A Comparative Analysis of DIT over MVG to Improve Quality of Software

Vikash Chauhan[1] and Dharmendra Lal Gupta[2]

(1) Dept. of Computer Science & Eng., Kamla Nehru Institute of Tech. Sultanpur U.P. INDIA
E-mail: vikashit56@gmail.com

(2) Dept. of Computer Science & Eng., Kamla Nehru Institute of Tech. Sultanpur U.P. INDIA
E-mail: dlgupta2002@gmail.com

ABSTRACT

To predict the quality of software, software metric is one of the very important elements. The relationship of Depth of Inheritance Tree metric with cyclomatic complexity is a significant matter. Here in this paper the relationship of DIT (Depth of Inheritance Tree) and MVG (McCabe's Cyclomatic Complexity) have been explained using three real projects developed in JAVA language. The authors have also empirically computed DIT and MVG metrics of these projects and found the correlations between these two. It is found that on increasing DIT, MVG also increases in polynomial form which is showing the directly proportional relationship. This paper is providing an optimal value of DIT up to that software will be quality software.

Keywords: Correlation of DIT and MVG, DIT, MVG

1- INTRODUCTION

Delivering high quality and highly reliable software is a mandatory goal for software development companies. Basically quality of software depends on its source code. If source code of any software is written in good way then its quality will increase. Thus in this way the amount of resources needed to support the software will reduce. So today's focus is on enhancing the software development process [1].

There is an critical need for presenting the of various statistics for evaluating the change-proneness for knowing what is wrong with our projects in progress so that there is no burn out between the stakeholder working in the life cycle of the application. Since, components of software are like organic compounds that change internally and externally with multiple environmental and business reasons. The major concern is maintenance and further development of the software without conflicts, issues and bugs. Changeability and upgradeability in software is very risky so to reduce this risk and to measure of impact of changes we have to maintain our source code.

Changeability is the ease with which a source code can be changed. It is evaluated through metrics calculated from the history of changes made.These metrics reflect how well or bad is the change for the project in terms of it degree however, the optimal of these metrics is matter of real concern as they

must be chosen in such a manner that they must measure the true image and state of the software components with respect to the basic principles of software stability with openness for further change [2].

The testing stage plays very important role to ensure the quality of the software products in software development life cycle model, but it is the most resource consuming step in terms of time, effort, and costs. This testing activity represents 50 to 70 percent of the total costs of a project [3]. To reduce these resources we have to maintain our source code and also maintain its metrics. These resources also depend on complexity of source code. If program is very complex, it will be difficult to understand by the programmer. Complexity plays a very important role to decide the effort, costs, and time of any project or software. Sometimes complex program can be understood easily by the computer or machine but it may be hard to understand by the programmer.

The main goal of this analysis study is to improve the quality of software. Improving the quality of this software implies a reduction in development and maintenance costs, hence a reduction in the costs of the whole company. Higher quality software might also increase customer satisfaction and assurance in the company and its products [4]. To carry out this improvement this study will focus on computing software metrics over the software's source code and relationship among these metrics. This correlation study of metrics will help us to measures the strength of relationships between two metrics, to regression analysis, which determines the mathematical expression of the relationship [5].

2- SOFTWARE QUALITY METRICS

Software metrics and software quality factors compose the software quality metrics. These metrics provide measures of the software attributes and may be in the form of checklists used to grade a document produced during the development [6].

Software Quality Metrics (SQM) = Software Metrics (SM) + Software Quality Factor (SQF)

Relationships between the set of metrics related to quality attributes (factors) and rating of quality factors have been established via regression analysis performed on empirical data. This relationship can be shown via linear equation. An example is given below.

$$r_f = c_1 m_1 + c_2 m_2 + c_3 m_3 + \cdots\cdots\cdots\cdots + c_i m_i$$

where:

$r_f = rating\ of\ the\ quality\ factor, f$

$c_i = regression\ coefficients$

$m_i = various\ measurements\ identified\ as\ relating\ to\ quality\ factor, f$ [7]

By creating the above relationship it is used as predictor. The measurements m_i are applied at specific times during the development.

Here data are created of three projects which will be discussed in further section. The above mathematical relationship and other statistical analysis are done on all three projects separately. There are following aspects of this approach.

- At highest level it is user-oriented
- At lower level oriented it is software-oriented
- Provide attributes' qualifications
- It is easy to use and can be applied any time during the software development

Additional metrics, function, and criteria can be added as the software technology changes [7].

3- SOFTWARE QUALITY METRICS

3-1 C & K (CHIDAMBER AND KEMERER) MATRICS

In 1994, Chidamber and Kemerer described [8] six metrics which are now called 'CK metrics'.These six metrics have been successfully correlated with the likelihood of error in software in many investigations. C&K metrics are used

- To measure unique aspects of the OO approach.
- To measure complexity of the design.
- To improve the development of the software

These 6 metrics are given below one by one.

1) Weight Methods per Class (WMC)
WMC is a measure of number of methods implemented within a class. If all method complexities are considered to be one unit, then WMC equals to the number of methods.

2) Depth of Inheritance Tree (DIT)
The length of the maximum path from the node to the root of the tree is DIT metric.

3) Number of children (NOC)
NOC is the number of immediate subclasses of a class in the hierarchy. This is an indicator of the potential influence a class can have on the design and on the system hierarchy.

4) Coupling between objects (CBO)
CBO is a count of the number of other classes to which a class is coupled. The methods of one class use the methods or attributes of the other class is called coupled classes.

5) Response for a Class (RFC)

The number of methods that can be invoked in response to a message in a class is called RFC. This measures the amount of communication with other classes.

6) Lack of Cohesion in Methods (LCOM)

The notion of degree of similarity of methods is used by LCOM metric. LCOM measures the amount of cohesiveness present, how well a system has been designed and how complex a class is [9].

3-2 MCCABE CYCLOMATIC COMPLEXITY

To evaluate the complexity of a method Cyclomatic complexity (McCabe) can be used [10].

- It is used to calculate important information about constancy and maintainability of software system from source code.
- It also provides guidance to the developer during the software project to help control the design.
- This metrics provide detail information about software module to identify the areas of possible instability in the testing and maintain phase.

This metric measures the complexity of a control flow graph of a method or procedure. The idea is to draw the sequence a program may take as a graph with all possible paths. Using this relation “connections - nodes + 2”, complexity is calculated and will give a number denoting how complex the method is. An example is given in Figure 1. Errors and Complexity are proportional to each other. Since complexity will increase the possibility of errors, a too high McCabe number should be avoided [11].

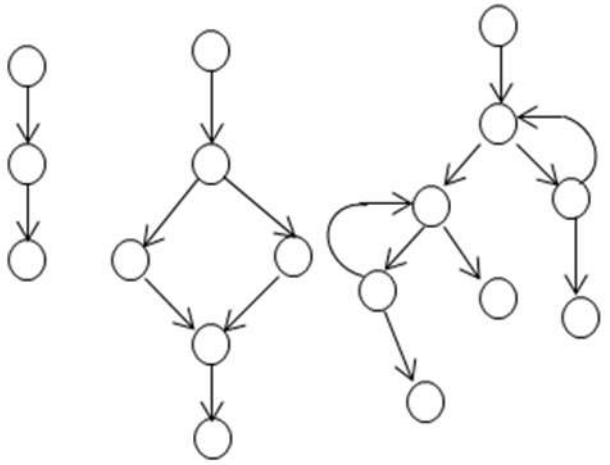

Figure 1 The McCabe complexity metrics.

N= 2-3+2 = 1 N= 6-6+2 = 2 N= 11-8+2 = 5

As described in Laing et al. [12], McCabe et al. [11] mention cyclomatic com-

plexity, based on graph theory, is a measure of a module control flow complexity. The cyclomatic complexity of individual methods can be combined to calculate the complexity of the class [10]. A high 10 cyclomatic complexity specifies that the code may be of low quality and difficult to test and maintain [12]. In the next section authors discussed system that has important role in this literature to find metrics.

4- SYSTEM DESCRIPTION

In this system there are three projects P0, P1, and P2 on which we have applied following steps for regression analysis of metrics.

4-1 PROPOSED METHODOLOGY

Table 1 Proposed methodology

Steps	Description
1	Project P0, P1 and P2 are written in java (Object Oriented Programming Language).
2	Every project is compiled and tested on Net Beans 8.0.
3	Metrics of project P0, P1, and P2 are calculated using CCCC, Source Code Monitor, SLOC and VIZZ Analyser etc.
4	The different combinations of CK metrics and MVG are made.
5	Statistics analysis of these metrics is done in step 6 and 7.
6	Linear and multi linear regression of metrics are done.
7	Correlation matrix of different metrics is calculated.
8	Model parameters are calculated.
9	Modal equations (mathematical expressions) of these metrics are calculated.
10	Graphs of these metrics are drawn using MATLAB.
11	Analyse the behaviour of these metrics according to graph.
12	Predict the optimal value of MVG using modal equation.

4-2 LOC MEASUREMENT OF SYSTEM

To find the volume of code we have to need LOC detail of system. LOC measurement is important because all the empirical analysis done on this whole system. In project P0 we used 10 classes, 375 Source Lines of Code (SLOC), 163 lines for comments, 115 blank lines, and this total is 653. In project P1 we used 20 classes, 433 Source Lines Of Code (SLOC), 168 lines for comments, 118 blank lines, and this total is 719. In project P2 we used 30 classes, 494 Source Lines of Code (SLOC), 168 lines for comments, 168 blank lines, and this total is 830. So in this way we analysed 60 classes and 2202 Lines of Code in our whole system. This analysis report is depicted in Table 2.

Table 2 LOC details of system

Projects	SLOC	Comments	Blank Lines	Total LOC	Classes
P0	375	163	115	653	10
P1	433	168	118	719	20
P2	494	168	168	830	30
Subtotal	1302	499	401	2202	60

4-3 FUNCTIONAL DETAILS OF SYSTEM

The complexity of any class depends upon its functionality. So functionality is the important factor of system. Here it is discussed the details of methods and variables of every class of every project. The functional details of system are given in subsection.

1) Project P0:

This project has 10 main classes. There is no inheritance in any class. Every class has different complexity level. The detail description of these classes is given in Table 3.

Table 3 Detail description of project P0

Class Name	Description
PercentageCalculation1	This class contains simple variables
PercentageCalculation2	This class contains 1 for loop
PercentageCalculation3	This class contains 1 for loop and if else
PercentageCalculation4	This class contains 1 for loop, if else, and if else inside for loop
PercentageCalculation5	This class contains 1 for loop, if else, if else inside for loop, and 1 other function GradeDisplayFunction()
PercentageCalculation6	This class contains 1 for loop, if else, if else inside for loop, and 2 other functions GradeDisplayFunction(), PercentageCalculateFunction()

PercentageCalculation7	This class contains 1 for loop, if else, if else inside for loop, and 3 other functions GradeDisplayFunction(), PercentageCalculateFunction() and third Display-GradeMessage() inside GradeDisplayFunction()
PercentageCalculation8	This class contains 1 for loop, if else, if else inside for loop, and 4 other functions GradeDisplayFunction(), PercentageCalculateFunction() and third Display-GradeMessage() inside GradeDisplayFunction() 4th IsTotalMarksConditionSatisfied()
PercentageCalculation9	This class contains 1 for loop, if else, if else inside for loop, and 3 other functions GradeDisplayFunction(), PercentageCalculateFunction() and third Display-GradeMessage() inside GradeDisplayFunction() and switch case
PercentageCalculation10	This class contains 2 nested for loop

2) Project P1:

This project has 10 main classes and total 20 classes. There is single level inheritance. Every class has different complexity level. The detail description of these classes is given in Table 4.

Table 4 Detail description of project P1

Class Name	Description
I1PercentageCalculation1	This program has single level inheritance
I1PercentageCalculation2	This program has single level inheritance and 1 for loop
I1PercentageCalculation3	This program has single level inheritance and 1 for loop and if else
I1PercentageCalculation4	This program has single level inheritance and 1 for loop, if else, and if else inside for loop
I1PercentageCalculation5	This program has single level inheritance and 1 for loop, if else, if else inside for loop, and 1 other function GradeDisplayFunction()
I1PercentageCalculation6	This program has single level inheritance and 1 for loop, if else, if else inside for loop, and 2 other functions GradeDisplayFunction(), PercentageCalculateFunction() 2 function in parent class
I1PercentageCalculation7	This program has single level inheritance and 1 for loop, if else, if else inside for loop, and 3 other functions GradeDisplayFunction(), PercentageCalculateFunction() and 3rd DisplayGradeMessage() inside GradeDisplayFunction(); 3 function in parent class

I1PercentageCalculation8	This program has single level inheritance and 1 for loop, if else, if else inside for loop, and 4 other functions GradeDisplayFunction(), PercentageCalculateFunction() and 3rd DisplayGradeMessage() inside GradeDisplayFunction(), IsTotalMarksConditionSatisfied; 4 function in parent class
I1PercentageCalculation9	This program has single level inheritance and 1 for loop, if else, if else inside for loop, and 4 other functions GradeDisplayFunction(), PercentageCalculateFunction() and 3rd DisplayGradeMessage() inside GradeDisplayFunction(), IsTotalMarksConditionSatisfied; 4 function in parent class and switch cases
I1PercentageCalculation10	This program has single level inheritance and 2 nested for loops

3) Project P2:

This project has 10 main classes and total 20 classes. There is multilevel inheritance. Every class has different complexity level. The detail description of these classes is given in Table 5.

Table 5 Detail description of project P2

Class Name	Description
I2PercentageCalculation1	This program has 2 level inheritance
I2PercentageCalculation2	This program has 2 level inheritance and 1 for loop
I2PercentageCalculation3	This program has 2 level inheritance and 1 for loop and if else
I2PercentageCalculation4	This program has 2 level inheritance and if else, and if else inside for loop
I2PercentageCalculation5	This program has 2 level inheritance and 1 for loop, if else, if else inside for loop, and 1 other function GradeDisplayFunction()
I2PercentageCalculation6	This program has 2 level inheritance and 1 for loop, if else, if else inside for loop, and 2 other functions GradeDisplayFunction(), PercentageCalculateFunction()
I2PercentageCalculation7	This program has 2 level inheritance and 1 for loop, if else, if else inside for loop, and 3 other functions GradeDisplayFunction(), PercentageCalculateFunction() and 3rd DisplayGradeMessage() inside GradeDisplayFunction(); 3 function in parent class
I2PercentageCalculation8	This program has 2 level inheritance and 1 for loop, if else, if else inside for loop, and 3 other functions GradeDisplayFunction(), PercentageCalculateFunc-

	tion() and 3rd DisplayGradeMessage() inside GradeDisplayFunction(); 3 function in parent class
I2PercentageCalculation9	This program has 2 level inheritance and 1 for loop, if else, if else inside for loop, and 4 other functions GradeDisplayFunction(), PercentageCalculateFunction() and 3rd DisplayGradeMessage() inside GradeDisplayFunction(), IsTotalMarksConditionSatisfied; 4 function in parent class and switch cases
I2PercentageCalculation10	This program has 2 level inheritance and 2 nested for loops

5- EMPERICAL STUDY

5-1 CALCULATION OF DIT AND MVG METRIC

We calculated DIT of these three projects separately using CCCC software. There are Table 6, 7 and 8 which consist of DIT values of project P0, P1, and P2 respectively.

1) Calculation of DIT and MVG of project P0:

Table 6 DIT and MVG details of project P0

Module Name	DIT	MVG
PercentageCalculation1	0	1
PercentageCalculation10	0	5
PercentageCalculation2	0	2
PercentageCalculation3	0	6
PercentageCalculation4	0	7
PercentageCalculation5	0	8
PercentageCalculation6	0	9
PercentageCalculation7	0	10
PercentageCalculation8	0	12
PercentageCalculation9	0	15
TOTAL	0	75

2) Calculation of DIT and MVG of project P1:

Table 7 DIT and MVG details of project P1

Module Name	DIT	MVG
ChildI1PercentageCalculation1	1	1
ChildI1PercentageCalculation10	1	5
ChildI1PercentageCalculation2	1	2
ChildI1PercentageCalculation3	1	6
ChildI1PercentageCalculation4	1	7
ChildI1PercentageCalculation5	1	8
ChildI1PercentageCalculation6	1	3
ChildI1PercentageCalculation7	1	3
ChildI1PercentageCalculation8	1	3
ChildI1PercentageCalculation9	1	6
I1PercentageCalculation1	0	1
I1PercentageCalculation10	0	1
I1PercentageCalculation2	0	1
I1PercentageCalculation3	0	2
I1PercentageCalculation4	0	2
I1PercentageCalculation5	0	2
I1PercentageCalculation6	0	6
I1PercentageCalculation7	0	7
I1PercentageCalculation8	0	10
I1PercentageCalculation9	0	10
TOTAL	10	86

3) Calculation of DIT and MVG of project P2:

Table 8 DIT and MVG details of project P2

Module Name	DIT	MVG
ChildI2PercentageCalculation1	1	1
ChildI2PercentageCalculation10	1	5
ChildI2PercentageCalculation2	1	2
ChildI2PercentageCalculation3	1	6
ChildI2PercentageCalculation4	1	7
ChildI2PercentageCalculation5	1	3
ChildI2PercentageCalculation6	1	3
ChildI2PercentageCalculation7	1	3

ChildI2PercentageCalculation8	1	3
ChildI2PercentageCalculation9	1	6
DchildI2PercentageCalculation1	2	1
DchildI2PercentageCalculation10	2	1
DchildI2PercentageCalculation2	2	1
DchildI2PercentageCalculation3	2	1
DchildI2PercentageCalculation4	2	1
DchildI2PercentageCalculation5	2	1
DchildI2PercentageCalculation6	2	1
DchildI2PercentageCalculation7	2	1
DchildI2PercentageCalculation8	2	1
DchildI2PercentageCalculation9	2	1
I2PercentageCalculation1	0	2
I2PercentageCalculation10	0	2
I2PercentageCalculation2	0	2
I2PercentageCalculation3	0	2
I2PercentageCalculation4	0	2
I2PercentageCalculation5	0	7
I2PercentageCalculation6	0	7
I2PercentageCalculation7	0	8
I2PercentageCalculation8	0	10
I2PercentageCalculation9	0	10
TOTAL	30	101

6- CORRELATION ANALYSIS

Using the result of Table 6, 7, and 8 we can make new Table 9.

Table 9 DIT and MVG of project P0, P1, and P2

Projects	DIT	MVG	MVG Increased %
P0	0	75	
P1	10	86	14.66
P2	30	101	34.6667

6-1 CORRELATION ANALYSIS BETWEEN DIT AND MVG

By using this correlation analysis we calculated the strength of relationship between DIT and MVG. We also established a mathematical expression of relationship and graphs between these two metrics.

1) Correlation coefficient (r) of DIT and MVG:
Using Table 10 we calculated the Correlation coefficient *r*.

Table 10 Correlation coefficient (r) of DIT and MVG

Projects	DIT (x)	MVG (y)	x2	y2	xy
P0	0	75	0	5625	0
P1	10	86	100	7396	860
P2	30	101	900	10201	3030
Total	Σx=40	Σy =262	Σx2= 500	Σy2=23222	Σxy= 3890

$$\sum dx^2 = \sum x^2 - \frac{\left(\sum x\right)^2}{n} = 466.667 \qquad \sum dy^2 = \sum y^2 - \frac{\left(\sum y\right)^2}{n} = 340.667$$

$$\sum dxdy = \sum xy - \frac{\sum x \sum y}{n} = 396.667 \qquad r = \frac{\sum dxdy}{\sqrt{\sum dx^2 \sum dy^2}} = 0.99485$$

This value of *r* is positive which indicates that the slope will rise from left to right in DIT and MVG's graph.

2) Graph between DIT and MVG:
We used statistical data of Table 9 to draw linear graph in MATLAB (Matrix Laboratory). We took MVG on Y-axis and DIT on X-axis. This graph is shown in Fig 2.

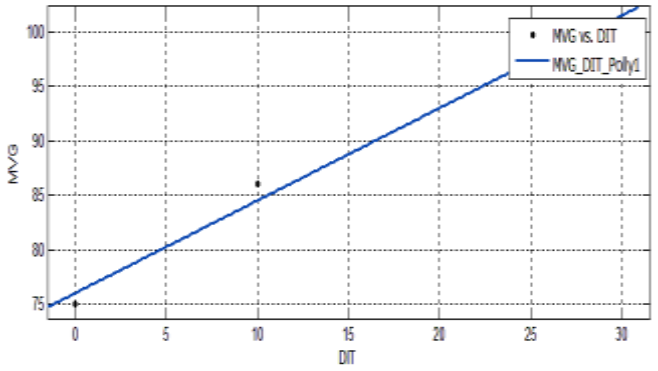

Figure 2 Linear graph between DIT and MVG

3) Linear Equation between DIT and MVG:
Using the above statistical data given in Table 9 we found the following mathematical Equation 1. Here in this equation *y* is MVG and *x* is the DIT of the

system. Coefficients of this equation are p1 and p2.

$$y = p_1 * x + p_2 \quad (1)$$

$$p_1 = \frac{n\sum xy - \sum x.\sum y}{n\sum x^2 - (\sum x)^2} \qquad p_2 = \frac{\sum y}{n} - p_1 \frac{\sum x}{n}$$

p_1 =0.85 p_2 = 76

On putting x (DIT) = 50 in Equation 1 we find
y = 0.85*50 + 76, y (MVG) = 118.5

7- OBSERVATIONS

Now we can say that when we change DIT to 50, MVG will be 118.5. This DIT value indicates that if we want to restrict our system with MVG's value = 118.5 then we have to take at most 50 value of DIT in our system. If our system cannot handle greater than 100 MVG then we have an alarm (threshold) value 50 of DIT for our system.

Here we discussed about the regression analysis of DIT metric with respect to MVG (complexity). It is measured that when we increase DIT, MVG also increases in polynomial form. This paper provides an optimal value of DIT at which we can have quality software after that value of complexity will increases exponentially.

8- CONCLUSION AND FUTURE SCOPE

The empirical relations of DIT over MVG metrics have been computed in this paper. It is found that this relationship provides a maximum limit of DIT after which complexity will increase exponentially. The same has also been elaborated by graphical and mathematical expressions in this paper. The future scope of this work is shown point wise below:

1. This study may be replicated by using many new metrics and by making different combinations of those and after making the combinations they can find the correlations among those metrics.
2. One can make a system which tells them the relationship among those new metrics.
3. One can also consider our project as reference for various predictions of quality software related to metrics.
4. Using the concept of this paper the maximum as well as minimum limit of any metric with complexity can be set.

REFERENCES

[1] S. Dick and A. Kandel, “Fuzzy Clustering of Software Metrics,” The 12th IEEE International Conference on Fuzzy Systems, 2003. FUZZ ’03, Volume 1, pp. 642 – 647, May 2003.

[2] A. Urvashi, and A. Chhabra, “Change-Proneness of Software Components,” IOSR Journal of Computer Engineering (IOSR-JCE), Volume 16, Issue 2, Ver. VIII, pp. 45-48, Mar-Apr. 2014.

[3] H. Jie Lee, L. Naish, and K. Ramamohanarao, “Study of The Relationship of Bug Consistency with Respect to Performance of Spectra Metrics,” 2nd IEEE International Conference on Computer Science and Information Technology, 2009, ICCSIT 2009, pp. 501 –508, Aug. 2009.

[4] C. Jin, Shu-Wei Jin, Jun-Min Ye, and Qing-Guo Zhang, “Quality Prediction Model of Object-Oriented Software System Using Computational Intelligence,” 2nd International Conference on Power Electronics and Intelligent Transportation System (PEITS), 2009, Volume 2, pp. 120 –123, Dec. 2009.

[5] C. Zhang; Budgen, D., “What Do We Know about the Effectiveness of Software Design Patterns?” IEEE Transactions on Software Engineering, Volume 38, no.5, pp. 1213, 1231, Sept.-Oct. 2012.

[6] V. Chauhan, D.L. Gupta and S. Dixit, “Role of Software Metrics to Improve Software Quality,” International Journal of Computer Science and Information Technologies (IJCSIT), Volume 5 (3), ISSN: 0975-9646, pp. 4167-4170, 2014.

[7] J. P. Cavano, “A Framework for the Measurement of Software Quality,” Rome Air Development Center, James A. McCall General Electric Company.

[8] S.R. Chidamber and C.F. Kemerer, “A Metrics Suite for Object Oriented Design Software Engineering,” IEEE Transactions, 20(6), pp. 476 –493, Jun. 1994.

[9] Chidamber, Shyam, Kemerer and Chris F, “A Metrics Suite for Object-Oriented Design,” M.I.T. Sloan School of Management E53-315, 1993.

[10] McCabe and Associates, “Using McCabe,” QA 7.0, 9861 Broken Land Parkway 4th Floor Columbia, MD 21046, 1999.

[11] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, “Object-Oriented Software Engineering: A Use-Case Driven Approach,” Addison-Wesley, 1992.

[12] V. Laing and C. Coleman, “Principal Components of Orthogonal OO Metrics," Software Assurance Technology Center (SATC), 2001.