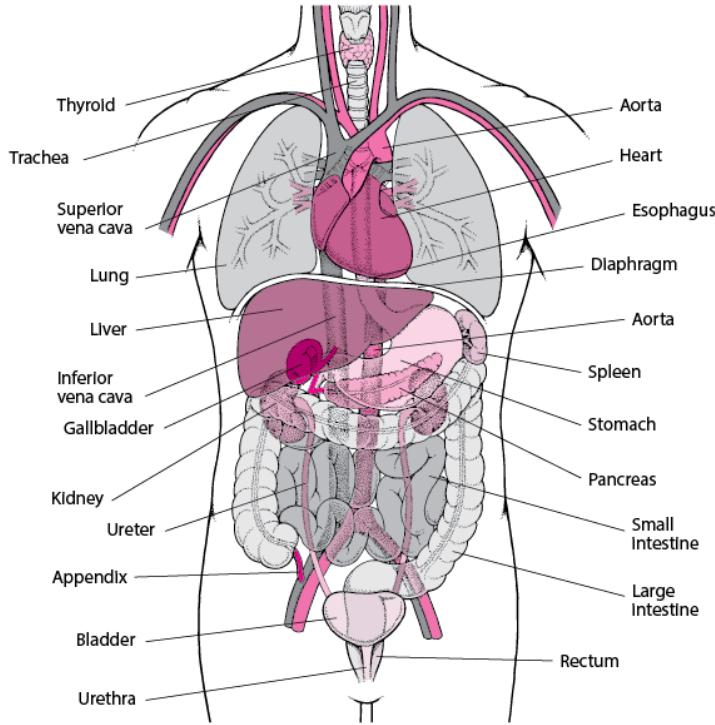


Figure 2: Human organs [Villa-Forte \(2022\)](#)

4.1.2 Coordinate Systems

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements. [Nejad \(2017\)](#)

While working with medical images, it is necessary to be familiar with the different coordinate systems of the medical literature and how data (voxels' orientation) is interpreted in different medical and nonmedical software. Each coordinate system uses one or more numbers (coordinates) to uniquely determine the position of a point (in the medical context, we refer to each point as a voxel). The purpose of this section is to introduce some coordinate systems related to the medical imaging. There are different coordinate systems to represent data. A knowledge of the following coordinate systems is needed to work with the medical images.

Cartesian Coordinate System A Cartesian coordinate system is a coordinate system that specifies each point uniquely in a 2D plane by a pair of numerical coordinates or in a 3D space by three numerical coordinates. We assume a right-hand Cartesian coordinate system throughout this document.

World Coordinate System World Coordinate System (WCS) is a Cartesian coordinate system that describes the physical coordinates associated with a model such as an MRI scanner or a patient. While each model has its own coordinate system, without a universal coordinate system such as WCS, they cannot interact with each other. For model interaction to be possible, their coordinate systems must be transformed into the WCS. Figure 3 shows the WCS corresponding space and axes.

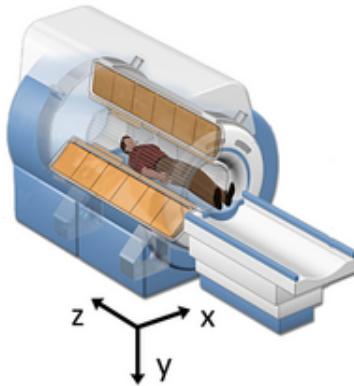


Figure 3: World Coordinate System Space and Axes [sli \(2014\)](#)

Anatomical Coordinate System Anatomical coordinate system, also known as patient coordinate system, is a right-handed 3D coordinate system that describes the standard anatomical position of a human using the following 3 orthogonal planes:

- Axial / Transverse plane: is a plane parallel to the ground that separates the body into head (superior) and tail (inferior) positions.
- Coronal / Frontal plane: is a plane perpendicular to the ground that divides the body into front (anterior) and back (posterior) positions.
- Sagittal / Median plane: is a plane that divides the body into right and left positions.

Figure 4 shows this coordinate system.

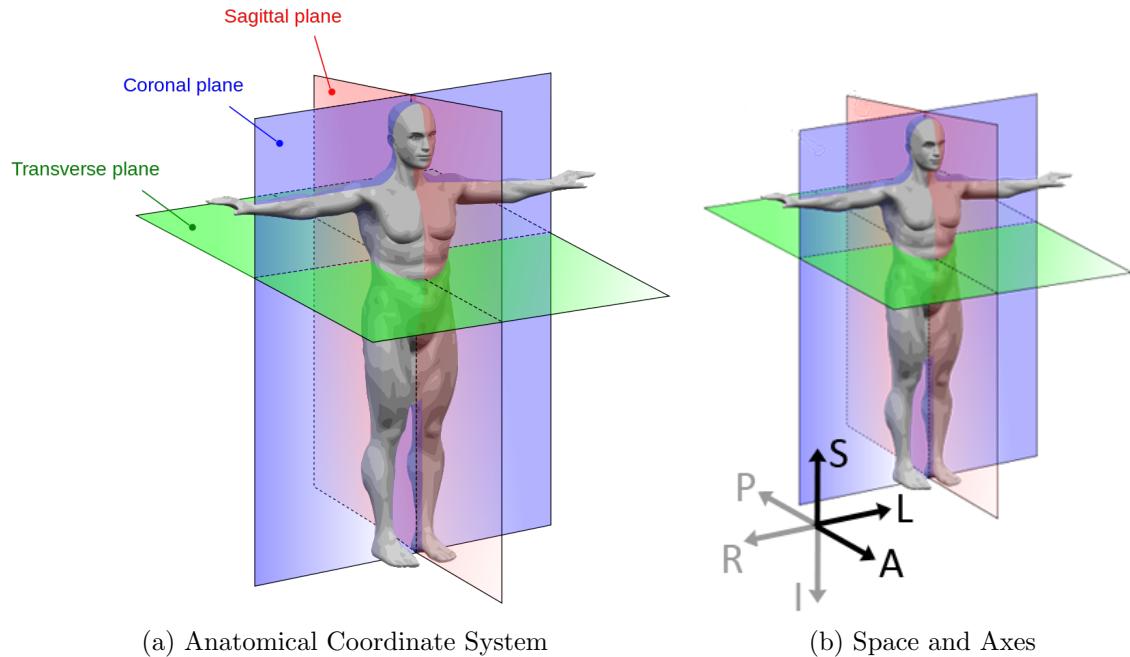


Figure 4: Anatomical Coordinate System Space and Axes [sli \(2014\)](#)

Medical applications follow an anatomical coordinate system to store voxels in sequences. Depending on how the data is stored, this coordinate system can be divided into different bases. The most common ones are:

- LPS Coordinate System:

The LPS coordinate system is used in DICOM images. In this system, voxels are ordered from left to right in a row, rows are ordered from posterior to anterior, and slices are stored from inferior to superior. LPS stands for Left-Posterior-Superior which indicates the directions that spatial axes are increasing.

- RAS Coordinate System:

The RAS coordinate system is the preferred basis for Neurological applications such as 3dfim+, and 3D Slicer. RAS stands for Right-Anterior-Superior is similar to LPS with the first two axes flipped.

Image Coordinate System To specify locations in an image we need to know to which coordinate system it is referenced. Different software may use different orders as their index convention.

Each of the coordinate systems mentioned above are used by different systems.

4.1.3 Physical System Description

We do not study the physical system for the images or how the data is actually generated.

4.1.4 Goal Statements

Given the DICOM image that includes patient's chest, the descending aorta center voxel coordinate, and the ascending aorta center voxel coordinate, the goal statements are:

GS1: Extract the three-dimensional segmentation of the aorta.

4.2 Solution Characteristics Specification

4.2.1 Assumptions

This section simplifies the original problem and helps in developing the theoretical model by filling in the missing information for the physical system. The numbers given in the square brackets refer to the theoretical model [T], general definition [GD], data definition [DD], instance model [IM], or likely change [LC], in which the respective assumption is used.

A1: The 3D image provided by the user must contain a visually distinguishable aorta volume [IM1].

A2: User should select a valid region of interest [IM2].

A3: User should input a singular volume (3 dimensional image) even if the data format supports the 4th dimension (time) [IM1].

4.2.2 Theoretical Models

There are no theoretical models used in this document.

4.2.3 General Definitions

There are no general definition used in this document.

4.2.4 Data Definitions

This section collects and defines all the data needed to build the instance models.

Number	DD1
Label	Voxel
Symbol	$v : \mathbb{R}$
SI Units	-
Equation	-
Description	A slice (DD2) consists of $n \times n$ voxels. A real number is assigned to each voxel to reports the intensity on a grey-scale image.
Sources	Nejad (2017)
Ref. By	DD2
Number	DD2
Label	Image/Slice
Symbol	$slice : \mathbb{R}^{m \times n}$
SI Units	-
Equation	-
Description	A visual representation that is using only two spatial dimensions with a sequence of arrays where a voxel (DD1) represents the color or intensity. Each move in the transverse plane (Figure 4) is considered as one slice
Sources	Nejad (2017)
Ref. By	DD3
Number	DD3
Label	Volume
Symbol	$V : \mathbb{R}^{m \times n \times p}$
SI Units	-
Equation	-
Description	A three-dimensional image is a sequence of some images/slices (DD2).
Sources	-
Ref. By	IM1

4.2.5 Data Types

There are no additional data types used in this document.

4.2.6 Instance Models

This section transforms the problem defined in Section 4.1 into one which is expressed in mathematical terms. It uses concrete symbols defined in Section 4.2.4 to replace the abstract symbols in the models. There are no theoretical models or general definitions used in this document.

The goals GS1 are solved by finding IM1 and perform IM2 on the aorta.

Number	IM1
Label	Region of interest
Inputs	$V_{\text{in}} : \mathbb{R}^{m_i \times n_i \times p_i}$, $Start : \mathbb{N}^3$, $m_o, n_o, p_o : \mathbb{N}$, with the following constraints: $\begin{aligned} 0 &\leq Start[0] < (m_i - 1) \\ 0 &\leq Start[1] < (n_i - 1) \\ 0 &\leq Start[2] < (p_i - 1) \\ 0 &< m_o \leq (m_i - Start[0]) \\ 0 &< n_o \leq (n_i - Start[1]) \\ 0 &< p_o \leq (p_i - Start[2]) \end{aligned}$
Output	$V_{\text{out}} : \mathbb{R}^{m_o \times n_o \times p_o}$ such that $\begin{aligned} \forall(i, j, k : \mathbb{N} \mid \\ i \in [Start[0]..Start[0] + m_o] \wedge \\ j \in [Start[1]..Start[1] + n_o] \wedge \\ k \in [Start[2]..Start[2] + p_o] : \\ V_{\text{out}}[i][j][k] = V_{\text{in}}[i][j][k]) \end{aligned}$
Description	The regions of interest is a subset (shaped like a box) of the 3D V_{out} . This subset contains the anatomical structure that the users wants to read, process or extract.
Sources	
Ref. By	IM2

Number	IM2
Label	Segmentation
Input	$V_{\text{in}} : \mathbb{R}^{m \times n \times p}$, $Seed_a : \mathbb{N}^3$, $Seed_d : \mathbb{N}^3$
Output	$V_{\text{out}} : \mathbb{R}^{m \times n \times p}$ such that $\forall (i, j, k : \mathbb{N} \mid i \in [0..m - 1] \wedge j \in [0..n - 1] \wedge k \in [0..p - 1] : (V_{\text{in}}[i, j, k] \in \text{structure} \implies V_{\text{out}}[i, j, k] = HIGH \mid V_{\text{in}}[i, j, k] \notin \text{structure} \implies V_{\text{out}}[i, j, k] = LOW))$ <p>The inputs $Seed_a$ and $Seed_d$ are used to determine whether a given element of V_{in} is in structure or not.</p>
Description	The process of extract an anatomical structure from the original 3D volume. The extracted anatomical structure is represented with high intensity pixel value. The rest of the image should have a lower intensity pixel value. The segmentation needs the region of interest from IM1 to process less noise data. A seed is what the algorithm needed as the inputs to perform segmentation, the type of seed is different among different algorithm. The seeds in this section are the centre coordinate of the descending aorta and the ascending aorta. The yellow dots shown in Figure 5 are the example of the seed.
Sources	
Ref. By	R3, LC1

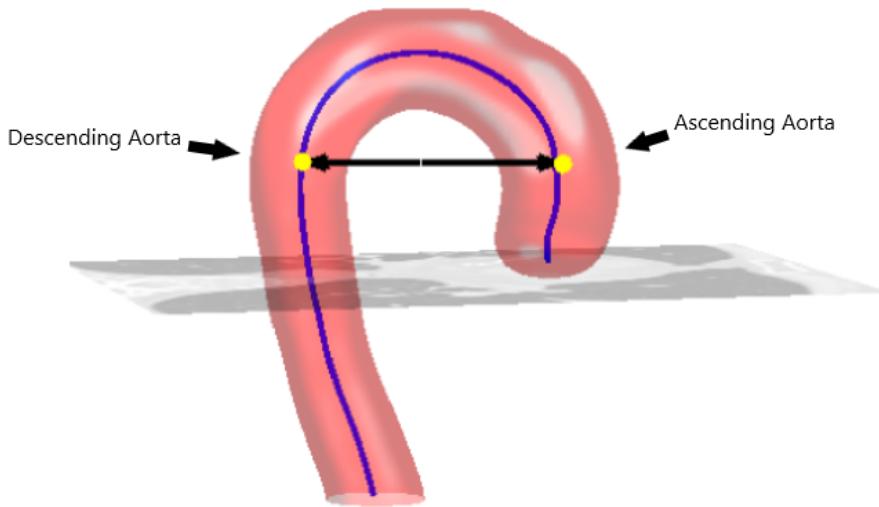


Figure 5: Aorta Seeds [Kurugol et al. \(2012\)](#)

4.2.7 Input Data Constraints

The only software constraint is the input volume data. It must be an acceptable file type to the system processing the data. For example, using ITK-Snap software to perform organ segmentation, the input data must be of VTK file.

4.2.8 Properties of a Correct Solution

A correct solution cannot be measured, but it can be confirmed by visually comparing the intersection of the extracted anatomical structure and the original volume.

5 Requirements

This section provides the functional requirements, the business tasks that the software is expected to complete, and the nonfunctional requirements, the qualities that the software is expected to exhibit.

5.1 Functional Requirements

R1: Input the following functions, data and parameters:

symbol	description
V	CT Scans volume (DD3)
$Seed_a$	The seed of ascending aorta centre coordinate (IM2)
$Seed_d$	The seed of descending aorta centre coordinate (IM2)

R2: Use the volume in R1 to create a second volume, the region of interest (IM1) that contains all voxels of the aorta.

R3: Perform segmentation (IM2) on the volume created in R2.

R4: Visualize a volume (DD3).

5.2 Nonfunctional Requirements

NFR1: **Usability** AortaGeomRecon allows a user that meets the user characteristics (Section 3.2) to import any DICOM files, input the required parameters, and begin the segmentation effortlessly. The number of steps it takes using AortaGeomRecon should be at least 30% less than the number of steps it takes by using ITK-Snap (bubble method mentioned in Section 2).

NFR2: **Safety** For a valid image, the AortaGeomRecon provides a correct solution, or no answer.

NFR3: **Learnability** The user interface and documentation should allow a user that meets the user characteristics (Section 3.2) to learn how to do an aorta segmentation in at least 30% of the time it takes to learn and use ITK-Snap (bubble method mentioned in Section 2).

NFR4: **Accuracy** For a given image the segmentation found by AortaGeomRecon should match that found by an expert using ITK-Snap. Whether two segmentations match is something that would be judged by a medical imaging expert.

NFR5: **Consistency** The coordinate system may be modified through the calculations, but any transformations will not alter the meaning of the data.

Other NFRs that might be discussed in the future include verifiability, and reusability.

6 Likely Changes

LC1: IM2 There are various segmentation algorithms, each has a different procedure and inputs.

7 Unlikely Changes

UC1: IM1 The method to retrieve a region of interest from a volume is fixed.

8 Traceability Matrices and Graphs

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Every time a component is changed, the items in the column of that component that are marked with an “X” may have to be modified as well. Table 2 shows the dependencies of theoretical models, general definitions, data definitions, and instance models with each other. Table 4 shows the dependencies of instance models, requirements, and data constraints on each other. Table 3 shows the dependencies of theoretical models, general definitions, data definitions, instance models, and likely changes on the assumptions.

The purpose of the traceability graphs is also to provide easy references on what has to be additionally modified if a certain component is changed. The arrows in the graphs represent dependencies. The component at the tail of an arrow is depended on by the component at

the head of that arrow. Therefore, if a component is changed, the components that it points to should also be changed.

	DD1	DD2	DD3	IM1	IM2
DD1	X				
DD2	X	X			
DD3		X	X		
IM1			X	X	
IM2				X	X

Table 2: Traceability Matrix Showing the Connections Between Items of Different Sections

	A1	A2	A3
DD1			
DD2			
DD3			X
IM1	X		X
IM2		X	X
LC1	X	X	X
UC1			X

Table 3: Traceability Matrix Showing the Connections Between Assumptions and Other Items

	DD ₁	DD ₂	DD ₃	IM ₁	IM ₂	R ₁	R ₂	R ₃	R ₄	NFR ₁	NFR ₂	NFR ₃	NFR ₄	NFR ₅
DD ₁	X													
DD ₂	X	X												
DD ₃		X	X											
IM ₁			X	X										
IM ₂				X	X									
R ₁			X		X	X								
R ₂			X	X			X							
R ₃			X		X			X						
R ₄			X						X	X	X			
NFR ₁		X	X	X	X	X	X	X	X				X	
NFR ₂			X	X	X	X	X				X			
NFR ₃				X	X	X	X	X	X			X		
NFR ₄	X	X	X	X								X		
NFR ₅	X	X	X	X	X	X	X	X	X				X	

Table 4: Traceability Matrix Showing the Connections Between Requirements and Instance Models

9 Reference

References

Coordinate systems, June 2014. URL https://www.slicer.org/wiki/Coordinate_systems.

Nirmitha Koothoor. A document drive approach to certifying scientific computing software. Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2013. URL <http://hdl.handle.net/11375/13266>.

Sila Kurugol, Raul San Jose Estepar, James Ross, and George R. Washko. Aorta segmentation with a 3d level set approach and quantification of aortic calcifications in non-contrast chest ct. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 2343–2346, 2012. doi: 10.1109/EMBC.2012.6346433.

Mojdeh Sayari Nejad. A case study in assurance case development for scientific software. Master's thesis, McMaster University, Hamilton, ON, Canada, July 2017. <http://hdl.handle.net/11375/23075>.

W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ågerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

Alexandra Villa-Forte. Tissues and organs. <https://www.merckmanuals.com/en-ca/home/fundamentals/the-human-body/tissues-and-organs>, 2022.

Appendix B

Module Guide for AortaGeomRecon

Module Guide for AortaGeomRecon

Jingyi Lin

July 5, 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
2023-07-05	1.2	Added Traceability matrix between modules and source code. Added User Hierarchy between modules.

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

1 Revision History	i
2 Reference Material	ii
2.1 Abbreviations and Acronyms	ii
3 Introduction	1
4 Anticipated and Unlikely Changes	2
4.1 Anticipated Changes	2
4.2 Unlikely Changes	2
5 Module Hierarchy	3
6 Connection Between Requirements and Design	3
7 Module Decomposition	3
7.1 Hardware Hiding Modules (M1)	4
7.2 Behaviour-Hiding Module	4
7.2.1 Input Format Module (M2)	4
7.2.2 Input Parameter Module (M3)	5
7.2.3 Control Module (M4)	5
7.2.4 Volume Visualization Module (M6)	5
7.2.5 Crop Module (M7)	6
7.2.6 Aorta Segmentation Module (M8)	6
7.2.7 Digital Enhancement Module (M11)	6
7.3 Software Decision Module	6
7.3.1 GUI Module (M5)	6
7.3.2 Image Processing Module (M9)	7
7.3.3 Multi-Dimensional Array Processing Module(M10)	7
8 Traceability Matrix	8
9 Use Hierarchy Between Modules	14
10 References	15

List of Tables

1 Module Hierarchy	4
2 Trace Between Requirements and Modules	8
3 Trace Between Anticipated Changes and Modules	8
4 Trace Between Modules and Code	13

List of Figures

1	Use hierarchy among modules	14
---	---------------------------------------	----

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

M11: Digital Enhancement 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. *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.

Level 1	Level 2	Level 3
Hardware-Hiding Module		
	Input Format Module	
	Input Parameter Module	
	Control Module	
Behaviour-Hiding Module	Volume Visualization Module	
	Crop Module	
	Aorta Segmentation Module	
	Digital Enhancement Module	
	GUI Module	
Software Decision Module	Image Processing Module	
	Multi-Dimensional Array Processing 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 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' attributes](#)

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](#)

7.2.7 Digital Enhancement Module (M11)

Secrets: The algorithm to enhance a digital image.

Services: Accepts a 2D image, output a 2D image with the same dimension as the input, improves visual quality of its greyscale image

Implemented By: AortaGeomRecon

Source: [AortaGeomReconDisplayModuleLogic's transform_image function](#)

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: Provide 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	M ₁ , M ₂ , M ₃ , M ₄
R2	M ₃ , M ₇
R3	M ₈ , M ₉ , M ₁₀
R4	M ₆
NFR1	M ₃ , M ₄ , M ₅
NFR2	M ₄ , M ₈ , M ₉ , M ₁₀
NFR3	M ₃ , M ₄ , M ₅ M ₆ , M ₇ , M ₈
NFR4	M ₇ , M ₈ , M ₉ , M ₁₀
NFR5	M ₃

Table 2: Trace Between Requirements and Modules

AC	Modules
AC ₁	M ₁
AC ₂	M ₂
AC ₃	M ₈
AC ₄	M ₃
AC ₅	M ₅
AC ₆	M ₇
AC ₇	M ₆
AC ₈	M ₄
AC ₉	M ₉ , M ₁₀

Table 3: Trace Between Anticipated Changes and Modules

Modules	Code
M1	-
M2 Input Format Module	<pre> # AortaGeomReconDisplayModule.py # https://github.com/smiths/aorta/blob/main/src/SlicerExtension/ # AortaGeometryReconstructor/AortaGeomReconDisplayModule/ # AortaGeomReconDisplayModule.py#L543-L554 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") # https://github.com/smiths/aorta/blob/main/src/SlicerExtension/ # AortaGeometryReconstructor/AortaGeomReconDisplayModule/ # AortaGeomReconDisplayModule.py#L824-L855 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 </pre> <hr/>

M3 Input Parameter Module

```
# AortaSegmenter.py
# https://github.com/smiths/aorta/blob/main/src/SlicerExtension/
# AortaGeometryReconstructor/AortaGeomReconDisplayModule/
# AortaGeomReconDisplayModuleLib/AortaSegmenter.py#L53-L82
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

```
# https://github.com/smiths/aorta/blob/main/src/SlicerExtension/
# AortaGeometryReconstructor/AortaGeomReconDisplayModule/
# AortaGeomReconDisplayModule.py#L537-L583
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
# https://github.com/smiths/aorta/blob/main/src/SlicerExtension/
#   AortaGeometryReconstructor/AortaGeomReconDisplayModule/
#     AortaGeomReconDisplayModule.py#L139-L245
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.ascaAortaSeed.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
[3D Slicer's Volume Rendering Source Code](#)

M7 Crop Volume Module 3D Slicer's Crop Volume Module
[3D Slicer's Crop Volume Module Source Code](#)

M8 Aorta Segmentation Module AortaSegmenter class

M9 Image Processing Module SimpleITK

M10 Multi-Dimensional Array Processing Module NumPy

M11 Digital Enhancement Module

```
# AortaGeomReconDisplayModule.py
# https://github.com/smiths/aorta/blob/main/src/SlicerExtension/
#   AortaGeometryReconstructor/AortaGeomReconDisplayModule/
#   AortaGeomReconDisplayModule.py#L739-L769
def transform_image(self, cropped_volume):
    """
    Histogram Equalization for Digital Image Enhancement.
    https://levelup.gitconnected.com/introduction-to-histogram-
        equalization-for-digital-image-enhancement-420696db9e43
    """
    cropped_image = sitkUtils.PullVolumeFromSlicer(cropped_volume)
    img_array = sitk.GetArrayFromImage(
        (sitk.Cast(sitk.RescaleIntensity(cropped_image), sitk.
        sitkUInt8)))
    histogram_array = np.bincount(img_array.flatten(), minlength=256)
    num_pixels = np.sum(histogram_array)
    histogram_array = histogram_array/num_pixels
    chistogram_array = np.cumsum(histogram_array)
    transform_map = np.floor(255 * chistogram_array).astype(np.uint8)
    img_list = list(img_array.flatten())
    eq_img_list = [transform_map[p] for p in img_list]
    eq_img_array = np.reshape(np.asarray(eq_img_list), img_array.
        shape)
    eq_img = sitk.GetImageFromArray(eq_img_array)
    eq_img.CopyInformation(cropped_image)
    median = sitk.MedianImageFilter()
    median_img = sitk.Cast(median.Execute(eq_img), sitk.sitkUInt8)
    self._cropped_image = median_img
```

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.

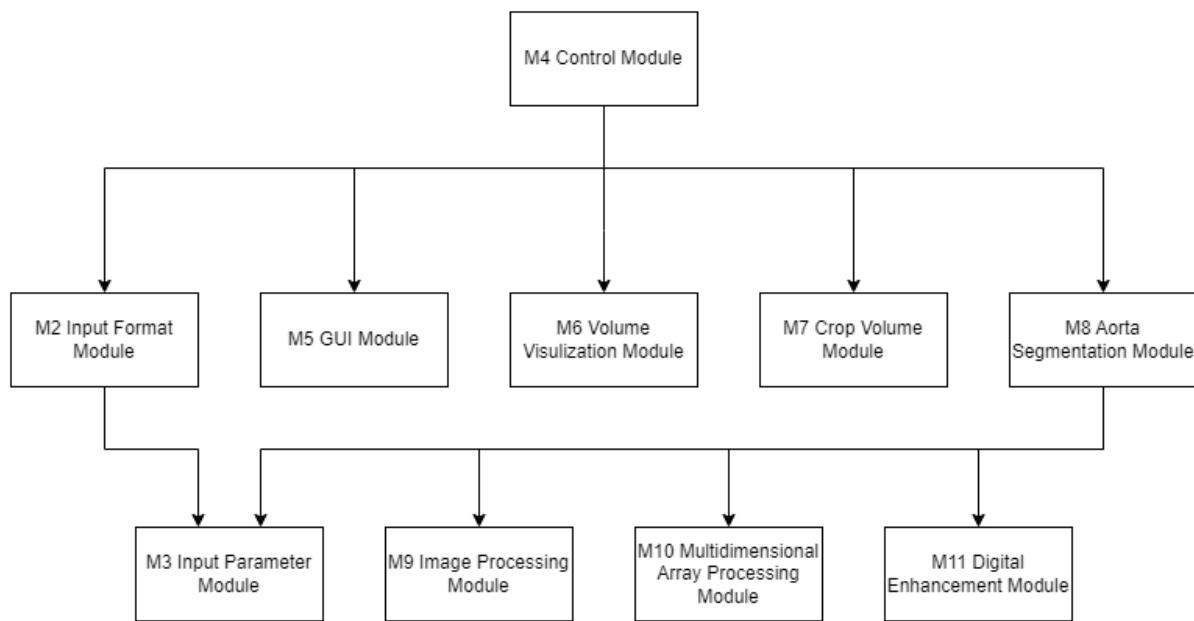


Figure 1: Use hierarchy among modules

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.