Software Development Project

Project 1, Lines of Code Counting

Report By: Nicholas Doherty

Course: Software Engineering, 3034, 01

Project Dates: 10/1/17 – 10/10/2017

Date Submitted: [Date Report Submitted]

[**General Comments**:

* Your report should stand on its own; delete all of ***MY*** blue comments in the square brackets (INCLUDING the brackets) and replace them with ***your own*** content. Each section should include a ***brief*** summary explanation (one or two sentences) as to the purpose and/or content of the section.
* One important goal of this class is to help you to become more effective in communicating complex and abstract ideas to others (both technical and non-technical).
* As you work through your project and record the associated documentation, try to keep in mind the thought “What if someone else were assigned to implement or modify my software?” Consider the following realistic scenarios:
  + Immediately after completing your project, you are promoted or assigned to a new project. Your software (THIS project) requires a change (enhancement, fix, new customer, etc.) “RIGHT AWAY!” There is no margin for error, and in your new role or assignment, you are too busy to make the change. How much difficulty would a new team member have (given your project documentation - requirements, design, and test - and source code) modifying your software (requirements, design, code) ***with no mistakes***? Could they repeat your testing to make sure that they did not break anything that used to work correctly? (Also keep in mind that this lucky person modifying your software might be **YOU** six months or two years from now... Did you leave yourself enough clues to remember what you were thinking way back when?)
  + You are managing a project, and you have the domain expertise to completely understand the customer’s requirements and expectations. However, you will be overseeing a team of developers as ***they*** develop the software. Will the development team understand your statement of requirements ***without misinterpretation***? Will an independent validation team be able to use your statement of requirements to develop test cases to validate the final product against the customer’s expectations?
  + You are a lead engineer on a team of developers. You understand the requirements, and you have a concept of how to decompose the software into “components” (whether functions or objects). You will need to distribute the development work among the team members to complete the project on schedule. How will you communicate High-Level Design (your concept of the program structure) to you team so that they can independently design, implement, and test their components? The High-Level Design includes the overall decomposition into components, the interfaces and interactions between the components, call/return hierarchy and interface structure (function prototype).

**General Suggestions**:

* In general, write your project report as if you were presenting it to somebody who has knowledge of the subject matter but was not involved with the project. Include diagrams, tables, etc. throughout your report as needed for the reader to understand your report.
* Label ALL figures, diagrams, tables, etc. throughout the report.
* The general flow of the report should be something like:
* Summarize project Purpose, Mission, and Vision
* Time and Size Estimates & Plan
* Requirements Analysis
* Design
* Implementation
* Testing
* Retrospective Analysis
* Appendices

THIS IS A GENERAL LAYOUT SUGGESTION, **NOT** A REQUIRED LAYOUT, FORMAT, ETC.

Also understand that the various sections of the template represent a GENERAL project/development workflow. Include the information that is specific to **YOUR** project/development workflow.]

# Introduction:

The Lines of Code (LOC) counter is meant to receive a project source file and count the active lines of code contained within the file. The purpose of such an instrument is to give a programmer an idea of the size of their program and thus how long it might take to compile. As a result, a programmer can make a sound decision as to whether or not they need to trim the fat from their program or if they are clear to add more without overburdening a PC.

# Estimates/Plans:

Since the project simply opens a file, reads it, and reports back the amount of valid Lines of Code (LOC), the project should go relatively quickly. The code will be simple enough, probably taking only somewhere around 3 hours to code and debug. The Interface design will probably only take 2 hours to complete and implement. The project plan will be to use a sprint model to set aside a day in order to code and debug; with another day test individual test cases before submitting the project before the deadline of Tuesday, October 10, 2017.

# Requirements Analysis:

The final product will open to main, where the user will be able to input the file path to the desired file(s). Once the user does so and closes the window, the program will open each selected file and pass it to a function that then counts all active LOC contained therein. It is important to note that a single LOC is counted as a valid line of ascii characters that contribute to the over function of a program. This excludes any and all comments or blank lines. As scope brackets define the code contained within functions or loops, a LOC might therefore include nothing more than a scope bracket. Upon completion, the function will return the integer indicating the total LOC. The program will then output the name of the file and the resultant LOC before closing the file and repeating the process. At the end, the program will output the total number of files counted and then close.

# Design:

The LOC counter will consist of a main function that takes user input and outputs information to the user as well as a LOC counting function that contains the code definitions of what is and is not a valid LOC. Upon booting the program, the main function will accept the input describing which, if any, source files are to be counted and then opens the file for counting, passes the contents to the counter, and then closes the file when finished and outputs the results. The counter function will not manage the files or accept user input. In fact, the user will never know that the counter is a separate function. The counter function will be a back-end function that accepts the contents of the file and checks them against its definitions of what counts as code. It will then return the results and conclude its business.

Main Inputs:

User interaction variables

Input files

Main outputs:

File names

Count results

Counter inputs:

File contents

Counter outputs:

Line count

[Include or summarize (along with diagrams or references to the appendices containing them) any more detailed Design analysis as to *HOW* the various parts of the program will be integrated and will interface (“Architecture” or “High-Level Design”) and *HOW* the individual components of the program will work (“Detail Design”).

Include a Context Diagram (either in the Requirements or Design section) showing ALL system/software Inputs and Outputs, along with an indication of the source of the input and destination of the output (these will be devices, “Actors”, etc.)

A “High-Level” or “Architectural” design of the overall system/software shows all major “components” of the system/software (whether objects or functional blocks) and their relationships and/or interfaces (how they share information, call and control hierarchy (who calls who), etc. As a starting point, consider any “components” that you assumed during “decomposition” in the planning and estimating process; revise and refine these initial decomposition assumptions as necessary.

For each “component”, a “Low-Level” or “Detail” design should define all Inputs and Outputs of the component (like a Context Diagram for the component), all sub-components and their interactions and interfaces, and implementation details (algorithm, logic, states and transitions, equations, flow chart, structured language, etc.) on how the component transforms its inputs into its outputs and/or provides its intended behavior.]

# Implementation:

[Summarize details related to the implementation methodology (such as coding standards, code review, etc.)  
  
**DO NOT PASTE SOURCE CODE LISTINGS HERE**.]

# Testing:

[Include or summarize (along with reference to appropriate appendices) any more detailed Verification/Validation plans and outcomes. Consider here how will you demonstrate (to yourself and to your customer) that the system/software (individual components and the overall integrated system) functions as expected.

Document your test cases (what you tried; any special set-up and the inputs that you provided), expected results (how you expected the system/software to respond/behave) and observed results. For each test case, did the system/software respond/behave as expected (“PASS”) or not (“FAIL”)?

The “Test Log Template.xlsx” is provided to help with the test planning and logging.

Self-check: Could your testing be performed by or repeated and confirmed by an independent tester based on the information that you have provided?]

# Conclusion/Project Retrospective Analysis:

[This section contains closing remarks summarizing the key outcomes and observations from the project. Consider the data in the both the Time Log and Defect Log in this analysis. Include a summary of the estimated vs. actual Time and Size. Include a summary of “Things Gone Right” and “Things Gone Wrong.” Include a brief discussion of any “Lessons Learned” that can be applied in future projects.]

|  |  |  |  |
| --- | --- | --- | --- |
|  | Estimated: | Measured: | Analyze: %Error |
| Time | Hours | Hours | % |
| Size | LOC | LOC | % |

EXAMPLE: Actual Project Time and Program Size vs. Estimates

# Appendices:

[Include Time and Defect Logs, along with any detailed planning worksheets, project schedule, Requirements Analysis, Design Documentation, Testing Plans and Outcomes, etc. that are referred to elsewhere in the report.

The Time Log is essentially a journal with entries (single line entries are fine) that include:

* Date
* Development phase or activity
* Time duration
* Comments, if desired.

(The Process Dashboard contains all of the expected fields for Time Log entries.)

Evaluating the Time Log should allow you to analyze your workflow patterns for possible process improvement actions.

The details in the Defect Log are important for identifying areas that need improvement and effective ways to avoid future occurrences of the same problems. Each Defect Log entry should include:

* A brief description of the defect or the observed “misbehavior”
* Some classification of the type of defect (logic, syntax, requirement specification, etc.)
* The phase of the project in which the defect was introduced
* The phase of the project in which the defect was discovered
* The time that it took to diagnose, find, and fix the defect
* An indication of whether the defect was introduced in the process of resolving another defect (a “Fix Defect”).

(The Process Dashboard contains all of the expected fields for Defect Log entries.)

If using Process Dashboard, do not include “screen shots” of the Time and Defect logs; these are often very difficult to read. **There is documentation provided by the instructor on how to export these logs to Excel format**. Include the Time and Defect logs either as Excel sheets or as Tables in Word pasted from Excel.]