Advanced Computing: An International Journal (ACIJ), Vol.3, No.2, March 2012

ASPECT-ORIENTED SOFTWARE QUALITY MODEL: THE AOSQ MODEL

Pankaj Kumar
Noida Institute of Engineering & Technology, Greater Noida
unpankaj@gmail.com

ABSTRACT

Nowadays, software development has become more complex and dynamic; they are expected more flexible, scalable and reusable. Under the umbrella of aspect, Aspect-Oriented Software Development (AOSD) is relatively a modern programming paradigm to improve modularity in software development. Using Aspect-Oriented Programming (AOP) language to implements crosscutting concerns through the introduction of a new construct Aspect like Class is defined as a modular unit of crosscutting behavior that affect multiple classes into reusable modules.

Several quality models to measure the quality of software are available in literature. However, keep on developing software, and acceptance of new environment (i.e. AOP) under conditions that give rise to an issue of evolvability. After the evolution of system, we have to find out how the new system needs to be extensible? What is the configurable status? Is designed pattern stable for new environment and technology? How the new system is sustainable?

The objective of this paper is to propose a new quality model for AOSD to integrating some new quality attributes in AOSQUAMO Model based which is based on ISO/IEC 9126 Quality Model, is called Aspect-Oriented Quality (AOSQ) Model. Analytic Hierarchy Process (AHP) is used to evaluate an improved hierarchical quality model for AOSD.

KEYWORDS

AOP, AOSD, Aspect, Crosscutting Concern, Model Development, Quality Attributes, Software Quality Engineering, Software Quality Models

1. INTRODUCTION

Software engineering is related to the development and evolution of large, complex and critical software-intensive system. These systems are expected to be more flexible, scalable and reusable. In order to achieve these objectives, development techniques that support abstraction and modularization in software development system can be useful. Software Modularity surpassing traditional abstraction is necessary for developing complex modern systems - specifically software and software-intensive systems.

Aspect-Oriented Software Development (AOSD) and other new types of modularity and abstraction approaches are attracting lot of attention over many domains within, and beyond computer science [1, 2, 18]. AOSD is comparatively a modern Programming Paradigm aimed at improving modularity under the umbrella of Aspect. Implementations using an Aspect-Oriented Programming (AOP) language attempts to encapsulate crosscutting concerns. Crosscutting concerns are introduced through a new construct class like Aspect which is, defined as a modular unit of crosscutting implementation. It encapsulates behavior affecting multiple classes into reusable modules.

Any new addition to the existing code may further worsen the situation, if the integration is not carried out carefully. Since the target application will have its behavior changed, it can cause an

impact on software quality parameters like reliability, functionality, performance and efficiency. The application of AOP paradigm can simplify the up gradation, maintenance and evolvability of the software. However, the incorrect usage of AOP paradigm may not lead to the desired quality level of the software. Further, existence large number of process paradigms and existence varied product standards, the assessment of quality becomes pertinent.

The demand for quality has been part of human nature for a long time, but the quantification of quality and establishment of formal quality standards are a 20th century phenomena [1, 3]. Practitioners, Researchers and Developers have proposed several metrics and quality models [21]. In general, the expert's definitions of quality fall into two categories: Level one quality applies to products or services whose measurable characteristics satisfy a fixed set of specifications that are usually numerically defined. Level two quality products and services need only satisfy customer expectations [3]. In this paper, level one category is followed.

The remainder of this paper is organized as follows: Section 2 discusses the chronological development of software quality models. Section 3 discusses the background of software quality models. Section 4 proposes a new software quality model for AOP i.e. called Aspect-Oriented Software Quality (AOSQ) Model and Section 5 presents conclusion and future work directions.

2. CHRONOLOGICAL DEVELOPMENTS OF QUALITY MODELS

Over the last five decades, there are number of software quality models in software engineering literature. The quality Models are divided into two categories: Hierarchical quality Model and Non-Hierarchical quality Model. In this paper, only hierarchy quality models are described. Each one of these quality models consist of a set of high quality characteristics/factors and sub-characteristics/sub-factors.

In late 70's, two principal models were proposed one after another. In 1977, McCall et al. [7] proposed a quality model called McCall's Software Quality Model and it is also called Classical Quality Model. McCall's Quality Model was later adapted and revised as the MQ Model by Watts in 1987. Next year in 1978, Boehm et al. [8] proposed another quality model using McCall's quality model, called Boehm's Software Quality Model.

Later on in late 80's, three quality Models (in 1987, Evans & Marciniak's Quality Model and FURPS Quality Model and next year 1988, Deutsch & Will's Quality Model) were proposed. Among these quality models, FURPS Quality Model [9, 10] is more popular because it is first industrial approach based quality model, proposed by Hewlett-Packard (HP). Later on, the model was extended by IBM Rational Software into FURPS+, widely used in the software industry now.

Till 90's, number of software quality models were proposed. This led to lot confusion among practitioners, which model to actually follow. Therefore, International Organization for Standardization/International Electro-technical Commission (ISO/IEC) began to develop and standardize a new quality model considering the entire repository of various quality models proposed so far. . In 1991, ISO/IEC proposed a quality model, called ISO/IEC Quality Model. Later on, the name was changed to ISO/IEC 9126 Quality Model [11, 12, 13] since ISO 9126 was part of the ISO 9000 standard. Later on in 1995, R.G. Dromey [14] proposed a quality model adding one characteristic into ISO/IEC 9126 Quality model. The model is called Dromey's Software Quality Model.

All the above defined software quality models were derived based on either legacy software or object-oriented software. The upraise of new technologies like Aspect oriented programming (AspectJ), the software development architecture focus on maintaining the overall quality of

software systems through their lifecycle. AOP Application consists of Class and Aspect. The quality assessment of Implementation of Class modules is measured by above defined quality models. The quality of aspect modules cannot be by the above discussed models and software quality model for assessing the quality of projects developed using AOP need to be developed. In 2009, Software quality model for AOSD was proposed by Kumar et al. [15] and it is called Aspect-Oriented Software Quality Model (AOSQUAMO). Another AOSD based quality model is proposed by I. Castillo et al. [25] in 2010. It is a common framework, based on UML conceptual model the REASQ (REquiremets, Aspects and Software Quality) model. The model exists as a specification definition as an integrated ontology implemented with the Protégé Tool, for the reasoning, understanding, handling and reuse of the main notation related to AOSD Paradigms.

So, integrating some new characteristics/factors and sub-characteristics/sub-factors of AOSD in AOSQUAMO Model as a base ISO/IEC 9126 Quality Model and proposed a new quality model for Aspect-Oriented Programming Paradigm, is called Aspect-Oriented Software Quality (AOSQ) Model.

3. SOFTWARE QUALITY MODELS BACKGROUND

Several software quality models were proposed, in order to evaluate different types of software products. This section presents the most popular quality models.

3.1. McCall's Quality Model

One of the most oldest and renown predecessors of today's software quality model developed by McCall et al. [7] also known as the General Electric (GE) Model originates from US Air Force, the Rome Air Development Center (RADC), to improve the quality of software products. Main purpose of this model is to estimate the relationship between external factors and product quality criteria.

The structure of McCall's Quality Model [7] is shown in figure 1.

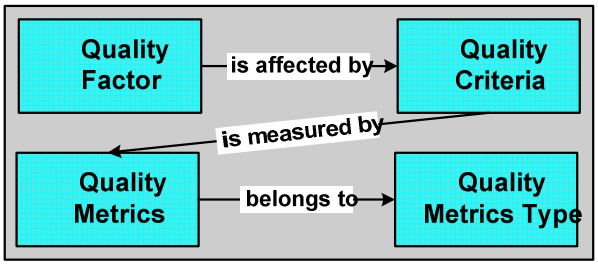

Figure 1: Structure of McCall's Quality Model

The McCall's Quality Model is divided into highest three major perspectives: Product Operation, Product Revision and Product Transaction. All the three major perspectives are divided into 11 external factors which describe the external view of software system (i.e. User View) and all the external factors are divided into 23 quality's criteria which describe the internal view of software system (i.e. Developer View). Quality's criteria associated with a set of quality metrics are defined and used to provide a scale and method for measurement [6]. The factors and criteria are shown in table 1. The main contribution of this quality model is the relationship between quality factors and metrics. However, the quality model does not take into account the quality aspect of various functionalities of the software product.

Table 1: McCall's Quality Model

Quality Type	Product Perspective	Factors	Criteria
Quality Software Product	Product Operation	Correctness	Traceability
			Completeness
			Consistency
		Reliability	Accuracy
			Error-Tolerance
			Consistency
		Efficiency	Execution Efficiency
			Storage Efficiency
		Integrity	Access Control
			Access Audit
		Usability	Operability
			Training
			Communicativeness
	Product Revision	Maintainability	Simplicity
			Conciseness
			Self-Descriptiveness
			Modularity
		Testability	Instrumentation
			Self-Descriptiveness
			Simplicity
			Modularity
		Flexibility	Simplicity
			Expandability
			Generality
			Modularity
	Product Transaction	Portability	Simplicity
			Software System Independence
			Machine Independence
		Reusability	Simplicity
			Generality
			Modularity
			Software System Independence
			Machine Independence
		Interoperability	Communications Commonality
			Data Commonality

3.2. Boehm's Quality Model

The second renowned predecessors of today's software quality model was developed by Boehm et al. (1978), adding emphasis on the maintainability for software product into McCall's Quality Model is called Boehm's Quality Model [8] and is shown in table 2.

The importance of this model is to describe the current coexisting deficiency of McCall's Quality Model that automatically and quantitatively evaluate the quality of software product. Hence, characteristics of Boehm's quality model are represented in hierarchical form to manage total quality. The validity of model is mostly assumed for common sense reasons, rather than on empirical evidence of their accuracy as a model.

Table 2: Boehm's Quality Model

Quality Type	Product Perspective	Factors	Criteria
General Utility		Portability	Device-Independent
			Completeness
	As is utility	Reliability	Accuracy
			Completeness
			Consistency
		Efficiency	Device Efficiency
			Accessibility
		Human Engineering	Communicativeness
			Accessibility
	Maintainability	Testability	Communicativeness
			Accessibility
			Structuredness
			Self-Descriptiveness
		understandability	Consistency
			Structuredness
			Self-Descriptiveness
			Conciseness
			Legibility
		Modifiability	Structuredness
			Augment-ability

3.3. FURPS Quality Model

All the models proposed so far mentioned in section 3.1 and 3.2 were developed by academicians as a research activity only. So far industry had not show any interest in the quality issues of the software development processes. Robert Grady and Hewlett-Packard are the first one to propose model with the industrial approach. This quality model is known as FURPS Quality Model [9, 10]. The model aimed at improving the management of software development processes by software industry.

FURPS Quality Model includes top five level attributes (Functionality, Usability, Reliability, Performance and Supportability) as shown in table 3. Further, the model was extended by IBM Rational Software into FURPS+, widely used in the software industry now.

3.4. ISO/IEC 9126 Quality Model

Since, the number of software quality models were proposed, the confusion occurred and new standard quality model was essential.

Thus, ISO/IEC Joint Technical Committee (JTC) [11, 12, 13] started to develop model using the required consensus and inspire standardization worldwide. In 1991, ISO/IEC JTC – 1 proposed a quality model called ISO/IEC Quality Model. Further, name was changed to ISO/IEC 9126 Quality Model. The model is an extension of previous work did by McCall (1977), Boehm (1978) and FURPS (1987) etc.

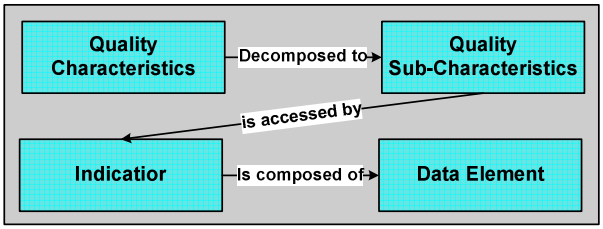

Figure 2: Structure of ISO/IEC 9126 Quality Model

The structure of ISO/IEC 9126 quality model is shown in figure 2 and the characteristics and sub-characteristics are shown in table 4.

ISO/IEC 9126 quality model is divided into two perspectives (i.e. first is External & Internal Quality and second is Quality in Use) for evaluating the quality of software products. The defined characteristics in external & internal quality perspective are applicable to each and every type of software products.

Though, ISO/IEC 9126 quality model reasonably covers most of the quality characteristics, and sub characteristics, the product perspective are taken as external and internal quality. The model did not take into account the reusability feature.

Table 3: FURPS Quality Model

Quality Type	**Product Perspective**	**Characteristics**	**Sub- Characteristics**
Quality Software Product		Functionality	Feature Set
			Capabilities
			Generality
			Security
		Usability	Human Factors
			Aesthetics
			Consistency
			Documentation
		Reliability	Frequency/Severity of Failure
			Recoverability
			Predictability
			Accuracy
			Mean Time to Failure
		Performance	Speed
			Efficiency
			Resource Consumption
			Throughput
			Response Time
			Testability
			Extensibility
			Adaptability

		Supportability	Maintainability
			Compatibility
			Configurability
			Serviceability
			Install-ability
			Localizability
			Portability

3.5. Dromey's Quality Model

Dromey's quality model [14] states that every software product has its own process evaluation. So, there are some dynamic ideas required for process modeling. Hence, Dromey proposed a software quality model in 1995 called Dromey's Quality Model to integrate Reusability and Process Maturity as characteristics in ISO/IEC 9126 Quality Model. The main objective of this quality model is to obtain a model in broad area for variety of application. The characteristics and sub-characteristics are shown in table 5.

Dromey's quality model is associated with reliability and maintainability. So, it is typical to judge, that model is feasible before the software system is operational in development area or not.

Table 4: ISO/IEC 9126 Quality Model

Quality Type	Product Perspective	Characteristics	Sub- Characteristics
Quality Software Product		Functionality	Suitability
			Accuracy
			Interoperability
			Security
		Reliability	Maturity
			Fault Tolerance
			Recoverability
		Usability	Understandability
			Learn-ability
			Operability
			Attractiveness
		Efficiency	Time Behavior
			Resource Utilization
		Maintainability	Analyzability
			Changeability
			Stability
			Testability
		Portability	Adaptability
			Replace-ability
			Install-ability
			Co-Existence

Table 5: Dromey's Quality Model

Quality Type	Product Perspective	Characteristics	Sub- Characteristics
Quality Software Product	Implementation	Correctness	Functionality
			Reliability
		Internal	Maintainability
			Efficiency
			Reliability
		Contextual	Maintainability
			Reusability
			Portability
			Reliability
		Descriptive	Maintainability
			Efficiency
			Reliability
			Usability

3.6. AOSQUAMO Model

All the above defined quality models belongs to either legacy software or Object-Oriented software but not to AOP.

Kumar et al. [15] proposed first AOP based software quality model called Aspect-Oriented Software Quality (AOSQUAMO) Model in 2009 which is an extension of ISO/IEC 9126 quality model. Four sub-characteristics (i.e. Reusability, Complexity, Code-Reducibility and Modularity) are integrated under different characteristics of ISO/IEC 9126 quality model which is shown in table 6.

But this model also lacks some characteristics/factors and sub-characteristics/sub-factors which is important for Aspect-Oriented Programming based applications.

Table 6: AOSQUAMO Model

Quality Type	Product Perspective	Characteristics	Sub- Characteristics
Quality Software Product		Functionality	Suitability
			Accuracy
			Interoperability
			Compliance
			Security
			Reusability
		Reliability	Maturity
			Fault Tolerance
			Recoverability
		Usability	Understandability
			Learn-ability
			Operability
			complexity
		Efficiency	Time behavior
			Resource behavior
			Code-reducibility
		Maintainability	Analyzability
			Changeability
			Stability
			Testability
			Modularity

		Portability	Adaptability
			Replace-ability
			Conformance

4. PROPOSAL OF SOFTWARE QUALITY MODEL: AOSQ MODEL

Over the last 50 years, the increasing trend to evolve complex software system has emphasized the need to consider software quality as an integral part of software system development. There are so many programming paradigms coming over this period. Every programming paradigm has its own characteristics and sub-characteristics. On the way of evolution, AOP programming paradigms was proposed by Kiczales et al. (1997) [22] and its main objective is to improve software quality by providing better modularization and separation of concern (SoC).

Most of all the software quality models which are proposed after ISO/IEC 9126 Quality Model (1991) [11, 12, 13], are derived from ISO/IEC 9126 Quality Model. Example:

Reusability is integrated as a characteristic by R. G. Dromey (1995) [14] to obtain a model in broad area for variety of application.

Bansiya et al. (2002) [35] proposed a quality model for Object-Oriented Design (QMOOD) which is extension of Dromey's quality model. The QMOOD provides a way to define Object-Oriented Design properties with their associated metrics.

Bertoa et al. (2002) [23] proposed a quality model for Component-based Software Development (CBSD) which is called Quality Model for COTS (Commercial off the Shelf) Components. In this model sub-characteristics are divided into two categories: Runtime and Life Cycle based on their nature and two new sub-characteristics also integrated: Compatibility and Complexity in the category of life cycle which indicated whether previous version of component is compatible with its new version.

Rawashdeh et al. (2006) [24] divided characteristics into different types of stakeholder and also removed two sub-characteristics Stability and Analyzability from Maintainability. After that, integrating two sub-characteristics Compatibility into Functionality and Complexity into Usability proposed a quality Model.

Kumar et al. (2009) [15] proposed a first quality model for AOSD to integrate Reusability into Functionality, Complexity into Usability, Code-reducibility into efficiency and Modularity into Maintainability.

Castillo et al. (2010) [25] proposed a conceptual quality model to clarify the AOSD emergent terminologies: Aspect, Composition Concern (Functional, Non-functional and Crosscutting), Quality (Functional and Non-functional) or Property (Inherited and Assigned) requirements for the software product. This conceptual model is called REASQ Model which is integration of ISO/IEC 9126 and ISO/IEC 25030 and expressed in UML.

On the way of defining a new software quality model for AOSD, we should have to try to integrate almost all the characteristics of AOSD. Most of the ideas of AOSD are borrowed from Object-Oriented Programming (OOP) and introduce new Abstraction technique. So that AOP has all the characteristics of OOPs. However, problems begin from introduction of new Abstraction features.

Due to rapid change of development system, real world software system is evolved continually to meet challenges between the user requirement and operational environment. The nature of change action can be corrective, adaptive and perfective. Development of software concern is

well modularized to achieve desirable characteristics like Extensibility, Sustainability, Design Stability and Configurability [21] which is missing form quality model. Among all the desirable characteristics Design Stability is most important. Integrating Extensibility, Sustainability, Design Stability and Configurability as Sub-characteristics under Evolvability characteristic into AOSQUAMO Model, proposed a new quality model is called Aspect-Oriented Software Quality (AOSQ) Model which is derived from ISO/IEC 9126 quality Model. All the characteristics and sub-characteristics of AOSQ Model are shown in table 7 and newly integrated characteristics are highlighted.

5. CHARACTERISTICS IDENTIFICATION AND DEFINITION FOR PROPOSED QUALITY MODEL

As we know, it is based on the ISO/IEC 9126 quality model and all the characteristics belong to AOSQUAMO Model. Four new sub-characteristics are integrated: Extensibility, Sustainability, Design Stability and Configurability under Evolvability Characteristics in AOSQUAMO Model. Rest of the characteristic's definition is similar to AOSQUAMO Model. Definition of new characteristic is as follows:

5.1. Evolvability

Any real-world software needs evolution after a certain period of time to fulfill the current trends, technology, changes in user requirement and operational environment. In the process of developing software, every well modularized concern needs stability, maintainability, changeability, and extensibility. Programming in aspect-oriented languages has been suggested a way to realize these characteristics [21].

Due to the changing nature of real-world software, we proposed Evolvability as characteristic of AOSQ Model. After the evolution of system, we must find out the how much new evolved system is extensible?, what are the configurable status?, is designed pattern is stable for new environment and technology? And how much new evolved system is sustainable?

On the basis of above questions, we short out some new sub-characteristics such as Extensibility, Sustainability, Design Stability and Configurability under Evolvability. Definitions of new sub-characteristics are as follows:

Table 7: Proposed AOSQ Model

Quality Type	Product Perspective	Characteristics	Sub-Characteristics
Quality Software Product		Functionality	Suitability
			Accuracy
			Interoperability
			Security
			Reusability
		Reliability	Maturity
			Fault Tolerance
			Recoverability
		Usability	Understandability
			Learn-ability
			Operability
			Attractiveness
			complexity
		Efficiency	Time behavior
			Resource behavior

			Code-reducibility
		Maintainability	Analyzability
			Changeability
			Stability
			Testability
			Modularity
		Portability	Adaptability
			Replace-ability
			Install-ability
			Co-Existence
		Evolvability	**Extensibility**
			Sustainability
			Design Stability
			Configurability

5.1.1. Extensibility

AO Software development continues to grow beyond the scope. AOSD is likely to have positive effect on performance, modularity and evolution. So, it becomes important to reuse components. We focus on a specific kind of reusing of component called extensibility, i.e. the extension of software without accessing existing code to edit or copy it. Extensibility is a systemic measure of the ability to extend a software design principle where the implementation takes into consideration future growth.

Specifically, our definition prevents two acts: The first is source modification, which can introduce unexpected behavior and structural changes. The second is copying of code, which increase the clerical effort needed to maintain program by introducing potential inconsistencies [26, 27]. Extensibility is particularly critical for a developer who wishes to delivered software that clients can customize, but who does not want to reveal proprietary source code.

Cody et al. [27] conducted a case study on FreeBSD Operating System. They used the evolution of the FreeBSD operating system of three different versions. They focused on the evolution of specific crosscutting concerns in isolation. They found that in the AO implementation of each concern, changes to the concern itself were better localized due to textual locality, configuration changes mapped directly to modification to pointcuts and/or make file options, and aspectization solutions provided extensibility due to improved modularization.

So, we propose Extensibility as sub-characteristics in Evolvability Characteristics.

5.1.2. Sustainability

Legacy software system faces problems in new software system structure. The start of new software system structure such as OOP, AOP etc., marked by improved separation of concern, is often preceded by the darkness in which the old software system structure must be degraded. Though aspects have been shown to be effective as a center point for evolving crosscutting concerns, the fact that they rely on explicit external interaction infers the aspects could have negative affect under these extreme conditions - when the code that is crosscut, or the dominant decomposition - is undergoing structure re-composition [29]. Low level system infrastructure need to be fast and flexible. While unpleasant to many developers due to their lack of semantic leverage in traditional language construct of C, C++ and Java, this is a reality in today's software system infrastructure.

Gibbs et al. [29] conducted an experimental on Sustainability of aspects in software system using the rapidly evolving Memory Management Tool kit (MMTK) with the Jikes Research Virtual Machine (RVM). The RVM is an open source project in Java. Sustainability is the long-

term maintenance of software system, which has environmental and economic for management of all types of resource.

In that manner one more sub-characteristics Sustainability is proposed under Evolvability Characteristics.

5.1.3. Design Stability

Design Stability covers the sustenance of system modularity characteristics and the absence if ripple-effects in the presence of change. Development of stable design has increasingly been a deep challenge to software engineers due to the high volatility of systemic concern and their dependencies. Some recent industrial case studies have demonstrated that around 50% of object-oriented code is altered between two releases, and 68% of change requests are accepted and implemented [30]. It has been empirically observed that design stability is directly dependent on the decomposition mechanisms.

The definition of AOP indicate that better modularity and changeability of crosscutting concerns are obtained through the use of new composition mechanisms, such as pointcut-advice and inter-type declarations. AOP decompositions promote better design stability in realistic software development process, especially when experiencing changes of a diverse nature [30].

In that manner, one more sub-characteristics Design Stability is proposed under Evolvability Characteristics.

5.1.4. Configurability

A Middleware platform, like CORBA, DCOM, J2EE and .NET offers abstraction and simplicity for the complex and heterogeneous computing environment with high quality of distributed applications with a shortest development cycle and a much smallest coding effort. Many middleware features do not exist in modular forms and crosscut implementations of other functionalities. So, AOP based newer middleware technologies, such as J2SE, J2EE, and J2ME, appear to have taken the same direction. But a serious limitation of these solutions is increased complexity of development and maintenance, is that they only provide a fixed set of options for users [32].

The effective solution of this problem is to achieve using a high degree of configurability in the middleware architecture and to customize middleware according to a specific user need, a concrete usage scenario, and a particular deployment or runtime instance [32].

So, we proposed Configurability as sub-characteristics in Evolvability.

6. CONCLUSION AND FUTURE WORK

Several surveys have been conducted by researchers to investigate the effect of AOP on non-AOP characteristics for software development since 1997 when the AOP was born. On the way of investigation, the effects of AOP on code size (i.e. size, redundancy), cognition (i.e. understandability, development efficiency), language mechanism (i.e. exception handling) performance, modularity (i.e. design quality, pattern composition) and evolvability (i.e. changeability, maintainability, extensibility, configurability, design stability) related characteristics are used. A few of product related characteristics are examined. Some product related characteristics can help to understand the true potential of AOSD i.e. Evolvability. Among above defined characteristics, some of the characteristics are still left in software quality model.

In this paper, Integrating Evolvability as characteristic under Extensibility, Sustainability, Design Stability and Configurability as Sub-characteristics into AOSQUAMO Model, a new quality model called Aspect-Oriented Software Quality (AOSQ) Model; derived from ISO/IEC 9126 quality Model is proposed. This paper also contains chronological development of software quality models with description.

Every proposed model required evaluation. To evaluate the proposed quality model for AOP, Analytics Hierarchy Process (AHP) approach could be used which addresses uncertainty and imprecision in evaluation during pre-negotiation stages, where comparative judgments of characteristics based on decision maker with the help of fuzzy logic.

REFERENCES

[1]. R. Laddad, "Aspect-Oriented Programming Will Improve Quality," *IEEE Computer Society,* 2003.

[2]. AOSD Web Site: http://www.aosd.net/2012/.

[3]. R. W. Hoyer and B. B. Y. Hoyer, "What is Quality?," *American Society for Quality*, Page No.: 53 - 62, July 2001.

[4]. K. Khosravi and Y. Gueheneuc, "On Issues with Software Quality Models," Page No.: 70 - 83, 2004.

[5]. R. E. Ai-Qutaish. "Quality Models in Software Engineering Literature: An Analytical and Comparative Study," *Journal of American Science*, Volume 6 Number 3, 2010.

[6]. B. Al-Badareen, M. H. Selamat, M. A. Jabar, H. Din and S. Turarv, "Software Quality Model: A Comparative Study," *Springer ICSECS'11*, Page No.: 46 - 55, 2011.

[7]. J. A. McCall, P. K. Richards and G. F. Walters, "Factors In Software Quality - Concept and Definitions of Software Quality," *Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York*, Volume 1, Number 3, November, 1977.

[8]. B. W. Boehm, J. R. Brown and M. Lipow, "Quantitative Evaluation of Software Quality," *IEEE Computer Society Press*, Page No.: 592 - 605, 1978.

[9]. R. Grady, D. L. Caswell, "Software Metrics: Establishing a Company-wide Program," *Prentice Hall*, 1987.

[10]. R. Grady, "Practical software metrics for project management and process improvement," *Prentice Hall*, 1992.

[11]. ISO/IEC 9126-1: Software Engineering - Product Quality- Part 1: Quality Model, *International Organization for Standardization*, Switzerland, 2001.

[12]. ISO/IEC 9126-2: Software Engineering - Product Quality- Part 2: External Metrics, *International Organization for Standardization*, Switzerland, 2002.

[13]. ISO/IEC 9126-3: Software Engineering - Product Quality- Part 3: Internal Metrics, *International Organization for Standardization*, Switzerland, 2003.

[14]. R. G. Dromey, "A Model for Software Product Quality," *IEEE Transactions on Software Engineering*, Volume 21 Number 2, Page No.: 146 - 162, February 1995.

[15]. A. Kumar, P. S. Grover and R. Kumar, "A Quantitative Evaluation of Aspect-Oriented Software Quality Model," *ACM SIGSOFT Software Engineering Notes* Volume 34, Number 5, Page No.: 1 - 9, September 2009.

[16]. Web Site: http://en.wikipedia.org/wiki/FURPS

[17]. D. Galin, "Software Quality Assurance-From theory to implementation," *Addison Wesley-Pearson Education Limited*, 2004.

[18]. P. Kumar, "Aspect-Oriented Programming - A New Programming Paradigm," *IEEE International Advance Computing Conference* (IACC'09), Page No.: 1018 – 1022, March 2009.

[19]. Rawashdeh and B. Matalkah, "A New Software Quality Model for Evaluating COTS Components," *Journal of Computer Science*, Volume 2 Number 4, Page No.: 373 - 382, 2006.

[20]. J. Tian, "Software Quality Engineering-Testing, Quality Assurance and Quantifiable Improvement," *IEEE Computer Society*, 2005.

[21]. M. S. Ali, M. A. Babar, L. Chen and K. Stol, "A Systematic Review of Comparative Evidence of Aspect-Oriented Programming," *Elsevier Journal of Information and Software Technology*, Volume 52, Page No.: 871 - 887, May 2010.

[22]. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtie and J. Irwin, "Aspect-Oriented Programming," *ECOOP'97*, Page No.: 220 - 242, June 1997.

[23]. M. F. Bertoa and A. Vallecillo, "Quality Attributes for COTS Components," *6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'02)*, June 2002.

[24]. Rawashdeh and B. Matalkah, "A New Software Quality Model for Evaluating COTS Components," *Journal of Computer Science*, Volume 2 Number 4, Page No.: 373 - 381, 2006.

[25]. Castillo, F. Losavio, A. Matteo and J. Boegh, "REquirements, Aspects and Software Quality: the REASQ model," *Journal of Object Technology*, Volume 9, Number 4, Page No.: 69 - 91, 2010.

[26]. S. Krishnamurthi and M. Felleisen, "Toward a formal Theory of Extensible Software," *ACM SIGSOFT Conference*, Page No.: 88 - 98, 1998.

[27]. Y. Coady and G. Kiczales, "Back to the Future: A Retroactive Study of Aspect Evolution in Operating System Code," *ACM Aspect-Oriented Software Development'03*, Page No.: 50 - 59, 2003.

[28]. R. G. Dromey, "Software Product Quality: Theory, Model, and Practice," *Software Quality Institute, Griffith University, Australia,* Page No.: 1 - 31, March 1998.

[29]. C. Gibbs, C. R. Liu and Y. Coady, "Sustainable System Infrastructure and Big Bang Evolution: Can Aspects Keep Pace?," *ECOOP'2005*, Page No.: 241 – 261, 2005.

[30]. P. Greenwood, T. Bartolomei, E. Figueiredo, M. Dosea, A. Garcia, N. Cacho, C. Santanna, S. Soares, P. Borba, U. Kulesza, and A. Rashid, "On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study," *ECOOP'2007*, Page No.: 176 – 200, 2007.

[31]. E. Figueiredo, N. Cacho, C. Santanna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Filho, F. Dantas, "Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability," *ACM ICSE'2008*, Page No.: 261 – 270, 2008.

[32]. C. Zhang and H. Jacobsen, "Resolving Feature Convolution in Middleware Systems," *ACM OOPSLA'04*, Page No.: 188 – 205, 2004.

[33]. C. Chang, C. Wu and H. Lin, "Integrating Fuzzy theory and Hierarchy Concepts to Evaluate Software Quality," *Springer Software Quality Journal*, Volume 16, Page No.: 263 – 276, 2008.

[34]. Technical White Paper on Producing High-Quality Software with Aspect-Oriented Programming, *SHARTCRAFTERS*. July 2011.

[35]. A. Bansiya and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Transaction on Software Engineering*, Volume 28 Number 1, January 2002.