

SE 3XA3: Test Report Gifitti

Team #2, Gifitti
Pavle Arezina arezinp
Riley McGee mcgeer
Nicolai Kozel kozeln

December 8, 2016

Contents

1	Functional Requirements Evaluation	1
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
2	Nonfunctional Requirements Evaluation	2
2.1	Performance	2
2.1.1	Speed	2
2.1.2	Precision	3
2.1.3	Safety Critical	3
3	Unit Testing	4
4	Changes Due to Testing	4
5	Automated Testing	5
6	Trace to Requirements	5
7	Trace to Modules	7
8	Code Coverage Metrics	9

List of Tables

1	Revision History	ii
2	Trace Between Test Cases and Requirements	6
3	Trace Between Test Cases and Modules	8

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Dec 5	1.0	Functional Testing
Dec 7	1.1	The rest

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 a 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

All tested non-functional requirements fall under the Performance Category of our system. These requirements were deemed most needed to have the system pass software revision one successfully.

2.1.1 Speed

Two tests were conducted to validate speed performance of Gifitti, one test for loading, another for modifying and exporting a GIF.

The C# debug timer was utilized to validate that exportation of a GIF, when the user invoked the process to export a sub GIF a time stamp was collected, which was then compared to the ending time stamp after export returned.

All collected times for GIFs with a frame length of 100 had export times strictly less than `MAX_EXPORT_TIME`.

For loading GIF images into Gifitti, a similar process was used. When the tester invoked the process to open a GIF a time stamp was collected, which was then compared to the ending time stamp after the system returned to its base state. All collected time differences for images of frame length 100 were recorded to load in a time strictly less than `MAX_UI_LOAD`.

Overall the system has been verified to uphold to speed performance requirements.

2.1.2 Precision

A single test case was conducted to validate precision. The test validates that exported sub GIF frames fully represent the testers request. The test selected the first five frames of an arbitrary GIF, using a 3rd party software system the GIF was dismantled, the export from the test case was compared to the complete set of ordered frames from the 3rd party software. The output of Gifitti matched the expected output.

2.1.3 Safety Critical

Three test cases were developed to validate safety critical components of Gifitti.

The first test had the tester export a set of frames to a known location, this step was then repeated. As user data integrity is vital to the usability of the system a prompt for checking if the user is okay with overwriting their data is required. Gifitti prompted the user, notifying them of this action, thus passing the test.

The second test had the tester save a GIF to a location with restricted access. In this case the restricted access was a lack of memory. A partitioned folder was created with a max folder size of 1kB on windows. This folder was then selected for save by the tester, with a loaded GIF. Windows denied the save and notified the user in the save dialog. Thus through only being usable on Windows systems the product passes.

The third test validates invalid user inputs to all text fields in the system. All text fields were tested with alpha-numeric, strictly alphabetical, negative, and invalid ranges of numbers for what can be represented. All inputs did not update the system and kept it in a safe state.

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

Refer to Test Plan to see corresponding Tests and SRS to see corresponding Requirements.

Test	Requirements
ID1	FR1
ID2	FR1, FR10
ID3	FR1, FR10
ID4	FR1, FR10
ID5	FR3, FR8
ID6	FR2
ID7	FR2
ID8	FR5, FR6
ID9	FR4, FR7
ID10	FR4, FR7
ID11	FR11, FR13
ID12	FR9
ID13	FR9
ID14	FR9
ID15	FR9
ID16	FR12
ID17	FR14
ID18	FR14, FR15
ID19	NFR1
ID20	NFR2
ID21	NFR3
ID22	NFR7
ID23	NFR9
ID24	NFR11

Table 2: Trace Between Test Cases and 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	M7, M9
ID10	M7, M9
ID11	M7, M9
ID12	M4, M6
ID13	M4, M6
ID14	M4, M6
ID15	M4, M6
ID16	M9
ID17	M7
ID18	M7
ID19	M5, M9
ID20	M7, M8
ID21	M7
ID22	M1, M2
ID23	M3
ID24	M7, M9

Table 3: Trace Between Test Cases and Modules

8 Code Coverage Metrics

Through an analysis of Trace to Requirements and Trace to Modules sections, the system defined and the system tested almost fully match. Excluding hard to test components of the system, edge coverage of system inputs was tested.

GIF manipulation was tested through auto manipulation of a GIF and comparisons to the expected output. This solution provided a guaranteed way to validate output with what a user should expect. However, due to the system depending on user interaction, most testing had to go through a manual test plan for system validation and verification. All possible input types are tested within the test plan, which verifies full edge coverage of user inputs to the system. However not all sequences could be tested manually due to the immense time it would take to complete this task. Thus the system is tested on an edge coverage basis, not a path coverage basis.

Finally, hard to test requirements such as non-functional requirement 3.2.1 is relative to specific test groups not available during this development time. Having a person of age MIN_USE_AGE (10) test the system was not present during the development phase, thus remains unverified. Furthermore, non-functional requirement 3.3.1 would require a testing of an extreme amount of GIF images to validate the speed requirement is met. Few other examples of hard to test requirements exist, thus passing the system as valid is a safe assumption.