ISSN No. 0976-5697

Volume 4, No. 8, May-June 2013

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Software Code Complexity Framework; Based on an Empirical Analysis of Software Cognitive Complexity Metrics using an Improved Merged Weighted Complexity Measure

Dr Wafula Joseph
Director ICT
JKUAT – Juja, Kenya.

Dr. Waweru Mwangi
Director - ICSIT
JKUAT - Juja, Kenya.

Stephen N. Waweru*
Student - Masters in Software Engineering -
Institute of Computer Science & Information Technology (ICSIT)
JKUAT – Juja, Kenya
stiiv2005@yahoo.com

Abstract*:* This research paper proposes a Software Complexity Code Framework Based on an empirical analysis of Software Cognitive Complexity Metrics using an Improved Merged Weighted Complexity Measure. Software Development Industry in Kenya is dominated by a myriad of Small Software Developers firms. It was observed that majority of the Small Software Developers Organizations have 2 - 20 employees indexed by 62.4%, whereas Large Software Developers Organizations index 30.4% [1]. The increased complexity of modern software applications also increases the difficulty of making the code reliable and maintainable. This research paper measures one internal measure of software products, namely software complexity. I develop a Software Code Complexity Framework using a proposed cognitive complexity metric for evaluating design of object-oriented (OO) code. The proposed metric is based on important features of the Object Oriented Systems: Inheritance, Control Structures, Nesting and Size. The proposed metric is applied on a real project for empirical validation and compared with Chidamber and Kemerer (CK) metrics suite [2]. The practical and empirical validations and the comparative study prove the robustness of the measure. The outcome of this Model leads to a development of Software Code Complexity Framework; a tool-set for static analysis of Java/C/C++ source code: a combination of automatic code review and automatic coding standards enforcement.

Keywords: Software Code complexity, Metric, Framework, Object Oriented System, Cognitive Complexity

I. INTRODUCTION

Object Oriented scheme have come to control software engineering over the last two decades. The improvement and modification in these techniques are still undergoing research [1]. More and more organizations are adopting these techniques into their software development practices. A result of the growth in popularity of object oriented programming is the introduction of number of software design metrics. Although most of these metrics are applicable to all programming languages, some metrics apply to a specific set of programming languages [2], among these metrics some have been proposed based on cognitive complexity called Cognitive Complexity Metrics. Wang [3] observed that the traditional measurements cannot actually reflect the real complexity of software systems in software design, representation, cognition, comprehension, and maintenance. Today, the relevant literature provides a variety of object oriented metrics [4-5], to compute the complexity of software.

The metrics presented in this survey are for Object–oriented programming. Still there are other Object Oriented Cognitive Complexity Metrics specially developed for Object Oriented programs. Cognitive complexity measures are the human effort needed to perform a task difficulty in understanding the software. The cognitive complexity is an ideal measure of software functional complexities and sizes, because it represents the real semantic complexity by integrating both the operational and architectural complexities. In this research paper, an attempt has been made to develop a very simple Framework for calculating the complexity of code in terms of cognitive weights. This method is the most suitable due to not only its simplness but also it provides the complete information about the information contents of a Program code. There is continuous effort to find a comprehensive complexity metric, which addresses most of the parameters of software. The outcome of this research paper leads to automation of the ***Merged Weighted Complexity Metric*** including other complexity metrics by developing a; *Software Code Complexity Framework; a tool-set for Static Analysis of Object Oriented Systems Source Code. a* combination of Automatic Code Review and Automatic Coding Standards Enforcement.

II. SOFTWARE COMPLEXITY

Bill Curtis [6] has reported two types of software complexity; (i) Psychological complexity affects the performance of programmers trying to comprehend or modify a class/module. (ii) Algorithmic or computational complexity characterizes the run-time performance of an algorithm. Brooks [7] states that the complexity of software is an essential attribute, not an accidental one. Essential complexity arises from the nature of the problem and how

deep a skill set is needed to understand a problem. Accidental complexity is the result of poor attempts to solve the problem and may be equivalent to what some are calling complication. Implementing wrong design or selecting an inappropriate data structure adds accidental complexity to a problem. Software complexity has been defined differently by many researchers. Zuse [8] defines software complexity as the difficulty to maintain, change and understand software.

It deals with the psychological complexity of programs. According to Henderson-Sellers [9] the cognitive complexity of software refers to those characteristics of software that affect the level of resources used by a person performing a given task on it. Basili [10] defines software complexity as a measure of the resources expended by a system while interacting with a piece of software to perform a given task. Software complexity cannot be defined by a single definition because it is multidimensional attribute of software. So, different researchers/users have different view on software complexity. Therefore, no standard definition exits for the same in literature. However, knowledge about software complexity is useful in many ways. It is indicator of development, testing, maintenance etc. efforts, defect rate, fault prone modules and reliability. Complex software/module is difficult to develop, test, debug, maintain and has higher fault rate. Complexity [11] is defined as "the degree to which a system or component has a design or implementation that is difficult to understand and verify" *i.e.* complexity of a code is directly dependent on the understandability. All the factors that makes program difficult to understand are responsible for complexity.

Figure 1 shows the quality attributes, sub attributes and metrics that defines software maintainability. Complexity forms a segment of sub-attributes, which can be measured by various complexity metrics i.e. nesting path, depth of inheritance and size. The International Standard ISO/IEC 9126 has been taken as a baseline.

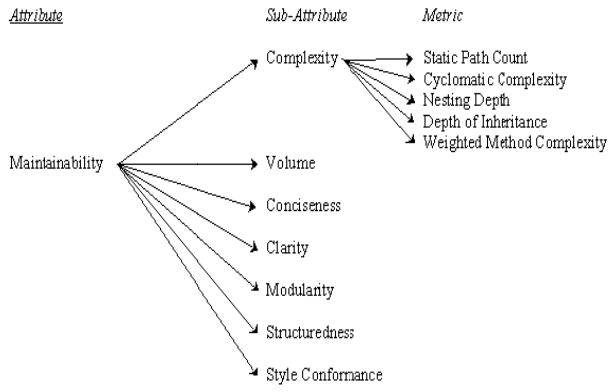

Figure 1: Maintainability Attributes, Sub-attributes and Metrics [6]

III. RESEARCH CONCEPT

A. Problem Statement:

There are too many popular and simple tools for measuring and analyzing software code that do not include the most important complexity factors. As already noted before, various old tools for measuring and analyzing software code complexity are under several criticisms. Most of the available tools do not consider the *Cognitive Characteristics* in measuring and analyzing the complexity of a code, which directly affects the *Cognitive Complexity.* Complexity of a code directly affects comprehension. The understanding of a code is known as program comprehension and is a cognitive process..The *Cognitive Complexity* is defined as the mental burden on the user who deals with the code, for instance, the developers, the testers and the maintenance staff [6].

B. Research Motivation:

The aim of this survey is to list out some of the existing Cognitive Complexity Object Oriented Metrics and to make the small software developers aware of their existence. The survey will also allow small software developers to use Software Code Complexity Framework - a tool for measuring and analyzing software code complexity and empower them to predict rate of errors, predicts maintenance effort, scheduling and reporting projects, measure overall quality of their programs, measure the minimum effort and best areas of concentration during testing and guide the testing process by limiting the program logic during development because the framework do not require in-depth analysis of programming structure, its simple to use, easy to apply and can be used for any Object Oriented programming language i.e. C/C++ and Java.

C. Reserch Survey:

The main purpose of this research survey was to find out the impact of software complexity in software design Object Oriented and to externally validate design complexity by investigating its statistical relationship to external software quality. To ascertain this, information from a group of small software developers companies was needed. This was a typical example of research in the large, which means that a survey was appropriate to use. The survey was conducted in order to identify possible complexity metrics that can be used as indicator of external quality attributes and what they think is most important for their company in terms of processes/activities. The approach used for this study for collecting information was a quantitative approach. The reason for this was that the questions used in the survey were categorized and the answers were divided into three groups using a scale from one to three. Having categorized questions and answers is typical for a quantitative collection approach. Another issue that affected the choice of approach was that the result was to be analyzed and statistically investigated in order to find relationships between the questions. To achieve this task a questionnaire was designed for Small Software Developers.

In Table 1 and figure 2 below, this research survey find out that a small number (4.8%) of small software developers use Metrics as a tool for measurement and anlaysis. Infact majority (95.8%) have no idea of what metrics are, they have no indication of what metrics are, their functions and how they work.Very few developers agreed to have come across metrics but they ignore their importance and functinalities.

Table 1: Respondents who use Software Design Complexity Metrics to measure and analyze their Software codes

	equen	ercen	id Perc	umulati Percent
Val YE	1	4.2	4.2	4.2
NO	23	95.8	95.8	100.0
Tot	24	100.0	100.0	

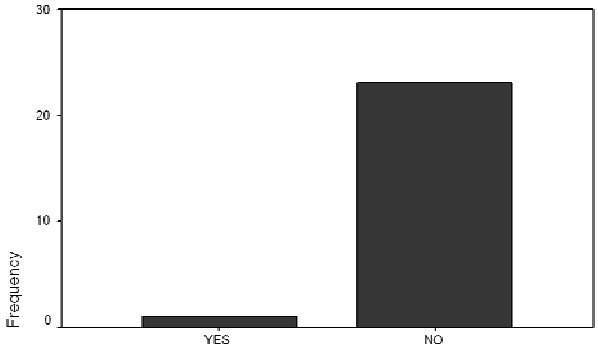

Figure 2: Plot of Respondents who use Design Complexity Metrics to measure and analyze their Software codes

In table 2 and figure 3 below the survey finds out that a small number 12.5% of small software developers do not agree there is an overall influence on complexity on external quality attributes. The remaining 87.5% do not agree there is any influence; in fact they see software quality as an outcome of coding without errors (bugs) and including better Graphical User Interfaces.

Table 2: Respondents who agree there is an overall influence of complexity on External Quality Attributes

	querce	Peerce	nula
V: YI	3 2.5	2.5	2.5
N	21 7.5	7.5	0.0
T	24 0.0	0.0	

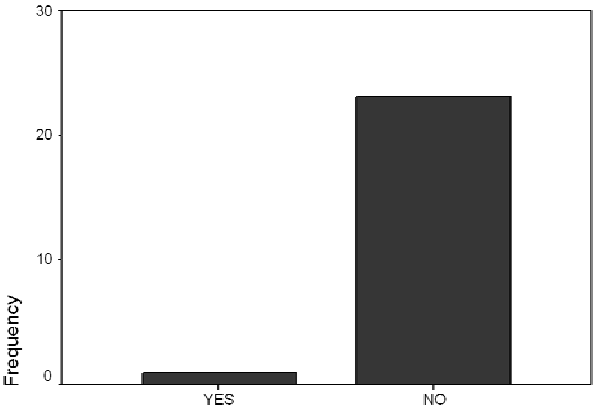

Figure 3: A plot of Respondents who agree there is an overall influence of complexity metrics on External Quality Attributes

The survey also finds out that 4.2% of respondents connect the impact of code complexity to software Quality. 95.8% of respondents do not agree on the association of software code complexity with software quality. They consider a good code should always give good software; the idea of software quality is the last consideration during their software design. Table 3 and Figure 4 below illustrate the association of software code complexity with software quality.

Table 3: Respondents who connect the Impact of Code Complexity to Software Quality

	querce	Peerce	nula
V: YI	1 4.2	4.2	4.2
N	23 5.8	5.8	0.0
T	24 0.0	0.0	

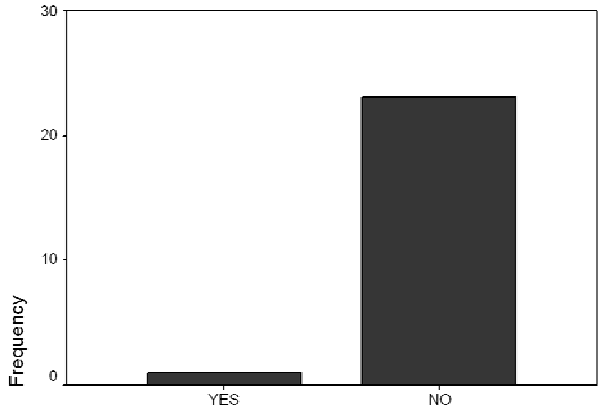

Figure 4: A plot of Respondents who connect the Impact of Code Complexity to Software Quality

IV. LITERATURE REVIEW

There are many well known software complexity measures that havee been proposed such as McCabe's cyclomatic number [12], Halstead programming effort [13], Oviedo's data flow complexity measures [14], Basili's measure [15],Wang's cognitive complexity measure[16] and Knot complexity [17]. All the reported complexity measures are supposed to cover the correctness, effectiveness and clarity of software and to provide good estimate of these parameters. Out of the numerous proposed measures, selecting a particular complexity measure is again a problem, as every measure has its own advantages and disadvantages.

V. COGNITIVE WEIGHTS AND INFORMATICS

The field of cognitive informatics, it is found that the functional complexity of software in design and comprehension is dependent on internal architecture of the software. Basic control structures (BCS), sequence, branch and iteration [18] is the basic logic building blocks of any software. The cognitive weight of software is the extent of difficulty or relative time and effort for comprehending given software modeled by a number of BCS's. There are two different architectures for calculating W_{bcs}: either all the BCS's are in a linear layout or some BCS's are embedded in others. For the former case, sum of the weights of all n BCS's; are added and for the latter, cognitive weights of inner BCS's are multiplied with the weights of external

BCS's. The cognitive weights for Basic Control Structures are as under:

Table 5: Basic control structures and their cognitive Weight

Category	BCS	Weight
Sequence	Sequence (SEQ)	1
Branch	If-Then-Else (ITE)	2
	Case	3
Iteration	For-do	3
	Repeat-until	3
	While-do	3
Embedded Component	Function Call (FC)	2
	Recursion (REC)	3
Concurrency	Parallel (REC)	4
	Interrupt (INT)	4

VI. THE ANALYSIS AND FINDINGS OF THE EXISTING COMPLEXITY MEASURES BASED ON COGNITIVE INFORMATICS

According to Wang [16] the major problems yet to be solved in Cognitive Informatics are: the architectures of the brain, mechanisms of the natural intelligence, cognitive processes, mental phenomena and personality. It is interesting in computing and software engineering arena to explain the mechanisms and processes of memory, learning and thinking. It is expected that any breakthrough in Cognitive Informatics will profoundly pave the way to the development of the next generation technologies in informatics, computing, software, and cognitive sciences. There are a number of metrics which were developed based on Cognitive Informatics among them; (i) Weighted Class Complexity Metric (ii) Class Complexity (iii) Extended Weighted Class Complexity (iv) Class complexity Due to Inheritance and (v) Average Complexity due inheritance.

This research paper analyses (i) Weighted Class Complexity (WCC) Metric, it gives an analyses of the metric description, its equation, research findings and its limitation in comparison to the proposed Merged Weighted Complexity Measure.

VII. WEIGHTED CLASS COMPLEXITY (WCC) [20]

A. Computation Of Weighted Class Complexity [20]:

Mishra [20] ***modified*** the CC metric and proposed a new metric called weighted class complexity (WCC). It considered *Object Orientation* as a form of expression relation between the *Data and Function*, the class can be assumed as a set of data and set of method accessing them. The complexity of the class should be measured by complexity of ***Methods and Attributes.***

B. Weighted Class Complexity Equation:

Weighted Class Complexity (WCC) [20] can be calculated using the equation (1)

$$WCC = N_a + \sum_{p=1}^{s} MC_p \qquad (1)$$

Where,

Na is the Number of Attribute

MC is the Method Complexity, which is calculated by equation (2)

If there are *y* classes in an object oriented code, then the total complexity of the code is given by the sum of weights of individual classes.

$$\text{Total Weighted Class Complexity} = \sum_{x=1}^{y} WCC_x \qquad (2)$$

C. Research Findings

In his proposed measure, the complexity of a class was the sum of complexity of the operation in methods, Complexity Due To Data Members (Attributes) and Complexity Due To Message Call. Further, complexity of method is calculated by Complexity of the Code of Operation in Method and as well as on the Number of Attributes in the Method.

D. Weighted Class Complexity Limitations:

Weighted Class Complexity includes the complexity due to the internal structure of methods and attributes. By using, Weyuker's [19] properties Weighted Class Complexity has been validated and found that it satisfies six properties out of nine, which established this measure as well, structured one. Weighted Class Complexity can be used to calculate the complexity of Object Oriented code with different size. High Weighted Class Complexity value indicates that understandability and maintainability of the code is difficult. Ultimately, it helps the software developer for better design information. However a better Object Oriented metric should not only consider the internal structure of method and the number of attributes in a class but it should also consider the concepts of OOP like *Inheritance, Encapsulation, Overloading and Polymorphism.*

VIII. THE MERGED WEIGHTED COMPLEXITY MEASURE (C_w) METRIC

Software complexity can not be computed by a single parameter of program/software because it is multidimensional attribute of software. The prominent factors which contribute to complexity of a program/software are:

A. Inheritance Level of Statements in Classes:

A statement which is at deeper level of inheritance of classes is harder to understand and thus contributes more complexity than otherwise. We take effect of inheritance level of classes by assigning weight 0 to statements at level one (the outermost level/class) i.e in the base class, weight 1 for those statements which are at level 2 i.e first derived class, weight 2 to those statements which are in next derived class (level 3) and so on.

B. Types of Control Structures:

A program/class with more control structures is considered to be more complex and vice-versa but, we assume that different control structures contribute to the complexity of a program/class differently. For example,

iterative control structures like for loop, while .. do, do .. while contribute more complexity than decision making control structures like if .. then .. else. Therefore, we assign different weights to different control structures.

C. Nesting of Control Structures:

A statement which is at the deeper most level of nesting (the inner most level) of control structures is harder to understand and thus contributes more complexity than otherwise. We also take effect of nesting of control structures by assigning weight 1 to statements at level one i.e. the outer most level, weight 2 for those statements which are at level 2 i.e. the next inner level of nesting and so on. The weight for sequential statements is taken as zero.

D. Size:

Size is also considered one of the parameter of program/class complexity. A class with more methods is harder to understand than a class with less number of methods and hence contributes more complexity [23]. Large programs incur problem just by virtue of volume of information that must be absorbed to understand the program and more resources have to be used in their maintenance. So, size is a factor which adds complexity to a program/class.

By taking these factors into account, a weighted complexity measure for an object-oriented program P was suggested as:

$$C_w = \sum_{k=1}^{n} E_k * (W_t)_k \quad \text{..... Equation 1}$$

Where;

C_w - Improved Merged Weighted Complexity Measure [IMWCM] of Program P,

n - Total Number of Executable Statements in Program P,

Σ - Summation Symbol,

* - Multiplication Symbol,

k - Index Variable,

$(W_t)_k$ - Total Weight of **kth** Executable Statements in Program P,

ΣE_k - Summation of **kth** Executable Statements in terms of {Operators, Operands, Methods, Strings & Functions}.

$$\sum E_k = E_o + E_s + E_m + E_s + E_f \quad \text{..... (1)}$$

Where;

E_o - Number of Operators in an Executable Statement [P]
E_s - Number of Operands in an Executable Statement [P]
E_m - Number of Methods in an Executable Statement [P]
E_s - Number of Strings in an Executable Statement [P]
E_f - Number of Functions in an Executable Statement [P]

$$W_t = W_n + W_i + W_c \quad \text{..... (2)}$$

Where;

W_n = weight due to *Nesting Level of Control Structures* where it is;
= 0 for sequential statements,
= 1 for statements inside the outer most level of control structures,
= 2 for statements inside the next inner level of control structures and so on.

W_i = weight due to *Inheritance Level of Statements* in classes where it is;
= 0 for statements inside the outer most level of inheritance i.e. inside base class,
= 1 for statements inside the next level of inheritance i.e. first derived class,
= 2 for statements inside the next deeper level of inheritance i.e. next derived class
and so on.

W_c = weight due to *Types of Control Structures* where it is;
= 0 for sequential statements,
= 1 for decision making control statements like if .. then .. else,
= 2 for decision making control statements like while .. do, for loop, do .. while,
= ***n*** for switch statement with ***n*** cases.

IX. COMPARATIVE STUDY OF THE WEIGHTED COMPLEXITY MEASURE METRIC

In this section, the research paper adapts a 'C' program for analysis of the result then calculates the *Cognitive Weight Complexity Measure (CWCM)* for the program codes. Finally compares *Cognitive Weight Complexity Measure* with *Cognitive Functional Size*. The value of Cognitive weight complexity measure and cognitive functional size are given in the following respective tables below..

Table 6: Calculation of Metric Cw for Prog 1- *[(S)j*]

Statement Number	Executable Statement	(S)j
s1	**void Abc::input()**	**4**
s2	**count<<"enter value"**	**3**
s3	**cin>>a>>b>>c**	**7**
s4	**void Abc::output()**	**4**
s5	**count<<"A="<<a**	**5**
s6	**count<<"B="<<b**	**5**
s7	**count<<"C="<<c**	**5**

Table 7: Calculation of Metric Cw for Prog 1 – *[(Wt)j]*

Statement Number	Executable Statement	Wa	Wi	Wc	(Wt)j
s1	void Abc::input()	0	1	0	1
s2	count<<"enter value"	0	1	0	1
s3	cin>>a>>b>>c	0	1	0	1
s4	void Abc::output()	0	1	0	1
s5	count<<"A="<<a	0	1	0	1

s6	count<<"B="<<b	0	1	0	1
s7	count<<"C="<<c	0	1	0	1

Table 8: Calculation of Metric Cw for Program 1

Statement Number	Executable Statement	(S)j	(Wt)j	Cw=(S)j*(Wt)j	Cw
s1	void Abc::input()	4	1	4*1=	4
s2	count<<"enter value"	3	1	3*1=	3
s3	cin>>a>>b>>c	7	1	7*1=	7
s4	void Abc::output()	4	1	4*1=	4
s5	count<<"A="<<a	5	1	5*1=	5
s6	count<<"B="<<b	5	1	5*1=	5
s7	count<<"C="<<c	5	1	5*1=	5

X. WEIGHTED COMPLEXITY MEASURE (C_W) COMPUTATION

n = 7 (statements) therefore; C_w (P) = 33

XI. WEIGHTED COMPLEXITY MEASURE (C_W) ANALYSIS

The table below shows that a high complexity value was achieved when I used MWCM compared to others which give very low values. This is an indication that MWCM provides a dependable and reliable value that provides complete complexity details of a program code.

Therefore the proposed *Software Code Complexity Framework* based on MWCM will be very accurate and will provide software developers with the actual, realistic and detailed reports for determining their software codes complexity.

Table 9: Computation Analysis Table

Metric Tested	Experimentation for Program 1
WMC	**2.5**
WOC	**3.5**
DIT	**4.5**
CBO	**5.4**
MWCM	**6.2**

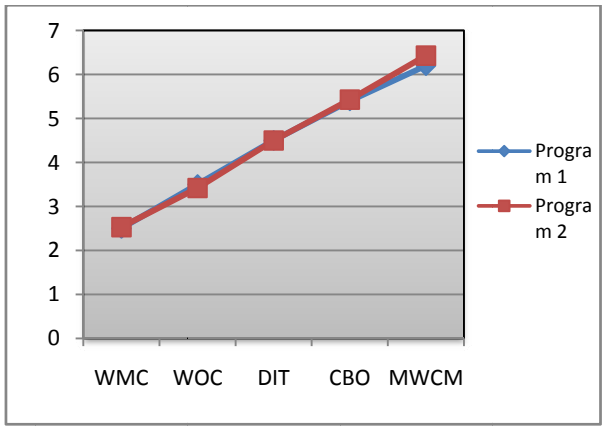

Figure 5: Computation Analysis Chart

XII. THIS RESEARCH PAPER FINDINGS

a. Thus, we infer that, the proposed Merged Weighted Complexity Measure detects complexity and gives realistic estimates.
b. This is because the Merged Weighted Complexity Measure also takes into account the complexity due to factors not considered earlier Metrics suite, LOC and McCabe's measure.
c. These factors are size, types of control structures, and their nesting level.
d. As a software system is developed in terms of classes/programs, so the proposed measure can be used to calculate and compare the complexities of classes/programs and hence of *Object-oriented Software Systems*

XIII. COMPARATIVE INSPECTION

The comparative inspection of the implementation of *Merged Weighted Complexity Measure* (C_w) versus eLOC, CC [McCabe's] , CK metrics and Halstead has shown that:

a. *Merged Weighted Complexity Measure* (C_w) shall make more sensitive measurement, so that it will enable developers to differentiate even small complexity differences between codes.
b. Halstead's assumptions may sometimes mislead developers, whereas *Merged Weighted Complexity Measure* (C_w) has the least amount of assumptions and those assumptions are based on cognitive aspects.
c. CC was not able to make sensitive measurement; most of the similar codes had the same CC values.

XIV. ANALYSIS AND FINDINGS OF THE EXISTING TOOLS FOR SOFTWARE CODE COMPLEXITY ANALYSIS AND MEASUREMENT [46]

There exist many tools for code analysis and measurement in the market, for example QA-C and Testwell CMT++. Some of them are freeware and open-source while others are proprietary tools. They differ a great deal in their features, support for languages, support for platforms, licensing prices and other aspects. This research paper discus and analyses Understand.

XV. UNDERSTAND TOOL

Understand is a commercially available static analysis tool for maintaining, measuring and analyzing source code. This tool is developed by Scientific Toolworks Inc. (SciTools). This tool is thoroughly evaluated to see up to which extent this tool meets with the specific requirements for this project.

A. Language Support:

This tool can analyze 14 languages including C/C++, Java, FORTRAN and some web programming languages like PHP. OS support. It is available for all major operating systems including Solaris.

B. Metrics:

It can gather large number of metrics including many basic metrics, some advance metrics and some custom metrics. During the evaluation of this tool it is found that some of the commonly used metrics aren't included in this tool, for example is it missing all Halstead metrics. It is also observed that all metrics aren't available for all the languages the tool can analyzed. However it can gather about 74 metrics and most of the metrics are available for C/C++.

C. Features:

It is a GUI tool for all operating systems this tool supports. It outputs reports in graphical, textual and HTML format. It comes with a programming editor. It has a code check feature that checks the code for coding standards. The coding standards it has for code check are Effective C++ (3rd Edition), MISRA-C 2004 and MISRA-C++ 2008. A very nice feature of this tool is the plug-in support which allows users to define and add new metrics to the functionality.

D. Critical Analysis:

This tool is more useful for the developers than the management. It has features like code check and code editor that aren't generally needed for the management. Although this tool can gather large variety of metrics but it doesn't provide any recommended maximum and minimum values for the metrics. The feature of comparison among various releases of software is also not a part of this tool. Project is created manually and since it has the GUI interface it is maybe hard to automate it. Manually created projects with an in-built feature of code editing allow for unwanted changes to the code and lead to the problems of code security and safety. It has no in-built support for integration with any code repository; therefore code has to be available locally before the analysis can perform. During the analysis on actual code it produced some strange results. Value of 4576376.0000 is calculated for a metric “CountPath” when this tool was analyzing a package "Test". This value is way too strange and with no recommended maximum or minimum guidance, it even becomes more confusing.

XVI. EVALUATION SUMMARY OF ALL TOOLS

The tools evaluation proved that functionality wise the tools differ greatly. Some of the tools try to do too much, from complexity measurements to coding standard checks, from function level to project level, from bug detection to bug prevention and some provide code editing facility as well. On the other side, some tools keep their focus on complexity measurements only with either small or large number of metrics. Language support also varies among tools. Some tools support multiple languages at the same time while others are one language specific tools. Some tools work as standalone but some has integration features with IDE's. Visual Studio has its own complexity measurement feature in it. LOC or its variants and McCabe Cyclomatic complexity are the most common metrics for every tool. The way of creating code analysis reports is also different in most tools. Most of the tools are designed to help software developers, especially those tools that can be integrated with IDE's and gather large number of metrics.

XVII. PROPOSED SOFTWARE CODE COMPLEXITY FRAMEWORK TOOL BASED ON A MERGED WEIGHTED COMPLEXITY MEASURE METRIC

Among all the tools analyzed in this research paper there were no tools that adapted or used Cognitive Informatics metrics. Therefore this shows that there is a big gap and a great need for a new *Software Code Complexity Analysis And Measurement Tool* that adapts or is based on Cognitive Weights and Informatics for greater comprehensibility of software codes and entire complete programs at large out there in the market.

Software Code Complexity Framework will provide a way to check your code using;

a. The Improved Metric [*Merged Weighted Complexity Measure C_w]*
b. Published Coding Standards
c. And Custom Standards.

These Checks can be used to verify *Naming Guidelines, Metric Requirements, Published Best Practices,* or any other *Rules or Conventions* that are important. Complexity Measurement and Analysis is a process where key figures are derived from existing source code. These key figures then serve as indicators to judge a source code’s quality. Obtaining key figures is accomplished by certain arithmetic’s directly applied to the source code. In General, arithmetics that are applied in complexity measurements are called *Metrics*. They make up a set of formulae that is provided by code metrics tools. **Software Code Complexity Framework [Software Code Complexity Framework]** is tool dedicated to measure the complexity of source code written in C, C++ .

A. Published Coding Standards [40]:

You will find many "checks" straight from the following published standards:

i. Effective C++ (3rd Edition) Scott Meyers
ii. MISRA-C 2004
iii. MISRA-C++ 2008

B. Software Code Complexity Framework Metrics:

Software Code Complexity Framework is very efficient at collecting metrics about the code it analyzes. These metrics can be extracted automatically via command line calls, viewed graphically, dynamically explored in the GUI, or customized via the Software Code Complexity Framework Java - NetBeans API. They can also be reported at the Project Level, for Files, Classes, Functions or User Defined Architectures. Most of the metrics in Software Code Complexity Framework - can be categorized in the following;

i. Improved Metric *[Merged Weighted Complexity Metric C_w]*
ii. Complexity Metrics (e.g. McCabe Cyclomatic)
iii. Volume Metrics (e.g Lines of Code)

iv. Object Oriented (e.g. Coupling Between Object Classes)

C. Software Code Complexity Framework Metrics Api's:

There are other metrics APIs provided for in Software Code Complexity Framework for Complexity Analyses, but due to space I have discussed just three.

a. Average Number of Blank Lines (Include Inactive):

API Name :AltAvgLineBlank
Description :Average number of blank lines for all nested functions or methods, including inactive regions.
Metric can be used for
C/C++ : Project, File, Class, Struct, Union
Java : Project, File, Class, Interface

a) Applicable Formula:

SUM (AltCountLineBlank of each function in scope) / # of functions

b. Number of Immediate Base Classes:

API Name :CountClassBase
Description :Number Of Immediate Base Classes.

a) Metric can be used for:

C/C++ : Class,Struct,Union
Java : Class,Interface

For example if we are provided with the program codes below, the CountClassBase metrics computes as follows;

```
Class BoatCar private Car public Boat protected DualPurpose{
public:
//Public Instance Function
BoatCar():Car(4),Boat(),minWater(false),mColor("Blue"){}
virtual int passengers()const{return 4;}
static int numRegistered(){return sRegistered;}
bool minWater//Public Instance Variable
protected:
void toggleinWater(bool inWater){minWater=inWater;}
char*mColor;//Protected instance Variable
friend void init(){}
static int sRegistered;
static double calcSpeed(double distance,double time){
return distance/time;
}
private:
int mMaxPassengers;
        void travel(){}
};
```

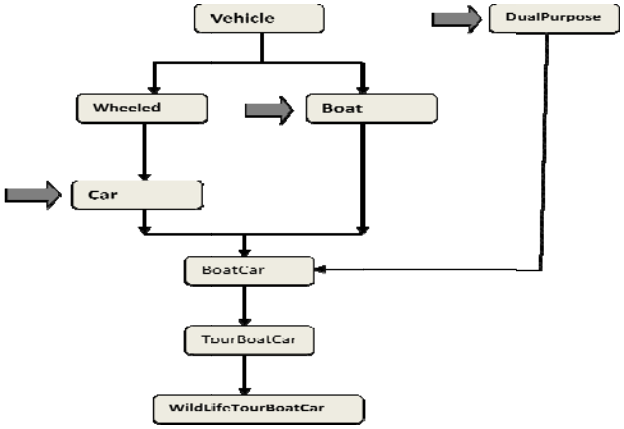

Figure 4: A Class Vehicle Flowchart Diagram

Results for class BoatCar = 3 classes

c. Depth of Inheritance Tree:

API Name : MaxInheritanceTree
Description : The depth of a class within the inheritance hierarchy is the maximum number of nodes from the class node to the root of the inheritance tree. The root node has a DIT of 0. The deeper within the hierarchy, the more methods the class can inherit, increasing its complexity.

a) Metric can be used for:

C/C++ : Project,Class,Struct,Union
Java : Project,File,Class,Interface,Method

MaxInheritanceTree Software Code Complexity Framework Computation:

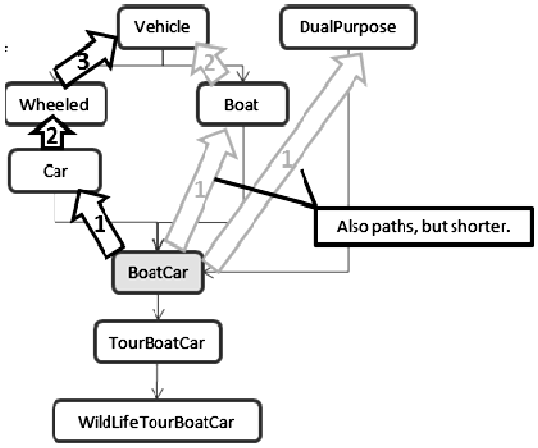

Figure 7: Showing Depth of Inheritance Tree

Table 12: Other metrics provided in Software Code Complexity Framework for Complexity Analyses

McCabe's Cyclomatic Complexity		**Halstead**	
v(G)	Complexity of a programs control flow	**B**	Estimated number of bugs
Lines of Code		**D**	Difficulty level
LOCphy	Physical lines of source code	**E**	Effort to implement

LOCpro	Program lines of source code (comments)	**L**	Program Level
LOCcom	Program lines of source codes (program code}	**N**	Program Length
LOCbl	Blank lines of source code	**N1**	Number of Operators
Maintainability Indexes		**n**	Vocabulary size
MI	Maintainability index	**n1**	Number of unique Operators
MIwoc	Maintainability index without comments	**n2**	Number of Unique Operands
MIcw	Maintainability index with weighted comments	**T**	Size of implementation of an Algorithm (volume)

XVIII. SOFTWARE CODE COMPLEXITY FRAMEWORK FUNCTIONAL GRAPHICAL USER INTERFACE

A. Software Code Complexity Framework Dependency Browser:

The Dependency Analysis capability provides these features:

a. Rapid browsing of dependencies for files and *Software Code Complexity Framework* architectures
b. List "dependent", and "dependent on" entities, for files and architectures
c. Spreadsheet export of dependency relationships
d. A Dependency Browsing dock that shows all dependency information

B. Software Code Complexity Framework References:

To calculate dependency we examine every reference in a *Software Code Complexity Framework* project. We then build up dependency data structures for every file and architecture. This includes the nature of the dependency and the references that caused the dependency. All of this data is instantly available for quick exploration and browsing.

C. Software Code Complexity Framework Supported Languages:

The following list provides a brief overview of the language versions and/or compilers supported by ***[Software Code Complexity Framework]***:

a. **C/C++:** Software Code Complexity Framework analyzes K&R or ANSI C source code and most constructs of the C++ language. Software Code Complexity Framework works with any C compiler, and has been tested with most of the popular ones. Note that C++ templates are not yet supported.
b. **Java:** Software Code Complexity Framework supports JDK.

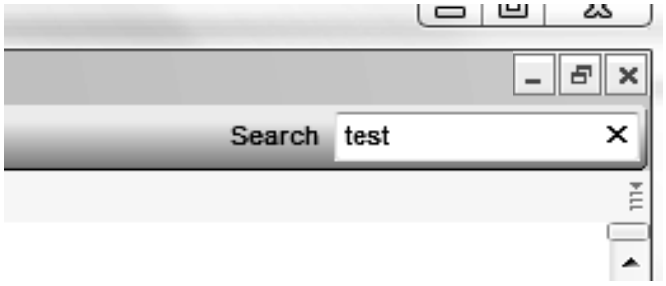

Figure 8: Showing Software Code Complexity Framework Instant Search Dialogue Box

D. Search Index:

By default Instant Search looks at all your code. For C++ and Java it breaks up the code following syntactic and lexical conventions of the language. For other languages it breaks it up using white space and punctuation.

So, for instance, in this line:

foreach (i=1,i<10,i++)

you could look for ***"foreach", i, 1***, and ***10***. These are called "search terms".

Additionally, Instant Search divides search terms up into fields. These fields are "***string***", ***"comment***", and ***"identifier***". By default it searches among all three fields, but you can limit searches to a particular field.

E. Software Code Complexity Framework Search Syntax:

The simplest is to type the term you are searching for, remembering that ALL searching is case sensitive: But the syntax is richer than that, permitting some relatively complicated queries.

Examples:

String: test	- searches for "text" only in "strings"
Identifier: test	- searches for identifiers named test
Comment: test	- searches for comments with "test" in them
test this	- matches anything test OR this

XIX. SOFTWARE CODE COMPLEXITY FRAMEWORK GENERATED REPORTS

Software Code Complexity Framework generates a wide variety of reports. The reports available in your project may vary based off of the project language, but the reports fall into these categories:

a. Cross-Reference reports show information similar to that in the Info Browser, except that all entities are shown together in alphabetic order.
b. Structure reports show the structure of the analyzed program.
c. Quality reports show areas where code might need to be examined.
d. Metrics reports show basic metrics such as the number of lines of code and comments.

A. Calculated Metrics For Contents I:

i. Total Files (For multiple file metrics)
ii. Total Lines
iii. Code Lines
iv. Comment Lines
v. Whitespace Lines
vi. Average Line Length
vii. Comment Lines/File (For multiple file metrics)
viii. Code/Comments Ratio
ix. Code/Whitespace Ratio
x. Code/(Comments + Whitespace) Ratio

B. Report [1] Calculated Metrics Content I:

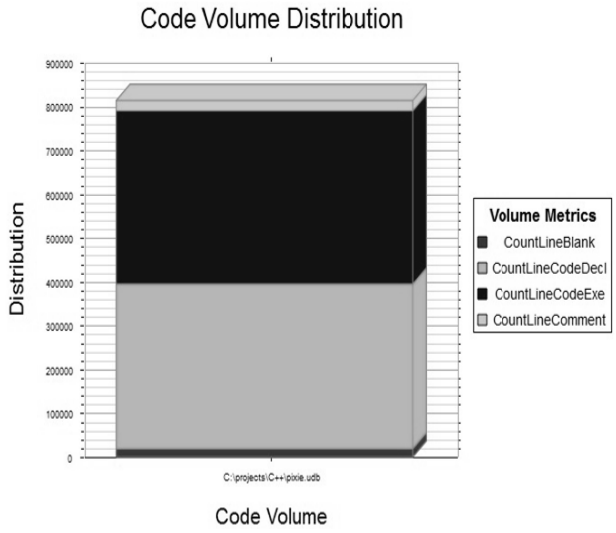

Figure 9: Software Code Complexity Framework Output Report 1 - screen shot

Report [2] For Calculated Metrics Content I

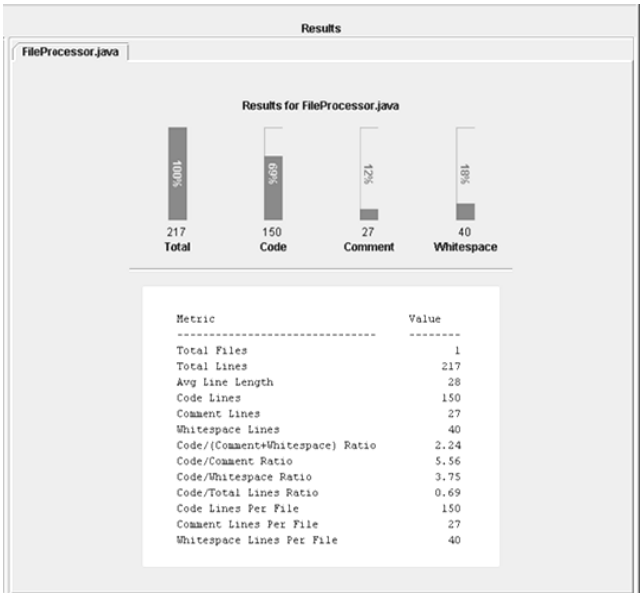

Figure10: Software Code Complexity Framework Output Report 2 - screen shot

XX. SOFTWARE CODE COMPLEXITY FRAMEWORK RICH GRAPHICAL USER INTERFACE

Software Code Complexity Framework provides a graphical user interface that enables managers, developers as well as quality and test engineers to collaborate easier on and to contribute easier to source code quality. It accelerates the way metrics are understood by an entire development team and not just by single individuals involved in improving the quality of software. Software Code Complexity Framework performs static source code inspection with one mouse-click, and provides various views that visualize the analysis results in a rich graphical user interface with hypertext links for ease of navigation. The overall statistics is presented in list and chart form with various search and filtering options. It is an embedded systems tool that performs standard code checking, automatically verifying source code compliance, and pointing out any code lines that breaks any of the coding standard rules. Software Code Complexity Framework [Software Code Complexity Framework] Provides checks in C/C++ and Java software projects.

A. Software Code Complexity Framework Scopes:

Software Code Complexity Framework strongly works with *Scopes*. The concept of a *Scope* implies that every metrics value that is calculated from source code has to be regarded relative to a context (*Scope*). This means, if *Software Code Complexity Framework* measures metrics values in a particular context (*Scope)*, there is no room for interpreting the values' meaning other than by its context (*Scope*) it was measured in. Measuring in particular *Scopes* is not only to avoid misinterpretation of metrics values, it is also done to cope with the several levels (*Scopes*) source code itself with its hierarchical structure implies.

B. Software Code Complexity Framework Main Menu:

The Main Menu of *Software Code Complexity Framework* is divided into four main areas. These four main areas are dedicated to accomplish different tasks each. Figure 8 shows these four main areas within the graphical user interface of *Software Code Complexity Framework* before they are described in more detail.

a. The *Main Menu* and the *Quick Menu,* for basic operations like open/save *Metrics Projects*, switch views or create reports.
b. The *Projects Explorer,* to manage items like the *Metrics Projects*, *Snapshots* or *Source Files*.
c. The view area for the two views *Dashboard View* and *Metrics View,* to examine the different s*copes* for metrics graphically or precisely.
d. The *Command History,* where commands of *Software Code Complexity Framework are logged.*

C. Software Code Complexity Framework Main Menu Option Group:

File > New > Metrics Project

Click *New* then click on *Metrics Project* or press *Ctrl + N* to set up a new *Metrics Project* that will contain all *Snapshots* of analyses results later on.

File > New > Snapshot

Click *New* then click on *Snapshot* or press *Ctrl + Alt + N* to set up a new *Snapshot* for an existing *Metrics Project*. This *Snapshot* will be marked bold within the *Projects Explorer* implicating that it is the latest in time which means that it holds the latest metrics values of the latest complexity analysis.

File > New > Report > PDF Report

Click *New* then click on *Report*, then click *PDF Report* to set up a new PDF report of the current *Main Snapshot* (latest, marked bold) of the current *Main Project*. A dialog

will show up allowing you to set further options to customize your report.

File > New > Report > XML Report

Click *New...* then click on *Report*, then click *XML Report* to set up a new XML report of the current *Main Snapshot* (latest, marked bold) of the current main project. This XML report is generated by *Software Code Complexity Framework*.

File > New > Report > CSV Report

Click *New* then click on *Report*, then click *CSV Report* to set up a new CSV report of the current *Main Snapshot* (latest, marked bold) of the current main project. This CSV report is generated by *Software Code Complexity Framework*. It allows you to export metrics data into spread sheet software like *Microsoft Excel*.

File > New > Report > Text Report

Click *New* then click on *Report*, then click *Text Report* to set up a new plain text report of the current *Main Snapshot* (latest, marked bold) of the current *Main Project*. This plain text report is generated by *Software Code Complexity Framework*. You can open it with any text editor.

File > Open...

Click *Open* to load an existing *Metrics Project* from disc.

File > Open Recent Solution

Click *Open Recent Project* to load a recently saved *Metrics Project*. You can choose from 10 recently saved *Metrics Projects* at total.

File > Save

Click *Save* to save the current *Metrics Project* to your predefined *Project Location* (also called *Working Directory*) or press *Ctrl + S*.

File > Save As

Click *Save As* to save the current *Metrics Project* to a desired path.

File > Exit

Click *Exit* to quit *Software Code Complexity Framework* or press *Alt + F4*.

D. Software Code Complexity Framework Quick Menu Icons:

The Quick Menu gives you a quick access to a set of frequently needed operations. Figure 9 shows this menu for a better understanding.

Figure 13: *Software Code Complexity Framew* Quick Menu

E. Software Code Complexity Framework Quick Menu Options:

- ***a. Open:*** Click *Open* to load an existing *Metrics Project* from disc.
- ***b. Save:*** Click *Save* to save the current *Metrics Project* to your predefined *Project Location*.
- ***c. Dashboard:*** Switch to the *Dashboard View* to examine metrics graphically.
- ***d. Metrics:*** Switch to the *Metrics View* to examine metrics precisely.

F. Software Code Complexity Framework Quick Menu Reports:

- ***a. PDF:*** Generate a PDF report from the current *Main Snapshot*.
- ***b. XML:*** Generate an XML report from the current *Main Snapshot*.
- ***c. CSV:*** Generate a CSV report from the current *Main Snapshot*.
- ***d. TXT:*** Generate a text report from the current *Main Snapshot*.

XXI. SOFTWARE CODE COMPLEXITY FRAMEWORK INSTALLATION

a. Through the Integration package, Software Code Complexity Framework becomes an integral part of the netbeans IDE.

b. The software code complexity framework functionality will be automatically available within the IDE at installation of the netbeans Integration for software code complexity framework.

c. The software code complexity framework functionality can be accessed in 2 ways i.e. through the software code complexity framework menu or through the software code complexity framework buttons. Both options are fully integrated in the netbeans IDE.

XXII. CONCLUSION AND FUTURE WORK

A. Future Work:

Further research is required to add more complexity factors and simplify the metric so that it becomes more practical. Although the study has tried to include most of those factors, it is possible to add more. After the research and empirical validation of the proposed metric, some deficiencies about the metric were also realized and they are as follows: Considering the complexity so broadly and detailed makes the measurement difficult to apply. Therefore, simplification may be one of the most essential needs of further research. Usually, the weight of cohesion gives a very low value, as if it is not of much importance. Factors of cohesion may be improved so that those numbers decrease the complexity value in a more significant amount. Comments and indentation may be important factors of cognitive complexity, but those factors were excluded from the scope of this thesis research. These deficiencies can be the source of further research. Although the improved metric has some weaknesses as noted above, because of its major advantages, it can be considered a valuable contribution to the literature in this field, since combining several cognitive aspects with functional aspect is a new attempt in this area.

XXIII. CONCLUSION

The outcome of the Framework developed from the improved merged weighted complexity metric shows that; The tool is capable of performing code analysis automatically on regular basis. It proves that the automatic measurement of source code complexity is possible to implement. The tool is helpful for developers to view the quality of their code in terms of code metrics. Potentially fault-prone code can easily be identified which can suggest developers about the code that requires refactoring. The change of code may also help the testers to focus their testing efforts on those parts of code that are changed. The automatic logging feature will allow for real-time monitoring of the tool. The configuration interface shall allow for simple addition/alteration of configuration specification. As complexity and code size grow for each year, so does the problem of releasing high-quality software. An Improved Weighted Composite Complexity Measure has been used which takes into account different aspects of complexity: *size, control structures, their nesting and inheritance level of statements in classes.* The research paper shows that the effect of these structures on complexity is quite significant. It is a robust method because it encompasses all major parameters and attributes that are required to find out complexity.

XXIV. REFERENCES

[1] R. Caves, Multinational Enterprise and Economic Analysis, Cambridge University Press, Cambridge, 1982.

[2] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 1951-1957, 1999.

[3] H.H. Crokell, "Specialization and International Competitiveness," in Managing the Multinational Subsidiary, H. Etemad and L. S, Sulude (eds.), Croom-Helm, London, 1986.

[4] K. Deb, S. Agrawal, A. Pratab, T. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithms for Multiobjective Optimization: NSGA II," KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[5] J. Geralds, "Sega Ends Production of Dreamcast," vnunet.com, para. 2, Jan. 31, 2001.The Mythical man Month: Essays on Software Engineering. Addison. Wesley, 2012.

[6] IEEE Std. 1061-1998 –IEEE Computer Society: Standard for Software Quality Metrics Methodology, 1998.

[7] Basili, V.: Qualitative Software Complexity Models:A summary, In tutorial on Models and Methods for Software Management and Engineering. IEEE Computer Society Press. Los Alamitos, CA 1980.

[8] Zuse, H. : Software Complexity Measures and Methods, W. de Gruyter, New York. 1991.

[9] Curtis, B.: Measurement and Experimentation in Software Engineering. Proc IEEE conference, 68.9. 1144-1157, September 1980.

[10] Sellers, B. H.: Object Oriented Metrics: Measures of Complexity, Prentice Hall, New Jersey, 1996.

[11] McCabe. T.H.(1976),A complexity measure. IEEE Transactions Software Engineering, (SE-2,6):308-320.

[12] Halstead. M.H.(1997), Elements of software science, Elsevier North-Holland,New York

[13] Oviedo, E.I. (1980). Control flow, data and program complexity. Proc. IEEE COMPSAC, Chicago, IL, pages 146-152.

[14] Basili. V.R.,(1980), Qualitative software complexity models: A summary in tutorial on models and methods for software management and engineering. IEEE Computer Society Press, Los Alamitos,CA.

[15] Wang. Y. and Shao J. (2003). A new measure of software complexity based on cognitive weights, Can.J.Elect. Comput. Eng.,28,2,69-74.

[16] Woodward, M. R., Hennel. M. A., David. H.,(1979) A measure of control flow complexity in program text. IEEE Transaction on Software Engineering, SE-5, Vol. 1, pages 45-50.

[17] Baker, A.L., and Zweben, S.H. (1980), A comparison of Measures of control flow Complexity, IEEE Transaction on Software Engineering, 6, 506-511.

[18] Weyuker. E.J.,(1988), Evaluating software complexity measure. IEEE Transaction on Software Complexity Measure, 14(9): 1357-1365.

[19] Misra S. and Misra. A. K.(2004), Evaluating Cognitive Complexity Measure with Weyuker's properties. Proceedings of third IEEE International Conference on Cognitive Informatics, 103-108.