Matthew Espinosa

**Word Count Software Design Report**

Project Beginning:

This project began as project to implement software engineering techniques to code production and deployment. For my project, I needed to create a program that could take in a data file which contained a passage of text of varying sizes and contents, and appropriately display what words exist inside the document. There were different types of statistics that needed to be conveyed to the user, originally the project required that the program could display how many words were in the file, how many unique words were in the file, and handle errors that could occur in the process.

To tackle this problem, I knew that I would need to utilize a data structure that could not only hold how many of each word was present in the file, but I also needed to have it be able to quickly iterate through all elements for later processing. Therefore, I decided to go with a linked list structure, it was not only able to add on new words as the text file is processed, but it could be iterated through for a quick check to see if the word already existed or was a unique word. I ended up reusing an old linked list library that I wrote for my CS1713 class a long time ago, and found it easy to adapt it to holding information about individual words. With this I was able to satisfy all the requirements and created a program that was able to display the statistical information appropriately.

First Requirement Update:

Once I was given the new requirements of needing to count all the lines inside of the file and checking how many individual characters are in the words of the file, I was happy to say the least. All I needed to do to accomplish the first part was to simply declare a variable that iterated every time a line was read in, and even better, to find the characters in the file all I needed to do was check through my linked list from before. Now all I needed to do was add another variable that could hold character count, and have it tally through the words with an strlen() of the word being added at each node. Even though I could have just done all these things in one function, I decided it would be best to create individual functions that could address each problem and call them as through my main program. I felt this would be the best approach, given that we were going to need to create unit test in the future requirements.

Last Requirement Update:

The last requirement update was not as bad as I thought it would be initially, because I felt that maybe the replacement of words would cause problems with the way my linked list pulled words. However, I found that I could simply check to see if the word that needed to be replaced was the word that was currently being read in the file. If it were that word, then I would replace it with the new word, and if it were not then it would not be replaced. When I was creating my unit tests, my creation of individual functions to address each requirement helped me write the unit tests that correlated to each function. All I would need to do was call my functions, and feed in the various possible variables that could be allowed for execution of the function. I did this for each function and followed a similar structure for each one as well, and the tests showed that all variations of variables considered passed. I was a bit shocked to see this, as I figured maybe the test where I attempt to replace a small word with a large word in a large document might fail somewhere, but my data structure held strong through all tests of all functions. Knowing this made me happy, because it let me know that the way I designed my program caused it to remain strong even when I was trying to think of ways to ruin my program, which is a sign of good software design.