Analysis of Static and Dynamic Metrics for Productivity and Time Complexity

Manik Sharma
Assistant Professor & Head
Deptt. of Computer Science & Applications
Sewa Devi S.D. College
Tarn Taran, India

Dr. Gurdev Singh
Professor & Head
Deptt. of Computer Sci. & Engineering
Adesh Institute of Engg. & Tech.
Faridkot, India

ABSTRACT

The aspiration of this study is to perform the comparative analysis of static and dynamic metric for structured programming environment. Software metrics is one of the vital tools that can be worn to find significant estimates for software products and directs us in intriguing managerial and technical decisions. Software metrics have become an integral part of software development and are used during every phase of the software development life cycle. Research in the area of software metrics tends to focus predominantly on static metrics that are obtained by static analysis of the software artifact. But software quality attributes such as execution time, performance and reliability depend on the dynamic activities of the software artifact. With the help of conventional static metrics we are not able to analyze various facts of software's. It is very important to understand the dynamic behaviour of the program or an application in developing new effective strategies in computer science. This becomes the basis for working on dynamic metrics in place of traditional static metrics. Dynamic metrics gives more accurate result than static metrics as they are able to capture the dynamic behaviour of the software system during measurement.

Keywords

Software, Metric, Accuracy, Performance.

1. INTRODUCTION

Software metric is one of the important aspects of software engineering acts as an indicator for software attribute. It plays an important role in understanding the important concepts in the field of software engineering. The name software metric [1][2] is associated with diverse measurements of computer software and its development. It helps us in measuring the performance of various features of the software. With the help of software metric once can measure some property of software or its component. Computer science researchers are putting all their efforts in measuring quantitative information from software component. Software metric [3] are helpful in improving the quality of software, planning the budget, its cost estimation etc., In other words software metric is a way to understand software product in an effective way. We apply some software logical of mathematical technique to software process or product to supply or improve engineering and management information [4]. Some of the software metric's objective [5][6] are perception, software inspection, planning, optimization and quality improvement.

1.1 Objectives: The objectives of this article are:

- To study the foundation of static and dynamic metrics
- Measuring static and dynamic metric for software productivity and time complexity.
- Comparative analysis of static and dynamic metrics for productivity and time complexity.
- Comparative analysis of different programming approaches for time complexity.

2. STATIC AND DYNAMIC METRIC

The research cycle of software metrics starts in 1970, it was Wolverton [6] who performs a research on production ratio of the programmer by using the concept of LOC i.e. line of code. According to Somerville the metric can be classified into two categories i.e. control metric and predictive metric. Predictive metric are normally associated with software product. With the help of predictive metric [7] we are able to determine both static as well as dynamic characteristics of the software. There are two major types of predictive metrics i.e. Static and Dynamic Metrics.

2.1 Static metric: First static metric [8] (LOC/KLOC) was used to measure the productivity of a program. The most commonly used complexity metric before 1990 was cyclomatic [9] complexity that was measured by McCabe. He uses the flow graph and some mathematical equations to compute software complexity. This metric was used in code development risk analysis [10], change risk analysis in maintenance and in test planning. In 1976 McCabe [11] defined the cyclomatic complexity number metric. The metric measures the number of independent paths through a software module. Although cyclomatic complexity is widely used, critique on it exists claimed that it's based on poor theoretical foundations and an inadequate model of software development. The cyclomatic complexity has been selected to be a part of the benchmarks.

Since the initiate of software engineering engineers have been counting the lines of code they wrote. Counting lines is used for estimating the amount of upholding or maintenance required and it can be used to normalize other software metrics. For Example consider the following effective segment code of a program:

```
clock_t start, end;
```

```
clrscr();
start = clock();
//perform calculations for which performance needs to be checked
for(i=1;i<=100;i++)
{
        for(j=1;j<=100;j++)
    {
        cout<<"hello";
        }
    }
end = clock();
```

Table 1: LOC - A Static Metric

I	J	Number of Characters	Line of code
100	100	1	12
100	100	2	12
100	100	3	12
100	100	4	12
100	100	5	12
100	100	6	12
100	100	7	12
100	100	8	12
100	100	9	12
100	100	10	12

Now again consider the same code with functional approach. We observe that again the line of code remain independent of the number of printed character on console, however now line of code (LOC) is more as compare to code where no functional approach is used.

```
clock_t start, end;
clrscr( );
start = clock( );
//perform calculations for which performance needs to be checked
hello( );
end = clock();
void hello()
{
for(i=1;i<=100;i++)
{
        for(j=1;j<=100;j++)
        {
                cout<< "hello";
                }
        }
}
```

Table 2: LOC - A Static Metric

I	J	Number of Characters	Line of code
100	100	1	14
100	100	2	14
100	100	3	14
100	100	4	14
100	100	5	14
100	100	6	14
100	100	7	14
100	100	8	14
100	100	9	14
100	100	10	14

Again consider the static measure LOC [12] for same effective segment code with recursion. Again it is observed that the LOC is independent of number of printed characters, but again LOC is different from iterative as well as functional approach.

```
clock_t start, end;
clrscr();
start = clock();
//perform calculations for which performance needs to be checked
hello();
end = clock();
void hello( )
{
        static int flag=1;
        if (flag<=100*100)
        {
                cout<<"hello World";
                flag++;
                hello();
        }
}
```

Table 3: LOC - A Static Metric

I	J	Number of Characters	LOC
100	100	1	16
100	100	2	16
100	100	3	16
100	100	4	16
100	100	5	16
100	100	6	16
100	100	7	16
100	100	8	16
100	100	9	16
100	100	10	16

The static metric LOC (Lines of code) will never change if we change the number of character in printing statement. It will always give the same measure that is when we print just “H”, “He”, “Hel”, “Hell”, “Hello world” etc. this means the number of character in printing statement does not give any impact in LOC.

But the execution time of the printing statement is heavily depends upon the number of character that can be measure by using the dynamic metrics only. The following chart give the comparative analysis of number of character and programming approach used in C language.

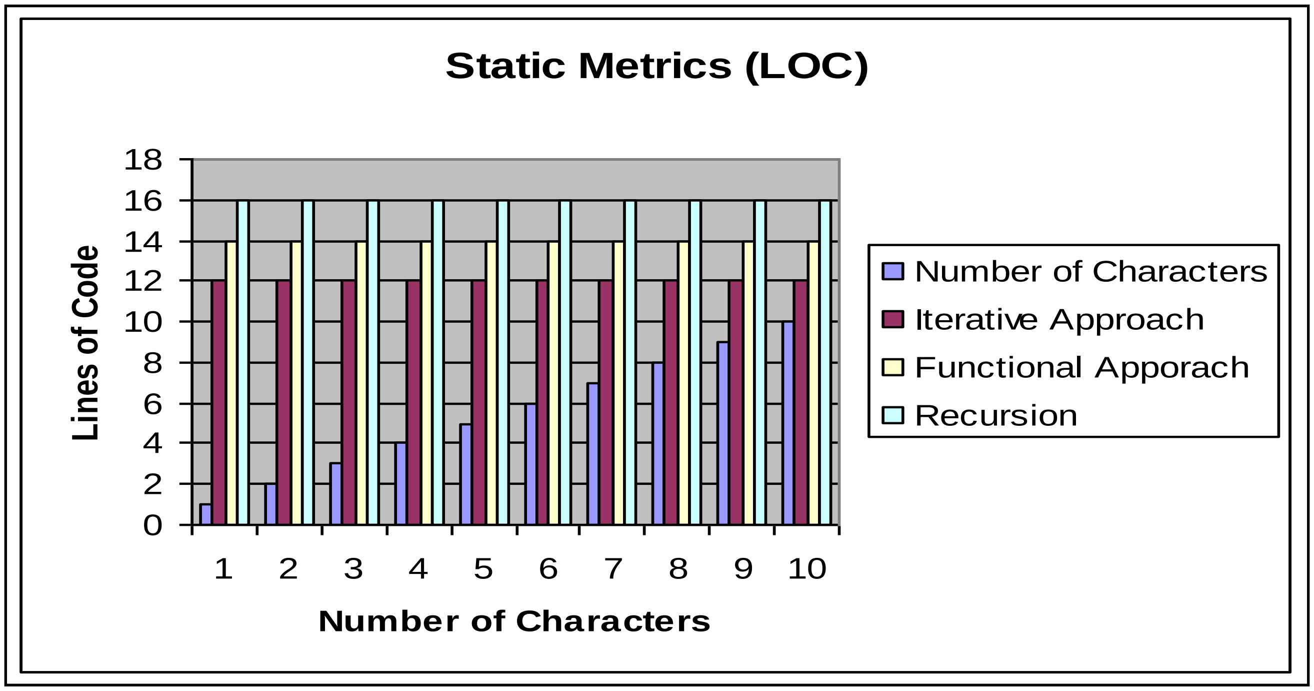

Figure1: Number of character versus LOC

From the above analysis it is crystal clear that static software metric does not change by changing the contents of a program. I also lag behind various factor while measurements.

2.2 Dynamic Metric: Dynamic metrics [13] that can be used to evaluate relevant runtime properties of programs, with the vital goal of establishing some standard metrics that could be used for quantitative analysis of standard programs. Dynamic Metrics are derived from an analysis of code while it is executing. Thus dynamic metrics can only be calculated on the software as it is executing. For example: extent of class usage, Dynamic Coupling, and Dynamic Lack of Cohesion

Dynamic metrics in contrast to static metric [14] have a time aspect and the values tend to vary over time. Dynamic metric give us more accurate and efficient result as comparative to static metric because dynamic metric analyze the program in working or running environment. Dynamic metrics differs from static metrics in various respects as discussed below. Static metrics measures are associated with static program i.e. non executing code, on the other hand dynamic metrics are associated with executing code. Static metrics is independent of the input test data where as dynamic metrics is heavily based on it. Static metric is independent of running environment where as majority of dynamic metrics are measured under running environment by taking care of machine architecture, operating system, language used, compiler used etc. Further it should be noted that Static metric are affected by non executing code like comments, blank line, blank space where as Dynamic metric is not affected by these factors. In concern of accuracy obviously dynamic metrics are more accurate as compare to static metrics. At last Static metrics are greatly affected by the programming technique used for developing program, in contrast Dynamic metric are least or even independent of the programming technique used.

A lot of research has been focused on the measurement of source code of programs now for experiment considers the above 'C' language code, the execution time depending upon the number of character is measures and is shown as below:

Table 4: Average Execution Time for Characters

Number of Characters (C)	**Execution Time**		**Average Execution Time (T)**
	Min	Max	
1	0.10989	0.164835	0.137363
2	0.274725	0.32967	0.302198
3	0.384615	0.43956	0.412088
4	0.549451	0.604396	0.576924
5	0.714286	0.769231	0.741759
6	.879191	.879121	0.879121
7	1.043956	0.989011	1.016483
8	1.153846	1.208791	1.153845
9	1.263736	1.318681	1.291207
10	1.483516	1.538462	1.428569

```
start = clock( );
//perform calculations for which performance needs to be checked
hello( );
end = clock();
void hello()
{
for(i=1;i<=100;i++)
{
        for(j=1;j<=100;j++)
        {
                cout<<"hello";
                }
        }
}
```

Number of Characters	Execution Time		Average Execution Time
	Min	Max	
1	0.10989	0.164835	0.137363
2	0.274725	0.32967	0.302198
3	0.384615	0.43956	0.412088
4	0.549451	0.604396	0.576924
5	0.714286	0.769231	0.741759
6	.879191	.879121	.879121
7	1.043956	0.989011	1.016483
8	1.153846	1.208791	1.153845
9	1.263736	1.318681	1.291207
10	1.483516	1.538462	1.428569

```
start = clock();
//perform calculations for which performance needs to be checked
hello();
end = clock();
void hello( )
{
        static int flag=1;
        if (flag<=100*100)
        {
                cout<<"hello world";
                flag++;
                hello();
        }

}
```

Number of Characters	Execution Time		Average Execution Time
	Min	Max	
1	0.10989	0.164835	0.137363
2	0.274725	0.32967	0.302198
3	0.384615	0.43956	0.412088
4	0.549451	0.604396	0.576924
5	0.714286	0.769231	0.741759

6	.879191	.879121	.879121
7	1.043956	0.989011	1.016483
8	1.153846	1.208791	1.153845
9	1.263736	1.318681	1.291207
10	1.483516	1.538462	1.428569

In the above table the I,J refer to looping variable, min and max are the minimum and maximum time taken by the code. From the above table it is very clear that the execution time is greatly influenced by the number of characters, i.e. the execution time increases with increased number of characters.

The graphical representation of the above concept is as shown below:

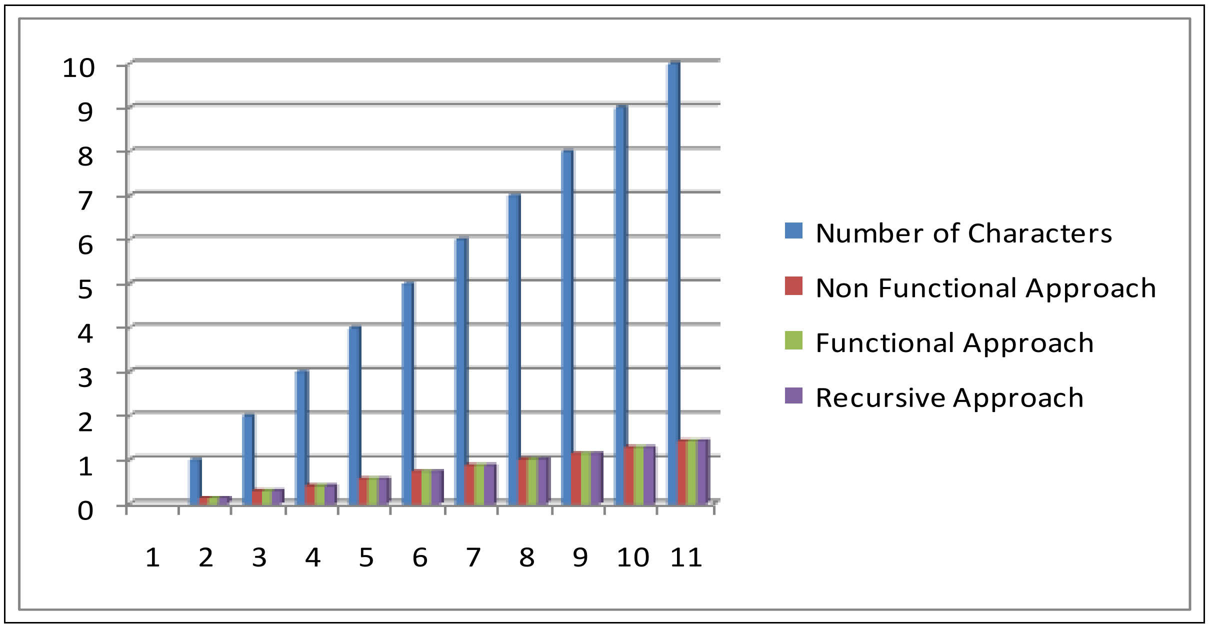

Figure2: Analysis of Number of character and execution time

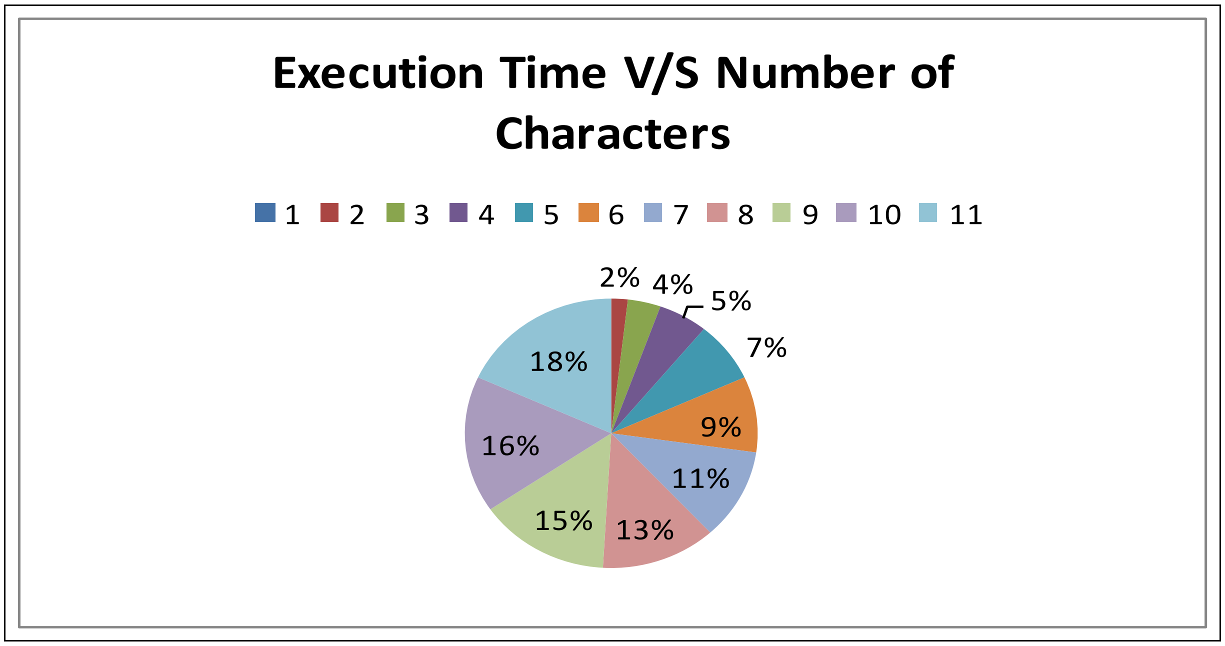

Figure3: Number of Character versus exaction Time

From the above graphical representation it is very clear that execution time for character printing is independent of the structured programming approach used.

Keeping I, J loop variable as constant we can derive the dynamic metric by using the concept of lagranginan interpolation. Let us consider the first three case of dynamic metric having number of characters and average execution time as given below only.

Number of Characters (C)	Average Execution Time (T)
1	0.137363
2	0.302198
3	0.412088

In order to derive the basic dynamic metric we will consider the second order polynomial made up from Number of characters (C) and Average execution time (T) as follow:

$$\boldsymbol{T(C) = a1 * (c1 - c2) * (c1 - c3) + a2 * (c2 - c1) * (c2 - c3) + a3 * (c3 - c1) * (c3 - c2)} \qquad \textbf{1.1}$$

The above equation gives the relationship that exists between number of character and its average execution time. For simplicity we have taken a polynomial of second degree.

Number of Character (C)		
c1	$T(c1) = (c1 - c2) * (c1 - c3)$	$a1 = T1/(c1 - c2) * (c1 - c3)$
c2	$T(c2) = a2 * (c1 - c2) * (c1 - c3)$	$a1 = T2/(c2 - c1) * (c2 - c3)$
C3	$T(c3) = a3 * (c3 - c1) * (c3 - c2)$	$a1 = T3/(c3 - c1) * (c3 - c2)$

By substituting the value from above table in equation 1.1 we are able to derive the relation as follow:

$$T(C) = T1/(c1 - c2)\,) * (c1 - c3) * (c1 - c2) * (c1 - c3) + T2/(c2 - c1) * (c2 - c3) * (c2 - c1) * (c2 - c3) + T3/(c3 - c1) * (c3 - c2) * (c3 - c1) * (c3 - c2) \qquad 1.2$$

Similarly by taking n value we are able to derive the relation between number of characters and their execution time as specified by the above code segment. The derived dynamic metric entitled "Dynamic Execution Character Time" i.e. DECT is as follow:

$$T(C) = \sum_{i=1}^{3}(Ti \prod_{j=i\ \&\ j<>i}^{3} (C - Cj)/(Ci - Cj)$$

3. CONCLUSION:

Metrics can identify potential areas of problems that may lead to problems or errors. Finding these areas in the phase they are developed decreases the cost and avoids major ripple effects from the changes, later in the development life cycle. From the above calculation we come to know that the dynamic metric for character printing remain same independent of technique i.e. the time complexity is independent of iterative, functional and recursive programming approach. Though the static metric that gives us the measure of LOC which is different for different programming technique (iteration, function and recursion). The static metric are simple measure of some statistics can sometimes leads to vague result, on the other hand the dynamic metrics are more efficient and accurate than static metric because they are based on the running environment.

4. ACKNOWLEDGMENT

Authors are highly thankful to Dr. Gurvinder Singh, Associate Professor, DCSE, Guru Nanak Dev University Amritsar for his valuable guidance and precious time. Authors are also thankful to all the experts of the field of software engineering those have given contribution in the field of software metrics. Authors are also highly thankful and indebted to their family for giving a support for such tasks. At last but not least authors are indebted to the almighty.

5. REFERENCES

[1] H F Li, W K Cheung "An Empirical Study of Software Metrics" Software Engineering IEEE Transactions on (1987) Volume: SE-13, Issue: 6, Pages: 697-708

[2] N E Fenton "Software Metrics" Conference Proceedings of on the future of Software engineering ICSE 00(2000) Volume: 8, Issue: 2, Publisher: ACM Press

[3] Kuljit Kaur Chahal , Hardeep Singh "Metrics to study symptoms of bad software designs" ACM SIGSOFT Software Engineering Notes (2009) Volume: 34, Issue: 1, Pages: 1

[4] 12 Steps to Useful Software Metrics by Linda Westfall, [online] *www.westfallteam.com/Papers/12_steps_paper.pdf*

[5] Manik Sharma , Gurdev Singh "Static and Dynamic metrics- A Comparative Analysis", Emerging Trends in Computing and Information Technology 2011.

[6] Tu Honglei, Sun wei, Zhang Yanan, "The Research of Software metric and software complexity metrics" International Forum on Computer Science Technology and Applications (2009) Publisher: IEEE, Pages: 131-136

[7] Somerville "Software Engineering" 6th Edition, Editor: Addison Wesley.

[8] Li, H.F., Cheung, W.K. “An Experimental investigation of software metric and their relationship to software development effort”, IEEE Transaction on software engineering 649-653, Piscataway, NJ, USA.

[9] Thomas J McCabe, “A Complexity Measure”, IEEE Transaction on Software Engineering, Vol. SE-2 No. 4 [308-320]

[10] Van Doren “Cyclometic Complexity” [Online] web publication access in: http://www.sei.cmu.edu/str/decriptions/cyclometic_body.html

[11] Geoffery K. Gill, Chris F. Kemerer, “Cyclomatic Complexity Metrics Reivisted: An empirical Study of Software Development and Maintenance” Center for Information System research.

[12] Norman Fenton and Martin Neil “Software Metrics and Risk” proceeding of FESMA 99 2nd European Software Measurement Conference.

[13] Gurdev Singh, Dilbag Singh et. al “A Study of Software Metrics” International Journal of Computational Engineering and Management. vol. 11. 2230-7893.

[14] Kamaljit Kaur, Kirti Minhas et. al “Static and Dynamic Complexity Analysis of Software Metrics”, World Academy of Science, Engineering and Technology 56 2009