

SE 3XA3: Test Plan Gifitti

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

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	4
2.5	Testing Schedule	4
3	System Test Description	4
3.1	Tests for Functional Requirements	4
3.1.1	Open GIF	4
3.1.2	Save GIF	6
3.1.3	GIF Start, Stop, Modify Length	7
3.1.4	Save All Frames in a GIF to Another Known Format	10
3.1.5	Help Context	10
3.1.6	Resize	11
3.2	Tests for Non-functional Requirements	12
3.2.1	Performance Requirements	12
4	Tests for Proof of Concept	14
4.1	Opening a GIF file for playback	15
4.2	Saving a GIF file's frames	15
5	Comparison to Existing Implementation	15
5.1	Graphics/UI	15
5.2	Performance	16
6	Unit Testing Plan	16
6.1	Unit testing of internal functions	16
6.2	Unit testing of output files	17
7	Bibliography	17

8	Appendix	18
8.1	Symbolic Parameters	18
8.2	Usability Survey Questions	18

List of Tables

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

List of Figures

Table 1: **Revision History**

Date	Version	Notes
October 22	1.0	Adding sections 1,2, and 7
October 24	1.1	Adding functional req tests
October 25	1.2	Adding non functional req tests.
October 30	1.3	Adding Unit tests, and automated functional tests
November 30	1.4	Final edits for revision 1

This document describes the test plan for the Gifitti application developed for 3XA3 at McMaster University.

1 General Information

1.1 Purpose

The purpose of this testing plan is to establish a set of tests that will test the product in its entirety to ensure that it fulfills the intended purpose. This will be accomplished by verifying that Gifitti satisfies the different functional and non-functional requirements that were assigned to it. Having test plans for any product is essential to be able to understand how well the product is satisfying the clients needs and if there is room for improvement.

1.2 Scope

This testing plan will utilize different testing methods such as automated and user created as well as various techniques such as black box and white box testing, to establish if the project has any need for improvement. Two different products will be analyzed through these tests, the Proof of Concept and the first iteration of the final product. The Proof of Concept will be tested to ensure that a basic representation of the product will be demonstrated. The first iteration of the final product will be tested to verify the requirements.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: Table of Abbreviations	
Abbreviation	Definition
GIF	Graphics Interchange Format
BMP	Bitmap
JPEG	Joint Photographic Experts Group
PNG	Portable network Graphics
TIFF	Tag Image File Format

Table 3: **Table of Definitions**

Term	Definition
GIF	A lossless format for image files that supports both animated and static images.
Sprite Sheet	A series of images (usually animation frames) combined into a larger image (or images).
Dialog Class	Specifies the base class used for displaying dialog boxes on the screen.
File Path	A path, the general form of the name of a file or directory, specifies a unique location in a file system.
Bitmap	A representation in which each item corresponds to one or more bits of information, especially the information used to control the display of a computer screen.
Frame	One of the many still images which compose a complete moving picture.
Pixel	A minute area of illumination on a display screen, one of many from which an image is composed.
TIFF	A common format for exchanging raster graphics (bitmap) images between application programs, including those used for scanner images.
JPEG	A commonly used method of lossless compression for digital images, particularly for those images produced by digital photography.
PNG	A raster graphics file format that supports lossless data compression.

1.4 Overview of Document

This testing plan is broken up into distinct parts. Under the heading Plan, the basic information will be given on the product and the tests. System Test Description will contain the specific tests for the functional requirements stated for the product. These tests will broke down into what type of tests they are and the results they achieve depending on their specific input. Testing for non-functional requirements will follow the same format where input will be given and the output will be measured for all of the provided non-functional requirements. Under tests for proof of concept, the same format will be utilized as the functional testing but it will not be testing the requirements for the project, but for the goals of the proof of concept. Furthermore there will be tests to compare Gifitti to the original product it was based on and unit testing plans to ensure correct output is achieved through proper internal functions.

2 Plan

2.1 Software Description

Gifitti is a software product that allows the common user to be able to manipulate GIFS for their entertainment or commercial needs. With an intuitive design, it allows a person who has never done any kind of graphical editing to be able to manipulate a GIF to achieve the person's particular design. This product can also serve a commercial purpose in allowing the quick production of GIFs.

2.2 Test Team

The team to implement the test plan for the project will be Pavle Arezina, Riley McGee, Nicolai Kozel.

2.3 Automated Testing Approach

This test plan will utilize automated testing for verification that the GIF manipulation functionality changes the GIF properly. That is to say that the exported GIF image will match what is created by the user of our product.

2.4 Testing Tools

The only tool to be utilized to test this product will be the Microsoft Unit Test Framework that is native with Visual Studio.

2.5 Testing Schedule

The testing schedule is listed on the Gantt Chart which can be found [here](#).

3 System Test Description

Testing allows the developers to detect errors that the test cases cover. With the functional and non-functional requirements the developer knows the expected results and can evaluate the results of tests accordingly. Pass or fail values are assigned to executed tests discreetly unless the test is to have an error tolerance for passing.

3.1 Tests for Functional Requirements

Functional requirements prescribe what services the software should provide. They capture the intended software effects on the environment and applicability conditions. These tests ensure that the functional requirements of Gifitti are fulfilled or the tests will help to discover any functional requirements not satisfied by Gifitti.

3.1.1 Open GIF

The User is able to open a GIF from a specified location

1. Select Properly Formatted GIF- id1

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: File path for GIF. Output: System loads gif into memory, displays it to the user in the gif view.

How test will be performed:

- (a) Launch the program
- (b) Select Open

- (c) From the Open dialog specify a path to a known gif image
- (d) After the image is loaded verify that it is being displayed, and resides in system memory

2. No File Selected in File Dialog-id2

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: None. Output: No image loaded.

How test will be performed:

- (a) Launch the program
- (b) Select Open
- (c) Select open option with no file path specified
- (d) Verify program remains open, and no image is loaded

3. Close File Dialog-id3

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: None. Output: No image loaded.

How test will be performed:

- (a) Launch the program
- (b) Select Open
- (c) Close the file dialog
- (d) Verify program remains open, and no image is loaded

4. Open Random Non-GIF File-id4

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: File that is not a GIF. Output: No image loaded.

How test will be performed:

- (a) Launch the program

- (b) Select Open
- (c) Try and select a file that is not a GIF or specify a file path to a known file
- (d) Verify program remains open, no image is loaded, and an error message is displayed to the user

3.1.2 Save GIF

The user is able to save a GIF to a specified location

1. Save a GIF to a Known Location-id5

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: File path, GIF file.

Output: GIF file saved to specified location.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Select "Save As..."
- (d) Specify a known system file path and a saved image name
- (e) Verify the loaded and saved GIFs are identical

2. Save GIF to a Non-existent Location- id6

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: File path, GIF file.

Output: GIF file not saved to specified location.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Select "Save As..."
- (d) Specify a system file path known not to exist and a saved image name
- (e) Verify the user is informed that the file path does not work and the system remains running

3. Save GIF to Opened Location- id7

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: File path, GIF file.

Output: GIF file saved to specified location.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Modify the GIF
- (d) Select the save image option
- (e) Verify new GIF is saved over the originally opened GIF

3.1.3 GIF Start, Stop, Modify Length

The user is able to start and stop GIFs as well as shorten the GIF and change the playback speed.

1. Play Stop GIF-id8

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: GIF

Output: None.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Select Play Option
- (d) Verify the GIF is iterating over frames as expected
- (e) Select Play option again, verify no change occurs
- (f) Select Stop
- (g) Verify GIF stops playing
- (h) Select Stop again, verify no change occurs

2. GIF shorten-id9

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: GIF

Output: Shortened GIF.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Shorten gif frames in use
- (d) Play the GIF to verify it is correct
- (e) Export the GIF
- (f) Verify that the exported GIF represents the one playing, and is shorter than the original

3. Automated GIF Shortening-id10

Type: Automated Functional.

Initial State: Image reading module driver

Input: GIF; Expected output GIF;

Output: Expected output GIF

How test will be performed:

- (a) Test will load GIF from a known location
- (b) Test will subset the GIF to a known frame set
- (c) The GIF will then be compared to an expected GIF in a frame-by-frame manner
- (d) Each frame is to be cast to a bitmap then compared in a pixel-by-pixel manner
- (e) Pass if 100 percent of all pixels match

4. GIF speed-id11

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: GIF

Output: GIF inputted with different frame rate.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Increase frame hold time
- (d) Play the GIF to verify it is slower than original
- (e) Export the GIF
- (f) Verify that the exported GIF represents the one playing, and is slower, with respect to frame rate, than the original

3.1.4 Save All Frames in a GIF to Another Known Format

The user is able to save a GIF frame by frame to PNG, JPEG, BMP and TIFF formats

1. Save PNG Frames to Known Location-id12

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: Folder path, GIF file.

Output: PNG Frames in specified folder named corresponding to frame.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Choose to export the image as a PNG set
- (d) Select the folder where the PNGs will be dumped
- (e) Verify the folder fills with GIF frames as PNGs
- (f) Ensure a 3rd party program can load the PNGs. Software such as Photoshop or MS paint is satisfactory

2. Save JPEG frames to known location-id13

This test is identical to id1 with PNG replaced with JPEG

3. Save BMP frames to known location-id14

This test is identical to id1 with PNG replaced with BMP

4. Save TIFF frames to known location-id15

This test is identical to id1 with PNG replaced with TIFF

3.1.5 Help Context

Verifies and validates the Help context

1. Help Context-id16

Type: Manual Functional.
Initial State: Program loaded
Input: None
Output: None

How test will be performed:

- (a) Launch the program
- (b) Select Help Context
- (c) Goes through all Help Context options, verify spelling for professionalism
- (d) Close help context, verify program remains in same state as launch put it in

3.1.6 Resize

Verifies a GIF can be exported and resized

1. resizing-id17

Type: Manual Functional.
Initial State: Program loaded
Input: GIF, Width, Height
Output: Resized GIF

How test will be performed:

- (a) Launch the program
- (b) Load a GIF
- (c) Click Resize
- (d) Enter a width and height value
- (e) Choose save as and specify a file path.
- (f) Verify the GIF was exported successfully and resized accordingly.

2. resizing-id18

Type: Manual Functional.

Initial State: Program loaded

Input: GIF, Width, Height

Output: Resized GIF

How test will be performed:

- (a) Launch the program
- (b) Load a GIF
- (c) Click Resize
- (d) Enter boundary cases for width and height values.
- (e) Verify these boundary cases do not cause the program to crash.

3.2 Tests for Non-functional Requirements

Non-functional requirements constrain how the functional requirements and services should be provided. They are related to the quality of the product, such as the safety of utilizing the software or how accurate it is. These tests ensure that the non-functional requirements of Gifitti are fulfilled.

3.2.1 Performance Requirements

Speed

1. speed-id19

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF with number of frames on the magnitude of 100's of frames.

Output: Subsection of frames chosen to be exported.

How test will be performed: Create a timer inside the program to output the time it takes to output the files, to a debug.txt file. The time should be less than MAX_EXPORT_TIME.

2. speed-id20

Type: Manual Structural

Initial State: Program loaded;

Input: GIF.

Output: NA

How test will be performed: Load a GIF into the program. The time to load the GIF and start playback should be less than MAX_UI_LOAD. MAX_UI_LOAD is derived from the original system, GIF Viewer, so make sure this test is performed on the same system as the benchmark MAX_UI_LOAD is acquired from.

Precision

1. precision-id21

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF

Output: Subsection of frames chosen to be exported.

How test will be performed: Export the first 5 frames of the GIF. Navigate to the save location and verify that the number of frames saved is equal to 5.

Safety Critical

1. safety-id22

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF

Output: NA

How test will be performed: Export a subsection of frames from the GIF to a folder. Try to export to the same folder again. The program should ask if you are okay with overwriting the existing files.

2. safety-id23

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF

Output: NA

How test will be performed: Create a small partition of the HDD and fill it so there is no available space. Try to export frames to this location. The program should not continue the operation and inform the user that disk space needs to be cleared before continuing.

3. robustness-id24

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF, Frame Range

Output: Selected frames if the input is a valid range.

How test will be performed: Attempt to enter invalid input to the frame selection text boxes and record the result. Program should function normally (and as expected) for all entries. This includes negative numbers, letters or other characters, and invalid ranges.

4 Tests for Proof of Concept

The proof of concept is a demonstration of the feasibility of the software project. Ensuring that the Proof on Concept is producing the intended results can only be achieved through tests specifically designed to root out any of the errors present in the build of the proof of concept. By successfully testing the proof of concept and achieving the intended goals, it is shown that the software-to-be is a possible entity.

4.1 Opening a GIF file for playback

Open GIF

1. OpenGif-01

Type: Manual Functional

Initial State: Program must be in normal state (form window is open and playback window is blank).

Input: GIF File

Output: GIF is shown in playback window of the form.

How test will be performed: Click open button, select a file of type .gif, and verify that the GIF loads and begins to playback within the form window.

4.2 Saving a GIF file's frames

Save GIF

1. SaveGif-01

Type: Manual Functional

Initial State: Program must be in playback state (form window is open and playback window is playing GIF).

Input: GIF File

Output: GIF's frames are saved as .bmp in folder specified.

How test will be performed: Click save frames button, select a folder, and verify that the GIF's frames are saved to the specified folder as .bmp.

5 Comparison to Existing Implementation

5.1 Graphics/UI

1. GIF playback resolution is the same or better than Gif Viewer.

2. Program has the same button scheme as Gif Viewer (Open button to open file, Extract Frames button to save frames).
3. Program has a help menu available that is similar to Gif Viewer, but is available at all times.
4. Program's color scheme and design resembles Gif Viewer.

5.2 Performance

1. GIF playback is at the same smoothness/framerate or better than Gif Viewer.
2. Opening and saving a file takes the same amount of time or less than Gif Viewer.

6 Unit Testing Plan

The unit testing plan is the devised set of tests that the code and the system modules must undergo regardless of the system integration level. Unit testing allows the project to have white box testing on the individual system components where each unit test can be executed via testing frameworks in C Sharp.

The testing framework being utilized is the native Visual Studio unit testing framework: `Microsoft.VisualStudio.TestTools.UnitTesting`.

6.1 Unit testing of internal functions

Unit testing will be 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 files that present the same). A method with no expected error tolerance should be implemented as discreetly pass XOR fail. All unit test inputs must fall in a known domain where the expected output can be assumed. No stubs will be required for unit testing of functions. Image reading is needed as a driver for any method using GIFs or other image types. Any written unit test must utilize 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. We plan to have unit testing cover approximately 50 percent of our major methods due to project time constraints.

6.2 Unit testing of output files

To test the validity of outputted GIF images, the testing system will hold image resources of input frames, input GIFs, as well as the expected GIF outputted. A known GIF should be 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 should be cast to a known image format such as a bitmap, then compared pixel-by-pixel. Frames should have a 100 percent pixel match for the expected GIF to pass. A method for casting Frames to bitmaps and then comparing the pixels, should be added to the testing package prior to testing GIF manipulation. This function can then be verified by simply sending the same image(s) as parameters.

7 Bibliography

Microsoft Visual Studio Unit Testing Framework, Microsoft Incorporated;
<https://msdn.microsoft.com/en-us/library/dd264975.aspx>

Questionnaire Template, Chin, J.P., Diehl, V.A., Norman, K.L.;
<http://garyperlman.com/quest/quest.cgi?form=QUIS>

8 Appendix

This section contains symbolic parameters for this document and the usability survey that will be delivered to a focus group upon initial completion of the application.

8.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

1. MAX_ULLOAD: 3 Seconds
2. MAX_EXPORT_TIME: 10 Seconds

8.2 Usability Survey Questions

The survey will be delivered in the same format as the questionnaire for User Interface Satisfaction. This questionnaire is composed of various questions pertaining to several sub categories on a 0-9 scale. This includes the screen, terminology and system information, learning, and system capabilities. It also allows the user to list the most positive and negative aspects of the program. The questionnaire can be found at garyperلمان.com