International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-6, March 2020

Software Product Quality Management Methodology & the Quantitative Assessment of Analyzability Indicators

Boumedyen Shannaq, Richmond Adebiaye

Abstract*— The research paper developed a new software metric methodology for evaluating the analyzability indicator for software products. The proposed research methodology provided an objective and quantitative assessment in accordance with the requirements, limitations, purpose and specific features of software products. Forty-one (41) java programs were analyzed to extract and evaluate the software metrics described in 'Halstead metrics. The mathematical classification model was developed to replace the expert output in the evaluating process as related to the software metric indicators. The output of the algorithm was applied to identify the metrics with the greatest analyzability influence. The result indicated that 13 measured metrics with 98% of "analyzability" are relevant to seven (7) software code metrics with the remaining six (6) metrics making up only ~ 5% of "analyzability". The analyzed ROC-curves were similarly computed to test the performance of the proposed methodology compared to the expert's metric evaluation. The ROC-curves indicator for the proposed methodology showed resultant scores of ROC = 7.4 as compared to 7.3 from the experts' evaluation. However, both methods were correlated effectively after analytical computations with a resultant performance which showed that the proposed method outperforms the expert's evaluation.

***Keywords*— Software Metrics; Software Quality; Quantitative Assessment; Analyzability**

I. INTRODUCTION

The development of programming has led to the creation of an entire industry for the production of software products. A software product today is an object of industrial production from a community of people that are involved in its creation, as the circulation of such products reaches hundreds of millions of copies [1]. The range of tasks that use software products is expanding. It is also obvious that a favorable situation has been created for the deeper integration of the software product in all spheres of human life [2]. On the way to the rapid development of programming, the transition from unit production to industrial has created a number of unresolved problems, one of which is quality management of the software product. The impossibility of achieving reliable and considerable quality control is due to the fact, that at present there are no formalized, quantitative methods for measuring the current level of quality, or ways to manage the software process. Obviously, this state of concern is undesirable in the industrial production of a product, since it is an additional risk factor that includes failure to meet production deadlines, exceeding the budget, etc [3][4]. In addition, the level of confidence in software products is growing unreasonably quickly, creating huge reliance and trust in areas of human activity associated with high responsibility including human survival [5]. Currently, the most common and used international standard in the field of software product quality is ISO / IEC 9126 [6]. Fig.1 describes the 'Maintainability' metrics with the new software sub-characteristics.

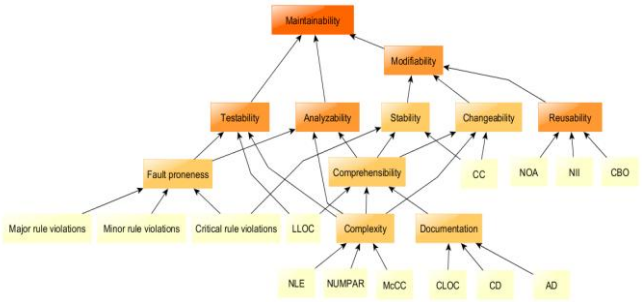


Fig.1 Maintainability metrics [6]

The quality model defined in this standard is taken as the basis for this research. The presence of inconsistency determines the existence of an urgent scientific and technical problem, leading to the urgent need to develop methods for quantitative assessment of software quality. This research aims to develop a methodology for quality indicators preferably - "analyzability" using quantitative assessment.

II. BACKGROUND AND LITERATURE REVIEW

Many large international standardization organizations deal with the problem of evaluating and ensuring the quality of software products, which emphasizes the existence and relevance of the problem. To date, several dozens of international standards have been issued that regulate issues related to the quality of software products [1].

Manuscript received on February 10, 2020.
Revised Manuscript received on February 20, 2020.
Manuscript published on March 30, 2020.
* Correspondence Author
Boumedyen Shannaq*, MIS , University of Buraimi, Al buraimi, 968, Sultanate of Oman, Email: boumedyen@uob.edu.om
Richmond Adebiaye, Dept. of Informatics & Engineering Systems, University of South Carolina Upstate, Spartanburg, USA. Email: radebiay@uscupstate.edu

Currently, the most common and used international standard in the field of software product quality is ISO / IEC 9126 which was reviewed in [6]. The quality model defined in this standard is taken as the basis for this study. Today, [2] the "analyzability" of the program code refers to quality indicators of software quality and is measured using the expert's judgment.

However, such an approach to measuring quality indicators has a number of significant drawbacks: subjectivity of the assessment and conscious subjectivity is possible, i.e. interest in making any decision, the high cost of such an assessment, the absolute repeatability of the results and finally, the inability to conduct an assessment due to the lack of members of the expert council. Obviously, there are needs for new methods for quantitative assessment as the basis of quality indicators for the measurement. [6] In the field of software development, software quality assessment is still in its transitional state and has yet to be fully implemented. At this stage in the development of programming, there is no acceptable description of the quality of software. There is also no common understanding of what quality should consist of or other ways to implement quality measurement. [7][8]

There are many definitions of the concept of quality, which, in essence, boil down to the totality of the technical, technological and operational characteristics of the products or processes, by which they are able to meet and satisfy the requirements of the customer. Currently, the most common and used international standard in the field of software quality assessment is ISO / IEC 9126 which defines quality as "the entire scope of signs and characteristics of a product or service that relates to their ability to satisfy the established or its anticipated needs". In the ISO / IEC 9126 standard, six basic quality characteristics are given, of which, today, only a small part can be measured quantitatively. While the remaining quality indicators relate only by measurement accuracy, to qualitative or categorical-descriptive analysis. Obviously, this state of affairs leads to unreasonable management decisions in the field of quality management made on the basis of informal, intuitive quality assessments, which leads to unjustified risk, failure to meet project deadlines, exceeding budgets and, ultimately, to financial losses [9]. [10] expanded in related research that machine learning algorithms could be used to predict software bugs. [11] reviews many kinds of literature and formulate a description of the software quality. [12] proposed new use of information flow tracing to confuse the control flow of a program, to overthrow hateful code injection when considering the code size and the time. [13] proposed model LCC and LCOM measurements as cohesion metrics. Chidamber and Kemerer coupling metrics were used for assessing coupling. Their experiments found that measuring class cohesion based on totally shared attribute practice and method call is not sufficient. [15] proposed an empirically model for quantifying analyzability using object-oriented software. In their study, they found that the Strategy complexity of software is also an influencing factor of analyzability but with a negative impact. [16] discusses numerous metrics in each of five types of software quality metrics: product quality, in-process quality, testing quality, maintenance quality, and customer satisfaction quality.

III. ALGORITHM FOR ANALYZABILITY

The presentation of the algorithm could be different in terms of verbal description, representation appearance, description in pseudo-code, and etc. Each algorithm, from the point of view of human analysis, has a complexity that can be expressed in time that a specialist needs to spend in order to understand the principles of the algorithm. If any algorithm inherently already has initial complexity, it is obvious that different algorithms will have different initial complexity. It is also likely that the algorithm for adding two numbers will have less initial complexity than the algorithm for solving the traveling salesman problem. However, in the process of implementing the algorithm, some more factors are added to its initial complexity. The algorithm refers:

Let IC be the initial complexity of the algorithm.

Let 'cof' be a coefficient depending on the presentation form an algorithm of a description language.

Let IC` be the complexity of the same algorithm for a specific implementation,

therefore, the Expression will take the form:

$$(cof * IC` / IC) >= 1 \qquad (1)$$

It can be seen from Expression (1) that if the presentation form of the algorithm does not complicate its understanding (cof = 1), and the implementation does not complicate it, then the complexity of a particular implementation matches with the initial complexity of the algorithm. It is necessary to introduce a correction for the complexity of perception of the presentation form itself, i.e. the implementation of an algorithm using different presentation forms (for example, in different programming languages) will have different complexity, *cof1 x IC`*, *cof2 x IC`* etc. The same description of the algorithm using the same presentation form can also have different complexity, i.e. time spent by a specialist on his understanding will be different *cof1 x IC1`*, *cof2 x IC2`*. This difference in the complexity of specific implementation depends on the quantitative indicators of the representation on which the algorithm is described. For example, for imperative programming languages, this is the complexity of the control flow, data flow, the size of the program and any of its elements. In fact, if there exists a certain coefficient *Acof* capable of considering these measurable characteristics of various implementations of the algorithm, then Expression (1) must be rewritten as follows

$$(Acof * cof * C` / C) >= 1 \qquad (2)$$

Using Expression (2), we can give a definition of analyzability - Analyzability is a quantity *Acof* that distinguishes different implementations of the same algorithm, and the descriptions using the same presentation form, from each other that affects the time that a specialist needs to understand the principles of the algorithm.

IV. METHODOLOGY

The methodology of quantitative assessment could be implemented using two stages - recognizer construction described in Fig.1, and obtaining a quantitative assessment described in Fig. 2.

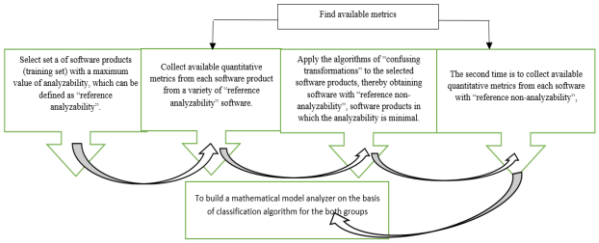


Fig.1 Recognizer construction

Recognizer construction. Carrying out all the necessary work on the collection, preparation of data, the development of a recognizer and training in its effective assessment of the Analyzability metric.

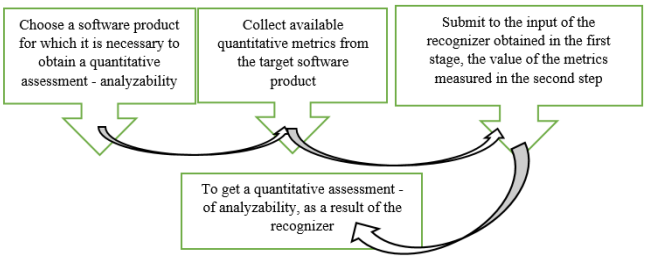


Fig. 2 Quantitative Assessment

Obtaining a quantitative assessment (analyzability) on the basis of the recognizer obtained is the first step. At the stage of recognizer construction, the participation of experts is necessary to form an expert assessment and specialists to obtain reliable recognizer results. Once the recognizer model is obtained, it can be used to obtain a quantitative assessment - analysis without having the special knowledge needed in the first stage.

V. ANALYZABILITY MODEL

To develop the proposed software module, Fig. 3 demonstrated the main tasks that were solved.

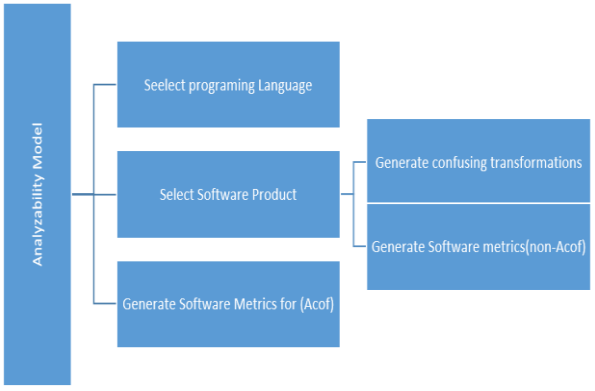


Fig. 3 Software model tasks

JAVA programming language was selected for evaluating the program code of which a quantitative assessment will be calculated to measure the analyzability. The Development software is based on the work from [17][18], and A set of 41 software products have been collected from [19] and Fig. 4 shows a sample of the selected software product.

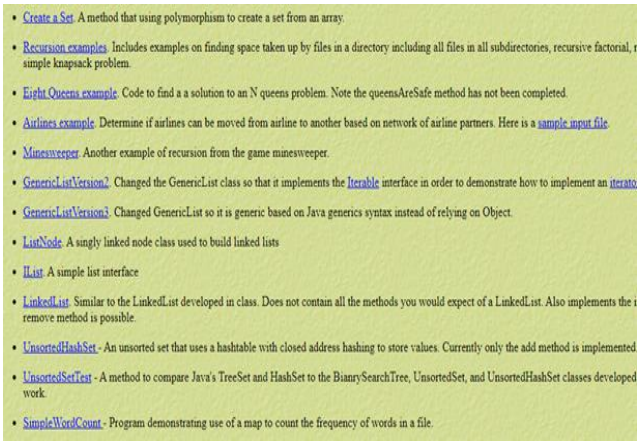


Fig.4 A sample of the selected software product

The selected software product was configured with a "reference analysis" that made up the training sample. A set of metrics were generated that is suitable for a given programming language and fully describe the source code of the programs, in so far as it relates to "analyzability". The software metrics were extracted from [14] http://www.virtualmachinery.com/sidebar2.htm: The Halstead metrics and as described in table 1.

TABLE I
A SAMPLE OF HALSTEAD METRICS

Metric	Metric code	Description
unique operators	(n1)	the number of distinct operators
unique operands	(n2)	the number of distinct operands
total operators	(N1)	the total occurrences of operators
total operands	(N2)	the total occurrences of operands
Program vocabulary	(n)	n = n1 + n2
Program length	(N)	N = N1 + N2
Calculated program length	(N')	N'=n1log2(n1)+n2log2(n2)
Volume	(V)	V = Nlog2(n)
Difficulty	(D)	D= (n1/2) * (N2/n2)
Effort	(E)	E= DV
The time required to program	(T)	T= E/18 seconds
Delivered bugs	(B)	B=V/3000
cyclomatic complexity	(Cc)	Cc= edges- Nodes + 2

A set of "confusing transformations" applicable to a given programming language has been formed. A study has been made of the influence of "confusing transformations" on the "analyzability" of the program code. A recognizer model was created using the Weka Tool. The topology, the number of hidden layers, neurons, the activation function and the learning algorithm of an artificial neural network was selected. The smart program was written in C# to extract the software metrics described in Table 2, all forty-one (41) program codes were classified as 'analyzability' with the help of software engineering experts.

Then, a group of seven professional programmers were required to modify the forty-one (41) program codes to complicate the control flow, data flow, the size of the program and any of its elements. This collection is called "confusing transformation" and considered as "reference unanalyzable" software. The modified program codes with the support of the experts were classified as non-analyzability. Finally, the data set was processed, developed and prepared for the classification process. Fig. 5 describes a sample of the developed data set. Each code example is designed as a separate application and represents a full-fledged Java software. Since each of these applications was prepared by the company's specialists as a "reference-analyzability" example of using the Java language, an assumption is made about the "reference-analyzability" analysis of these applications as one of the indicators of their quality.

For a metric evaluation of software products, a set of thirteen (13) metrics were generated that quantitatively measure various properties of software.

n1	n2	n	N11_A	N21_A	N1_A	N1_B	V	D	E	T	B	complexity	STD	Average	Res_Analyz
10	10	20	14	10	24	66.43856	103.7263	5	518.6314	28.81285	0.034575	1	140.1801	62.43413	analyzable
9	8	17	20	10	30	52.52933	122.6239	5.625	689.7594	38.31996	0.040875	1	186.9436	77.22295	analyzable
13	13	26	19	13	32	96.21143	150.4141	6.5	977.6915	54.31619	0.050138	2	264.8695	107.9372	analyzable
11	9	20	26	12	38	66.58307	164.2333	7.333333	1204.377	66.90985	0.054744	3	327.2789	125.2686	analyzable
12	8	20	25	12	37	67.01955	159.9113	9	1439.202	79.95567	0.053304	1	391.7433	143.8571	analyzable
12	8	20	25	12	37	67.01955	159.9113	9	1439.202	79.95567	0.053304	1	391.7433	143.8571	analyzable
14	17	31	37	23	60	122.7898	297.2518	9.470588	2815.149	156.3972	0.099084	3	767.6117	275.8583	analyzable
18	18	36	43	21	64	150.1173	330.8752	10.5	3474.19	193.0105	0.110292	3	947.9901	335.5233	analyzable
15	21	36	58	27	85	150.842	439.4436	9.642857	4237.492	235.4162	0.146481	3	1157.063	409.0756	analyzable
19	24	43	53	28	81	190.7497	439.5274	11.08333	4871.429	270.635	0.146509	3	1330.585	464.1978	analyzable
19	18	37	47	27	74	155.7693	385.4995	14.25	5493.369	305.1871	0.1285	2	1503.471	506.0156	analyzable
17	21	38	53	31	84	161.7255	440.8259	12.54762	5531.316	307.2953	0.146942	5	1512.931	515.6044	analyzable
8	19	27	61	57	118	104.7106	561.0767	12	6732.921	374.0512	0.187026	1	1843.979	621.2266	analyzable
18	17	35	59	32	91	144.5455	466.7648	16.94118	7907.544	439.308	0.155588	4	2168.349	710.0969	Non_analyzable
15	20	35	69	38	107	145.0419	548.8333	14.25	7820.874	434.493	0.182944	5	2143.037	711.7443	Non_analyzable
17	19	36	66	34	100	150.1975	516.9925	15.21053	7863.728	436.8738	0.172331	2	2155.316	712.0904	Non_analyzable
20	20	40	61	32	93	172.8771	494.9393	16	7919.029	439.9461	0.16498	5	2170.3	716.4582	Non_analyzable
13	32	45	88	50	138	208.1057	757.8757	10.15625	7697.175	427.6209	0.252625	10	2104.969	729.0143	Non_analyzable
22	33	55	74	37	111	264.5725	641.7309	12.33333	7914.681	439.7045	0.21391	5	2164.812	739.249	analyzable
13	29	42	86	50	136	188.9872	733.3552	11.2069	8218.636	456.5909	0.244452	10	2249.175	767.3092	Non_analyzable
20	21	41	72	32	104	178.6772	557.1854	15.2381	8490.444	471.6914	0.185728	3	2326.838	769.7248	analyzable

Fig. 5 Data set sample

The proximity matrix was setup and calculated to discover more information about the dataset. Fig. 6 shows the proximity matrix analysis process.

Proximity Matrix

Matrix File Input

Case	n1	n2	n	N11_A	N21_A	N1_A	N1_B	V	D	E	T	B	complexity
n1	.000	303.996	200.561	297.677	311.270	300.841	319.437	362.921	105.837	452.679	452.679	362.921	328.399
n2	303.996	.000	11.109	8.846	8.935	7.920	1.721	10.483	135.077	56.440	56.440	10.483	31.142
n	200.561	11.109	.000	18.231	20.716	17.956	15.912	32.112	87.857	89.188	89.188	32.112	43.639
N11_A	297.677	8.846	18.231	.000	3.995	.608	7.034	4.247	105.585	33.252	33.252	4.247	21.942
N21_A	311.270	8.935	20.716	3.995	.000	1.496	8.670	5.221	108.694	35.239	35.239	5.221	30.698
N1_A	300.841	7.920	17.956	.608	1.496	.000	6.775	3.829	105.497	33.529	33.529	3.829	24.593
N1_B	319.437	1.721	15.912	7.034	8.670	6.775	.000	5.987	137.495	43.017	43.017	5.987	23.428
V	362.921	10.483	32.112	4.247	5.221	3.829	5.987	.000	140.666	20.852	20.852	.000	22.441
D	105.837	135.077	87.857	105.585	108.694	105.497	137.495	140.666	.000	170.864	170.864	140.666	123.708
E	452.679	56.440	89.188	33.252	35.239	33.529	43.017	20.852	170.864	.000	.000	20.852	36.506
T	452.679	56.440	89.188	33.252	35.239	33.529	43.017	20.852	170.864	.000	.000	20.852	36.506
B	362.921	10.483	32.112	4.247	5.221	3.829	5.987	.000	140.666	20.852	20.852	.000	22.441
complexity	328.399	31.142	43.639	21.942	30.698	24.593	23.428	22.441	123.708	36.506	36.506	22.441	.000

Fig. 6 A sample of Proximity Matrix

Using "Weka" classifiers functions, a multilayer perceptron was selected for the classification process and to generate the analyzability model. Since at the moment, there are no reliable methods for the analytical determination of the number of neurons, hidden layers, activation function, and method of training Multilayer-Perceptron, the selection of the best configuration was carried out by completely sorting out the possible values for these characteristics. With the help of Weka 3.9 tools. Fig. 7 and Fig. 8 show a sample of the stages of choosing the best Multilayer-Perceptron configuration, which shows the configuration of the training and the ability to generalize Multilayer-Perceptron with one hidden layer and ten neurons in the hidden layer.

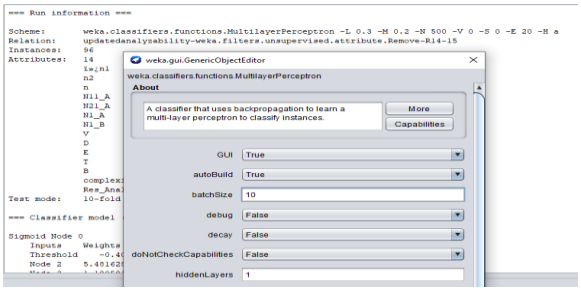


Fig. 7 Sample of Multilayer-Perceptron configuration

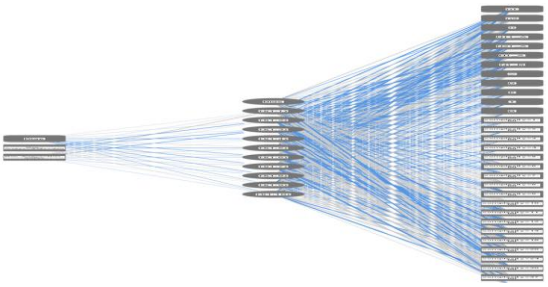
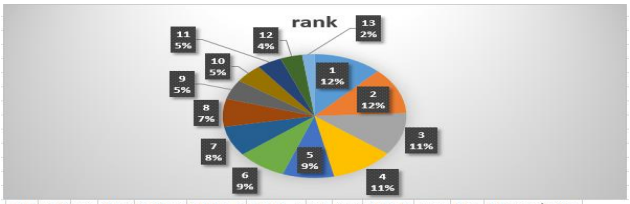


Fig. 8 A sample of the generated Multilayer-Perceptron layers

This work also calculates the information-gain for each software metric selected in this work, and Fig. 9 demonstrates the results. The information gain was carried out to identify the metrics that have the greatest influence on analyzability.



10 E	11 T	8 V	12 B	6 N1_A	4 N11_A	5 N21_A	3 n	9 D	7 N1_B	2 n2	1 n1	13 complexity
0.6	0.6	0.5	0.54	0.436	0.418	0.378	0	0.2	0.236	0.21	0.2	0.104

Fig. 9 The influence rank for each software metric

It can be seen from the diagram that the indicated metrics are more influenced by the assessment and analyzability. The finding also showed that Efforts, Time and Bugs play an important role in defining the level of analyzability. The main purpose of the experiment was to prove the effectiveness of the proposed approach to assessing the quality indicator "analysis". The following can be distinguished as additional objectives of the experiment: Specialists in the field of software development, with a total number of seven software engineers who worked as experts to evaluate and test the software in terms of the quality of "analyzability". Before the start of the experiment, we explained the essence of the experiment and proposed to evaluate each of the software products in terms of the quality of "analyzability".

After that, the values obtained on the basis of the methodology of quantitative assessment - analyzability were compared with an expert assessment.

In order to evaluate the diagnostic effectiveness of the method, considering the consequences of false decisions, characteristic curves were used. These reflect the mutual dependence of false-positive and true-positive results. To quantify the methods used, a comparative analysis of the areas under the ROC curves was implemented.

The curve of the proposed method has a larger area than the curve of the expert's evaluation. As can be seen in Fig. 10, the values obtained using the quantitative assessment methodology - analyzability, and Fig. 11 shows the values of "analyzability" obtained using expert evaluation are quite well connected. This indicates the effectiveness of the developed methodology in assessing the "analyzability" of software products.

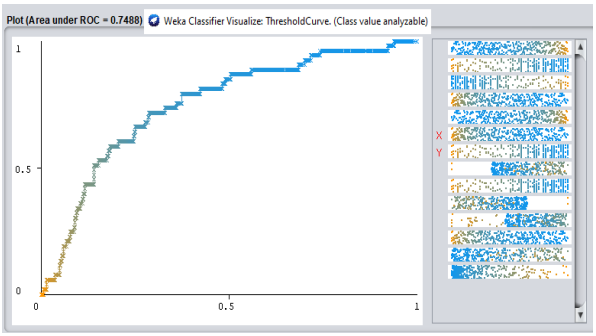


Fig. 10 ROC quantitative assessment methodology

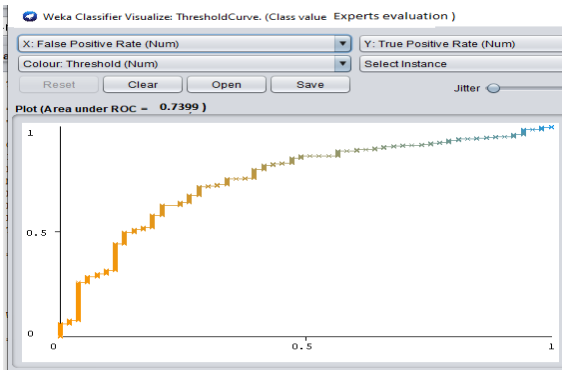


Fig. 11 ROC Expert Evaluation

IV. RESULTS

It is imperative to understand that the ROC Characteristic curves make it possible to visually compare the diagnostic effectiveness of various research methods. To plot curve points, this research work calculated their ordinates. In this case, the ordinate axis corresponds to the probability of truly positive decisions while the abscissa axis corresponds to the probability of false-positive decisions.

When analyzing ROC-curves, the following principle was adhered to: the closer to the upper left corner of the coordinate grid the curve is located, the higher the informative results of the investigated diagnostic method or the better the quality of the data display system. If the curve is adjacent to the diagonal (or coincides with it), then the results of the method are negligible. It should be noted that the criterion of "sensitivity" may act as truly positive decisions and the criterion of "1 - specificity" as false positives.

The ROC analysis method allows a comparative assessment of the information content of two visualization methods. This also compares the capabilities of the proposed method and the evaluation of the experts in detecting the level of the analyzability. The procedure for constructing ROC-curves for each of the diagnostic methods under consideration was then carried out. In Fig. 10, the relationship of ROC-curves is established: the curve that is located above corresponds to a more informative method.

The method of ROC analysis allows us to determine the quantitative value of the reliability of the differences in the information content of the studied methods. To do this, we calculated the area under the curves (Fig. 9.5) and, using special formulas, establish the confidence interval in the difference in the information content of the methods. It is generally accepted that the coefficient of the area of the curve in the range of *0.9-1.0* should be considered as an indicator of the highest informative of the diagnostic method - *0.8-0.9* - good, *0.7-0.8* - acceptable, *0.6 -0.7* - weak, *0.5-0.6* - extremely weak.

In Fig. 10, ROC-curves was constructed to compare the information content of the proposed method. The Curve is closer to the upper left corner with *ROC=7.4*. Therefore, it is more informative than a curve of expert evaluation with *ROC=7.3*. Both methods are accepted. However, the proposed method outperforms the expert evaluation. This experiment confirmed that "analysis" is not described by anyone metric but is a complex combination of several metrics. It is therefore important to note that the study made it possible to obtain a number of new scientific and technical results that are essential for quantitative assessment of the quality of software products. Based on the analysis of the state of the issue, in the field of evaluating and controlling the "analyzability" of software products, the main shortcomings of existing approaches were identified. This analysis made it clear of the need for a formalized definition of "analyzability". A methodology was developed for evaluating the program code for the subject of "analyzability", which allows one to obtain an objective quantitative assessment of the quality indicator "analyzability" in accordance with the requirements, limitations, purpose and specific features of the product. For validity, a program code recognizer was developed according to the "analyzability" quality indicator, with the ability to recognize a "new" program code, i.e. with the ability to generalize. It was found that for test software from thirteen (13) measured metrics, 98% of "analyzability" represents only seven software code metrics, the remaining six metrics make up only ~ 5% of "analyzability".

Similarly, a set of "confusing transformations" was formed using Java programming language, which affects the "pars ability" of program code. In the end, a study of the influence of "confusing transformations" on the metrics of the program code was carried out to allow us to create tools for studying the properties of program codes and its use in the construction of a recognizer - analyzer.

The analytical representation of the "analyzability" of the program code makes it possible, when designing software products, to set quantitative characteristics of product properties. For reliability, a complex of algorithms was then developed to implement an automated solution to the problem of a quantitative assessment of the quality indicator "analyzability". On this basis, a software complex was equally used to implement a quantitative assessment of analyzability through a dialogue with the user.

The resultant effects showed that the conducted simulation reflected a satisfactory convergence of the results of the expert assessment of "analyzability" and —analysis, measured by the developed methodology.

V. CONCLUSIONS

In conclusion, the experiments performed in this study discovered three major metrics (Effort, Time and Bugs) that potentially define the analyzability level as explained earlier. Since the aim of the proposed research work is to develop a methodology for quantitative assessment considering the analyzability quality indicators of software products, it is pertinent that solutions are provided based on the analysis in the field of evaluation and control of the software products, where the main weaknesses of existing approaches were earlier identified. The scientific novelty is that the problem was formulated and set-out for obtaining a formalized assessment of the quality indicator "analyzability", in accordance with the concept of analyzability of program code as introduced, defined and formalized.

During the methodology phase, a quantitative assessment of the quality indicator "analyzability", a recognizer analyzer were developed that was capable of effectively evaluating the "analyzability" of program codes. A set of metrics was identified that allows us to effectively evaluate the quality indicator "analyzability" in a particular class of programming languages. An analytical representation was obtained - of the program code being analyzed, which confirms the effectiveness of the proposed "analysis" assessment. The practical value of the work showed clearly the required quantitative value of the "analyzability" quality indicator as a technical task for software development. This helps maintain this value throughout the entire software product development cycle and helps in managing decisions based on quantitative indicators of a high-quality product in terms of "analyzability". At the conclusion of the study, a practical tool was also created for assessing the quality of software products in terms of "analyzability" to generate a competitive environment in order to form a set of metrics for the program code, and completely evaluate the quality indicator "analyzability". This provided an avenue to round up by formulating a confusing transformations test directly to affect the analyzability of the program code in order to test the influence on the program code metrics. The results ultimately produced an analytical representation from the program code which eventually confirms the effectiveness of the proposed "analyzability" assessment in this study. Finally, the outcomes of this study represented a set of algorithms that could implement an automated solution to solving the problems of assessing the software quality indicator "analyzability".

REFERENCES

1. Shannaq B, Alshamsi I, Azaw F (2020)," Innovative Algorithm for Managing the Number of Clusters", accepted of 'Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)' published in the 'International Journal of Recent Technology and Engineering (IJRTE)' that will publish at Volume-8 Issue-5, January 2020.
2. Sas, D. & Avgeriou, P. (2019) Quality attribute trade-offs in the embedded systems industry: an exploratory case study ", Software Quality Journal, https://doi.org/10.1007/s11219-019-09478-x,
3. Shannaq B., Ibrahim A, Saif I, (2019) Management Information System for Predicting Quantity Martials".TEM Journal, 8(4), 1143-1149, DOI: 10.18421/TEM84
4. Baqais A.,.& Alshayeb M. (2019) Automatic software refactoring: a systematic literature review", Software Quality Journal https://doi.org/10.1007/s11219-019-09477-y,
5. Arash H., Maryam A., (2019)" Extension of the model of manufacturing supply chain quality management: an empirical study", International Journal of Productivity and Quality Management, Vol. 28.
6. Bedir T., Richard S. (2017) Quality concerns in large-scale and complex software-intensive systems
7. Raibulet C., Carrillo C. (2017) An Overview on Quality Evaluation of Self-Adaptive Systems". Managing Trade-Offs in Adaptable Software Architectures, 2017.
8. Aleem, L. Capretz F, & Ahmed F. (2015)," BENCHMARKING MACHINE LEARNING TECHNIQUES FOR SOFTWARE DEFECT DETECTION ", International Journal of Software Engineering & Applications, DOI: 10.5121/ijsea.2015.6302
9. , Vol.6, No.3,
10. Okutan A & Yıldız O. (2014) "Software defect prediction using Bayesian networks", Empirical Software Engineering, Vol. 19, pp. 154-181,2014.
11. Ermiyas B, Feidu G., & Shumet N.(2018) ANALYSIS OF SOFTWARE QUALITY USING SOFTWARE METRICS ", International Journal on Computational Science & Applications, Vol.8, No.4/5.
12. Haibo C., Yuan L., Huang B., &. Pen C. (2009) Control Flow Obfuscation with Information Flow Tracking", National Science Foundation of China,.
13. Saradhi M.., Sastry B. (2010) Quality Indicator for Software Interoperability ", International Journal of Engineering Science and Technology, Vol. 2(7), 2587-2594, .
14. Suhel K., Raees K. (2013) Analyzability Quantification Model of Object-Oriented Design", International Journal of Software Engineering and Its Applications, Vol. 7, No. 4,.
15. Lee M., & Chang T.(2013) Software Measurement and Software Metrics in Software Quality", International Journal of Software Engineering and Its Applications, Vol. 7, No. 4, .
16. Ghuman S.(2014) Software Quality Matrics- An Overview", International Journal of Computer Science and Mobile Computing, Vol.3 Issue.5, pg. 1303-130,
17. **Shannaq B.,** Al Shamsi I.(2019) Innovative Web Service for Streaming Student Tweeting in Real-Time Technology", International Journal of Innovative Technology and Exploring Engineering (IJITEE)ISSN: 2278-3075, Volume-9, Issue-2,
18. Azawi F., Shannaq B.(2019) Fuzzy Analysis Tool for Classifying Exams Questions Based on Bloom's Taxonomy
19. Verbs , REVISTA AUS 26.4/ DOI:10.4206/aus.2019.
20. www.cs.utex.edu (2019) Java Coding Samples: website. [Online], retrieved: https://www.cs.utexas.edu/~scottm/cs307/codingSamples.htm.
21. n26.4.9

AUTHORS PROFILE

Dr. Boumedyen Shannaq, is a skilled university professor for 10 years, primarily teaching Information Systems and Information Technology Courses. Interested in Electronic Learning, Innovation, Artificial Intelligence, Analysing Big Data. Professional skills in Public Speaking, Time Management, Self-motivation, Inter-personal communication and Record-keeping, Focussed on Analyze ideas and use logic to determine their strengths and weaknesses. Manage oneself, people, time, and things. Obtained a Ph.D. in Information Systems (data Engineering-Technical Issues).

Dr. Richmond S. Adebiaye, is a Faculty at the University of South Carolina Upstate. He is CISSP & CISM certified. He has over fourteen (14) years' experience teaching and developing various courses in Information Systems, Digital Forensics and Security. He also has over 22 published research articles in various high impact scholarly journals. His main expertise and research interests involve Security Architecture, Data Analytical Science, Privacy-enhancing technologies imprecision, Single sign-on, Security Level Management, Audit-proof Network management and protocols