*British Journal of Applied Science & Technology*
*4(21): ……., 2014*

**SCIENCEDOMAIN** *international*
www.sciencedomain.org

# Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance

**Ming-Chang Lee[1*]**

*[1]National Kaohsiung University of Applied Sciences (Taiwan), 415 Chien Kung Road, Kaohsiung, Taiwan.*

***Author's contribution***

*This whole work was carried out by authors MCL.*

***Original Research Article***

***Received 31st March 2014***
***Accepted 1st May 2014***
***Published 2nd June 2014***

## ABSTRACT

**Aims:** Software quality assurance is a formal process for evaluating and documenting the quality of the work products during each stage of the software development lifecycle. The practice of applying software metrics to operational factors and to maintain factors is a complex task. Successful software quality assurance is highly dependent on software metrics. It needs linkage the software quality model and software metrics through quality factors in order to offer measure method for software quality assurance. The contributions of this paper build an appropriate method of Software quality metrics application in quality life cycle with software quality assurance.
**Design:** The purpose approach defines some software metrics in the factors and discussed several software quality assurance model and some quality factors measure method.
**Methodology:** This paper solves customer value evaluation problem are: Build a framework of combination of software quality criteria. Describes software metrics. Build Software quality metrics application in quality life cycle with software quality assurance.
**Results:** From the appropriate method of Software quality metrics application in quality life cycle with software quality assurance, each activity in the software life cycle, there is one or more QA quality measure metrics focus on ensuring the quality of the process and the resulting product. Future research is need to extend and improve the methodology to extend metrics that have been validated on one project, using our criteria, valid measures of quality on future software project.

**Corresponding author: E-mail: ming_li@mail2000.com.tw;*

*Keywords: Software quality assurance; software metric; software quality factor; software life cycle.*

## 1. INTRODUCTION

Software quality assurance (SQA) is a technique to help achieve quality. SQA is becoming a critical issue in software development and maintenance [1]. SQA can monitor that the software engineering processes and methods used to ensure quality. Software metric deals with the measurement of software product and software product development process and it guides and evaluates software development [2]. Software metric is quantitative measure of the extent to which a system, component, or process. Software factors are going importance and acceptance in corporate sectors as organizations grow nature and strive to improve enterprise quality. The metrics are the quantitative measures of the degree to which software processes a given attribute that affects its quality. SQA is a formal process for evaluating and documenting the quality of the products produced during each stage of the software development lifecycle.

There have four quality models: McCall's quality model [3], Boehm's quality model [4], FURPS mode; [5], Dromey's quality model [6], and ISO/IEC 25000 standard. Each model contains different quality factors and quality criteria [7]). They are indicators of process and product and are useful in case of software quality assurance [8]. The aim of this paper present what are software quality factors and their criteria and their impact on the SQA function.

The remaining part of this paper is organized as follows section 2 describes literature review, it discuses the content of the relation of quality factors with quality criteria. Section 3 builds a relationship of software quality criteria between metrics. Section 4 describes software metrics, it found in the software engineering. Section 5 builds an appropriate method of Software quality metrics application in quality life cycle with software quality assurance. Fig. 1 is a research framework.

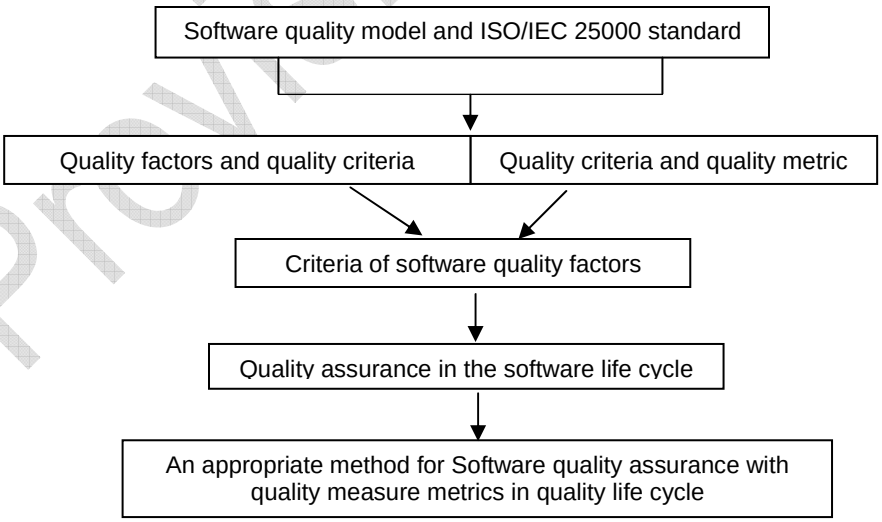


**Fig. 1. A Research framework**

## 2. LITERATURE RESEARCH

### 2.1 Software Quality Assurance Model

In this section, it discusses the contents of the following quality assurance model: McCall quality model, Boehm quality model, FURPS model, and Dromey model.

The McCall quality model [3] has three quality of software product: product transition (adaptability to new environment), product revision (ability to undergo changes), and product operations (its operation characteristics). Product revision includes Maintainability, Flexibility, and Testability. Product Transition includes Portability, Reusability, and Interoperability. This model contains 11 quality factors and 23 quality criteria. The quality factors describe different types of system characteristics and quality criterions are attributes to one or more of the quality factors. Table 1 is denoted as the factors and criteria of McCall quality mode

**Table 1. The factors and criteria of McCall quality mode**

| Category | Software metrics | 11 Quality factors | Quality criteria |
|---|---|---|---|
| McCall's quality | Product Operation | Correctness | Completeness, consistency, operability |
| | | Reliability | Accuracy, complexity, consistency, error tolerance, modularity, simplicity |
| | | Efficiency | Concision, execution, efficiency, operability |
| | | Integrity | Audit ability, instrumentation, security |
| | | Usability | Operability, training |
| | Product Revision | Maintainability | Concision, consistency, modularity, instrumentation, self-documentation, software independence |
| | | Flexibility | Generality, hardware independence, modularity, self-documentation, software independence |
| | | Testability | Audit ability, complexity, instrumentation, modularity, self-documentation, simplicity |
| | Product Transition | Portability | Complexity, concision, consistency, expandability, generality, modularity, self-documentation, simplicity |
| | | Reusability | Generality, hardware independence, modularity, self-documentation, software independence |
| | | Interoperability | Communications commonality, data communality |

Boehm quality model attempts to automatics and qualitatively evaluate the quality of software. The high – level characteristics address three classification; general utility into as utility, maintainability, portability. In the intermediate level characteristics, Boehm quality model have 7 quality factors like portability, reliability, efficiency, Usability, Human engineering, understandability, flexibility [4,9]. Table 2 is denoted as the quality factors and quality criteria of Boehm quality mode.

**Table 2. The factors and criteria of Boehm quality mode**

| Factors | Criteria |
|---|---|
| Portability | Self contentedness, device independence |
| Reliability | Self contentedness, accuracy, completeness, robustness/ integrity, consistency |
| Efficiency | Accountability, device efficiency, accessibility |
| Usability | Completeness |
| Human engineering (testability) | Accountability, communicativeness, self descriptiveness, structuredness |
| Understanding | Consistency, structured, conciseness |
| Modifiability (Flexibility) | Structured, augment ability |

Dromey quality model proposed a framework for evaluate requirement, design and implementation phases. The high-level product properties for the implementation quality model include: correctness, internal, contextual, and descriptive ([6], [10]). Table 3 is denoted as the factors and criteria of Dromey quality mode.

**Table 3. The factors and criteria of Dromey quality mode**

| Factors | Criteria |
|---|---|
| Correctness | Functionality, Reliability |
| Internal | Maintainability, Efficiency, Reliability |
| Contextual | Maintainability, Reusability, portability, reliability |
| Descriptive | Maintainability, Efficiency, reliability, usability |

Furps model originally presented by Grady [5], then it is extended by IBM Rational Software ([11] ~ [12]) into FURPS+. The "+'' indicates such requirements as design constraints, implementation requirements, interface requirements and physical requirements [11]. There are four characteristics in FURPS model. Table 4 is denoted as the factors and criteria of FURPS quality mode.

**Table 4. The quality factors and quality criteria of FURPS quality mode**

| Factors | Criteria |
|---|---|
| Functionality | Capabilities, and security |
| Usability | Consistency, user documentation, training materials |
| Reliability | Frequency and security of failure, Recoverability, predictability, accuracy, mean time between failure |
| Performance | Speed efficiency, availability, accuracy, throughput, response time, recovery time, resource usage |
| Supportability | Testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, install ability, localizability |

ISO 9000 – it provides guidelines for quality assurance [8]. ISO 9000 is a process oriented approach towards quality management [13]. It processes designing, documenting, implementing, supporting, monitoring, controlling and improving [14]. Recently, the ISO/IEC 9126-1: 2001 software product quality model, which defined six quality characteristics, has replaced by ISO/IEC 205010:2011 system and software product quality model [15]. ISO 25010 is the most commonly used quality standard model. It contains eight quality factors:

Functional suitability, reliability, operability, security, performance efficiency, compatibility, maintainability, and portability. The 28 quality factors are arranged in six quality characteristics. Table 5 is denoted as the factors and criteria of ISO/IEC 2510 quality mode ([15) ~ [16]).

**Table 5. The factors and criteria of ISO/IEC 25010 quality mode**

| Factors | Criteria |
|---|---|
| Functional suitability | Functional appropriateness, accuracy |
| Performance efficiency | Time behavior, resource utilization |
| Reliability | Maturity, fault tolerance, recoverability, Availability |
| Operability | Appropriateness reconcilability, Ease of use, User error protection, User interface aesthetics, Technical learn ability, technical accessibility |
| Security | Confidentiality integrity, Non-repudiation, Accountability, Authenticity |
| Compatibility | Co-existence, Interoperability |
| Maintainability | Modularity, Reusability, Analyzability, Modifiability, testability, |
| Portability | Adaptability, install-ability, replace-ability |

The quality models described above contain several factors in common, like Maintainability, Efficiency, and Reliability. However, some of factors like correctness, understandability, modifiability and supportability are not so common and are in one or two models. Table 5 compared of fours quality model and ISO/IEC 25010. Table 6 is denoted as a comparison of the factors of four quality model and ISO/IEC 25010.

**Table 6. A comparison of criteria of the four quality model and ISO/IEC 2510**

| factors | McCall | Boehm | Dromey | FURPS | ISO/IEC25010 |
|---|---|---|---|---|---|
| Correctness | * | | | | |
| Integrity | * | | | | |
| Usability | * | | | * | * |
| Efficiency | * | * | * | | * |
| Flexibility | * | * | | | |
| Testability | * | | | | * |
| Maintainability | * | | | | * |
| Reliability | * | * | * | * | * |
| Portability | * | * | * | | * |
| Reusability | * | | * | | |
| Interoperability | * | | * | | |
| Human engineering | | * | * | | |
| Understandability | | * | | | * |
| Modifiability | | * | | | * |
| Functionality | | | | * | * |
| Performance | | | * | * | |
| Supportability | | | | * | |
| Security | | | | | * |
| 17 | 11 | 7 | 7 | 5 | 8 |

*Extended Al-Outaish [17]*

## 3. COMBINATION OF SOFTWARE QUALITY CRITERIA AND SOFTWARE METRICS

Under the software quality assume model and ISO/IEC 25000 standard, it found the combination of software quality factors and criteria. In this section, it need describe the relationship of software quality criteria and software quality metrics.

There are four reasons for developing a list of criteria for each factor:

1. Criteria offer a more complete, concrete definition of factors.
2. Criteria common among factors help to illustrate the interrelation between factors.
3. Criteria allow audit and review metrics to be developed with greater ease.
4. Criteria allow us to pinpoint that area of quality factors which may not be up to a predefined acceptable standard.

### 3.1 Software Quality Factors and Quality Criteria

Criteria are the characteristics which define the quality factors. The criteria for the factors are the attributes of the software product or software production process by which the factor can be judged or definition. The relationships between the factors between the criteria can be found in Table 7.

**Table 7. The relationships of factors with criteria of software quality**

| Factors | Criteria |
|---|---|
| Correctness | Completeness, consistency, operability |
| Efficiency | Concision, execution, efficiency, operability |
| Flexibility | Complexity, concision, consistency, expandability, generality, modularity, self-documentation, simplicity |
| Integrity | Audit ability, instrumentation, security |
| Interoperability | Communications commonality, data communality |
| Maintainability | Concision, consistency, modularity, instrumentation, self-documentation, software independence |
| Portability | Generality, hardware independence, modularity, self-documentation, software independence |
| Reliability | Accuracy, complexity, consistency, error tolerance, modularity, simplicity |
| Reusability | Generality, hardware independence, modularity, self-documentation, software independence |
| Testability | Audit ability, complexity, instrumentation, modularity, self-documentation, simplicity |
| Usability | Operability, training |
| Modifiability | Structure, augment ability, |
| Understandability | Consistency, Structure, conciseness. legibility |
| Documentation | Completeness |
| Functionality | Capability, security |
| Performance | Flexibility, efficiency, Reusability |
| Supportability | Testability, extensibility, maintainability, compatibility |

## 3.2 Quality Criteria and Related Factors

Table 8 is criteria for software quality factors. It provides an illustration of the relationship between these criteria and the factors [3].

**Table 8. Criteria of software quality factors**

| Criterion | Definition | Related factors |
|---|---|---|
| Traceability | Those attributes of the software that provide a thread from the requirements to the implementation with respected to the specific development and operational environment. | Correctness |
| Completeness | Those attributes of the software that provide full implementation of the function required | Correctness |
| Consistency | Those attributes of the software that provide uniform design and implementation techniques and notation. | Correctness<br>Reliability<br>Maintainability |
| Accuracy | Those attributes of the software that provide the required precision in calculation and outputs. | Reliability |
| Error Tolerance | Those attributes of the software that provide continuity of operation under monomial conditions. | Reliability |
| Simplicity | Those attributes of the software that provide implementation of functions in the most understandable manner. (usually avoidance of practices which increase complexity) | Reliability<br>Maintainability<br>Testability |
| Modularity | Those attributes of the software that provide a structure of highly independent modules | Maintainability<br>Flexibility<br>Testability<br>Portability<br>Reusability<br>Interoperability |
| Generality | Those attributes of the software that provide breadth to the functions performed | Flexibility<br>Reusability |
| Expandability | Those attributes of the software that provide for expansion of data storage requirements or computational functions. | Flexibility |
| Instrumentation | Those attributes of the software that provide for the measurement of usage identification of errors. | Testability |
| Self-Descriptiveness | Those attributes of the software that provide explanation of the implementation of function. | Flexibility<br>Testability<br>Portability<br>Reusability |
| Execution Efficiency | Those attributes of the software that provide for minimum processing time. | Efficiency |
| Storage Efficiency | Those attributes of the software that provide for minimum storage requirements during operation. | Efficiency |

***Table 8 Continued…***

| | | |
|---|---|---|
| Access Control | Those attributes of the software that provide for control of the access of software and data | Integrity |
| Access Audit | Those attributes of the software that provide for audit of the access of software and data | Integrity |
| Operability | Those attributes of the software that determine operation and procedure concerned with the operation of the software | Usability |
| Training | Those attributes of the software that provide transition from current operation or initial familiarization | Usability |
| Communicativeness | Those attributes of the software that provide useful inputs and outputs which can be assimilated | Usability |
| Software System Independence | Those attributes of the software that determine its dependency on the software environment (operating systems, utilities, input/output routines, etc.) | Portability<br>Reusability |
| Machine independence | Those attributes of the software that determine its dependency on the hardware system. | Portability<br>Reusability |
| Communications Commonality | Those attributes of the software that provide the use of standard protocols and interface routines | Interoperability |
| Data Commonality | Those attributes of the software that provide the use of standard data representations. | Interoperability |
| Conciseness | Those attributes of the software that provide for implementation of a function with minimum amount of code. | Maintainability |

## 3.3 Criteria of Software Quality Factors

The following table lists all software metrics. They copied from volume II of the specification of software quality attributes software quality evaluation guidebook [18]. Table 9 is denoted as the relationship of criteria between software quality metrics.

## 3.4 Quality Assurance in the Software Life Cycle

Product metrics, process metrics, and project metrics are three important types of software metrics. Product metrics measures the efficiency of accomplishing product targets for instance size, complexity, design features, performance, and quality level. Process metrics measures the efficiency of performance the product development process for instance turnover rate. Project metrics measures the efficiency of product development process, for instance schedule performance, cost performance, team performance [19].

In order to be efficient, quality assurance activities should following stage in the software life cycle. For each activity in the software life cycle, there is one or more QA support activities focus on ensuring the quality of the process and the resulting product. A concept framework of QA support software quality life cycle as shown in Fig. 2.

**Table 9. The relationship of criteria between software quality metrics**

| Criteria | Software quality metrics |
|---|---|
| Accuracy | Accuracy checklist |
| Self-Descriptiveness | Quality of comments<br>Effectiveness of comments<br>Descriptiveness of language |
| Simplicity | Design structure<br>Structured language<br>Data and control flow complexity<br>Coding simplicity<br>Halstead's level of difficulty measure |
| System accessibility | Access control<br>Access audit |
| System clarity | Interface complexity<br>Program flow complexity<br>Application functional complexity<br>Communication complexity<br>Structure clarity |
| System compatibility | Communication compatibility<br>Data compatibility<br>Hardware compatibility<br>Software compatibility |
| Traceability | Documentation for other system<br>Cross reference |
| Document accessibility | Access to documentation<br>Well structured documentation |
| Efficiency process | Processing effectiveness measure<br>Data usage effectiveness measure |
| Efficiency communication | Communication effectiveness measure |
| Efficiency storage | Storage effectiveness measure |
| Functional | Function specifically<br>Function commonality<br>Function selective usability |
| Generality | Unit referencing<br>Unit implementation |
| Independence | Software independence for system<br>Machine independence |
| Modularity | Modular design |
| Operability | Operability checklist<br>User output communicativeness<br>User input communicativeness |
| Training | Training checklist |
| Virtual | System/data independence |
| Visibility | Unit testing Integration testing<br>Case testing |
| Application independence | Database management<br>Database implementation<br>Database independence<br>Data structure<br>Architecture standardization<br>Microcode independence<br>Function independence |

| | |
|---|---|
| Augment ability | Data storage expansion<br>Computation extensibility<br>Channel extensibility<br>Design extensibility |
| Completeness | Completeness checklist |
| Consistency | Procedure consistency<br>Data consistency |
| Autonomy | Interface complexity<br>Self- sufficiency |
| Re-configurability | Restructure checklist |
| Anomaly management | Error tolerance Improper input data<br>Communications faults Hardware faults<br>Device error Computation failures |

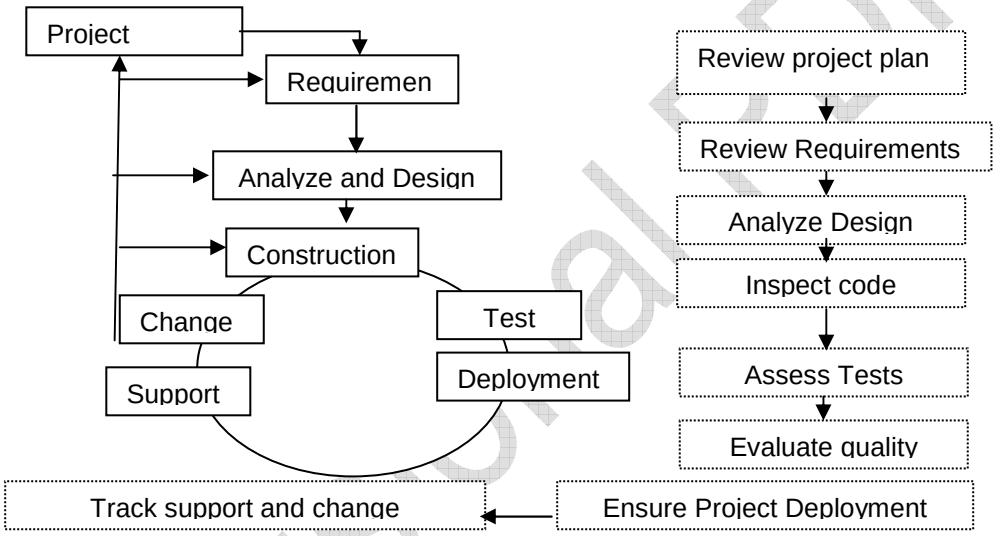


**Fig. 2. A concept framework of QA support software quality life cycle**

## 4. SOFTWARE QUALITY METRICS

This section concentrates on different metrics found in the software engineering literature. A classical classification of the software quality metrics: Halstead's software metrics, McCabe's cyclomatic complexity metric, RADC's methodology, Albrecht's function points metric, Ejiogu's software metrics, and Henry and Kafura's information metric.

### 4.1 Halstead's Software Metrics

Halstead's measure for calculation of module conciseness is essentially based on the assumption that a well structured program is a function of only its unique operators and operands. The best predictor of time required to develop and run the program successfully was Halstead's metric for program volume.

Halstead [20] defined the following formulas of software characterization for instance.

The measure of vocabulary: $n = n_1 + n_2$

Program length: $N = N_1 + N_2$

Program volume: $V = N \log_2 n$

Program level: $L = \frac{V^*}{V}$

Where

$n_1$ = the number of unique operators
$n_2$ = the number of unique operand
$N_1$ = the total number of operators
$N_2$ = the total number of operands

Christensen et al. [21] have taken the idea further and produced a metrics called difficulty. $V^*$ is the minimal program volume assuming the minimal set of operands and operators for the implementation of given algorithm:

Program effort: E = $\frac{V^*}{L}$

Difficulty of implementation: D = $\frac{n_1 N_2}{2n_2}$

Programming time in seconds: T = $\frac{E}{S}$

Difficulty: $n_1/2 + N_2/n_2$

With S as the Stroud number ($5 \le S \le 20$) which is introduced from the psychological science. $n_2^*$ is the minimal set of operands. $E_0$ is determined from programmer's previous work. The based on difficulty and volume Halstead proposed an estimator for actual programming effect, namely.

Effort = difficulty * volume

Table 10 is denoted as the formulas of Halstead's software metrics with software quality factors.

### 4.2 McCabe's Cyclomatic complexity metrics

McCable [22] has proposed a complexity metric on mathematical graph theory. The complexity of a program is defined in terms of its control structure and is represented by the maximum number of "linearly independent" path through the program. Software developer can use this measure to determine which modules of a program are over-complex and need to be re-coded.

**Table 10. The formulas of Halstead's software metrics with software quality factors**

| Software metrics | Software quality factors | Formulas |
|---|---|---|
| Implementation length N | Maintainability<br>Number of Bugs<br>Modularity<br>Performance<br>Reliability | $N = N_1 + N_2$<br>$= n_1 \log_2 n_1 + n_2 \log_2 n_2)$ |
| Volume V | Complexity<br>Maintainability<br>Number of Bugs<br>Reliability<br>Simplicity | $V = N \log_2 n$ |
| Potential Volume $V^*$ | Conciseness Efficiency | $V^* = (n_1 + n_2^*)\log_2(n_1 + n_2^*)$ |
| Program Level L | Conciseness Simplicity | $L = \frac{V^*}{V}$ |
| Program Effort | Clarity Complexity<br>Maintainability Modifiability<br>Modularity Number of Bugs<br>Performance Reliability<br>Simplicity Understandability | $E = \frac{V^*}{L}$ |
| Number of Bugs | Maintainability Number of BugsTestability. | $B = \frac{V}{E_0}$ |

The formulas for the cyclomatic complexity proposed by [3] are:

$$V(G) = e - n + 2p$$

Where

e = the number of edges in the graph
n = the number of nodes in the graph
P = the number of connected components in the graph.

The Cyclomatic complexity metric is based on the number of decision elements (IF-THEN-ELSE, DO WHILE, DO UNTIL, CASE) in the language and the number of AND, OR, and NOT phrases in each decision. The formula of the metric is: Cyclomatic complexity = number of decisions +number of conditions + 1[23].

The Essential complexity metricb is based on the amount of unstructured code in a program. Modules containing unstructured code may be more difficult to understand and maintain. The essential complexity proposed by McCable [22]:

$$EV(G) = V(G) - m$$

Where

V (G) = the cyclomatic complexity
m = the number of proper sub graphs

McCabe's Cyclomatic complexity measure has been correlated with several quality factors. These relationships are listed in Table 11.

**Table 11. The formulas of McCabe's Cyclomatic complexity metrics**

| Software metrics | Software quality factors | Formulas |
|---|---|---|
| Cyclomatic complexity V(G) | Complexity<br>Maintainability<br>Number of Bugs<br>Modularity<br>Simplicity<br>Reliability<br>Testability<br>Understandability | V(G) = e - n + 2p |
| Essential Complexity EV(G) | Complexity<br>Conciseness<br>Efficiency<br>Simplicity | EV (G)= V(G) - m |

## 4.3 RADC's Methodology

RADS expanded Boehm model. The metrics discussed in this section are based on continuing development effort [18]. The requirements present a ratio of actual occurrence to the possible number of occurrence for each situation: these results in a clear correlation between the quality criteria and their associated factors. Table 12 is denoted as the formulas of RADC's methodology.

**Table 12. The formulas of RADC's methodology**

| Software metrics | Software quality factors | Formulas (for example) |
|---|---|---|
| Traceability Completeness Consistency Accuracy Error Tolerance Simplicity Structures programming Modularity Generality Expandability Computation extensibility Instrumentation Self-Descriptiveness Execution efficiency Storage Efficiency Access control Access Audit Operability Training Communicativeness Software system independence Machine independence Communication commonality Data commonality Conciseness | Completeness<br>Consistency<br>Correctness<br>Efficiency<br>Expandability<br>Flexibility<br>Integrity<br>Interoperability<br>Maintainability<br>Modularity<br>Portability<br>Reliability<br>Survivability<br>Usability<br>Verifiability | **Traceability** (1)<br>Cross reference relative modules to requirements<br>**Completeness** (2)<br>1. Unambiguous references (Input, function, output)<br>2. All external data references defined, computed or obtained from external source<br>3. All detailed functions defined<br>4. All conditions and processing defined for each decision point<br>5. All defined and reference calling sequence parameters agree<br>6. All problem reports resolved<br>7. Design agree with requirements<br>8. Code agree with design |

*Source: Bowen et al. [18]*

$$^{1}\ Traceability = (Number\ of\ itemized\ requirements\ traced / total\ number\ of\ requirements)$$

$$^{2}\ Completeness = \left(\frac{1}{9}\sum_{1}^{9} score\ for\ element\ i\right)$$

### 4.4 Albrecht's Function Points Metrics

Albrecht developed a metric to determine the number of elementary functions, hence the value of source code. This metric was developed to estimates the amount the effort needed to design and develop customer applications software [24].

1. Calculation the function counts (FCs) based on the following formula:

$$FC = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} \times x_{ij}$$

Where $w_{ij}$ are the weighting factors of the five components by complexity level (low, average, high) and $x_{ij}$ are the numbers of each component in the application.

It is a weighted of five major components [25] are:

- External input: Low complexity, 3; average complexity, 4; high complexity, 6
- External output: Low complexity, 4; average complexity, 5; high complexity, 7
- Logical internal file: Low complexity, 5; average complexity, 7; high complexity, 10
- External interface file: Low complexity, 7; average complexity, 10; high complexity, 15
- External inquiry: Low complexity, 3; average complexity, 4; high complexity, 6

2. Calculation the value adjustment factor, it involves a scale from 0 to 5 to assess the impact of 14 general system characteristics in terms of their likely on the application. There are 14 characteristics: data communication distributed function, heavily used configuration, transaction rate, online data entry, end user efficiency, online update, complex processing, reusability, installation ease, operational ease, multiple sites, and facilitation of change.

   The scores (ranging from 0 to 5) for these characteristics are then summed, based on the following formulas, to arrive at the value adjustment factor (VAF)

$$VAF = 0.65 + 0.01 \sum_{i=1}^{14} c_i$$

   $c_i$ : the score of general system characteristics.

3. The number of function points is obtained by multiplying function counts and the value adjustment factor:

$$FP = FC \times VAF$$

### 4.5 Ejiogu's Software Metrics

Ejiogu's software metrics uses language constructs to determine the structural complexity of a program. The syntactical constructs are nodes. These metrics are related to the structural complexity of a program. They are also related to other quality factors, such as usability, readability, and modifiability.

The structural complexity metrics gives a numerical notion of the distribution and connectedness of a system's components [26].

$$S_C = H \times R_t \times M$$

Where

H: the height of the deepest nested node,
$R_t$: the Twin number of the root,
M: the Monadicity [27]

The height for an individual node is the number of levels that a node is nested below the root node. The Twin number is the number of nodes that branch out from a higher level node. Monads are nodes that do not have branches emanating from them. They also are referred to as "leaf nodes".

Software size is the size of a set of nodes of source code. It is calculated using the number of modes in the tree.

$$S = total\ umber\ of\ nodes - 1$$

Where

1: represents the root node.

## 4.6 Henry and Kafura's Information Metrics

Information flow complexity (IFC) [28] describes the amount of information which flows into and out of a procedure. This metrics use the flow between procedures to dhow the data flow complexity of a program. The Formula is:

$$IFC = Length \times (fan - in * fan - out)^2$$

Where

Fan-in: The number of local flows into a procedure plus the number of global data structures from which a procedure retrieves information.

Fan-out: The number of local flows into a procedure plus the number of global data structures from which a procedure updates.

Length is the number of lines of source code in the procedure. In implementing this count, embedded comments are also counted, but not comments preceding the beginning of the executable code.

## 4.7 Project Metrics

PMI PMBOK (Project management institute's project management body of knowledge) describes Project Management Processes, tools and techniques and provides one set of high level businesses for all industries. The PMBOK includes all nine knowledge areas and all associated with them tools and techniques: Integration management, Scope management, Time management, Cost management, Quality management, Human Resource management, Communication Management, Risk management, and Procurement management (PMBOK [29]).

Some of those processes often are not applicable or even irrelevant to Software Development industry. CMM (Capability Maturity model) speaks about software project planning processes without mentioning specific methodologies for project estimating described in PMBOK (PMBOK). Basic key process areas (KPA) of the SEI CMM is requirement management, project planning, project tracking and oversight, subcontract management, quality assurance, and configuration management. The Table 13 is mapping of some relevant to CMM activities, tools and techniques:

**Table 13. Mapping of project management processes to process groups and knowledge areas**

| Knowledge Areas / Process Group | Initiating | Planning | Executing | Controlling | Closing |
|---|---|---|---|---|---|
| Project integration management | | Project plan development | Project plan execution | Integrated change control | |
| Project scope Management | Initiation scope definition | Scope planning | Scope change control | Scope verification | |
| Project tine management | | Activity definition<br>Activity sequencing<br>Activity duration estimating<br>Schedule development | | Schedule control | |
| Project cost management | | Resource planning<br>Cost estimating<br>Cost budgeting | | Cost control | |
| Project quality management | | Quality planning | Quality assurance | Quality control | |
| Project Human resource management | | Organization planning<br>Staff Acquisition | Team development | | |
| Project communication management | | Communication planning | Information distribution | Performance reporting | Administrative closure |
| Risk project management | | Risk management planning<br>Risk identification<br>Qualitative risk analysis<br>Risk response planning | | Risk monitoring and control | |
| Project procurement management | | Procurement planning<br>Solicitation planning | Solicitation<br>Source selection<br>Contract administration | | Contract closeout |

## 4.8 Reliability Metrics

A varies often used measure of reliability and availability in computer-based system is mean time between failures (MTBF) [30]. The sum of mean time to failure (MTTF) and mean time to repair (MTTR) gives the measure, i.e.

$$MTBF = MTTF + MTTR$$

The availability measure of software is the percentage that a program is operating according to requirement at a given time and is given by the formula:

$$Availability = MTTP / (MTTF + MTTE) * 100\%$$

The reliability growth models assume in general that all defects during the development and testing phases are correct, and new errors are not introduced during theses phases. All models seem to include some constraints on the distribution of defects or the hazard rate, i.e. defect remaining in the system.

Increase software reliability gives the metrics:

$$\text{Failure rate (FR)} = \frac{Number\ \ of\ \ failures}{Execution\ \ time}$$

## 4.9 Readability metrics

Walston and Felix [31] defined a ratio of document pages to LOC as:

$$D = 49L^{1.01}$$

Where

D= number of pages of document
L = number of 1000 lines of code.

## 4.10 Metrics-Based Estimation Models

### 4.10.1 COCOMO Model

Most of the models presented in this subsection are estimators of the effort needed to produce a software product. Probably the best known estimation model is Boehm's COCOMO model [32]. The first one is a basic model which is a single-value model that computes software development effort and cost as a function of program size expressed as estimated lines of code (LOC). The second COCOMO model computes software development effort as a function of program size and a set of "coat drives" that include subjective assessment of product, hardware, personal, and project attributes.
The basic COCOMO equations are:

$$E = a_i (KLOC)^{b_i}\ ,\ D = c_i E^{d_i}$$

Where

E is the effort applied in person-month.
D is the development time in chronological months

The coefficients $a_i$ and $c_i$ and the exponents $b_i$ and $d_i$ are given in Table 14.

**Table 14. Basic COCOMO**

| Software project | $a_i$ | $b_i$ | $c_i$ | $d_i$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.36 |
| Semi-detached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

The second COCOMO has some special features, which distinguish it from other ones. The usage of this method is very wide and its results are accurate. The equations are use to estimate effort and schedule see [33].

### 4.10.2 Putnam estimation model

The Putnam estimation model ([34] ~ [35]) assumes a specific distribution of effort over the software development project. The distribution of effort can be described by the Royleigh-Norden curve. The equation is:

$$L = c_k K^{1/3} t_d 4/3$$

Where

$c_k$ is the state of technology constant (the environment indictor),
$k$ is the effort expended (in person-years) over the whole life cycle.
$t_d$ is the development time in year.

The $c_k$ valued ranging from 2000 for poor to 11000 for an excellent environment is used [36].

### 4.10.3 Source line of code

SLOC is an estimation parameter that illustrates the number of all comments and data definition but it does not include instructions such as comments, blanks, and continuation lines. Since SLOC is computed based on language instructions, comparing the size of software which uses different language is too hard. SLOC usually is computed by considering $S_L$ as the lowest, $S_H$ as the highest and $S_M$ as the most probable size [37].

$$S = \frac{S_L + 4S_M + S_H}{6}$$

### 4.10.4 Productive estimation model

Walston and Fellix [31] give a productivity estimator of a similar form at their document metric. The programming productivity is defined as the ratio of the delivered source lines of code to the total effort in person-months required to produce the delivered product.

$$E = 5.2L^{0.91}\ \text{E}$$

Where

E is total effort in person-month
L is the number of 1000lines of code.

## 4.11 Metrics for software maintenance

During the maintenance phase, the following metrics are very important: [37]

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

Fix backlog is a workload statement for software maintenance. To manage the backlog of open, unresolved, problems is the backlog management index (BMI). If BMI is large then 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.

$$BMI = \frac{Number\ of\ \ problrms\ closed\ during\ the\ month}{Number\ of\ \ problem\ arrivals\ during\ the\ month} \times 100\%$$

## 4.12 Customer Problem Metrics

The customer problems metric can be regarded as an intermediate measurement between defects measure and customer satisfaction. The problems metric is usually expressed in terms of problem per user month (PUM). PUM is usually calculated for each month after the software is released to market, and also for monthly averages by user.

Several metrics with slight variations can be Constructed and used, depending on the purpose of analysis. For example: ([38] ~ [40]).

- Percent of completely satisfied customers.
- Percent of satisfied customers (satisfied and completely satisfied)
- Percent of dissatisfied customers(dissatisfied and completely dissatisfied)
- Percent of non (neutral, dissatisfied, and completely dissatisfied).
- Customer – founded defects (CFD) total

$$CFD\ total = \frac{Number\ of\ \ customer - \ founded\ defects}{Assembly - equivalent\ total\ \ source\ size}$$

- Customer – founded defects (CFD) delta

$$CFD\ total = \frac{Number\ of\ \ customer - \ founded\ defects\ caused\ by\ increasemental\ software\ development}{Assembly - equivalent\ total\ \ source\ size}$$

PUM = Total problems that customers reported (true defects and non-defects- orients problems) for a time period + Total number of License- months of the software during the period.

Where Number of license- months = Number of install license of the software× Number of months in the calculation period.

## 4.13 Test Product and Process Metrics

Test process metrics provide information about preparation for testing, test execution and test progress. Some testing metrics ([36,41,42]) as following:

1. Number of test cases designed
2. Number of test cases executed
3. DA = Number of defects rejected / Total number of defects *100%
4. Bad Fix Defect =Number of Bad Fix Defect / Total number of valid defects *100%
5. Test case defect density = (Number of failed tests / Number of executed test cases) *100
6. Total actual execution time/ total estimated execution time
7. Average execution time of a test case

Test product metrics provide information about the test state and testing status of a software product. Using these metrics we can measure the products test state and indicative level quality, useful for product release decision [42].

1. Test Efficiency = $(D_T/(D_T+D_U)*100$
2. Test Effectiveness = $(D_T/(D_F+D_U)*100$
3. Test improvement TI = number of defects detected by the test team during / source lines of code in thousands
4. Test time over development time TD = number of business days used for product testing / number of business days used for product
5. Test cost normalized to product size (TCS) = total cost of testing the product in dollars / source lines of code in thousands
6. Test cost as a ration of development cost (TCD) = total cost of testing the product in dollars / total cost of developing the product in dollars
7. Test improvement in product quality = Number of defects found in the product after release / source lines of code in thousands
8. Cost per defect unit = Total cost of a specific test phase in dollars / number of defects found in the product after release
9. Test effectiveness for driving out defects in each test phase = $(D_D/(D_D+D_N)*100$
10. Performance test efficiency (PTE) = requirement during perform test / (requirement during performance time + requirement after signoff of performance time) * 100%
11. Cost per defect unit = Total cost of a specific test phase in dollars / number of defects found in the product after release
12. Estimated time for testing
13. Actual testing time
14. % of time spent = (actual time spent / Estimating time)*100

Where

$D_D$ : Number of defects of this defect type that are detected after the test phase.

$D_T$ : Number of defects found by the test team during the product cycle
$DU$ : Number of defects of found in the product under test (before official release)
$D_F$ : Number of defects found in the product after release the test phase
$D_N$ : Number of defects of this defect type (any particular type) that remain uncovered after the test phase.

## 4.14 Method of Statistical Analysis

The revisions on the software measurement methods, developed with the purpose of improving their consistency must be empirically evaluated so as to determine to what extent is the pursued goal fulfilled. The most used statistical methods are given in the following table ([43] ~ [48]). Some commonly used statistical methodology (include nonparametric tests) are discussed as follow:

1. Ordinary least square regression models: Ordinary least square regression (OLS) model is used to subsystem defects or defect densities prediction
2. Poisson models: Poisson analysis applied to library unit aggregation defect analysis
3. Binomial analysis: Calculation the probability of defect injection
4. Ordered response models: Defect proneness
5. Proportional hazards models: Failure analysis incorporating software characteristics
6. Factor analysis: Evaluation of design languages based on code measurement
7. Bayesian networks: Analysis of the relationship between defects detecting during test and residual defects delivered
8. Spearman rank correlation coefficient: Spearman's coefficient can be used when both dependent (outcome; response) variable and independent (predictor) variable are ordinal numeric, or when one variable is an ordinal numeric and the other is a continuous variable.
9. Pearson or multiple correlations: Pearson correlation is widely used in statistics to measure the degree of the relationship between linear related variables. For the Pearson correlation, both variables should be normally distributed
10. Mann – Whitney U test: Mann – Whitney U test is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other
11. Wald-Wolfowitz two-sample Run test: Wald-Wolfowitz two-sample Run test is used to examine whether two samples come from populations having same distribution.
12. Median test for two samples: To test whether or not two samples come from same population, median test is used. It is more efficient than run test each sample should be size 10 at least.
13. Sign test for match pairs: When one member of the pair is associated with the treatment A and the other with treatment B, sign test has wide applicability.
14. Run test for randomness: Run test is used for examining whether or not a set of observations constitutes a random sample from an infinite population. Test of randomness is of major importance because the assumption of randomness underlies statistical inference.
15. Wilcoxon signed rank test for matcher pairs: Where there is some kind of pairing between observations in two samples, ordinary two sample tests are not appropriate.
16. Kolmogorov-Smirnov test: Where there is unequal number of observations in two samples, Kolmogorov-Smirnov test is appropriate. This test is used to test whether there is any significant difference between two treatments A and B.

## 5. SOFTWARE QUALITY METRICS IN QUALITY LIFE CYCLE WITH SOFTWARE QUALITY ASSURANCE

Software quality metrics focus on quality aspects of product metrics, process metrics, maintenance metrics, customer metrics and project metrics.

Product metrics are measures of the software product at any stage of its development, from requirements to installed system. Product metrics may measure the complexity of the software design, the size of the final program, or the number of pages of documentation production. Process metrics are measure of the software development process, such as overall development time, the average level of experience of the programming staff, or type of methodology used. The test process metrics provide information about preparation for testing, test execution and test progress. Some test product metrics are number of test cases design, % of test cases execution, or % test cases failed. Test product metrics provide information of about the test state and testing status of a software product and are generated by execution and code fixes or deferment. Some rest product metrics are Estimated time for testing, average time interval between failures, or time remaining to complete the testing.

The software maintenance phases the defect arrivals by time interval and customer problem calls. The following metrics are therefore very important: Fix backlog and backlog management index, fix response time and fix responsiveness, percent delinquent fixes, and fix quality.

Subjective metrics may measure different values for a given metric, since their subjective judgment is involved in arriving at the measured value. An example of a subjective product metric is a classification of the software as “organic”, ”semi-detached” or “embedded” as required in the COCOMO cost estimation model.

From the customer’s perspective, it is bad enough to encounter functional defects when running a business on the software. The problems metric is usually expressed in terms of problem per user month (PUM). PUM is usually calculated for each month after the software is released to market, and also for monthly averages by user.

The customer problems metric can be regarded as an intermediate measurement between defects measure and customer satisfaction. To reduce customer problems, one has to reduce the functional defects in the products, and improve other factors (usability, documentation, problem rediscovery, etc.). Table 15 is denoted as software quality assurance with quality measure metrics in quality life cycle.

**Table 15. Software quality assurance with quality measure metrics in quality life cycle**

| Category | Description | Project perspective | Software quality factors | Software quality measure metric |
|---|---|---|---|---|
| Project metrics | Describe the project's characteristics and execution | | Resource allocation<br>Review effectiveness schedule performance, cost performance, team performance | ∘Product estimation model<br>∘Project metrics<br>∘Software process timetable metrics |
| Requirements gathering | Examine Requirements | | Completeness Correctness Testability | ∘Requirement specification. |
| Product metrics | Describe the characteristics of the product | Product Operation | Correctness Reliability Efficiency Integrity Usability | □Productivity metrics<br>□Execution Efficiency |
| | | Product Revision | Maintainability Flexibility Testability | □Software system independence.<br>□Machine independence. |
| | | Product Transition | Portability Reusability Interoperability | □Software system independence. |
| Process metrics | Describe the effectiveness and quality of the process that produce the software product | Requirements | Understandability Volatility Traceability Model clarity | □Function points metrics<br>□Requirement specification |
| | | Analysis and design | Structure Component Completeness Interface complexity Patterns Reliability | □Complexity metrics<br>□Structural design<br>□Kafurd's information flow<br>□MTBF |
| | | Code | Complexity Maintainability Understandability Reusability Documentation | □Halstead' measure<br>□Cyclometric measure<br>□Structural programming<br>□Ejiogu's metrics<br>□Error remove effectiveness |
| | | Testing | Correctness Test effectiveness | □Test efficiency<br>□Test process metrics<br>□Test product metrics<br>□Error rate |

| | | | | |
|---|---|---|---|---|
| | | Implementation | Resource usage<br>Completion rates<br>Reliability | Reliability<br>Software corrective maintenance productivity<br>Process quality metrics |
| Ensure Project Deployment | Describe the customer satisfaction metrics | | Usability, Documentation, Problem rediscover | Customer problem metrics<br>Failure density metrics<br>Productivity metrics<br>Effectiveness metrics |
| Track support and change Management | Describe the maintenance metrics | Changes | Correctness<br>Documentation | Defect remove<br>Backlog management index. |
| | | Support | Completion rates<br>Maintainability | Fix backlog<br>Software maturity index<br>Statistical metrics<br>Readability metrics |

*Source: this study*

## 6. CONCLUSION

Software quality metrics focus on quality aspects of product, process, and project. They group into six categories in accordance with the software life cycle: project metrics, requirements gathering, product metrics, process metrics, ensure Project deployment (customer satisfaction metrics), track support and change management (maintenance metrics). In order to understand the relationship of criteria of software quality factor, we have discussed software quality model and standard, quality factors and quality criteria, quality criteria and quality metric. We detail discussed software quality metrics. It includes Halstead's software metrics, McCabe's Cyclomatic complexity metrics, RADC's methodology, Albrecht's function points metric, Ejiogu's software metrics, Henry and Kafura's information metric, project metric, Reliability metrics, Readability metrics, Metrics-based estimation models, Metrics for software maintenance, In- process quality metrics, Customer problem metrics, Test product and process metrics, and Method of statistical analysis. Under the above 15 software quality metrics, we give table of software quality assurance with quality measure metrics in quality life cycle. It contains software quality factors and software quality measure metric in each software development phase.

In order to continue to improve its software product, processes, and customer services. Future research is need to extend and improve the methodology to extend metrics that have been validated on one project, using our criteria, valid measures of quality on future software project.

## COMPETING INTERESTS

Author declares that there are no competing interests.

## REFERENCES

1. Vennila G, Anitha P, Karthik R, Krishnamoorthy P. A study of evaluation information to determine the software quality assurance, International Journal of Research and Reviews in Software Engineering. 2011;1(1):1-8.
2. Ma Y, He K, Du D, Liu J, Yan Y. A complexity metrics set for large-scale object-oriented software system, In proceedings of the Sixth IEEE International Conference on Computer and Information Technology, Washington, DC, USA. 2006;189-189
3. McCall JA, Richards PK, Walters GF. Factors in software quality, RADC TR-77-369: 1977. (Rome: Rome Air Development Center)
4. Boehm BW, Brow JR, Lipow M, McLeod G, Merritt M. Characteristics of software quality. North Holland Publishing. Amsterdam, the Netherlands; 1978.
5. Grady, RB. Practical software metrics for project management and process improvement, Prentice Hall; 1992.
6. Dromey RG. Concerning the Chimera (software quality). IEEE Software. 1996;1:33-43.
7. Drown DJ, Khoshgoftaar TM, Seiya N. Evaluation any sampling and software quality model of high assurance systems, IEEE Transaction on systems, Mean and Cybernetics, Part A: Systems and Human. 2009;39(5):1097-1107.
8. Tomar AB, Thakare VM. A systematic study of software quality models, International Journal of software engineering & application. 2011;12(4):61-70.

9. Boehm BW, Brown JR, Lipow M. Quantitative evaluation of software quality, In Proceeding of the 2nd International Conference on Software engineering. 1976;592-605.
10. Dromey RG. A model for software product quality. IEEE Transaction on Software Engineering. 1995;21:146-162.
11. Jacobson I, Booch G, Rumbaugh J. The unified software development process, Addison Wesley; 1999.
12. Krruchtem P. The rational unified process: an introduction, Addison Wesley; 2000.
13. ISO 9001:2005, Quality management system Fundamentals and vocabulary; 2005.
14. ISO 9001:2001, Quality management system Requirements; 2001.
15. ISO /IEC25010: Software engineering– system and software quality requirement and evaluation (SQuaRE)- system and software quality model; 2011.
16. Esaki K. System quality requirement and evaluation, importance of application of the ISO/IEC 25000 series, Global Perspectives of Engineering Management. 2013;2(2):52-59.
17. Al-Qutaish RE. Quality models in software engineering literature: An analytical and comparative study. Journal of American Science. 2010;6(3):166-175.
18. Bowen TP, Gray BW, Jay TT. RADC-TR-85-37, RADS, Griffiss Air Face Base N. Y.; 1985.
19. Shanthi PM, Duraiswamy K. An empirical validation of software quality metric suits on open source software for fault-proneness prediction in object oriented system, European journal of Scientific Research. 2011: 5(2):168-181.
20. Halstead, MH. Elements of software Science, New York, North-Holland; 1978.
21. Christensen, K., FIstos, P and Smith, CP. A perspective on the software science, IBM systems Journal. 1988;29(4):372-387.
22. McCable TJ. A complexity measure, IEEE Transaction on Software Engineering, 1976; SE-2(4):308-320.
23. Arthur, LJ., Measuring programmer productivity and software quality, John Wiley & Son, New York; 1985.
24. Albrech AJ, Gaffney JE. Software function, source lines of code and development effort function: a software service validation, IEEE Transaction on Software Engineering. 1983;SE-9(6):639-648.
25. Kemerer CF, Porter BS. Improving the reliability of function point measurement: an empirical study, IEEE Transactions on Software Engineering. 1992;18(11):1101-1024.
26. Fjiogu LO. A unified theory of software metrics, Softmetrix, Inc. Chicago, IL. 1988;232-238.
27. Fjiogu LO. Beyond structured programming, An introduction to the principle of applied software metrics, structured programming, Springer- verleg, N. Y.; 1990.
28. Henry S, Kafura D. The evaluation of software systems' structure using qualitative software metrics, Software- practice and Experience. 1984;14(6):561-573.
29. PMBOK, A guide to the project management Body of Knowledge. Project Management Institute Standards Committee; 2002.
30. Cavano JP. Software reliability measurement: Prediction, estimation, and assessment, Journal of Systems and Software. 1984;4:269-275.
31. Walston CE, Felix CPA. Method of programming measurement, and estimation, IBM Systems Journal. 1977;16: 54-73.
32. Boehm BW. Software Engineering Economics. Englewood Cliffs, NJ, Prentice Hall; 1981.
33. Khatibi V, Jawawi DNA. Software cost estimation methods: a review, Journal of Emerging Trends in Computing and Information Sciences. 2011;2(1):21-29.

34. Putnam LH. A general empirical solution to the macro software and software sizing and estimating problem. IEEE Transaction on Software Engineering. 1978;SE-4(4):345-361.
35. Kemerer CF. An empirical validation of software code estimation models, Communications of the ACM. 2008;30(5):416-429.
36. Premal BN, Kale KV. A brief overview of software testing metrics, International Journal of Computer Science and Engineering. 2011;1(3/1):204-211.
37. Pressman RS. Making Software engineering happen: A Guide for instituting the technology, Prentice Hall, New Jersey; 1988.
38. Kan SH. Metrics and models in software quality engineering, chapter 4, software quality metrics overview, Addison-Wesley professional; 2002.
39. Basili VR, Weiss DM. A methodology for collecting valid software engineering data, IEEE Transactions on Software Engineering. 1984;SE-10:728-738.
40. Daskalantonakis, MK. A practical view of software measurement and implementation Experience within Motorola. IEEE Transactions on Software Engineering. 1992;SE-18: 998-1010.
41. Kuar A, Suri B, Sharma A. Software testing product metrics – A Survey, In Proceeding of national Conference in Challenges & Opportunities in Information Technology, RIMT-JET, Mandi Gobindgarti; 2007.
42. Farooq SU. Quadri SMK, Ahmad N. Software measurements and metrics: role in effective software testing. Internal Journal of Engineering Science and Technology. 2011; 3(1):671-680.
43. Lei S, smith MR. Evaluation of several non-paramedic bootstrap methods to estimate Conference Interval for Software Metrics, IEEE Transactions on Software Engineering. 2003;29(1):996-1004.
44. Pandian CR. Software metrics – A guide to planning, Analysis, and Application, CRC press Company; 2004.
45. Juristo, N. and Moreno, AM. Basic of software Engineering Experimentation, Kluwer Academic, publisher, Boston; 2003.
46. Dumake R, Lother M, Wille C. Situation and treads in Software Measurement – A statistical Analysis of the SML Metrics Biolography, Dumke / Abran: Software Measurement and Estimation, Shaker Publisher. 2002;298-514.
47. Dao M, Huchard M, Libourel T, Leblance H. A new approach to factorization – introducing metrics. In proceeding of the IEEE Symposium on Software Metrics, METRICS. 2002;27:236.
48. Fenton N, Krause P, Neil M. Software measurement: Uncertainty and causal modeling, IEEE Software, July / August. 2002;116-122.




*Peer-review history:*
*The peer review history for this paper can be accessed here:*
*http://www.sciencedomain.org/review-history.php?iid=541&id=5&aid=4777*