International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, Volume 2, Issue 5, May 2012)

Analysis Of Source Lines Of Code(SLOC) Metric

Kaushal Bhatt[1], Vinit Tarey[2], Pushpraj Patel[3]
[1,2,3] *Kaushal Bhatt MITS ,Datana Ujjain*
[1]kaushalbhatt15@gmail.com
[2]vinit.tarey@gmail.com
[3]pushpraj.patel@yahoo.co.in

***Abstract*—Source lines of code (SLOC) is a software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code. SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced.There are some drawbacks in SLOC metrics that affect the quality of software because of SLOC metric output is used as a input in other Software Estimation methods Like COCOMO Model.**

***Keywords*— SLOC, Metrics, Software Estimation, COCOMO, Performance Analysis**

I. Introduction

There sometimes is a decidedly dark side to SLOC software metric that many of us have observed, but few have openly discussed. It is clear to us that we often get what we ask for with software metrics and we sometimes get side effects from the metric that overshadow any value we might derive from the metrics information. Whether or not our models are correct, and regardless of how well or poorly we collect and compute software metrics, people's behaviors change in predictable ways to provide the answers management asks for when metrics are applied. For this reasons we Consider Source lines of code metrics for our analysis. SLOC Metric works on Source Lines of code but when we talk about the different Programming Languages the Lines of code to perform task is vary language to language because different languages have different structures. Our in-depth analysis in this field provide a comprehensive view to understand what SLOC metric do and how much one can rely on SLOC metric.

II. Advantage And Disadvantages Of loc Metrics

Advantages:-

1. *Scope for Automation of Counting:*

Since Line of Code is a physical entity; manual counting effort can be easily eliminated by automating the counting process. Small utilities may be developed for counting the SLOC in a program.

However, a code counting utility developed for a specific language cannot be used for other languages due to the syntactical and structural differences among languages.

2. *An Intuitive Metric*:

Line of Code serves as an intuitive metric for measuring the size of software because it can be seen and the effect of it can be visualized. Function points are said to be more of an objective metric which cannot be imagined as being a physical entity, it exists only in the logical space. This way, LOC comes in handy to express the size of software among programmers with low levels of experience [4].

Disadvantages:-

1. *Lack of Accountability*:

Lines of code measure suffers from some fundamental problems. Some think it isn't useful to measure the productivity of a project using only results from the coding phase, which usually accounts for only 30% to 35% of the overall effort [2].

2. *Lack of Cohesion with Functionality:*

Though experiments have repeatedly confirmed that effort is highly correlated with SLOC, functionality is less well correlated with SLOC. That is, skilled developers may be able to develop the same functionality with far less code, so one program with less SLOC may exhibit more functionality than another similar program. In particular, SLOC is a poor productivity measure of individuals, because a developer who develops only a few lines may still be more productive than a developer creating more lines of code - even more: some good refactoring like "extract method" to get rid of redundant code and keep it clean will mostly reduce the lines of code[2].

3. *Adverse Impact on Estimation:*

Because of the fact presented under point 1, estimates based on lines of code can adversely go wrong, in all possibility.

4. *Developer's Experience:*

Implementation of a specific logic differs based on the level of experience of the developer. Hence, number of lines of code differs from person to person.

An experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience does, though they use the same language.

5. *Difference in Languages:*

Consider two applications that provide the same functionality (screens, reports, databases). One of the applications is written in C++ and the other application written in a language like COBOL. The number of function points would be exactly the same, but aspects of the application would be different. The lines of code needed to develop the application would certainly not be the same. As a consequence, the amount of effort required to develop the application would be different (hours per function point). Unlike Lines of Code, the number of Function Points will remain constant.

6. *Advent of GUI Tools:*

With the advent of GUI-based programming languages and tools such as Visual Basic, programmers can write relatively little code and achieve high levels of functionality. For example, instead of writing a program to create a window and draw a button, a user with a GUI tool can use drag-and-drop and other mouse operations to place components on a workspace. Code that is automatically generated by a GUI tool is not usually taken into consideration when using SLOC methods of measurement. This results in variation between languages; the same task that can be done in a single line of code (or no code at all) in one language may require several lines of code in another.

7. *Problems with Multiple Languages:*

In today's software scenario, software is often developed in more than one language. Very often, a number of languages are employed depending on the complexity and requirements. Tracking and reporting of productivity and defect rates poses a serious problem in this case since defects cannot be attributed to a particular language subsequent to integration of the system. Function Point stands out to be the best measure of size in this case.

8. *Lack of Counting Standards:*

There is no standard definition of what a line of code is. Do comments count? Are data declarations included? What happens if a statement extends over several lines? – These are the questions that often arise. Though organizations like SEI and IEEE have published some guidelines in an attempt to standardize counting, it is difficult to put these into practice especially in the face of newer and newer languages being introduced every year.

9. *Psychology of Programmer:*

A programmer whose productivity is being measured in lines of code will have an incentive to write unnecessarily verbose code. The more management is focusing on lines of code, the more incentive the programmer has to expand his code with unneeded complexity. This is undesirable since increased complexity can lead to increased cost of maintenance and increased effort required for bug fixing.

III. MEASUREMENT OF LOC METRIC

For measurement of SLOC Metric We can use Metric Computational Tool (LOC Metrics). [6]

Tool Compute the following aspects of Source Code.

1. *Physical Lines:* Physical Lines are program's source code including comment lines.
2. *Logical Lines:* Logical Lines are number of executable statements.
3. *Blank Lines:* Lines in Code which is Blank.
4. *Total Lines of code:* Total Lines of code include Physical lines with Blank Lines
5. *Executable Physical:* Physical executable source lines of code is calculated as the total lines of source code minus blank lines and comment lines.[4]
6. *Executable Logical:* Executable Logical is number of statements that is executed.
7. *Comment*: Comment lines in code.
8. *Words in Comment*: Total No. of words in comment lines.
9. *Header Comment*: Comment of Header Part.
10. *Header Words*: Total words in Header Comment.
11. *McCab VG Complexity*: The McCab complexity is directly measures the number of linearly independent paths through a program's source code. The complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command[7].

Mathematical Representation of McCabe complexity is:-

$$M = E - N + 2P$$

Where M= Complexity
E= Number of Edges in graph
N= Number of Nodes in graph
P= Number of connected component(exit nodes)

Kaner's Ten Measurement Factors [1]

1. The purpose of the measure. What the measurement will be used for.
2. The scope of the measurement. How broadly the measurement will be used.
3. The attribute to be measured. E.g., a product's readiness for release.
4. The appropriate scale for the attribute. Whether the attribute's mathematical properties are rational, interval, ordinal, nominal, or absolute.
5. The natural variation of the attribute. A model or equation describing the natural variation of the attribute. E.g., a model dealing with why a tester may find more defects on one day than on another.
6. The instrument that measures the attribute. E.g., a count of new defect reports.
7. The scale of the instrument. Whether the mathematical properties of measures taken with the instruments are rational, interval, ordinal, nominal, or absolute.
8. The variation of measurements made with this instrument. A model or equation describing the natural variation or amount of error in the instrument's measurements.
9. The relationship between the attribute and the instrument. A model or equation relating the attribute to the instrument.
10. The probable side effects of using this instrument to measure this attribute. E.g., changes in tester behaviors because they know the measurement is being made.

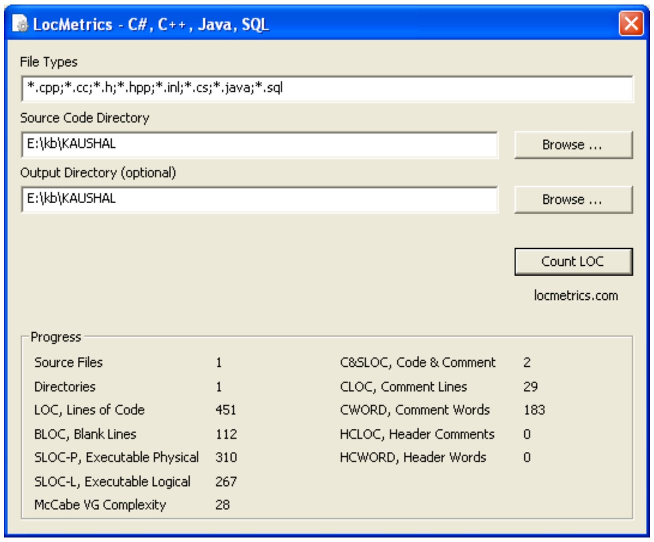
IV. Realization

```
Code :
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class Calculator implements ActionListener{

- -
- -
- -
- -
        fmi1 = new MenuItem(" Copy   ");
        fmi2 = new MenuItem(" Paste  ");
        fmi3 = new MenuItem(" Quit    ");
        EditMenu.add(fmi1);
        EditMenu.add(fmi2);
- -
- -
- -
```

```
- -
 ("=");
        backspace = new Button    ("Backspace");
        clear = new Button    ("C");
 }

- -
- -
- -

    public void launchFrame(){

        tField.setText("0.");
        tField.setEnabled(false);

    - -
- -
        p6.add(bAdd);
        p6.add(equals);
- -
- -
GridLayout (6, 1) );
- -
- -
        f.setMenuBar(menuBar);
        f.pack();

    // ACTION LISTENERS
- -
- -
- -
        fmi1.addActionListener(this);
        fmi2.addActionListener(this);
        fmi3.addActionListener(this);
    }

 /* |------------ START OF ACTION EVENTS ---------
---| */

 public void actionPerformed(ActionEvent a){

- -
- -
- -            tField.setText(value);
        }
        if(a.getSource()==num7){
            value+=7;
            tField.setText(value);
        }
- -
- -
- -
        /* |-- EQUALS ACTION --| */
           if(a.getSource()==equals){
                       value="";
                       v2 =
    Double.parseDouble(tField.getText());
                 if(o=='+'){
    - -
    - -
    - -
    if(o=='%'){
                       ctr=0;
                       answer = v1 % v2;
                       tField.setText("" +answer);
                       value=""; v1=null; v2=null;
                 }
                 else{}
           }

        /* |-- EQUALS ACTION --| */
    - -
    - -
    - -
        // END OF ACTION EVENTS
     public static void main (String args[]){
        Calculator s = new Calculator();
        s.launchFrame();
     }
    }
```

The following results has been generated for the calculator code written in java using the SLOC Calculation

Compilation of Code in tool:-



LocMetrics - C#, C++, Java, SQL

File Types
.cpp;.cc;*.h;*.hpp;*.inl;*.cs;*.java;*.sql

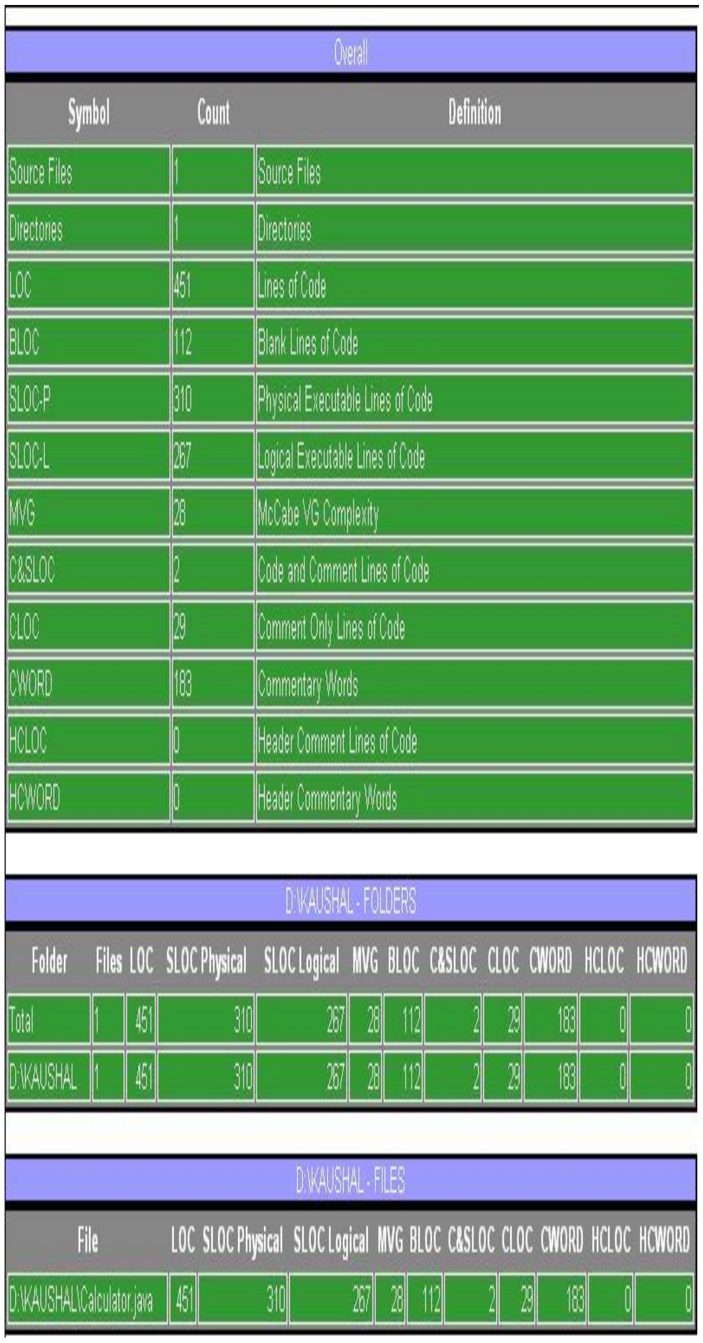
Source Code Directory
E:\kb\KAUSHAL

Output Directory (optional)
E:\kb\KAUSHAL

Progress

Source Files	1	C&SLOC, Code & Comment	2
Directories	1	CLOC, Comment Lines	29
LOC, Lines of Code	451	CWORD, Comment Words	183
BLOC, Blank Lines	112	HCLOC, Header Comments	0
SLOC-P, Executable Physical	310	HCWORD, Header Words	0
SLOC-L, Executable Logical	267		
McCabe VG Complexity	28		

Metric Calculation By SLOC calculation Software [3]*:-*



Overall

Symbol	Count	Definition
Source Files	1	Source Files
Directories	1	Directories
LOC	451	Lines of Code
BLOC	112	Blank Lines of Code
SLOC-P	310	Physical Executable Lines of Code
SLOC-L	267	Logical Executable Lines of Code
MVG	28	McCabe VG Complexity
C&SLOC	2	Code and Comment Lines of Code
CLOC	29	Comment Only Lines of Code
CWORD	183	Commentary Words
HCLOC	0	Header Comment Lines of Code
HCWORD	0	Header Commentary Words

D:\KAUSHAL - FOLDERS

Folder	Files	LOC	SLOC Physical	SLOC Logical	MVG	BLOC	C&SLOC	CLOC	CWORD	HCLOC	HCWORD
Total	1	451	310	267	28	112	2	29	183	0	0
D:\KAUSHAL	1	451	310	267	28	112	2	29	183	0	0

D:\KAUSHAL - FILES

File	LOC	SLOC Physical	SLOC Logical	MVG	BLOC	C&SLOC	CLOC	CWORD	HCLOC	HCWORD
D:\KAUSHAL\Calculator.java	451	310	267	28	112	2	29	183	0	0

V. Conclusion

SLOC is most useful Metric in Software Estimation but Some restrictions that is involved in SLOC Metric makes it vurnable for relying on it. Here we can present the depth Analysis of SLOC Metric for the purpose to expose some of its disadvantages and provide knowledge of SLOC Metric Measurement.

REFERENCES

[1] Kaner, C. "Rethinking Software Metrics," Software Testing and Quality Engineering vol. 2, no. 2 (2000).

[2] The Darker Side of Metrics by Douglas Hoffman

[3] http://www.locmetrics.com

[4] Wheeler, David A. (June 2001). "Counting Source Lines of Code (SLOC)"

[5] Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric NIST Special Publication

[6] Park, Robert E., et al.. "Software Size Measurement: A Framework for Counting Source Statements"

[7] Rich Sharpe. "McCabe Cyclomatic Complexity: the proof in the pudding