Project Title: Unit Verification and Validation Plan for MISEG

Ao Dong

December 17, 2019

1 Revision History

Date	Version	Notes
Dec 14	1.0	Initial Draft

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

For the other symbols, abbreviations and acronym, see SRS Documentation at https://github.com/Ao99/MIA/blob/master/docs/SRS/SRS.pdf

This document describes procedures concerning the unit testing for MISEG for compliance with the requirements.

Some general information such as introduction to the software and testing scopes are included in Section 3. Verification plans and test tools are in Section 4.

3 General Information

3.1 Purpose

The software going through the test is Medical Imaging Segmentation (MISEG).

Segmentation, separation of structures of interest from the background and from each other Bankman (2000). Image segmentation is the process of partitioning an image into different meaningful segments. In medical imaging, these segments often correspond to different tissue classes, organs, pathologies, or other biologically relevant structures Forouzanfar et al. (2010).

MISEG uses one of many segmentation algorithms - the Intensity Threshold method. It also uses Otsu's Method to find the optimal threshold value(s). After receiving input medical image from the users, MISEG calculates the optimal threshold value(s), and output the processed segmentation image.

3.2 Scope

The Input Module relies on a library developed by others, so it will not be tested within the scope of this document.

The modules that are going to be tested are Image Data Structure, Optimal Thresholds Calculation, Output Module and Image Verification.

4 Plan

4.1 Verification and Validation Team

- Ao Dong
- Prof. Spencer Smith
- Peter Michalski
- Zhi Zhang
- Sasha Soraine
- Sharon Wu

4.2 Automated Testing and Verification Tools

The Java unit testing framework JUnit will be used to test the modules.

4.3 Non-Testing Based Verification

The non-testing based verification will be done in the System VnV.

5 Unit Test Description

All modules should meet the specifications defined in MIS Dong (2019).

5.1 Tests for Functional Requirements

5.1.1 Image Data Structure

This is an abstract data type module. It should correctly construct a data type instance if valid parameters are given, and throw exceptions if invalid ones are given.

1. Valid parameter

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.validSize, SYMBOLIC_CONSTANTS.validSize, SYMBOLIC_CONSTANTS.validSequence

   Output: A message saying that a new ImageData is successfully constructed, with the width, length and pixel value sequence displayed.

   Test Case Derivation: A new ImageData is successfully constructed and no exception is thrown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

2. Invalid parameter1

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.negSize, SYMBOLIC_CONSTANTS.validSize, SYMBOLIC_CONSTANTS.validSequence

   Output: A message saying that a new ImageData is not successfully constructed, with the badWidthInput error message from MIS shown.

Test Case Derivation: A new ImageData is not successfully constructed and the correct exception is thrown.

How test will be performed: It will be performed by test classes built with the help of JUnit.

3. Invalid parameter2

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.validSize, SYMBOLIC_CONSTANTS.negSize, SYMBOLIC_CONSTANTS.validSequence

   Output: A message saying that a new ImageData is not successfully constructed, with the badWidthInput error message from MIS shown.

   Test Case Derivation: A new ImageData is not successfully constructed and the correct exception is thrown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

4. Invalid parameter3

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.validSize2, SYMBOLIC_CONSTANTS.validSize2, SYMBOLIC_CONSTANTS.validSequence

   Output: A message saying that a new ImageData is not successfully constructed, with the badPixelValueLength error message from MIS shown.

   Test Case Derivation: A new ImageData is not successfully constructed and the correct exception is thrown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

5.1.2 Optimal Thresholds Calculation

This module should update chosenThresNum according to user's choice, and show an error message if invalid input is given.

1. Valid choice input

   Type: Automatic

Initial State: none.

Input: SYMBOLIC_CONSTANTS.validChoice

Output: A message saying that chosenThresNum is successfully updated, with the value of chosenThresNum displayed.

Test Case Derivation: chosenThresNum is successfully updated and no exception is thrown.

How test will be performed: It will be performed by test classes built with the help of JUnit.

2. Inalid choice input1

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.invalidChoice1

   Output: A message saying that chosenThresNum is not successfully updated, with the badChoiceInput error message from MIS shown.

   Test Case Derivation: chosenThresNum is not successfully updated and the correct exception is thrown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

3. Inalid choice input2

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.invalidChoice2

   Output: A message saying that chosenThresNum is not successfully updated, with the badChoiceInput error message from MIS shown.

   Test Case Derivation: chosenThresNum is not successfully updated and the correct exception is thrown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

5.1.3 Output Module

This module should correctly segment input images.

1. Segmentation

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.validImg1

   Output: A correctly segmented instance of ImageData, with width 100, height 100, and a valid pixelValue which is a sequence of $\mathbf{N}$ with elements in $[0, 255]$ This sequence should be the same as the one documented in This sequence is documented in test/segImage.txt

   Test Case Derivation: segImage is successfully updated and the result is correct.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

5.1.4 Image Verification

This module should correctly return verification result and throw proper exceptions.

1. Valid image

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.validImg1

   Output: Boolean true and no error message.

   Test Case Derivation: a valid instance of ImageData is correctly identified.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

2. Invalid image1

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.invalidImg1

   Output: Boolean false, with the badSize error message from MIS shown.

   Test Case Derivation: an invalid instance of ImageData is correctly identified and the proper error message is shown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

3. Invalid image2

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.invalidImg2

   Output: Boolean false, with the badPixelData error message from MIS shown.

   Test Case Derivation: an invalid instance of ImageData is correctly identified and the proper error message is shown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

4. Consistent sizes

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.validImg1, SYMBOLIC_CONSTANTS.validImg1

   Output: Boolean true and no error message.

   Test Case Derivation: two instances of ImageData with the same size are correctly identified.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

5. Inconsistent sizes

   Type: Automatic

   Initial State: none.

   Input: SYMBOLIC_CONSTANTS.validImg1, SYMBOLIC_CONSTANTS.validImg2

   Output: Boolean false, with the badSize2 error message from MIS shown.

   Test Case Derivation: two instances of ImageData with different sizes are correctly identified and the proper error message is shown.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

5.2 Tests for Nonfunctional Requirements

The performance of module Optimal Thresholds Calculation will be tested.

5.2.1 Optimal Thresholds Calculation

1. Max thresholds calculation speed

   Type: Automatic

   Initial State: none

   Input: SYMBOLIC_CONSTANTS.maxThresNum

   Output: Time used to finish the optimal thresholds calculation.

   How test will be performed: It will be performed by test classes built with the help of JUnit.

5.3 Traceability Between Test Cases and Modules

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Table 1 shows the dependencies between the test cases and the requirements.

Test Cases	Modules
5.1.1	Image Data Structure
5.1.2, 5.2.1	Optimal Thresholds Calculation
5.1.3	Output Module
5.1.4	Image Verification

Table 1: Traceability Matrix showing the connections between modules and tests

References

Isaac N. Bankman. Preface. In Isaac N. Bankman, editor, *Handbook of Medical Imaging*, Biomedical Engineering, pages xi – xii. Academic Press, San Diego, 2000. ISBN 978-0-12-077790-7. doi: https://doi.org/10.1016/B978-012077790-7/50001-1. URL http://www.sciencedirect.com/science/article/pii/B9780120777907500011.

Ao Dong. Module interface specification for miseg. 2019. URL https://github.com/Ao99/MISEG/blob/master/docs/Design/MIS/MIS.pdf.

Mohamad Forouzanfar, Nosratallah Forghani, and Mohammad Teshnehlab. Parameter optimization of improved fuzzy c-means clustering algorithm for brain mr image segmentation. *Engineering Applications of Artificial Intelligence*, 23(2):160 – 168, 2010. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2009.10.002. URL http://www.sciencedirect.com/science/article/pii/S095219760900150X.

6 Appendix

6.1 Symbolic Parameters

- SYMBOLIC_CONSTANTS.negSize = -1
- SYMBOLIC_CONSTANTS.validSize = 100
- SYMBOLIC_CONSTANTS.validSize2 = 200
- SYMBOLIC_CONSTANTS.validSequence = $< 1, 1, ..., 1 >$ which is a sequence of **N** with length of 10000, and every element is 1.
- SYMBOLIC_CONSTANTS.validChoice = 2
- SYMBOLIC_CONSTANTS.invalidChoice1 = 6
- SYMBOLIC_CONSTANTS.invalidChoice2 = A
- SYMBOLIC_CONSTANTS.validImg1, an instance of ImageData, with width 100, height 100, and a valid pixelValue which is a sequence of **N** with elements in $[0, 255]$. This sequence is documented in test/inputImage.txt
- SYMBOLIC_CONSTANTS.validImg2, an instance of ImageData, with width 200, height 200, and a valid pixelValue which is a sequence of **N** with elements in $[0, 255]$.
- SYMBOLIC_CONSTANTS.invalidImg1, an instance of ImageData, with width -1, height -1, and a valid pixelValue which is a sequence of **N** with elements in $[0, 255]$.
- SYMBOLIC_CONSTANTS.invalidImg2, an instance of ImageData, with width 100, height 100, and a valid pixelValue which is a sequence of **N** containing elements -1 and 256.
- SYMBOLIC_CONSTANTS.maxThresNum = 2.