As part of the software development cycle for the strings and words project. There were a series of requirements that were implemented onto the design. First the feature to be able to get the frequency count of words within a document was applied. As part of this process the project was setup by first specifying the requirements and stories as a usual project would go by. Next, a series of test cases were analyzed before developing the code. Once the previous steps were completed, I began by adding the ability to read from a document and creating a class that would handle all the statistical analysis for the project. For testing a junit class was created to obtain code coverage on the methods and potential paths the class would use when running the analysis on the document. A few weeks later a second requirement was given, which specified that words could be replaced if a replacement word was given. Therefore, as a result I had to revisit the code and figure out what part of the code would be mostly modifiable to try and implement the results. Of course, given the feature was a new specification, it was treated as a new sprint in the development process. As a result, new user stories were created and new requirements along with test cases were added to the existing ones. For the development on this requirement, I decided to go back and modify the existing method I had created for the first requirement. Given that it already read from the file, I decided that the word searching line by line could be used to also replace a word if it was encountered through the design process. Ultimately, the change resulted successful and the unit tests proved the same. Finally, the last requirement was given and it specified that a new feature called grepline was to be added. This feature required words within a document to return their given line number. The input was a keyword and a document. For this requirement, I followed the same process as the previous by creating new requirements and test cases before applying the code changes. After that was complete, I decided that the best way to apply this feature was to add a new method to the existing class I used for document analysis. The function followed the requirements by passing parameters for a keyword and a document. Within the function I decided to reuse existing code since I was using the line-by-line parsing of the document to analyze the tokens and words from the document. With this in mind, I simply searched for the word on the tokenized line and compared whether it was a match or not. If there was a match, I updated a Boolean variable to true and returned the line number, which was stored in an integer type variable that incremented each time a new line was fed to the method. Ultimately, the software development process for this project turned out to be successful as I obtained close to a 97% code coverage from the unit testing cases I implemented. In addition, to satisfy the tool application for static bug detection and code clone detection on the codebase, I decided to use a few plugins for eclipse. One of them was a plugin called SpotBugs, which I used as part of the static bug detection. What this tool did was check my code for bug patterns, such as null pointer references, infinite loops, and bad uses of libraries or parts of code that resulted in deadlocks. This tool helped me find a bug under one of my conditions that resulted in a null pointer exception. Therefore, I had to go back and correct the conditional statement as well as create a unit test. Aside from that major bug I was able to identify minor problems such as undefined variables and unused code that was left behind from previous “sprints” in the process. I felt I had more issues with the code clone detection as I was not really familiar with the tools that could really give me the best results. However, I attempted to use an existing open-source tool called simpleCC. My understanding for these tools was that there are two ways they analyze code clone detection. One by tokenization and the other by abstract syntax tree application. In this case, simpleCC used the tokenization approach. First it extracted a list of tokens from the java file and then applied normalization rules to then detect code clones. What I ended up seeing was mainly code clone detection where I repeated code, such as in the last requirement where I copied code to analyze line by line the document.