OBJECT ORIENTED SOFTWARE QUALITY ASSESSMENT USING METRICS THRESHOLD VALUES

ISSN: 2141 – 3290
www.wojast.com

[1]UDO, E.N. AND [2]AKWUKWUMA, V. V. N.
[1]Department of Computer Science, University of Uyo, Uyo, Nigeria
[2]Department of Computer Science, University of Benin, Nigeria
Email: edwardudo@yahoo.com, +2348023339501
[2]vakwukwuma@yahoo.com, +2348033440003

ABSTRACT

Quality of object oriented software is expressed by measuring its internal attributes using object oriented design metrics. Metrics values alone cannot predict quality of software without its threshold values. In this work, a software analyzer has been developed to measure the internal attributes (coupling, cohesion, inheritance and complexity) of 40 open java source codes, at class level, using Chidamber and Kemerer (CK); Danisaya and Davies metric suites to examine the quality of source codes. The analyzer does not only measure the metric values of these internal properties but also incorporates decision rules to ascertain the level of these metric values when matched against estimated thresholds. Threshold values of these metrics act as determinant point in measuring software quality. The desirable level of the selected internal object oriented software properties is coupling – low; cohesion – high; inheritance – low and complexity – low. The results from this analyzer fell within the minimum and maximum values when compared with the results from other metrics tools. Since the analyzer developed in this work uses threshold values and decision rules together with desirable level of the internal software properties, its result is robust and reliable, which can be used by software developers for assessment of the quality of object oriented source codes they developed for appropriate corrective measures before the software is finally released.

INTRODUCTION

As individuals, corporate organizations and government use several software products to enhance effectiveness of their operations, the demand for quality software continues to increase. Therefore there is need to improve software productivity and quality. ISO, 2005 defined quality as the "degree to which a set of inherited characteristics fulfill requirements." The only way to ascertain the quality of software product is by measuring it. In object oriented approach, quality is expressed by measuring the internal attributes of software. Attributes of object oriented software such as coupling, encapsulation, cohesion, size, polymorphism etc, can act as pointers for assessing the quality of object oriented software. These attributes are termed internal quality attributes - those that can be measured purely in terms of the product, process or resource itself (Fenton and Pfleeger, 1996). They can be measured by observing only the process, product or resource without considering its behaviour. This measurement can be done at the source code level. Quality of the source code has a lot of impact on the quality of the software (Baggen *et al*., 2012). Software quality can be measured by evaluating key software attributes using metrics (Kayarvizly and Kanmani, 2011).

Metric is a quantitative measure of degree to which a system, component or process possesses a given attribute. It assigns a value (number or symbol) to the attributes of entities in the real world based on clearly defined rules (Yang *et al*., 2010). Software metrics refer to measurement that can be applied to check the indicators of processes, projects and software products. Measurement is viewed as the process by which these values are assigned (Fenton, 1991).

Many object oriented metrics have been proposed for assessing the design of software system, however most of the approaches used in measuring these design metrics take into consideration only few aspects of object oriented design paradigms. As such it becomes quite unclear of the quality of the source code and gives no practical support to software developers.

The objective of this work therefore is to design a metric tool using properties that consider the complexity, interrelationship within and between classes, dependency and composition of classes in object oriented software system that would be able to measure and reflect the quality of the source code thereby giving an insight of the code to the developers. The properties used include coupling, cohesion, inheritance and complexity.

Many software quality models use software metrics to determine quality attribute of software product. An appealing operational approach for quality management using object oriented measures is to develop thresholds (Benlarbi *et al*., 2000). Thresholds are heuristic values used to set ranges of desirable and undesirable metric values for measured software. The aim of these thresholds is to provide a benchmark for the quantitative evaluation and assessment of the internal quality of software system (Filo *et al*., 2015). The values that are greater than a threshold value are considered to be problematic, the values below the threshold value are considered to be normal.

METHODOLOGY

The approaches used in this work are:

a. The analysis of source codes written in java using a software analyzer to extract the values of coupling, cohesion, inheritance and complexity using Chidamber and Kemerer (1994) object oriented design metrics suite (CBO, RFC, LCOM1,DIT, NOC and WMC) and NOM/LCOM2 metrics by Banisaya and Davies (2002). Metrics values above these threshold values were considered as high (1) while those below were considered as low (0).
b. Rules were formulated using the threshold values range and the desirable level of the internal software properties: coupling (low), cohesion (high), inheritance (low) and complexity (low) to determine the attributes level of given software. Low were depicted as **0** while high is depicted as **1**. Attributes levels were computed for 40 different open source object oriented codes written in java.

Chidamber and Kemerer (CK) metrics suites were selected, because of their wide acceptance among the engineering community (Elish and Alshayeb, 2012). Number of Methods (NOM) and Lack of Cohesion in Methods 2 (LCOM 2) metrics were added to help in accurate measure of the selected internal software properties. Software metrics in CK suite have been tested and validated (Basili and Melo, 1996; Anthony, 2013). Their threshold values have also been validated through theories, experiments and statistical analysis.

OVERVIEW OF METRICS USED AND THEIR THRESHOLD VALUES

i. Coupling Between Object (CBO) – The degree of dependency between the object within a class.
ii. Response For a Class (RFC) – The number of sets of methods in a class. The number of methods that can be invoked in response to a message in a class.
iii. Lack of Cohesion in Method 1 (LCOM1) - The count of the number of method pairs whose similarity is zero minus the count of method pairs whose similarity is not zero.
iv. Lack of Cohesion in Methods 2 (LCOM2) - The percentage of methods that do not access a specific attribute averaged over all attributes in the class
v. Depth of Inheritance Tree (DIT) - This is the length of longest path from the class to the root class of the tree. This metrics calculates how far down a class is declared in the inheritance hierarchy.

vi. Number of Children (NOC) - The number of classes that inherit directly from a given class. It measures how many sub-classes are going to inherit the methods of the parent class.
vii. Weighted Method per Class (WMC) – The number of all declared methods and constructors within a class.
viii. Number of Methods (NOM) – The number of methods implemented (those declared and those not declared) in a given class (Table 1).

Table 1 – Metrics Threshold Value

METRICS	THRESHOLD VALUE	SOURCE
CBO	5	Rosenberg *et al*. (1998); Benlarbi *et al.* (2000)
RFC	100	Rosenberg *et al.*,(1998); Benlarbi *et al.* (2000)
LCOM1	1	Chandra and Linda (2010)
LCOM2	2	Chandra and Linda (2010)
DIT	6	Chandra and Linda (2010)
NOC	6	Chandra and Linda (2010)
WMC	100	Rosenberg *et al*. (1998); Benlarbi *et al.* (2000)
NOM	20	Herbold *et al.*, (2010)

Many researchers have used metrics threshold values to predict external software attributes such as maintainability, faut-proness, reusability, testability, understandability etc (Benlarbi *et al.*, 2000; Chandra and Linda, 2010; Bakar *et al*., 2014; Singh and Kahlon, 2014).

In this work, apart from threshold values, decision rules are formulated to determine whether a given attribute is below or above the specified threshold value. This provides practical help to software developers to ascertain the quality of the developed software, which would enhance employment of corrective measures by the developers during coding. This assures quality software.

SYSTEM DESIGN

The architecture of the software analyzer developed and used in this work is shown in Figure 1. The file system module is a retrieval system that gets a software file and sends it to the filter for filtering. The filter will extract out all non codes documents and bring out the software source code for analysis. The analysis phase analyzes the source code, extracts metric values relating to the selected internal software attributes - coupling, cohesion, inheritance and complexity, using some selected object oriented software metrics. The values obtained are matched against the estimated threshold values for each of the metrics.

The knowledge base houses the decision rules to be extracted by the inference engine to determine whether the calculated metrics values are above or below the thresholds. The rules also help the analyzer to know the level of coupling, cohesion, inheritance and complexity in the analyzed software.

The user interface displays the result of the source code analyses and the result is stored as a file in the file system

.

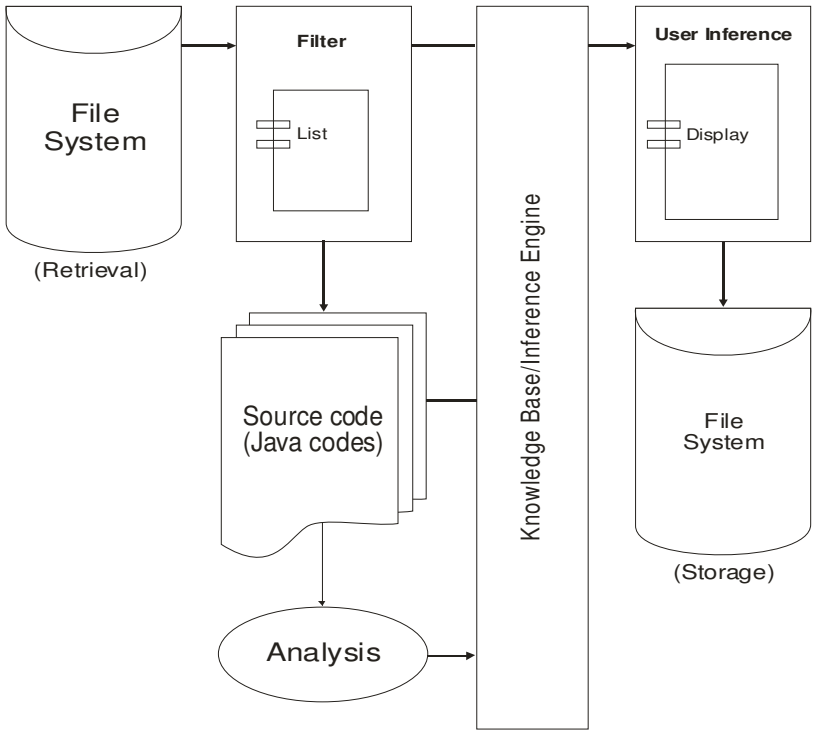


Fig. 1 – Software Analyzer's Architecture

The language used in developing the software analyzer is Java, which is an object oriented language. The class diagram in Figure 2 is used to describe the structure of the source code of the software analyzer showing the classes, attributes, operations, composition, inheritance and the relationships between classes.

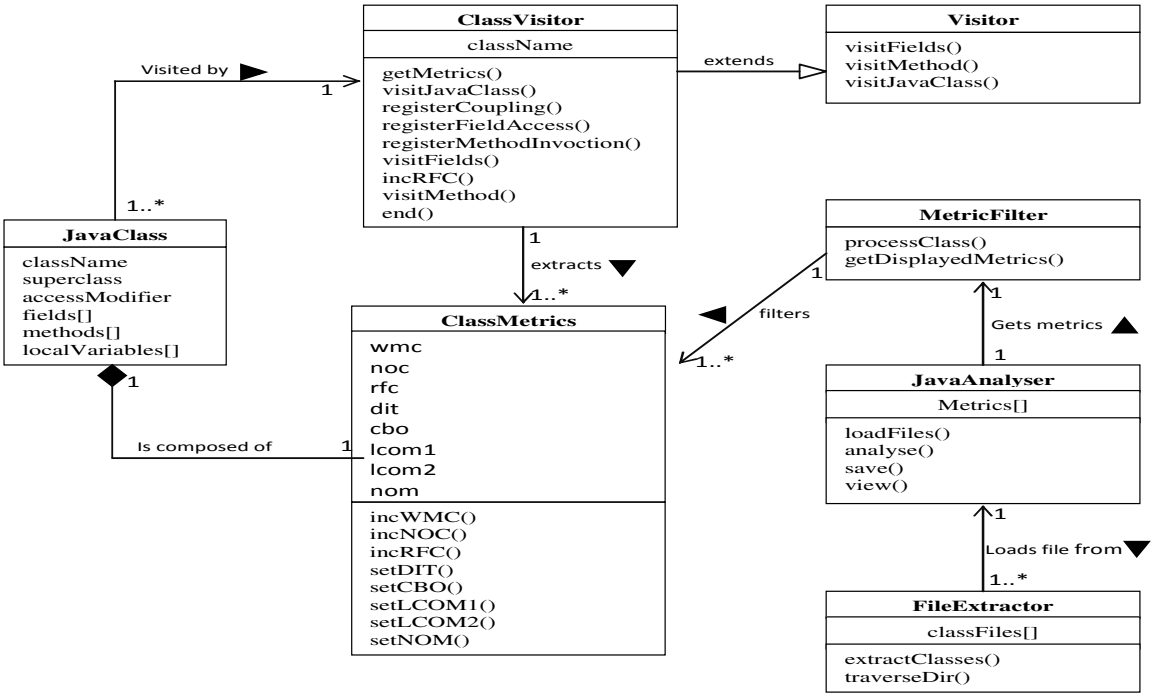


Fig. 2 - Class Diagram Showing Association between Classes in the Analyzer's Source Code

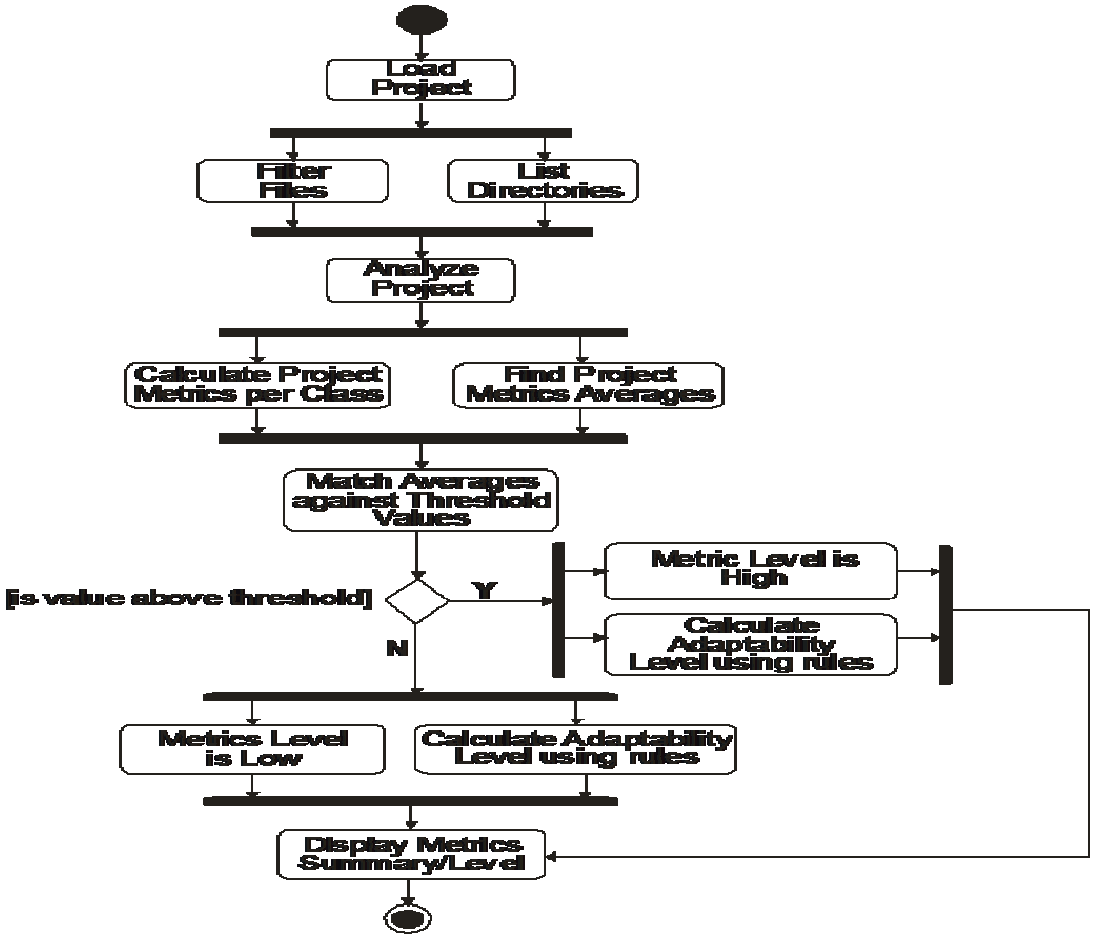


Fig. 3 - Activity Diagram Showing the Behaviour of the Software Analyzer

Activity diagram in Figure 3 is used to show the behaviour of the different modules of the software analyzer.

DECISION RULES

To determine whether the level of these attributes in a given software is above or below the threshold, rules are employed. Low (0) is for value below the threshold and high (1) is for the value above the threshold. The rules to determine attribute Levels are as follows:

(a) Low Coupling will occur only when CBO $\leq$ 5 and RFC $\leq$ 100. Other conditions will yield High Coupling.
(b) High Cohesion will occur only when LCOM1 $\leq$ 1 and LCOM2 $\leq$ 2. Other conditions will yield Low Cohesion.
(c) Low Inheritance will occur only when DIT $\leq$ 6 and NOC $\leq$ 6. Other conditions will yield High Inheritance
(d) Low Complexity will occur only when WMC $\leq$ 100 and NOM $\leq$ 20. Other conditions will yield High Complexity.

IMPLEMENTATION

Source codes of 40 open source software written in java were analyzed using the developed analyzer in order to retrieve the required software metrics such metrics are WMC, NOC, RFC, CBO, DIT, LCOM1, LCOM2 and NOM. A typical screen showing the analysis process is depicted in Figure 4. During this process, the averages of all the software metrics are also calculated by dividing the sum of all the metrics in each class by the total number of classes. The summary of the analysis for each metrics where the level of coupling, cohesion,

inheritance and complexity is highlighted is shown at the end of each analysis. Figure 5 shows the metrics summary screen.

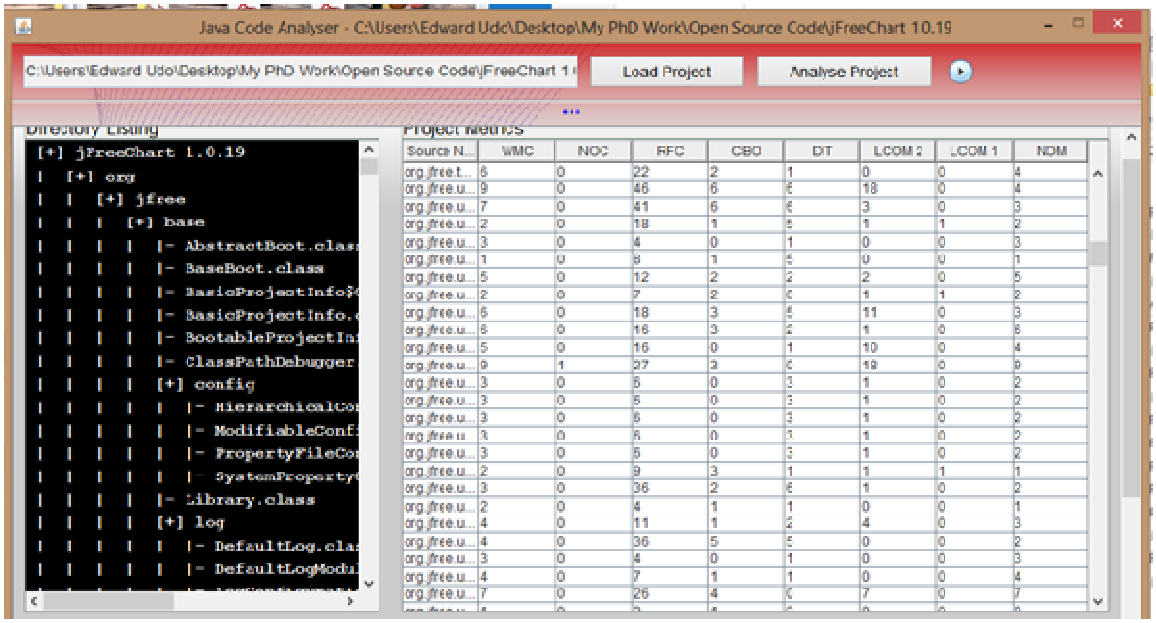


Fig. 4 - Software Analysis Process

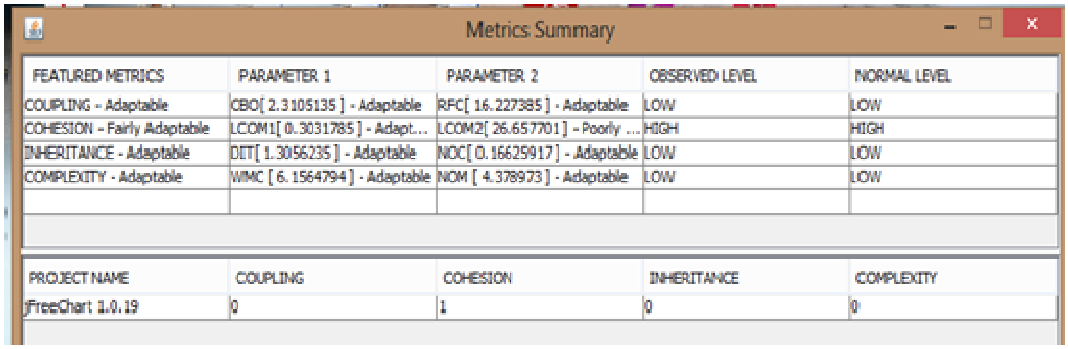


Fig. 5 – Metrics Summary

The names of the 40 open source codes that were analyzed together with their threshold level for coupling, cohesion, inheritance and complexity are shown in Table 2.

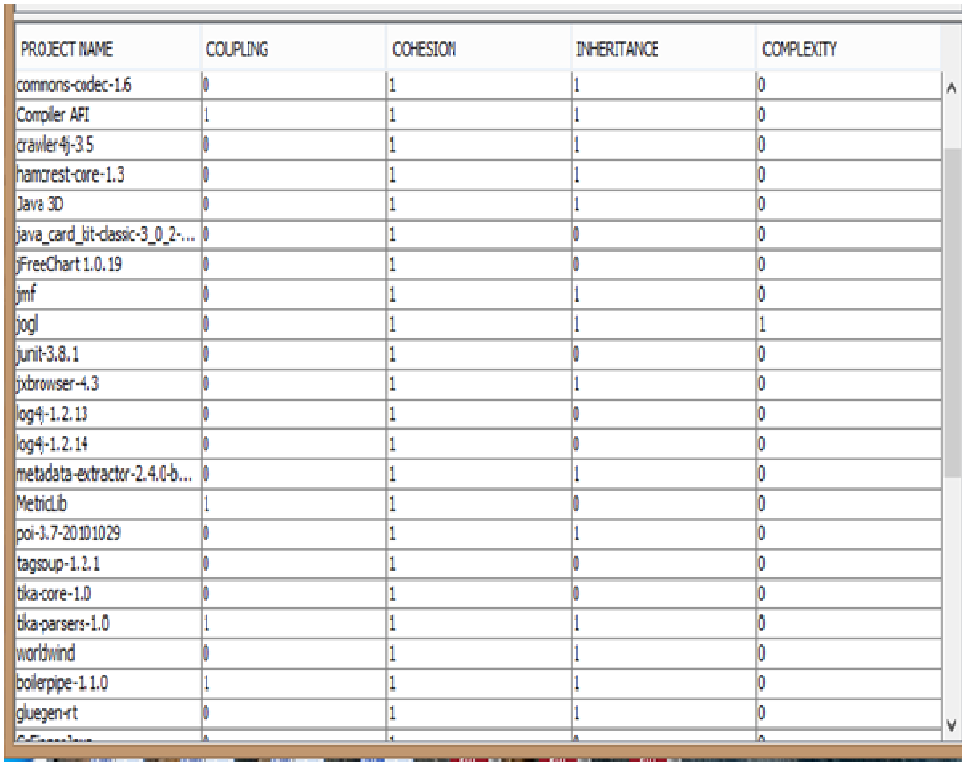
VALIDATION OF ANALYZER'S RESULTS

In order to be sure of the accuracy of the result of the software analyzer used in this work, its results was compared with the results of other software metrics tools (analyzers) which are capable of deriving metrics for Java programs.

Since most metrics tools are not readily downloadable and not all those available are freeware and even commercial versions do not provide suitable evaluation licenses, the results of metrics tools used for comparison were gotten from the work of Lincke *et al*., (2008). In their work, "Comparing Software Metrics Tools", metrics values were calculated for 3 software systems developed with Java programming language using 9 metrics tools. These metrics tools individually calculated the metrics values for the 3 software systems.

The criteria for the selection of these metrics tools, as carried out by Lincke *et al*., (2008) focused on language (Java), metrics they calculated (well known object oriented metrics) and the license type (freely available or evaluation licenses).

Table 2 - Analyzed Source Codes and their Threshold Values Level



PROJECT NAME	COUPLING	COHESION	INHERITANCE	COMPLEXITY
commons-codec-1.6	0	1	1	0
Compiler API	1	1	1	0
crawler4j-3.5	0	1	1	0
hamcrest-core-1.3	0	1	1	0
Java 3D	0	1	1	0
java_card_jit-classic-3_0_2-...	0	1	0	0
jFreeChart 1.0.19	0	1	0	0
jmf	0	1	1	0
jogl	0	1	1	1
junit-3.8.1	0	1	0	0
jxbrowser-4.3	0	1	1	0
log4j-1.2.13	0	1	0	0
log4j-1.2.14	0	1	0	0
metadata-extractor-2.4.0-b...	0	1	1	0
MetricLib	1	1	0	0
poi-3.7-20101029	0	1	1	0
tagsoup-1.2.1	0	1	0	0
tika-core-1.0	0	1	0	0
tika-parsers-1.0	1	1	1	0
worldwind	0	1	1	0
boilerpipe-1.1.0	1	1	1	0
gluegen-rt	0	1	1	0

The software systems selected for their work were jaim, jTcGUI and ProGuard. The metrics values of the software systems gotten by the different metrics tools are displayed in Table 3.

Table 3 - Differences Between Metrics Tools for Analyzed Software

PROJECTS	ANALYZERS	CBO	RFC	LCOM1	LCOM2	DIT	NOC	WMC	NOM	LOC
	Analyst4j	3.46	14.48	0.50	-	1.63	0.67	10.02	5.65	-
	CCCC	2.83	16.39	23.57	-	0.48	0.67	-	6.09	-
	C & K Java Metrics	4.04	-	-	-	0.87	0.67	-	5.65	-
	Dependency Finder	-	-	-	-	1.67	0.67	-	6.61	28.13
	Eclipse Metrics Plugin 1.3.6	-	-	-	0.17	1.91	0.67	9.48	5.46	-
Jaim	Eclipse Metrics 3.4	-	-	9.31	0.26	-	-	9.80	-	-
	Semmle	-	11.13	21.20	0.52	-	0.67	-	5.00	46.80
	Understand for Java	4.50	-	39.44	-	1.91	0.67	-	5.65	68.59
	VizzAnalyzer	2.63	7.59	16.37	-	0.67	0.67	7.15	4.48	73.24
	AdaptAnalyzer	2.52	14.02	22.13	0.29	0.50	0.67	9.69	4.67	-
	Analyst4j	17.00	73.60	0.90	-	2.00	0.20	24.00	12.80	-
	CCCC	8.00	55.20	165.80	-	3.00	0.20	-	15.80	-
	C & K Java Metrics	8.20	-	-	-	1.00	0.20	-	9.20	-
	Dependency Finder	-	-	-	-	1.20	0.20	-	19.60	108.20
	Eclipse Metrics Plugin 1.3.6	-	-	-	0.66	4.40	0.20	22.00	12.60	-
jTcGUI	Eclipse Metrics 3.4	-	-	2.00	0.65	-	-	22.20	-	150.40
	Semmle	-	57.40	97.00	0.90	2.60	0.20	-	12.60	200.60
	Understand for Java	17.60	-	82.20	-	4.40	0.20	-	12.80	204.80
	VizzAnalyzer	1.00	15.60	86.40	-	0.20	0.20	19.60	11.80	
	AdaptAnalyzer	8.19	24.07	59.22	0.69	2.21	0.20	22.14	12.80	
	Analyst4j	6.78	25.04	0.46	-	1.48	0.48	19.93	9.04	
	CCCC	8.71	20.97	70.64	-	1.62	0.44	-	9.09	
	C & K Java Metrics	14.75	-	-	-	1.02	1.50	-	8.63	57.00
	Dependency Finder	-	-	-	-	1.50	0.44	-	9.40	-
	Eclipse Metrics Plugin 1.3.6	-	-	-	0.19	1.67	0.48	18.59	8.50	-
ProGuard	Eclipse Metrics 3.4	-	-	4.17	0.20	-	-	18.07	-	92.35
	Semmle	-	16.94	71.57	0.42	2.14	1.49	-	7.90	143.62
	Understand for Java	9.40	-	34.88	-	1.55	1.49	-	8.14	149.88
	VizzAnalyzer	6.65	15.69	52.77	-	0.95	1.49	12.75	7.73	-
	AdaptAnalyzer	6.71	18.43	4.71	0.29	1.57	0.48	18.86	8.43	-

The 3 software systems were also downloaded from http://sourceforge.net/projects/jaim, http://sourceforge.net/projects/jtcgui and http://sourceforge.net/projects/proguard respectively. Their source codes were extracted and metrics values computed using the analyzer developed in this work (AdaptAnalyzer). The values were also entered in Table 3 for comparison.

It is quite obvious that the metrics values calculated by these tools differ significantly for some metrics in all the test systems. Lincke *et al*., (2008) concluded that existing software metrics tools interpret and implement the definitions of object-oriented software metrics differently.

The metrics values computed by Adapt Analyzer, as can be observed from Table 3, are within the ranges of values gotten by other metrics tools. For instance, values of NOC for jaim and jTcGUI are 0.67 and 0.20 respectively for all the metrics tools. The value of NOC for ProGuard using AdaptAnalyzer is 0.48 which tallies with the values by three other metrics tools. Also the value of NOM for jTcGUI using AdaptAnalyzer is 12.80 which is the same with the value obtained by Understand for Java and Analyst4j metrics tools.

Generally, Adapt Analyzer's values for CBO, RFC, LCOM1, LCOM2, DIT, WMC and NOM for all the 3 software projects fall within the minimum and maximum values. For example, AdaptAnalyzer's value for CBO for ProGuard project is 6.71 which falls between the minimum and maximum values of 6.65 and 14.75 respectively gotten by VizzAnalyzer and C&K Java Metrics.

CONCLUSION

This work used eight object oriented metrics, two for each internal software property, to ensure proper computation of the values for the internal properties. The 8 metrics threshold values are used as pointers to identify the quality of software. The software analyzer developed in this work incorporated rules formulated in conjunction with the threshold values level for each of the internal software property. The result from our analyzer is robust and reliable when compared with other metrics tools.

From Table 2, the values of the threshold level for each of the analyzed software can be seen. For instance, the software, **jmf**, has the values of 0, 1, 1, 0 for coupling, cohesion, inheritance and complexity respectively. This shows that the inheritance level in the software is high, which is not desirable. The code developer can therefore make adjustments in the inheritance hierarchy of the classes in the source code to ensure quality.

Generally, all the source codes with values 1 for coupling; 0 for cohesion; 1 for inheritance and 1 for complexity should be adjusted for enhanced quality.

REFERENCES

Anthony, P. (2013): Predicting Reliability of Software using Thresholds of CK Metrics. *International Journal of Advanced Network Applications*. 4: 1778 -1785.

Baggen, R., Corrina, J., Schill, K., Visser, J. (2012): Standardized Code Quality benchmarking for Improving Softyware Maintainability. *Software Quality Control*. 20(2): 287 – 307.

Bansiya, J. and Davies, C. G. (2002): A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transactions on Software Engineering*, 28 (1)

Basili, V., and Melo, W. (1996): A validation of Object Oriented Design Metrics as Quality Indicators, *IEEE Transactions on Software Engineering*, 22 (10)

Benlarbi, S., Emam, K., Goel, N., and Rai, S. (2000): Threshold for Object Oriented Measures. NCCR. *Proceeding of the 11th International Symposium on Software Reliability Engineering, IEEE Society, Washington DC, USA*, pp 24 – 37.

Chandra, E. and Linda, P. (2010): Class Break Point Determination using CK Metrics Thresholds. *Global Journal of Computer Science and Technology*. 10(14). Pp 73 – 77.

Elish, K. and Alshayeb, M. (2012): "Using Software Quality Attributes to classify Refactoring to Patterns". *Journal of Software, (*7) 2. Pp 408 – 419.

Fenton, N. E and Pfleeger, S. L. (1996): Software Metrics, a Rigorous and Practical Approach. *International Thompson Computer Press, London*. 423P.

Fenton, N. E. (1991): Software Metrics, A Rigorous Approach, *Chapman and Hall, London*, 2nd edition. 600P

Filo, T., Bigonha, M., Feireira, K. (2015): A Catalogue of Thresholds for Object-Oriented Software Metrics. *In proceedings of the First International Conference on advances and Trends in Software Engineering*. SOFTENG 2015. Pp 48 – 55

Herbold, S., Grabowski, J., and Waack, S. (2010): Calculation of Optimization of Thresholds for sets of Software Metrics. *Technical Report No.* IFI-TB-2010-01, ISSN 1611 – 1044, Gottingen, Germany.

ISO/IEC (2005): ISO/IEC Standard No. 9000. International Organization for Standardization (ISO)/*International Electrotechnical Commission (IEC), Geneva, Switzerland*.

Kayarvizly, N. and Kenmani, S. (2011): Analysis of Quality of Object-Oriented Systems using Object-Oriented Metrics. *In proceeding of International conference on Electronics Computer Technology*. Pp 203 – 206.

Lincke, R., Lundberg, J. and Lowe, W. (2008): Comparing Software Metric Tools. In proceedings of the 2008 *International Symposium on Software Testing and Analysis, Seattle, Washington, USA*. Pp 1 – 10.

Rosenberg, L., Hammer, T. and Shaw, J. (1998): Software Metrics and Reliability. 9th *International Symposium on Software Reliability, Germany*.

Singh, S. and Kahlon, K. (2014): Object-Oriented Software Metrics Threshold Values at Quantitative Acceptable Risk Level. *CSI Transactions on ICT.* 2(3): 191-205.

Yang, H., Chen, R., and Liu, Y. (2010): A Metrics Method for Software Architecture Adaptability. *Journal of Software* 5(10), pp 1091-1098