REVIEW ARTICLE | UHD JOURNAL OF SCIENCE AND TECHNOLOGY

An Empirical Evaluation of Metrics on Aspect-oriented Programs

Mazen Ismaeel Ghareb[1], Gary Allen[2]

[1]Department of Computer Science, College of Science and Technology, University of Human Development, Sulaymaniyah, Kurdistan Region of Iraq, [2]Department of Informatics, School of Computing and Engineering, University of Huddersfield, Huddersfield, England

ABSTRACT

The quality evaluation of software metrics measurement **is** considered as the primary indicator of imperfection prediction and software maintenance in various empirical studies of software products. However, there is no agreement on which metrics are compelling quality pointers for new software development approaches such as aspect-oriented programming (AOP) techniques. AOP intends to enhance programming quality by providing fundamentally different parts of the systems, for example, pointcuts, advice, and intertype relationships. Hence, it is not evident if quality characteristics for AOP could be extracted from direct expansions of traditional object-oriented programming (OOP) measurements. Then again, investigations of AOP do regularly depend on established static and dynamic metrics measurement; notwithstanding the late research of AOP in empirical studies, few analyses been adopted using the International Organization for Standardization 9126 quality model as useful markers of flaw inclination in this context. This paper examination we have considered different programming quality models given by various authors every once in a while and distinguished that adaptability was deficient in the current model. We have testing 10 projects developed by AOP. We have used many applications to extract the metrics, but none of them could extract all AOP Metrics. It only can measure some of AOP Metrics, not all of them. This study investigates the suitable framework for extract AOP Metrics, for instance, static and dynamic metrics measurement for hybrid application systems (AOP and OOP) or only AOP application.

Index Terms: AspectJ, Aspect-oriented Development, Aspect-oriented Programming, Aspect-oriented Programming Metrics, Hybrid Application System, Software Quality Metrics

1. INTRODUCTION

The research aims to investigate new metrics of aspect-oriented programming (AOP) quality measurements. It has been used open-source tools for collecting static metrics. These static metrics will feed to external quality characteristics. The combination of other dynamic metrics will expand the quality measurement for the AOP system based on specific quality International Organization for Standardization standards. The theoretical contribution of other software quality frameworks has been discussed. This thesis proposed a unique software product quality for hybrid software applications (object-oriented programming [OOP] and AOP) to identify product quality metrics. Static measurements are used to evaluate the quality of computer code. Cohesion, for example, is the degree to which components of a module work together with each other. At present, there are not many measurements for aspect-oriented (AO) frameworks [1], for example, cohesion, and coupling, separation of concerns, size, and so forth. Cohesion is one of the essential quality properties for AO frameworks. Coupling measures the level of interaction between modules. Low coupling and high cohesion are considered necessary

Access this article online

DOI: 10.21928/uhdjst.v3n2y2019.pp74-86 | **E-ISSN:** 2521-4217 **P-ISSN:** 2521-4209

Corresponding author's e-mail: Mazen Ismaeel Ghareb, Department of Computer Science, College of Science and Technology, University of Human Development, Sulaymaniyah, Kurdistan Region of Iraq. E-mail: mazen.ismaeel@uhd.edu.iq/mazen.ghareb@hud.ac.uk

Received: 05-08-2019 Accepted: 21-10-2019 Published: 23-10-2019

for good design. In Cazzola *et al.* [2], W. Cazzola inferred one structure given dynamic element metrics and run time execution of software reports from several research papers [3]-[5] and concentrates on element estimations dismissing static, traditional measurements. The importance of software measurement has been increasingly recognized object-oriented (OO) software engineers as the metrics have been proven to be pointers of vital quality properties, for example, the fault-proneness of the final system [4]. In this manner, the quality pointers for AOP can be gotten from direct expansions of traditional OO measurements. In any case, observational investigations of AOP do frequently depend on established coupling measurements.

2. STATE OF ART

In the past decades, many OO metrics have been proposed. The most well-known metrics are the Chidamber and Kemerer (C&K) metrics [6] and metrics for OO Design (MOOD) [7], which are applied to the quality of OOP at different stages. To be specified, C&K measurements are, for the most part, used to assess single classes, while MOOD measurements are used to survey entire frameworks. All the metrics can be divided into seven categories [8].

1. Size and complexity: Number of methods (NOM) and number of attributes (NOA) are used to measure the size of the class in terms of method and attributes. Weight method complexity (WMC) and class complexity are connected to measure classes and are used to measure total complexity by calculating the total number of functions/methods in different ways. Since classes are proposed to be designed as succinctly as possible, these measurements are required to be low in their qualities.
2. Cohesion: Cohesion is measured with four class level measurements, which are calculated in various approaches to reflect the collaborations between part function/methods. The four levels are Lack of Cohesion in Methods (LCOM), Tight Class Cohesion, Loss Class Cohesion, and information-based cohesion.
3. Reusability: Reuse ratio and specification ratio are both framework level reusability measurements. They are calculated as the ratios of subclasses to all classes and superclasses, separately. Since classes are relied on to be profoundly reused, extensive reusability metric qualities are desired.
4. Polymorphism: NMO overridden (NMO) by the class and polymorphism factor (PF) are polymorphism measurements at various levels. To be particular, NMO is a class level metric, which measures the NMO by a single subclass, while PF is a framework level metric, which measures the degree of method overriding in the entire system.
5. Inheritance: Number of children (NOC) and depth of inheritance tree (DIT) are class level measurements, which express class inheritance through the number of relatives and the depth of the inheritance, respectively. By comparison, method inheritance factor and attribute inheritance factor are framework level measurements, which refer to method inheritance and attribute inheritance.
6. Encapsulation: Method hiding factor and attribute hiding factor are indicators to show how well methods and attributes are hidden inside classes. These measurements are measured at the framework level.
7. Coupling: Five measurements are used to assess class coupling from other points of view. The coupling factor metric is used to determine the coupling of all classes at the framework level. By examination, the other four measurements measure coupling at the class level. Among these measurements, response for the class (RFC) and message passing coupling are utilized to survey technique coupling, data abstract coupling encapsulates information coupling among classes, and coupling between objects (CBO) indicates a coupling between class occurrences.

2.1. AO Metrics

Several reviews (Rønningen and Steinmoen, Zhang and Jacobsen, Mickelsson, Coady and Kiczales, and Tsang *et al.* [9]-[13]) have been conducted into the use of metrics within AOP, more often by applying the measurements characterized for OOP. Little work has been done to identify measurements suitable specifically to AOP. In his research, Mickelsson [11] concentrates on size measures such as several classes, functions, and source explanations. Coady and Kiczales [12] considered runtime costs and the position of hidden concerns in working framework code. Zhang and Jacobsen [10] utilized cyclomatic complexity, estimate, the weight of class, coupling amongst articles, and reaction time for their assessment. Tsang *et al.* [13] connected the C&K measurement suite in their evaluation of AO systems. They concentrated on the quality components understandability, viability, reusability, and testability, and the C&K meanings of measurements suited for measuring these elements. These measurements are weighted technique calls, DIT, NOC, the CBOs, the reaction for a class, and LCOMs. Zakaria and Hosny [14] described the C&K measurements suite and the impact of AO on these measurements. Burrows *et al.* [15]

utilized 10 metrics for their analysis. These measurements were characterized in Ceccato and Tonella's measurements suite and established OO measures proposed by C&K measures. Adding to that, they offered a new metric that evaluates coupling between the base and Aspect code, named base-aspect coupling. Burrows *et al.* [16] utilized the C&K metric for their review, and they concentrated on identifying coupling in AO developed systems. Dhambri *et al.* [17] proposed a complex AO software analysis using visualization techniques. The initial phase defines the characteristics and measurement for AOP programming, intended to measure the general principle of quality. They used the app metrics tool to extract many AOP Metrics based on OOP metrics extend to AOP. They have been used visualization techniques of the selected system for analysis of quality for large-scale software systems [17]. Dhambri *et al.* [17] suggested that metric definitions, as well as quality analysis, should be adapted in an AO form, and they have recorded some interesting open questions that are relevant to any related methodologies. Specifically, they attempt to formally characterize the advantages of AOP, such as the separation of concerns and (Un)pluggability.

Proposed an AOP Metrics tool measures all code written in the AspectJ language. The tool exploits a static analyzer developed in the source transformation tool TXL. The main module takes as inputs all the source classes, interfaces, and aspects and performs standard OO reverse engineering. The second module can then be run, performing further propelled reverse engineering. The next module of the tool identifies the method call relationship. Furthermore, it finds the field-access relationships between operations and fields. The fourth step is the most complex one. It finishes weaving by settling all the pointcuts in the aspect code, in this manner creating the comparable join points in the captured code. The last step concerns the calculation of the metrics. Their results show that essential properties, for example, the proportion of the system affected by aspects and the amount of knowledge an aspect has of the modules it crosscuts, are captured by the proposed metrics (CDA and coupling on intercepted modules [CIM], respectively).

2.2. Tools for AOP Metrics

The Ajatoo tool [18] has been evaluated for gathering the aspect metrics based on C&K [14] metric measurements by applying it to three different design patterns implemented in OOP and AspectJ (Observer, Decorator, and Adapter). As shown in Fig. 1, the software supports that various metrics include sizes (vocabulary size, NOA, number of operations, weight operator per component, lines of codes [LOC]) and coupling (DIT). No other metrics were supported or implemented yet. What is needed is a tool to help all AspectJ metrics and AOP Metrics generally.

It has been chosen projects developed by AOP, as shown in Fig. 2.

The drawback of Ajato tool does not support all versions of AO projects; it shows errors to extract and evaluate.aj extension files, as shown in Fig. 3.

A second tool is called AOP Metrics. This tool is developed in AspectJ. It measures many C&K measurements such as lines of class code, weighted operations in module, DIT,

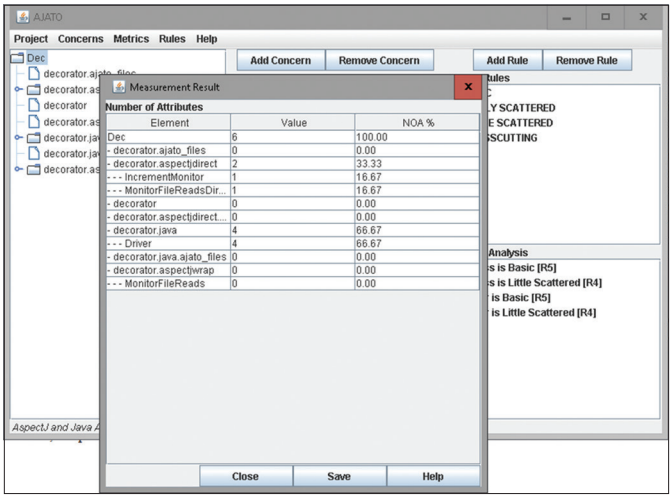

Fig. 1. Using Ajato tool.

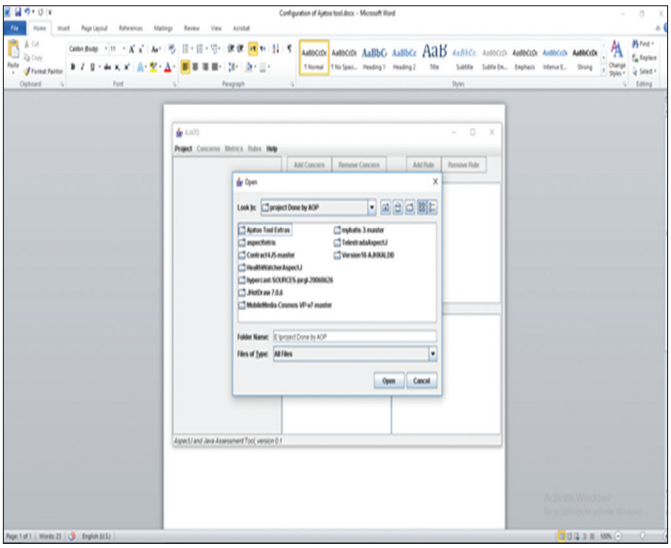

Fig. 2. Ajato tools read the projects.

NOC, lack of cohesion in operations, CIM, and package dependencies such as Abstractness (A) and number of types [Ref]. The drawback of the tool is developed with old versions for both AspectJ and Java and has not been updated for 8 years.

Moreover, it will not support aspect metrics to measure aspects independently [19]. Another tool Verifysoft's CMTJava [20] is a complexity measure apparatus expected to be a guide in testing, quality assurance, and implementing company standards for code complexity. CMTJava utilizations McCabe's cyclomatic number, lines of code measurements, number of semicolons, and Halstead's measurements in its computations. Another tool JDepend [21] plans to break down the design of the framework as far as extensibility, reusability, and practicality and is an aid to oversee and control the package dependencies. The tool proposed to utilize is to automatically watch that the designs display expected qualities while experiencing persistent refactoring by the engineers. JDepend gives measurements to Number of Classes and Interfaces, Afferent and Efferent Couplings, Abstractness, Instability, Distance from the Main Sequence, and Package Dependency Cycles. Adding more JMetric [22] is a result of an exploration extends at Swinburne University and expects to bring OO metrics and measurement research to the practitioners. JMetric gathers data from Java source documents and assembles a measurement display. The model is then populated with measurement data, for example, LOC, statement count, LCOM, and cyclomatic complexity. Measurements 1.3.5 are an open-source module for the Eclipse IDE [23]. It ascertains 17 unique measurements and gives a package dependencies analyzer, and is accordingly an overall apparatus.

Rapid miner tool has been tested for creating a new model design for process any rules design by the user. Our objective is to have AOP Metrics and shows statistical and graphical relationships between them. Unfortunately, it only takes text and CSV files, and we manually need to customize the rules, so it was not suitable for our experiments [24], as shown in Fig. 4.

Adding more there are very powerful texts engineering software called Gate has been tested for finding AOP source

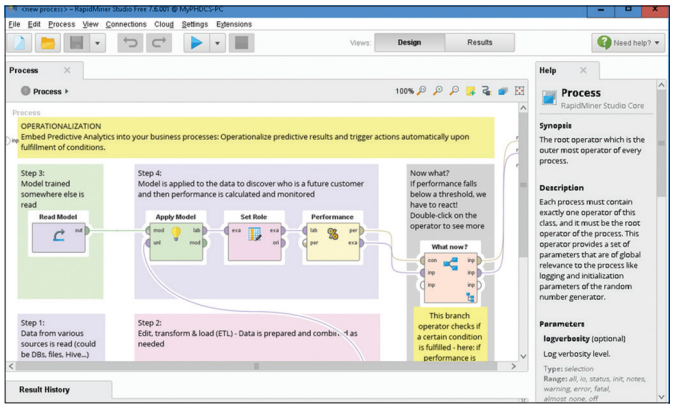

Fig. 4. Using rapid miner tool.

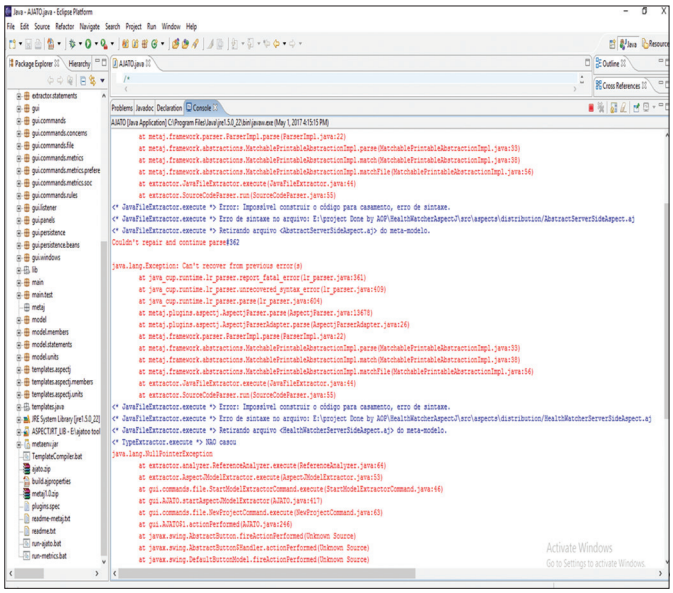

Fig. 3. Errors in Ajato tool.

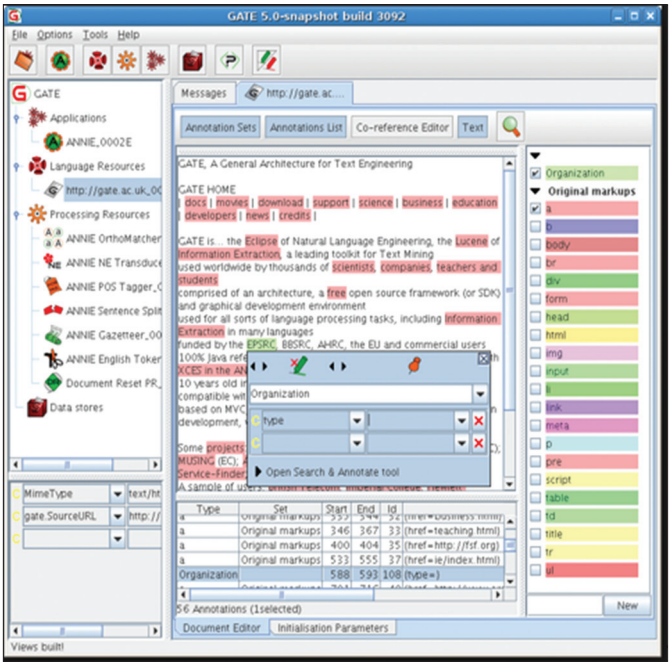

Fig. 5. Using gate text engineering tool.

code patterns. This tool help to find AOP Metrics and give statistical measures, after testing it shows some drawbacks, which is not given statistical information about each text pattern for each class and not read java file, as shown in Fig. 5.

3. AOP QUALITY METRICS

There are different models to evaluate the quality of OO and module arranged approaches. Various quality models for programming quality are given by Lincke *et al.*, [22], Yeresime and Pati [25], Eric and Bernstein [26], Lee [27]. However, less work is done to assess the quality attributes of the AO system. Different quality characteristic is complexity, coupling, reusability, changeability, maintainability, cohesion, and so forth.

- Complexity is the technique to analyze the code, endeavors required in change, and modification of modules
- Coupling is the level of relatedness among the modules that are an association between modules. Low coupling is needed
- Reusability is utilizing the module again to decrease the coding. There are different programming measurements for these qualities for inheritance systems. However, not very many measures are talked about for an AO system. Programming measurements go about as pointer of nature of a framework, i.e., give quantitative premise
- Maintainability is a change of programming item after delivery
- Cohesion is the level of relatedness among components of the modules. High cohesion is attractive.

There are numerous product quality models that propose approaches to integrate distinctive quality attributes each model aides in seeing how few quality components add to the entire variety. To assess the whole nature of the product item, we should see this large picture.

The objective of the AOP Metrics version 0.3 tool is to give a typical measurement instrument to the article arranged and the perspective situated programming. The task means to provide the following highlights (not executed ones are in italics):

The aspect arranged augmentations of the following measurements suite:
- C&K measurements suite (CK measurements)
- Robert Martin's measurements suite (package dependencies measurements)
- Henry and Li measurements suite.

Measurements actualized by AOP Metrics tasks can be connected to classes and aspects. Consequently, the module will be used as a typical term for classes and Aspects. Also, methods and advice will be demonstrated by the task term [28].

CK figures class level and metric level code measurements in Java extend by methods for static investigation (for example, no requirement for accumulated code). At present, it contains a massive arrangement of measures, including the celebrated CK.

This tool uses Overshadowing's JDT Center Library in the engine for AST development. At present, the consistency rendition is set to Java 11.

This apparatus will separate these static measurements:
- CBO: It tallies the number of conditions a class has. The device checks for any sort utilized in the whole class (field revelation, technique return types, variable statements, and so on). It overlooks conditions in Java itself (for example, java.lang.String)
- DIT: It tallies the quantity of "fathers" a class has. All classes have DIT at any rate 1 (everybody acquires java.lang.Object). Classes must exist in the undertaking (for example, if a class relies on X, which depends on a container/reliance document, and X relies on different classes, DIT is considered 2)
- Several fields: It checks the number of fields. Exact figures for an absolute number of fields, static, open, private, ensured, default, last, and synchronized fields
- Several methods: It checks the NOM. Explicit numbers for all outnumber of techniques, static, open, theoretical, private, secured, default, last, and synchronized methods
- Number of static summons: It checks the number of calls to static methods. It can just tally the ones that can be settled by the JDT
- Reaction for a Class (RFC): It checks the quantity of special strategy summons in a class. As summons are settled through static examination, these execution methods when a technique has overloads with the same number of parameters, however, various sorts
- Weight methods class (WMC) or McCabe's unpredictability: It checks the number of branch directions in a class

- LOC: It tallies the lines of the check, disregarding void lines
- LCOM: Figures LCOM metric. This is the absolute first form of metric, which is not stable. LCOM-HS can be better (ideally, you will send us a force demand)
- Amount of returns: The number of return instructions
- Amount of loops: The number of loops (i.e., for, while, do-while upgraded for)
- Amount of correlations: The number of examinations (i.e., ==)
- Amount of try/catch: The quantity of try/catch
- Amount of parenthesized expression: The amount of expression inside the bracket
- String literals: The number of string literals (e.g., "Kurd people"). Rehashed strings consider commonly as they show up
- Amount of number: The number of numbers (i.e., int, long, two-fold, skim) literals
- Amount of math tasks: The number of math activities (times, separate, leftover portion, besides, short, left poop, right move)
- Amount of variables: Number of announced variables
- Max settled blocks: The most astounding number of blocks decided together.

Amount of anonymous classes, subclasses need to be measure. It is important to check name of the class, count of each keyword in the class and all the when you execute the project. It recommended to Use of every factor, for instance, how much every element was used inside every method, adding more it recommended it need using of each field, and calculated inside every method [29].

McCall *et al.* [30] proposed a product quality factor structure and arranged the quality properties into three general classes:

i. Product task elements,
ii. Product amendment variables,
iii. Product change components.
    - Item task factors: The variables which add to item activity are rightness, unwavering quality, productivity, honesty, and ease of use
    - Item modification factors: The components which add to item update are practicality, adaptability, and testability
    - Item change factors: The variables which add to item progress are movability, reusability, and interoperability.

Boehm model is like the McCall model. Boehm spoke to their quality model as a various leveled tree and broken quality attributes into subqualities, which is given in Fig. 6. Boehm additionally incorporated the equipment yield attributes which were not considered in the McCall model. Utilizing this model, quality as a single parameter can be assessed due to the progressive nature of the model. This model does not give rules to quantify recorded attributes [31].

Dromey [32] proposed a quality model that gives a straightforward procedure for building quality conveying properties into programming. This model builds up the connection between unmistakable item qualities and less substantial traits. This model aids us where to search for imperfections and shows the properties that should be damaged to make abandons. This model tends to item quality by characterizing all the related sub-attributes so that they can be blended and amalgamated into higher-level qualities. The model backings are incorporating variety with programming, the meaning of language-specific coding benchmarks, efficient arranging quality deformities, and the advancement of mechanized code inspectors for identifying quality imperfections in programming. Dromey included reusability as a quality trademark in his model.

Deutsch and Wills [33] sorted programming quality as programming methodology quality and programming item quality. As a normal for programming item advancement process, programming strategy quality comprises programming designing related components such as technology, tools, workforce, association, and equipment. As a normal for programming item, programming item quality incorporates record transparency, trustworthiness, follow capacity, association, program rerisk, and test honesty. In this model, the direction has been given on what venture to follow to acquire the wanted item.

Word and Venkataraman [34] proposed that product quality measures may incorporate at least one of the accompanying: (i) Client based: As assessed by clients, programming quality alludes to the level of fulfillment of client desires. (ii) Item conveyance based: Determined by the item as evaluated by the originator, the level of framework viability, and program practicality; (iii) assembling based: The advancement procedure, stressing quality control, and the board; and (iv) authoritative control based: venture costs, generation time, asset control, and hazard the board.

Sharma *et al.* [35] proposed a quality model in the setting of Part Based Programming Improvement (CBSD) and have gotten their model from ISO/IEC 9126. In their model, they included reusability, intricacy, track capacity, and adaptability as new subqualities in the six attributes of ISO/IEC 9126.

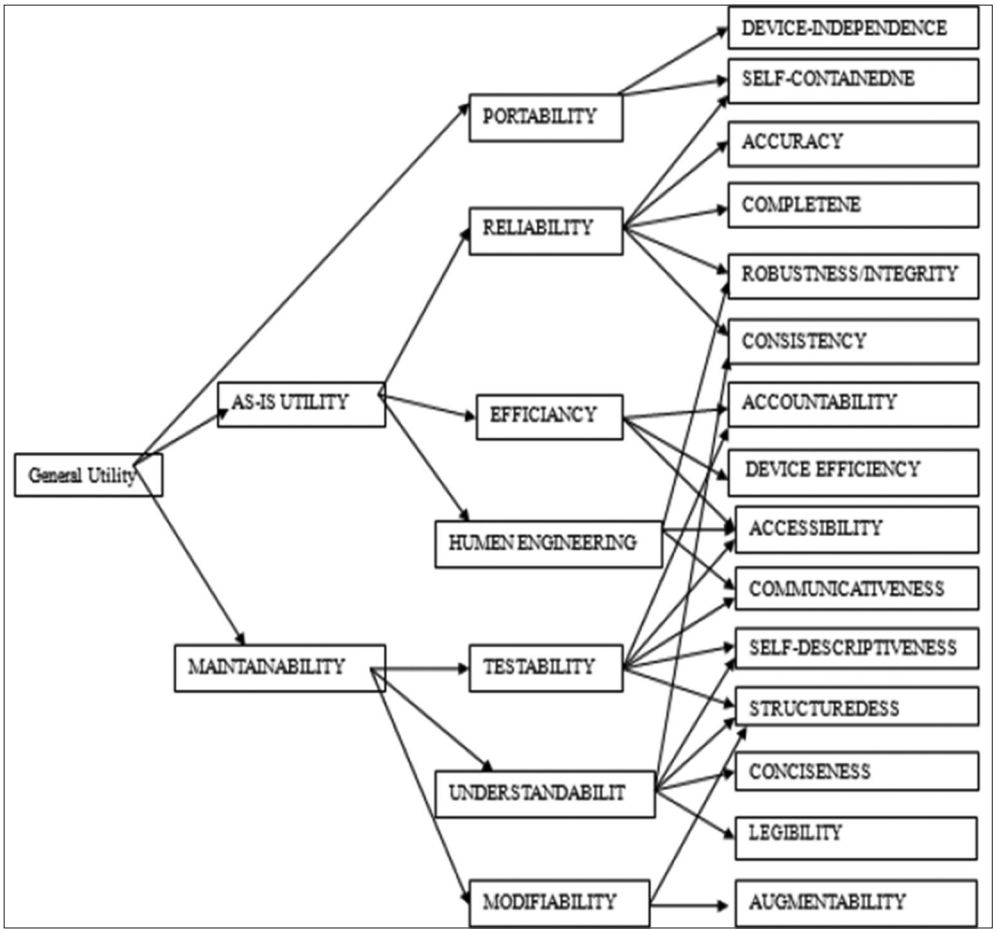

Fig. 6. Boehm quality model.

To assess the absolute nature of a segment, they utilized a systematic, progressive system process (analytic hierarchy process [AHP]), and for the weight estimations of value qualities and sub-attributes, an overview was led.

Chang *et al.* [36] gave rules to assess programming quality by incorporating a fuzzy hypothesis and AHP. They connected this new idea to the ISO/IEC 9126 quality model. Rather than taking the mean of gathered example information, they compared the fuzzy hypothesis to get relative loads of qualities and sub-attributes. They have not proposed any new quality model, yet have given rules to assess programming quality utilizing existing models [37], [38].

ISO/IEC 9126 tends to three programming quality aspects: (I) Process quality, (ii) item quality, and (iii) item being used quality. ISO/IEC 9126 arrangement guidelines have presented a various leveled model with six noteworthy quality attributes. These quality attributes are isolated into 21 sub-attributes, which add to the inner quality. ISO/IEC 9126 1 is concerned basically with the meaning of value qualities and sub-attributes in the last items. ISO/IEC 9126 2 gives external measurements to estimating traits of six external quality attributes characterized in ISO/IEC 9126 1. ISO/IEC 9126 2 distinguishes external measures; ISO/IEC 9126 3 describes internal sizes, and ISO/IEC 9126 4 characterizes quality being used measurements for the estimation of value attributes or the subqualities [39], [40]. Inward analyses estimate programming itself; outside measurements estimate conduct of the computer-based framework that includes the product, and quality being used sizes measured the impacts of utilizing the product in a particular set of utilization. ISO/IEC 9126 2:2003 is proposed to be used together with ISO/IEC 9126 1. ISO/IEC 9126 4 contains:

- A clarification of how to apply programming quality measurements;
- A fundamental arrangement of measurements for every trademark; and
- A case of how to apply measurements during the product item life cycle.

Even though the ISO/IEC 9126 quality model is genuinely detailed, it does not cover some significant quality attributes, which add to the nature of the AOP [41], [42].

To characterize the programming quality model, which should cover every one of the highlights of the AO programming framework, we need to see what are the new stresses and confinements of the latest technology as AO innovation is an expansion of MO or OO innovation. AO programming languages are additionally augmentations of local programming language, for example, perspective C is an expansion of C, Aspect C++ is an augmentation of C++, AspectJ is an expansion of Java, CaesarJ is an expansion of Java, Aspect XML is an expansion of XML, etc. Since AO technology cannot exist without anyone else, it will have every one of the highlights of the innovation from which it is determined. For instance, AspectJ has every one of the highlights of Java and extra highlights, which has been included AspectJ. At the end of the day, if AO Technology is gotten from OO Technology, at that point, it will have every one of the highlights OO innovation and extra highlights added to aspectual code. Other quality attributes/subqualities are required to be included, which can cover new highlights of crosscutting concerns (aspect(s)) and the combination of it with essential concerns (classes). The redefinition of existing attributes/subqualities, in the setting of AO innovation, is likewise required [43], [44].

The vast majority of the product quality models, which are proposed after the meaning of programming quality principles ISO/IEC 9126, have been gotten from ISO/IEC 9126. For instance, Rawashdeh and Matalkah [45] have included similarity as a sub-trademark to usefulness, unpredictability as a sub-trademark to ease of use and reasonability as a sub-trademark to viability in their product quality model. They evacuated subqualities strength and analyzability from practicality. They likewise included another brand in their quality model as partners, who are the individuals from the group in charge of creating, keeping up, incorporating, and utilizing the best framework. This model spotlights on the estimation of the nature of part based framework. Bertoia and Vallecillo [46] proposed a quality model, which characterizes attributes and sub-attributes in segment-based frameworks. In this model, sub-attributes have been partitioned into runtime and lifecycle classifications dependent on their inclination. They likewise included ability as a sub-trademark to usefulness, which demonstrates whether the previous variant of the part is perfect with its present form.

Quality Type	Characteristics	Sub-characteristics
Software Product Quality	Functionality:C1	Suitability:SC11
		Accuracy:SC12
		Interoperability:SC13
		Compliance:SC14
		Security:SC15
		Reusability:SC16
	Reliability:C2	Maturity:SC21
		Fault tolerance:SC22
		Recoverability:SC23
	Usability:C3	Understandability:SC31
		Learn-ability:SC32
		Operability:SC33
		Complexity:SC34
	Efficiency:C4	Time behavior:SC41
		Resource behavior:SC42
		Code-reducibility:SC43
	Maintainability:C5	Analyzability:SC51
		Changeability:SC52
		Stability:SC53
		Testability:SC54
		Modularity:SC55
	Portability:C6	Adaptability:SC61
		Install-ability:SC62
		Replace-ability:SC63
		Conformance:SC64

Fig. 7. Aspect-oriented software quality model.

Fig. 8. SQuaRE series of standards.

TABLE 1: The correlation of various qualities attributes in different programming quality models.

Model	Mcall (1977)	Boehm model (1978)	ISO9126 (1991)	FURPS model (1991)	Ghezzi model (1991)	ISO9126 V.4 (1992-2001)	Dromey (1995)	AOSQAMO (2009)	ISO/IEC 25010 (2010)	REASQ (2010)	AOSQ (2012)
Function											
Structure	Hierarchical	Hierarchical	Hierarchical	Hierarchical	Hierarchical	Hierarchical	Hierarchical	Hierarchical	Hierarchical	Hierarchical	Hierarchical
Number of levels	2	3	2	2	2	3	2	2	3	2	3
Advantage	Evaluation criteria	Hardware factors included	Evaluation criteria	Separation of FR and NFR	Evaluation criteria	Evaluation criteria	Evaluation criteria	Evaluation criteria	evaluation criteria	Evaluation criteria	Evaluation criteria
Disadvantage	Components overlapping	Lack Of criteria	Generality	Portability not considered	Lack of criteria	Lack of criteria	Comprehensiveness	Lack of criteria	Generality	Segments overlapping	Generality
Reliability	*	*	*	*	*	*	_	*	*	*	*
Efficiency	*	*	*	_	_	*	_	*	*	*	_
Integrity	*	_	_	_	_	_	_	_	_	_	_
Usability	*	_	*	*	*	*	_	*	*	*	_
Maintainability	*	*	*	_	_	*	_	*	*	*	_
Testability	*	*	_	_	_	_	_	_	_	_	_
Flexibility	*	_	_	_	_	_	_	_	_	_	_
Portability	*	*	*	_	_	*	_	*	*	*	*
Reusability	*	_	_	_	_	_	*	*	*	_	_
Understandability	_	*	_	_	_	_	_	_	_	_	_
Modifiability	_	*	_	_	_	_	_	_	_	_	_
Functionality	_	_	_	*	_	*	_	*	*	*	*
Supportability	_	_	_	*	_	_	_	_	_	_	_
Performance	_	_	_	*	_	_	_	_	_	_	_
Evolvability	_	_	_	_	_	_	_	_	_	_	*
Interoperability	*	_	_	_	_	_	_	_	_	_	_

The variables which are available in the model are set apart as "*," which means that this factor has been incorporated into the individual model, and the variables which are not present are set apart as "–" saying that these elements are not incorporated into the model. FR: Functional requirement, NFR: Non-functional requirement

According to Kumar *et al.* [47] proposed augmentation of the ISO/IEC 9126 quality model. In this quality model, we have included particularity as a sub-trademark under practicality, code reducibility as a sub- characteristic under effectiveness, multifaceted nature as a subfeature under ease of use, and reusability as a sub-characteristic under usefulness. The new model is given in Fig. 7. In this table, qualities and sub-attributes have likewise been allotted marks such as C2 (specific number 2) and SC32 (specific number 2 of distinctive number 3), which will be alluded in the following segments. It is important to show the significance of (ISO/IEC 9126) regarding qualities and sub-attributes, in the setting of AO technology.

SQuaRE comprises a group of principles under the general title of software product quality requirements and evaluation (Fig. 8 represents the association of these families or divisions [41], [48], [43]. The divisions inside the SQuaRE Model are as follows: ISO/IEC 2500n – Quality Management Division, ISO/IEC 2501n – Quality Model Division, ISO/IEC 2502n – Quality Measurement Division, ISO/IEC 2503n – Quality Requirements Division, and ISO/IEC 2504n – Quality Evaluation Division. In standard, ISO/IEC 25000-Guide to SQuaRE speaks to the umbrella archive of the SQuaRE arrangement; it gives a general outline and advisers for utilizing the SQuaRE series. This record contains the SQuaRE engineering, wording, planned clients, and associated parts of the arrangement. ISO/IEC 25000 presents the entire SQuaRE arrangement as an accumulation of value designing instruments. We are keen on the ISO/IEC 25030 (quality prerequisites) and the ISO/IEC 25010 (quality model, once in the past called ISO/IEC 9126-1), which will be displayed in what follows. In Fig. 2 – explains the architecture of the SQuaRE framework [41], [48], [43]. Adding more ISO/IEC 9126-1 and ISO/IEC 25010 it focus on software components is built to be agreeable with specific needs, required by its user. Their quality is resolved in the measure that these necessities are accomplished.

ISO/IEC 25010 is an update of ISO/IEC 9126-1 [49], with minor changes. According to the draft variant [41], it essentially keeps up similar definitions and structure of ISA/IEC (2001), anyway it offers eight attributes: A similar six characteristics of ISO/IEC (2001) or more interoperability and security, which were disposed of from the usefulness sub-attributes, for a sum of eight abnormal state characteristics. This decision reacts to the quality necessities particular of current software applications, for instance, web administration applications, where interoperability and security are building primary concerns. This work will consider ISO/IEC 9126-1 because it is formally received a standard.

The requirements, aspects, and software quality (REASQ) applied model, communicated in unified modeling language [50], encourages thinking on the fundamental ideas innate to a perspective situated quality prerequisites designing discipline. In the model, the product necessity, concern, and quality trademark components are the primary thoughts used to interrelate the wording of three ambits: The prerequisites building discipline.

4. CONCLUSION AND FUTURE WORK

Software users consider programming to be an apparatus to be utilized to help them in the manner they work together in their particular Structure. Quality is a structure of numerous attributes. Therefore, quality is usually caught in a model that portrays the characteristics and their connections. The models are helpful; they show what individuals believe is significant when talking about quality. Extraordinary associations utilize distinctive quality models dependent on the aspect-oriented software development (AOSD) worldview, and the product item quality detail. Using REASQ, a mapping is built up between the ISO/IEC guidelines and the developing AOSD discipline. Non-practical concerns and quality necessities are connected with at least one quality attribute of the standard quality model (potential cross-cutting concerns), which is a principal objective of AOSD [6], [51]). There is a general concurrence on the way that a perspective takes care of the issue of the crosscutting concerns (gave these are recognized), by typifying them in a particular structure, through an arrangement component [51]-[53]. The technique is possible from the get-go in the product improvement process through a structure table [54], while demonstrating the framework engineering, for example, to encourage the plan and execution stages.

Ghezzi *et al.* expressed that interior characteristics manage the structure of programming, which helps the product engineers to accomplish the external features just as essential attributes of programming, which are accuracy, flexibility, integrity, practicality, portability, reliability, reusability, and convenience [55], [43].

5. IDENTIFY TOOLS FOR QUALITY MEASUREMENTS FOR AOP

In the previous section, we have discussed various software quality models that support AOP. In Table 1, there is

a detail of the criteria of comparison and advantages and disadvantages of these software quality models the prerequisites.

Various ideas of programming quality qualities are evaluated and discussed in this paper. Too near investigation of different programming quality models utilized by different associations is being examined in this paper. A lot of effort is invested in the procedure quality improvement. At the point, when a task is embraced, the point is to convey the correct item at the ideal time with the exact functionalities. It is a typical situation that the one at the less than desirable end consistently wants/anticipates that the best should be conveyed to them. The onus lies on the engineers and testers to guarantee that they can meet the expectations for their customers. In this paper, we have examined different programming quality models given by various creators now and again and recognized that adaptability was inadequate in the current model. It is essential for the present framework to be able to oblige an expanding number of components to process developing volumes of work agilely, what is more, to be helpless for expansion. Henceforth, another sub-trademark versatility has been added under the viability of the AOSQ model. Each proposed model needs evaluation. We have been analysis most of the quality model for supporting measuring of AOP, some of them argue the standard metrics of OOP. We suggest proposing a framework for evaluating state metrics for AOP then moving on dynamic metric for a hybrid software application. In future, research will recommend a quality measurement framework and applied on static and dynamic parameters for AOP.

As it is described in Table 1, there is not software could give a total number of AO quality metrics. Many software used difference quality models to extract exact quality metrics. However, they have been succeeding in obtaining standard quality metrics but failed in hybrid application system or AO software.

The future work will be proposing a unique software product quality for hybrid software applications (OOP and AOP) to identify product quality metrics. The framework will help software engineering to measure AOP Metrics, which adapted ISO 9126 software quality model; this means that any hybrid system can measure with this new framework. The unique quality measurement framework of this research is the extension of quality model ISO 9126.

6. ACKNOWLEDGMENT

We would like to thank the University of Human Development in Kurdistan of Iraq and the University of Huddersfield for their usual support for our study.

REFERENCES

[1]. M. Ceccato and P. Tonella. "Measuring the effects of software aspectization". Vol. 12. In *1st Workshop on Aspect Reverse Engineering*, 2004.

[2]. W. Cazzola, A. Marchetto and F. B. Kessler. "AOP-hiddenmetrics: Separation, extensibility, and adaptability in SW measurement". *Journal of Object Technology*, vol. 7, no. 2, pp. 53-68, 2008.

[3]. E. Arisholm, L. C. Briand and A. Foyen. "Dynamic coupling measurement for object-oriented software". *IEEE Transactions on Software Engineering*, vol. 30, no. 8, pp. 491-506, 2004.

[4]. A. Mitchell and J. F. Power. "Using object-level run-time metrics to study coupling between objects". In *Proceedings of the 2005 ACM Symposium on Applied Computing*, pp. 1456-1462, 2005.

[5]. D. Ng, D. R. Kaeli, S. Kojarski and D. H. Lorenz. "Program comprehension using aspects". In *ICSE 2004 Workshop WoDiSEE'2004*, 2004.

[6]. L. Cheikhi, R. E. Al-Quraish, A. Idri and A. Sellami. "Chidamber and Kemerer object-oriented measures: Analysis of their design from the metrology perspective". *International Journal of Software Engineering and Its Applications*, vol. 8, no. 2, pp. 359-374, 2014.

[7]. A. Kaur, S. Singh, K. Kahlon and P. S. Sandhu. "Empirical analysis of CK and MOOD metric suit". *International Journal of Innovation, Management, and Technology*, vol. 1, no. 5, p. 447, 2010.

[8]. N. Fenton and J. Bieman. "*Software Metrics: A Rigorous and Practical Approach*'. CRC Press, Boca Raton, FL, 2014.

[9]. E. Rønningen and T. Steinmoen. "*Increasing Readability with Aspect-Oriented Programming*". Department of Computer and Information Science (IDI), 2003.

[10]. C. Zhang and H. A. Jacobsen. "Quantifying aspects in middleware platforms". In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, pp. 130-139, 2003.

[11]. M. Mickelsson. "*Aspect-Oriented Programming Compared to Object-Oriented Programming when Implementing a Distributed, Web Based Application*". Department of Information Technology, Uppsala University, 2002.

[12]. Y. Coady and G. Kiczales. "Back to the future: A retroactive study of aspect evolution in operating system code". In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, pp. 50-59, 2003.

[13]. S. L. Tsang, S. Clarke and E. Baniassad. "*Object Metrics for Aspect Systems: Limiting Empirical Inference Based on Modularity*". Submitted to ECOOP, 2004.

[14]. A. A. Zakaria and H. Hosny. "Metrics for aspect-oriented software design". Vol. 3. In *Proc. Third International Workshop on Aspect-Oriented Modeling*, AOSD, 2003.

[15]. R. Burrows, F. C. Ferrari, A. Garcia and F. Taiani. "An empirical evaluation of coupling metrics on aspect-oriented programs". In *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*, pp. 53-58, 2010.

[16]. R. Burrows, A. Garcia and F. Taiani. "Coupling metrics for aspect-oriented programming: A systematic review of maintainability studies". In *International Conference on Evaluation of Novel Approaches to Software Engineering*, pp. 277-290, 2008.

[17]. K. Dhambri, J. F. Gélinas, S. Hassaine and G. Langelier. "Visualization-based Analysis of Quality for Aspect-Oriented Systems". *ACM international Conference*, Long Beach, CA, 2005.

[18]. E. Figueiredo, A. Garcia and C. Lucena. "AJATO: An AspectJ Assessment Tool". In *European Conference on Object-Oriented Programming (ECOOP Demo),* France, 2006.

[19]. K. Sirbi and P. J. Kulkarni. "AOP and its impact on software quality". *Elixir Computer Science and Engineering*, vol. 54, pp. 12606-12610, 2013.

[20]. H. R. Bhatti. "*Automatic Measurement of Source Code Complexity"*. Tore Cane, Italy, 2011.

[21]. M. Wilhelm and S. Diehl. "Dependency viewer-a tool for visualizing package design quality metrics". In *Visualizing Software for Understanding and Analysis, 2005*. VISSOFT 2005. 3rd IEEE International Workshop on IEEE, pp. 1-2, 2005.

[22]. R. Lincke, J. Lundberg and W. Löwe. "July. Comparing software metrics tools". In *Proceedings of the 2008 International Symposium On Software Testing and Analysis*. ACM, pp. 131-142, 2008.

[23]. V. Yadav and R. Singh. "February. Predicting design quality of object-oriented software using UML diagrams". In *Advance Computing Conference (IACC)*, 2013 IEEE 3rd International IEEE., pp. 1462-1467, 2013.

[24]. Y. U. Mshelia and S. T. Apeh. "Can software metrics be unified"? In *International Conference on Computational Science and Its Applications*. Springer, Cham, pp. 329-339, 2019.

[25]. S. Yeresime, J. Pati and S. K. Rath. "Review of software quality metrics for object-oriented methodology". In *Proceedings of International Conference on Internet Computing and Information Communications*. Springer, India, pp. 267-278, 2014.

[26]. B. J. Eric and M. E. Bernstein. "*Software Engineering: Modern Approaches"*. Waveland Press, Long Grove, Illinois, USA, 2016.

[27]. M. C. Lee. "Software quality factors and software quality metrics to enhance software quality assurance". *British Journal of Applied Science and Technology*, vol. 4, no. 21, pp. 3069-3095, 2014.

[28]. M. I. Ghareb and G. Allen. April. "State of the art metrics for aspect oriented programming". In *AIP Conference Proceedings*. Vol. 1952. AIP Publishing, p. 020107, 2018.

[29]. M. Aniche. Static Metrics Measurments, Jul. 2019. Avaialble from: https://www.github.com/mauricioaniche/ck. [Accessed: 31-July-2019].

[30]. J. A. McCall, P. K. Richards and G. F. Walters. "*Factors in Software Quality, Griffiths Air"*. Force Base, N.Y. Rome Air Development Center Air Force Systems Command, 1977.

[31]. B. W. Boehm, J. R. Brown and M. L. Lipow. "Quantit ative Evaluation of Software Quality". *Proceedings of the 2nd International Conference on Software Engineering*, IEEE Computer Society Press, San Francisco, California, United States, pp. 592 605, 1976.

[32]. R. G. Dromey. "A model for software product qual ity". *IEEE Transactions on Software Engineering*, vol. 21, no.2, pp.146-162, 1995.

[33]. M. S. Deutsch and R. R. Wills. "*Software Quality Engineering; A Total Technical and Management Ap-proach*". Prentic-Hall, Inc., Upper Saddle River, NJ, 1998.

[34]. W. A. Word and B. Venkataraman. "Some Observations on Software Quality". In *Proc. of the 37th Annual Southeast Regional Conference*, Mobile, AL, 1999.

[35]. S. Arun, R. Kumar and P. S. Grover. "Estimation of quality for software components: An empirical approach". *ACM SIGSOFT Software Engineering Notes*, vol. 33, no. 6, pp. 1-10, 2008.

[36]. C. Chang, C. Wu and H. Lin. "Integrating fuzzy theory and hierarchy concepts to evaluate software quality". *Software Quality Control*, vol. 16, no. 2, pp. 263-276, 2008.

[37]. A. Kaur, P. S. Grover and A. Dixit. "*Performance Efficiency Assessment for Software Systems. In Software Engineering*". Springer, Singapore, pp. 83-92, 2019.

[38]. P. Nistala, K. V. Nori and R. Reddy. "Software quality models: A systematic mapping study". In *Proceedings of the International Conference on Software and System Processes*. IEEE Press, New York, pp. 125-134, 2019.

[39]. H. Noviyarto and Y. S. Sari. "Testing and implementation outpatient information system using ISO 9126". *International Educational Journal of Science and Engineering*, vol. 2, no. 3, p. 11, 2019.

[40]. Y. S. Sari. "Testing and implementation ISO 9126 for evaluation of prototype knowledge management system (KMS) e-procurement". *International Educational Journal of Science and Engineering*, vol. 2, no. 3, p. 1, 2019.

[41]. ISO/IEC 9126 1, 2001, ISO/IEC 9126 2, 2003, ISO/IEC 9126 3, 2003 and ISO/IEC 9126 4. "*Information Technology Product Quality Part1: Quality Model, Part 2: External Metrics, Part3:Internal Metrics, Part4: Quality in use Metrics*". International Standard ISO/IEC 9126, International Standard Organization, 2004.

[42]. M. Yan, X. Xia, X. Zhang, L. Xu, D. Yang and S. Li. "Software quality assessment model: A systematic mapping study". *Science China Information Sciences*, vol. 62, no. 9, p.191101, 2019.

[43]. H. Kuwajima and F. Ishikawa. "*Adapting SQuaRE for Quality Assessment of Artificial Intelligence Systems*". Machine Learning, arXiv preprint arXiv:1908.02134, 2019.

[44]. G. O'Regan. "Fundamentals of Software Quality. In *Concise Guide to Software Testing.* Springer, Cham, pp. 1-31, 2019.

[45]. R. Adnan and B. Matalkah. "A new soft-ware quality model for evaluating COTS components". *Journal of Computer Science*, vol. 2, no. 4, pp. 373-381, 2006.

[46]. M. Bertoa and A. Vallecillo. "Quality Attributes for COTS Components". In *the Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)*, Spain, 2002.

[47]. A. Kumar, P.S. Grover and R. Kumar. "A quantitative evaluation of aspect-oriented software quality model (AOSQUAMO)". *ACM SIGSOFT Software Engineering Notes*, vol. 34, no. 5, pp. 1-9, 2009.

[48]. H. Rashidi and M. S. Hemayati. "Software quality models: A comprehensive review and analysis". *Journal of Electrical and Computer Engineering Innovations*, vol. 6, no. 1, pp. 59-76, 2019.

[49]. N.R. Mead and T. Stehney. "*Security Quality Requirements Engineering (SQUARE) Methodology*". Vol. 30. Technical Report, ACM, pp. 1-7, 2005.

[50]. J. Rumbaugh, G. Booch and I. Jacobson. *El Lenguaje Unificado de Modelado: Manual de Referencia*". Addison Wesley, Madrid, 2000.

[51]. S. Clarke and R. J. Walker. Generic Aspect-Oriented Design with Theme/UML. "*Aspect-Oriented Software Development*". In: R. E. Filman, T. Elrad, S. Clarke, and M. Aksit, editors. Addison-Wesley, Boston, 2005.

[52]. M. Ghareb and G. Allen. "*Improving the Design and Implementation of Software Systems uses Aspect-Oriented Programming*".

University of Human Development Suilamanay, Iraq, 2015.

[53]. G. Allen and M. Ghareb. Identifying similar pattern of potential aspect-oriented functionalities in software development life cycle". *Journal of Theoretical and Applied Information Technology*, vol. 80, no. 3, pp. 491-499, 2015.

[54]. I. S. Brito and A. M. Moreira. "*Advanced Separation of Concerns for Requirements Engineering*". In JISBD, Spain, pp. 47-56, 2003.

[55]. M. W. Suman and M. D. U. Rohtak. "A comparative study of software quality models". *International Journal of Computer Science and Information Technologies*, vol. 5, no. 4, pp. 5634-5638, 2014.