Chapter 3

Assurance Cases and Selected Evidence for AortaGeomRecon

In this chapter, I discuss the scope of our work, which is building the evidence to support each claim of our AC for AGR. The top level claims of the AC developed in previous work [18] is correct and complete, thus I have a list of evidence that can support the arguments and the top level claims. Our work focus on providing the correct evidence for the assurance case, and the following material is presented: the Software Requirements Specification of AGR, the Design Document, the Module Guide, the Test Plan, the Algorithm Review, the User Manual, the User Instruction Video, and a Warning Message implemented in AGR.

3.1 Assurance Case Development

Assurance Case is build with claims, subclaims, contexts and the evidence. The parts of the AC add up to an argument for why the top level claim is true. By using Astah

System Safety software to present the Goal Structuring Notation (GSN) arguments [2][10], I want to show that our software delivers correct outputs when used for its intended use/purpose in its intended environment, and within its assumed operating assumptions. The Figure 3.1 shows the top level of the assurance cases. With the goal of arguing that the software delivers correct outputs, I decompose the goal into 4 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 of getting a 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, and the design and the implementation will be based on the requirement document.

The Figure 3.2 demonstrates the claims on the goal of correct requirements of AGR. On the left branch, GR_3C implies the goal of a 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 node. The S_Correctness and S_Completeness.4 node require a domain expert to review the quality of the

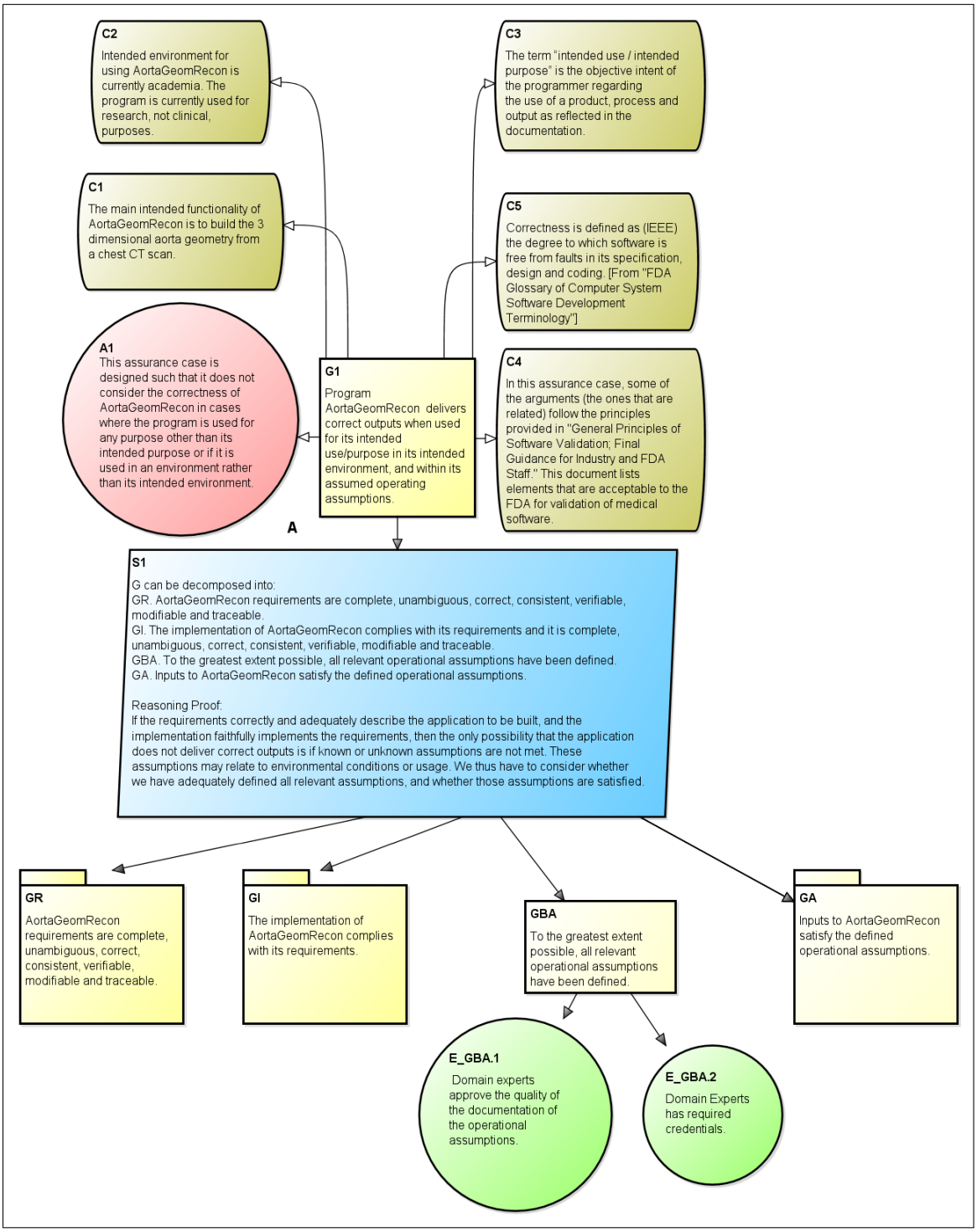


Figure 3.1: AGR Assurance Cases Top Level

document to ensure the documentation ~~met~~ the goal of the claims. The other characteristics of the documentation are supported with evidence as shown in the other branches in GR.

As ~~explains~~ in the assurance case GR, one of most important ~~statement~~ ~~of~~ SRS ~~having these characteristics is using~~ a standard template. ~~This document is attached as Appendix A.~~ Using a standard template ~~means that~~ all necessary requirements have been defined, ~~and it~~ allows a domain expert who has used this template to verify the quality of the document. ~~I~~ used a template tailored for research software [19], which is the standard template in the evidence E_Completeness.1, and all other evidences where a template is ~~included.~~

3.2.1 The Chapters of SRS

The chapters in the SRS and some of the most important sections in the chapter 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 SRS document. These tables ensure the consistency and the unambiguous characteristics of the document, ~~as the~~ evidences E_Consistency.1 and E_Unambiguous.1 shown in ~~the~~ Figure 3.2. They are located at the very beginning of the document, so the reader can first look at these tables before reading the entire document.

- Introduction

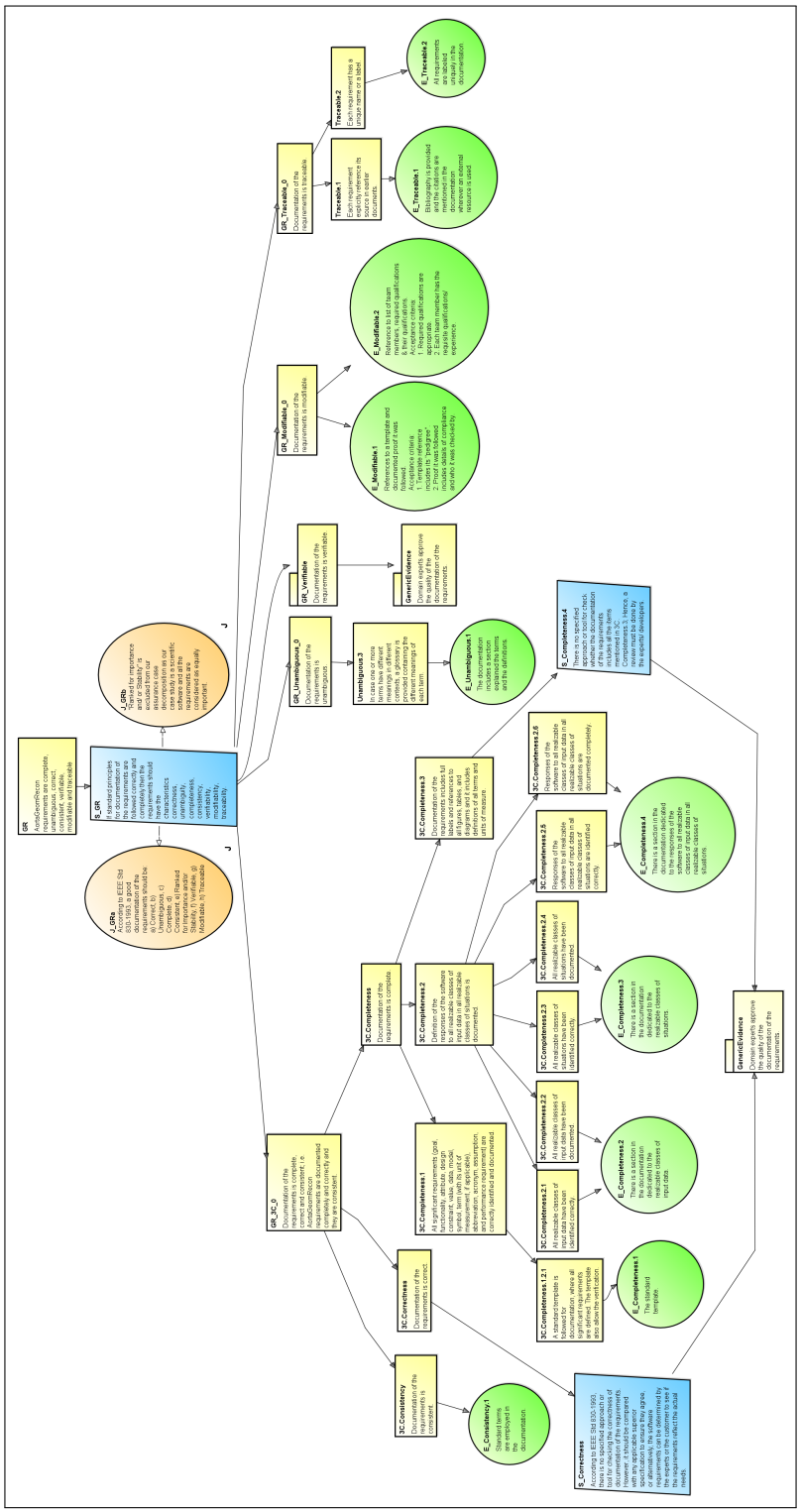


Figure 3.2: AGR Assurance Cases GR

In the introduction section, I introduced the problems and the scope of the document. These are subsections for explaining the purpose of document, abstracting the scope of requirements and defining the characteristics of the intended reader. This is provided for the reader to eliminates unambiguous in reading the document.

- General System Description

  The general system description includes a system context diagram which explains the relationship between the users, the inputs given by the user and the outputs of the AGR program. User responsibility and AGR responsibility are defined.

- Specific System Description

  In this section, I present more details 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 problem, and Goal Statements. The goal for AGR is to extract the three-dimensional segmentation of the aorta.

  In the next subsection, Solution Characteristics Specification, I started with the assumptions, as shown in Figure 3.3, to clearly define the scope of the requirement document, as shown in Figure 3.3. In the subsection Data Definitions, I defined Voxel, Image/Slice, and Volume with mathematical notation so that the developer can easily interpret, as shown in Figure 3.20. Next, in the subsection Instance Model, I showed the mathematical meaning of Region of Interest, in Figure 3.5, and Segmentation in Figure 3.6, which are the two essential models

that the developer must know ~~to develop the~~ solution. The Data Definitions and Instance Model sections ~~are the~~ evidences 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 responss of the software to all realizable classes of input data in all realizable classes of situation. The Data Definition section defines 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 distingushable 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 3.3: 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 3.4: 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: $$0 \leq Start[0] < (m_i - 1)$$ $$0 \leq Start[1] < (n_i - 1)$$ $$0 \leq Start[2] < (p_i - 1)$$ $$0 < m_o \leq (m_i - Start[0])$$ $$0 < n_o \leq (n_i - Start[1])$$ $$0 < p_o \leq (p_i - Start[2])$$
Output	$V_{\text{out}} : \mathbb{R}^{m_o \times n_o \times p_o}$ such that $$\forall(i, j, k : \mathbb{N} \mid$$ $$i \in [Start[0]..Start[0] + m_o] \wedge$$ $$j \in [Start[1]..Start[1] + n_o] \wedge$$ $$k \in [Start[2]..Start[2] + p_o] :$$ $$V_{\text{out}}[i][j][k] = V_{\text{in}}[i][j][k])$$
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 3.5: AGR SRS Instance Model Region Of Interest

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 $\forall(i, j, k : \mathbb{N} \mid$ $i \in [0..m-1] \wedge\ j \in [0..n-1] \wedge\ k \in [0..p-1] :$ $(V_{\text{in}}[i, j, k] \in \text{structure} \implies V_{\text{out}}[i, j, k] = HIGH \mid$ $V_{\text{in}}[i, j, k] \notin \text{structure} \implies V_{\text{out}}[i, j, k] = LOW))$ The inputs $Seed\_a$ and $Seed\_d$ are used to determine whether a given element of V_{in} is in structure or not.
Description	The process of extract an anotomical structure from the original 3D volume. The extracted anotomical 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

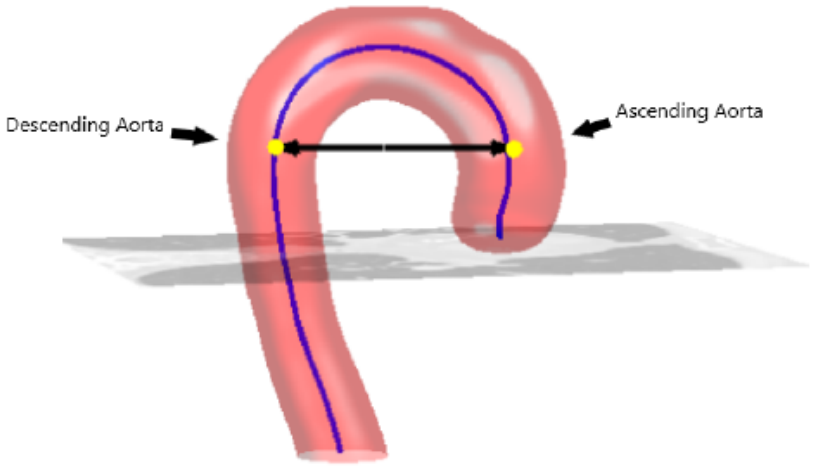


Figure 5: Aorta Seeds Kurugol et al. (2012)

Figure 3.6: AGR SRS Instance Model Region Of Interest

- Requirements

  With all the information in the document, I can now present the Functional Requirements and the Non-Functional Requirements for the program AGR. The Functional Requirements are defined by using the terms I presented in Data Definitions, Instance Model, and based on the other Functional Requirements, as shown in Figure 3.7. The Non-Functional Requirements usually have a measurement such as execution time, the effort of manual works. The Figure 3.8 demonstrates the Non-Functional Requirements: Usability, Safety, Learnability, Accurarcy and Consistency. The Functional Requirements and Non-Functional Requirements are labeled as the evidence E_Traceable.2 stated

5.1 Functional Requirements

R1: Input the following functions, data and parameters:

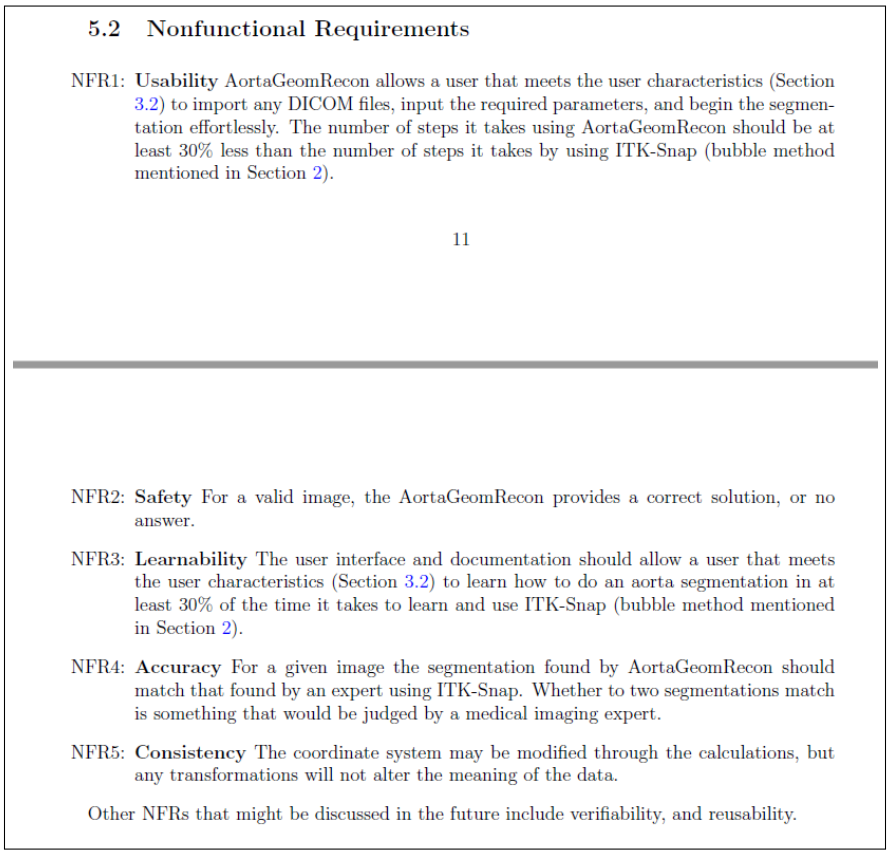
symbol	description
V	CT Scans volume (DD3)
$Seed\_a$	The seed of ascending aorta centre coordinate (IM2)
$Seed\_d$	The seed of descending aorta centre coordinate (IM2)

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 3.7: 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 to 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 3.8: AGR Non- Functional Requirements

- Likely Changes and Unlikely Changes

  This section discussed the likely changes that the developer might expect a change in the future works, and the unlikely changes that is are not going to change for a justified reason. 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 is the method of retrieving a region of interest, Most methods take a starting point and sizes in different dimensions to get the region of interest.

- Traceability Matrix and Graphs

  The traceability matrices ~~are~~ to provide easy references on what has to be ~~additionally~~ modified if a certain component is changed. ~~Below shows the traceability matrices of different sections.~~ ~~The~~ Figure 3.9 is the traceability matrix of the Data Definitions and Instance Models. ~~The~~ Figure 3.10 shows the relationship between the requirements and other sections, and ~~The~~ 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 3.9: AGR Traceability Matrix between Data Definitions and Instance Model

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

Table 3: Traceability Matrix Showing the Connections Between Requirements and Instance Models

Figure 3.10: AGR Traceability Matrix Between Requirements and Other sections

	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

- Bibliography A Bibliography is provided at the end of SRS documentation, where it points to the template [19] that I use to write this documentation, and a list of citations whenever an external resource is used. This is related to the evidences 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 to a template and documented proof it was followed.

3.2.2 Documentation Review

Documentation Review is necessary to ensure the documentation's correctness and completeness. When there is an update in the documentation, I used GitHub Issues and post a documentation review request, as shown in Figure 3.12, to Dr. Spencer Smith, who has the requisite qualifications/experience to review the completeness and the correctness of the documentation. Additionally, the goal of the documentation being verifiable is also reviewed by Dr. Spencer Smith, and he approves the qualify of the documentation with the characteristic.

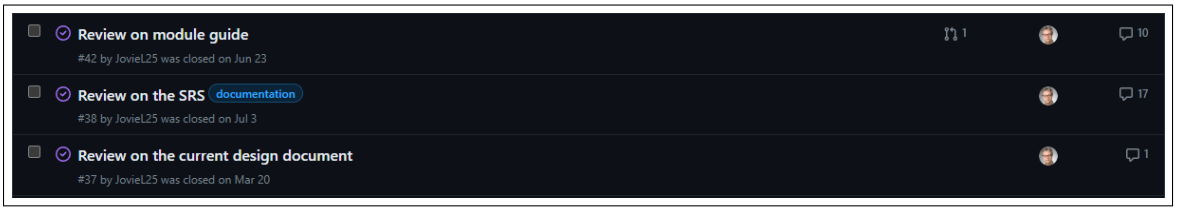


Figure 3.12: GitHub Repo Documentation Review Requests

3.3 Assurance Case for the Implementation

The goal of implementation is that it fully complies with the SRS. As Figure 3.13 shows, I argue this in two ways:

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

~~In~~ In this section, ~~I will~~ I focus on matching the developed artifacts to the evidence shown in Figure 3.13. ~~In~~ In the first sub-section ~~I will~~ discuss the test plan of the AGR, particularly ~~on how I build~~ the continuous integration test infrastructure, test cases and ~~provide a~~ the test procedure ~~to test~~ for testing all requirements of the software. Next, ~~I will~~ this section show ~~my~~ the design documents, including the Module Guide, ~~to demonstrate~~ which documents the system architecture, and a design document for a detailed design explanation. Finally, ~~I will talk~~ we present ~~about~~ the Code Walkthrough and the Algorithm Review, which helped us to eliminates bugs, errors, and increase our confidence in the implementation's completeness, and correctness.

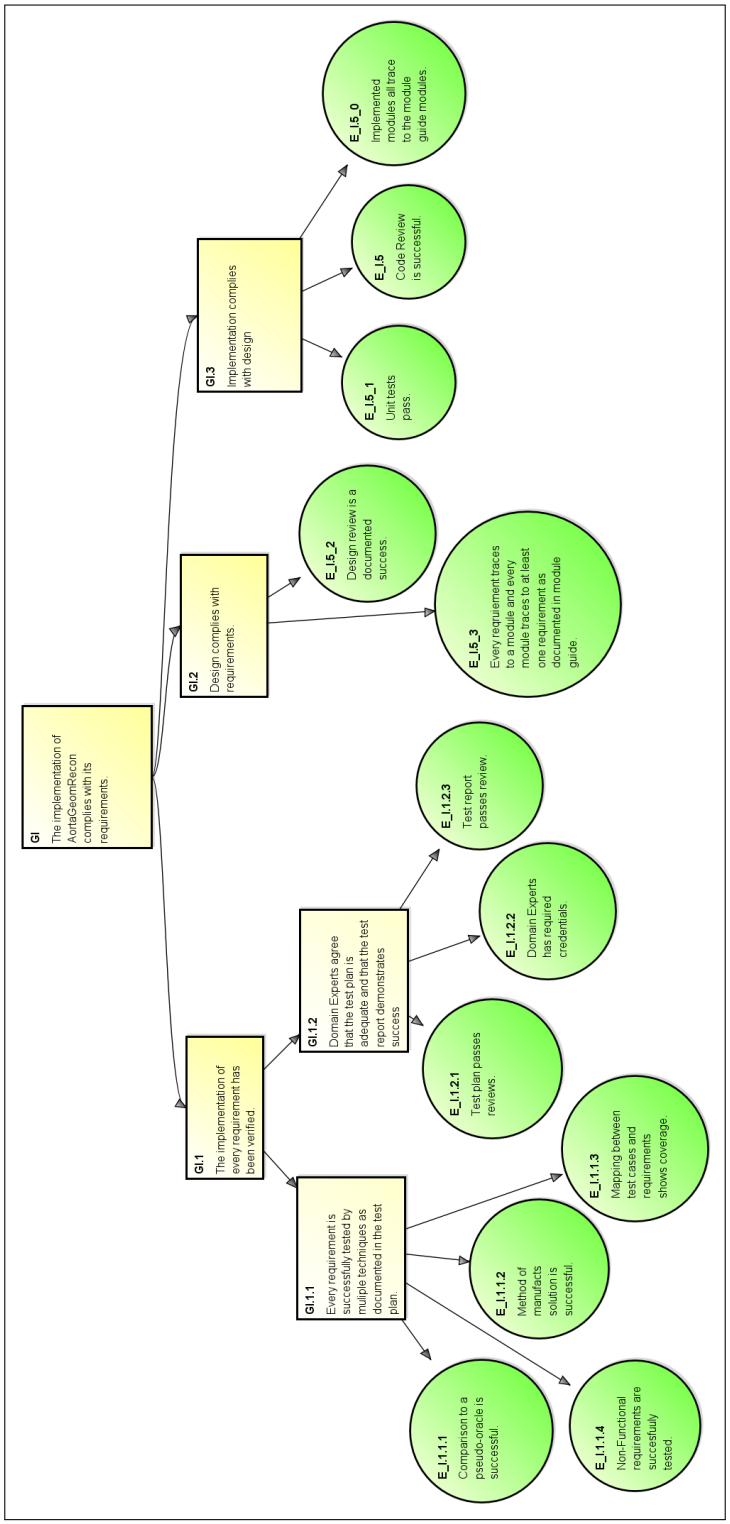


Figure 3.13: AGR Assurance Cases For Implementation

3.3.1 Test Plan

GI.1 states the goal of the implementaion of every requirement has been verified, so I need a test plan that is approved by a Domain Expert who has required credentials, and the tests cases covers all of the Functional Requirement and Non-Functional Requirements. Unlike the other algorithm that can easily be tested with a ground truth test case, our ground truth case is build by using another more accurate method such as ITK-Snap's bubble method, then manually crossing the unwanted pixels.

In this section, I will discuss on how I build a "Ground Truth" data with a verified version of the algorithm, then compare it with the result generated from the new updated version of the algorithm using Dice Similarity. Next, I will show how I use GitHub Actions workflow as the continuous integration infrastcture perform static code analysis and continuous integration tests. Then, I provide a test procedure to cover the Functional Requirements and Non-Functional Requirements, as the evidences under GI.1 state in Figure 3.13. Finally, I discuss on the test plan approval and the test report to includes the artifacts related to the evidences under GI.1.2.

3.3.1.1 Build "Ground Truth" data

Since learning on how to build a true "Ground Truth" test case by using ITK-Snap is out of the scope of our project, I continuously build the "Ground Truth" test cases with a previously verified version of the algorithm and a set of tuned hyperparameters. The only method to know which test case would be better is to have a manual review on the generated test case by a domain expert. The process of building the test case data is described as follows:

1. Generate test case data with the previous satisfied 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. If there is a strong difference in the DSC value, use visualization tool such as 3D Slicer to see the actual difference, and decide which test case to keep for the future.

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 [24]. With a small DSC value, 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

This leads to our Continuous Integration infrastructure, implemented with GitHub Actions workflow. A workflow is a configurable and automated process that will run one or more jobs on the desired system. GitHub Actions workflow used a YAML file to define the events and the commands to be executed on the temporary system, which has the build of the repository [8].

I have set up two automated process which 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 bugs that need

extra fixes. A "pull" event happens when a feature branch is going to merge 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 and until there are no errors, a pull-request cannot be merged with the main branch.

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. I used Python Flake8 as our linter to find bugs and errors, and ensures that program's readability by striking the source code with Google's published Python Style Guide. [22]

The second automated process is our continuous integration tests. By setting up Git Large File System (LFS) and upload to pre-build ground truth test data in the repository, I can now pass the cropped volume as the input data, the same aorta seeds and the hyperparameters that I have used to generate the ground truth test data to the algorithm and verify the results. By calculating the DSC value of both images, I can now set a limit such that the update to the algorithm is passing or failing our test if the DSC value is within the limit. This indicates that our evidence E_I.5_1 is accomplished.

3.3.1.3 Test Procedure

In this section, I will introduce our test procedure for Functional Requirements and Non-Functional Requirements.

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 test procedure for testing the Functional Requirements is described as follow:

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 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.

If step 1-4 did not proceed correctly, there is an error with 3D Slicer, which is out of the scope of ~~the~~ 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 is only concerns with the hyperparameters and the algorithm's implementation, and step 9 is affected by 3D Slicer and the result generated from step 8. This test procedure has all Functional Requirements match to a step, where R1 matches to step 2, R2

matches to step 3, R3 matches to step 8, and R4 matches to step 9. Thus, we have our evidence E_1.1.3.

The Non-Functional Requirements is difficult to test, because each user with different experience could result in different learning time, and total time it takes from scratch to generating a segmentation result. The test procedure used in Functional Requirement test can be used for Non-Functional Requirements test. Both NFR1 Usability and NFR3 Learability requires a measurement in time comparing to another software. NFR2 Safety, NFR4 Accuracy, and NFR5 Consitency requires a verification in the segmentation result. NFR4 Accuracy also requires a segmentation result generated by ITK-Snap. By performing all of the above action, I have all Non-Functional Requirements tested, which is the evidence 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 the Figure 3.14.

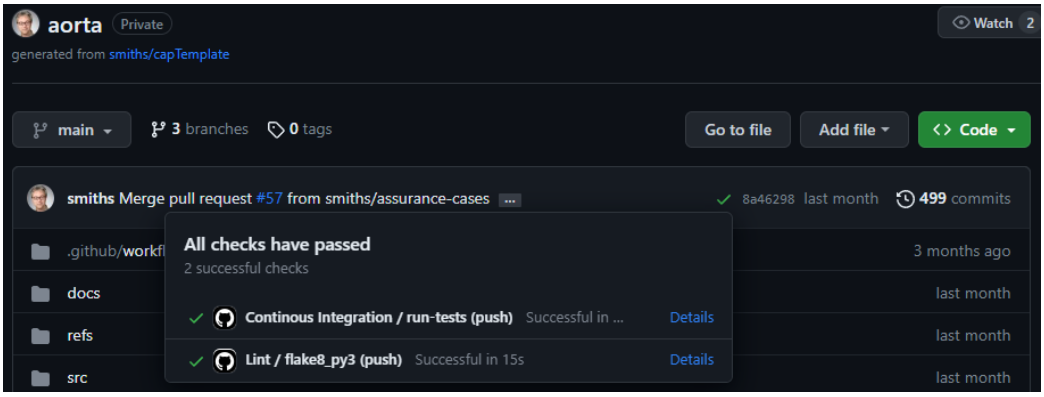


Figure 3.14: AGR Test Report

The GitHub will send an email to all the contributors when some checks were not successful, thus I are awared of these errors.

3.4 The Design Documentation of the AortaGeomRecon

In this section, we will discuss the design documents of the AortaGeomRecon. There are multiple aspects of the design document to discuss:

1. System architecture, or high levels design.
2. Detailed design explaning 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 binary in HTML and is hosted on a web server.

3.4.1 Module Guide

An important document to show that the design is complete, correct, and consistent design is Module Guide (MG), which is attached as Appendix B. As explained previously, MG demonstrates the system architecture, or the high level design of the AortaGeomRecon. The designer typically keeps these anticipated changes isolated to a single module so if the change happens, only one module is impacted. The anticipated changes are listed in the Figure 3.15 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 3.15: AGR Anticipated Changes

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. The module hierarchy and a part of the module decomposition is demonstrated in Figure 3.16 and Figure 3.17. 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 3.16: 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' atributes

7.2.3 Control Module (M4)

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

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

Implemented By: AortaGeomRecon

Source: AortaGeomReconDisplayModuleWidget module

7.2.4 Volume Visualization Module (M6)

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

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

Implemented By: 3D Slicer

Figure 3.17: AGR Module Decomposition Example

Now that I have listed the anticipated changes and the modules, I use traceability matrices to show the relationships between the modules and the anticipated changes, and between the modules and the requirements, as shown in Figure 3.18. This indicates that the design is fully complying with the requirements, as I stated in the evidence E_I.5_3, under the GI.2 in the Figure 3.13.

matrix shows all req. are mapped to module, which is part of the evidence for

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

Table 2: Trace Between Requirements and Modules

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

Table 3: Trace Between Anticipated Changes and Modules

Figure 3.18: AGR Modules Traceability Matrices

On top of relating the modules to the requirements, I am relating the actual source code to the modules, which is a strong evidence of our implementation complies with the requirements, as shown in Figure 3.19. Module 11, Digital Enhancement Module, is an example of Module mapping to a piece of the source code. In the comments on top, I added the source file where this piece of the source code is located, as well as the exact places with GitHub and the exact lines highlighed. This table demonstrates that the implemented modules all traces to the module guide modules, as stated in the 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	

```
# 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

Figure 3.19: AGR Part Of The Traceability Matrix On Modules And Code

3.4.2 Detailed Design Document

The purpose of the Design Document [14] is to explain in details how the algorithm works, and why it worked. Similar to what section 2.2.3 wrote, the design document explains in plain text the workflow of the algorithm. The design document is a piece of evidences that demonstrate unambiguity. Moreover, this document can let the domain expert to do a design review without reading the source codes directly, which helps building the evidence E_I.5_2.

To show and automate the detailed design, I used Sphinx, a Python Documentation Generator that can build module's documentation from the comments in the source code. Moreover, using reStructuredText to write the Algorithm Overview, I can build HTML code which can be published on a web server, as shown in Figure 3.20, which shows the index page of the website.

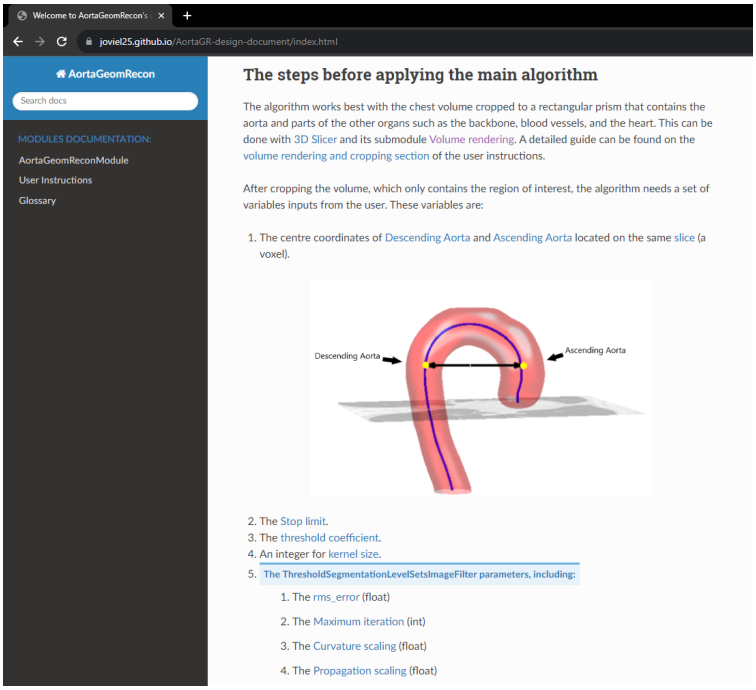


Figure 3.20: AGR Design Document Website

Another important section in detailed design document is the Glossary. It has a rich vocabulary explanation, images, and links to the outside source to let the reader understands as much as possible, as demonstrated in Figure 3.21.

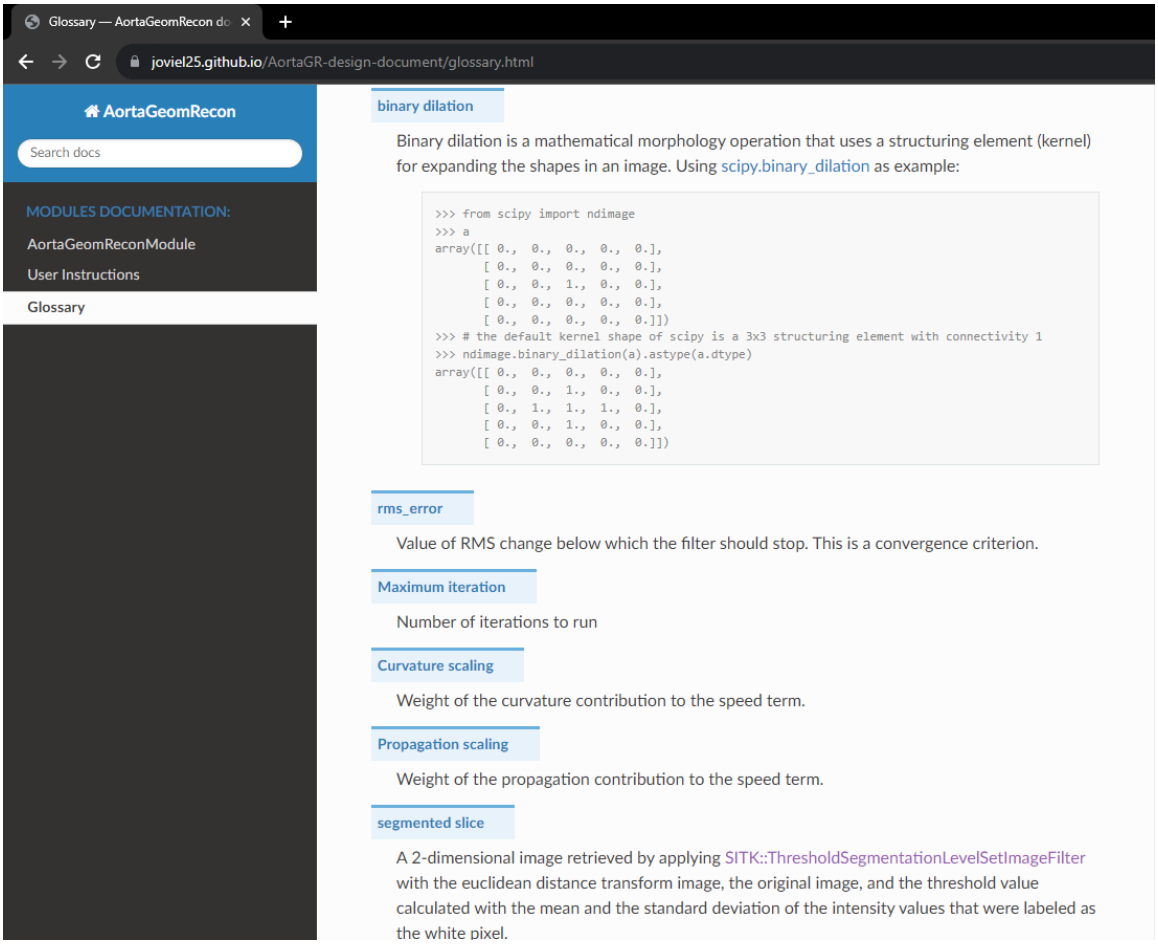


Figure 3.21: AGR Design Document Glossary

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, I will discuss the Code Review with Kailin Chu, and the key takeaways from this meeting. Then, I will discuss the Algorithm Review with DrDean Inglis, which the meeting has reinforced our confidence in the design. Finally, I 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 and started working the semi-automadial aorta segmentation algorithm as a summer researcher. The meeting happened on Thursday, April 20, 2023, and the duration is about an hour. Along with Dr. Smith Spencer, I was aiming to increase our confidence in the code via a Code Walkthrough. This code walkthrough did not increase our confidence in the software and it became a Code Review, because the code was developed by Kailin from two years ago, so some details and design decisions were missing, and some variables were decided by trial and error. Despite that the code walkthrough

has turned into an algorithm review, and it did not achieve what I wanted in the first place, this meeting was still very helpful. Originally, the old method to generate a label image that we have discussed in the section 2.2.4.1 is partialy random when the algorithm is segmenting on the superior direction. This happened when from axial view going toward the head direction, we are observing that ascending aorta and the descending aorta is going to merge into one piece. However, the segmentation on the aortic arch requires the algorithm to generate a label image to cover part of the descending aorta. This coverage is sometime missing when executing on certain test case. Knowing that this part of the algorithm was partially based on trial and error, I was confident on improving the algorithm by 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 when the aortic arch region is reached. Thus, it generates a better label image, and it will generate a more accurate segmentation result.

3.4.3.2 Algorithm Review with Dr. Dean Inglis

The algorithm review was conducted with Dr. Dean Inglis, an experienced professor, Medical Image Analyst, and Software Developer. The meeting happened on May 17, 2023, and the duration is about one and half hour. I presented our segmentation algorithm to him and requested validation of our approach, or suggestions for a potentially superior algorithm. Dr. Dean Inglis provided his insights on the algorithm, which I 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 ?? were very common in image analysis. The level sets segmentation is also a 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 allows the user to interactively browse the variables and the objects in debugging mode [17].

Figure 3.22: Spyder Variable Explorer [17]

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.

3.5 Assurance Case for Operational Assumptions

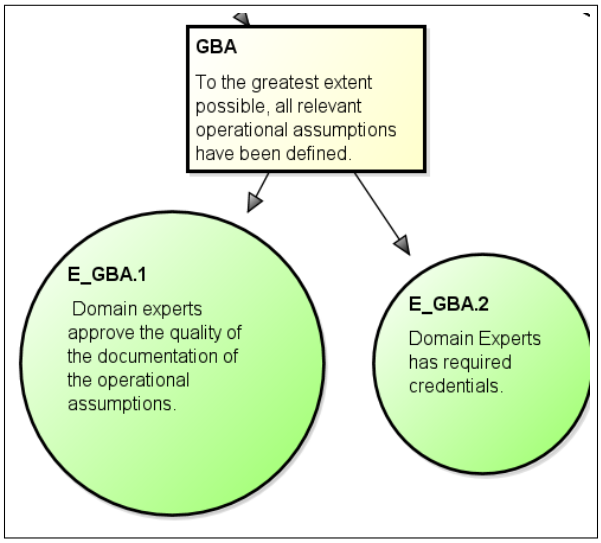


Figure 3.23: AGR Assurance Case Operational Assumptions

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 experts 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 I want 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 3.24, 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, assuming that the users might not be the repository contributors.

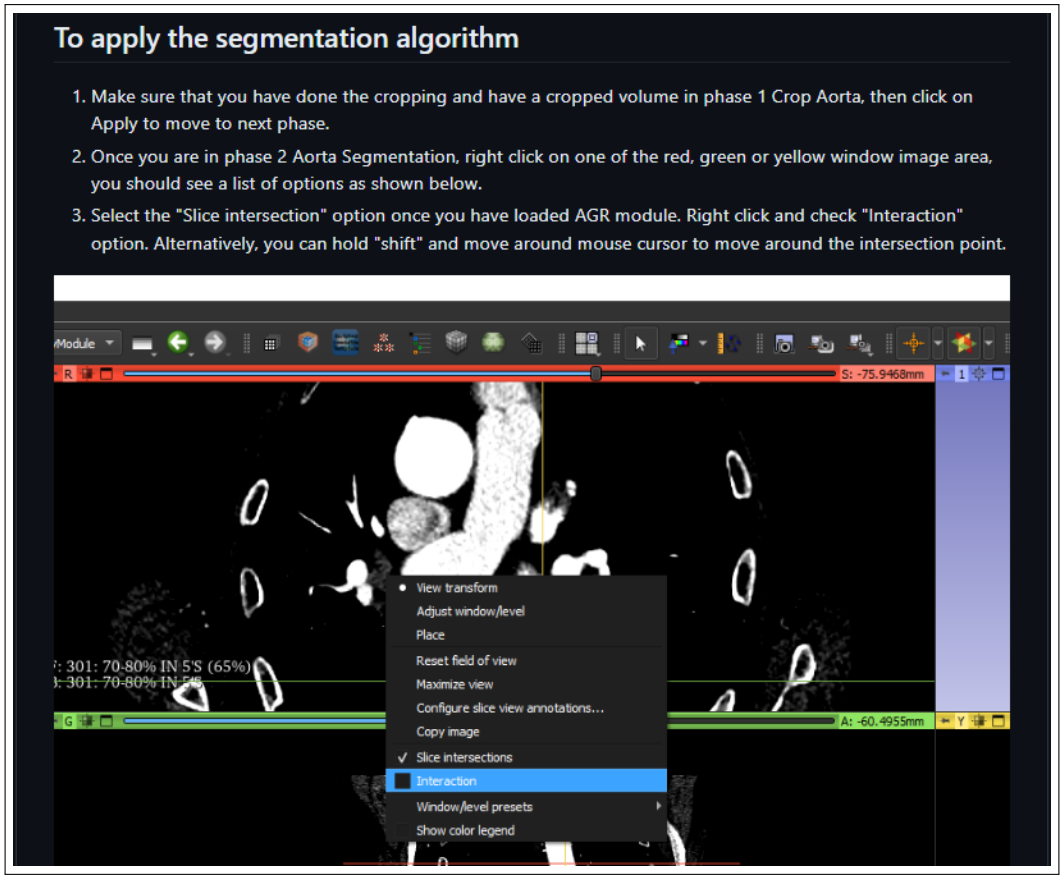


Figure 3.24: AGR User Manual On GitHub README

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 I make step-by-step instruction with voice over to instruct user. The Figure 3.25 shows the playing 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.

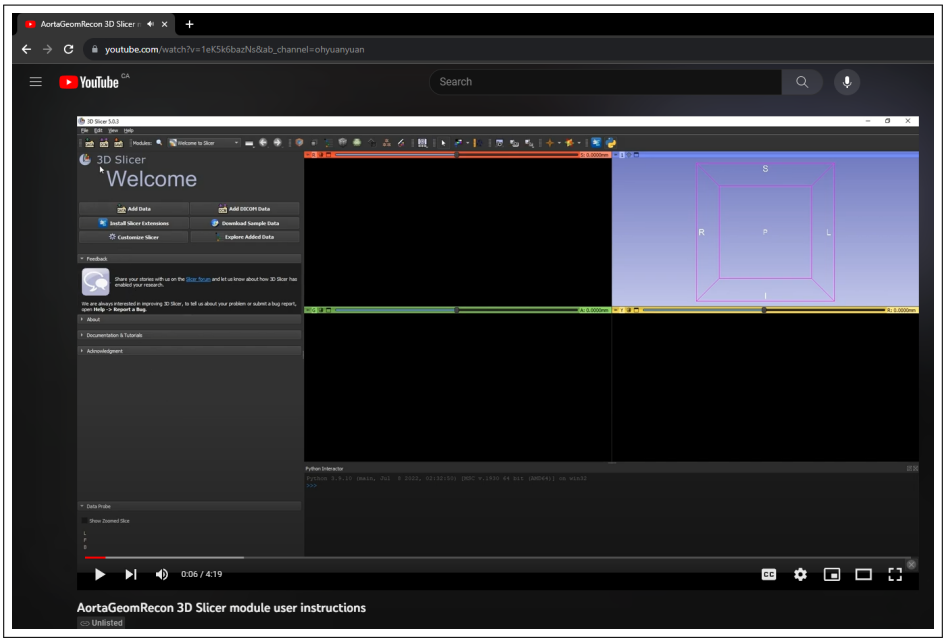


Figure 3.25: AGR User Instructions on YouTube

3.6 Assurance Case for Inputs Assumptions

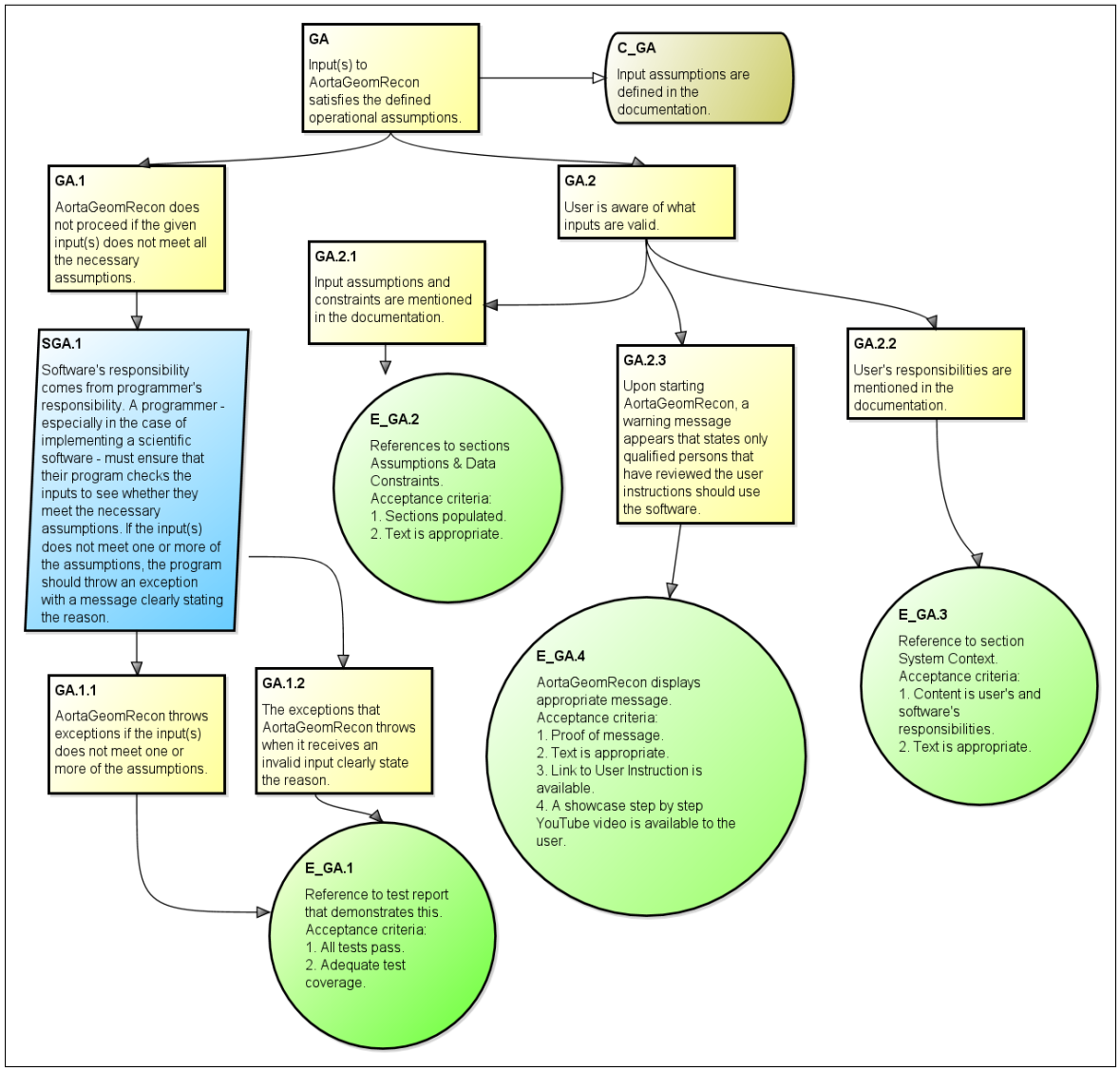


Figure 3.26: AGR Assurance Case Inputs Assumptions

The Figure 3.26 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. When the software gets unexpected inputs, it should not proceed to the next step, which could

result in unexpected ~~outputs~~.

3.6.1 AortaGoemRecon'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 proceed ~~into~~ 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 will 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. This implies that our evidence E_GA.1 is satisfied.

3.6.2 Warning Message

As I initially planned, the references to sections Assumptions, Data Constraints, and System Context is available in the User Manual and User Instruction Video, where I showed the user how to import DICOM patient's data, and operate on the inputs' data ~~till I~~ 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, I need to effectively guide the user to the User Manual, whether the user has used this software before or is a first time user.

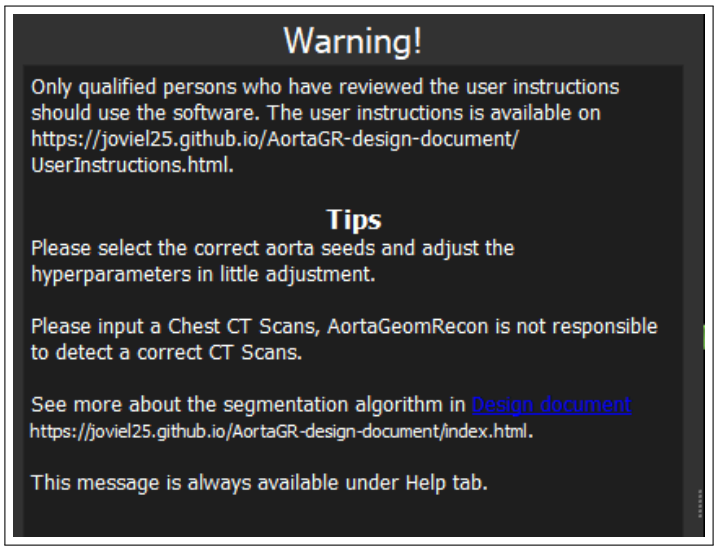


Figure 3.27: AGR Warning Message

As mentioned in the section 2.3.3.2, when the user first starts 3D Slicer and click on the AGR module, this warning message appears, which is also referred as the appropriate message 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.