

Three Levels Quality Analysis Tool for Object Oriented Programming

Mustafa Ghanem Saeed¹, Kamaran HamaAli Faraj⁴

Computer Science Dept, College of Science
Cihan University-Slemani
Slemani, Iraq

Maher Talal Alasaady²

Computer system Dept,
Northern Technical University,
Mosul, Iraq

Fahad Layth Malallah³

Computer and Information Technology,
College of Electronic Engineering,
Ninevah University, Mosul, Iraq

Abstract—In terms of evolution of software engineering methods for complex software developments techniques, new concepts have been emerged in the software languages, which used to develop software quality models. In this research, the Multi Levels Quality Analysis Tool (MLQA) is proposed as a tool for computer-aid software engineering, which classifies software complexity into three levels of analysis, namely the program package analysis, class analysis (program class) and finally the analysis at the level of the program method. MLQA is able to support a method of visual analysis of the software contents with color alerts, and recommendations systems, which can give a quick view of the software development and its complexity. The methodology of this work is a new suggested software quality model based on the standards object-oriented programming complexity metrics as well as threshold limits. In addition, a new quality attribute namely clean code attribute has been proposed and integrating it with the proposed software quality model in a way that enables the user of the model relies on this attribute and reduces the dependence on the software experience, which is expensive and rare at times.

Keywords—Software quality models; software measurements; clean code; source code complexity metrics

I. INTRODUCTION

The need for software quality has been extensively continuous because of the increasing in the community need for software in all aspects of life. The importance of providing the highest quality standards is no longer an advantage, but it is necessary for companies to be successful and competitive [1, 2]. Therefore, there is a unified agreement on the need for software quality, and a number of software quality models and software measurements appeared to solve this problem [2]. Everything in life, if it can be measured, gives the ability to deal with it smoothly, so the importance of software measurements provides indications on the quality and strength of software [3,4]. In addition, the level of development and improvement in the quality models to show the enhancement and defects in the software from one version to another is an important area to be studied [5]. The concept of the clean code is used to distinguish the quality of the code in terms of readability, understanding, structure and complexity [6-9]. All

these characteristics are difficult to be identified by depending only on the experiences of programmers and developers [8]. It is necessary to come out with approaches that solve these problems more effectively than the experience of programmers, which may not be available at any time or expensive for the software developer.

The contribution of this paper is proposing a general software quality model to give more flexibility and control during the development of software by making the proposed quality model work on three levels during the analysis of the software product. The proposed model can also display visual chart indicators with recommendations in case some expected defects in software product are found. For this purpose, a tool has been built to measure the quality of the software with an object-oriented programming that measures the quality at three levels: package, class, and function (Method). At the package level, the tool gives the package details in terms of complexity measures and threshold limits on the aspect and quality attributes of Bansya quality model [4, 20] normalized by threshold value to explain high level of abstraction in software. The Class level has been treated with the extraction of special measures that can be extracted from the class only. With comparison to the appropriate threshold limits, the quality of this class can be judged. At low level of source code (Function level), a quality attribute has been proposed for clean code, which is a set of parameters that are only within the function limits, by which the function can be classified as either having the clean programming code attribute or not. Moreover, the proposed software quality model suggested using of visual color alerts, and recommendations systems in order to help software engineer in the process of evaluating software product.

The organization of the paper is as follows: Section 2 covers the literature review regarding overall previous works on quality models and source code matrix, Section 3 explains the proposed software quality model. While in Section 4 presents the experiment details of this research with the results and discussions. Finally, Section 5 concludes this research with the possible future work.

II. LITERATURE REVIEW

This section reviews the previous works related to the most widely used software quality model to determine the ambiguity of the concept of software quality and the scope of those models in real applications developments. In addition, these works focus on the complexity of the source code (Source Code Complexity Metrics) in terms of the use mechanisms, analysis, and the appropriate threshold for each metrics.

In 2006, Gitter conducted a study on how to apply the Bansya software quality model to evaluate the development of 19 Azureus versions by making comparisons using object-oriented metrics rather than the metrics proposed by Bansya and his colleagues. The researcher demonstrated the ability of this model to track the evolution of design quality in several versions by providing access to important information in the internal life of the software. This information can support design decisions at higher levels of abstraction. His proposed model may require additional inputs to cover the highest levels of abstraction to assess all aspects of the quality model at this level [10]. Another study by Panfowski (2008) presented a new assessment of software product quality, which focused on assessing the quality of the external features of the software product, which means evaluating the behavior of the software product when implemented. In addition, the study focused on the development of the quality model (ISO / IEC 9126) at the level of software metrics. The study relied on seven samples of the software product and evaluated them using ISO / IEC 9126-2 quality model. In his work, Panfowski concluded that external product quality attributes are an area or category that can be adopted, and that the metrics provided by ISO / IEC 9126-2 can be considered as a starting point for the definition of standards, but are not ready to use in their present form. The metrics of the software product need to be more adapted to show better information [11]. Borgherth (2008) discussed the method of code profiling by using a static analysis. The study was done on (19) industrial samples and (37) samples of students' programs. He has analyzed software samples through software metrics. The results of this study indicated that the code pattern could be a useful technique for rapid program comparisons and quality observation in the field of industrial application and education [12]. Moreover, Bhatti (2010) explored the area occupied by the software metrics. He used a QA-C tool to measure software metrics automatically on the code written in C programming language through expressing the association between software metrics and the complexity of the source code. He attempted to demonstrate the values of these metrics graphically only, without considering the quality features and threshold limits relationship [13]. Another work in 2010 is the impact of code complexity and usability, either in monitoring software complexity during development, or in evaluating the complexity of legacy software. The researchers of this work, Goran and Dahiden proposed a new coupling metrics (Ecoup), and introduced the Java met tool, which works in a static analysis of programs written in Java with respect to coupling, flow control, complexity and coherence [14]. In the same year, Chandra et al. proposed the use of Object Oriented metrics that introduced by Chidamber and Kemerer (1994)[15] to assess program quality at the class level. The proposed tool

can be used to verify the class design conforms to the design specifications of the Object Oriented programming, through using the threshold for each metric [16].

The following is a summary of the most important software quality models:

1) McCall's Factors-Criteria (FCM) model presented by McCall in 1977. The McCall model is the first model of quality [1, 10, 17, 18].

2) ISO / IEC 9126: 2001, which was submitted by the International Organization for Standardization (ISO) in 1991, and has developed six quality metrics. This model was updated in 2001, Quality ISO / IEC 9126: 2001 [10, 13, 18, 19].

3) Dormey's Quality Model, presented by Jeff Dormy (1998) as a quality assessment model, by analyzing the quality of program components through measuring concrete quality characteristics [4, 10].

4) Boehm Software Quality Model presented by the scientist Erwin Bohm in 1978. This model seeks to determine the quality of the program through a predefined set of metrics and measures [10, 11, 19].

5) FURPS Quality Model, introduced by Grady Robert and Hewlett-Packard in 1987. This model focuses on the analysis of quality characteristics in two categories of requirements: functional requirements and non-functional requirements [11, 17, 20].

6) Bansya Quality Model, proposed by Bansya in 2002. This model focuses on the quality of Object Oriented Design (QMOOD). It uses the source code metrics extract directly from the software source code to give the quality attribute through the use of mathematical equations [1, 10].

The Bansya Quality model gives a way to assign source code measurements to higher abstraction levels [10]. Although the experiment results in this model were acceptable, the use of new and non-standard measures in OOP metrics makes this model not widely used, and this is why Gitter in [20] tried to change the measures used in this model to the stranded OOP metrics so that the model becomes more dependable. For this reason, Bansya model with stranded OOP metrics was used at the software packages analysis stage in the proposed model. Standard OOP metrics used in the proposed model can be viewed in [6, 10, 14, 16, 21-27, 29]. Furthermore, the metrics threshold values that used for the recommendations and alerts are selected of this proposed model due to their usage in the following references and full described in [9, 16, 23, 27-29], which are the reason of the selection.

III. THE FRAMEWORK OF PROPOSED QUALITY MODEL

The previous quality models discussed in section II were based on building a relationship between software metrics and design features on the one hand, and design features and quality attributes on the other hand. The relationship could be direct or indirect, either through a paper questionnaire (specific questions prepared by the quality model) or by creating a mathematical relationship with each other as in the Bansya software quality model. Actually, the idea of using a

mathematical relationship between source code metrics extracted directly from the software and the quality attribute is better than using indirect methods such as questionnaires, because indirect methods take a lot of time and a deep knowledge from a software engineer in the project under development. Moreover, it requires a lot of paperwork that exhausts the software engineer. In addition, the analysis process may involve human feelings that affect the accuracy of the judgment.

Therefore, the proposed software quality model suggests four ideas for the analysis of the software product quality: Firstly, the analysis process should be automatic (or semi-automatic) and applied directly on the software source code to reduce time, effort and cost on the software developer and to reduce human errors. Secondly, the software quality model should give informative details of the software product under developments, such as quality attributes to higher abstraction levels, intermediate structure and low-level details. All of these levels must be supported by error detection, alerts, and recommendations systems, because some metrics are just numbers (e.g. line of code) and knowing their impact in software product may require strong software engineering experience. Thirdly, the quality models should use a visual representation with color indicators to demonstrate analysis result that gives a quick full picture of the software product. This enables the software engineer to diagnose the location of the strengths and the weaknesses in the software product under development. Fourthly, the quality models support the ability to view the source code for classes and functions while displaying their own measurements and recommendations.

The proposed model is based primarily on the principles of OOP, which is currently used in almost all software systems,

to make the proposed quality model close to the need of developers and programmers. The proposed model singles out the software under evolution into three parts as shown in Fig.1 which includes (Use cases) tool for quality and actor model (Actor), which represents the end-user model. The idea of dividing the software is to enable the developer team to correct errors and reduce complexity as much as possible in the next development cycle or the rewriting of the software code as in the extreme programming development approach.

The three parts of the analysis of software product are as follows:

1) Quality attributes that affect the quality of packages in general. In this level of analysis, the proposed model suggests using version of Bansya Quality Model (with proposed modification), because it is very informative in this level.

2) The quality attributes that affect the quality of the class. The proposed model used the set of metrics suitable for object-oriented design suggested by Gadabber and some other measures of the OOP and linking them with the metrics thresholds to generate an evaluation and recommendations for the source code of the class.

3) Quality attributes that affect the quality of the functions (Methods) in the class. The proposed model suggested new attributes of the clean code. Through this attribute, the user can evaluate the quality of the function.

The software metrics, especially the complexity metrics of the source code, should be classified according to the three levels of analysis of the proposed quality model as shown in Fig. 2, which describes the classification of these metrics according to their impact on quality aspects.

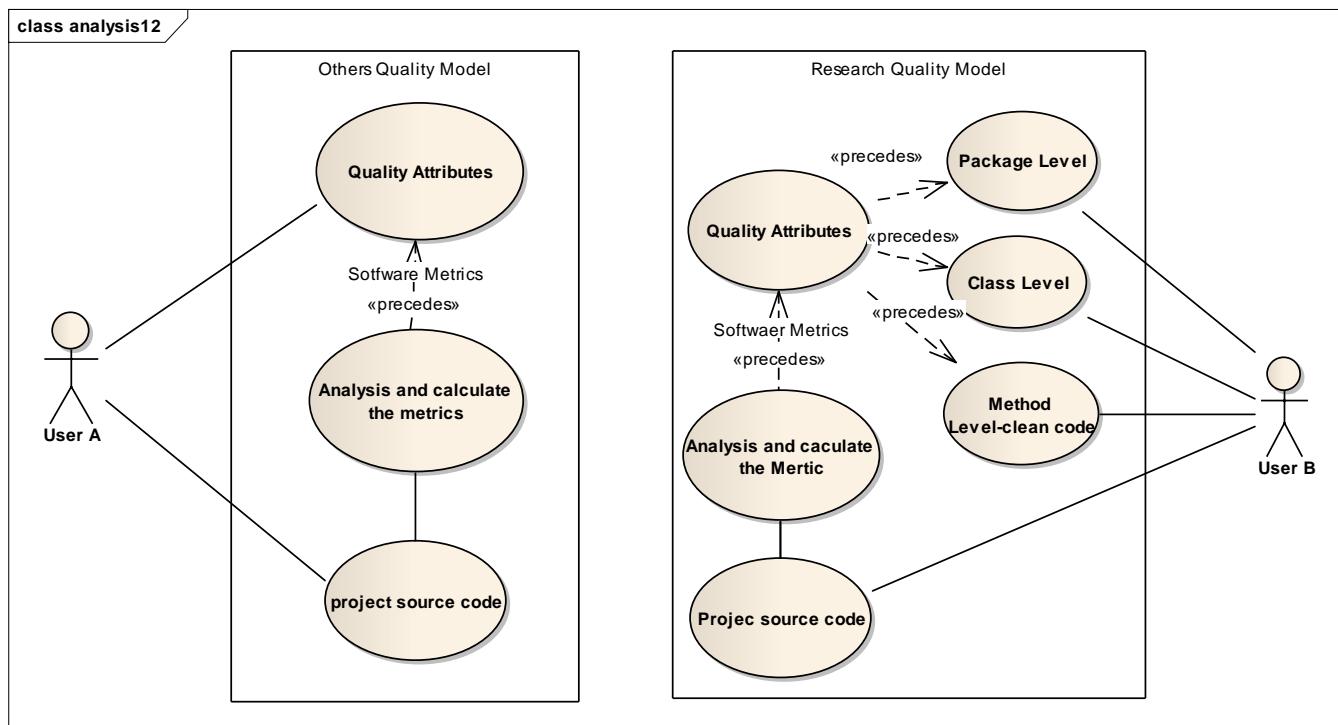


Fig. 1. The proposed quality model with other quality models from the point of view of the user.

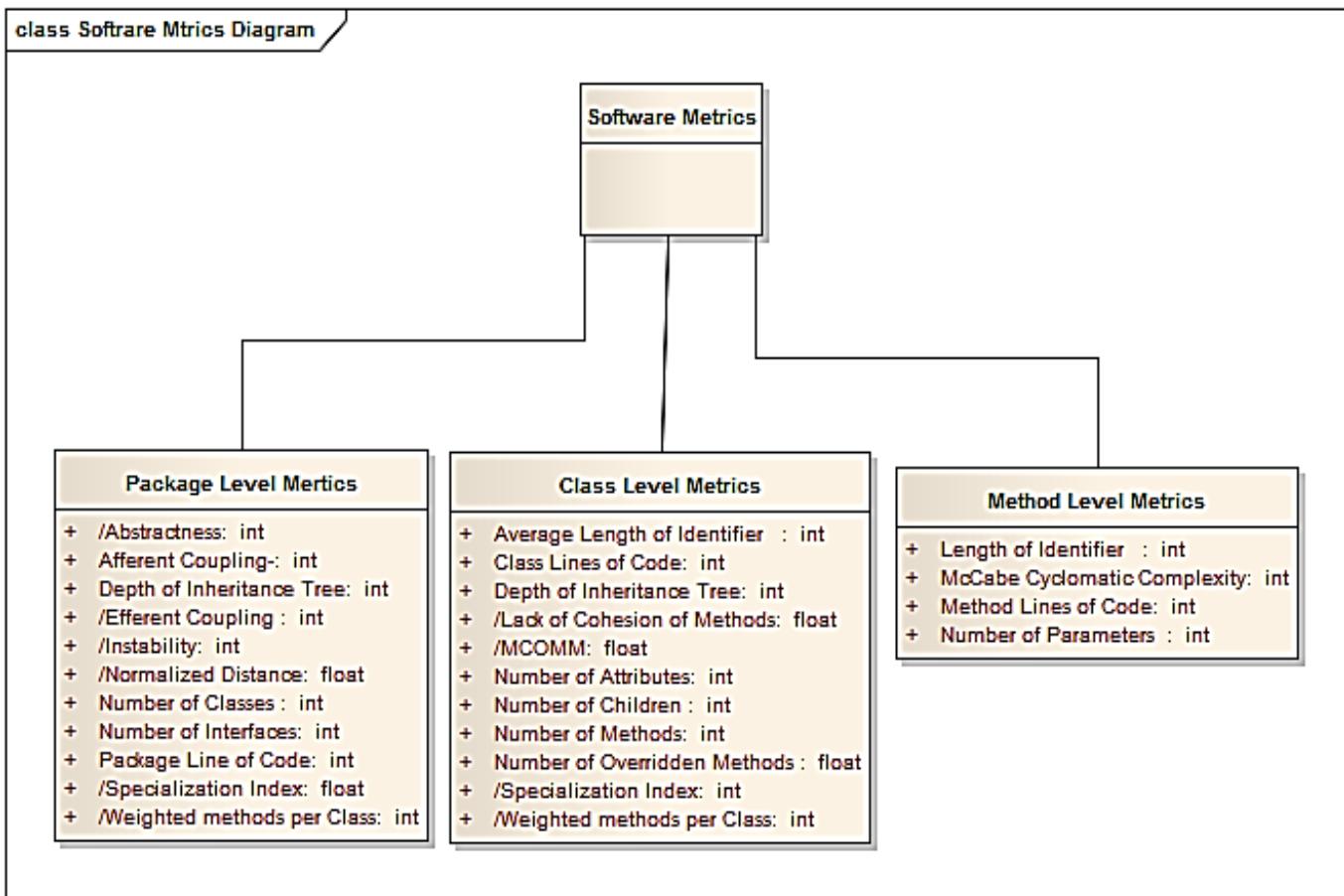


Fig. 2. The class diagram describing the classification of software metrics.

This model focuses on showing the complexity of the source code at the level of the software function within the class because the previous quality models focused on the software as a whole and did not identify points that are likely to be more informative; instead of that, these provided just general indicators. Analysis of the complexity at the level of the source code (low-level components in the software package) is very close to the software developer team. Therefore, the proposed quality model has more effective and practical results to help software developer team during decision-making and improvement of the software product. One of the most important things that affect the accuracy of the analysis of software product at this level is which complexity metrics with suitable threshold value to give clear and efficient assessments.

A. Proposed Model for Function Level

The proposed quality attribute at the function level is the clean code attribute, and the value of this attribute is the binary type (the function either has the character of the clean code or not). This value is determined by the function complexity metrics by linking it to the appropriate threshold value to ensure that the source code of the function has low-complexity, readable and easy to understand. This makes software engineering more efficient in inspection and maintenance of the function source code. As shown in Table I, the complexity of the source code is within the limits of the clean code at the function level.

TABLE I. THRESHOLD LIMITS AND RECOMMENDATIONS AT THE FUNCTION LEVEL

Metric	Threshold limit	Recommendations over the Threshold limit	Recommendations under Threshold limit
Method Lines of Code (mLOC)	Less than 50	Segment the function into new functions by re-structuring the function.	No Recommendations
McCabe's cyclomatic complexity (CC)	Threshold limit in the Table II	Code Reconstruct and optimize the function to reduce complexity.	Recommendations as in the Table II
Number of parameters	Less than or equal to 3	Check the function and make sure that the function performs one task	No Recommendations
Average length of identifier	Greater than 10	Rename identifiers in order to be more clearly.	No Recommendations

According to [22], the threshold limit of McCabe Cyclomatic complexity should be classified into the categories as in Table II which shows the appropriate threshold limit of Cyclomatic complexity.

TABLE II. RECOMMENDATIONS ON THE MEASURE OF MCCABE'S CYCLOMATIC COMPLEXITY (CC) AND ITS RELATION TO RISK [24]

Cyclomatic complexity value	Functions Types	Possible Risk
1 to 4	Simple function	Low
5 to 10	The function is well structured and stable	Low
11 to 20	A complex function	Medium
21 to 50	A complex and troubling function	High
More than 50	A function that generates errors and is Extremely annoying and unstable	Very high

TABLE III. THRESHOLD LIMITS AND RECOMMENDATIONS ON THE LEVEL OF CLASS

Metric	Threshold limit	Recommendations above the Threshold limit	Recommendations under Threshold limit
Class Lines of Code (cLOC)	Greater than 500	Class segmentation to more than one Class	No Recommendations
Average number of McCabe's cyclomatic complexity (CC)	Greater than 10	The class is complex	A well-structured Class
Number of Methods (NM).	Greater than 20	Functional examination of the class	No Recommendations
Number of direct Children (NOC)	Greater than 6	High reusable thus requires examination of class carefully because it depends upon a large number of Classes	Indicating no reuse in the class
Number of Methods overridden (NMO).	Greater than 3	The class is complex and difficult to understand	No Recommendations
Weighted Methods per Class(WMC)	Greater than 15	The class is complex	A well-structured Class
Depth of Inheritance Tree (DIT)	Greater than 5	The complexity of the class as a whole is increasing and there is difficulty in calculating the behavior of the class	No Recommendations
Average length of IDs	Less than 8	IDs are not clear enough	No Recommendations
Lack of Cohesion in Object Methods(LCOM)	Greater than 0.6	Class achieves different purposes and should be divided into several sub-classes	No Recommendations

TABLE IV. THRESHOLD LIMITS AND RECOMMENDATIONS ON THE LEVEL OF PACKAGE

Metric	Threshold limit	Recommendations above the Threshold limit	Recommendations under Threshold limit
Average Weighted Methods per Class(aWMC)	Greater than 3.	The package is complex	A well-structured package
Average Number of Methods overridden (aNMO).	Greater than 15.	The classes are complex and difficult to understand	No Recommendations
Abstractness – RMA	Greater than (0.5)	Abstract package	Cohesive package
Normalized Distance from Main Sequence- Dn	Greater than (0.5)	The package is unstable	The package is stable
Instability-RMI	Greater than (0.5)	The package must be inspected and re-structured	The package is well-designed

The value of the clean code attribute will be evaluated as a clean code if the threshold is exceeded; otherwise, it will be evaluated as a non-clean code.

B. Proposed Model for Class Level

The proposed quality model at this level generates recommendations on measures that affect the complexity of the product depending on the threshold limit as in Table III.

C. Proposed Model of Package Level

The proposed quality model at this level generates recommendations on measures that affect the complexity of the product depending on the threshold limit as in Table IV.

D. Modified Bansya model of Package Level

At the level of the package, the proposed model also suggests using the attributes of Bansya quality model. These attributes can be used for the purpose of making comparisons between software packages to know the best in terms of the closest to the requirements of the user, and this helps the software engineer to choose the best packages during reuse or use of commercial packages. The quality model of the object-oriented design [1] consists of four levels and three relationships among them as in Fig. 3. First Level (L_1) represents quality attributes and it should be wide enough to cover all aspects of design quality.

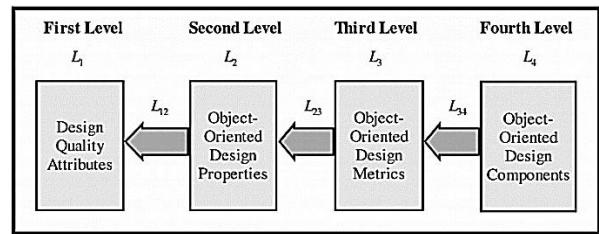


Fig. 3. A Quality model of the object oriented [1].

These qualities are targeted at design quality. A definition of quality characteristics are well explained in Table V.

TABLE V. DEFINITION OF QUALITY ATTRIBUTES OF BANSIYA [5]

Quality Attributes	Definition
Reusability	Reflects the existence of characteristics of object-oriented design that allow the design to be reused to a new problem without much effort.
Flexibility	Characteristics that allow inclusion of changes in design, or adaptive design ability to provide functional-related capabilities.
Understandability	Design characteristics that make the design easy to learn and understand. This relates to the complexity of the design structure directly.
Functionality	The responsibilities assigned to the design of the classes, which are provided through the public interfaces.
Extendibility	Refers to the presence and use of features in the current design that allows integration with the new requirements.
Effectiveness	Refers to the design ability to achieve desired functionality and behavior using object-oriented design concepts and techniques.

TABLE VI. QUALITY CHARACTERISTICS OF BANSIYA AND ITS MEASUREMENTS BY OBJECT-ORIENTED DESIGN METRICS

Design characteristics	Definition	Measured with object-oriented design metrics [10]
Design size	Measures the number of classes used in the design	<i>NumberofClasses</i>
Hierarchies	Hierarchy is used to represent different concepts of generalization and specialization in design. It is calculated from the number of inherited classes that have children in design.	<i>DepthofInheritanceTree</i>
Abstraction	A measure of the aspects of generalization and specialization in design. Items in design that have one or more offspring carrying the property of abstraction	<i>Abstractness</i>
Encapsulation	It is defined as combining data and behavior within a single structure.	No equivalent in object-oriented design measures [20]. This value has been counted as one.
Coupling	Specifies the interconnection of an object with other objects in the design. It is a measure of the number of objects to be accessed by a particular object until it works correctly.	<i>Instability</i>
Cohesion	Evaluates the relationship between functions and variables in the class. Strong overlap in function parameters, variable types is a sign of strong cohesion.	(1)
Composition	Measures relationships: (part-of), (has), (consist-of-part), and (part - whole) relationships, which are aggregation relationships in object-oriented design.	<i>NumberofAttributes</i>
Inheritance	A measure of the "is-a" relationship between the classes. This relationship is related to the level of nesting of the hierarchical structure.	(2)
Polymorphism	The ability to replace objects that have similar interfaces with each other at runtime. It is a measure of the services that are dynamically determined at runtime in an object.	<i>NumberofOverriddenMethods</i>
Messaging	Calculates the number of public functions that provide services for other classes. This is a measure of the services provided by the class.	<i>NumberofMethods</i>
Complexity	A measure of the degree of difficulty in understanding and absorbing internal and external structural classes and their relationships.	<i>WeightedMethodsperClass</i>

The third level (L3) represents design characteristics. Design characteristics as explained in Table VI can be measured by standard object-oriented design metrics instead of using non-standard metrics that suggested by the original model. It expresses the degree of design compatibility to suit the specific properties in L2. L3 is an additional level compared to the Dormey's Quality Model [4].

The tangible elements of L4 are converted into digital form and this made the proposed model gain greater objectivity [10]. As it can be seen, the design of software components written using the object-oriented programming shown in Fig. 3 represents the fourth level of the model (L4). These components are mostly determined through the programming language (for example, functions, objects, and classes). L4 delivers sources that are source code (functions, classes, packages, etc.) that will be measured at the top of level (L3). The next step consists of setting design metrics for design properties (L23). This model selects only one metric for each of the design characteristics. The L2 and L3 relationship was direct and traceable, as shown in the third column of Table VI since the object-oriented design metrics are used to evaluate design characteristics. L12 step probably is the most important step; this step is blending design characteristics with quality attributes. The model uses weights for the design characteristics for obtaining quality attributes value as shown in equations (3), (4), (5), (6), (7), and (8) [1, 10].

The weights can be either positive or negative. The algebraic sign indicates that a particular design characteristic has a positive or negative effect respectively on the quality Attributes [1, 10]. For example, Reusability is positively affected by the Design size (the greater the number of classes, the greater the possibility of reuse). All weights of design characteristics have a specific range between [-1 and +1], so that all quality attributes are in the same range. Positive effects have the values of initial weights (+1) or (+0.5). The negative effects were selected (-1) or (-0.5), and then this value was changed relatively to bring the total results of weights (± 1).

For a better illustration of the relationship between quality characteristics and design characteristics, Table VII explains these relationships. The symbols in the table VII indicate the type of relationship, where the symbol \blacktriangle indicates a positive correlation between design characteristics and quality attributes, i.e. the better the design size value, the better the reuse. However, the symbol \blacktriangledown indicates a negative relationship, i.e. high coupling value reduces flexibility. The original model was based on the choice of one of the software package as a basis for the process of normalizing the results of quality attributes to display the last results. This makes the results of this model unstable because the results will be changed whenever the basis of comparison is changed. Therefore, the researcher believes that relying on the limits of the threshold in the normalization of quality attributes will increase the stability of the results of the proposed model and this leads to increase the reliability of the proposed quality model.

$$\frac{1}{\text{Number of OverriddenMethods}} \quad (1)$$

$$1 - \frac{\text{Number of OverriddenMethods}}{\text{Number of Methods}} \quad (2)$$

$$\text{Reusability} = -0.25 * \text{Coupling} + 0.25 * \text{Cohesion} + 0.5 * \text{Messaging} + 0.5 * \text{DesignSize} \quad (3)$$

$$\text{Flexibility} = 0.25 * \text{Encapsulation} - 0.25 * \text{Coupling} + 0.5 * \text{Composition} + 0.5 * \text{Polymorphism} \quad (4)$$

$$\text{Understandability} =$$

$$-0.33 * \text{Abstraction} + 0.33 * \text{Encapsulation} - 0.33 * \text{Coupling} + 0.33 * \text{Cohesion} - 0.33 * \text{Polymorphism} - 0.33 * \text{Complexity} - 0.33 * \text{DesignSize} \quad (5)$$

$$\text{Functionality} =$$

$$0.12 * \text{Cohesion} + 0.22 * \text{Polymorphism} + 0.22 * \text{Messaging} + 0.22 * \text{DesignSize} + 0.22 * \text{Hierarchies} \quad (6)$$

TABLE VII. THE RELATIONSHIP BETWEEN QUALITY ATTRIBUTES AND CHARACTERISTICS OF THE DESIGN [1]

Design characteristics	Quality attributes					
	Reusability	Flexibility	Understandability	Functionality	Extendibility	Effectiveness
Design size	\blacktriangle		\blacktriangledown	\blacktriangle		
Hierarchies		\blacktriangle		\blacktriangle		
Abstraction			\blacktriangledown		\blacktriangle	\blacktriangle
Encapsulation		\blacktriangle	\blacktriangle			
Coupling	\blacktriangledown	\blacktriangledown	\blacktriangledown		\blacktriangledown	
Cohesion	\blacktriangle		\blacktriangle	\blacktriangle		
Composition		\blacktriangle				\blacktriangle
Inheritance					\blacktriangle	\blacktriangle
Polymorphism			\blacktriangledown	\blacktriangle	\blacktriangle	\blacktriangle
Messaging	\blacktriangle			\blacktriangle		
Complexity			\blacktriangledown			

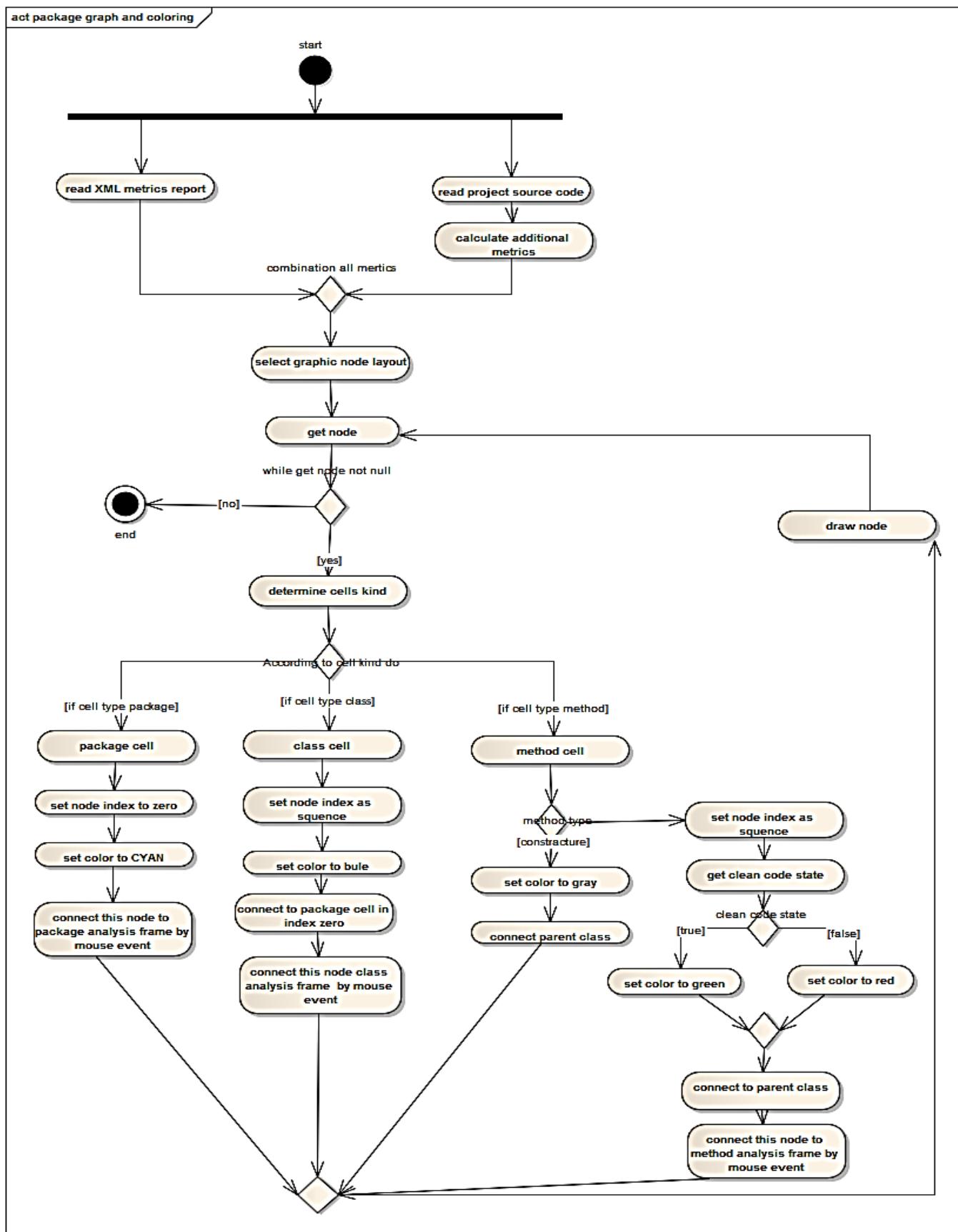


Fig. 4. Proposed graphical analysis algorithm for the package and the coloring.

$$\text{Extendibility} = 0.5 * \text{Abstraction} - 0.5 * \text{Coupling} + 0.5 * \text{Inheritance} + 0.5 * \text{Polymorphism} \quad (7)$$

$$\begin{aligned} \text{Effectiveness} &= 0.2 * \text{Abstraction} + 0.2 * \\ \text{Encapsulation} &+ 0.2 * \text{Composition} + 0.2 * \\ \text{Inheritance} &+ 0.2 * \text{Polymorphism} \end{aligned} \quad (8)$$

E. Visual Representation of the Proposed Quality Model

Graphical representation and color have been used for the proposed software quality model to illustrate the software parts as one of the quality models requirements. The graphical representation, in general, is more intuitive and efficient in understanding than just a table of numbers and values. Therefore, that makes software engineers create a complete and rapid view for software under development. Different colors can also be used to illustrate the complexity of each part of the software under development, in which the algorithm for the analysis and colorization of the parts of the software under development is illustrated in Fig. 4 by using the UML activities diagram, which is designed in line with the Java presser package. The algorithm description of the proposed model is as following steps:

Step 1: Read the source code of the project (Java files) to be analyzed.

Step 2: Read the software metrics from the xml file for the project to be analyzed.

Step 3: Calculate the additional metrics from the source code of the project to be analyzed in the first step.

Step 4: Integrate all software metrics together from Step 3 & 2.

Step 5: Determine the package components graphically: (radial tree, tree, compact tree, fast organic).

Step 6: For each node in the drawing, apply Step 7 and 8.

Step 7: Determine the characteristics of each node in the graphical analysis by type

1) If the node type is a package, give the node following characteristics:

- It is defined as the root of the tree in the graphic.
- The cyan color (CYAN) is given.
- Binds with the display interface of packet analysis.

2) If the node type is a class, give the node following characteristics:

- An address shall be given according to its sequence in the graphic.
- The blue color (BLUE) is given.
- Binds to the display interface of the class analysis.

Step 8: If the node type is a function, do the following:

1) If the function type is Constructor(special function for building the object), give the node following characteristics:

- An address shall be given according to its sequence in the graphic.

- The gray color (GRAY) is given.

2) If the function is a regular function, give the node following characteristics:

- An address shall be given according to its sequence in the graphic.
- A green color (GREEN) is given if the node has a clean code attribute and a red color (RED) is given if not.
- Binds with display interface function analysis

Step 9: Show the graphic.

IV. EXPERIMENT WITH RESULT AND DISCUSSION

The proposed software namely Multi Levels Quality Analysis Tool (*MLQA*) is developed to analyze and evaluate the engineered software. The tool was programmatically based on Metrics 1.3.6 (Eclipse Metrics plugin-Provide metrics calculation and dependency analyzer plug in for the Eclipse platform) as well as a *Java doc parser* to calculate the non-supported metrics in Metrics 1.3.6. *Java doc parser* was also needed to extract code information. In addition, for graphical representation, the *MLQA* used package *Jgraph 5.13*. Moreover, *XML Doc Parser* to read XML reports from Metrics 1.3.6 was built as part of *MLQA* tool.

To test the proposed algorithm, three Java programs have been tested in this experiment to demonstrate the proposed tool results and their practicality in software analysis, these programs are:

1) Patience game: This game is used for the purpose of learning how to deal with arrays and the structure of data in Java language. The source code of the game was downloaded from the website (<http://www.neiljohan.com/java>).

2) Syntaxchecker game: This game is used to learn how to handle matrices and data structures in Java. The source code of the game was downloaded from the website (<http://www.neiljohan.com/java>).

3) Payroll System (payRoll): An accounting system that deals with databases and reports in Java. The source code (payRoll) was downloaded from (<http://www.projectsparadise.com>).

To illustrate the test of the three projects, the results of the project (Patience Game) were explained at all levels of analysis. The remaining projects will be later compared with the first project results. The Patience Game consists of seven classes including two interfaces (interface). As illustrated in Fig. 5, graphical analysis of the source code was executed using the tool (*MLQA*) and the style used to display the package components is the radial tree.

The color of each node in Fig. 5 has its different meaning as explained in section 3.5. The results of the Fig. 5 are explained in Table VIII.

TABLE VIII. RESULTS OF THE GRAPHICAL ANALYSIS OF THE PATIENCE GAME PACKAGE

Class Name	Number of Total Functions	Number of clean code functions	Number of unclean code functions
CardTest	2	1	1
PlayCardImp	7	7	0
Patience	5	2	3
NeilClass	4	0	4
PackImp	10	9	1

Table VIII shows that the **PlayCardImp** functions all have the clean code attribute. This is a good indicator of the design quality of this class. The **PackImp** class had one function, which does not have the clean code attribute out of 10 functions, the researcher believes that this is a good indicator. The **CardTest** class has two functions, one with a clean code attribute and the other without it. While the **Patience** class has five functions, three of them do not have the clean code attribute, and some complexity can be found in this class. All **NeilClass** functions do not have clean code, this is an indicator of poor design quality, and this is the most complex class in this package, so it should be further emphasized during inspection and maintenance.

The proposed algorithm has been designed as GUI to give details for the user, these details view of nodes components with a separate interface for each node, which gives the software engineer the ability to view software complexity metrics and recommendations with source code. The results can be presented according to the three levels of the proposed quality model, in which for each one of these 3-level, there is a separated GUI. First level, Package-level analysis results as depicted in Fig. 6 shows package details in terms of complexity metrics and threshold limits as well as the quality attributes of update Bansiya quality model for the highest level of abstraction.

Package-level results can be used to compare or track package development during software development. It is observed in Analysis and Recommendations for this package section in Fig. 6, that the Patience Game package has good properties in terms of the NORM metric, which is affecting the readability and comprehension, while the package is complex according to WMC metric because this metric directly affects on the complexity of the package in general. Moreover, the Abstractness (RMA) and Normalized Distance from Main Sequence (D_n) metrics affect on the quality of the package design in terms of incoming and outgoing pairing, and these measures were within the threshold. However, the Instability (RMI) metric was far from the threshold, so this package is unstable.

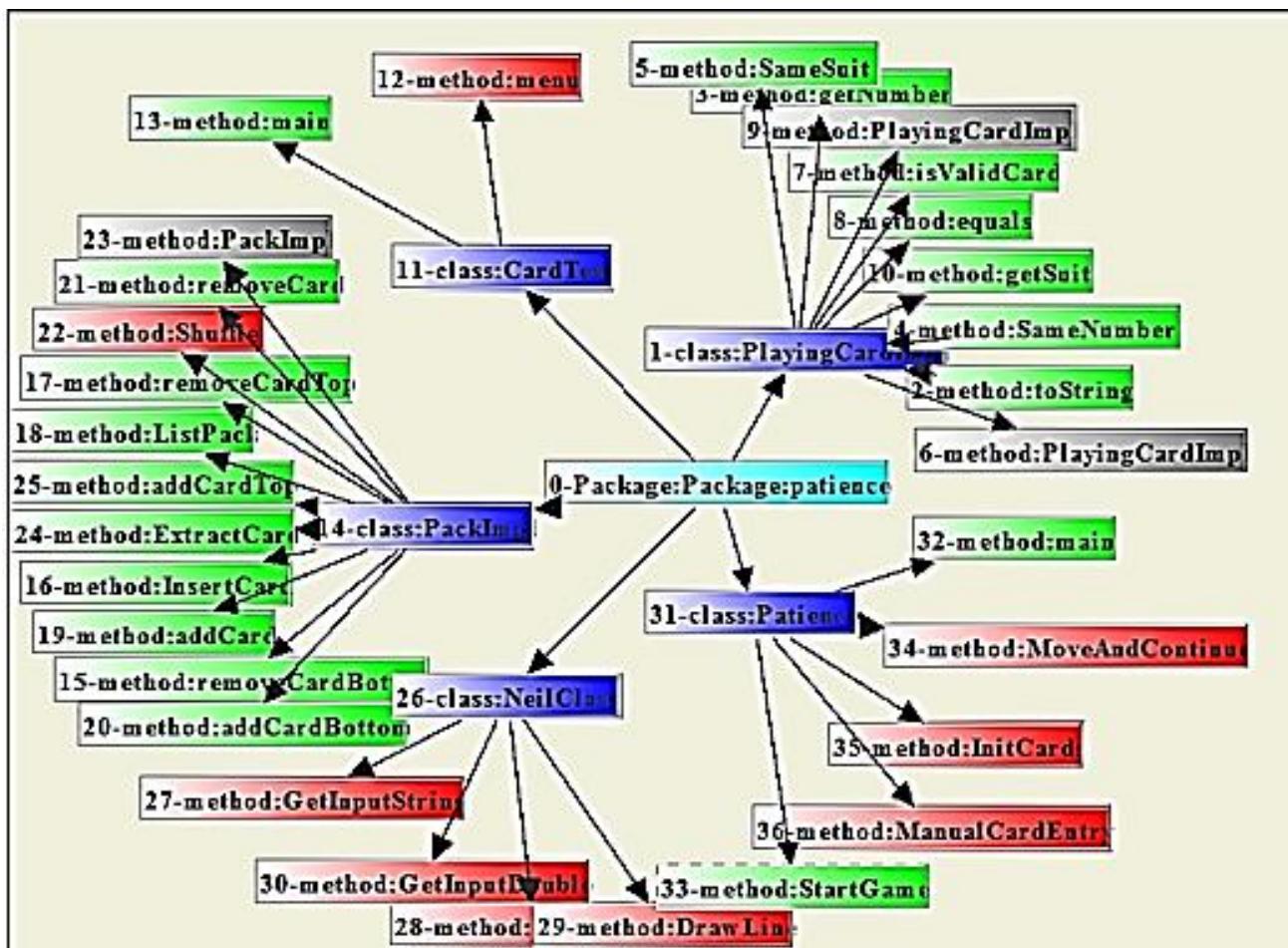


Fig. 5. Shows the graphical analysis of the components of the Patience Game.

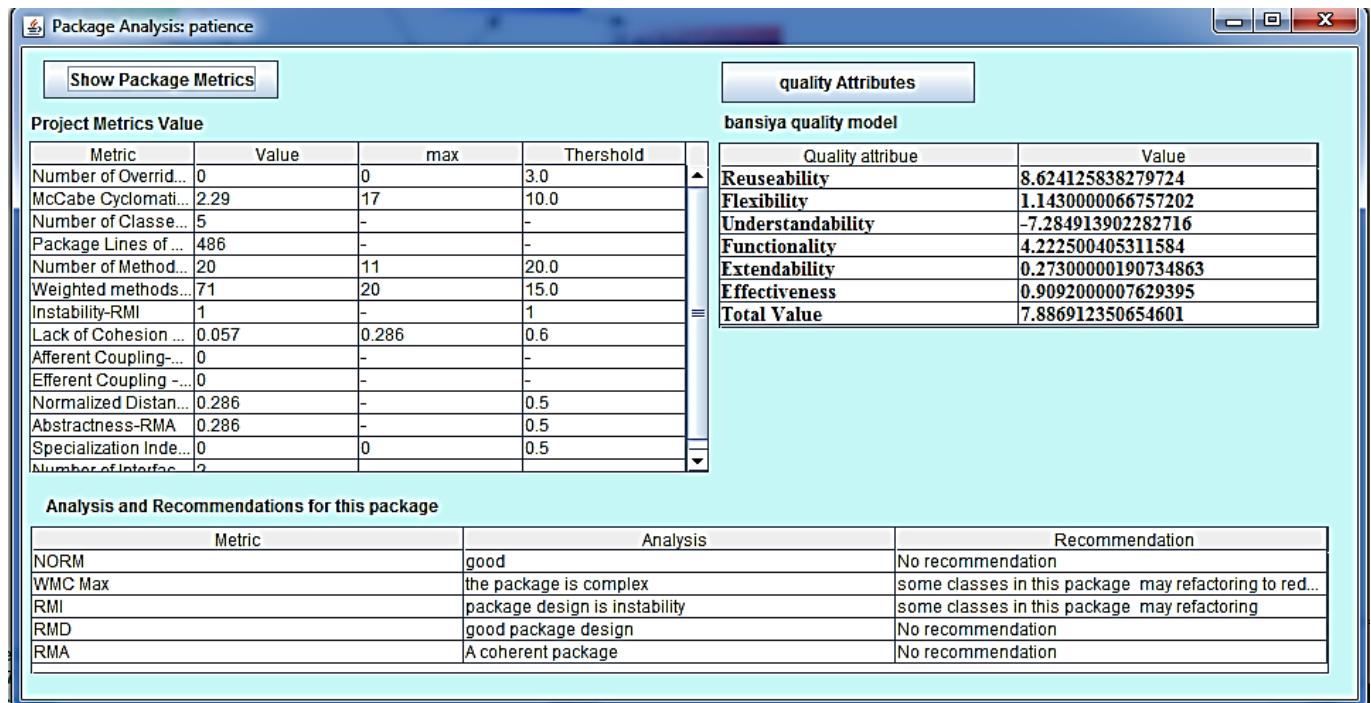


Fig. 6. Results of the package level analysis.

Secondly, Class-level analysis results consisting of the values of the complexity metrics of the class with the recommendations depend on the threshold for these metrics.

To illustrate the ability of the tool at this level, **PlayCardImpl** class has been chosen to explain the results of class analysis support by MLQA tool as shown in Fig. 7.

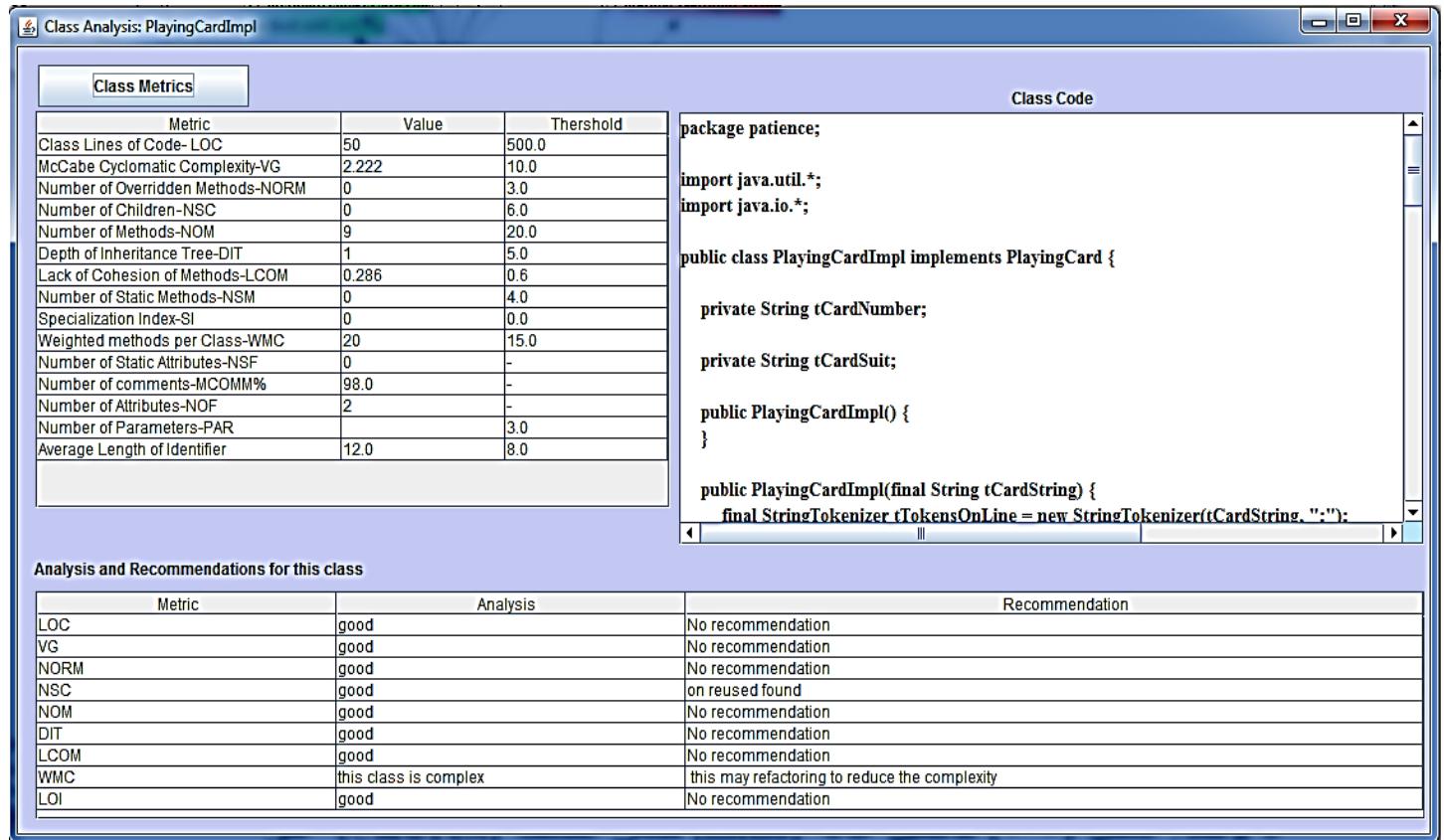


Fig. 7. Analysis results of the AnalysisCardImp class.

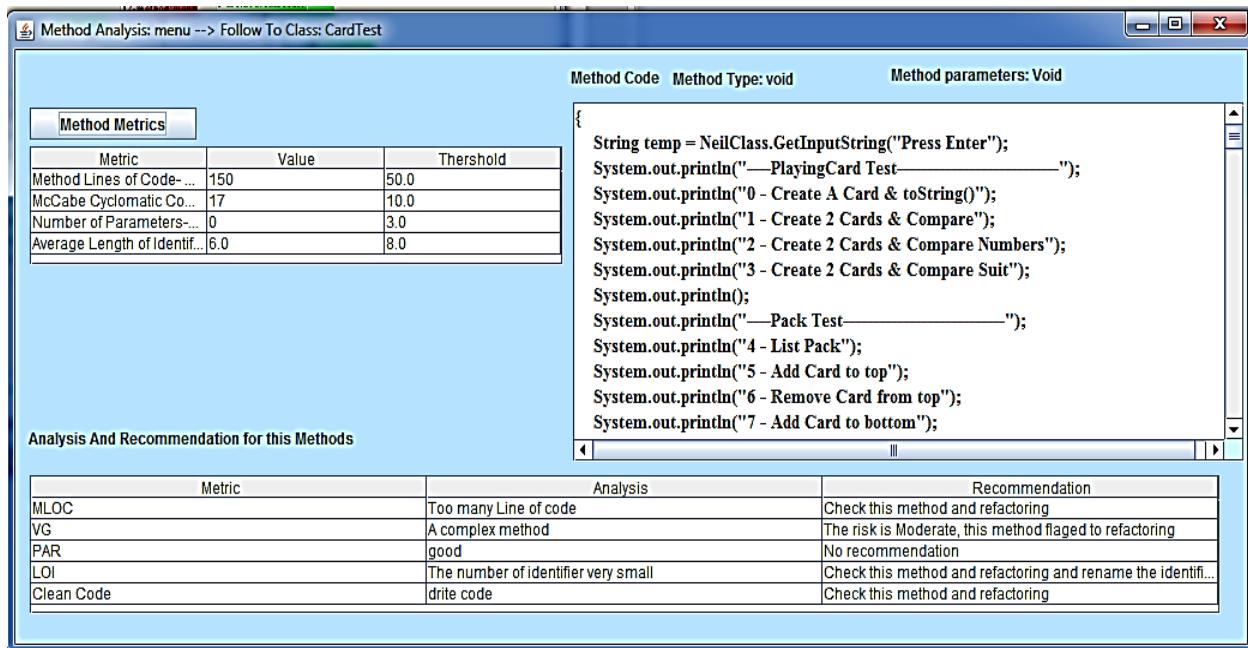


Fig. 8. Shows the internal details of the function (menu) that does not have the clean code attribute.

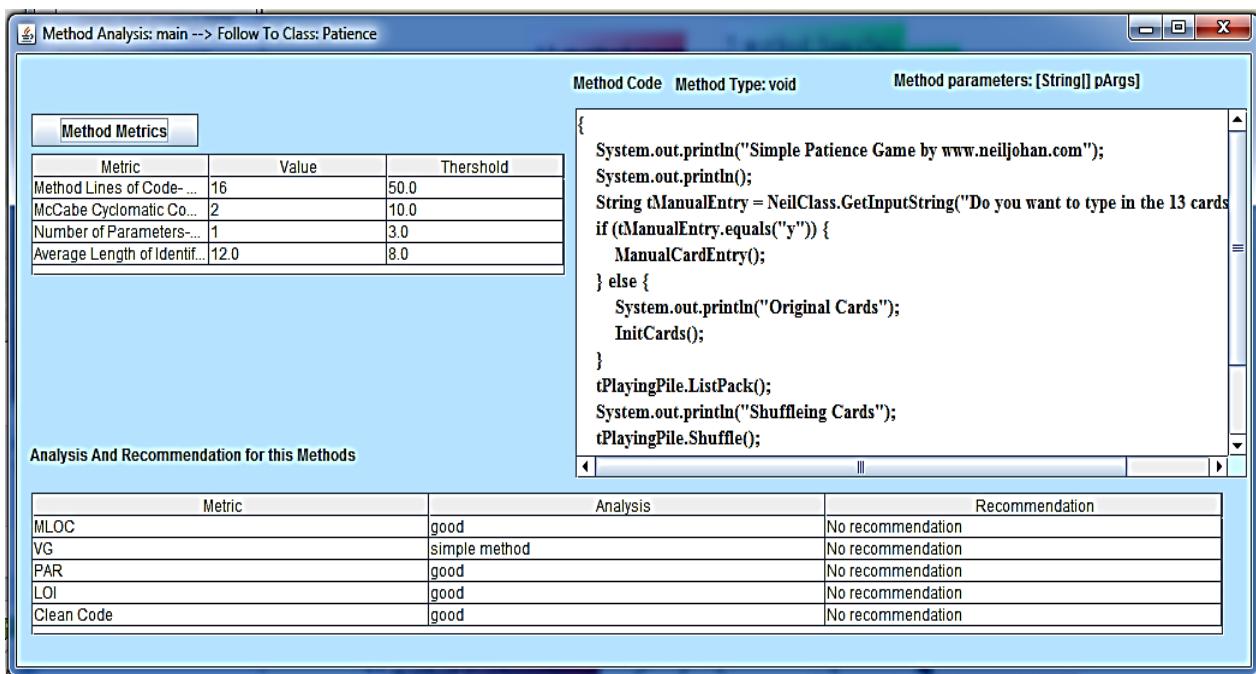


Fig. 9. Internal details of the function (main) that have the clean code attribute.

Here the point of view of the researchers observed that **PlayCardImpl** class has a good quality design in terms of Class Lines of Code (cLOC), the Average number of McCabe's Cyclomatic complexity (CC), the Number of Methods (NM), the Number of direct Children (NOC), Depth of Inheritance Tree (DIT), the Average length of IDs, Number of Methods overridden (NMO) and Lack of Cohesion in Object Methods (LCOM) because they were within the limits of threshold. However, this class is considered complex because the Weighted Methods per Class (WMC) metric has moved away from the appropriate threshold, exceeded by (20)

while the appropriate threshold limit is (15). Moreover, the results of any class in the package can be displayed in this way.

Thirdly, Function-level analysis, in order to clarify this level, the two functions in two different classifications were selected, as illustrated in Fig. 8 function details (**menu**) within the class (**CardTest**), which classified as it does not have the clean code attribute.

The researcher noted that the menu function was classified unclean code because the software complexity measures are

beyond the scope of the clean code attribute. The researcher believes that this function should be restructured or it will lead to difficulties in the inspection, maintenance and reuse process in class. While the function (**main**) in Patience class described in Fig. 9 was classified as it has the clean code attribute because the software complexity measures are within the scope of the clean code attribute, which is a good indicator of the design of this

The researcher noted that the menu function was classified unclean code because the software complexity measures are beyond the scope of the clean code attribute. The researcher believes that this function should be restructured or it will lead to difficulties in the inspection, maintenance and reuse process in class. While the function (**main**) in Patience class described in Fig. 9 was classified as it has the clean code attribute because the software complexity measures are within the scope of the clean code attribute, which is a good indicator of the design of this function. Moreover, the same method of displaying results at the function level can be used for all functions within the package to be analyzed. In order to illustrate the quality attributes of the update Bansiya quality model at the package level, the researcher chose three software (patience game, syntaxchecker game, payRoll accounting system), which were analyzed using the MLQAtool, and the results were shown in Table IX.

TABLE IX. COMPARISON OF SOFTWARE (PATIENCE, SYNTAXCHECKER, PAYROLL), USING THE QUALITY ATTRIBUTES OF THE UPDATE BANSIYA QUALITY MODEL

Software Name	Total value	Effectiveness	Extendability	Functionality	Understandability	Flexibility	Reusability
Patience	3.944905671	0.680666667	0.368333325	0.92044707	0.379666092	0.571666673	1.024125844
syntaxChecker	5.158801762	1.386395192	0.799321353	0.91110431	-0.898028028	2.359166563	0.600842372
PayRoll	26.56566665	7.519047606	0.797619015	2.070666655	-2.254999934	17.83333331	0.599999994

The researchers observed from the Table IX that an increase in the Effectiveness attribute and the Flexibility attribute strongly conflict with Understandability attribute, while with Reusability, functionality and Extendability attributes, the effect has been reduced, although the values are also relatively large. However, the differentiation between the values of these attributes is due to the non-functional requirements of the customer, so that the software engineer can make appropriate decisions in the light of these values. These features were illustrated graphically in Fig. 10 to give a more comprehensive picture of the relationship among these attribute.

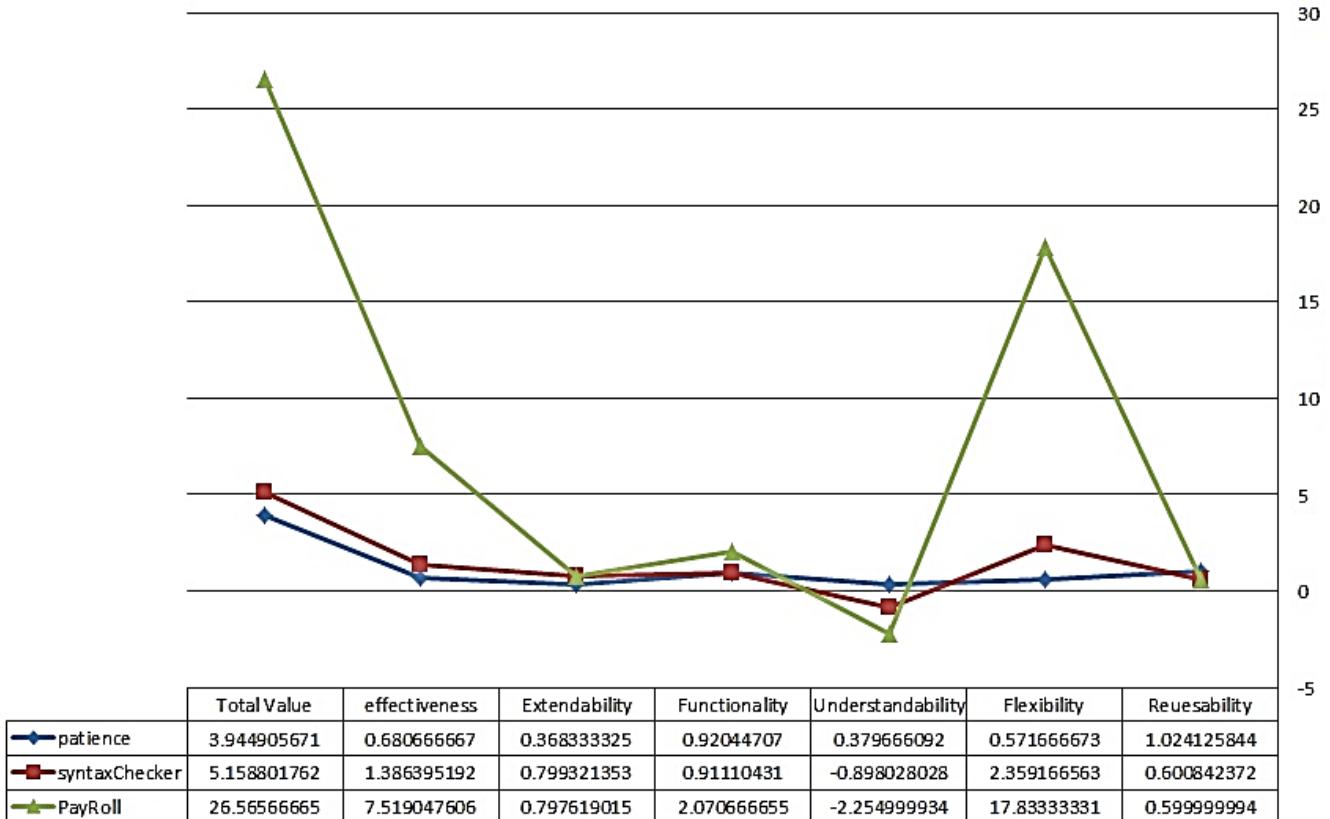


Fig. 10. Chart for quality attributes in the three tested software projects.

V. CONCLUSION

To sum up this paper, the current software quality models suffer from some ambiguities during the analysis of the quality of the software as there is difficulty for the software engineer during apply quality model because it requires direct intervention for all operations manually and exhaustively, which may affect on the process of software development. Therefore, quality model and support tools are developed in this paper to be used by software engineers to control software product written in Java programming language named Multi Levels Quality Analysis Tool (*MLQA*) as evaluation algorithm for software quality model comprising software product, software metrics, and clean code programming has been proposed based on three levels of abstraction as *package, class, and function*, as it is proved that three levels give more accurate results and recommendations from analysis results of code rather than from one level of abstraction, because software measures that are appropriate for function-level analysis may not be suitable for the class or package level. Moreover, the proposed *MLQA* has been boosted with a graphical analysis of color discrimination to give a quick look to the software engineer about the complexity of the software, as well as to give more ability to update the source code of the software package with viewing the metrics values and recommendations in the single environment regarding the software source code during development. Besides that, it has been concluded that IDs metric should be included in the source code complexity metrics due to its great effect for the clean code, in which IDs is more important than the measure of comments ratio. Proposed *MLQA* is based on *Bansiya* model, which is succeeded in finding a mathematical relationship between quality attributes and software metrics in terms of design characteristics, however, it is weak in the normalization process in packages compare, therefore, the modification of the proposed work is to change normalization by using threshold values instead of using one of the packages as the base of normalization process so as to make this model more effective to track the development of software from one version to another. The aforementioned results have come out from the experiment conducted on three Java codes, which have been tested in this experiment to demonstrate the validity of the proposed tool results and their practicality in software analysis. For the future work, there is a possibility of applying artificial intelligence techniques in the field of building tools of computer aided software engineering in the field of quality models, especially the use of fuzzy logic in the field of selecting the appropriate threshold for the software metrics. Moreover, the possibility of incorporating the idea of adding a quality model as part of a development environment of programming languages will increase their adoption. Finally, extend the scope of the proposed quality tool to include more languages that support object oriented programming such as C++, C #.

REFERENCE

- [1] Bansiya, Jagdish., "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 28, NO. 1. pp. 4-17, 2002.
- [2] Mrinal, R. Singh. And Mittal , Arpita, "Survey on Impact of Software Metrics on Software Quality", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 3, No. 1 .pp. 137-141,2012.
- [3] Lee, Alice T., " Software Analysis Handbook: Software Complexity Analysis and Software Reliability Estimation and Prediction ", Information Systems Journal, NASA Center for AeroSpace Information, Available from ston.jsc.nasa.gov,1994.
- [4] Dromey, R. Geoff., "SOFTWARE PRODUCT QUALITY: Theory, Model, and Practice". Software Quality Institute ,Griffith University, AUSTRALIA, 1998.
- [5] L. Smith, et al., "Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems, Management Information Systems," Department of the Air Force-Software Technology Support Center,(February), 2003.
- [6] R. Martin, "OO design quality metrics," An analysis of dependencies, vol. 12, pp. 151-170, 1994.
- [7] Honglei, Tu., Wei, Sun., Yanan, Zhang., " The Research on Software Metrics and Software Complexity Metrics ", IEEE journals, International Forum on Computer Science-Technology and Applications, ISBN: 978076953930 Vol: 1 Pages: 131-136 , 2009.
- [8] Martin, Robert C., " Clean Code - A Handbook of Agile Software Craftsmanship ", Published by Pearson Education, Inc, ISBN 0-13-235088-2, 2008.
- [9] Martin, Robert C., " The clean coder : a code of conduct for professional programmers ", Pearson Education, Inc, ISBN-13: 978-0-13-708107-3, ISBN-10: 0-13-708107-3, 2011.
- [10] Jetter, Andreas., "Assessing Software Quality Attributes with Source Code Metrics", Diploma Thesis, Department of Informatics, University of Zurich, 2006.
- [11] Panovski , Gregor, " Product Software Quality ", MASTER'S THESIS, Department of Mathematics and Computing Science, TECHNISCHE UNIVERSITEIT EINDHOVEN, 2008.
- [12] Borchert, Thomas., " Code Profiling: Static Code Analysis ", MASTER'S THESIS, Department of Computer Science, Karlstad University, Sweden, 2008.
- [13] Bhatti, Hassan R., "Automatic Measurement of Source Code Complexity ", MASTER'S THESIS, Department of Computer Science, Electrical and Space Engineering, Lulea University of Technology,2010.
- [14] Widheden, Göran. , " Software Complexity: Measures and Measuring for Dependable Computer Systems ", MASTER'S THESIS, Department of Computer Science and Engineering, University of Gothenburg, Goteborg, Sweden,2010.
- [15] Chidamber, S. R., & Kemerer, C. F., " A metrics suite for object oriented design", IEEE Transactions on software engineering, 20(6), 476-493,1994.
- [16] Chandra, E., Linda, P. Edith," Class Break Point Determination Using CK Metrics Thresholds ", Global Journal of Computer Science and Technology, Vol.10 ,Issue.14, Pp:73-77,2010.
- [17] Jamwal, Ranbirshwar S.,Jamwal, Deepshikha. & Padha, Devanand., " Comparative Analysis of Different Software Quality Models", Proceedings of the 3rd National Conference; INDIACom,Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi,2009.
- [18] KITCHENHAM, BARBARA., "SOFTWARE QUALITY: THE ELUSIVE TARGET", IEEE SOFTWARE, Vol. 13, No. 1: JANUARY 1996, pp. 12-21,1996.
- [19] Wang, Su-Hua., Samadhiya, Durgesh. & Chen, Dengjie., "Software Quality: Role and Value of Quality Models", International Journal of Advancements in Computing Technology Volume 3, Number 6, , pp. 65-74,2011.
- [20] Jamwal. Dr. Deepshikha., " Analysis of Software Quality Models for Organizations", International Journal of Latest Trends in Computing ,E-ISSN: 2045-5364,Volume 1, Issue 2, pp. 19-23,2010.
- [21] Abdul, Jabbar. P, S.Sarala," Advanced Program Complexity Metrics and Measurement ", International Journal of Computer Applications (0975 – 8887), Val 23, No: 2, Pages: 29-33, 2011.

- [22] Laird, Linda M., "Software measurement and estimation: a practical approach ", Published by John Wiley & Sons and IEEE Computer Society, ISBN 0-471-67622-5,2006.
- [23] Herbold, S. , " Calculation and Optimization of Thresholds for Sets of Software Metrics ", Technical Report, Institute of Computer Science, University of Goettingen, Germany, No. IFI-TB-2010-01, ISSN 1611-104, 2011.
- [24] Sastry, J.S.V.R.S., " MEASURING OBJECT-ORIENTED SYSTEMS BASED ON THE EXPERIMENTAL ANALYSIS OF THE COMPLEXITY METRICS ", International Journal of Engineering Science and Technology (IJEST), ISSN : 0975-5462 ,Vol. 3, No. 5 , Pages: 3726-3731, 2011.
- [25] El-lateef, T. Abd., Yousef, A. H., "Object Oriented Design Metrics Framework based on Code Extraction", Computers and Systems Department, Faculty of Engineering, Ain Shams University,2006.
- [26] Pressman, Roger S.," SOFTWARE ENGINEERING: A PRACTITIONER'S APPROACH", 8th EDITION, McGraw-Hill, ISBN 978-0-07-802212-8,2014.
- [27] S. Misra, et al., "A suite of object oriented cognitive complexity metrics," IEEE Access, vol. 6, pp. 8782-8796, 2018.
- [28] Bruntink, M., van Deursen, A., "Predicting Class Testability using Object-Oriented Metrics", Proceedings of the Fourth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'04). Pages: 136-145, ISBN: 0-7695-2144-4,2004.
- [29] Abran, A., "SOFTWARE METRICS AND SOFTWARE METROLOGY", IEEE Computer Society, Published by John Wiley & Sons, Inc., ISBN: 978-0-470-59720-0, 2010.