TABLE OF CONTENTS

TEST SPECIFICATION

Group members: Aimal Khan (0946636), Sean McLellan (0963077)

1.0 Introduction

1.1 Goals and objectives

Ascii Generator 2 is a tool used to convert image files of various formats into high-resolution ascii art. The purpose of the tool is to save users the time it takes to carefully plan out complicated ascii art for display on text only message boards, and forums. The program supports conversions using variable width fonts, real-time output adjustment, output as black and white or color to text, XHTML, RTF, or as an image, as well as support for batch conversions.

The testing process for the Ascii Generator 2 has a specific set of goals. The software is going to be tested for bugs, errors in logic, reasonable performance across different machines, ease of use for a user who is not a power user, as well as testing for an aesthetic and functional look-and-feel closely resembling that of a typical Windows program.

1.2 Statement of scope

Ascii Generator 2 will be tested using white-box unit testing methods to ensure that the functions behave appropriately and give the expected results. This program will also be tested using integration testing methods, after the initial unit tests have been concluded, to make sure that when combined, the modules' expected behaviours are maintained. This form of integration testing will be done using a black-box bottom-up approach.

Validation testing will occur to make sure that the system meets users' expectations for the functionality the system is supposed to provide. During validation testing, design principles will also be examined, including the usability and ease of navigation regarding the interface, in addition to the successful and reliable import and export of data into a variety of expected/specified file types.

The program's code should produce bug free results, and data that is read in from different image files should convert to the expected ascii form. This data should then be easily exported into a specified/fixed set of data or image file types. Additionally, any valid data manipulations should not corrupt the data, and reversibility should exist to prevent permanent alteration of data before export.

Testing will not be conducted for printing and printing related functions, this is due to that functionality not being a necessity. If the program can successfully convert the data into a valid image file or text file, that file can be printed using a variety of other software available by default such as windows image viewer, etc.

1.3 Major constraints

There is no drop-dead date as this program has already released version 2.0.0 as of 2011-09-26.

One known technical constraint is that the OS required to run this program is Windows XP or higher.

The tests conducted for this program are limited to the specifications of 3 machines:

Machine 1
CPU: Intel Core i5-2500K @ 3.3GHz
Graphics Card: AMD 6970 HD 2GB
OS: Windows 7 Professional 64-bit
Memory: 8GB DDR3 @ 1600MHz
HDD Capacity: 250GB

Machine 2
CPU: Intel Core i7 920 @ 2.67GHz
Graphics Card: NVIDIA GeForce GTX 660 Ti 2GB
OS: Windows 7 Professional 64-bit
Memory: 16GB DDR3 @ 1600MHz
HDD Capacity: 500GB

Machine 3
CPU: Intel Core i3-350M @ 2.26GHz
Graphics Card: NVIDIA GeForce GT330M 1GB
OS: Windows 8 Professional 64-bit
Memory: 8GB DDR3 @ 1333MHz
HDD Capacity: 320GB

2.0 Test Plan

2.1 Software (SCIs) to be tested

User Interface Components to be Tested

- Zoomable Picture Thumbnail
- Option Controls (Levels/Brightness/Contrast/Dither Tabs)
- Visibility Controls for View Settings
- Top Panel (Character, Ramp, Size)

Backend Components to be Tested

- Load Image
- Batch Conversion
- Save Image
- Font Handler
- Character Handler
- Output Size Handler
- Levels/Brightness/Contrast/Dither Handler

Help Components to be Tested

- Tutorial
- FAQ Search

Installation Components to be Tested

- Software Install/Run Process
- Software Uninstall/Removal Process

2.2 Testing strategy

2.2.1 Unit testing

Unit testing will be conducted using the white box method. Testing will be done most frequently using basis path testing, branch coverage testing, condition testing and range testing due to the fact that most of these tests are simple to conduct as well as effective in detecting errors in logic and output. These simple tests are useful for testing this program because the program's purpose is very specific and does not span a large variety of complex functions (such as those with many iterative loops).

The components that will be tested using our unit test specifications include:

User Interface Components to be Unit Tested

- Zoomable Picture Thumbnail
- Option Controls (Levels/Brightness/Contrast/Dither Tabs)
- Visibility Controls for View Settings
- Top Panel (Character, Ramp, Size)

Backend Components to be Unit Tested

- Load Image Handler
- Batch Conversion Handler
- Save Image Handler
- Font Handler
- Character Handler
- Output Size Handler
- Levels/Brightness/Contrast/Dither Handler

2.2.2 Integration testing

Integration testing will occur in a bottom-up fashion, building the test cases off the initial unit test cases done for each component. Not all components listed in the unit testing section can necessarily be tested together as they may control functionality that is separate (unrelated) from the other components of the program. Testing in this step will be done incrementally to keep track of any bugs that may appear at different levels of integration, and all tests at this stage will be conducted using the black box method of testing.

The order of integration by software function is as follows:

1. Load Image Handler
2. Character Handler
3. Font Handler
4. Levels/Brightness/Contrast/Dither Handler

5 Output Size Handler
6 Save Image Handler

2.2.3 Validation testing

User interface testing will be done as a whole, as all elements do work well together and are easy to debug if there are any problems. Connections between the various UI elements will be recorded. UI-specific testing will be done in a black-box fashion, as the backend will be done separately.

Back end testing will be performed to link back to the interface to ensure all changes on the UI are actually being performed in the code through debug points. A reasonable level of performance is expected, as well as understandable, documented code and ease of usability.

Ascii Generator 2 does not have available documentation relating to the System Requirements Specification, therefore this testing specification will use its own reasonable criteria for quality testing and usability/ease of use testing for the interface and other user purposes.

2.2.4 High-order testing

Ascii Generator 2 is not being handled by the team undertaking this test specification and therefore has no access to any metrics that may be obtained over an alpha or beta stage for any future implementations/versions. Users of this open source project are able to submit feedback via the *request a feature* and *report a bug* tools in the program for future consideration and improvements.

2.3 Testing resources and staffing

Resources

No special resources are required for testing beyond the machines mentioned in the major constraints section.

Staffing

- Test Team Leader - Aimal Khan/Sean McLellan
- Unit Testing/Integration Testing/Validation Testing Coordinator - Aimal Khan/Sean McLellan

2.4 Test work products

No additional software was developed for the process of testing the system in the specification.

2.5 Test record keeping

Microsoft Excel will be used to record and evaluate the test results of various unit cases, integration tests, and validation tests. A log will be kept using a Google Document file, which will keep track of any bugs that occur, noting what the bug was, what section of testing this bug was encountered, what conditions caused the bug, and the date it was found. The document will also contain notes relating to the user experience during validation testing, highlighting what the user liked about the program, and what issues the user had with the program or its interface. The program contains the ability to report bugs or request features right within the program, so any bugs that are found will be reported using this tool so that the developer can track any bugs found.

2.6 Test metrics

Layout Appropriateness: a function of layout entities, the geographic position and the “cost” of making transitions among physical layout entities.

Cyclomatic Complexity: the count of the number of linearly independent paths through the source code (or function/method). If a function has a relatively large value for this metric, the function may require a larger amount of unit test cases due to the higher possibility of errors in logic.

Error Templates: this will contain the type of error, the conditions to cause the error, and the priority of the error to be corrected.

2.7 Testing tools and environment

Microsoft Visual C# 2010 is used as the testing environment and testing tool. There will be a fixed set of images that will be used to test the functionality of the program across each machine. These images will not change between machines in order to keep consistency across platforms. No other special tools are required to test this system.

2.8 Test schedule

Assuming a schedule where the results of the testing are due on the 1st of April, the tentative schedule is as follows:

Unit Testing: March 18th - 22nd
Integration Testing: March 22nd - 26th
Validation Testing: March 26th - 28th
Performance Testing: March 28th - 30th
Testing Summary: April 1st

3.0 Test Procedure

3.1 Software (SCIs) to be tested

User Interface Components to be Tested

- Zoomable Picture Thumbnail
- Option Controls (Levels/Brightness/Contrast/Dither Tabs)
- Visibility Controls for View Settings
- Top Panel (Character, Ramp, Size)

Backend Components to be Tested

- Load Image
- Batch Conversion
- Save Image
- Font Handler
- Character Handler
- Output Size Handler
- Levels/Brightness/Contrast/Dither Handler

Help Components to be Tested

- Tutorial
- FAQ Search

Installation Components to be Tested

- Software Install/Run Process
- Software Uninstall/Removal Process

3.2 Testing procedure

3.2.1 Unit test cases

Component: *Zoomable Picture Thumbnail*

3.2.1.1 Testing Procedure for Component

The Picture Thumbnail will be tested in a black box fashion. After the test image has been loaded, all available functions within the frame will be tested (such as select, drag, scale, zoom).

3.2.1.2 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.3 Test Cases for Component

- Test image will be loaded into the application via double click and should display properly in the thumbnail frame.
- Test image will be loaded into the application via drag & drop and should display properly in the thumbnail frame.
- The frame will have a selection (done with the mouse) performed on it.
- The frame will have a drag & zoom (done with the mouse) performed on it.
- The frame will have an image pan (done with the mouse) performed on it.

3.2.1.4 Purpose of Tests for Component

The purpose of these tests is to ensure correct operation of all controls and mouse operations in the Picture Thumbnail frame, as well as to verify proper execution of the rest of this application, as this frame is effectively the “input panel” for the main ASCII window.

3.2.1.5 Expected results for Component

The Picture Thumbnail is expected to allow the user full control of the input image’s parameters, output results to the ASCII screen, as well as work within its design specifications.

Component: *Option Controls*

3.2.1.6 Testing Procedure for Component

The Option Controls will be tested in a black box fashion. The level, Brightness/Contrast and Dither tools will all be tested individually to ensure their output works as it should.

3.2.1.7 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.8 Test Cases for Component

- The black, white and middle level settings will be incremented and decremented to test for changes in output and to determine proper functionality.
- The Brightness/Contrast sliders will be incremented and decremented to test for changes in output and to determine proper functionality.
- The dither slider will be incremented and decremented to test for changes in output and to determine proper functionality.

3.2.1.9 Purpose of Tests for Component

The purpose of the tests is to ensure that the output image remains visible after being edited and that there are no errors with any of the controls.

3.2.1.10 Expected results for Component

This component is expected to edit the image content by replacing certain characters with blank space or smaller characters to portray brightness, contrast and the other affiliated changes above.

Component: *Visibility Controls*

3.2.1.11 Testing Procedure for Component

The visibility controls will be tested in a black box fashion. The visibility controls will hide and appear each individual additional view available (text-settings and input image) and ensure that they remain functional after being revealed.

3.2.1.12 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.13 Test Cases for Component

- The button that controls the text-setting view will be toggled off and then on, and tested for functionality.

- The button that controls the input image view will be toggled off and then on, and tested for functionality.

3.2.1.14 Purpose of Tests for Component

The purpose of the tests is to ensure that the views remain functional after being hidden and that there are no graphical errors upon gaining visibility.

3.2.1.15 Expected results for Component

This component is expected to continue to hide and show both windows (depending on which is toggled) while not affecting their separate functionalities.

Component: *Top Panel (Character, Ramp, Size)*

3.2.1.16 Testing Procedure for Component

The top panel which controls characters, ramp, and size of the image, will be tested in a white box fashion. Testing will be done by inputting various acceptable values into the fields and validating the output with the on-screen display.

3.2.1.17 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.18 Test Cases for Component

- An image will be loaded into the program, then manipulations to the value of the size field will be done and output on screen checked.
- An image will be loaded into the program, then manipulations to the value of the ramp field will be done and output on screen checked.
- An image will be loaded into the program, then manipulations to the value of the characters field will be done and output on screen checked.

3.2.1.19 Purpose of Tests for Component

The purpose of the tests is to ensure that the each field correctly controls and manipulates the appropriate property, while protecting against potential invalid inputs.

3.2.1.20 Expected results for Component

The component is expected to appropriately alter the ascii image based on the alteration of values in either the size, ramp or characters field.

Component: *Load Image Handler*

3.2.1.21 Testing Procedure for Component

The load image handler will be tested in a white box fashion. A designated image will be loaded into the ascii generator and examined for completeness.

3.2.1.22 Stubs and/or drivers for Component

The required C# code is used as a test bed.

3.2.1.23 Test Cases for Component

- Various image file types will be loaded into the generator.
- Various images of a different range of sizes will be loaded into the generator.
- A file type which is not an image type will be loaded into the generator.

3.2.1.24 Purpose of Tests for Component

The purpose of the tests are to ensure that file types which are not valid are rejected safely, and that images which are accepted are correctly parsed and loaded into the generator.

3.2.1.25 Expected results for Component

The load image handler is expected to correctly parse all the images within an acceptable period of time.

Component: *Batch Conversion*

3.2.1.26 Testing Procedure for Component

The batch conversion tool will be tested in a white box fashion. A set of images will be loaded into the ascii generator and have the appropriate properties changed in each ascii image and output to the designated file type and directory, being tested for completeness, and file integrity (no corruption).

3.2.1.27 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.28 Test Cases for Component

- A set of designated files will be added to the list of images to be batch processed individually and a variety of changes to the output properties will be applied.

- A directory containing the set of designated files will be added to the list of images to be batch processed and a variety of changes to the output properties will be applied..

3.2.1.29 Purpose of Tests for Component

The purpose of the tests are to ensure that each file is being correctly parsed, altered and that the output is complete and correct using either specific file additions or directory additions.

3.2.1.30 Expected results for Component

This component is expected to successfully alter a directory of images, or a list of images and output them to the designated folder with the correct file type extension in an acceptable amount of time.

Component: *Save Image Handler*

3.2.1.31 Testing Procedure for Component

The save image handler will be tested in a white box fashion. A designated image will be loaded into the ascii generator and then saved as any of the acceptable file types which will then be examined for completeness.

3.2.1.32 Stubs and/or drivers for Component

The required C# code is used as a test bed.

3.2.1.33 Test Cases for Component

- The ascii generated image will be saved as each of the acceptable file types.

3.2.1.34 Purpose of Tests for Component

The purpose of the test is to ensure that the save image handler correctly saves the image on the ascii screen into the designated file type, as well as not corrupting the image or text.

3.2.1.35 Expected results for Component

This component is expected to save the file into the correct file type and produce a file which can be opened by a standard image viewer or text editor.

Component: *Font Handler*

3.2.1.36 Testing Procedure for Component

The font handler will be tested in a white box fashion. A designated image will be loaded into the ascii generator and then the font/font style will be changed, determining whether the appropriate font manipulations are being applied to the ASCII image.

3.2.1.37 Stubs and/or drivers for Component

The required C# code is used as a test bed.

3.2.1.38 Test Cases for Component

- A variety of font sizes, font styles and font types will be tested.

3.2.1.39 Purpose of Tests for Component

The purpose of the test is to determine that the correct application of each font size, style and type is reflected in the ASCII output.

3.2.1.40 Expected results for Component

This component is expected to change the font of the output based on the selected properties in an acceptable period of time.

Component: *Character Handler*

3.2.1.41 Testing Procedure for Component

The character handler will be tested in a white box fashion. A set of characters will be loaded into the ASCII generator (preset) and then the character set will be changed, determining whether the appropriate character manipulations are being applied to the ASCII image.

3.2.1.42 Stubs and/or drivers for Component

The required C# code and installed system fonts are needed for this component to work.

3.2.1.43 Test Cases for Component

- A variety of character presets, including added and removed characters, will be tested (by removing/adding characters from the character and ramp lists to see effects on the image)

3.2.1.44 Purpose of Tests for Component

The purpose of this test is to determine that the correct number and type of characters is present in the ASCII image output.

3.2.1.45 Expected results for Component

The component is expected to change the numbers/types of characters in the output based on the selected character set in a reasonable timeframe.

Component: *Output Size Handler*

3.2.1.46 Testing Procedure for Component

The Output Size Handler will be tested in a black box fashion. The size (X * Y) will be modified to match the test image's resolution, and then will be modified to several different iterations.

3.2.1.47 Stubs and/or drivers for Component

This component uses the resolution of the original image as a baseline for how large to scale the ASCII image.

3.2.1.48 Test Cases for Component

- The X parameter will be tested to see if the Y parameter matches up.
- The Y parameter will be tested to see if the X parameter matches up.
- Different variations of the original image's size will be used to see if the effects on the ASCII image are reasonable.

3.2.1.49 Purpose of Tests for Component

The purpose of these tests are to ensure that the X and Y parameters remain matched up to each other (preventing the image from being skewed or stretched), and to keep the image on screen.

3.2.1.50 Expected results for Component

This component is expected to correctly display the image (and continue to do so) after each resizing of the image using the Output Size Handler.

Component: *Levels/Brightness/Contrast/Dither*

3.2.1.51 Testing Procedure for Component

The Levels/Brightness/Contrast/Dither tools will be tested in a white box fashion. An image will be loaded into the Ascii Generator and various manipulations will be applied to the ascii generated image, testing the functionality of each effect, the mathematical correctness and ensuring that the appropriate changes are applied to the generated image.

3.2.1.52 Stubs and/or drivers for Component

The C# code is the test bed for these functions.

3.2.1.53 Test Cases for Component

- An image will be loaded and the *levels* slider will be moved to various positions on the range of values.
- An image will be loaded and the *brightness* slider will be moved to various positions on the range of values.
- An image will be loaded and the *contrast* slider will be moved to various positions on the range of values.
- An image will be loaded and the *dither* slider will be moved to various positions on the range of values.

3.2.1.54 Purpose of Tests for Component

The purpose of the tests is to ensure that each image manipulation function applies the appropriate effect to the picture (correctness).

3.2.1.55 Expected results for Component

This component is expected to correctly translate a value on a range of values into the appropriate intensity for the levels adjustment, brightness, contrast, or dither functions, and apply the corresponding effect to the ascii image.

Component: *Tutorial*

3.2.1.56 Testing Procedure for Component

The tutorials will be tested in a white box fashion. The tutorials will be tested for ease of use and completeness as well as comprehensibility.

3.2.1.57 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.58 Test Cases for Component

- The tutorial will be followed step by step to reach the desired result explained in the tutorial title.

3.2.1.59 Purpose of Tests for Component

The purpose of this test is to ensure that each tutorial located in the tutorial section of this program is easily understandable, correct and complete.

3.2.1.60 Expected results for Component

The tutorial component is expected to successful transfer the desired knowledge from the software to the user.

Component: *FAQ Search*

3.2.1.61 Testing Procedure for Component

The FAQ Search component will be tested in a white box fashion. The FAQ Search component will be tested for completeness, accuracy of search, correct answers, and response to feedback.

3.2.1.62 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.63 Test Cases for Component

- FAQ Search will search for an answer to a variety of different problems that are related to the software.
- FAQ Search will search for an answer to a problem that is unrelated to the software.
- FAQ Search will search for an answer to a random set of characters.

3.2.1.64 Purpose of Tests for Component

The purpose of this test is to ensure that the FAQ Search function is robust and functional while also providing correct and understandable answers in a timely manner.

3.2.1.65 Expected results for Component

The FAQ Search is expected to return the best possible answer to a question that is related to the program, or return a message to inform the user that the question could not be answered.

Component: *Software Install/Uninstall Process*

3.2.1.66 Testing Procedure for Component

The software installation process will be tested in a black box fashion. The installation and extraction of all files will be tested to verify that all the appropriate files have been installed and that the program can successful run without encountering unexpected crashes due to missing files or corrupted files.

The software uninstallation will be tested in a black box fashion. The uninstallation and deletion of all files will be tested to verify that the appropriate files were removed.

3.2.1.67 Stubs and/or drivers for Component

There are no stubs or drivers for this component.

3.2.1.68 Test Cases for Component

- The software will be installed by unzipping a .rar file containing the .exe.
- The user will need to run the .exe file to run the software without crashing to desktop.
- The software will be uninstalled by deleting the extracted contents of the zipped folder.

3.2.1.69 Purpose of Tests for Component

The purpose of this test is to ensure that the program can execute appropriately and that no runtime errors occur upon initialization, and that the uninstallation of this program does not result in registry errors or is unable to be removed.

3.2.1.70 Expected results for Component

It is expected that the program starts up without encountering an unexpected error, and that all files are not corrupted after unzip. It is also expected that the program will be easily deleted and sent to the recycling bin upon deletion.

3.2.2 Integration testing

3.2.2.1 Testing procedure for integration

The system will be integrated incrementally; this will be done to control the amount of bugs that appear after each module is integrated into the framework of the full system, allowing bugs to be traced back to the trigger. The order of integration by software function is as follows:

1. Load Image Handler
2. Character Handler

3 Font Handler
4 Levels/Brightness/Contrast/Dither Handler
5 Output Size Handler
6 Save Image Handler

These components are tested using unit test cases, and are then integrated for functionality together, using a bottom-up approach to isolate and identify bugs and errors which use a black box method of testing.

3.2.2.2 Stubs and drivers required

No stubs or drivers are required for integrated testing besides the actual C# code of the program.

3.2.2.3 Test cases and their purpose

- Each component listed above contains functionality that can be tested in separate unit tests, however the purpose of the integrated testing is to ensure that each component can perform its task within the environment at the same time as other manipulations are applied using other functions. The order of testing is specific to the typical order these functions would be called upon during regular use of the ascii generator.

3.2.2.4 Expected results

The system is expected to integrate without major flaws, bugs, or errors.

3.2.3 Validation testing

3.2.3.1 Testing procedure for validation

The features and functionality of the final system will be compared to the expected functionality from the description of the program and user feedback for usability testing due to a lack of publicly available software requirements specification documents for this system.

3.2.3.2 Test Cases and Their Purpose

- The features of the system will be compared to the expected functionality of the program based on the program description.
- Usability will be determined based on the user feedback gathered from tester experiences.

3.2.3.3 Expected results

It is expected that the program will have positive user feedback and perform all tasks that the description claims it can do.

3.2.3.4 Pass/fail criterion for all validation tests

Since there are no publicly available software requirements specification documents, the pass/fail criterion will be based on tester feedback and functionally compared to the software description on Source Forge.

3.2.4 High-order testing

3.2.4.1 Recovery testing

No recovery testing will be performed for this system as unexpected termination of the program is acceptable although undesirable.

3.2.4.2 Security testing

There are no security tests being performed for this system.

3.2.4.3 Stress testing

The Ascii Generator will be forced to batch process an unusually large number of images with a specified file size (< 2Mb) in order to determine how many images can be processed at once.

3.2.4.4 Performance testing

The time between images being processed and loaded will be compared, based on varying file sizes in order to determine the efficiency of the program.

3.2.4.5 Alpha/beta testing

Alpha/Beta testing will not be conducted as the program is already in a stable public release build; however improvements are constantly being made to the program via the *request a feature* tool and *report a bug* tool.

3.2.4.6 Pass/fail criterion for all validation tests

Since there are no existing design document requirements publicly available, the only pass/fail criterion for validation will come from the usability, expected functionality and ease of use tests for the Ascii Generator.

3.3 Testing resources and staffing

Resources

No special resources are required for testing beyond the machines mentioned in the major constraints section.

Staffing

- Test Team Leader - Aimal Khan/Sean McLellan
- Unit Testing/Integration Testing/Validation Testing Coordinator - Aimal Khan/Sean McLellan

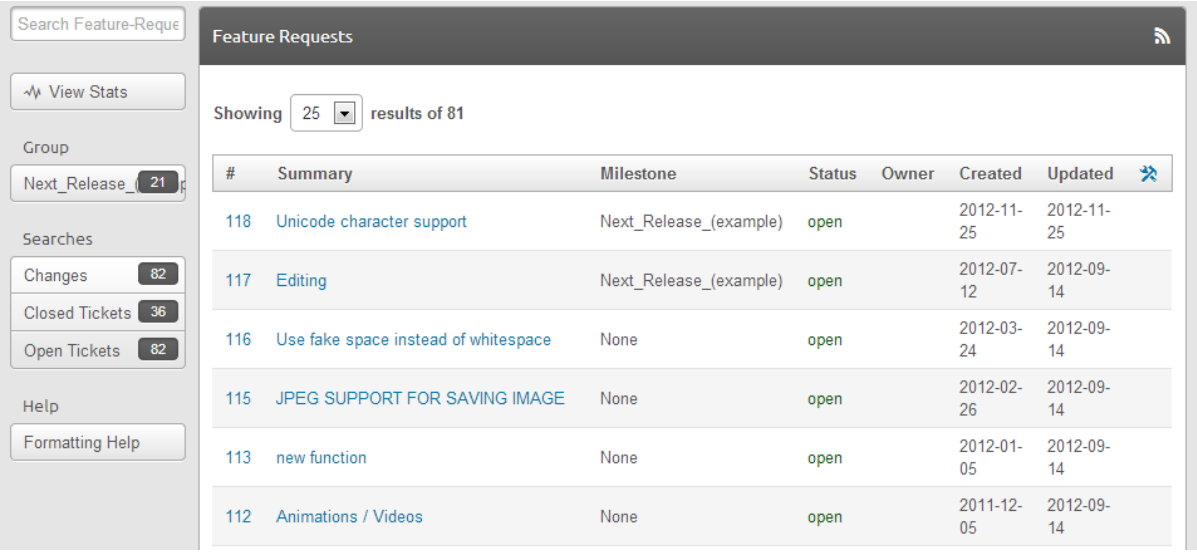
3.4 Test work products

No additional software was developed for the process of testing the system in the specification.

3.5 Test record keeping and test log

Microsoft Excel will be used to record and evaluate the test results of various unit cases, integration tests, and validation tests. A log will be kept using a Google Document file, which will keep track of any bugs that occur, noting what the bug was, what section of testing this bug was encountered, what conditions caused the bug, and the date it was found. The document will also contain notes relating to the user experience during validation testing, highlighting what the user liked about the program, and what issues the user had with the program or its interface. The program contains the ability to report bugs or request features right within the program, so any bugs that are found will be reported using this tool so that the developer can track any bugs found.

Example of feature request tool:



Example of report bug tool: