Software Development Project

LOC Counter

Report By: Gabriel Stroe

Course: Software Engineering

Project Dates: September 14th 2016

Date Submitted: September 26th 2016

# Introduction:

The goal of this project was to take an existing the LOC counter program based on the first project and add a new feature: the ability to count the lines of code for each function. The program should count the total lines of code as per the last assignment, print it out, then count the lines of code for every function and print those out along with the names of the functions.

This program should help developers count their lines of code to better understand the current scale of their program and event use past programs to understand what kind of scale you could be looking at in the next project. Specifically targeting the lines of code per function could show how complicated each function is as well as how much work was put in each function.

In the future this project would be adaptable enough to include in as an extension to other more complicated programs. Ideal using a chart or plot to understand the data presented for larger scale programs would be the ideal way to represent this data. For now, the simplicity of this as a command line will satisfy the requirements given to us.

# Estimates/Plans:

The overall estimations for the project is that it will take a total time of 7.98 hours and about 240 lines of code. Given that some of the code was already given and would only need to be slightly modified, an accurate estimate for the count function and main function that handles input and UI can be done. The new countFuncLOC function is an educated guess based on the idea that it will be more complicated than the countLOC function.

The plan for this project is to spend is that I spend 9/25 working on the design features and coding of the program. On 9/26/2016 the code would be tested and a report will be written up. Ideally the time spent on the project should be across a larger span of time, but due other items these are the times that are being designated to it.

# Requirements Analysis:

The requirements for this project are that the program is able to take a sample code and count the number of lines in that code. In addition for every function in the application, the program should count the lines of code and print it with the function name. The program will be required to have a function called countFuncLOC that does the counting for the functions and the total lines of code. The old countLOC function can be used as an extension of this function if desired. The code must be built on top of an already existing example which is provided by the instructors. It also must have a UI that the user can input parameters and see the results for this counting function. Because given example uses a terminal interface, this extension of the originally program will continue to use the same terminal interface.

# Design:

The basic idea of this application is that the programs will that the program will take in a file name, convert it to a buffered reader send it to the original countLOC function. The countLOC function will read the buffer one line at a time, and then determine if that line counts as a line of code for the whole file.

Then the program will scan each line to determine the start and end of a function. When a function is found, it will be put into a string from its start to its end, and then sent into a buffered reader so that the countLOC function can count it as if it was a file. This is then stored in a list with the other functions. Also, when the start and end of a function is found it will store the name of the function based on the whole word (no spaces between) before the parenthesizes.

Below is the logical design diagram of the identification of a function. For every line starting with the one that begins the function it will be added into a buffer that will be sent to countLOC when it is complete. A function is noted as completed when the total number of ‘{‘matches the number of ‘}’.



Unfortunately during the testing phase I had realized that this logic does not account for parenthesizes inside of a comment. So I need to spend some time redesigning this logic. The following diagram shows the logic for this function with the looking for

comments.

G:\Users\Gabe\Downloads\Project 2 org diag (1).png

# Implementation:

The implementation for this program is a Java program created in Intellij Studio. The coding conventions used are traditional java coding conventions.

# Testing:

The base testing involves two files. A C file and a Java file. The two file formats have slightly different schemes for how the code is implemented. This is especially import for the function count test. When tested for the total LOC function both files work as expected on the first try.

For the function counting tests, the first test case an error where no data was printed. I was able to find the issue and resolve it. The problem was that it would send the data to the function counter that is already on the end of the file. This solution was solved by loading in the file twice, one for the total LOC and the other for the function LOC.

After that was fixed the next error that was found was that if the bracket was not on the same line as the starting of the function, it would not be counted. The solution was to move the point in the code where the initial function line gets added. Based on how I made my diagram this is how the code should have been in the first place.

Following that it appeared that the code was working properly, until I had released that there is a situation where my code would not work and that was if there were a set of parenthesizes are inside a comment. So I added a fix for that and modified the java file to test for it directly. It worked on the first try.

Lastly while writing the conclusion for this report I encountered an issue when trying to use the program to test its own LOC. It would show the countFuncLOC function as having 0 lines of code. After a while I had realized that this was because the function contained braces inside “ ” and ‘ ’. This cause the function to never state that it is finished and not submit anything to the countLOC function before it reached the end of the file.

For a more detailed analysis of the test cases please refer to the Test Log.xlsx file.

# Conclusion/Project Retrospective Analysis:

In conclusion this project took significantly longer then I was expecting. This is largely due to the number of issues I actually had to fix and how long the report to me to write. It took quite some time to turn my program design elements into a digital format and write out my observations and analysis. I found that I had also underestimated the complexity of being able to distinguish a function accurately with every possible scenario. In terms of lines of code It would appear that my initial estimations where actually fairly accurate.

For the next project I will be looking to overestimate some of my coding and debugging time. Clearly the time estimations and the overall start time of this project could have be a lot earlier.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Estimated: | Measured: | Analyze: %Error |
| Time | 7.98 Hours | 13.25 Hours | 39.8 % |
| Size | 240 LOC | 254 LOC | 5.5 % |

Actual Project Time and Program Size vs. Estimates

# Appendices:

* Project Log.xlsx – The project log that contains estimates, time logs and defect logs.
* Test Log.xlsx – A spreadsheet of the different test performed on the code.
* LOCTest.c – The first file tested for both functions.
* LOCcounter.java – The base file given to build this project off of. Used in the test cases.
* Project 2.jar – The executable for this project.