SE 3XA3: Test Report
The Lena Project

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

December 8, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Dec 7, 2016	1.0	Completed Initial Report
Dec 8, 2016	1.1	Proof read with some minor changes

1 Overview

This Test report will cover the testing that was undertaken during the completion of The Lena Project. It will summarize the manual and automated testing for the front and back end of the project. Test cases for valid and abnormal inputs have been made, any changes that were made in response to test results have been documented throughout the report.

The focus of the project as well as many of the requirements are based around usability and performance, thus manual testing was crucial to the success of the project. Using various metrics, thorough test cases were designed to test multiple non-functional requirements. Automated testing was used for quick accurate results which would not have been possible with manual.

The GUI has been made for this project, to allow users to effortlessly interact with the application. Manual testing will be used for this component of the project with inputs being clicks and outputs being the response the GUI has to each click, these will all be visually inspected.

2 Functional Requirements Evaluation

Testing of the functional requirements were created through a series of manual unit tests majority of which follow a black box testing approach. The testing was used to ensure certain functional requirements are met and to verify the correct results occurred after a specific input.

3 Nonfunctional Requirements Evaluation

To test many of the non-functional requirements a survey was designed to cover an array of requirements. This survey was given to the beta testers to fill out. The reason for the implementation of the survey was because success of the non-functional requirements were heavily dependent on users.

Table 2: **Test Results**

Test Case	Initial State	Input	Expected Output	Result
#1	Picture is displayed in the form it was originally uploaded in	Click each processing button (test will be repeated for each button, not clicked concurrently)	The picture will be re displayed but the filter we be applied and it will be processed from its original state.	PASS
#2	Picture is displayed with one filter already applied to it	Click each processing button	The picture will be re displayed but the filter will be applied and it will once again be processed form and already processed form	PASS
#3	No picture is uploaded or displayed	Upload an image with file extensions (JPG, PNG, JPEG)	Picture will successfully be uploaded and displayed.	PASS
#4	Picture is displayed in app and a filter has been applied	Save an image	Image will be saved in desired location	PASS
#5	Image processing app is open	The Minimize "-", maximize "O" and close buttons "x" are pressed	the app will Minimize, Maximize then close	PASS
#6	Image processing app is open	The bottom corner will be clicked then dragged	The app window will resize to the desired position	PASS

3.1 Look and Feel

Survey Question: On a scale from 1-10 how appealing was the look of the app?
Response: Average of 8/10

Test Case	Initial State	Input	Expected Output	Result
#3	No GUI has been launched	Application is launched and picture is uploaded	GUI will be built and image will appear in the application	PASS

3.2 Usability

Survey Question: How many clicks were needed to: open, filter and save an image
Response: An average of 8 clicks
Survey Question: How easy on a scale from 1-10 was the application to use?
Response: rating of 10

Test Case	Initial State	Input	Expected Output	Result
#6	Closed application	Variety of inputs to run through the program	User will have saved a filtered image	PASS

3.3 Performance

This section is broken down into related subsections.

3.3.1 Speed

Test Case	Initial State	Input	Expected Output	Result
#1	Program has no main GUI or picture uploaded(still in file explorer)	Image is selected	The GUI will open in PROCESS TIME with the image uploaded and displayed in UPLOAD TIME	Refer to graph below

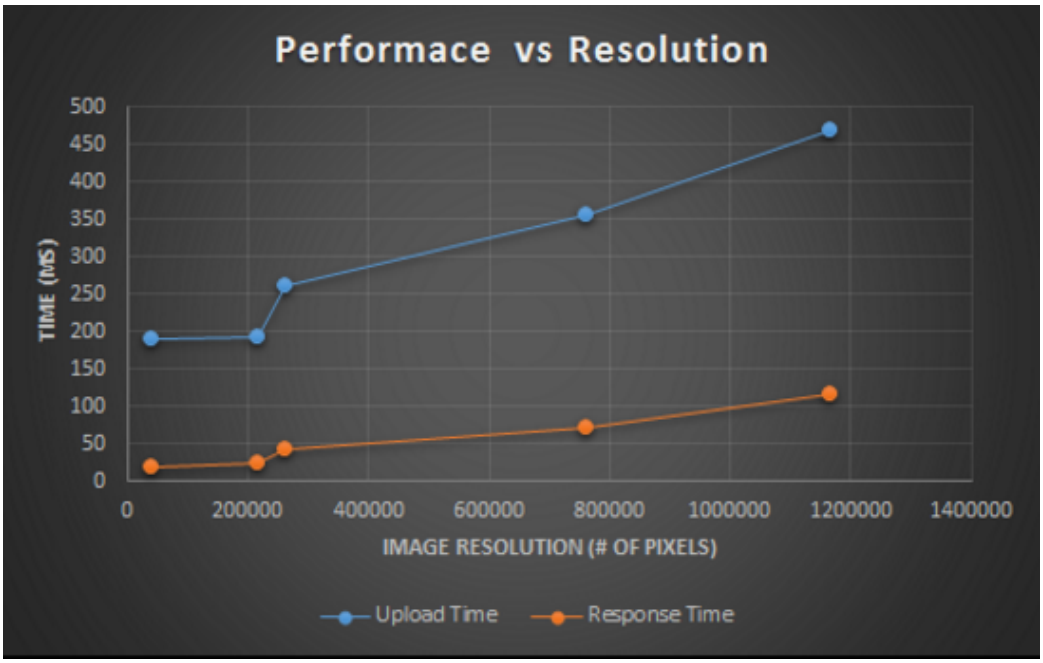

Figure 1: Performance Graph

3.3.2 Reliability

Test Case	Purpose	Input	Result
#2	Test will look for any extreme cases that may not be handled with exceptions	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	PASS Found errors in exception handling

3.3.3 Quality

Test Case	Initial State	Input	Expected Output	Result
#5	Image is uploaded and displayed on GUI	Process the image with multiple filters	The image will process according to the filters chosen but the quality will not be affected	PASS

3.4 Operational/Environment

Test Case	Initial State	Input	Expected Output	Result
#4	No picture is uploaded or displayed	Upload an image with file extensions (JPG, PNG, JPEG)	Picture will successfully be uploaded and displayed.	PASS

3.5 Maintainability

Test Case	Initial State	Input	Expected Output	Result
#7	Program is stored on USB	Launch Application	Application will open	PASS

3.6 Security

Test Case	Purpose	Input	Result
#8	Check to see if Users can access the complex algorithms from the Marvin Framework that are passed into our program	The code will be scrutinized for any access point to the Marvin Framework algorithms	PASS There are no points of access to the framework

3.7 Cultural

Test Case	Purpose	Input	Result
#9	To find any aspects of the program that maybe offensive to some users	A survey will be held asking users if they found any aspects of our program to be culturally offensive	PASS No offensive aspects of the program were reported in the survey

3.8 Legal

Test Case	Purpose	Input	Result
#10	To ensure no legal suits can be held against us	We will manually go through all the patents that relate to image processing	PASS Marvin is open sourced therefore we are legally entitled to use their resources

3.9 Health and Safety

Test Case	Purpose	Input	Result
#11	Identify the amount of strain our program puts on a users eye	Users will be asked to spend ten minutes using our application and rate their eye strain on a scale of 1 to 10	PASS Eye strain results had a consensus of a 1/10 rating

3.10 Robustness

The Lena Processing application only supports uncorrupted Raster image file formats. This includes processes of opening and saving an image. To test robustness we attempted openning file types that were accepted, not accepted, and corrupted. We then tried to save the image by renaming the extension of a type that is accepted, not accepted, and the boundary case of no extension.

4 Comparison to Existing Implementation

This section does not apply to our project.

5 Unit Testing

Our unit testing plan broke down into the testing of internal functions and testing of output files.

5.1 Internal Functions

The main internal function of our project is the actionPerformed module which is used to listen to the actions in the GUI. A series of if-statements connect each button to the correct plugins subsequently applying the correct filter or correct action. Each button was tested based on its functionality. All the filter buttons correctly modified the photo to apply their designated filter. The reset button was successful in restoring the image to its original state. Lastly the save button does its job in opening the file explorer to allow for users to choose their desired file destination.

5.2 Output Files

The output file is the processed version of the users original image that they would like to save onto their local drive. The save button mentioned in the section above opens the file explorer which prompts the user to select a file type, a destination folder, and a name for the image which they wish to save. The testing of this included trying to save images with different file formats. The tests passed when the user tried to save the image with a correct file extension. The tests however failed when an incorrect or unsupported file format was chosen. We have implemented new safety measures to correct the failure in these test. These implementations are explained in the Changes Due to Testing section.

Table 3: **Opening**

Test Case	Input	Expected Output	Result
#12	PNG/JPG/GIF	Picture will open in main GUI	PASS
#13	SVG	Picture will open in main GUI	FAIL Nothing happened, no picture or GUI opened
#14	Word File	Warning of wring file type will appear	FAIL Nothing happened, no picture or GUI opened
#15	Corrupted PNG	Warning of wring file type will appear	FAIL Nothing happened, no picture or GUI opened

Table 4: **Saving**

Test Case	Input	Expected Output	Result
#16	.PNG	Picture will save to desired directory	PASS
#17	.SVG	Warning of wrong file extension will appear	FAIL Nothing is saved
#18	No extension	Save as default extension	FAIL Nothing is saved

6 Changes Due to Testing

This section of the document will outline all the Changes were made to the project based on the aspects of testing that had failed. Error handling and robustness was one of the largest areas of failure in our testing. We knew we needed to add safety or limitation so users wouldnt run into such issues. In addition our surveys feedback section revealed features of The Lena Project that could have been improved on or added. We took this into consideration and implemented many of the features that we had initially not thought of.

6.1 Error Handling

This section is broken down into opening and saving.

6.1.1 Opening

Our initial thought to solving the issue of opening wrong file formats was to just issue a warning prompt. We instead implemented a check in the file explorer so users are only able to choose an image that has a file format our app accepts.

6.1.2 Saving

The way we implemented safety towards file saving was very similar to the way we implemented it for opening a file. The issue was that file saving was very dependent on user input as the file name comes from a string. The first step was to allow any file extension that was of a type our app supported. If there was no file extension given, the image would be saved with the same file extension as the imported image. If the file extension was not supported or there was an issue in the way the user input the file name an error prompt appeared and the user was forced to rename the project.

6.2 Additions

The survey we implemented asked users to give us feedback based on any problems they faced or any features they would like to see added. This section of the survey deemed to be much more useful than originally expected and led to many changes to out app.

Feedback: "Picture wont fit on screen"
Solution: We realized that different image sizes would open differently on users computers. To compensate for this we made it possible for the user to resize the image and the window.

Feedback: "Cant go back?"
Solution: It is inevitable that users will make mistakes or will want to change any actions they performed. We decided to add an undo button so users can take back any changes they made.

Feedback: "I wanted to filter a different photo"
Solution: Originally opening file was only an option when the application first opened. An open button was added so users can choose a different image even with the GUI already opened.

7 Automated Testing

Most of our project is based on buttons, clicks, and action listeners. This model leaves little room for automated testing as most test cases are run manually. However file import and export is a very crucial part of the testing and with many different possibilities of file formats it became necessary to include automated testing. JUnit was implemented to automatically test the file open and file save modules. Different file types were inputted into the modules to analyze whether the output was the correct result. When supported file formats were put in, the system acted as it should and the result was correct. When unsupported or unusual formats were inputted into the system crashed.

8 Trace to Requirements

Table 5: **Functional Tests**

Test Case	Req 1	Req 2	Req 3	Req 4	Req5
#1		X	X		
#2		X	X		
#3	X				
#4				X	
#5					X

Table 6: **Non Functional Tests**

Test Case	Look and Feel	Usability	Performance	Operational	Maintainability	Security	Cultural	Legal	Health and Safety	Robustness
#1			X							
#2			X							
#3	X									
#4				X						
#5			X							
#6		X								
#7					X					
#8						X				
#9							X			
#10								X		
#11									X	
12-18										X

9 Trace to Modules

Table 7: **Functional Tests**

Test Case	M1	M2	M3	M4
#1			X	X
#2			X	X
#3	X	X		
#4	X			
#5			X	
#6			X	

Table 8: **Non Functional Tests**

Test Case	M1	M2	M3	M4
#1		X	X	
#2	X			
#3		X	X	
#4	X			
#5			X	X
#6	X	X	X	X
#7	X			
#8				X
#9	X		X	
#10				X
#11	X		X	
12-18	X	X	X	

10 Code Coverage Metrics

We used JCov to measure and analyze dynamic code coverage of our app. When the app runs the code coverage data is collected. Unlike static code coverage JCov is able to collect the data while the program is being executed. We collected method, linear block, and branch coverage. JCov also shows our programs source code annotated with coverage information. In addition each function had test cases specifically made to cover all aspects of the code within the method. For example the FileExplorer test cases were designed such that all the if statements were executed (statement coverage).