Integrating and Optimizing Object-Oriented Software Metric Tools

Vijay Yadav, Raghuraj Singh, Vibhash Yadav
Department of CSE, Dr. A. P. J Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India
Department of C.S.E, Harcourt Butler Technical University, Kanpur, Uttar Pradesh, India
Department of I.T, Rajkiya Engineering College, Banda, Uttar Pradesh, India

Abstract:
Software Metrics provide critical information about the reliability and maintainability of the system and hence are used to improve the productivity and quality of software. As the mathematical nature of metrics calls for precise definitions of the same metric depending on the implementation language, it is suggested to express and define metrics using a language-independent Meta-model based on graphs. Several software metric tools are available, using different methods to assess metric-based software systems and hence project different results. The results are thus tool dependent and are in question for validation. Here an attempt is made to integrate five different object-oriented commercial and free metric tools, and their results have been optimized. An Empirical study has been done to calculate the metrics values using the same standard metrics for two software projects of different sizes. The results have been presented and discussed here to show the variations in results from different tools for the same metrics. To support the validation, a manual investigation has been done on a small java file to check the validity of tools for optimization of the different values obtained for the same metrics from different tools.

Keywords: Object-oriented metrics, Software product Metrics, Software metric tool, and optimization
DOI: 10.24297/j.cims.2023.1.13

1. Introduction

Metrics are essential in several disciplines of software engineering. Different metrics need to be used in projects, but this paper focuses on Object-Oriented (OO) metrics. These days, Object Oriented design is more prevalent in software development environments. One of the reasons for this is that the traditional metrics measures are generally used in the structured programming paradigm, where the design structure and data structure are measured independently. But an Object Oriented metric can treat function and data as integrated Objects [5] [6]. The popularity of Object-Oriented software development is due to its powerful features like encapsulation, object abstraction, inheritance, polymorphism, dynamic binding, and reusability. Software metrics are the tools to evaluate the quality of the design.

Measurements using some of these metrics have been automated in this paper [9][10] to facilitate the collection of measurement results. Many of the proposed metrics have not been included in this survey. The reason for this is that either they have very similar characteristics/slide variations to metrics included in this survey or that they reapply a traditional metric to object-oriented programming.

2. Object-Oriented Metrics

The metrics presented here are class-related metrics, method-related metrics, inheritance metrics, metrics measuring coupling, and metrics measuring general (system) software production characteristics.

In this paper, nine metrics are considered for optimization. These metrics are: DIT (Depth Of Inheritance), NOC (Number of Children), CBO (Coupling Between Objects), RFC (Response For A Class), NPM (Number Of Public Methods), LOC (Lines Of Code), WMC (Weighted Method Complexity), AMC (Average Method Complexity), WAC (Weighted Attributes Per Class).

3. Software Metric Tool Selection

To find a set of suitable software metrics tools, a free search on the internet was conducted [1]. According to analyzable languages, metrics calculated, and availability/license type, as mentioned in Table 1, it was decided to choose the set of five tools.

Table 1 Requirements as a basis for the selection of Tools

S.No.	Requirements	Type to suite requirement
1.	Supporting language	java
2.	measuring metrics	object-oriented metrics
3.	license type	freely available
4.	characteristics	command line tool

The selected tools are listed below:

A.CCCC (C and C++ code counter)

It is an open-source command-line tool that analyzes C++ and Java files and generates reports on various metrics, including Lines Of Code and metrics proposed by Chidamber & Kemerer [4][7] and Henry & Kafura [5]. It is developed by Tim Littlefair of Edith Cowan University [8].

B.CKJM (Chidamber & Kemerer Java Metrics)

It is an open-source command-line tool. It calculates the C&K object-oriented metrics by processing the byte code of compiled Java files [10].

C.JMT (java measurement tool)

The Java Measurement Tool realizes a static measurement of Java applications. It analyzes the Java classes and the relations between them. It is developed by *Ingo Patett* as his Diplom-Arbeit, Magdeburg, Germany, Department of Software Engineering [11][12].

D.Dependency Finder

It is an open-source command line tool that analyzes compiled Java code. Its core is a dependency analysis application that extracts dependency graphs and mines them for valuable information [9].

E.Automated Pop Analyzer

This tool is user-friendly; hence easy, to use as a support tool for the measurement of the pop count. The tool claims to provide pop count values more accurately and efficiently. The POP metric is a good indicator of software size, which can be easily seen through the results of POP calculations of the APA tool [6].

4. Metric Selection For Optimization

The metrics selected are the ones with the most significant common subset of the metrics measured by all selected software metrics tools. During the study, some metrics are found which are measuring the same parameter with different names, and hence these metrics are also compared under the same name during assessment [2]. Finally, nine software metrics have been selected for this study. These metrics work on different program entities, e.g., method, class, package, program, etc. A detailed definition of all these metrics, along with the procedure to measure respective metrics, has been explained below:

A.DIT (Depth of Inheritance Tree)

Depth of inheritance is the maximum inheritance path from the class to the root class. In cases involving multiple inheritances, the DIT will be the maximum length from the node to the tree's root [15][17]. It measures how many ancestor classes can potentially affect this class. Fig.1 shows the method for measurement of DIT.

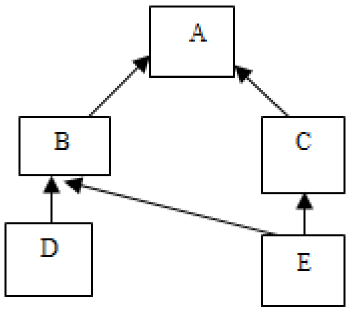


Fig. 1: Sample Measurement of DIT

Consider the class inheritance tree in Fig. 1,

DIT (A) = 0 because A is the root class.

DIT (B) = DIT(C) = 1 because the length from class B and C to the root A is one each. And,

DIT (D) = DIT (E) = 2 because the maximum length from class D and E to root A is two each.

B.NOC (Depth of Inheritance Tree)

It is the maximum inheritance path from the class to the root class. NOC is the number of immediate sub-classes subordinated to a class in the class hierarchy. It is a measure of how many subclasses are going to inherit the methods of the parent class [15] [16].

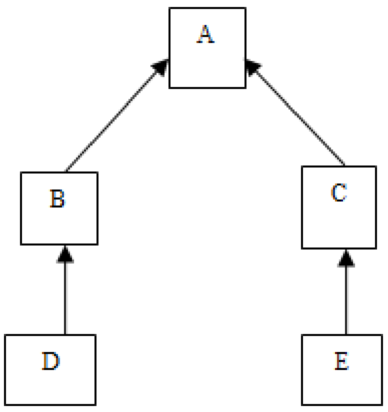


Fig. 2: Sample Measurement of NOC

E.g., consider the class inheritance diagram in Fig 2.

NOC (A) = 2 because the number of their immediate children is two.

NOC (B) = NOC (C) = 1 because the number of its immediate child is one.

NOC (D) = NOC (E) = 0 because they have no children.

C.LOC (Lines of Code)

LOC counts the lines of code of a class. It is the number of physical lines of active code. Size can be measured to count all physical lines of code. In LOC count, counting starts from 0 [12] [13].

D.CBO (Number of Children)

It is the number of immediate subclasses subordinated to a class in the class hierarchy.CBO of a class is defined as the number of other classes to which it is coupled. CBO determines whether a class is using an attribute in another class or not. As stated earlier, since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class [17].

E.RFC (Number of Methods)

RFC is the total number of all methods within a set that can be invoked in response to a message sent to an object to operate. All methods accessible within the class hierarchy are included in the count.

RFC = | RS |

where RS is the response set for the class [14] [17].

E.g., the RFC for Vehicle shown in Figure 3, is the number of methods that can potentially be executed in response to a message by itself, by Car, and by Emergency Vehicle. Thus the RFC for Vehicle is 3, for Car is 4 (drive, turn, indicate, sense), and for Emergency Vehicle is 6. In the above example, if an aspect affects a method in the Vehicle class, then the RFC for it would increase by the number of methods in the aspect. RFC was increased in all real-time areas.

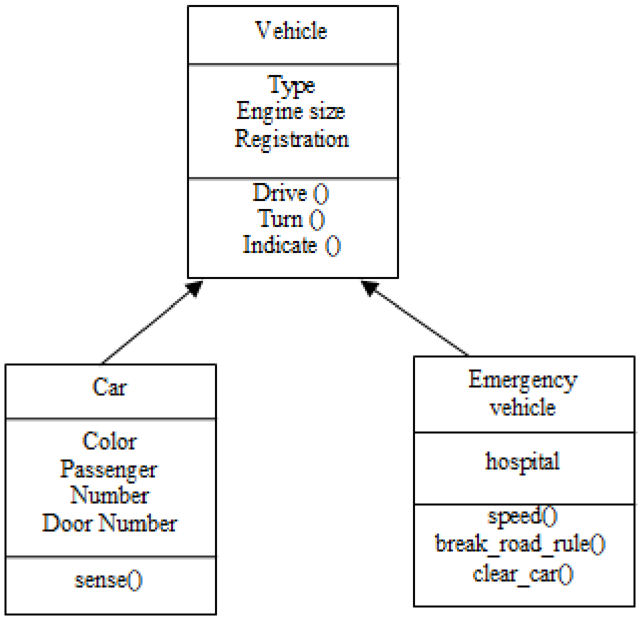


Fig. 3: Sample Measurement of RFC

F.NPM (Number of public methods)

It is the set of methods that can be executed in response to a message received by an class object. It counts all the methods in a class that is declared public. It can be used to measure the size of an API provided by the package [17].

G.WMC (weighted method complexity)

WMC is the count of methods implemented within a class or the sum of the complexities of the methods. WMC relates directly to the complexity of a thing since methods are properties of object classes, and complexity is determined by the cardinality of its set of properties. The number of methods is, therefore, a measure of the class definition and attributes of a class since attributes correspond to properties.

Consider a Class C1, with methods M1... Mn *that are defined in the class*. Let c_1... c_n be the complexity of the methods[6] [14].

$$WMC = \sum_{i=1}^{n} Ci$$

If all method complexities are considered unity, then WMC = n, the number of methods.

E.AMC (Average method complexity)

This metric measures the average method size for each class. The size of a method is equal to the number of java binary codes in the method.

F.WAC (weighted method complexity)

This metric calculates the weighted attributes per class. These are the number of attributes of the considered class.

Table 2 Summary of Tools and Metrics Used In Empirical Study

TOOLS	METRICS								
	DIT	NOC	LOC	CBO	RFC	NPM	WMC	AMC	WAC
CCCC	*	*	*	*			*		
CKJM	*	*	*	*	*	*	*	*	
JMT	*	*	*	*	*	*	*		*
DEPENDENCY FINDER	*	*	*			*	*		*
APA	*	*					*	*	

5. Software Metric Optimization Tool

There are different tools available in the market, but they measure different values for the same metrics. In this proposed system, five different Object-oriented tools are integrated and made as a single add-on for the users’ flexibility and user-friendliness. All the tools are first converted to a single platform so that the same metrics may be compared on the same platform. It compares the metric values and optimizes the result by choosing the correct value obtained through manual investigation. The java decompiler, an open source also introduced into this tool to convert the class files into java files.

6. Description Of Empirical Study

SourceForge.NET provides a large variety of open-source software projects. Over 30.000 are written in Java, and it is possible to search directly for Java programs. To analyze, projects are chosen in different size categories. Two projects have been chosen for Empirical study. The first project is the address book shown in Table 3; it is used in proxy models to display different views of data from a single model.

Table 3 OO Metrics values measured through software metric optimization tool for project address Book

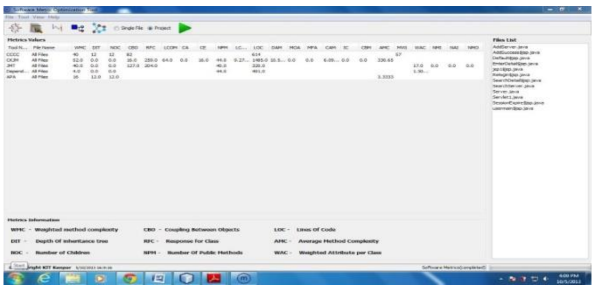
TOOLS	OOMETRICS								
	DIT	NOC	CBO	WMC	RFC	NPM	LOC	AMC	WAC
CCCC	12	12	82	40			614		
CKJM	0	0	16	52	259	44	1485	330	
JMT	0	0	127	40	204	40	320		17
DEPENDENCY FINDER	0	0		4		44	491		1.30
APA	12	12		16				3.33	
OPTIMIZED VALUE	12	12	82	16	259	40	614	3.33	17

Another project is the airways reservation system shown in Table 4, which is used to connect to the airline database in order to show the flight module like fare, time, etc.

Table 4 OO Metrics values are measured through software metric optimization tool for project airways reservation system

TOOLS	OOMETRICS								
	DIT	NOC	CBO	WMC	RFC	NPM	LOC	AMC	WAC
CCCC	27	34	248	38			985		
CKJM	65	0	28	25	254	14	3255	1531	
JMT	0	0	148	38	129	27	635		127
DEPENDENCY FINDER	0	0		6.35		14	782		9.07
APA	27	34		0				1.4	
OPTIMIZED VALUE	27	34	248	0	254	27	985	1.4	127

Fig. 4 shows the snapshot of the optimized tool showing metric values obtained through different tools after the integration of Tools.



Software Metric Optimization Tool

Single File (●) Project

Metrics Values

Tool N...	File Name	WMC	DIT	NOC	CBO	RFC	LCOM	CA	CE	NPM	LC...	LOC	DAM	MOA	MFA	CAM	IC	CBM	AMC	MVG	WAC	NMI	NAI	NMO
CCCC	All Files	40	12	12	82							614								67				
CKJM	All Files	52.0	0.0	0.0	16.0	259.0	64.0	0.0	16.0	44.0	9.27...	1485.0	10.5...	0.0	0.0	6.09...	0.0	0.0	330.65					
JMT	All Files	40.0	0.0	0.0	127.0	204.0				40.0		220.0									17.0	0.0	0.0	0.0
Depend...	All Files	4.0	0.0	0.0						44.0		461.0									1.30...			
APA	All Files	16	12.0	12.0															3.3333					

Files List

AddServer.java
AddSuccess$jsp.java
Default$jsp.java
EnterDetail$jsp.java
jsp1$jsp.java
Relogin$jsp.java
SearchDetail$jsp.java
SearchServer.java
Server.java
Servlet1.java
SessionExpire$jsp.java
usermain$jsp.java

Metrics Information

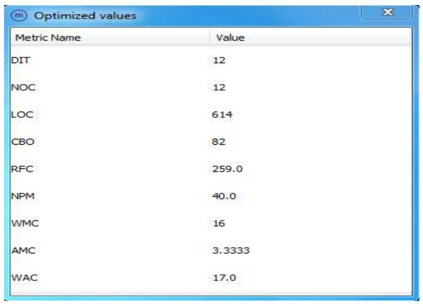
WMC - Weighted method complexity	CBO - Coupling Between Objects	LOC - Lines Of Code
DIT - Depth Of inheritance tree	RFC - Response for Class	AMC - Average Method Complexity
NOC - Number of Children	NPM - Number Of Public Methods	WAC - Weighted Attribute per Class

© Copyright KIT Kanpur

Software Metrics(completed)

Fig. 4 Sample values through the optimized tool

Fig. 5 shows the optimized values of all nine metrics obtained after optimization.



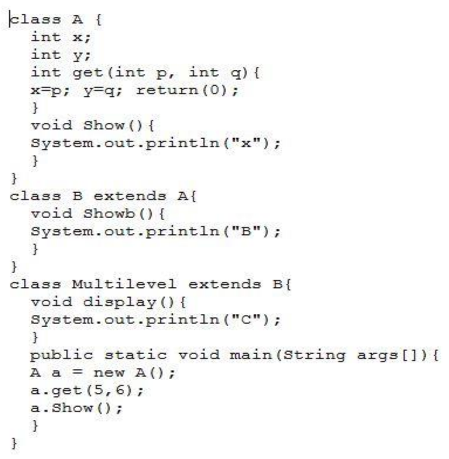
Optimized values

Metric Name	Value
DIT	12
NOC	12
LOC	614
CBO	82
RFC	259.0
NPM	40.0
WMC	16
AMC	3.3333
WAC	17.0

Fig. 5 Sample optimized values through the tool

7. Evaluation And Result Analysis

Looking at the above individual metrics values per class, it is visible that there are differences in how the tools calculate these values. If all tools deliver the same values, then there would not be any need for optimization. From Table 3 and Table 4, a significant difference for some metrics values has been found, measured through the different tools. Thus for validation of chosen correct value for optimization, a simple project file has been taken to investigate manually. Fig. 6 shows the code of chosen project file taken for manual investigation.



```
class A {
   int x;
   int y;
   int get(int p, int q){
   x=p; y=q; return(0);
   }
   void Show(){
   System.out.println("x");
   }
}
class B extends A{
   void Showb(){
   System.out.println("B");
   }
}
class Multilevel extends B{
   void display(){
   System.out.println("C");
   }
   public static void main(String args[]){
   A a = new A();
   a.get(5,6);
   a.Show();
   }
}
```

Fig. 6 Project file taken for manual investigation

The above java file has three classes A, B, and Multilevel. Class A has two methods which are show() and get(). Class B has only one method, that is showb(), and class multilevel has two methods which are display() and main().

In inheritance measurement, class multilevel inherits class B and class B inherits class A. The DIT value of class A is 0 because it does not inherit any class. The DIT value of class B is 1 because it inherits class A, and the DIT value of class multilevel is 3 because it inherits B, which inherits A. Thus the total number of DIT value for the java file is 3(3=0+1+2), which is the same as the CCCC tool, so this value is chosen for optimization.

In NOC measurement, for class A, the value of NOC is 1 because it has one child, i.e., B. The NOC of class B is also 1 because it also has one child, i.e., class multilevel. The NOC of multilevel is 0 because it does not have any child. Thus the total number of children is 2(2=1+1+0), which is also the same as the value given by the CCCC tool, so this value is chosen for optimization.

In CBO measurement, A shows coupling as a client of B. B shows two couplings, one multilevel as a client and the other A as a supplier. Multilevel shows two types of coupling as supplier and String as a parameter, and String shows coupling as a client of multilevel. Hence total no. of coupling for this class is 6.

Class A response has three methods which get (), show(), and return(). For class B, it counts the number of methods used by class A plus the methods of class B, i.e., 4. Multilevel class counts the number of methods of class B plus the methods (display(), main(), a(), get(), show(), string()) of class multilevel, i.e., 10. Hence total no. of RFC value is 17, which is the same as given by the CKJM tool.

NPM is one, i.e., primary (), which is the same as given by jmt tool.

WMC value is measured as 0, which is correct using the APA tool.

SLOC count is 25; counting starts from 0, CCCC measures correct lines of code the same as counted manually.

WAC value is 2, which is correctly measured by the JMT tool.

Table 5 shows the OO Metrics values measured manually and their corresponding matching with the values obtained through different tools hence helping in getting the optimized values for chosen metrics.

Table 5 OO Metrics values measured through manual investigation

TOOLS	OOMETRICS								
	DIT	NOC	CBO	WMC	RFC	NPM	LOC	AMC	WAC
CCCC	3	2	6	5			24		
CKJM	1	3	10	11	25	2	73	21.66	
JMT	3	2	5	5	15	1	9		2.0
DEPENDENCY FINDER	0	3		2.2		2	27		0.4
APA	3	2		0				1.6	
MANUALLY COUNT	3	2	6	0	25	1	24	1.6	2.0
OPTIMIZED VALUE	3	2	6	0	25	1	24	1.6	2.0

8. Conclusion And Future Scope

Today a large number of software metrics tools exist. But give different values for the same projects, and hence none of them have been validated experimentally for the software metric values they measure. Most tools computed different values for the same metrics on the same projects. From the study, it is observed that metrics-based results cannot be compared when using different metrics tools as the metric values are tool dependent. To use them safely practitioners, a new tool has been proposed in the paper by integrating five available tools. Since metrics results are strongly dependent on the implementing tools, a validation in terms of manual investigation only supports the applicability of some metrics as implemented by a particular tool. All nine different object-oriented metrics measured by them have been optimized by investigating the results manually.

Here five metric tools have been investigated and validated for few metrics. All the validated metric values or optimized results have been made the part of the new tool.

This paper also suggests which tool correctly measure which metric value. It has been found that CCCC gives the best results for DIT, NOC, LOC and CBO Metrics. CKJM tool gives the best value for RFC metric. JMT tool gives the best value for NPM and WAC metrics. APA tool gives the best value for WMC and AMC Metrics. The same may be useful for the practitioners to use.

The study has been done on two projects which should be further extended in order to make the results comparable and more general. Further, more tools can be taken for integration in order to check their validity or comparison and hence by this more software metrics may be empirically validated and hence their measurement tool.

References

1. Abdallah M. and Alrifaee M., "A Heuristic Tool for Measuring Software Quality Using Program Language Standards," *The International Arab Journal of Information Technology*, vol. 19, no. 3, 2022, pp. 314-322 .
2. Nassif, A.B., Azzeh, M., Idri, A. and Abran, A., "*Software development effort estimation using regression fuzzy models*," . Computational intelligence and neuroscience, 2019.
3. Rüdiger Lincke, Jonas Lundberg and Welf Löwe, "Comparing software metrics tools" , software technology group school of mathematics and systems engineering växjö university, Sweden issue, July 20–24, 2008, Seattle, Washington, USA.
4. Kemerer, C. F. "An empirical validation of software cost estimation models" ,Comm.ACM 30,5 (May 1987), pp. 416-429.
5. W. li and S. henry "Maintenance metrics for the object oriented paradigm" . in proc. software metrics symposium, 1993, pp. 52-60.
6. Shubha Jain, Vijay Yadav and Prof. Raghuraj Singh," OO estimation through automation of the predictive object points sizing metric" , international journal of computer engineering& technology (IJCET) volume 4, issue 3, May-June (2013), pp. 410-418.
7. S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 1994.
8. http://cccc.sourceforge.net/. Accessed on 5/11/2013
9. http://depfind.sourceforge.net/. Accessed on 6/7/2013
10. http://www.spinellis.gr/sw/ckjm/. Accessed on 10/8/2013

11. www2.informatik.hu-berlin.de/swt/intkoop/jcse/**tools**/jmt.html
12. http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/tools
13. Seyyed Mohsen Jamali" , Object Oriented Metrics (A Survey Approach)" , Department of Computer Engineering, Sharif University of Technology ,January 2006,Tehran Iran.
14. Victor R. Basili, Lionel Briand and Walcélio ," A validation of object-oriented design metrics as quality indicators" , Melo technical report, Univ. of Maryland, dep. of computer science, college park, md, 20742 USA, April 1995.
15. Sanjay Misra, Ibrahim Akman, Murat Koyuncu," An inheritance complexity metric for object-oriented code: a cognitive approach" , Springer June 2011, Volume 36, Issue 3, pp 317-337.
16. B. Henderson-Sellers, L. Constantine and I. Graham. "Coupling and Cohesion (To-wards a Valid Metrics Suite for Object Oriented Analysis and Design)" . In proc. Object Oriented Systems, Vol. 3, 1996, pp. 143-158
17. M. Xenos, D. Stavrinoudis, Zikouli, Christodoulakis," Object-Oriented Metrics a Survey" , federation of European software measurement associations, Madrid, Spain, 2000