A Review Report on Software Quality Measurement and Estimation

Faizan Berlas ¹

¹Virtual University of Pakistan

October 30, 2023

Abstract

Software quality definition encompasses six key factors. These are low levels of defects, high reliability, efficient structure that minimizes insertion of new defects during defect repairs, effective customer support, high user satisfaction, and a high defect repair rate, especially for high-severity defects. Several methods have been proposed for measuring software quality. These include fuzzy logic based frameworks, neuro fuzzy models, unsupervised learning methods, queuing models, and software quality metrics. This research paper reviews different frameworks, models, metrics, and evaluation techniques that have been published in various research papers for estimating and assessing software quality.

Research publications related to software quality measurement, estimation, and assessment were collected for the literature review. These publications were accessed from online research databases that consist of open access (OA) journals.
A Review Report on Software Quality Measurement and Estimation

Muhammad Faizan Berlas
Department of Computer Science
Virtual University of Pakistan
Lahore, Pakistan
ms220400075mfb@vu.edu.pk

Abstract—Software quality definition encompasses six key factors. These are low levels of defects, high reliability, efficient structure that minimizes insertion of new defects during defect repairs, effective customer support, high user satisfaction, and a high defect repair rate, especially for high-severity defects. Several methods have been proposed for measuring software quality. These include fuzzy logic based frameworks, neuro fuzzy models, unsupervised learning methods, queuing models, and software quality metrics. This research paper reviews different frameworks, models, metrics, and evaluation techniques that have been published in various research papers for estimating and assessing software quality.


I. INTRODUCTION

The term software quality is directly correlated with high-quality software. This is because the perception of software quality is different under different circumstances and it is more informative to determine the characteristics of high-quality software rather than define software quality [1]. According to Gaffney [14, p. 126], software quality can be defined as conformance to requirements. Conformance is the ability of the software to meet user needs and satisfy the stated performance criteria [14]. Several models have been proposed to define software quality based on numerous quality factors and criteria [11]. The research study by Gaffney [14, p. 127] proposes a few quality factors as quality metrics [14]. Yadav and Yadav [7] proposed a membership function for software metrics. Sheoran et al. [4] have applied the advanced neural network technique for predicting software reliability. Rizvi et al. [5] proposed a fuzzy logic framework for quantifying software reliability. Similarly, Tyagi and Sharma [6] established a neuro fuzzy model for estimating the software reliability of component-based software systems.

Software measurement is the quantification of characteristics of a software product or a software process [3]. Measurement of software quality is the quantification of characteristics that define software quality. Various software metrics and models have been proposed for software quality measurement and estimation.

In this paper, several publications are reviewed to provide an overview of the research work done in the field of software quality measurement and estimation. The surveyed publications include both traditional research work going as back as the 1970s and recent research that involves the use of machine learning, and neural networks for assessing software quality.

The first step in measuring software quality is to understand software quality. Hence, this paper first presents a review of different software quality models that are proposed for defining software quality [11, 13, 14, 16]-[21]. These quality models built up a necessary foundation for determining the characteristics that are required for measuring software quality. However, these software quality models do not quantify the quality characteristics [3]. For quantifying the software quality characteristics, research studies [2]-[6] have been reviewed and an overview of these studies has been provided.

Software estimation is estimating the development schedule and assessing productivity and quality [15]. Software quality estimation is the estimation of software quality during the development or pre-operation phase [12]. An overview of research work in the field of software estimation, reliability, and software metrics is also provided in this paper.

This paper is assembled as follows: Section II presents the research work that was collected and analyzed for the literature review. Section III provides the findings from numerous research papers and discusses them in the context of software quality measurement and estimation. Finally, section IV presents the conclusions, the limitations of the current research work, and ideas for future research.

II. RELATED WORK

Research publications related to software quality measurement, estimation, and assessment were collected for the literature review. These publications were accessed from online research databases that consist of open access (OA) journals.

Table I shows the list of 14 papers that were collected and presents them in ascending order by the year of publication. For each paper, the area of research work is presented along with a brief description of the research area and the proposed approach provided in the paper.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Year</th>
<th>Research Area</th>
<th>Proposed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>McCall et al. [13]</td>
<td>1977</td>
<td>Software quality measurement</td>
<td>Software quality model and metrics</td>
</tr>
<tr>
<td>Cavallo et al. [13]</td>
<td>1978</td>
<td>Software quality measurement</td>
<td>Software quality model and metrics</td>
</tr>
<tr>
<td>Gaffney [14]</td>
<td>1981</td>
<td>Software quality assurance metrics</td>
<td>Software quality metrics</td>
</tr>
<tr>
<td>Dromey [19]</td>
<td>1996</td>
<td>Software quality measurement</td>
<td>Software quality model and metrics</td>
</tr>
</tbody>
</table>
different software quality models

quality factors are

Portability
Operability, Security, Compatibility, Maintainability,

quality. These models

proposed software quality models based on different quality

factors and their respective characteristics. These models

illustrated in Fig. 1. Cavano [13] also identified and defined
10 different software quality factors. These are listed in Table

characteristics against define
criteria. For example, Correctness is a factor and
characteristics represented by the factor. For example, Correctness is a factor and the characteristics that must exist
in the software to achieve the correctness factor are
completeness, consistency, and operability. Once the software
characteristics for achieving the required quality factor are
identified, a metric is established for quantifying the identified
characteristics against defined criteria. A systematic model of
the software quality framework established by Cavano [13] is
illustrated in Fig. 1. Cavano [13] also identified and defined
10 different software quality factors. These are listed in Table

III. DISCUSSION

This section provides the findings and proposed approaches presented in the research work that was selected for the literature review. The research questions are designed to establish a relationship between research work and software quality measurement.

A. RQ1: How is software quality measured?

This question attempts to find the definition of software quality and determines the approaches that are used to measure quality. Cavano [13] attempts to quantify software quality by establishing a software quality framework. This framework is created by identifying the factors of software quality. These factors are a set of attributes that provide the characteristics represented by the factor. For example, Correctness is a factor and the characteristics that must exist in the software to achieve the correctness factor are completeness, consistency, and operability. Once the software characteristics for achieving the required quality factor are identified, a metric is established for quantifying the identified characteristics against defined criteria. A systematic model of the software quality framework established by Cavano [13] is illustrated in Fig. 1. Cavano [13] also identified and defined 10 different software quality factors. These are listed in Table II.

McCall [17], Boehm [18], Dromey [19], and Grady [20] proposed software quality models based on different quality factors and their respective characteristics. These models serve as the foundation for defining and quantifying software quality. Quantification of software quality based on Functional suitability, Performance efficiency, Reliability, Operability, Security, Compatibility, Maintainability, and Portability has been proposed by [21]. Overall, 17 different quality factors are used for measuring software quality in different software quality models.

Gaffney [14, p. 128] presents the relationship between several important software metrics and their association with qualitative factors of software quality. Gaffney [14, p. 128] also elaborated on the work of Halstead [16] by defining the four metrics developed by Halstead. Table III shows the list of metrics provided by Gaffney [14]. Furthermore, Gaffney [14, p. 129] presented the mathematical definitions of one of the four Halsted [16] metrics and discussed the significance and limitations of these metrics for measuring software quality.

### Table II. Software Quality Factors by Cavano [13]

<table>
<thead>
<tr>
<th>Software Quality Factor</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>The degree to which a software complies with its requirements and carries out the user's goal objectives.</td>
</tr>
<tr>
<td>Reliability</td>
<td>The degree to which a program should be anticipated to carry out its intended functions.</td>
</tr>
<tr>
<td>Efficiency</td>
<td>The number of computing resources and code needed by a program to carry out a task.</td>
</tr>
<tr>
<td>Integrity</td>
<td>The extent to which unlawful access to software or data can be regulated.</td>
</tr>
<tr>
<td>Usability</td>
<td>The amount of effort needed to operate a program, prepare input, and interpret results.</td>
</tr>
<tr>
<td>Maintainability</td>
<td>The time and effort needed to find and correct a bug in a working application.</td>
</tr>
<tr>
<td>Testability</td>
<td>The time and effort put into testing a program to make sure it serves its intended purpose.</td>
</tr>
<tr>
<td>Flexibility</td>
<td>The amount of effort needed to modify an operational program.</td>
</tr>
<tr>
<td>Portability</td>
<td>The time and effort needed to move a program between different hardware and/or software environments.</td>
</tr>
<tr>
<td>Reusability</td>
<td>The degree to which a program can be used in other applications. This factor is related to the packaging of the program and the scope of functions that are performed by the program.</td>
</tr>
<tr>
<td>Interoperability</td>
<td>The amount of work required to connect one system to another.</td>
</tr>
</tbody>
</table>

### Table III. Software Metrics Analyzed by Gaffney [14]

<table>
<thead>
<tr>
<th>Software Metrics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Volume or Intelligence</td>
<td>The function of a unique number of inputs and outputs to a software module.</td>
</tr>
<tr>
<td>Volume</td>
<td>The function between instructions and data labels.</td>
</tr>
<tr>
<td>Difficulty</td>
<td>A measure of redundancy.</td>
</tr>
<tr>
<td>Effort</td>
<td>Degree of difficulty in understanding a program.</td>
</tr>
<tr>
<td>Division</td>
<td>Degree of control complexity. This metric is related to testing effort and is inversely proportional to overall productivity [14, p. 128].</td>
</tr>
</tbody>
</table>

Research work from Lee [11] is a detailed comparative study between different quality models and their relationship with software quality metrics. Furthermore, it presents the relationship between software metrics and software quality assurance activities carried out throughout the software life cycle. Hence, this study can be considered one of the rare studies which attempt to quantify the software quality assurance process with the help of software quality metrics.

Software quality assessment also includes some non-quantifiable factors, such as the test engineer’s experience, software knowledge, domain knowledge, quality awareness, and so on. These non-quantifiable factors cause some fuzziness and uncertainty in the assessment of software quality, hence, it is important to build an unascertained model for measuring software quality [2].


Choquet Integral is an aggregation operator which generalizes the weighted arithmetic mean, uses the concept of fuzzy measure, and can account for multiple independent criteria for measuring the required parameter [3, p. 154]. Pasrija et al. [3] proposed a technique that utilizes ISO/IEC 9126 software quality model and Choquet Integral for measuring software quality. Each software quality criterion is provided as the input for weight. Afterward, a rating is given to each criterion by the decision-makers. Decision-makers can be users, developers, and higher management of the organization. Finally, the ratings collected from different perspectives are then aggregated by Choquet integral to give the required results.

An Advanced Neural Network utilizing a Hybrid Cuckoo Search (HCS) optimization algorithm was used by Sheoran et al. [4] for predicting maintainability and reliability factors of software quality.

Reliability is one of the factors of software quality. The current approaches for measuring software reliability are only useful for later phases of development, and they aid programmers at the end of the coding stage or while the product is being tested. When it happens, it is too late for developers to make improvements to reliability [5]. Rizvi et al. [5] proposed a comprehensive framework for quantifying reliability based on requirement and design stage measurements.

The work by Rizvi et al. [5] is of particular importance to researchers and industry personnel because it identifies the shortcomings in previously published models for quantifying reliability and presents a feasible solution that will overcome the identified problems and limitations. The complete Fuzzy Logic based Software Reliability Quantification (FLSRQF) is given in [5, Fig. 1].

The framework by Rizvi et al. [5] looks quite prospective in nature. Case studies involving the utilization of this framework for quantifying reliability in the early stages of the software development life cycle are necessary for evaluating the accuracy and validity of this framework.

Component-based software systems (CBSSs) are software systems that are designed and developed from reusable components [6]. An adaptive neuro fuzzy interface system (ANFIS) model combining the best features of fuzzy logic and neural networks model was proposed by Tyagi and Sharma [6] for the CBSS reliability estimation model.

The ANFIS model is useful for measuring software reliability in CBSSs, however, the case study in which the model was implemented used only four critical factors – reusability, component dependency, application complexity, and operation profile – for measuring reliability. Other relevant factors for measuring reliability can impact the result. Hence, this model needs to be validated in different CBSSs.

Black-box models for quantifying software reliability do not attempt to model internal structures for measuring reliability [10]. As per Goševa-Popstojanova and Trivedi [10], the white-box approach that takes into account the architecture of the software is more suitable for quantifying reliability in CBSSs.

One of the main limitations of the architecture-based software reliability prediction model is the complexity of the model and data collection.

B. RQ2: What is software quality estimation and how it is measured?

This question deals with the concept of software quality estimation, its relationship with software quality measurement, and the approaches that are used to estimate software quality. Software quality estimation is the ability to estimate the quality of a software product during its development or pre-operation phases [15]. In software quality estimation, software quality measurements and fault data from a prior system release or related software project created by the provided company are used to train or build a software quality classification or software fault prediction model [15]. Lewis and TsL [15] used K-means and Neural-Gas clustering algorithms for software quality estimation.

Fault detection, fault removal, and operational profile are the three main factors that affect software reliability estimation [9]. The fault rectification rate during debugging is significantly influenced by several factors, including the debuggers’ level of experience, the complexity of the faults, the environments and tools used for debugging, etc. As a result, rather than being continuous or smooth, the fault
correction rate is likely to vary at certain moments in time known as change-points (CPs) [9].

Huang and Hung [9] proposed an extended infinite server queuing model with multiple change-points to predict and estimate software reliability. Huang and Hung [9] showed that according to experimental results based on real failure data, their proposed model can depict the change in fault correction rates and predict the behavior of software development more accurately than traditional software reliability growth models (SRGMs).

According to Serban and Shaikh [8], estimating software reliability using design-level metrics is a new area of research. The Package Level Reliability Prediction Framework is given in [8, Fig. 1]. The Reliability prediction is the average of packages predicted as faulty with varying degrees, severity, category, and type of bugs [8, p. 910]. Likewise, the Reliability metric is defined as weighted bug aggregation of varying degrees, severity, and category [8, p. 910]. Based on these definitions, [8] developed a mechanism for quantifying reliability based on architectural level metrics and performed a thorough empirical study to understand the practical significance and value of API and Non-API based package level metrics towards reliability assessment.

The work by [8] is of significant importance because it provides a detailed empirical study for the assessment of software reliability based on package level metrics.

IV. CONCLUSION AND FUTURE WORK

Software quality measurement and assessment is extremely difficult as it involves the quantification of uncertain and subjective parameters. In this research paper, an effort is made to provide an overview of the methods and techniques that have been published for quantifying and assessing software quality. It can be seen from the literature review that the software quality factors and their respective characteristics are well-established and standardized. A lot of research has been published in this area and several software quality models exist for defining the software quality factors and their attributes.

On the other side, the quantification of software quality factors is a challenging task, and although some study has been published to date, the relevance and difficulty of the topic, it appears that only a small amount of work has been published thus far. In the published research work different approaches are employed to accurately measure software quality. Machine learning techniques such as Fuzzy Logic and Advanced Neural Networks are successful in overcoming the uncertain and fuzzy nature of the software quality measurement process and look promising, however, the research in this area is extremely limited. The published case studies do not include large software systems and development projects. Furthermore, there is a need for a detailed empirical study evaluating the usefulness of machine learning techniques in a variety of software development projects.

One of the limitations of this research paper is the small number of published research papers for the literature review. This is because only those research papers were gathered that exist in the open access (OA) journals of online research databases. Moreover, this literature review attempts to combine different approaches for software quality measurement. Software quality measurement is a vast field and it requires a more systematic approach to review its published measurement methods and assessment techniques. Reviewing the published literature focusing only on machine learning techniques for assessing software quality will provide a more focused overview of the reference.

REFERENCES


