

# SE 3XA3: Test Report

## Title of Project

Team #, Team Name  
Student 1 name and macid  
Student 2 name and macid  
Student 3 name and macid

December 8, 2016

# Contents

<b>1</b>	<b>Functional Requirements Evaluation</b>	<b>1</b>
1.1	Opening a GIF . . . . .	1
1.2	Save a GIF . . . . .	1
1.3	Start, Stop, Modify Length . . . . .	1
1.4	Save In Different File Format . . . . .	2
1.5	Help Context . . . . .	2
1.6	Resize . . . . .	2
<b>2</b>	<b>Nonfunctional Requirements Evaluation</b>	<b>2</b>
2.1	Performance . . . . .	2
<b>3</b>	<b>Unit Testing</b>	<b>2</b>
<b>4</b>	<b>Changes Due to Testing</b>	<b>3</b>
<b>5</b>	<b>Automated Testing</b>	<b>3</b>
<b>6</b>	<b>Trace to Requirements</b>	<b>4</b>
<b>7</b>	<b>Trace to Modules</b>	<b>5</b>
<b>8</b>	<b>Code Coverage Metrics</b>	<b>5</b>

## List of Tables

1	<b>Revision History</b> . . . . .	ii
2	Trace Between Requirements and Modules . . . . .	6

## List of Figures

Table 1: **Revision History**

<b>Date</b>	<b>Version</b>	<b>Notes</b>
Date 1	1.0	Notes
Date 2	1.1	Notes

This document ...

# **1 Functional Requirements Evaluation**

## **1.1 Opening a GIF**

There were four tests conducted to ensure that opening the GIF could be performed by Gifitti without any problems. The first test was the ability of the program to load a GIF. The program is loaded initially and a GIF is selected that will be loaded into memory, being displayed to the user. The results of this test matched the expected output. The next few tests pertain to scenarios where it would cause errors in the program. They all start with the program running but with different erroneous inputs such as no file path, null filepath, and incorrect file type. All these inputs do not cause any change in the running program which is the expected output. All the tests in opening a GIF file have been successful in determining proper functioning.

## **1.2 Save a GIF**

There are three tests conducted that verify that the GIF modified in the program is saved properly on the file system of the user's computer. They all start with an initial state of a GIF already loaded into the program and are saved either in known, chosen or non-existing locations. When the user selects a valid location, the program will save the GIF as it appears in the program but gives an error if you try to save it in a non-existent location. This is the expected output of the test performed.

## **1.3 Start, Stop, Modify Length**

The test pertaining to the stop and start of the animation has the initial conditions of a GIF already loaded in and will take input of a GIF either in an animation or stopped on a frame, with the start and stop button being pressed to pause or resume the animation. The program performs these functions as expected. The testing for the modification of the GIF was successful as the program takes a GIF and cuts it down to the specified frame length. For example, a GIF of 20 frames was successfully shortened to frames five to ten.

## **1.4 Save In Different File Format**

Four tests were conducted in the exact same fashion where a GIF, already loaded into the program, had its frames exported into an image format. These four tests were extremely successful in exporting the images in their specific file format (JPEG, PNG, TIFF, BMP) that did not overwrite each other. The output matched the expected output.

## **1.5 Help Context**

One test was performed to determine if the help section could be easily accessible and understandable. With the program loaded, the help tab was tested if it opened and the definitions given were viewable. This test proved to be successful in matching what the expected output was supposed to be.

## **1.6 Resize**

A GIF was loaded into the program and a specific size was specified by the user. The GIF would be resized to that specified size which matched the expected output of that function. There was also a test that went through the exact same process where it utilized edge case to determine if it crashed the program. As expected, it did not crash it.

# **2 Nonfunctional Requirements Evaluation**

## **2.1 Performance**

# **3 Unit Testing**

Unit testing has been done on system methods that return a value, or modify known objects in an expected way. A metric for passing is to be used on all methods with an error tolerance, such as manipulation of a GIF (where slight variations can exist in the output). A method with no expected error tolerance has been implemented as discreetly pass XOR fail. All unit test inputs fall in a known domain where the expected output can be assumed. The unit tests were utilized to ensure that certain edge cases were considered and that they did not crash the program.

To test the validity of outputted GIF images, the testing system holds image resources of input frames, input GIFs, as well as the expected GIF outputted. A known GIF was manipulated and then compared to the expected GIF in a frame-by-frame matter. To exclude meta-data issues associated with GIF images, each frame was cast to a known image format such as a bitmap, then compared pixel-by-pixel. Frames have had a 100 percent pixel match for the expected GIF to pass.

Any written unit test utilizes the C Sharp attributes for Unit Testing provided by the framework. This ensures the system does not build the testing package into the consumer version of the software.

## **4 Changes Due to Testing**

Through various test plans being enacted onto this program and other brute force testing by persons who had no understanding of code we were able to identify certain areas that were lacking. The general trend is that if a command is successfully inputted, that the expected output, that is the GIF, has been produced accurately for the majority of the time. The main issue the project had to be adjusted to is the different methods and input could be incorrectly given. For example, putting negative values for the frames wanted out of GIF. Through these testing methodologies, no big corrections were required but extensive testing revealed edge cases that were not previously considered that needed to be corrected. For example, one of the testers caught a work around the file selection to be able to select JPEG images as our input. Of course, further testing can always be done to catch a possible edge case that has not been considered before.

## **5 Automated Testing**

This test plan has utilized automated testing for the verification of the manipulation functionality that changes the GIF. That is to say that the exported GIF image will match what is created by the user of our product. The only tool to be utilized to test this product will be the Microsoft Unit Test Framework that is native with Visual Studio. Through this unit testing framework, we will have it set up that it automatically does comparisons

of the GIF output to the expected output where it can in a quick manner compare the pixel values to ensure complete accuracy. Throughout utilizing this framework, we have always gotten successful results which allowed us to identify other areas where the majority of the problems occurred.

## **6 Trace to Requirements**

## **7 Trace to Modules**

Refer to Test Plan to see corresponding Tests and Module Guide to see corresponding Modules.

Test	Modules
ID1	M5, M9
ID2	M5, M9
ID3	M5,, M9
ID4	M5, M9
ID5	M8
ID6	M8
ID7	M8
ID8	M8
ID9	MIL
ID10	MIL
ID11	MIL
ID12	M6
ID13	M6
ID14	M6
ID15	M6
ID16	MIL
ID17	MIL
ID18	MIL
ID19	MIL
ID20	MIL
ID21	MIL
ID22	MIL
ID23	MIL
ID24	MIL

Table 2: Trace Between Requirements and Modules



## 8 Code Coverage Metrics