PREDICTING ADAPTABILITY LEVEL OF OBJECT-ORIENTED SOFTWARE USING METRICS AND THRESHOLD VALUES

Veronica V. N. Akwukwuma[1] *and Edward N. Udo*[2]

ABSTRACT

Adaptability, a sub-characteristic of software quality has become so important in recent years and a key characteristic of well designed applications because modern software systems are expected to service swiftly revolutionizing business environments. Much effort have been invested to the development of object oriented design metrics to measure software properties such as coupling, cohesion and inheritance, but metrics to be used by software developers to predict, at the coding phase, the extent to which object oriented software adapt is needed. In this work a software analyzer has been developed using JAVA programming language to measure the values of some internal properties of object oriented software using combination of associated object oriented design metrics. Threshold values of these object oriented design metrics are grouped into levels taking into consideration the desirable values of the internal software properties. The measured values are matched against these threshold values and the level of each attribute is noted. Decisions rules are formulated and used in conjunction with binary logic combination of the possible internal software properties to aid predict adaptability level (Poorly Adaptable – 1; Fairly Adaptable – 2; Adaptable -3) of a given software. The analyzer is implemented on 12 open source JAVA projects and their adaptability levels are noted. The analysis revealed that source codes with low coupling, high cohesion, low inheritance and low complexity easily adapt to new operating environment with little or no modifications to the source codes.

[1]Department of Computer Science, University of Benin, Benin City, Nigeria, Email: vakwukwuma@yahoo.com.

[2]Department of Computer Science, University of Uyo, Uyo, Nigeria, Email: edwardudo@uniuyo.edu.ng

Keywords: adaptability level, software metrics, internal software attributes, threshold values, software analyzer.

I. INTRODUCTION

One of the fastest growing areas in the Information Technology (IT) industry is the area of Software; its products and technologies. Our modern society is becoming maximally dependent on software [19]. As many individuals, corporate organizations and even 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.

The term "quality software" is seen to comprise software products such as adaptability, completeness, maintainability and understandability [12]. Software products are intended to be adaptable. Therefore there is need to predict the adaptability level of any developed software.

Reference [11] described software quality in terms of six (6) quality attributes – Functionality, Reliability, Usability, Efficiency, Maintainability and portability. Sub-characteristics of portability are adaptability, usability, co-existence and replaceability[11]. Portability simply means the ease with which a software system can be transferred from one platform to another. It is therefore the ability of code-base feature to reuse the already existing code rather than writing new code when transferring software from one environment to another. What therefore affects portability also affects adaptability and other sub-attributes of portability. The realization of any of the sub-characteristics can be determined using metrics [31]. These sub-characteristics can be termed external quality attributes because they relate with the behavior of the software and as such cannot be directly measured but can only be measured through internal quality attributes such as coupling, cohesion, complexity etc. [21]

Measurement of external software attribute such as adaptability can be done at any phase of software development lifecycle. In this work, the measurement of adaptability is done at the source code phase. [27] investigated the result of object oriented design software metrics on fault- proneness for java applications which were empirically analyzed and tested using software tool at the source code level. [1] also analyzed the complexity of java programs, at source code level, using object oriented software metrics.

Metrics measure quantitatively the extent to which an object, system, component, method or procedure contain a given property. It assigns a value (number or symbol) to the properties of entities in the real life scenario based on distinctly formulated and defined set of rules [9]. Metrics help software experts and engineers to accurately measure and predict software processes and necessary resources needed for a project to produce the required result. It also predicts the products relevant for a software development job [2]. Software metrics are therefore used as pointers (estimators/predictors) to external quality attributes.

Metrics 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 [9]. Metrics make it possible for software engineers to measure and predict software processes and necessary resources for a project and work products relevant for a software development effort [2]. Software metrics can therefore be used as indicators (estimators/predictors) for external quality attributes.

The objective of this work is to develop and implement processes to predict adaptability level of object-oriented software using software metrics. Two metrics are used for each associated internal software properties to ensure accurate property measurement. The threshold values of these metrics are indicated and properly categorized into three levels of adaptability prediction. Rules are formulated using the threshold values to aid in proper prediction of adaptability level of object oriented software.

Since most metrics do not provide practical support to software designers and developers and lack qualitative as well as quantitative evaluation techniques [9, 1], this approach predicts adaptability level in object oriented software giving practical support to software developers.

In order to establish relationship between design construct and software quality, the influence of design constructs on software quality have been examined and revealed that cohesion, coupling, inheritance, encapsulation and complexity affect quality attributes to some extent. Software engineering experts assume that design with low coupling, high cohesion, less complexity and low inheritance leads to products that are of high quality – better testability, readability, reliability, maintainability etc [27].

The rest of this paper is organized as follows: Section 2 defines adaptability and some of its related terms. Section 3 describes internal software attributes, section 4 reviews related literature, section 5 states the research methodology, section 6 discusses the architecture of the software analyzer, section 7 shows the threshold values, section 8 formulates the decision rules and the values for determining adaptability level. Section 9 implements the software analyzer and discusses the results. Section 10 concludes the work.

II. Adaptability, its need and some terms

Adaptability as one of the properties of object-oriented software is among the most significant non-functional requirements in software [22]. Adaptability is one of the key considerations of well designed applications in recent years because present day software systems must be able to service the changing business environments rapidly, efficiently and with robust reliability over a long time interval. If a system is designed without bearing in mind adaptability, the system will degrade and malfunction in the face of continuous software requirements changes.

Several authors define adaptability differently. Reference [10] defines adaptability as "the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed". Reference [23] views adaptability as modification of behaviour in response to environment changes. Reference [24] defines adaptability as amendment of the changing requirements. Adaptability is the level of ease a system shows in being adjusted to meet a modified requirement. It therefore characterizes the ability of a system to still function properly in an event of change to new specifications or operating environment. Adaptability is the level to which a software adjust to change in its environment. It is vital for the survival, functionality and success of any software system and should therefore be optimized, even at the expense of other software quality characteristics.

For a software system to still function, it must undergo adaptive maintenance, which is the correction done on a software code or product after it might have been delivered to make it useful in a modified environment.

As time elapses, the software initial environment in which the software was developed for may change. These may include the CPU, operating platform, product characteristics etc, [26]. This environmental change is mostly due to changes in technology, organizational goal and structure, human feelings and needs, etc [16]. Other changes may be as a result of change in customer requirements, need for new software development, modification of software features, bugs and defect detection and fixing during the maintenance phase of software life cycle. Therefore, software adaptability measurement is a key concern to software designers, developers and other personnel within the software engineering community.

Adaptability metrics therefore measures how flexible and adjustable a system is to changes within its operating environment [8]. It measures the ease of change during the project development phase or in a postmortem fashion. This is the aim of this work.

To really understand what adaptability of a software system means, the definitions of the following terms are vital:

1. **System (S)** – An assembly of components connected together in an organized way and separated from its environment [4]. There is an interaction between a system and its environment.

2. **Environment (E)** – This is the space in which a system is settled. Environment is actually another system which surrounds the actual system. The subset of complete environment which interacts with the system is Relevant Environment(E_R) while the subset of the complete environment which has no interaction with the system is called the Irrelevant Environment (E_I). Environment is therefore Relevant Environment Union Irrelevant Environment.

$$E = E_R \cup E_I \qquad (1)$$

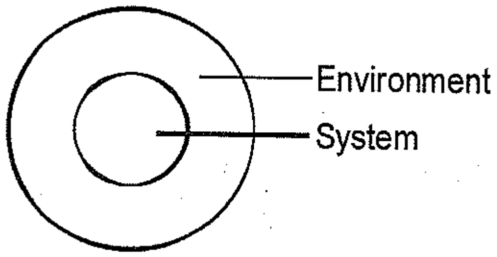


Figure 1 : Relationship between System and Environment

3. Domain (D) - The system under consideration together with the Relevant Environment forms the Domain.

$$D = S \cup E_R \qquad (2)$$

From Fig. 1, it can be deduced that System = Domain.

4. **Adaptation** - Change in the system to accommodate change in its environment. Adaptation of a software system (S) is caused by change (δ_E) from an old environment (E) to a new environment (E□) and results in new system (S□) that ideally meets the need of the new environment.

III. Internal Software Attributes

Internal software attributes are those that can be measured purely in terms of the product, process or resource itself [21]. The internal software attributes used in this work are Coupling, Cohesion, Inheritance and Complexity. Their definitions are highlighted below :

- **Coupling**: Amount of connections between classes [30].
- **Cohesion**: This is the extent to which the individual components of software are needed to perform the

same task [21]. Cohesion measures the degree of connectivity among the elements of a single class or object [6].

- **Inheritance:** This is the sharing of attributes and operations among classes based on a hierarchical relationship. Inheritance therefore occurs in all levels of a class hierarchy.
- **Complexity:** This describes the interactions between numbers of entities. As the number of entities increases, the number of interactions between them would increase exponentially, and it would get to a point where it would be impossible to know and understand all of them.

Suite of metrics by [30] is used in this study because of their wide acceptance among the engineering community [15]. Two other metrics: Number of Methods (NOM) and Lack of Cohesion in Methods 2 (LCOM2) are added to help in accurate measurement of the internal software properties.

The metrics suite by [30] consists of Coupling Between Objects (CBO), Response for a Class (RFC), Lack of Cohesion in methods1 (LCOM1), Depth of Inheritance Tree (DIT), Number of Children (NOC) and Weighted Methods per Class (WMC). Table 1 shows the metrics, their definition and their effect on adaptability.

IV. Related Work

Several software adaptability metrics have been developed using different techniques as architecture-based technique, component-based technique, dynamic adaptation technique etc.

Reference [7] proposed a component-based architecture by providing an adaptable interface to each component. The code in the adaptable interface can be changed as required to achieve the needed adaptation without making any change to the component.

Reference [22] modeled software adaptability as Non-Functional Requirement (NFR). Through this approach, consideration of design alternatives, analysis of tradeoffs, and rationalization of design decisions are all carried out in relation to the stated goals. Adaptation is supported at the architectural level by using NFRs framework which allows for decomposition of the NFR adaptability depending on the domain or the application and permits criticalities to be allocated to different NFRs of the decomposition.

Reference [16] proposed a framework called POMSAA (Process-Oriented Metrics for Software Architecture Adaptability) which aims at providing numeric scores representing the adaptability of software architecture as well as the intuitions behind these scores, utilizing Softgoal Interdependency Graph (SIG).

Reference [22] introduced two high level metrics to measure the adaptability of software. They defined an Element Adaptability Index (EAI) for each software unit. EAI was set to 1 for adaptable elements and 0 for non-adaptable elements. EAI can be measured at different levels of software granularity. Based on EAI, two metrics of Architecture

Table 1 : Metrics, Definition and their Effects on Adaptability Sources: [2], [5], [1] and [13]

PROPERTIES	METRICS	DEFINITION	EFFECT ON ADAPTABILITY
COUPLING	CBO	The number of distinct non-inheritance related classes to which a given class is coupled. It is the number of methods that can be called in response to a message in a class.	i. Too much coupling between objects prevents reuse, thereby reducing the probability of adaptability ii. The larger the number of couples the higher the sensitivity to changes and difficulty of maintenance. Difficult to maintain means less adaptability.
	RFC	Defined as |RS| where RS is the response set for a class. The response set for a class can be expressed as: RS = {M} U $\{R_i\}$ Where: $\{R_i\}$ = set of methods called by method i {M} = set of all methods in the class.	i. If the number of methods called in response to a message received by an object is large, then maintenance and testing becomes difficult which results in less adaptability ii. The larger the number of methods that can be called from a class, the greater the complexity of the class thereby reducing adaptability.

COHESION	LCOM1	The count of the number of method pairs whose similarity is zero minus the count of method pairs whose similarity is not zero. Let |P| be the number of null intersections between instance variables. Let |Q| be the number of non-empty intersections between instance variable sets, then:
		LCOM = |P| - |Q| is |P| > |Q|
		LCOM = 0 Otherwise.
			i. Cohesiveness of methods within a class is desirable because it promotes encapsulation and increase adaptability
			ii. A measure of disparateness of methods helps identify flaws in the design of classes
			iii. Low cohesion increases complexity thereby reducing adaptability level.

	LCOM2	The percentage of methods that do not access a specific attribute averaged over all attributes in the class. LCOM2 = 1 – sum (mA)/(m*a) Where: m = number of methods in class. a = number of attributes in class. mA = number of methods that access an attribute. sum(mA) = sum of mA over attributes of a class.	i. A higher value of LCOM2 indicates decreased encapsulation and increased complexity, thereby increasing the likelihood of errors. This affects adaptability negatively.
INHERITANCE	DIT	It calculates the distance to which a class is declared in the inheritance hierarchy. It also counts the number of ancestor classes that can potentially affect this class.	i. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex thus having a potential impact on adaptability. ii. Deeper trees constitute greater design complexity since more methods and classes are involved. This also affects adaptability negatively.
	NOC	It counts the number of sub-classes that are going to inherit the methods of the parent class. That is the number of immediate classes subordinated to a class in the class hierarchy.	i. The greater the number of children the greater the reuse, since inheritance is a form of reuse. This positively affects adaptability.

COMPLEXITY	WMC	Traditionally, this measures the complexity of an individual class (weighted sum of all the methods in a class). It calculates all declared methods and constructors of class. Consider a class C_1 with methods $M_1 \ldots M_n$. Let C_1 to C_n be complexity of the methods, then: $WMC = \sum_{i=1}^{n} C_i$ Where C is complexity of methods. If WMC is unity, then WMC is the number of methods per class.	i. The number of methods and the complexity of methods involved is a determinant of how much time and effort is required to maintain software thereby making it adaptable. ii. Classes with large number of methods are likely to be more application specific thereby limiting the possibility of reuse in a different environment.
	NOM	The number of methods implemented in a given class.	i. Since NOM includes methods that are declared and those that are not declared. This truly reveals software complexity. Higher the complexity, lower its likelihood to adapt.

Adaptability Index (AAI) and Software Adaptability Index (SAI) are calculated.

Considering the choice and validation of internal quality attributes, a number of researches have been done. Table 2 classifies internal quality attributes and external quality attribute(s) they affect.

Table 2 - Classification of Internal and External Quality Attributes

INTERNAL ATTRIBUTES	EXTERNAL ATTRIBUTES	SOURCE
Coupling	Maintainability	[33]
Coupling Cohesion	Maintainability	[3]
Cohesion Size metrics Complexity metric	Maintainability	[29]
C & K metrics MCC, LOC	Reusability	[25]
C & K metrics FOUT, NOM, LOC	Adaptability, Completeness, Maintainability Understandability Reusability, Testability	[14]
C & K metrics FOUT, NOM, LOC	Adaptability, Maintainability Understandability Reusability, Testability	[18]
C & K metrics FOUT, NOM, LOC	Adaptability, Completeness, Maintainability Understandability Reusability, Testability	[15]

V. Research Methodology

The method adopted to accomplish the objectives of this work follows a systematic approach :

(a) Development of software analyzer to measure the values of coupling, cohesion, inheritance and complexity of a source code using associated object oriented design metrics.

(b) Threshold values of these object oriented design metrics are grouped into adaptability levels taking into consideration the desirables values of the internal software properties.

(c) Values of the internal software properties are matched against estimated threshold values and computed for 12 different open source codes.

(d) Rules are used to formulate the level of each property in the source code and the adaptability levels of the software for appropriate predictions.

VI. Architecture Of Software Analyzer

Figure 2 shows the different components making up the software analyzer. 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 the 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 as well as the adaptability level of the software. The rules are shown in section 8. The user interface displays the result of the source code analyses and the result is stored as a file in the file system (storage).

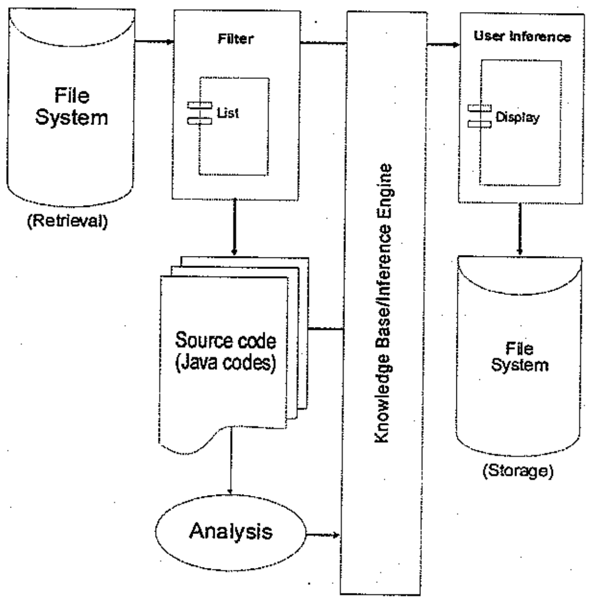


Figure 2 : Architecture of Software Analyzer

VII. Threshold Values

In order to determine the level in which a software possesses the value of the selected internal software properties, the value of each property is measured against a threshold. Low means the value is below a threshold while high means above the threshold. Thresholds are defined as "heuristic values used to set ranges of desirable and undesirable metric values for measured software. These thresholds are used to identify anomalies which may or may not be an actual problem [20]. Threshold values for the selected metrics are shown in table 3

Table 3 : Metrics Threshold values

PROPERTIES	METRICS	THRESHOLD VALUE	SOURCE
Coupling	CBO	5	[17] [28]
	RFC	100	[17] [28]
Cohesion	LCOM1	1	[5]
	LCOM2	2	[5]
Inheritance	DIT	6	[5]
	NOC	6	[5]
Complexity	WMC	100	[17] [28]
	NOM	20	[31]

VIII. 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.

A. *Determining Attribute Levels* – The rules for determining the level of Coupling, Cohesion, Inheritance and Complexity is given thus:

i. Low Coupling will occur only when CBO d" 5 and RFC d" 100. Other conditions will yield High Coupling.

ii. High Cohesion will occur only when LCOM1 d" 1 and LCOM2 d" 2. Other conditions will yield Low Cohesion.

iii. Low Inheritance will occur only when DIT d" 6 and NOC d" 6. Other conditions will yield High Inheritance

iv. Low Complexity will occur only when WMC d" 100 and NOM d" 20. Other conditions will yield High Complexity.

A. *Determining Adaptability Levels (AL) for each Attribute-* Before determining adaptability levels, the desirable condition (level) for the selected internal software properties is known. **Low coupling, high cohesion, low inheritance and low complexity** are desirable conditions for effective adaptability of object oriented software. Table 4 shows how threshold values are grouped for each level of adaptability for a particular desired condition. The threshold values for the different adaptability levels in Table 4 are used in establishing rules for adaptability levels of each of the software properties. The adaptability levels are Adaptable, Fairly Adaptable and Poorly Adaptable.

Table 4 : Threshold Ranges for Adaptability Levels

DESIRABLE FEATURE	METRICS	ADAPTABLE	FAIRLY ADAPTABLE	POORLY ADAPTABLE
Low Coupling	CBO	1 – 3	4 – 5	> 5
	RFC	1 – 69	70 – 100	> 100
High Cohesion	LCOM1	1	-	> 1
	LCOM2	0 – 1	2	> 2
Low Inheritance	DIT	0 – 4	5 – 6	> 6
	NOC	0 – 4	5 – 6	> 6
Low Complexity	WMC	1 – 69	70 – 100	> 100
	NOM	0 – 10	11 – 20	> 20

The rules are written out below:

COUPLING

If CBO = 1 to 3 and RFC = 1 to 69

Then Coupling = Adaptable

If CBO = 1 to 3 and RFC = 70 to 100

Then Coupling = Fairly Adaptable

If CBO = 1 to 3 and RFC > 100

Then Coupling = Poorly Adaptable

If CBO = 4 to 5 and RFC = 1 to 69

Then Coupling = Adaptable

If CBO = 4 to 5 and RFC = 70 to 10

Then Coupling = Fairly Adaptable

If CBO = 4 to 5 and RFC > 100

Then Coupling = Poorly Adaptable

If CBO > 5 and RFC = 1 to 69

Then Coupling = Adaptable

If CBO > 5 and RFC = 70 to 100

Then Coupling = Fairly Adaptable

If CBO > 5 and RFC > 100

Then Coupling = Poorly Adaptable

COHESION

If LCOM1 = 1 and LCOM2 = 0 to 1

Then Cohesion = Adaptable

If LCOM1 = 1 and LCOM2 = 2

Then Cohesion = Fairly Adaptable

If LCOM1 = 1 and LCOM2 > 2

Then Cohesion = Poorly Adaptable

If LCOM1 > 1 and LCOM2 = 0 to 1

Then cohesion = Adaptable

If LCOM1 > 1 and LCOM2 = 2

Then Cohesion = Fairly Adaptable

If LCOM1 > 1 and LCOM2 > 2

Then Cohesion = Poorly Adaptable

INHERITANCE

If DIT = 0 to 4 and NOC = 0 to 4

Then Inheritance = Adaptable

If DIT = 0 to 4 and NOC = 5 to 6

Then Inheritance = Fairly Adaptable

If DIT = 0 to 4 and NOC > 6

Then Inheritance = Poorly Adaptable

If DIT = 5 to 6 and NOC = 0 to 4

Then Inheritance = Adaptable

If DIT = 5 to 6 and NOC = 5 to 6

Then Inheritance = Fairly Adaptable

If DIT = 5 to 6 and NOC > 6

Then Inheritance = Poorly Adaptable

If DIT > 6 and NOC = 0 to 4

Then Inheritance = Adaptable

If DIT > 6 and NOC = 5 to 6

Then Inheritance = Fairly Adaptable

If DIT > 6 and NOC > 6

Then Inheritance = Poorly Adaptable

COMPLEXITY

If WMC = 1 to 69 and NOM = 0 to 10

Then Complexity = Adaptable

If WMC = 1 to 69 and NOM = 11 to 20

Then Complexity = Fairly Adaptable

If WMC = 1 to 69 and NOM > 20

The Complexity = Poorly Adaptable

If WMC = 70 to 100 and NOM = 0 to 10

Then Complexity = Adaptable

If WMC = 70 to 100 and NOM= 11 to 20

Then Complexity = Fairly Adaptable

If WMC = 70 to 100 and NOM > 20

Then Complexity = Poorly Adaptable

If WMC > 100 and NOM = 0 to 10

Then Complexity = Adaptable

If WMC > 100 and NOM = 11 to 20

Then Complexity = Fairly Adaptable

If WMC > 100 and NOM > 20

Then Complexity = Poorly Adaptable

A. *Determining Adaptability Level for entire Software-* The adaptability levels for a given software are Poorly Adaptable (1), Fairly Adaptable (2) and Adaptable (3). The possible combinations for levels of the properties are shown in table 5.

Table 5 – Possible Combinations for Property Level

S/N	COU	COH	INH	COM
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

Taking into consideration the desirable level for each property, table 5 is reduced to table 6 where Y (yes) replaces the desirable value and N (no) replaces the undesirable value. These values are used to determine the Adaptability Level (A.L) of software.

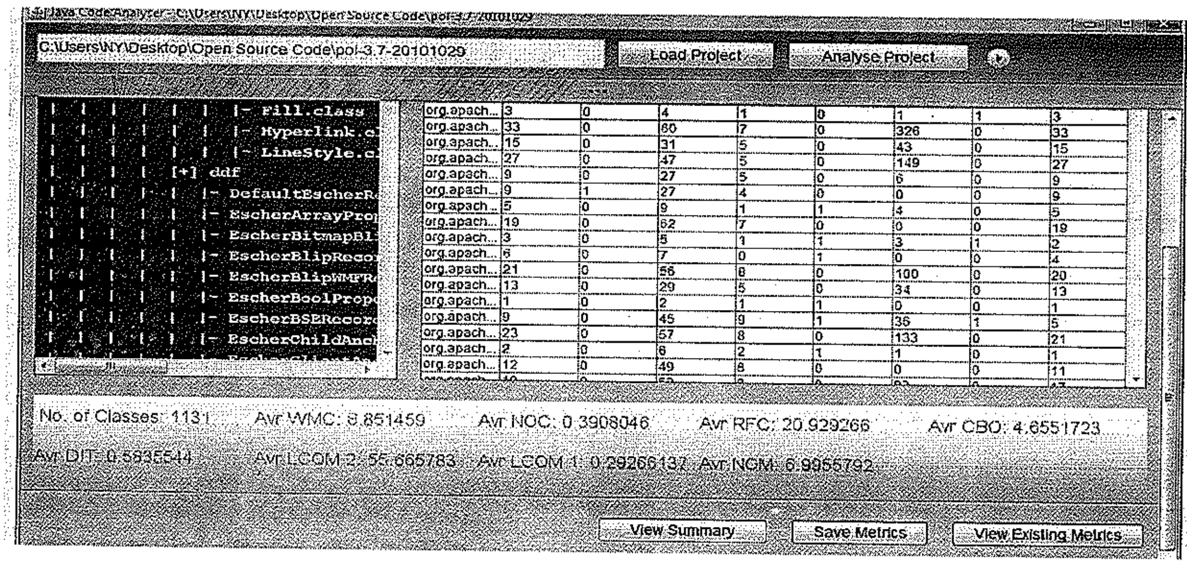
Table 6 – Values for determining A.L

S/N	COU	COH	INH	COM	A.L
1	Y	N	Y	Y	3
2	Y	N	Y	N	2
3	Y	N	N	Y	2
4	Y	N	N	N	1
5	Y	Y	Y	Y	3
6	Y	Y	Y	N	3
7	Y	Y	N	Y	3
8	Y	Y	N	N	2
9	N	N	Y	Y	2
10	N	N	Y	N	1
11	N	N	N	Y	1
12	N	N	N	N	1
13	N	Y	Y	Y	3
14	N	Y	Y	N	2
15	N	Y	N	Y	2
16	N	Y	N	N	1

IX. Implementation And Result

Twelve (12) JAVA projects were loaded into the designed software analyzer for analysis of the source codes. Information such as number of classes, average values of CBO, RFC, LCOM1, LCOM2, DIT, NOC, WMC and NOM were extracted. Figure 3 shows the analysis of one of the projects (poi-3.7-2010 1029) while figure 4 shows the metrics summary of the project, indicating the adaptability level for each of the properties in the analyzed project. Table 7 shows the displayed threshold levels for the 12 projects.

A. *Result Explanations* – Using table 6, it is seen that GrFingerJava will be adaptable to new environment with little or no modifications in the source code. The internal attributes of the source code are all below the threshold values. Jog1 will be fairly adaptable with a bit more modifications to the source code. Inheritance and Complexity values are above the threshold. The same explanations can be applied to all other projects.



Java CodeAnalyzer - C:\Users\NY\Desktop\Open Source Code\poi-3.7-20101029

C:\Users\NY\Desktop\Open Source Code\poi-3.7-20101029

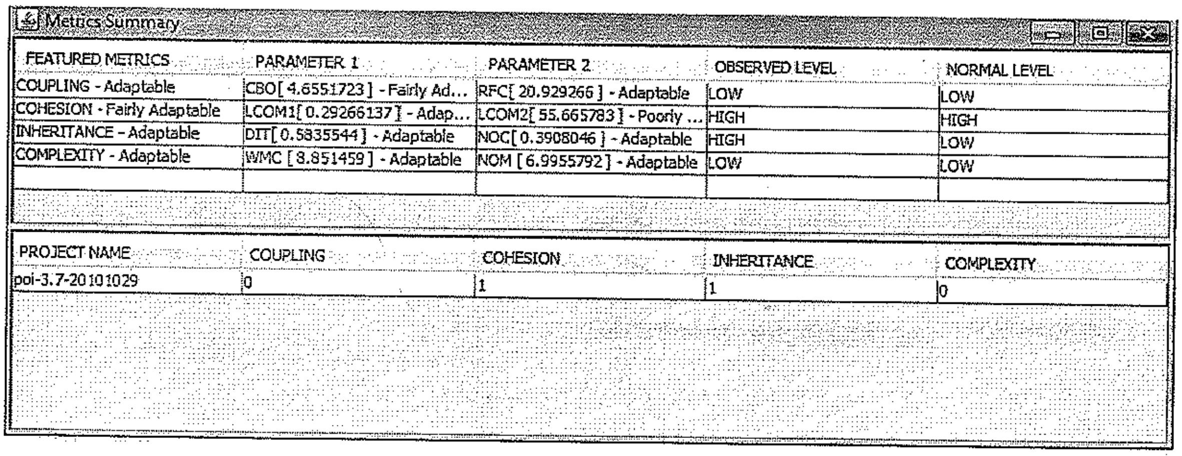
```
|  |  |  |  |  |   |- Fill.class
|  |  |  |  |  |   |- Hyperlink.c
|  |  |  |  |  |   |- LineStyle.c
|  |  |  |  [+] ddf
|  |  |  |  |  |- DefaultEscherR
|  |  |  |  |  |- EscherArrayPro
|  |  |  |  |  |- EscherBitmapBl
|  |  |  |  |  |- EscherBlipReco
|  |  |  |  |  |- EscherBlipWMFR
|  |  |  |  |  |- EscherBoolProp
|  |  |  |  |  |- EscherBSERecor
|  |  |  |  |  |- EscherChildAnc
```

org.apach...	3	0	4	1	0	1	1	3
org.apach...	33	0	60	7	0	326	0	33
org.apach...	15	0	31	5	0	43	0	15
org.apach...	27	0	47	5	0	149	0	27
org.apach...	9	0	27	5	0	6	0	9
org.apach...	9	1	27	4	0	0	0	9
org.apach...	5	0	9	1	1	4	0	5
org.apach...	19	0	62	7	0	0	0	19
org.apach...	3	0	5	1	1	3	1	2
org.apach...	6	0	7	0	1	0	0	4
org.apach...	21	0	56	8	0	100	0	20
org.apach...	13	0	29	5	0	34	0	13
org.apach...	1	0	2	1	1	0	0	1
org.apach...	9	0	45	9	1	36	1	5
org.apach...	23	0	57	8	0	133	0	21
org.apach...	2	0	6	2	1	1	0	1
org.apach...	12	0	49	8	0	0	0	11

No. of Classes: 1131   Avr WMC: 8.851459   Avr NOC: 0.3908046   Avr RFC: 20.929266   Avr CBO: 4.6551723

Avr DIT: 0.5835544   Avr LCOM 2: 55.665783   Avr LCOM 1: 0.29266137   Avr NOM: 6.9955792

Figure 3 : Typical Example of Software Analysis



Metrics Summary

FEATURED METRICS	PARAMETER 1	PARAMETER 2	OBSERVED LEVEL	NORMAL LEVEL
COUPLING - Adaptable	CBO[4.6551723] - Fairly Ad...	RFC[20.929266] - Adaptable	LOW	LOW
COHESION - Fairly Adaptable	LCOM1[0.29266137] - Adap...	LCOM2[55.665783] - Poorly ...	HIGH	HIGH
INHERITANCE – Adaptable	DIT[0.5835544] - Adaptable	NOC[0.3908046] - Adaptable	HIGH	LOW
COMPLEXITY - Adaptable	WMC [8.851459] - Adaptable	NOM [6.9955792] - Adaptable	LOW	LOW

PROJECT NAME	COUPLING	COHESION	INHERITANCE	COMPLEXITY
poi-3.7-20101029	0	1	1	0

Figure 4 : Sample Metric Summary Output

Table 7 – Threshold Values for the Projects

SOURCE NAME	COUPLING	COHESION	INHERITANCE	COMPLEXITY
GrFingerJava	0	1	0	0
Java 3D	0	1	1	0
jmf	0	1	0	0
junit-3.8.1	0	1	0	0
jxbrowser-4.3	0	1	0	0
log4j-1.2.13	0	1	0	0
MetricLib	1	1	0	0
poi-3.7-20101029	0	1	1	0
worldwind	0	1	1	0
java_card_kit-classic	0	1	0	0
javacomm20win32	0	1	0	0
jogl	0	1	1	1

X. Conclusion

Since adaptability of object oriented software has become so important in recent years, it therefore becomes pertinent to predict adaptability level of any developed object oriented software. This work has been able to predict adaptability level of 12 open source Java projects using metrics and threshold values. From this work it can be concluded that source codes with low coupling, high cohesion, low inheritance and low complexity values adapt easily to new operating environment with little or no modifications to the source codes.

References

[1] A. Chhikara and R. S. Chhillar. Analyzing the Complexity of Java Programs using Object-Oriented Software Metrics, International Journal of Computer Science Issues, Vol. 9, Issue 1, No. 3, Pp 364 – 372, 2012.

[2] A. Shaik, C. Reddy, B. Manda, C. Prakashini, and K. Deepthi.Metrics for Object Oriented Design Software Systems: A Survey, Journal of Emerging Trends and Applied Sciences (JETEAS), Vol. 1 No. 2. pp 190 – 198, 2010.

[3] B. DuBois, S. Demeyer, and J. Verelst. Refactoring - Improving Coupling and Cohesion of Existing Code.11th Working Conference on Reverse Engineering (WCRE'04), 2004, pp 144-151.

[4] B. Tekinerdogan.Modeling Adaptability in Object-Oriented Software Development, Unpublished.

[5] E. Chandra and P. Linda. Class Break Point Determination using CK Metrics Thresholds, Global Journal of Computer Science and Technology, Vol.10, Issue 14, pp 73 – 77, 2010.

[6] G. Booch, Object-Oriented Analysis and Design with Applications, 2nd ed., Benjamin Cumming, 1994.

[7] G. Heineman. Adaptation and Software Architecture. Proceedings of the 3rd International Workshop on Software Architecture, Orlando, Florida, USA, March 1999, pp 61 – 64.

[8] H. Helvajian, Microengineering Aerospace Systems, California: Aerospace Press, 1999.

[9] H. Yang, R. Chen, and Y. Liu. A Metrics Method for Software Architecture Adaptability, Journal of Software, Vol.5, No. 10, pp 1091-1098, 2010

[10] IEEE Standard 610.12. IEEE Standard Glossary of Software Engineering Terminology, New York: Institute of Electrical and Electronics Engineers, 1990.

[11] ISO/IEC Standard No. 9126. Software Engineering Product Quality, Part 1 – 4, Geneva, Switzerland: International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC), 2001 – 2004.

[12] I. Summerville, Software Engineering: Design Reliability and Management, New York: McGraw Hill Inc, 1996, Pp 81 – 88.

[13] J. Hogan, An Analysis of O O Software Metrics, Unpublished.

[14] K. Elish, and M. Alshayeb. A classification of Refactoring Methods based on Software Quality Attributes, The Arabian Journal of Science and Engineering, Vol. 36, 2011

[15] K. Elish, and M. Alshayeb. Using Software Quality Attributes to classify Refactoring to Patterns, Journal of Software, Volume 7 No. 2, Pp 408 – 419, 2012.

[16] L. Chung, and N. Subramanian. Process-oriented metrics for software architecture adaptability. In Proceedings of 5th IEEE International Symposium Requirements Engineering, 2001 pp 310 -311.

[17] L. Rosenberg, T. Hammer, and J. Shaw. Software Metrics and Reliability. 9th International Symposium on Software Reliability, Germany, 1998.

[18] M. Alshayeb. Empirical Investigation of Refactoring Effect on Software Quality, Information and Software Technology Journal, Vol. 51, pp 1319 – 1326, 2009

[19] M. Khaliq, R. Khan, and M. Khan. Significance of Design Properties in Object Oriented Software Product Quality Assessment, International Journal of Computing Science and Communication Technologies, Vol. 3, No. 2, pp 1- 4, 2011

[20] M. Lorenz and J. Kidd. Object Oriented Software Metrics, Eaglewood Cliffs, New Jersy, USA: Prentice Hall, 1994.

[21] N. Fenton, and S. Pfleeger. Software Metrics, a Rigorous and Practical Approach, London: International Thompson Computer Press, 1996.

[22] N. Subramanian, L. Chung. Software Architecture Adaptability: An NFR approach. In Proceedings of the 4th International Workshop on Principles of Software Evolution, 2001, Pp 52 – 61.

[23] P Oriezy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quinlici, D. Rosenblum, and A. Wolf. An Architecture-Based Approach to Self-Adaptive Software, IEEE Intelligent Systems, pp 54 – 62, 1999.

[24] Report on Adaptability in Object-Oriented Software Development Workshop, 10th European Conference on object-Oriented programming, Linz, Austria, July 8 – 12, 1996.

[25] R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi.Does refactoring improve reusability? 9th International Conference on Software Reuse (ICSR'06), 2006, pp 287-297.

[26] R. Pressman, Software Engineering – A Practitioner's Approach, 4th Edition, New York: McGraw-Hill Companies Inc, 2005.

[27] S. Amjan, N. Satyanarayana, M. Huzaifa, N. Shaik, M. Naveed, S. Rao and C. Reddy. Investigating the Result of Object Oriented Design Software Metrics on Fault Proness in Object Oriented Systems: A case study, Journal of Emerging Trends in Computing and Information Sciences, Vol 2 No. 4, pp 201 – 208, 2011.

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

[29] S. Bryton, and F. Abreu. Strengthening refactoring: towards software evolution with quantitative and experimental grounds, presented at the 4th International Conference on Software Engineering Advances, Porto, 2009.

[30] S. Chidamber, and C. Kemerer. A Metrics suite for Object Oriented Design, IEEE Transactions on Software Engineecrig, Vol. 20 No. 6 Pp 476 – 493, 1994

[31] S. Herbold, J. Grabowski and S. Waack. Calculation of Optimization of Thresholds for sets of Software Metric. Technical Report No. IFI-TB-2010-01, ISSN 1611 – 1044, Gottingen, Germany. 2010.

[32] W. Frakes, and C. Terry. Software Reuse: Metrics and Models, ACM computing Surveys, Vol. 28, No. 2, pp 415 – 435, 1996

[33] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya. A Quantitative Evaluation of Maintainability Enhancement by Refactoring. International Conference on Software Maintenance (ICSM'02), 2002, pp 576-585.

AUTHOR'S BIOGRAPHY

Dr. (Mrs.) V.V.N. Akwukwuma is an Associate Professor in the Department of Computer Science, Faculty of Physical Sciences, University of Benin, Benin City. Nigeria. Her research has focused on Software Engineering particularly in the areas of Software Metrics and software security. She is currently a member of MAN, CPN, NCS, INWES and OWSD.

Edward N. Udo is a PhD student at the University of Benin, Benin City, Nigeria. He received B.Sc and M.Sc degree in Computer Science from the University of Uyo, Uyo, Nigeria and University of Port Harcourt, Port Harcourt, Nigeria respectively. He is currently a lecturer at the department of Computer Science, University of Uyo, Uyo, Nigeria. His main research interests include among others object-oriented software engineering, software metrics and measurement. He is a member of NCS, CPN and IEEE.