SE 3XA3: Test Plan
TheLenaProject

Team 1, Team NAR
Nezar Dimitri - dimitn
Abeed Alibhai - alibhaa
Rahul Bablani - bablanr

December 8, 2016

Contents

1 General Information **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Acronyms, Abbreviations, and Symbols . . . 1
- 1.4 Overview of Document . . . 3

2 Plan **3**
- 2.1 Software Description . . . 3
- 2.2 Test Team . . . 3
- 2.3 Automated Testing Approach . . . 3
- 2.4 Testing Tools . . . 3
- 2.5 Testing Schedule . . . 4

3 System Test Description **4**
- 3.1 Tests for Functional Requirements . . . 4
- 3.2 Tests for Nonfunctional Requirements . . . 8

4 Tests for Proof of Concept **11**
- 4.1 Tests for Functional Requirements . . . 11
- 4.2 Tests for Nonfunctional Requirements . . . 12

5 Comparison to Existing Implementation **13**

6 Unit Testing Plan **13**
- 6.1 Unit testing of internal functions . . . 14
- 6.2 Unit testing of output files . . . 14

7 Appendix **16**
- 7.1 Symbolic Parameters . . . 16
- 7.2 Usability Survey Questions? . . . 16

List of Tables

1 **Revision History** . . . ii
2 **Table of Abbreviations** . . . 1
3 **Table of Definitions** . . . 2

Table 1: **Revision History**

Date	**Version**	**Notes**
October 21 2016	0.0	Rev0 - Completed Revision 0: General Information and Unit Testing Plan
October 24 2016	0.1	Rev0 - Completed Revision 0: Plan, System Test Description and Appendix
October 27 2016	0.2	Rev0 - Completed Revision 0: Comparison to Existing Implementation and Tests for Proof of Concept
December 7 2016	1.0	Rev1 - • Fixed grammar • Added abbreviations to table • Code coverage • Tests that throw exceptions

1 General Information

This section provides general information for TheLenaProject.

1.1 Purpose

The purpose of this project, is to re-implement the open source image processing project Marvin Frameworks. This Java framework allows clients to upload and process normal photos, into elegant images by applying one of the various filters available in our application, much like Instagram does. Once properly implemented, the processed image will then be available to export as an image file for use in social media, advertisements, etc.

1.2 Scope

This Test Plan is focused on describing how each test case should be constructed and implemented, showing appropriate inputs and expected outputs. It will not go in-depth how each function being tested is implemented

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
JPEG/JPG:	Joint Photographic Experts Group (Older version of JPG)
JAR:	Java Archive
GUI:	Graphical User Interface
PNG:	Portable Network Graphics
EULA:	End User License Agreement
SVG:	Scalable Vector Graphics
BMP:	Bitmap Image File
TIF:	Tagged Image Format File

Table 3: **Table of Definitions**

Term	**Definition**
Image Filter:	A software routine that changes the appearance of an image or part of an image by altering the shades and colours of the pixels in some manner. Filters are used to increase brightness and contrast as well as to add a wide variety of textures, tones and special effects to a picture.
Image Processing:	The analysis and manipulation of a digitized image, especially in order to improve its quality.
Framework:	In computer programming, a framework is an abstraction in which common code providing generic functionality can be selectively overridden or specialized by user code providing specific functionality.
JUnit:	A unit testing framework for the Java programming language. It plays a crucial role test-driven development

1.4 Overview of Document

This test plan document for TheLenaProject revision 0, will primarily be responsible for addressing the testing planning of the various functions implemented within our project.

2 Plan

This section outlines the plan for TheLenaProject

2.1 Software Description

The software of this project applies a Java framework that allows clients to upload and process normal photos, into elegant images by applying one of the various filters available in our application, much like Instagram does. Once properly implemented, the processed image will then be available to export as an image file for use in social media, advertisements, etc.

2.2 Test Team

The following project members are responsible for all procedures of the validation process including writing and executing tests:

- Nezar Dimitri
- Abeed Alibhai
- Rahul Bablani

2.3 Automated Testing Approach

Automated testing will be conducted using JUnit and will be used to test major functions.

2.4 Testing Tools

- J-Unit will be implemented as the primary testing framework to run tests on all sections of the code.

- Different images and file types will be used to test the file input and output of the application.
- Black box testing will be done on Windows, Mac, and Linux based operating systems to ensure all functional and non-functional requirements are fulfilled on all platforms.
- To report and measure Java code coverage we will use the tool JCov. JCov is capable of reporting the following types of code coverage: Block coverage, Line coverage, Branch coverage, Method coverage.
- For manual testing, we will be using Surveys for the focus groups and word processing for keeping track of test cases.

2.5 Testing Schedule

See Gantt Chart by clicking the following hyperlink: Gantt Chart

3 System Test Description

This section provides information about system tests within our project.

3.1 Tests for Functional Requirements

1. Filter Type: Functional Dynamic

   Initial State: Picture is displayed in the form it was originally uploaded in

   Input: Click each processing button (test will be repeated for each button, not clicked concurrently)

   Output: The picture will be re displayed but the filter we be applied and it will be processed from its original state.

   How test will be performed: The program will be ran and an image will be uploaded. One of the processing buttons will then be clicked and the corresponding filter should be applied. These steps will be repeated for all the processing options.

2. Multiple Filters Type: Functional Dynamic

   Initial State: Picture is displayed with one filter already applied to it

   Input: Click each processing button

   Output: The picture will be re displayed but the filter will be applied and it will once again be processed form and already processed form

   How the test will be performed: The program will be ran, a picture will be uploaded and then two filters will be applied without resetting the image.

3. Upload Type: Functional Dynamic

   Initial State: No picture is uploaded or displayed

   Input: Upload an image with file extensions (JPG, PNG, JPEG)

   Output: Picture will successfully be uploaded and displayed.

   How the test will be performed: when prompted a file extension will be input, next the file path and then a file location. The GUI will open and the image will be displayed

4. Upload Type: Functional Dynamic

   Initial State: No picture is uploaded or displayed

   Input: Upload an image with file extensions (SVG)

   Output: Nothing happens, no picture or GUI opened.

   How the test will be performed: when prompted a file extension will be input, next the file path and then a file location. The GUI will open and the image will be displayed

5. Upload Type: Functional Dynamic

   Initial State: No picture is uploaded or displayed

   Input: Upload an image with file extensions (Word file - .DOC)

   Output: Nothing happens, no picture or GUI opened.

   How the test will be performed: when prompted a file extension will be input, next the file path and then a file location. The GUI will open and the image will be displayed

6. Upload Type: Functional Dynamic

   Initial State: No picture is uploaded or displayed

   Input: Upload an image with file extensions (Corrupted PNG)

   Output: Nothing happens, no picture or GUI opened.

   How the test will be performed: when prompted a file extension will be input, next the file path and then a file location. The GUI will open and the image will be displayed

7. Save Type: Functional Dynamic

   Initial State: Picture is displayed in app and a filter has been applied

   Input: Save a .PNG image

   Output: Image will be saved in desired location

   How the test will be performed: A picture will be uploaded then filtered. The save button will be clicked. The following prompts will be filled in with the desired file extension, file path and file name.

8. Save Type: Functional Dynamic

   Initial State: Picture is displayed in app and a filter has been applied

Input: Save a .SVG image

Output: Nothing will be saved

How the test will be performed: A picture will be uploaded then filtered. The save button will be clicked. The following prompts will be filled in with the desired file extension, file path and file name.

9. Save Type: Functional Dynamic

   Initial State: Picture is displayed in app and a filter has been applied

   Input: Save an image with no extension

   Output: Handles exception error by saving as imported image file type

   How the test will be performed: A picture will be uploaded then filtered. The save button will be clicked. The following prompts will be filled in with the desired file extension, file path and file name.

10. Resize Type: Functional Dynamic

    Initial State: Image processing app is open

    Input: The Minimize "-", maximize "O" and close buttons "x" are pressed

    Output: the app will Minimize and Maximize then close

    How the test will be performed: The app will be opened. While open the three mentioned buttons will be clicked.

11. Manual Resize Type: Functional Dynamic

    Initial State: Image processing app is open

    Input: The bottom corner will be clicked then dragged

Output: The app window will resize to the desired position

How the test will be performed: The app will be opened. While open a mouse will be used to click and the bottom corner to resize the window

3.2 Tests for Nonfunctional Requirements

1. Performance Type: Structural Dynamic Automated

   Initial State: Program has no GUI or picture uploaded

   Input: The Enter button is clicked

   Output: The GUI will open in PROCESS_TIME with the image uploaded and displayed in UPLOAD_TIME

   How the test will be performed: A timer will be implemented to start right after the enter button is clicked. The timer will then stop after all the uploading and GUI commands are completed.

2. Reliability Type: Structural Static Manual

   Purpose: Test will look for any extreme cases that may not be handled with exceptions

   How the test will be performed: Will go through code and analyze any deterministic sections of code (if statements, user inputs, file types) to catch any boundary cases that need to be handled

3. Look and Feel Type: Structural Dynamic Manual

   Initial State: No GUI has been launched

   Input: Application is launched and picture is uploaded

   Output: GUI will be built and image will appear in the application

   How the test will be performed: The GUI will launched multiple times

with different images uploaded to ensure they all amount to the same final display.

4. Environmental Type: Functional Dynamic manual

   Initial State: No picture is uploaded or displayed

   Input: Upload an image with file extensions (JPG, PNG, JPEG)

   Output: Picture will successfully be uploaded and displayed.

   How the test will be performed: When prompted a file extension will be input, next the file path and then a file location. The GUI will open and the image will be displayed

5. Quality Type: Structural Dynamic Manual

   Initial State: Image is uploaded and displayed on GUI

   Input: Process the image with multiple filters

   Output: The image will process according to the filters chosen but the quality will not be affected

   How the test will be performed: The app will be opened and an image with QUALITY amount of pixels. The image will go through N changes and saved. The saved image should have QUALITY amount of pixels.

6. Usability Type: Structural Dynamic Automated

   Initial State: Closed application

   Input: Variety of inputs to run through the program

   Output: User will have saved a filtered image

   How the test will be performed: A function will be written to count the number of clicks that occur while the program is running. A User

will be asked to start, upload, filer then save and image. The number of clicks will be recorded and should equal CLICKS

7. Maintainability Type: Functional Dynamic Manual

   Initial State: Program is stored on USB

   Input: Launch Application

   Output: Application will open

   How the test will be performed: The program files will be transferred onto a USB. We will then run the program on a different computer straight off the USB.

8. Security Type: Structural Static Manual

   Purpose: Check to see if Users can access the complex algorithms from the Marvin Framework that are passed into our program

   How the test will be performed: The code will be scrutinized for any access point to the Marvin Framework algorithms

9. Cultural Type: Structural Static Manual

   Purpose: To find any aspects of the program that maybe offensive to some users

   How the test will be performed: A survey will be held asking users if they found any aspects of our program to be culturally offensive

10. legal Type: Structural Static Manual

    Purpose: To ensure no legal suits can be held against us

    How the test will be performed: We will manually go through all the patents that relate to image processing

11. Health and Safety Type: Static Manual

Purpose: Identify the amount of strain our program puts on a users eye

How the test will be performed: Users will be asked to spend ten minutes using our application and rate their eye strain on a scale of 1 to 10

4 Tests for Proof of Concept

This section provides tests for proof of concept within our project

4.1 Tests for Functional Requirements

1. Upload Type: Functional Dynamic

   Initial State: No picture is uploaded or displayed

   Input: Upload an image with file extensions (JPG, PNG, JPEG)

   Output: Picture will successfully be uploaded and displayed.

   How the test will be performed: when prompted a file extension will be input, next the file path and then a file location. The GUI will open and the image will be displayed

2. Save Type: Functional Dynamic

   Initial State: Picture is displayed in app and a filter has been applied

   Input: Save and image

   Output: Image will be saved in desired location

   How the test will be performed: A picture will be uploaded then filtered. The save button will be clicked. The following prompts will be filled in with the desired file extension, file path and file name.

3. Filter Type: Functional Dynamic

Initial State: Picture is displayed in the form it was originally uploaded in

Input: Click each processing button (test will be repeated for each button, not clicked concurrently)

Output: The picture will be re displayed but the filter we be applied and it will be processed from its original state.

How test will be performed: The program will be ran and an image will be uploaded. One of the processing buttons will then be clicked and the corresponding filter should be applied. These steps will be repeated for all the processing options.

4.2 Tests for Nonfunctional Requirements

1. Look and Feel Type: Structural Dynamic Manual

   Initial State: No GUI has been launched

   Input: Application is launched and picture is uploaded

   Output: GUI will be built and image will appear in the application

   How the test will be performed: The GUI will launched multiple times with different images uploaded to ensure they all amount to the same final display.

2. Environmental Type: Functional Dynamic manual

   Initial State: No picture is uploaded or displayed

   Input: Upload an image with file extensions (JPG, PNG, JPEG)

   Output: Picture will successfully be uploaded and displayed.

   How the test will be performed: When prompted a file extension will be input, next the file path and then a file location. The GUI will open

and the image will be displayed

3. Usability Type: Structural Dynamic Automated

   Initial State: Closed application

   Input: Variety of inputs to run through the program

   Output: User will have saved a filtered image

   How the test will be performed: A function will be written to count the number of clicks that occur while the program is running. A User will be asked to start, upload, filer then save and image. The number of clicks will be recorded and should equal CLICKS

5 Comparison to Existing Implementation

Compared to the existing implementation of the open source image processing project Marvin Frameworks we have added 3 major features that distinguish us from them. We have added a fairly simple but elegant interface that allows the user to import and export image files. The GUI is very simple to allow any regular computer user to easily navigate the application and successfully upload, customize and export their image. Our application has also been customized to allow a single image to be processed more than once with multiple filters to allow such features as: grayscale, invert and edge detection all at once on one single image as well as being able to process a single filter multiple times to intensify the effect of the filter. We have finally implemented an exporting tool to save the image as a .png or .jpg file which the original Marvin Frameworks project did not include. Next steps will be to implement error handling methodologies and look into the addition of video processing which may be out of the scope for the time restriction we are limited too but may be a future implementation of our application.

6 Unit Testing Plan

This section provides information about the unit testing plan within our project.

6.1 Unit testing of internal functions

Our project as of now has one internal function (besides the "main" function). This internal function is the **actionPerformed** function used to listen to the actions in the GUI. There are a series of if-statements for the logic connecting each button to the correct plug-ins, and subsequently applying the correct filters to the image, resetting the image back to it original state, or prompting the user to save the edited image whilst choosing a name and a destination for it.

There are 3 types of buttons associated to the GUI: filter buttons, a reset button, and a save button. The unit tests for this function will be reflected differently based on the type of button. For the filter buttons we will be testing the correctness of the filter applied based on the selected filter. We will test from different initial states, one where the photo is in its original state right before the filter is applied, and other where the photo has already been edited by another filter and the filter is being applied on top of that. The unit test will only pass if the correctly selected filter was applied without making any other alterations to the photo. The second type of button is the reset button, the unit test for this will be testing the correctness of the "reset" from any initial state and will only pass if the image is is fully restored to its original state. The final type of button is the save button. There will be two unit tests for this. The first one will be testing the checking if the user is prompted with the export interface to save their edited image, and will only pass if the interface is made available for the user. The second unit test is linked to the output files and will be explained in the next subsection (6.2 Unit testing of output files).

6.2 Unit testing of output files

The only output file generated in this project is the processed version of the image that the user desires to save to their local drive. This is possible through the save button, which subsequently prompts the user with the export interface where they can select the image type, destination folder, and name of the finished product. The first unit test for the save button was discussed in the previous subsection (6.1 Unit testing of internal functions), but as stated there, there is a second unit test which is linked to the output files. This will be testing the reliability, compatibility, as well as the actual correctness of the output file. The unit test will pass if, after the prompting

for export, the image is save with the correct formatting, with the right file path, and that the image is identical to how it looked in the preview window (with selected filters applied to it). This must consistently, work as well have some error handling to be able to notify the user if the selected format, file path, chosen name are invalid.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

- PROCESSING_TIME = 3 Seconds
- UPLOAD_TIME = 2 Seconds
- RESPONSE_TIME = 1/2 a Second
- STORAGE = Disk Space of User
- QUALITY = Number of pixels of uploaded image
- CLICKS = 8

7.2 Usability Survey Questions?

We plan to hold a small beta test, targeting a wide variety of age groups and computer skill levels in order to determine the ease of usability for our application. Such questions may be asked to our test group:

- Does the application open on your computer? If so, how long does it take?
- Are you able to successfully upload and save your file?
- Are you able to successfully apply filter(s)?
- On a scale from 1-10 how appealing was the look of the app?
- How many clicks were needed to: open, filter and save an image
- How easy on a scale from 1-10 was the application to use?
- Any feedback on how to improve our application?