ISSN 2347 - 3983
Volume 8. No. 6, June 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter96862020.pdf
https://doi.org/10.30534/ijeter/2020/96862020

Analysis and Assessment of Existing Software Quality Models to Predict the Reliability of Component-Based Software

Shivani Yadav[1], Bal Kishan[2]
[1,2]Department of Computer Science & Applications
Maharshi Dayanand University, Rohtak-124001, Haryana, INDIA
[1]shivaniyadav17@gmail.com, [2]balkishan248@gmail.com

ABSTRACT

Software engineering is an approach pre-owned by the researchers and innovators to reduce the ratio of crisis in software. Therefore the designer/ innovator can readily design a valuable quality software, by using various approaches like Component-Based Software Engineering (CBSE). The components quality has a high impact on the quality of a whole application. Several quality models for CBSE and Component-Off-The-Shelf (COTS) are available in the literature so this paper presents various quality models with defined parameters for quality prediction. Several models, like, Boehm's, McCall's, FURPS, ISO 9126, Dromey's, have been developed for quality evaluation using hierarchically related characteristics of quality indicators. Over the years, some models have been developed for structuring software quality for software. Nowadays, researchers are showing their passion for innovation in the area of software reliability. The reliability of a software component relies on the following factors like the reliability of services, environment frequency so finally this paper presents the analysis and assessment of software quality models and their quality parameters for CBS by going through various researchers and practitioners' work.

Key words: Reliability, quality models, component-based software engineering, assessment

1. INTRODUCTION

Software engineering is a process to provide a methodical, disciplined, measurable approach for the creation, deployment, and maintenance of software; in other words, the application of engineering to software [1]. Software engineering is the part of *the* engineering approach which aims to fulfill the organizational commitment for its product quality. Six Sigma, Total Quality Management (TQM), etc. maintain a culture of continuous improvement of process, and eventually, this culture leads to the development of more effective methods for software engineering. The earliest work in the field of software began with process abstraction, which was found to be not powerful enough for building big, complex programs. This led to the focus on the data processing view which emphasized function abstraction. It takes in the inputs and outputs a value after processing the input in the function body. The 1980s saw the introduction of the Object-Oriented (OO) approach with capability for both data and function abstraction. This is achieved through “class”, which encapsulates data and restricts access through a collection of objects. This abstraction paves the way for the construction of huge, complex systems, with capabilities to extend the features of an object through a hierarchy of objects. All this led to the emergence of Component-Oriented (CO) system development methodology, which consists of software components as building blocks. CO system maximizes the reusability and reliability of components. A new software product development exercise often requires new measures to ensure the quality of development processes, apart from quality requirements and quality measurement methods. The focus is on a robust quality management system that allows for the development of a quality product, rather than individual quality processes. This quality management system may comprise detailed methods and accepted standards for development processes. We can classify the existing international standards into 4 broad categories:

- Standards for development process documentation
- Standards for quality assurance process implementation
- Standards for software documentation
- Other related standards

Measures of quality assurance are determined by standards of software quality lifecycle and a quality model with defined parameters for quality evaluation. Several models, like, Boehm's, McCall's, FURPS, ISO 9126, Dromey's, have been developed for quality evaluation using hierarchically related characteristics of quality indicators. The ISO/IEC 25030 standard provides some models within its framework. This set of models can be combined with software development process quality models, which also affect the quality of the product. Although there are some characteristics like usability, efficiency, functionality, portability, and maintainability, a

common quality characteristic observed across all these basic models is reliability [2].

The quality of software developed is extremely important in software development. There can be various parameters, by which they can estimate software quality, like software architecture, conformance to functional specifications, ability to scale, adherence to development methodology, and other intangibles which are taken for granted by professionals [3, 4]. Researchers have their way of describing quality models, with the time different researchers gave different types of quality models. Like, Desissen Boeck *et al.* classified quality models into three different types [5, 6]:

- Definition model – defines or describes quality. It is difficult to conduct quality assurance as these models are generally too abstract and lack information about their operation.
- Assessment model – contains specific instructions, methods, and criteria for evaluation. These can generally be represented as a mathematical model aggregating quality factors metrics'.
- Prediction model – supports activities like defect prediction.

According to all the researchers, more or less concept of quality models is the same. All the models are based on basic quality models (Mc Call, Boehm, FURPS, Dromey, ISO). With time due to an increase in customer needs, making enhanced products all the models were evaluated for better models. On one thought all the models appear to be the same but there is a difference between all the models as a different model approach has a different purpose. The ISO 9126 defines quality, the assessment model uses the metric-based approach to assess the quality of a given system and the prediction model uses the reliability growth model for predicting quality.

An important metric for estimating software quality is reliability. It can be defined as the probability that the software will not fail for a specified time under specified conditions [7]. According to ISO/IEC 25010, reliability is defined as the degree to which software, product, component, or system performs specified functions under certain conditions for a dedicated time [8]. It can also be defined as a consolidation of availability, fault tolerance, and maturity. It is a crucial parameter that helps determine the efficiency of the software system. Its role in market success determination of software is significant as the economic implications can be huge. Many approaches can be used to measure software reliability [9, 10]. Quality is an ongoing concern throughout the software lifecycle, right from requirements specification till the delivery. Due to its importance in failure detection and error prevention, reliability modeling for estimation and prediction process is considered as the new area of research interest by many researchers. With the help of statistical methods, researchers have obtained parameters for predicting the number of error occurrences for use in reliability estimation [11]. The different scenarios encountered during reliability assessment are:

- code correction for the elimination of direct errors
- establishing reliability for identical use case scenarios and testing
- framework creation to reduce new errors when correcting detected errors
- determining the link between errors in source code and the probability of failure due to them

With decades of research for software engineering, the foundations for many object-oriented languages were established to support reuse in a more structured way to enhance quality and increase productivity. Making any software from the starting is a complex process, costly, and time-consuming, this problem can be solved by introducing the concept of reusability of the components, thus component-based systems are introduced. With these objectives, CBSE has been accepted as a new and effective subdomain of software engineering. It is a discipline that includes software component identification, development, adoption, and integration of these components in a large software system. The main aim of CBSE is to support the independent development of components of their compositional designs, analysis, and verification of the complete system. In this environment, it is very important to standardize the components and their integration process for the quality of CBSE. The quality of components has a high impact on the quality of a whole application [12]. Several quality models for CBSE and Component-Off-The-Shelf (COTS) are available in the literature [13]. Nowadays, researchers are showing their interest in working in the area of software reliability. The reliability of a software component relies on the following factors [14]:

- the reliability of services connected to the component, i.e. external services
- the usage profile of the component, including its frequency and parameters in operation
- the reliability of the environment where the component will be executed, including hardware as well as software environments

Till now proposed software reliability models can be categorized as:

- Architecture based models- It describes the operational and architectural profile of the software system for reliability modeling. These models are path-based and state-based.
- Mathematical based models- It includes mathematical formulas for component usage ratio calculation. The usage ratio for a component is the execution time of the components to the overall execution time. These models can be used for estimation of the overall reliability of the system especially in the case of components.
- Soft computing techniques- It is the new approach that is used nowadays for estimating the reliability of component-based systems. Soft computing techniques use imprecise data and are better suited for real-world problems.

2. SOFTWARE QUALITY

A quality model can be thought of as a combination of parameters like traceability, integrity, reliability, clarity of design and documentation, and relationships between different software components. These parameters can then be used to define and calculate quality requirements for a software product. Quality can be defined through different ways like the quality of any living /nonliving object in the universe can be described in the following terms:

"The measure of standard of any object/thing against the other objects/things of the same kind is known as the quality of that object/thing." Or *"It can also be defined as the degree of superiority or inferiority of something as compare to the other things of the same type."*

The above definition can be described with the help of an example, mobile phone quality (i.e. a product) highly depends on the functioning speed of the phone, consistency with the reliability of the phone, phone's ability to handle multiple applications simultaneously, and many other factors. Similarly, quality in software is focused on excellence and improvement in software products. Nowadays, due to the increase in the use of technology, there is a need to develop authentic and reliable software [15].

3. SOFTWARE QUALITY MODELS

Most of the contemporary quality models are classified according to the means and ways of their generation and are hierarchical [16]. These can be broadly categorized into:

- Assumed model - utilize hypothetical relationships among used variables.
- Data based model – use statistical analysis of related matrix.
- Hybrid model – use a combination of hunches and data analysis.

Research on software quality has been ongoing for a long period. In starting, from the various researchers quality models were categorized into two parts:

- Basic Models – which focus on complete and comprehensive product evaluation
- Tailored Quality Models – which focus on the evaluation of components

The Basic Models can be used for any kind of software products and are hierarchical in structure. The six most important are: Mc Call [1977], Boehm [1978], FURPS [1992], Dromey [1995], ISO 9126-1 [2001]; its variants for external (ISO / IEC 9126-2 [2003]), internal (ISO / IEC 9126-3 [2003]) and quality in use metrics: ISO / IEC 9126-4 [2004]. The ISO - 9126 model as a result of inputs from previous models, namely Boehm and McCall models. A new adapted model: ISO 25010 or ISO/IEC CD 25010 was established in 2007 [11]. This is also known as Software product Quality Requirements and Evaluation or SQuare.

Tailored Quality Models started coming up in 2001 with the Bertoia model. Further models were proposed like in 2003 by Georgiado, in 2005 by Alvaro, and Rawashdesh. These quality models are also known as component quality models. They differ from basic models because their applicability is confined to specific application domains, where the relative importance of features may vary according to that particular domain. The need for such models is driven by specialized organizations and their need for evaluation of individual components. Most of these models are adapted from Basic Models, with a slight modification to fulfill the goals of different domains.

Many software quality models are available these days for assessing software quality products. They are based on basic models, like McCall, Boehm, FURPS, Dromey, and ISO. These four models are considered as the basic quality models, all the other quality models are based on these models by enhancing them in one way or the other. This section describes some of the well-known software quality models, from various studies.

3.1 McCall's model (1977)

The first quality model was proposed by Jim McCall in 1977. It describes quality factors, which can be separated into two levels: external parameters, which can be measured directly, and quality criteria, which can be measured objectively or subjectively [17]. The software quality factors can be classified into three categories in this model: Product Operation, Product Revision, and Product Transition. Factors included in each of these categories are:

- Product Operation: Accuracy, Reliability, Efficiency, Integrity, and Usability. These factors determine a customer's satisfaction.
- Product Revision: Maintainability, Testability, and Flexibility. It is used to assess the ease of system adaptation and error correction.
- Product Transition: This determines the ability to adapt to changes in hardware configuration, like distributed processing [4].

3.2 Boehm's quality model (1978)

Barry W. Boehm presented a hierarchical structure for primitive, intermediate level and high-level characteristics. Boehm built upon the primitive and intermediate characteristics in McCall's model and also included hardware performance as a parameter, which was missing in McCall's model. The intermediate level characteristics used in this model are: Usability, Testability, Maintainability, Portability, Flexibility, Human Engineering, and Understandability [18].

3.3 FURPS (1987) and FURPS+ (2000)

FURPS stands for Functionality, Usability, Reliability, portability, and Supportability. This model was introduced by Robert Grady and specifies parameters for each of the 5 factors. Functionality includes capabilities and features of the software. Usability includes the ease of user interface, ease of training, and user documentation. Reliability includes accuracy,

mean time between failures and, frequency, and severity of the failure. Performance measures the conditions of operation such as response time, speed, efficiency, and resource usage. Supportability includes parameters that make software maintenance easier, like serviceability, adaptability, testability, and compatibility [4].

3.4 Dromey's quality model (1995)

Dromey tried to connect software quality attributed to software product properties by focusing on quality attribute and sub-parameters relationship. The 3 principal components to be considered [19] are i) Product properties, ii) high-level quality parameters, iii) linking properties with quality parameters. The properties can be classified into four parts: Descriptive, Contextual, Correctness, and Internal. However, the model does not define how product properties must be realized [20].

3.5 ISO 9126 model

The ISO 9126 model is based on Boehm and McCall models. It focuses on two aspects first one is internal and external quality parameters, and the second one is quality parameters showcasing usage characteristics. Internal quality parameters can be evaluated without execution, like source code, while external parameters require execution to be evaluated. External quality parameters can only be assessed during the system operation or maintenance phase. The quality in use parameters is concerned with the efficiency, effectiveness, and security of the software product and the resulting user satisfaction. The ISO-9126 model brought about standardization in software quality terminology and has subsequently been used for developing tailored quality models. This was made possible due to the standardization of terminology regarding software quality [11]. ISO includes parameters like reliability, usability, functionality, efficiency, maintainability, and portability. Furthermore, usage characteristics focus on productivity, effectiveness, safety, and satisfaction.

3.6 ISO 25010 model

ISO 25010 emerged in 2007, as an update of the existing ISO 9126 model. According to this model, software product quality can be divided into 8 key features and each feature has some specified characteristics. The aim of this model is quality driven software development. One major change in this model is the removal of portability as a key feature. Instead, security and compatibility have been used to encompass some characteristics previously considered part of portability, and also some other characteristics that weren't considered earlier. Portability has been clubbed under Transferability as a characteristic. ISO 25010 shares similarities with the ISO/IEC 9126 model along the lines of internal, external, and quality in use parameters. Parameters used for software quality are reliability, performance, security, maintainability, transferability, compatibility, operability, and functional suitability [11].

4. ANALYSIS OF SOFTWARE QUALITY MODELS

Over the years, several models have been developed for structuring software quality evaluation.

4.1 Capability maturity model (CMM 1991)

This model lays down a five-level path for refining an organization's software development process, with each level resulting in a more organized and systematic process. It was developed by the U.S. Department of Defense-sponsored research and development center, called Software Engineering Institute (SEI), for enhancing software engineering methodologies. The CMM is similar to ISO 9001, as both deals with software development and maintenance standards. However, unlike ISO 9001, which only lays down minimally acceptable quality standards, CMM provides a framework for continuous improvement along with transition steps. The five levels of the CMM model are:

- Initial level – At this level, the processes are not organized and much depend on individual efforts. The successful development of software is difficult to replicate as there are no defined processes.
- Repeatable level – An organization at this stage has established processes, which are properly defined and documented, making it possible to repeat successes
- Defined level – Here, the organization has refined the software development process according to its own needs through standardization, and integration.
- Managed level - The organization monitors and measures its processes for controlling performance quantitatively.
- Optimizing level – The organization focuses on process improvement by continuously monitoring feedback and using new tools and techniques.

4.2 Ghezzi model (1991)

According to this model software, developers can use internal qualities to control software structure and achieve the required external qualities. Internal qualities to be maintained are Accuracy, Reliability, Portability, Reusability, Maintainability, Flexibility, Usability, and Integrity.

4.3 IEEE model (1993)

It defines the standard for software maintenance, which is qualitative. It includes the following factors: Reliability, Usability, Maintainability, and Portability.

4.4 SATC's quality model (1996)

Software Assurance Technology Center (SATC) at NASA establishes metrics to measure the achievement of goals for improving software quality. The SATC defines four goals: requirements quality, product quality, and implementation effectively and testing effectively [4].

4.5 Bansiya's QMOOD model (2002)

The Quality Model for Object-Oriented Design is hierarchical and extends to Dromey's model. There are four levels in this model:

- Identifying design quality characteristics for Object-Oriented systems.
- Identifying Object-Oriented design properties for internal and external functionality of design components.
- Identifying Object-Oriented design metrics like no. of methods in classes.
- Identifying Object-Oriented design components for architecture design, like objects, class hierarchy.

4.6 Kazman model (2003)

This model focuses on quality characteristics including Functionality, Availability, Efficiency, Reusability, Testability, Security, Portability, Inheritability, and Modifiability.

4.7 Aspect-Oriented Software Quality Model (2006)

This model extends the ISO 9126-1 software quality model by introducing new sub characteristics; modularity, reusability, and complexity, to the ISO 9126- 1 model's original characteristics. The primary focus of aspect-oriented development is modularity [4].

4.8 Software Quality in Development (SQUID)

Software Quality in Development (SQUID) approach was proposed by Kitchenham et al. for defining software quality requirements. This approach defines the quality model using two components: structural component defines the model elements along with their relationships, and content component for linked entities in the structure. This model incorporates the operational behavior of the software, along with quality characteristics, to define quality requirements [21].

4.9 Ortega's systemic model

Ortega's Systemic Model considers the quality and sub-quality characteristics for arriving at quality metrics. The qualities are considered to be related, thereby resulting in a systemic global quality design comprising of four evaluation dimensions related to the product as well as process: product and process efficiency, product, and process effectiveness [22].

4.10 Factor-Strategy approach

Marinescu and Ratiu proposed a Factor-Strategy approach to overcome the limitations of other quality models in addressing the quality design aspect. The proposed model creates design rules quantifiably with the help of detection strategies [23]. The distinctive feature of this model is its use of a computational approach to quantify the association of quality factors to strategy.

4.11 Design Enhanced Quality Evaluation (DEQUALITE)

Khomh and Gueheneuc proposed the DEQUALITE model for object-oriented systems. This model uses quality parameters as well as design aspects for evaluating software quality [24]. The model defines quality metrics for parameters. These metrics, along with design patterns and quality parameters make up the key elements of the DEQUALITE model. The consideration of design elements is what makes this model unique.

4.12 Quality Meta-Model

Finne proposed a quality meta-model that lays down abstract concepts that can be instantiated for system development as and when required [25]. It suggests a three-level view for quality modeling: meta, general, and instance level. The model considers information systems, use cases, quality parameters, and quality metrics as key elements.

4.13 Quamoco

Quamoco is a meta-model proposed by Wagner et. al. The model uses operationalized quality models to try and bridge the gap between quality characteristics and quality measurement [26]. A unique feature of this model is the Product factor, like Dromey's quality carrying property. Product factors, quality aspects, and entities are the key elements of this model. However, it does not address the design or process aspects of quality.

4.14 Rawashdeh model

This model takes after Dromey and ISO 9126 model, focusing on COTS components and fulfilling the objective of satisfying different types of user needs [27]. The model details a four-step approach to creating a software quality model:

- Identify high-level quality parameters
- Obtain a set of subordinate parameters from high-level parameters
- Define internal and external metrics where internal metrics focus on parameters related to the source and external metrics focus on parameters related to system behavior
- Identification of parameters specific to different user types

4.15 Open source models

Open-source software is gaining traction due to the increased collaboration and freedom offered for use in multiple, diverse scenarios. Open source products are available for all types of computer software including Operating systems, middleware, and end-user applications. Quality models for Open Source software build upon models like ISO 9126 while incorporating some specialized parameters for Open Source software. However, there is no ideal model for Open Source software yet, which can capture all aspects. We'll look at four models in the further section of the paper which will be useful in the analysis of the quality of such software.

4.16 Cap gemini open source maturity model

This model defines indicators for product and application maturity. The model is scored on a scale between 1 and 5, with each indicator assigned a value to arrive at a total score.

4.17 Open BRR model

The Business Readiness Rating framework builds on the ISO 9126 and Cap Gemini models. The model specifies seven categories for accelerated evaluation of open source software, which can be further refined for simplification and examination at a granular level [27].

4.18 SQO-OSS model

This hierarchical model takes allows automatic calculation of metrics by taking the community process and source code into account. A major feature of this model is the ability to reduce user interference by focusing on automation in a continuous monitoring system. However, the model focuses only on the source code and not the product functionality. Even the community factors considered are the ones that can be automatically measured [27].

5. STUDY AND REVIEW OF LITERATURE

This study and literature review provides some of the best and relevant explanations/ suggestions to resolve problems of researchers or beginners who are working in the field of reliability. This review may help different researchers/ developers on how to predict the quality of software based on Component-Based Software. Software Quality Assurance (SQA) models define different types of quality attributes given such as availability, maintainability, reliability, and usability, etc

Suman and Manoj Wadhwa [4] analyzed multiple software quality characteristics concepts and discussed the comparative analysis software quality parameters of software quality models. Quality parameters which are compared with all quality models are functionality, efficiency, compatibility, usability, reliability, security, integrity, maintainability.

Padmalata Nistala et al. [20] compiled research on software quality models to envelope model components and support quality architecting. Systematic mapping was used to classify 238 primary papers based on the type of research, publication trends, and standard usage. The team found 40 papers on quality models and used them to analyze meta-model elements and their usefulness in supporting quality architecture using Bayer's reference architecture framework. The results showed 100% support for quality planning, 75% support for quality assessment, 40% support for quality documentation, and 13% support for quality realization aspect. According to the author, software processes control quality realization and, the evolution of quality models and software architectures is necessary to correlate quality definition and realization.

Singh and Bharti [27] reviewed the major existing models while listing out their pros and cons. According to the author, they stressed the importance of communication in software quality. It was further found that some existing models were quite general and not easily applicable to specific cases. Some tailored quality models were discussed, like the ISO 9126 and models for Open source software with an emphasis on community members' participation. These tailored quality models have been derived from basic models considering a specific domain and relevant features and sub-features. Since these models are built for a particular domain, they have limited scope.

Kenett et. al [28] explored an extended quality conceptual framework for assuring product quality. According to the author, the focus was on placing software quality in perspective by three performing critical activities:

- Requirement definition and change monitoring
- Implementation method design and quality achievement
- Evaluation of product and process quality.

Seffah et al. [29] highlighted the quality of service attribute from a usability perspective. Several methods were discussed for a cost-effective usability assessment. Cost-effectiveness was taken as a factor due to its direct relationship with stakeholder satisfaction.

Gupta and Kumar [30] presented a framework for future examination using software reliability facts based on current research. According to the author, the reliability management plan executed by a software engineer through the life of a software product was evaluated and the need to reduce efforts spent for measuring the reliability of object-oriented design within estimated budget and time was emphasized.

Singhal and Singhal [31] focused on reliability as the determining factor for predicting the struggle required for testing software. Based on the literature review of existing models of reliability and structure used by object-oriented design for reliability, a need for reduction in reliability measurement effort was identified for sticking to estimated time and budget.

Nagar and Thankachan [32] discussed traditional development models like Waterfall, spiral, and prototyping as well as advanced models like Agile before proposing an algorithm for identifying an appropriate model for software reliability improvement. According to the author, they cited an example of medicine and manufacturing to illustrate reliability in terms of structural and functional quality.

Khoshgoftarr et al. [33] used a neural network model for predicting the quality of the software. According to the author, the neural network showed better accuracy as compared to a non-parametric discriminant model. However, object-oriented faults were not sufficiently predicted using the metrics derived in this model.

Punia and Kaur [34] put forward a method to predict software maintainability levels on a five-level scale, ranging from very well to very poor, using soft computing techniques and MATLAB's fuzzy logic toolbox. The toolbox helped create rules and generate training and test data sets, which were then fed into a multilayer feed-forward neural networks. According to the author, the method was evaluated using Mean Relative Error (MRE) and Mean Absolute Relative Error (MARE). The experimental results showed reasonable levels of accuracy and usefulness of Artificial Neural Network (ANN) in predicting software maintainability.

Fahad et al. [35] conducted a comparative analysis of software quality models and various metrics associated with those models for predicting software reliability. According to the author, characteristics like size, performance, complexity, quality, etc. were considered for evaluation using three proposed models: McCALL, BOEHM, and ISO9126.

Bakota et al. [36] sought to build a probabilistic approach that could use expert knowledge to deal with imprecision while computing complex quality characteristics. It used the freedom offered by ISO9126 standard to propose a new approach while focusing on maintainability. An acyclic graph with nodes corresponding to inward-looking (source code) and outward-looking (execution performance) quality properties were constructed to determine quality characteristics. The measures of these characteristics were expressed as a goodness function on an interval scale, where 0 and 1 are the worst and best cases. According to the author, the results showed changes in quality models with the occurrence of maintenance activities in a positive correlation. Development activities could be revealed by changes in values.

Sibisi et al. [37] created a framework for quality requirement specification and defined the characteristics in ISO/IEC 9126-1 (2001). According to the author, the research focused on creating a framework for adapting software quality models that could work on an intermediate or end software product and meet different customer and organizational needs. While a general quality profile questionnaire is used to select reliable metrics and rating levels, it requires an objective approach to select appropriate characteristics and sub-characteristics at the product level. Results were validated by focusing on seven factors listed in ISO 9126-2: Reliability, Repeatability, Reproducibility, Availability, Inductiveness, Correctness, and Meaningfulness. It was found that the validation process was successful at the characteristic level, whereas, sub-characteristics level validity required further improvements.

Miguel et al. [11] took a user-centric approach for proposing models to identify quality issues leading to several new measures such as reusability, configurability, availability, lower cost, and better quality, were considered for evaluating the components. Some of the models, with a range in a small domain, have been adapted from ISO 9126. Basic models can also be adapted to build custom quality models as per requirement. Open-source models highlight community-driven requirements.

Menasce et al. [38] suggested using OoS-aware software components to mediate performance properties at runtime. The components could build a QN model to analyze OoS service requirement ability by using past performance properties.

Yang et al. [39] used a neural network called FALCON (Self-Adaptation Learning Control Network) on a set of artificial data. The model measures properties like performance, stability, and reliability. However according to the author, due to lack of application on an actual data set, the research is still in the preliminary stage.

Banerjee et al. [40] suggested an evaluation framework and metrics for a conceptual level component-based system along with their theoretical validations. According to the author, the framework was tested on the library management system and quality evaluation was conducted from two viewpoints: designer and use, covering complexity, completeness, expressiveness, and analyzability parameters.

Sheoran et al. [41] proposed a method for software quality prediction using a hybrid technique to be made up of improved particle swarm optimization and artificial neural network. According to the author, the system performance is measured using metrics maintainability, reliability, and cost. The hybrid algorithm uses a hierarchical model for quality prediction and shows improvement as compared with existing techniques.

Sharma and Bano [42] collected defect reports from six different IT organizations as data for identifying potential reliability factors of software. According to author findings, requirement analysis, cost and size estimation and software defects/ faults were important in determining reliability.

Khatetneh and Mustafa [43] predicted software failures using a newly developed fuzzy expert system. According to the author, the proposed model showed that predictive accuracy increased when particular dataset behavior was taken into account by the model.

Sharma and Dubey [44] predicted software reliability by analyzing various methods/approaches used in the past by reading various research papers. According to the author, their main focus is to explain the development of different models/approaches using metrics of object-oriented and predicting software reliability in a system. The results give some of the answers to research gaps found in the previous research and points out the key areas which help improve reliability.

Antony and Dev [45] measured software reliability using CK metrics. According to the author, a relationship between object-oriented metrics and reliability was established using values of metric parameters extracted from Java Class Analyzer. Findings showed that higher system reliability could be achieved by keeping the high value of NOC and low caules of LCOM, WMC, CBO, RFC, and DIT.

Mishra and Dubey [46] analyze the reliability of object-oriented software systems using the Analytic Hierarchy Process

(AHP) approach. The author used CK metrics mapped with reliability sub-characteristics.

Su-Hua Wang et al. [47] discussed the importance of good quality software, some basic quality models like Boehm, McCall, Dromey, and FURPS quality model. Also, all the models are compared to find the differences between them. According to the author, the paper concluded that McCall and Dromey model is not giving good results in the field of software quality engineering whereas, ISO/IEC 9126 model is the most suitable model which supports most approaches of quality.

Marcin Wolski et al. [48] defined a measurement framework, based on McCall and Boehm models. This framework was used for the evaluation of software products developed within the research and innovation project GÉANT. It includes the EU as an external stakeholder, which is the temporary funding agency hoping to make the project self- reliant in the future. According to the author, they published the results of the application of this model on 2 projects developed under the GEANT ecosystem, and one under the open-source system.

Karnavel and Dillibabu [49] presented the COQUALMO model for identifying defects and their limitations. The IT industry aims to decrease cost and time for evaluation without compromising on quality because residual defects are a major drain on cost, time, and effort. According to the author, they developed a novel model called Software Testing Defect Corrective Model (STDCM) to evaluate the faults occurring continuously in a software product. The developed model was validated using statistical inferences, showing enhanced products of software concerning quality.

Sharma and Sharma [50] discussed multiple quality models to compare them with each other while laying down the importance of metrics and software quality models to build quality software.

Deissenboeck et al. [51] segregated the software quality models into three broad areas, based on the purpose of the model: Definition, Assessment, and Prediction (DAP). According to the author, they listed down the scenarios in which these models were applicable and derived a set of requirements, to be used to either i) evaluate existing models or ii) further develop and improve existing quality models, in terms of DAP classification.

Gordieiev et al. [52] analyzed the development and features of existing quality models, including McCall, ISO/IEC 9126 (2001), and ISO/IEC 25010 (2010), in terms of software reliability and Green Software (GSW). According to the author, they applied a weighted metric approach for quality parameters, and the changes in those weights as the models evolved, to investigate and predict the complexity of software quality models.

Kläs and Münch [53] defined two types of quality models: i) fixed models and, ii) define- your-own models. According to the author, it helps overcome the challenges in selecting the appropriate quality model for software development projects according to their unique features by listing down the possible variables before finalizing the elements for a balanced quality model.

Shukla and Verma [54] took a philosophical and managerial view of software quality. According to the author, they compared the famous software quality models on various parameters and concluded that quality models are needed by software developers to attain optimum software quality.

Saini [55] studied software quality models to quantify factors that affect quality. It was found that maintainability is a critical parameter and should be taken into account right from the initial development stage. This helps detect and correct errors early in the lifecycle, thereby reducing the cost of development. According to the author, they provide an overview and analysis of maintainability in various software quality models.

Qiuying et al. [56] described a model that considered a combination of fault removal efficiency and testing coverage information along with error generation based on a non-homogeneous Poisson process. The model included fault introduction rate as well into software reliability analysis, expressing fault detection rate, and considering fault repair with the use of testing coverage and fault removal efficiency respectively. The model's efficiency was tested over three sets of real failure data and compared with multiple non-homogeneous Poisson process SRGMs on five criteria. According to the author, it was observed that the model yielded better predictive performance.

Liang et al. [57] incorporated a vast amount of background knowledge while building an informationally diverse model. Due to increased informational diversity, increased conflict was observed to the completion of tasks. On the other hand, conflicts based on task tend to provide learning opportunities which help in improving the software quality. According to the author, this was confirmed using survey data from 299 members from 75 software development teams. The learning aspect of the model includes helpful information from the larger pool.

Jie Xu et al. [58] presented several techniques and a step-by-step procedure to derive quality estimation models. The effectiveness of metrics is verified using a mix of machine learning and statistical techniques. To increase accuracy, the authors also adopted a neuro-fuzzy approach. For analysis, data from the ISBG repository is used. According to the author, the combination of statistical modeling and the neuro-fuzzy approach is a uniquely positive feature of this research, as seen from the observed improvements after the application of the recalibration method on a statistical model.

Pai and Dugan [59] used fault trees and Bayesian belief networks to propose a new software reliability estimation approach, which takes causal dependence between processes into account and results in a realistic estimate of software reliability.

Khoshgoftaar and Allen [60] suggested classification rules, which considered the efficiency and effectiveness of the software reliability models explicitly. The application of the rules was demonstrated using two case studies: 1) Modeling of class membership in fault-prone models as a function of a combination of process metrics depicting a module's development history in a military system. 2) Modeling of class membership of fault-prone models as a function of the number of interfaces to other modules in a telecommunication system. According to the author, balanced misclassification rates provided practical and useful SQ models as compared to other classification rules.

Pandey et al. [61] analyzed some software quality models, each considering a multitude of characteristics. According to the author, they emphasized the critical role of customer feedback and systematic processes across the SDLC in quality estimation. Reliability and Requirement factors were suggested as selection factors for quality models.

Gao et al. [62] highlighted the importance of attribute selection in data preprocessing for quality modeling. According to the author, they presented four techniques for attribute selection and demonstrated their applicability to a telecommunications software system. AUC and BGM parameters were considered for the evaluation of classification accuracy. KS method was shown to provide better results than PS, AHS, and RS methods. The KS method showed remarkable performance even when working with less than 15% on a subset of the original attribute set, comparable with a complete set of parameters. The model could be used to greatly enhance model evaluation, calibration, and validation times for evaluating software development.

Alshathry et al. [63] suggested a QA optimization approach by using pre-determined risk rating levels to prioritize investment efforts during evaluation. According to the author, the aim is to employ highly effective practices to high-risk rating work, to build a holistic model considering cost, schedule, and quality.

Reza and Abdul [64] tried to combine analysis of time series with OSS for obtaining software quality assurance, based on statistical techniques. According to the author, this could be used to forecast and predict probabilistic quality properties which can't be evaluated using existing models. An added benefit is the increased productivity and reliability of OSS components.

Khoshgoftaar et al. [65] proposed a classification-tree model to predict components having errors/bugs and then targeting them for enhancement efforts. According to the author, they applied the TREEDISC algorithm on a large legacy telecommunications system to show how TREEDISC models identify and label fault-prone software modules based on process and execution metrics.

Loh and Lee [66] proposed a divide-and-conquer strategy based quality model (QUAMO) for measuring the quality of OO systems using data mining techniques and OO design metrics. According to the author, the model was derived from five existing models, namely Boehm, McCall, Dromey, FURPS, and ISO 9126. The objective was to ease the process of comparing similar studies on software quality.

Seiffert et al. [67] addressed the class-imbalance problem in the context of software-quality prediction. The team examined data boosting and sampling techniques which help in decision-tree modeling while selecting software modules. About 50000 models were built using sampling techniques with 5 data and boosting with 15 software quality data sets. According to the author, it was seen that boosting provided better results than data sampling techniques.

Drown et al. [68] defined a data sampling method using genetic algorithms. This Evolutionary sampling method was used for designing a high-assurance system and comparison was made with different data sampling techniques like one-sided selection, random oversampling and undersampling, Synthetic Minority Oversampling Technique (SMOTE), cluster-based oversampling, Wilson's editing, and Borderline-SMOTE. According to the author, two real-world software systems were used as case studies and it was seen that Evolutionary Sampling provides better results than most data sampling methods

Huang and Zhu [69] applied a multi-instance perspective using a multi-instance kernel to analyze problems in OO software quality estimation. In the training phase, each class was taken as an instance, whereas class hierarchy was considered a bag. The objective was to estimate the likelihood of fault in untested class hierarchies using the altered data from the last phase and software metrics. A hierarchy with faults must include a minimum one negative (fault-prone) class. The evaluation was implied using industrial optical communication software on five datasets. According to the author, the results obtained from the combination of support vector algorithms and a dedicated multi-instance kernel showed more accuracy in estimating fault-proneness of a class hierarchy.

Yi Liu et al. [70] used datasets from seven NASA software projects to study software quality classification modeling effectiveness using multiple validation datasets. It was found that 70% of cases can locate the best two models among six by using five validation datasets. According to the author, they designed a GP-based statistical quality control classifier consisting of three phases: training, multiple dataset validation, and voting. The performance of this approach was better when multiple datasets were used from multiple software having similar reliability goals.

Xiong et al. [71] proposed a model named as Outsourcing Software Quality Management (OSQMM) that takes into account customer satisfaction by analyzing the voice of customers. Issues with the development of outsourcing software and current models assuring quality were analyzed for applying Quality Function Deployment (QFD). According to the author, the effectiveness of the model is verified using the development of an accounting system as a case study.

Khoshgoftaar and Gao [72] focused on identifying fault-prone modules during the development phase itself for improvement in software quality. This results in a focused approach to

software quality and reliability enhancement by allowing the team to estimate the faults likely to be in a given module/project. According to the author, they discussed the inability of logistic regression models to provide a quantitative value for the number of faults, while highlighting the usefulness of the Zero-Inflated Poisson (ZIP) and Poisson Regression Model (PRM) regression model which provides quantitative and qualitative prediction.

Khoshgoftaar and Liu [73] proposed a multi-objective optimizing classification solution using a decision tree model based on genetic programming. Genetic programming was used because of its suitability in solving multi-objective optimization problems. According to the author, the two main objectives: i) Minimize the 'Modified expected cost of misclassification', ii) Optimize the number of predicted fault-prone models in line with allocated resources. An industrial software system was used as a case study to show the usefulness of the proposed model.

Agrawal and Chari [74] studied the impact of mature processes on time, quality, and efforts by considering 37 CMM level 5 projects from four organizations using a linear regression model. It was found that software size was the only significant factor that impacted time, quality, and efforts when highly mature processes are used. According to the author, as a result, high levels of process maturity resulted in a reduction of variance in software development outcomes caused by factors other than size.

Huo et al. [75] compared the waterfall model with agile methodology and studied the effectiveness of agile processes in maintaining software quality while navigating the unstable requirement environment and time pressure. According to the author, they want to show the quality processes associated with both approaches.

Xiong et al. [76] focused on the role of the Quality Function Deployment (QFD) method in controlling the problem of requirements change in software development before proposing a new Dynamic Quality Function Deployment (DQFD) method based on SECI. According to the author, DQFD is combined with a software structure design method to control requirements change while amalgamating the practice and theory of knowledge management. The new software requirement change management approach is applied and validated by its application on a real-world development project.

Bettenburg and Hassan [77] According to the author, they studied the impact of social structures connecting end-users and developers on the quality of software using statistical models. Social information is mined from version control and issue tracking repositories of an open-source project to identify the predictor variable. The results obtained from the statistical models showed similar explanatory power as traditional models.

Alsultanny and Wohaishi [78] focused on increasing the productivity of the software by considering the complexities faced by designers and implementers by testing various factors that affect software quality. According to the author, they proposed a model for providing reliable and quality software that satisfied ISO 9126 requirements by studying reliability, usability and risk management for quality improvement, and understandability, maintainability, and compatibility for productivity improvement.

Kanellopoulos et al. [79] leveraged the ISO/IEC-9126 standards to develop a method for evaluating source code quality and software behavior. The elements are automatically derived from source code and enhanced with quality characteristic rankings, using software engineer's expert knowledge to weigh source code parameters. The metrics and parameters used are quite flexible. According to the author, the application of the proposed model on one proprietary and five open source systems showed that it can quantitatively capture expert opinions and quality trends regarding system quality.

Almakadmeh et al. [80] defined test techniques based on generic model-based methodology. These can help the developers and testers in error handling, especially the beginners, to effectively implement the testing process with optimal effort, time, and cost.

Cristescu et al. [81] showcased the utility of CMMI in software engineering modeling as well as process maturity assessment of an organization. The activities in Software Reliability Engineering (SRE) were described by the author along with their capability for control over the development cycle of software products. Software reliability estimation process models were considered successful in the discovery of the ideal framework for their application.

Mehdi Gheisari et al. [82] developed a mathematical model for optimal prediction of stakeholder satisfaction. The model used constraint equations and validated real data using the impact of relationships among different quality parameters. The successful results showed the usefulness of the proposed optimal model and further scope for its exploration.

Phama et al. [83] discussed the limitations of current models of reliability prediction based on component-based systems as they are unable to incorporate factors like error propagation, concurrently present errors and fault tolerance mechanisms, while modeling system reliability. The authors went on to present a modeling schema whose models are transformed into Markov models for reliability prediction using the authors' approach. The approach was then applied in two case studies for predicting reliability and analyzing sensitivity.

Li and Smidts [84] designed a ranking system for quality parameters. Expert estimation was used to arrive at a quantitative value for each measure used in ranking. Those measures were then combined to derive a single score using the utility of multiple parameters. The authors showed that a more accurate software reliability prediction could be obtained using this calculation method in each development phase.

6. ASSESSMENT OF SOFTWARE QUALITY MODEL AND THEIR QUALITY PARAMETERS RELATED TO CBS

Quality parameters for component-based systems are constrained by the limitations of OSS and COTS quality prediction. While OSS's emphasis is on the availability of code for reuse, COTS' focus is on user interfaces. The researchers need to focus on models that incorporate parameters for both types of software. The literature review shows how researchers have tried to overcome this need for a uniform assessment model.

Sharma et al. [85] analyzed the scope of software maintenance, encompassing any changes to the software product: error corrections, optimizations for better performance, capability enhancement, and removal of unwanted features. According to the author, they discussed the differences in maintenance activities required for component-based software (CBS) as compared to legacy software

Bosch et al. [86] discussed the emergence of CBSD is the major approach to even COTS products due to the focus on object-oriented development. According to the author, the reliability claims associated with CBSD products need to be investigated further.

Jing et al. [87] discussed the importance of Quality of Service (QoS) provisioning in the life cycle of distributed systems. This required understanding of more than just the functional properties of individual components in component-based systems. According to the author, it becomes difficult to predict the quality of service where there is little information about the components is limited like a black box.

Cai et al. [13] discussed how architecture design in component bases software systems can be enhanced to develop mature applications on top of the quality and efficiency offered by CBS. According to the author, this allows for reinforcement of process quality along with product quality due to the complementary features of the design pattern.

Sanz et al. [88] defined parameters for measuring the reusability of learning objects, namely modularity, traceability, modifiability, usability, self-contain ability, and standardization.

Bosch et al. [89] proposed the inclusion of usability as a quality attribute to be considered while designing the system, rather than measuring and implementing it in a fully finished system. According to the author, implementing usability changes after the implementation of results in high costs and limited improvements are possible due to its impact on the underlying architecture.

Gill et al. [90] discussed the advantages offered by component-based software systems and how the reliability of such systems is measured. According to the author, they proposed that the failure of one component did not affect the overall system equally as against the prevailing notion. The failure of one component could affect other components to varying degrees, depending on the relationship between those components.

Sheoran and Sangwan [91] used existing models for software quality prediction. The results from the Software Quality Model (SQM) were compared with ISO 25010, Component-based quality model (CBQM), ISO 9126, Bertoia, and Alvaro model. According to the author, the study was conducted using secondary sources of data. Several characteristics were considered, like reliability, usability, maintainability, and portability, along with sub-characteristics like understandability, performance, compatibility, and accuracy. It was found that the Alvaro model was able to provide better results, especially in terms of accuracy, testability, and understandability.

Palviainen et al. [92] presented a coherent approach for reliability evaluation during the design and implementation stage by combining measured and predicted values with heuristic estimates. According to the author, assessment processes of Component-level reliability and system-level reliability estimation activities are integrated for iterative development of CBSD systems, with increased reliability.

Tomar and Tomar [93] discussed the issue of preserving quality in component-based software system development, which is largely determined by reliability. Since most components are black- boxes, reliability prediction is quite difficult. According to the author, they developed the Component-Based Reliability Model (CBRM) to predict the reliability of components, individually and upon integration. Two factors were used to measure the output of CBRM: Component Reliability (CR) for individual components, and the Average number of interaction failures for integrated components.

Arora et al. [94] focused on the importance of quality of service while developing distributed systems. The author went on to discuss the benefit of CBSD systems while looking to improve the quality of service as compared to COTS systems, due to the black-box nature of COTS products.

Mahmood et al. [95] discussed the different development methodologies like object-oriented and CBSD, and how CBSD can be used in both COTS and open source development projects. This makes it especially relevant in software development discussions. As a result, there is a need for detailed requirement analysis and methods for quality attribute identification for CBSD.

Khoshgoftaar et al. [96] proposed a tool based on Case-Based Reasoning (CBR) for quality modeling. CBR provides an automated reasoning process which makes it attractive for quality analysis of high-reliability software systems. According to the author, they called this CBR tool SMART (Software Measurement Analysis and Reliability Toolkit), which supports three types of modeling: i) CBR based classification ii) Extended CBR classification with cluster analysis iii) Module-order models for predicting rank-order of modules according to a quality factor.

Panwar and Tomar [97] Software industry focus on two key aspects while defining software product quality: i) customer satisfaction, ii) profits. Component-based development aids in both as it reduces the development time which aids in customer satisfaction, as well as the costs of development by reducing man-hours required for development due to reusability of components. According to the author, they proposed a model for measuring the degree of stakeholder satisfaction (Q) through a combination of quality attributes. The results showed an improvement of 2.46% when the model was validated using the Shuffled Frog- Leaping Algorithm (SFLA).

Singh and Tomar [98] discussed the benefit of Component-based Software (CBS) development due to the rapidity of development once constituent components have attained maturity. However, reliability estimation is a concern. According to the author, they proposed a reliability prediction model using two measures: i) component impact factor, and ii) path propagation probability. The impact of individual components on overall reliability is gauged in the form of component impact factor and the reliability of the integrated system is factored in using the probability of propagation of errors along the execution path in the overall system. The proposed model was implemented in Java, on a sample case study, which showed that the model can be used in the initial stages of development for CBS systems' reliability estimation.

Software developers often use quality models to fit their requirements to alleviate customer apprehensions. However, the models may not be suitably applicable always. For example, assessing a component-based software system using ISO 9126 may be counter-productive as ISO 9126 does not include reusability as a parameter, which is especially important in component-based systems. ISO 9126-1 focuses mainly on 6 quality attributes:

- Reliability
- Functionality
- Efficiency
- Maintainability
- Portability
- Usability

The lack of reusability as a quality attribute makes it almost impossible to judge a component-based system using ISO 9126-1, as the reuse of components is the main driving force behind the idea of CBSD.

Other parameters like maintainability, reliability, etc are also important to analyze as these help in effective quality prediction and assessment of software products according to customer requirements. Having processes in place for development, execution, and deployment of products with these attributes in mind help in moving organizations up the Capability Maturity Model (CMM) ladder. However, it is also important to include context-specific parameters for models to be effective at fulfilling their purpose; like in the case of inclusion of reusability as a parameter for CBSD.

According to Panwar et al. [99], they emphasized the importance of stakeholder satisfaction as a key measure of the quality of a product, irrespective of the industry. The two major forms of software development markets, i.e. Commercial-off-the-shelf software and Open Source software are leaning towards component-based- software development. Hence, it is imperative to develop models suited for reliability prediction of Component-Based Software (CBS). The researchers used the firefly optimization technique of computational intelligence to derive an objective function for software quality prediction on MATLAB. The proposed model was tested on real data and provided better results than existing models.

As we all know that software is developed step by step according to software development life cycle, to achieve fault-free system testing at every phase of the cycle gives a promising result in the area of reliability especially. To make time and cost-efficient software, components are reused but the challenge is that components should be reliable, deployable, and reusable. For making a reliability prediction model, components should be chosen very carefully according to the requirements.[100,101,102]

When treating some quality characteristics it is important to know how to obtain them i.e. how to build software to attain the highest degree of stakeholder satisfaction. Developer-oriented quality factors have a great impact on the fundamental structure of the product. Quality characteristics also affect each other positively and negatively. The positive impacts of one attribute on other shows an increase in strength due to the previous one. And negative means a decrease in strength. So, it is important to make some trade-off between them, and at the time of designing a system, it is essential to consider quality characteristics in the fundamental design phase. The development- oriented quality attributes can be calculated using some metrics as shown in Table 1. [103]

Table 1: Quality attributes and metrics

Name of Attribute	Name of the sub-attribute	Purpose	Method of Application	Formula	Scale	Target
Reliability	Maturity	No. of failures during trial	Calculate the no. of detected failures	Z= X1/X2 X1= Detected failures X2= test cases	Absolute	Developer and Tester
	Recoverability	Availability of the system for a specific time	Test system availability to some parameters like repair time	Z=T/A T= Total System Downtime A= Number of breakdowns	Absolute	User and Maintainer
	Fault Tolerance	Breakdown caused due to software until system restart	Total numbers of Breakdown/ Total number of failures	Z= 1-X/Y X= Total breakdowns Y= Total failures	Absolute	User

7. CONCLUSION

In this study, the main focus is to analyze and assess the quality model and prediction of CBS using software quality assurance models, quality characteristics. CBS is a subfield of software engineering and has a tremendous scope of research due to its ability to enhance the quality and production rate of software development. Some parameters work as key quality parameters in the SQA domain. This study explores and analyses existing models to measure the software quality characteristics using parameters for quality prediction. Moreover, the relationship among these quality parameters to measure overall quality has not been explored but helps in identifying the quality factors for CBS. These attributes will help in assessing the various computational intelligence techniques for optimal quality prediction of CBS.

REFERENCES

[1] L. Bass, **Software Architecture in Practice**, *Pearson Education India*, 2007

[2] S. Sproge, and R. Cevere, **Quality Models in Software Product Development Life Cycle**, *International Conference on Applied Information and Communication Technologies*, pp. 69-73, 2012.

[3] H. Gumuskaya, **Core Issues Affecting Software Architecture in Enterprise Projects**, *World Academy of Science, Engineering and Technology*, Vol. 9, pp.32-37, 2005

[4] Suman and M. Wadhwa , **A Comparative Study of Software Quality Models**, *International Journal of Computer Science and Information Technologies*, Vol. 5 no. 4, pp. 5634-5638, 2014

[5] F. Deissenboeck, E. Juergens, K. Lochmann, and F. Informatik, **Software quality models: purposes, usage scenarios and requirements**, *In: Proceedings of the ICSE Workshop on Software Quality*, pp. 9–14, 2009 https://doi.org/10.1109/WOSQ.2009.5071551

[6] 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, Article no. 191101, 2019. https://doi.org/10.1007/s11432-018-9608-3

[7] ANSI/IEEE Std 729-1983, **IEEE Standard Glossary of Software Engineering Terminology**, *The IEEE, Inc*, New York.

[8] https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en, last accessed on-2015.

[9] M. Jedlicka, O. Moravcik and P. Schreiber, **Survey to Software Reliability**, *Central European Conference on Information and Intelligent Systems*, CECIIS, pp. 1-5 2008

[10] S. Khatri R.S., Chillar and A. Chhikara, **Analyzing the Impact of Software Reliability Growth Models on Object Oriented Systems during Testing**, *International Journal of Enterprise Computing and Business Systems*, Vol. 2, no. 1, pp. 1-10, 2012

[11] J. P. Miguel, D. Mauricio and G. Rodriguez, **A Review of Software Quality Models for the Evaluation of Software Products**, *International Journal of Software Engineering & Applications (IJSEA)*, Vol. 5, no. 6, pp. 31-53, 2014.
https://doi.org/10.5121/ijsea.2014.5603

[12] J. Gao, H.-S. Tsao, and Y. Wu, **Testing and Quality Assurance for Component-Based Software**, *Artech House*, 2003.

[13] X. Cai, M. R. Lyu, K.-F. Wong, and R. Ko, **Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes**, *in proceedings of 7th Asia-Pacific Conference on Software Engineering (APSEC)*, pp. 372–379, 2000.

[14] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, **Parameterized Reliability Prediction for Component-Based Software Architectures**, *QoSA 2010, LNCS 6093*, pp. 36–51, 2010

[15] O. Dahiya, K. Solanki, S. Dalal and A. Dhankhar, **An Exploratory Retrospective Assessment on the Usage of Bio-Inspired Computing Algorithms for Optimization**, *International Journal of Emerging Trends in Engineering Research*, Vol. 8, no. 2, pp. 414-434, 2020.
https://doi.org/10.30534/ijeter/2020/29822020

[16] Pressman, R. S. **Software Engineering a practitioner's Approach 7th Edition**. *McGraw-Hill, Inc.*, 2012

[17] D. P. Narayani, and P. Uniyal, **Comparative Analysis of Software Quality Models**, *International Journal of Computer Science and Management Research*, Vol. 2, issue 3, pp. 1911-1913, 2013.

[18] D. Jamwal, **Analysis of Quality Models for Organizations**, *International Journal of Latest Trends in Computing*, Vol. 1, issue 2, pp. 19-23, 2010.

[19] R. S. Jamwal, D. Jamwal and D. Padha, **Comparative Analysis of Different Software Quality Models**, *in: Proc. 3rd National Conference, Computing For Nation Development, Bharati Vidyapeeth's Institute of Computer Applications and Management*, pp. 1-5, 2009.

[20] P. Nistala, K. V. Nori and R. Reddy, **Software Quality Models: A Systematic Mapping Study**, *IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pp. 125-134, 2019.

[21] B. Kitchenham and S. Linkman, **The SQUID approach to defining a quality model**, *Software Quality Journal*, Vol. 6, pp. 211-223, 1997.

[22] M. Ortega, **Construction of a Systematic Quality Model for Evaluating a Software Product**, *Software Quality Journal*, Vol. 11, no. 3, pp. 219-242, 2003.

[23] R. Marinescu and D. Ratiu, **Quantifying the quality of object-oriented design: the factor-strategy model**, *in 11th Working Conference on Reverse Engineering, Delft*, pp. 192-201, 2004.

[24] F. Khomh and Y. G. Gueheneuc, **DEQUALITE: building design-based software quality models**, *in Proceedings of the 15th Conference on Pattern Languages of Programs*, pp. 1-7, 2008.
https://doi.org/10.1145/1753196.1753199

[25] A. Finne, **Towards a quality meta-model for information systems,** *Software Quality Journal*, Vol. 19, no. 4, pp. 663-688, 2011.

[26] S. Wagner, A. Goeb, L. Heinemann, M. Klas, C. Lampasona, K. Lochmann, and A. Trendowicz, **Operationalised product quality models and assessment: The Quamoco approach**, *Information and Software Technology*, Vol. 62 , pp. 101-123, 2015.

[27] D. K. Singh and A. K. Bharti, **A Comparative Studies Of Software Quality Model For The Software Product Evaluation**, *International Journal of Research in Engineering & Technology*, Vol. 6, Issue 8, pp. 1-18, 2018.

[28] R. S. Kenett, and E. Baker, **Process Improvement and CMMI® for Systems and Software**, *CRC Press*, 2010.

[29] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda, **Usability measurement and metrics: A consolidated model**, *Software Quality Journal*, Vol. 14, no. 2, pp. 159-178, 2006.

[30] N. Gupta and R. Kumar, **Reliability Measurement of an Object Oriented Design: A Systematic Review**, *International Journal of Scientific Engineering and Technology*, Vol. 3, no. 12, pp. 1483-1487, 2014.

[31] A. Singhal and A. Singhal. **A Systematic Review of Software Reliability Studies**, *Software Engineering: An International Journal (SEIJ)*, Vol. 1, no. 1, pp. 96-114, 2011.

[32] P. Nagar and B. Thankachan, **Software Reliability Engineering–A Review**, *International Journal of Applied Physics and Mathematics*, Vol. 1, no. 2, pp. 133-137, 2011.

[33] P. M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl and S.J. Aud., **Application Of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System**, *IEEE Transactions on Neural Network*, Vol. 8, issue 4, pp. 92-909, 1997
https://doi.org/10.1109/72.595888

[34] M. Punia and A. Kaur, **Software Maintainability Prediction using Soft Computing Techniques**, *International Journal of Innovative Science, Engineering & Technology*, Vol. 1, issue 9, pp. 431-442, 2014.

[35] S. F. Ahmad, M. R. Beg and M. Haleem, **A comparative study of software quality models**, *International Journal of Science, Engineering and Technology Research*, Vol. 2, issue 1, pp. 172-176, 2013.

[36] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc and T. Gyimothy **A Probabilistic Software Quality Model**, *Proc. Twenty seventh IEEE International Conference on Software Maintenance*, pp. 243-252, 2011, doi: 10.1109/ICSM.2011.6080791.

[37] M. Sibisi and C. C. V. Waveren, **A Process Framework for Customizing Software Quality Models**, *Pub.*

AFRICON, IEEE, pp. 1-8, 2007, doi: 10.1109/AFRCON.2007.4401495.

[38] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy, **Performance by Design**, *Prentice Hall*, 2004.

[39] B. Yang, L. Yao, and H. Huang. **Early software quality prediction based on a fuzzy neural network model**, *in Proc. ICNC' 07, IEEE*, pp. 760–764, 2007.

[40] P. Banerjee, and A. Sarkar, **Quality evaluation framework for component based software**, *in Proc. Second International Conference on Information and Communication Technology for Competitive Strategies*, pp. 1-6, Article no. 17, 2016.

[41] K. Sheoran, P. Tomar and R. Mishra, **Software Quality Prediction Using Hybrid Classifier Based on Improved PSO and Ann**, *Journal of Advanced Research in Dynamical and Control Systems*, Vol. 9, pp. 3016-3029, 2017.

[42] N. Sharma and P. Bano, **A Survey of Software Reliability Factor**, *IOSR Journal of Computer Engineering*, Vol. 12, no. 1, pp. 50-55, 2013. https://doi.org/10.9790/0661-1215055

[43] K. Khatatneh and T. Mustafa, **A Survey of Software Reliability Factor**, *IOSR Journal of Computer Engineering*, Vol. 26, no. 1, 2009.

[44] C. Sharma and S. K. Dubey, **A Perspective Approach of Software Reliability Models and Techniques**, *ARPN Journal of Engineering and Applied Sciences*, Vol. 10, no. 16, pp. 7300-7308, 2015.

[45] J. Antony and H. Dev, **Estimating Reliability of Software System Using Object- Oriented**, *International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR)*, 2013.

[46] A. Mishra and S.K. Dubey, **Fuzzy Qualitative Evaluation of Reliability of Object Oriented Software System**, *IEEE International Conference on Advances in Engineering and Technology Research (ICAETR)*, 2014.

[47] S. H. Wang, D. Samadhiya, and D. Chen, **Software Quality: Role and Value of Quality Models**, *International Journal of Advancements in Computing Technology*, Vol. 3, no. 6, pp. 65-74, 2011

[48] M. Wolski, B. Walter, S. Kupiński and J. Chojnacki, **Software Quality Model for A Research-Driven Organization—An Experience Report**, *Journal of Software: Evolution and Process*, Vol. 30, no. 5, pp. 1-14, 2017. doi:10.1002/smr.1911

[49] K. Karnavel and R. Dillibabu, **Development and Application of New Quality Model for Software Projects**, *The Scientific World Journal*, PMID: 25478594, PMCID: PMC4248366, pp. 1-11, 2014. http://dx.doi.org/10.1155/2014/491246

[50] K. Sharma, and K. Sharma, **Comparison Of Various Software Quality Models**, *Proc. of the Intl. Conf. on Recent Trends in Computing and Communication Engineering -RTCCE*, ISBN: 978-981-07-6184-4d, pp. 48-51, 2013.

[51] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, **Software Quality Models: Purposes, Usage Scenarios and Requirements**, *ICSE Workshop on Software Quality*, pp. 9-14, 2009

[52] O. Gordieiev, V. Kharchenko and M. Fusani, **Evolution of Software Quality Models: Green and Reliability Issues**, *CEUR Workshop Proceedings*, Vol. 1356, pp. 432-445, 2015

[53] M. Kläs, and J. Münch, **Balancing Upfront Definition and Customization of Quality Models**, *Proceedings of the Workshop Software-Qualitätsmodellierung und –bewertung*, pp. 26-30, 2008.

[54] H. S. Shukla, and D. K. Verma, **Analysis of Software Product Quality Models**, *International Journal of Emerging Technologies in Computational and Applied Sciences*, Vol. 15, no. 311, pp. 26-30, 2015.

[55] R. Saini, S. K. Dubey and A. Rana, **Analytical Study of Maintainability Models for Quality Evaluation**, *Indian Journal of Computer Science and Engineering*, Vol. 2, no. 3, pp. 449-454, 2011.

[56] Q. Li, and H. **A** Pham, **Testing-Coverage Software Reliability Model Considering Fault Removal Efficiency and Error Generation**, *PLoS One.* , Vol. 12, no. 7, pp. 1-25, 2017.doi:10.1371/journal.pone.0181524, pp. 1-25

[57] T. P. Liang, J. Jiang, G. S. Klein, and J. Y. C. Liu, **Software Quality as Influenced by Informational Diversity, Task Conflict, and Learning in Project Teams**, *IEEE Transactions On Engineering Management*, Vol. 57, no. 3, pp. 477-487, 2010. https://doi.org/10.1109/TEM.2009.2033049

[58] J. Xu, D. Ho and L. F. Capretz, **An Empirical Study on the Procedure to Derive Software Quality Estimation Models**, *International Journal of Computer Science & Information Technology*, Vol. 2, no.4, pp. 1-16, 2010

[59] G. J. Pai, and J. B. Dugan, **Enhancing Software Reliability Estimation Using Bayesian Networks and Fault Trees**, *International Symposium on Software Reliability Engineering (ISSRE) Fast Abstracts*, 2001.

[60] T. M. Khoshgoftaar and E. B. Allen, **A Practical Classification-Rule For Software-Quality Models**, *IEEE Transactions on Reliability*, Vol. 49, no. 2, pp. 209–216, 2000. doi:10.1109/24.877340

[61] S. Pandey, S. F. Ahmad, and M. Hussain, **A Critical Survey on Quality Models in Software Engineering**, *ACEIT Conference Proceeding, IJCSIT*, pp. 282-284, 2016.

[62] K. Gao, T. M. Khoshgoftaar, and H. Wang, **An empirical investigation of filter attribute selection techniques for software quality classification**, *IEEE International Conference on Information Reuse & Integration*, pp. 272-277, 2009, doi:10.1109/iri.2009.5211564

[63] O. Alshathry, H. Janicke, H. Zedan and A. AlHussein, **Quantitative Quality Assurance Approach**, *International Conference on New Trends in Information and Service Science*, pp. 405-408, 2009. doi:10.1109/niss.2009.114

[64] R. M. Parizi and A. A. A. Ghani, **Towards Automated Monitoring and Forecasting of Probabilistic Quality Properties in Open Source Software (OSS): A**

Striking Hybrid Approach, *Eighth ACIS International Conference on Software Engineering Research, Management and Applications*, pp. 329-334, 2010.doi:10.1109/sera.2010.48

[65] T. M. Khoshgoftaar, E. B. Allen, X. Yuan, W. D. Jones and J. P. Hudepohl, **Assessing uncertain predictions of software quality**, *Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403)*, Vol 1, pp. 1-10, 1999. doi:10.1109/metric.1999.809737

[66] C. H. Loh and S. P. Lee, **Predicting Quality of Object-Oriented Systems through a Quality Model based on Design Metrics and Data Mining Techniques**, *International Conference on Information Management and Engineering, IEEE*, pp 239-243, 2009.

[67] C. Seiffert, T. M. Khoshgoftaar and J. V. Hulse, **Improving Software-Quality Predictions With Data Sampling and Boosting**, *IEEE Transactions On Systems, Man, And Cybernetics-Part A: Systems And Humans*, Vol. 39, no. 6, pp. 1283-1294, 2009.

[68] D. J. Drown, T. M. Khoshgoftaar, and N. Seliya, **Evolutionary Sampling and Software Quality Modeling of High-Assurance Systems**, *IEEE Transactions On Systems, Man, And Cybernetics-Part A: Systems and Humans*, Vol. 39, no. 5, pp. 1097-1107, 2009.
https://doi.org/10.1109/TSMCA.2009.2020804

[69] P. Huang and J. Zhu, **A Multi-Instance Model for Software Quality Estimation in OO Systems**, *Fifth International Conference on Natural Computation, IEEE*, pp. 436-440, 2009.

[70] Y. Liu, T. Khoshgoftaar and J. F. Yao, **Building a Novel GP-Based Software Quality Classifier Using Multiple Validation Datasets**, *IEEE International Conference on Information Reuse and Integration*, pp. 644-650, 2007

[71] W. Xiong, X. T. Wang and Z. X. Wu, **Study of a Customer Satisfaction-Oriented Model for Outsourcing Software Quality Management using Quality Function Deployment (QFD)**, *in proceedings of 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1-5, 2008,

[72] T. M. Khoshgoftaar, and K. Gao, **Count Models for Software Quality Estimation**, *IEEE Transactions on Reliability*, Vol. 56, no. 2, pp. 212-222, 2007.

[73] T. M. Khoshgoftaar and Y. Liu, **A Multi-Objective Software Quality Classification Model Using Genetic Programming**, *IEEE Transactions on Reliability*, Vol. 56, no. 2, pp. 237-245, 2007.

[74] M. Agrawal and K. Chari, **Software Effort, Quality and Cycle Time: A study of CMM Level 5 Projects**, *IEEE Transaction on Software Engineering*, Vol. 33, no. 3, pp. 145-156, 2007.

[75] M. Huo, J. Verner, L. Zhu, and M. Ali Babar, **Software Quality and Agile Methods**, *proceedings of the 28th Annual International Computer Software and Applications Conference*, pp. 1-6, 2004.

[76] W. Xiong, Z. Wu and S. Z. Yu, **Research on The Application of QFD and Konwlege Management in the Outsourcing Software Quality Assurance**, *International Conference on Computer Technology and Development*, pp. 352-358, 2009

[77] N. Bettenburg and A. E. Hassan, **Studying the Impact of Social Structures on Software Quality**, *18th IEEE International Conference on `Program Comprehension*, pp. 124-133, 2010.

[78] Y. A. Alsultanny and A. M. Wohaishi, **Requirements of Software Quality Assurance Model**, *Second International Conference on Environmental and Computer Science*, pp. 19-23, 2009

[79] Y. Kanellopoulos, P. Antonellis, D. Antoniou, C. Makris, E. Theodoridis, C. Tjortjis, and N. Tsirakis, **Code Quality Evaluation Methodology using the ISO/IEC 9126 Standard**, *International Journal of Software Engineering & Applications (IJSEA)*, Vol. 1, no. 3, pp. 17-36, 2010.
https://doi.org/10.5121/ijsea.2010.1302

[80] K. Almakadmeh, and F. Abu-Zitoon, **A Generic Model-Based Methodology of Testing Techniques to Obtain High Quality Software**, *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication-IPAC, ACM*, Article No.: 49, pp. 1–6, 2015, doi:10.1145/2816839.2816903.

[81] M. P. Cristescu, J. A. Vasilev, M. V. Stoyanova, and A. M. R. Stancu, **Capability And Maturity. Characteristics Used In Software Reliability Engineering Modeling**, *Land Forces Academy Review*, Vol. XXIV, no 4(96), pp. 332-341, 2019

[82] M. Gheisari et al., **An Optimization Model for Software Quality Prediction With Case Study Analysis Using MATLAB,** *IEEE Access*, Vol. 7, pp. 85123-85138, 2019.

[83] T. T. Phama, X. D´efagoa, and Q. T. Huynhb, **Reliability Prediction for Component-Based Software Systems**, *Science of Computer Programming*, Vol. 97, part 4, pp. 426-457, 2014, http://dx.doi.org/10.1016/j.scico.2014.03.016

[84] M. Li, and C. S. Smidts, **A Ranking of Software Engineering Measures Based on Expert Opinion**, *IEEE Trans. Softw. Eng.*, Vol. 29, no. 9, pp. 811-824, 2003.

[85] A. Sharma, P. S. Grover, and R. Kumar, **Predicting Maintainability of Component-Based Systems by Using Fuzzy Logic**, *International Conference on Contemporary Computing*, pp. 581–591, 2009.

[86] J. Bosch, C. Szyperski, and W. Weck, **Black Box Programme Specilization**, *in proceedings of 4th International Workshop on Component-Oriented Programming*, 1999.

[87] E. Capra, C. Francalanci, and S. A. Slaughter, **Is Software 'Green'? Application Development Environments and Energy Efficiency in Open Source Applications**, *Information Software Technology*, Vol. 54, no. 1, pp. 60–71, 2012.

[88] J. Sanz-Rodriguez, J. M. Dodero, and S. Sanchez-Alonso, **Metrics-Based Evaluation of Learning Object Reusability**, *Software Quality Journal*, Vol. 19, no. 1, pp. 121–140, 2011.

[89] J. Bosch and N. Juristo, **Designing Software Architectures for Usability**, *in proceedings of 25thInternational Conference on Software Engineering*, pp. 757–758, 2003.

[90] N. S. Gill and P. S. Grover, **Component-Based Measurement: Few Useful Guidelines**, *ACM SIGSOFT Software Engineering Notes*, Vol. 28, no. 6, pp. 4-4, 2003.

[91] K. Sheoran and O. P. Sangwan, **An Insight of Software Quality Models Applied in Predicting Software Quality parameters: A Comparative Analysis**, *Pub. Fourth International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, pp. 1-5, 2015, doi: 10.1109/icrito.2015.7359355

[92] M. Palviainen, A. Evesti, and E. Ovaska, **The Reliability Estimation, Prediction and Measuring of Component-Based Software**, *The Journal of Systems and Software*, Vol. 84, no. 6, pp. 1054- 1070, 2011.

[93] D. Tomar, and P. Tomar, **New Component-Based Reliability Model (CBRM) to Predict the Reliability of Component-Based Software**, *International Journal of Reliability and Safety Publication, Inder Science*, Vol. 13, no.1, pp. 83-95, 2019, Print ISSN: 1479-389X Online ISSN: 1479-3903 (Indexed : Scopus)

[94] A. Arora, V. S. Arunachalam, J. Asundi, and R. Fernandes, **The Indian Software Services Industry**, *Res. Policy*, Vol. 30, no. 8, pp. 1267-1287, 2001.

[95] S. Mahmood, R. Lai, Y. S. Kim, J. H. Kim, S. C. Park, and H. S. Oh, **A survey of component based system quality assurance and assessment**, *Inf. Softw. Technol.*, Vol. 47, no. 10, pp. 693-707, 2005.

[96] T. M Khoshgoftaar, E. B. Allen, and J. C Busboom, **Modeling software quality: the Software Measurement Analysis and Reliability Toolkit,** *Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence*, pp. 54-61, 2000. doi:10.1109/tai.2000.889846 .

[97] D. Panwar, and P. Tomar, **New Mathematical Model for Software Quality Prediction of Component-Based Software Using Shuffled Frog Leaping Algorithm**, *International Journal of Computer Applications in Technology, Inderscience*, Vol. 55, no 4, ISSN: 0952-8091,pp. 266-275, 2017 (Scopus and ESCI Indexed).

[98] A. Pratap Singh and P. Tomar, **A New Model for Reliability Estimation of Component-Based Software**, *proceedings of 3rd IEEE International Advance Computing Conference*, ISBN: 978-1-4673-4529-3/12, pp. 1431-1436, 2013. (Indexed in IEEE Xplore)

[99] D. Panwar, M. Siddique, and P. Tomar, **Software Quality Assurance using Firefly Optimization Algorithm", in book entitled Communication and Computing Systems**, *proceedings of the International Conference on Communication and Computing Systems*, ISBN 978-1-138-029521, pp. 515-520, 2017.

[100] S. Yadav and B. Kishan, **Reliability of Component-Based Systems- A Review,** *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8, no. 2, 2019. https://doi.org/10.30534/ijatcse/2019/31822019

[101] O. Dahiya and K. Solanki, **Comprehensive cognizance of Regression Test Case Prioritization Techniques**, *International journal of emerging trends in engineering research*, Vol. 7 No. 11, pp. 638-646, 2019. https://doi.org/10.30534/ijeter/2019/377112019

[102] S. Yadav and B. Kishan, **Assessment of software quality models to measure the effectiveness of software quality parameters for Component Based Software (CBS),** *Journal of Applied Science and Computations*, Vol. VI, issue IV, 2019.

[103] R. D. Banker and R. J. Kauffman, **Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study**, *MIS Quarterly.*, pp. 375–401, 1991.