QUANTITATIVE EVALUATION OF SOFTWARE QUALITY

ABSTRACT:
In that paper we established a framework conceptually with some of the initial key result that helps to analyzed the software quality characteristics. These are provided the explicit attention to the software quality characteristics that can save the costs of the life-cycle of software. The state-of-art currently provided by the software limit our ability to evaluate the software quality quantitatively as well as automatically. A well-known, well-defined and well-differentiated feature of software quality has been developed, these are the higher level structure that represents the actual uses of the software qualities, and its lower level features compare closely with evaluation of actual software metric that can be performed. However a lot of software quality metrics evaluations have been identified, classified, defined as well as evaluated which represents the actual and potential benefits, their quantifiability, and ease of automation. Sufficient activities of software life-cycle have been established providing the significant impact on the quality of software.

This paper provides a well-defined framework, highlighting the issues associated with the quality of software via comprehensive set of definitions, consistent distinctions, mutually supportive guidelines as well as experiences cited. However this framework may be not complete certainly because the future refinements and extensions are considering on the viable basis.

Keywords: Accessibility, Accountability, Accuracy, Augmentability, Communicativeness, Completeness, Conciseness, Device-Independence, Human-Engineering, Legibility, Maintainability, Modifiability, Portability, Reliability, Robustness, Self-Containedness, Self-Descriptiveness, structured-ness, Testability, Understandability, and Usability.

1 – Introduction:

Why Quality of the Software to be Evaluate?
To answer that question, first of all think about a software product that is performed all of its function correctly and efficiently, as well as delivered within specific time and budget. Now, do you satisfy with it? The answer may be no, due to several reasons you may find:

1. May be that software product is difficult to be modify or understand, so you may be bear the excessive cost for the maintenance of that product.
2. May be this software product is difficult to use or may be it is easy to misuse.
3. May to the integration of your software product with other software is not easy, i.e. your software product is machine dependent.

Major Software Quality Decision Points:
Due to many of the strong reasons measure of software quality is necessary, and its need becomes important to having clear understanding of the different characteristics of the software quality.

1. *Preparation the specification of quality product*: It involves the functionality you needed with the required performance in case of speed and accuracy. While specifying the product quality, one should understandability and maintainability also needed in a testable fashion.
2. *Checking for compliance of the specification of quality product*: In case of meaningful quality specification it is essential that the compliance of the quality product specification performed adequately because often it is done with a huge investment by massive peoples.
3. *Establishing the design tradeoff between the development and operational cost*: The importance of this factor is due to limited budget and time schedule which case the projects to limits on maintainability, usability as well as portability.
4. *Package selection for the software*: Assessment relative to each package is the need of every user, today. So the adaptation in case of installation often changes the needs and the hardware.

The quality ever deals with cost, i.e. maintenance cost, to become a cost effective one should consider the increased capability to deal with software quality. If the quality is too little in case of maintainability than it will increase cost in case of life-cycle maintenance where error correction and user requirement can become bottle neck. To evaluating the quality of software that definitive work has not been done properly and it is future need.

1.1 – Pervious Studies:

Rubey and Hartwick [1] are those first who have been attempt the work on software quality evaluation.

Code "attributes" and their "metrics" methods of Ref. 3 are the simple expressions to achieve the desired quality of the software and the latter stage to define the attribute the mathematical functions can be relate as parameters. Some of the attributes such as program intelligibility, mathematical calculation performed correctly and easy modification are need to be further analysis to achieve more concreteness. These types of attributes need to further analysis that weather those are present in form of some degree in the software. Although a few numbers of matrices was given in the previous work and no further application was define, however this paper includes some of the major attributes.

Later Brown [2] includes some metrics formulation with application for experiment of the two computer programs. He added approximately 400 line of code in FORTRAN language and tries implementing same specification independently for these two computer programs. Although, he considered some of the limited number of attributes which are corresponds to intelligibility as well as easy of modification.

However there was a difference between these two programs leading to quality emphasis: one program done by the programmer as "hotshot" in which the code efficiency was addressed and second program done by the careful programmer who addressed the simplicity. The results of these two programs as under:

- In one thousand tests, the efficient program introduced errors as many as ten times, however these series of tests was identical.
- On the other hand, in the simple program the quality was significant higher indicating the relative operational reliability.

Similarly, other software engineers were characterizing the significance which is recognizing as other attributes of software quality. Wulf [3] characterize some attributes importantly and reasonably known as non-overlapping attributes such as: robustness, cost, performance, modifiability, maintainability, portability and human factors.

Abernathy [4] analyzed and characterizes some tradeoff of the operating systems. Weinberg [5] also experiment with the several programmers by given then various success criteria as clarity of the program, clarity of the output, core amount in used, number of statements and development speed. And the programmers achieved the highest goal according to the success criterion.

A lot of engineers were establishing the criteria for quality assurance, recognizing its importance, dealing implicitly and explicitly quality attributes of the software by addressing and establishment the best programming practices. Kernighan & Plauger [6] in the "The book on programming style", represents the best work examples. There are also a lot of reports [7] [8] concerning with the maintainability of software in which source material and checklist items for practices as well as features establishing and enhancing the software maintainability such as: modularity, uniformity, conceptual grouping, meaningfulness, top-down programming, comments, name, parentheses, compactness, transferability and naturalness.

In a report of software Portability [9] by Warren, an excellent summary provided as an alternative approaches concerning portability such as: interpretation, bootstrapping, emulation, simulation with some of the high languages features which especially deals with portability.

A lot number of engineers were trying to recognized important quality factors in software engineering, i.e. AIAA/ACM/IEEE/DOD Software Management Conferences was held, in which: DeRoze's presentation [10] recognized "Software quality specifications and tradeoff" which is the initiative of DOD as a high-priority; Kossiakoff [11] identify near about seven attributes as a good software specification; Whitaker [12] represent approximately twelve quality goals explicitly in the regard of new DOD programming languages; and Light's represents

[13] near about five important software quality measures on software quality assurance.

Key Issues in Software Quality Evaluation:
There are some of the major issues in software quality evaluation which are the following:

1. "Measurable as well as sufficiently non-overlapping characteristics of software quality", we will discussed in section II on "Characteristics of Quality Code" with the regard of software quality evaluation.
2. "Measuring the overall software quality and/or its individual characteristics", we will discussed it in section III on "Measuring the Quality of Code".
3. "Software quality characteristics used to improve the software life-cycle process", we will discuss this in Section IV.

2 – Characteristics of Quality Code:
Software requirements and software design reflect in code which is actually controlling the user's system operations. In this section our focus is on quality code dealing and addressing the different problems in coding. This section also and following section measuring the quality code, which is based on study by TRW for National Bureau of Standards [14] and also based on study of Software Reliability performed for Rome Air Development Center [15].

Initial Study Objectives and Conclusions:
The identification of software quality characteristics were the main objective of "Characteristic of Software Quality" [14] by defining the metrics such that:

1. The degree of associated characteristic quantitatively measure by the metric when an arbitrary program is given.
2. Metrics function values represent the quality of software as a whole.

Components of software such as test pans, operational manuals as well as functional specifications; all are concerned on metrics which one can be applied to the source program.

The study provided some of the conclusion initially such as;
First, in evaluation of software product, considered how and where rather than how the deficiency of product. So the good and valuable automated tools indicated deficiencies and anomalies in the program rather than only producing the numbers. This phenomenon become true in shape of complier diagnostics, where one could estimate that near about 1.17% of program statements suffered with unbalanced parentheses problems.

Secondly, the quantitative formulas are the indicators for quality of the software; we demonstrate it with simple example to establish its credibility.

1. The structure of a program module can be measured via a metric which calculate the average size of that module. Now if we have software in which the 100 of the control statements routines and a library of m which have 5 computational statement routines, which one can considered for value of m and n as well structured. Now if we have m = 98 and n = 2 then our module size have average of 6.9 statements, similarly if we have m = n = 10 then our module size will become as 52.2 statements as a whole.
2. Another metric which are known for "robustness" indicated the statements fractions w.r.t potential singularity, i.e. (square root, divide, logarithm, etc.), those statements which are tested for singularities. However this may be not possible some time with in context of an operation such that; if we calculate the hypotenuse some of the right triangle such as,
   Z = SQRT(X**2 + Y**2)
3. Another metric known as "self-descriptiveness" for the sack of comments, i.e. the average length of the comments, etc. However, shorter and comprehensive comments are more powerful and easy to understand than robust, lengthy and poorly written comments, and it may be become expensive some time.

Thirdly, the software metrics are being evolved rapidly addressing for various field of software technology which is reinforcing the current practice. However, this may be not good due to some reasons, i.e. using of data clustering [16] as well as automated type-checking, where the reliability of these metrics may be invalidate due to mixed-mode expression's checking caused for violation of parameter range.

Lastly, we found that performing the calculation the values of overall or single metric for the quality of software can cause trouble because individual characteristics for the quality may arises conflicts. If we add more and more efficiency it bears the cost of

portability, maintainability, understandability and accuracy and similarly conciseness can be create problems in form of conflict with legibility. Users always feel difficulties to quantify its preferences in case of conflict situation. Sometime these metrics while measuring the characteristics which are associated with them are incomplete. We summarize all of these conditions as follow;

1. Some the quality of the software product which are desirable by the users as their priorities are conflicting.
2. There are no single metric which one can fulfill the entire requirement of software quality via a rating.
3. However, using priorities and checklists a useful rating can be achieve the prospective user.
4. Since the metrics are not fully complete in all manners so the rating would be suggestion rather than prescriptive or conclusive.
5. Therefore, the metrics are the best for indications of anomaly individually or separately and one it can be used for test planning, maintenance, acquisition and for guidance for software development.

Classification and Identification of Quality Characteristics:

After reaching the above conclusion, the engineers was decided to develop the set of characteristics in a hierarchical structure and such type of metrics those detect the anomalies. However, the approaches were as follow;

1. Define only the important software characteristics which one can as non-overlapping and would be exhaustive reasonably.
2. Define the candidate metrics which can access the already defined characteristic of software.
3. Determining the characteristics of associated software metrics to examine the quality of the software w.r.t their correlation as well as determining the potential benefits w.r.t magnitude, ease of automation and quantifiability.
4. Evaluation each of the candidate metric according to above narrated criteria and according to the interaction with different metrics, i.e. dependencies, overlapping and shortcomings.
5. Base of the above evaluation the refinement the software characteristics in such a way which are exhaustive, supportive and mutually exclusive.
6. Refinement of the candidate metrics refines them as a revised set of characteristics.

The initial set of characteristics for the software was developed and defines which are as follow; *Understandability, Completeness, Conciseness, Portability, Consistency, Maintainability, Testability, Usability, Reliability, Structured-ness and Efficiency.*

In the second step, we represent the candidate metrics which serve as an indicator of understandability and maintainability of code. Since understandability and maintainability are relative in measure because maintainability needs to understandability first, without a proper understandability a maintainer cannot maintain the code. At the same time, maintainability may have nothing to measure for understandability, i.e. some of the features related to the testability which one can support the intermediate output and echo checking of input are relatively important which used some time to retest the functionality but nothing to need as an understandability.

Following Figure 1 show that relationship between the maintainability, testability and understandability, where the arrow directed as logical implication, where high degree of maintainability needs to high degree of understandability and then high degree of testability can be performed.

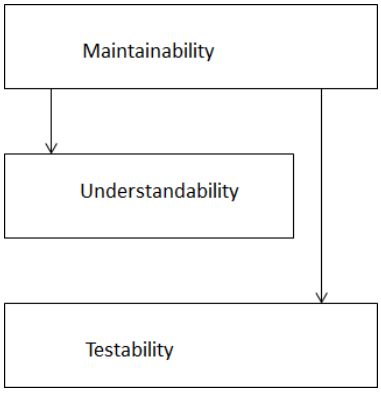

Figure 1: Relationship between the maintainability, understandability and testability.

There can be more concepts initially which can fall below the level of understandability and testability. For example a program can be more understandable if it is written in structured manners, and it should be consistent as well as concise. There are three additional characteristics belongs to the level of

understandability. Two of the others may be the legibility and self-descriptive but they many not applied by the tree above. In that way we find more and more characteristics and one can find that all these set of characteristics can be represent as a tree structure in which each of the initial characteristics can become more necessary condition of the more characteristics in general

The resulting tree in Figure 2, for the software quality represents the higher-level structure that one can represents the actual uses of evaluation for the software quality. Generally, when we need a software package we need to know the following three questions:

1. How well the features like easily, reliably, and efficiently can be use?
2. How much easy for maintenance, i.e. understandability, modifiability and retesting?
3. Can the software used in any of the changing environment?

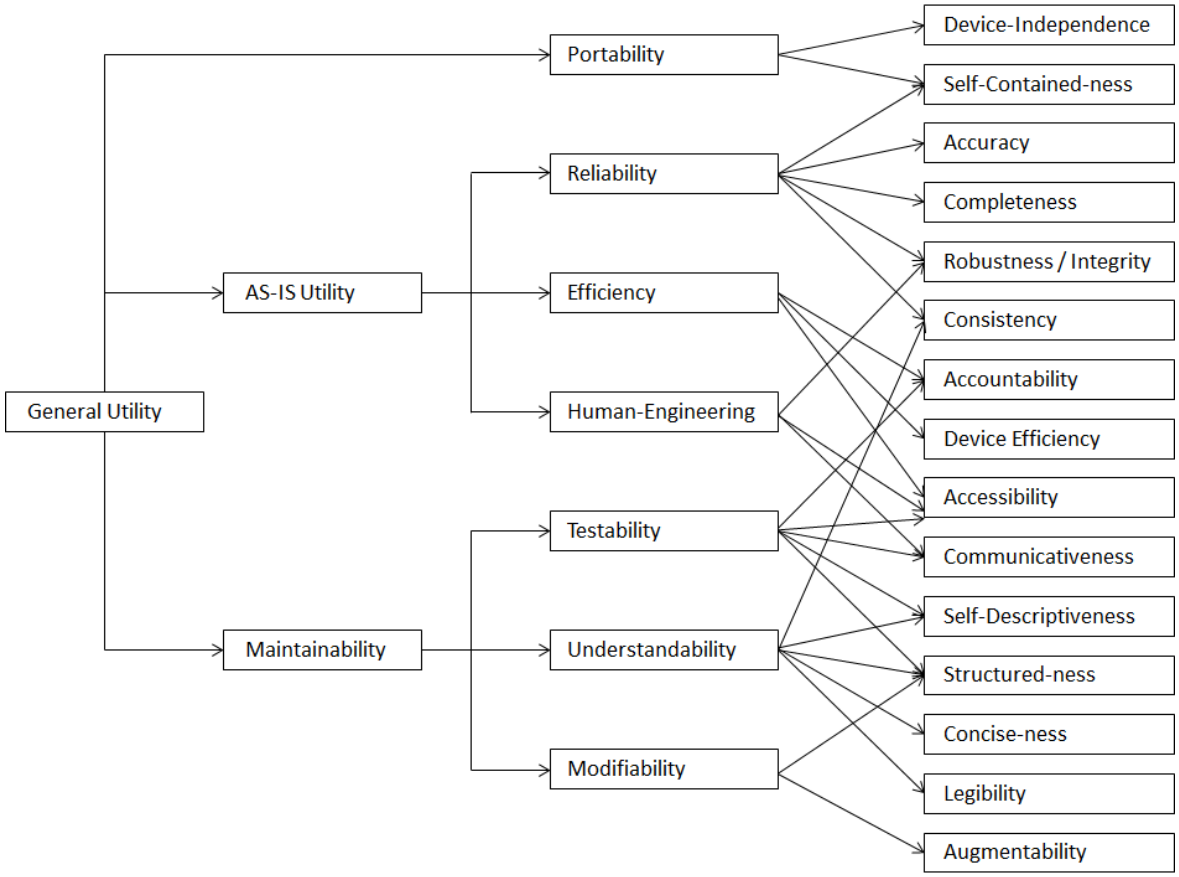

Figure 2: Software Quality Characteristics Tree

For the general utility, all of AS-IS utility, maintainability and portability are not sufficient, as these all requires more characteristics in hierarichy. As in the above figure 2, the AS-IS utility required to the program be more reliable, effieient and need human engineering. On the other hand, maintainability requires the testability, understandability as well as modifiability. More further donw in the hierarichy as represent in the figure 2. These low level charachteristic needs as initial characteristics which have difference with each other and the combination of these become set of intermediate level characteristics, these are necessary conditions. For example:

- Self-containdness is depend upon the fully initiliztaion of its own storage, therefore it may not be portable even if it is device independent completely.
- A program that is no device-independt and protable completely then even it may be self-contained completely.
- Portability needs satisfies the device-indepent and self-containedness conditions even it may not be satisfies these conditions like;

accuracy, completeness, consistent, accessibility, robustness, consistency, accountability, self-descriptivenss, device-efficiency, legibility, conciseness, augmentablilty and communicativity.

Quantitative metrics can be defined by using these primitive characteristics foundations, and then these metrics can be used measure the quantitative-ness of primitive as well as higher level characteristics. At the higher level it can be done by evaluating the utility of the software product, and at primitive level it can be done in prescribed direction. Now in the next section we define and evaluate these metrics.

3 – Measuring the Quality of Code:

Metrics are used to measure quality of product when processed, where product is referring as a piece of code here. As in the previous section we found that the developments as well as refinement of metrics and its characteristics are concurrently proceeding. A lot of metrics defined for Fortran was analyzed, refined and generalized in the previous section, now we examine its usefulness using the following criteria.

3.1 – Correlation with Software quality.

Every metric measuring its initial characteristic and the question is that did is it correlation with software quality notion? i.e. the correlation in which most of the computer programs for a certain metric would process it with relative primitive characteristics. However a statistical definition can be stated precisely but the evaluation becomes little subjective and that correlation measurement needs to data collection at extensive level from many diverse computer programs. We use the following scale while measuring the rate of each metric.

A+	Very high positive correlation, in which all of the programs with high metric score will possess the associated characteristic.
A	High positive correlation in which all of the programs (75% to 90%) with high metric score will possess the associated characteristic.
U	Usually all of the programs (50% to 75%) with high metric score will possess the associated characteristic.
S	Some program with high metric score will possess the associated characteristic.

3.2 – Potential Benefit of Metrics:

Some of the metrics facilitate with decision input for user and developer, others provide the interesting information may be some problem. However it is not loss even when a metric do not provide the high score in case of correlation with its associated quality characteristic. But at the same time we can measure the judgment for its potential benefit with an evaluator which is accessing the product. To measuring the potential benefits following scale was defined:

5	Extremely Important for highly score metric; potential problems otherwise
4	Important for highly score metric
3	Fairly for highly score metric
2	Some incremental values for high score metric
1	Slightly incremental values for high score metric; no real loss otherwise

3.3 – Metric Quantifiability and Feasibility of Automated Evaluation:

We can rate a metric on the basis of its correlation w.r.t quality as well as its potential benefit; however it may become time consuming and expensive some time due to the determination of its numerical values. It becomes an expensive when an expert is required to examine and analyzed the program for making a judgments and the numerical values provides less insight than the understanding for evaluation process.

Expert inspectors are needed for analysis the metrics which is so expensive, therefor automated programing and algorithm can be used to examine the program as well as for producing the numerical values. Checklist is another solution in which a list of desired quality characteristic provides to automated compliance checker which quantify w.r.t cost-effective rating for the given metrics. Following set of options we will use:

AL	Automated Algorithm: Can be done cost-effectively
CC	Automated Compliance Checker: can be done cost-effectively if given a check list is code auditor tool, described in Section 5)
UI	Untrained Inspector requires
EI	Expert Inspector requires
EX	Execution of the program is need

Automation the evaluation can be performed in easy manners, i.e. average module length counting and self-descriptive material checking. However, it is difficult to globally scan the program for some repeated sub-expressions and component guaranteeing that are not modified in between the repetitions.

On the other hand, judgment the descriptiveness in self-descriptive material cannot be automated. Since, some of the automated tools can be useful but provide partial support for the quality evaluation. Following scales used to rate metrics for ease-ness and completeness.

Ease of Developing Automated Evaluation:

E	Easy to development (automated algorithm), or compliance checker
M	Moderately difficult (to develop automated algorithm) or compliance checker
D	Difficult to develop (automated algorithm) or compliance checker

Completeness of Automated Evaluation:

C	Algorithm or checker provides total evaluation of metric
P	Algorithm or checker provides partial evaluation of metric
I	Algorithm or checker provides inconclusive results

Primitive Characteristics	Definition of Metrics	Correlation With Quality.	Potential Benefit	Quantifiability	Ease of Developing Automated Evaluation	Completeness of Automated Evaluation
Device Independence DI – 1	Are computations independent of computer word size for achievement of required precision or storage scheme?	A+	5	AL + EX + TI	E	P
DI – 2		A+	5	AL	M	P
Self-Contained-ness SC – 1	Does the program contain a facility for initializing core storage prior to use?	A+	5	AL	E	P
SC – 2	Does the program contain a facility for proper positioning of input/output devices prior to use?	A+	5	CC	E	P
Accuracy AR – 1	Are the numerical methods used by the program consistent with application requirements?	A+	5	TI	E	
AR – 2	Are the accuracies of program constants and tabular values consistent with application requirements?	A+	5	AL + TI	E	P

Completeness CP – 1	Are all program inputs used within the program or their presence explained by a comment?	U	3	AL	E	C
CP – 2	Are there no "dummy" subprograms referenced?	S	2	AL	E	C
Robustness R – 1	Does the program have the capability to assign default values to non-specified parameters?	A+	5	AL + TI	E	P
R – 2	Is input data checked for range errors?	A	5	AL + TI	E	P
Consistency CS – 1	Are all specifications of sets of global variables (i.e., those appearing in two or more subprograms) identical (e.g., labeled COMMON?	A	4	AL	E	C
CS – 2	Is the type (e.g., real, integer, etc.) of a variable consistent for all uses?	A	5	AL	E	P

Table 1: Evaluation of quality metrics

3.3.1 – Evaluation of Metrics:

Table 1 contained some criteria as an example for candidate metric development. 151 candidate metrics [16] can be included here. Table 1 shown six primitive characteristics evaluation of two associated metrics, here metric DI-1 found as ratting "A", i.e. highly correlated as device-independence and the extremely important w.r.t device-independence ratting as "5". These kind of automated tools, algorithms, its execution by trained personals all are cost effective to determining the degree as a characteristics of software product, i.e. ratting then like "AL + EX + TI". Moreover, the automated tool can checked the format statements for device-dependency and flag constants like PI = 3014159265359.

3.3.2 – Evaluation of Metrics VS Project Error Experience:

The evaluation of metrics as well as its rating is best exemplified in Table 1, in the extensive data is available to detect the software error types and then correcting it, however there are 13 major categories of software error, such as: (1) Software interface errors (2) Disk handling errors (3) User interface errors (4) Computation errors (5) Indexing and subscripting errors (6) Hardware interface errors (7) Output processing errors (8) Error message processing errors (9) Iterative procedures errors (10) Tape handling errors (11) Data base interface errors (12) Bit manipulation errors (13) Errors in preparation or processing of card input data.

These errors were analyzed and determine the stages of software development life cycle, where they can commit and where they can be correct; these stages are as follow; (1) Requirements Definition (2) Design (3) Code and Debug (4) Development Test (5) Validation (6) Acceptance (7) Integration (8) Delivery.

E	Error origin
F	Error found
CT – N	Consistency checking aid N applied at this phase would generally have detected error.
CM – N	Completeness checking aid N applied at this phase would generally have detected error.

Serial Number	Software Phase / Error Type	Requirements	Design	Code	Development Test	Validation	Acceptance	Integration	Delivery
1	Expectation of parameter by program in different format verse requirement specification.	--	E CT-13	--	--	--	F	--	--
2	Required parameter may or may not be accept by program.	--	E CT-13	--	--	--	--	F	--
3	Expectation of a parameter by program differently as it is specified.	--	E CT-13	--	--	--	--	F	--
4	Acceptation of data differently or not in its entire range.	--	E CT-13	--	--	--	F	--	--
5	Expectation of parameter in units differently as it is specified	--	E CT-13	--	--	--	F	--	--
6	Different specification of default value used by the program in absence of specific input data.	--	E	--	--	--	--	F	--
7	Acceptation of data outside the range.	--	E	--	--	--	--	--	F
8	No data accept by the program in allowable range.	--	E	--	--	--	--	--	F
9	Core table values overflow by the program within allowable range.	--	E	--	CM-9	--	--	--	F
10	Overflow of allotted space by program within allowable range.	--	E	--	CM-9	--	--	--	F
11	Succeed test case fail but first text case execute properly.	--	E	--	--	--	F	--	--
12	Expected parameter of program found in different location as specified.	--	E CT-13	--	--	--	F	--	--

Table 2: Evaluation of Error-Detecting Capabilities (Metrics) vs. Error Type (first 12 of 224 error types)

In table 2, "E" is placed in each column where error can be originated in corresponding phase, and "F" placed where error can be found and correct. For each error type an additional item is estimated which one can ensure the applicability as an evaluation of each metric, i.e. the phase where type of error mostly detected as well as corrected using that metric.

In table 2, we can see most of the errors originated at the design phase and found as well as correct at different phases. Here several extensions can be identifying by analysis effective in error detection and correction. Automated tools check for consistency by scan the standard software module header block about the nature of input and output, including: units, order of inputs, number of inputs, acceptable ranges, data type and format, associated storage locations, source (device or logical file or record) and access (read-only, restricted access).

Here CT-13 entries in the table shows if the module coding had been precede by module description (with some of the input output assertions) then error can be caught before the coding is began by using the automated consistency checker. Near about 12 type of card processing errors found in Table 2, where the consistency checker CT-13 caught 6 errors.

Metric / Primitive Characteristic	Error Type Corrected (No)	Phase Gain (Total)
Consistency	34	89
Robustness	29	47
Self-Containedness	15	28
Communicativeness	09	18
Structured-ness	02	02
Self-Descriptiveness	01	04
Conciseness	01	04
Accuracy	01	02
Accessibility	01	02

Table 3: Error Correction Effectiveness of Metrics

To detecting the software errors, the consistency metrics is most effective as shown in Table 3. Total phase gain here was 89 and it caught 34 of the 224 error types with the average of 2.5 phases per error. Next most metrics were robustness, self-containedness, communicativeness etc., in which software error detection and correction have significant improvement for early applications (semi-automated and automated). These are three primitive characteristics which associated with reliability.

4 – Improve the Software Life-Cycle Process using Quality Characteristics:

The software life-cycle process consist of software as well as system requirements followed by design, detailed designed, coding, testing and maintenance phase. There are four major goals that needs to be accomplished to achieve the quality characteristics:

4.1 – Explicitly Setting Software Quality Objective and Priorities:

Some of the experiments reported in Refs. 2 and 5 showed that the objectives of quality and priorities strongly correlated with the degree of quality. So, if the user requirements are more for maintainability and portability than efficiency of code than one should it reported to the developer in earlier phases. However user of the software allows determining that what kind of qualities will be present in the final product.

4.2 – Using Software Quality Checklist:

The software qualities metrics discus above and presented in Ref. 14 can be used in checklist of software quality as a basis an one these can be used in inspection, reviews and walkthroughs. These all are used primarily for the support of reliability objectives as well as for other objectives of software quality. The checklists are used to judge the self-descriptiveness for a computer program evaluating the maintenance of the program, its understandability, and its testability. Self-descriptiveness contained by a software product specify enough information to its reader for determination of objectives, constraints, input and output, assumptions, revision status and components. These checklists are:

a. A header block contain each program module describing the, name of program, purpose, effective date, requirement accuracy, restrictions and limitations, modification history, methods, input/output, assumptions and error recovery procedures.
b. Describe the alternate subsequent branching and decision points.
c. Function of modules and input/output defined adequately for module testing.
d. Providing comments for specific input values for specialized program testing.
e. Providing information about the change impact to the other portions of the program.
f. Identification of source code of program which need to handle a specific change.
g. Specification of module dependency in comments, documentation and in program structure inherently.
h. Descriptiveness of variable names which used to represent the functional or physical properties.
i. Adequate descriptive information, i.e. comments for uniquely recognizable functions to clear the purpose.
j. Provide an adequate description for correlation of variable name with its entities or physical properties.

4.3 – Establishing Explicit Quality Assurance Activities:

We are trying here to establish an explicit quality assurance activity as an increasingly advantageous for both quality product as well as cost-effectiveness standpoint. An independent view for product is need here for producing a deliverable product, for this manager of this activity directly report to project manager. The important task is tailoring to the project which actually depends upon on the size as well as scope of the project. This is an effective approach which ensures that the particular system application is responsive according to the quality requirement. To ensure the quality assurance activities following guideline should be considered:

- Planning – Preparation of quality assurance plan for software quality which interprets the requirements for quality program, its schedules, assigned tasks and other organizational responsibilities.
- Policy, Practice and Procedure Development – A standard manual need to be publish here which interprets all of the phases of software production, which includes: requirements, design, coding, testing, tailoring, etc.
- Software Quality Assurance Aids Development – to verifying the functionalities of the software procedures and ensuring its performance for requirements as well as its project quality

standards manual need to be developed and adopted.
- Audits – A review procedure is need to be initiate against the procedures of the projects and for documentation, ensuring the compliance is going according through standard development plans as well as in corrective manners.
- Test Surveillance – A mechanism for reporting the software problems against the error, analysis the assurance for corrective actions.
- Records Retention – Problems in software, i.e. in design, other test cases, test data, activity logs, need to be retention helps in quality assurance reviews.
- Physical Media Control – An inspection is needed which ensure the retention or physical transmittal media like disks, cards, tapes, etc. against the destroyable, alterable and mishandling contents.

These type of responsibilities need to be the reliability consideration which ensuring the compliance the standard requirements against the software product. However, other type of software qualities like maintainability and portability are also important characteristics which one can use as focal point for assurance.

4.4 – Using Quality Enhancing Tools and Techniques:

Different kind of software tools like configuration management procedures, cross-reference generators, and other software monitors have some kind of quality enhancement mechanism or technique. These are actually providing high leverage quality of enhancements for software product which firstly applying on requirement and design phase and then subsequent phases like coding.

4.4.1 – Requirement and Design:

By using checklists and guidelines the quality assurance characteristics can be obtained during the requirement as well as in design phases. At this stage we need to deal with problem criticality of varying level instead of measure the numerical values. Specifications of the software should be machine analyzable while consistency and completeness can be analyzed via automated aids software for requirements like; ISDOS [17] and SREP [18] [19]. These systems provide the considerable leverage, significantly increased assurance for specifications which actually limiting the cost during the testing and maintenance. Majority of errors arise due in requirement and design phase [20], i.e. faulty expression, incomplete designs, and these are less expensive and correct easily during the earlier stage of software development life cycle.

A biggest source for software problems is ambiguity, i.e. there is a lot of programmer who wrote the requirements in its own manner, and if different kind of groups, designers, tester, trainers and users interpret it differently, many of the operational problems will result. One of the solutions is that the requirement should be testable.

	Non-Testable	**Testable**
1	Accuracy shall be sufficient to support mission planning	Position error shall be: ≤ 50' in the horizontal ≤ 20' in the vertical
2	System shall provide real-time response to status queries	System shall respond to: Type A queries in ≤ 2 sec Type B queries in ≤ 10 sec Type C queries in ≤ 2 min
3	Terminate the simulation at an appropriate shift break	Terminate the simulation after 8 hours of simulated time

Table 4: Non-Testable VS Testable Requirements

Table 4 shows that testable requirements are less ambiguous as well as they are best suited for baseline designing, documenting, costing and maintenance of the system. However, requirement matrix is another technique which can be analyzed such kind of quality consideration explicitly during the requirement phase which is successful for small projects.

4.4.2 – Requirements Properties Matrix:

In that case the functional requirements stated individually in the column and major qualities (properties) of software product stated in row (or vice versa). The additional elements are supposed as matrix elements refer to implication of quality for each requirement. In example, in Table 4 represent the pair of requirements when treated in requirement properties matrix. In that way the specification those are generating as a result will always lead to higher software quality product. However, some of the additional effort would also be necessary to achieve the enhancement in the

software product. Life cycle cost can also be reduced if the programs have some maintenance and good deal of use.

Requirement / Property	Simulation terminate at an appropriate shift break	…
Testability	Simulation terminate after 8 hours of simulated time	…
Modifiability	Allow user to specify termination time as an input parameter, with a default value of 8 hours	…
Robustness	Provide an alternate termination condition in case the time criterion cannot be reached,	…
…	…	…
…	…	…
…	…	…

Table 5: Portion of a Requirements-Properties Matrix

As shown in Figure 2, that one of the necessary characteristics of code structure-ness is testability, understandability and modifiability, and all of these are necessary for maintainability purpose. FORTRAN constructs consist of some of the structured control basically, such as; SEQUENCE, IF-THEN-ELSE, CASE, DO-WHILE, and DO-UNTIL [21], so the programmers get some structured-ness for a large real-time software projects. STRUCT is used in FORTRAN to scan the source-code to determine each and every routine to check if it is properly nested according to allowable constructs and if it is not, the violation is recognized, identified and a diagnostic issued.

The CODE AUDITOR is an analysis tool for FORTRAN which can automatically checked the structuring standard of a source code for violation and its source location. All of the primitive characteristics in Figure 2 are measureable by the CODE AUDITOR. For example, the presence of standard module header commentary cards to explain transfer of control checked the Self-Descriptiveness. In that way the compliance with standard for parameter passing and checking for mixed-mode expressions can be used for Consistency.

5 – Future Expectations:

In the future, some of the standard language features will be done more efficiently, i.e. for structured programming and data typing, etc. However, automatic post-scanning of code is need which can assure the compliance for local standard (e.g., naming convention), and at the same time some of the potential quality problems need to check for partial indicators.

Ratings in Table 2 show that there are still needed for trained humans which can improve the evaluations of code as well as design to assure the quality requirements. However, for measuring and controlling the software quality the automated tools and human inspectors needed with variety of techniques, as stated in [22], where emphasis on relative merit for some experiences on large software projects. Design and code inspection [23] is a “cousin” technique in this required a structured walkthrough. Near about 58% errors can be eliminated during the design inspection originated in validation phase, an example of large project [17]. Fagan’s report [23] that approximately 23% errors can be reduce by performing the appropriate inspection at each stage, this should need also to be improved in near future.

6 – Conclusion:

Section 1: Significant saving can be achieved in software life-cycle costs by given the explicit attention to the characteristics of software quality.

Section 2: The quality of the software, while quantitatively and automatically evaluation is limited by current software stat-of-the-art. However well-defined and well-differentiated characteristics of definitive hierarchy has be established, where its higher-level structure represent the actual used of software evaluation and its lower-level characteristics are correlated closely with evaluation of software metric which one be performed.

Section 3: Evaluation of metrics have been classified, defined, evaluated w.r.t its potential benefits, quantifiability and ease of automation.

Section 4: Software development life-cycle activities have been identified and established which have the significant leverage on quality of the software, including;

- Establishing the objectives and priorities for software quality explicitly.
- Testable software requirements insurance.
- Establishment of quality assurance activities explicitly.
- Uses of benchmarking for measuring the software quality.

- Using of software specification which are machine-analyzable.
- Standard compliance checking by the automated Code Auditor.
- Uses of Matrix for requirements properties.
- Used of checklist for software quality.
- Structure code standard establishment.
- Code inspection and design performance.

Study reported in this paper involves well-defined framework which accessing actually the slippery issues that are associated with the quality of software, via sets of definition which are mutually and consistently supportive, guidelines, distinctions and experiences cited. However, this framework is not complete but it provides some cost-effectiveness for code-analysis evaluation and it may serve as bases for future refinements and extensions.

References:

01	"Quantitative Measurement of Program Quality," Rubey, R. J., and R. D. Hartwick, Proceedings, ACM National Conference, 1968, pp. 671-677.
02	“The quantitative Measurement of Software Safety and Reliability” by Brown, J. R., and M. Lipow, revised from TRW Report No. SDP-1776, August 1973, TRW Software Series (in press August 1976).
03	"Programming Methodology," by Wulf, W. A., Proceedings of a Symposium on the High Cost of Software, J. Goldberg (ed.), Stanford Research Institute, September 1973.
04	"Survey of Design Goals for Operating Systems" by Abernathy, D. H., et al, Georgia Institute of Technology Report GTIS-72-04.
05	"The Psychology of Improved Programmer Performance" by Weinberg, G. M., Datamation, November 1972, pp. 82-85.
06	“The Elements of Programming Style”, Kernighan, B. W., and P. J. Plauger, McGraw-Hill, 1974.
07	A Study of Fundamental Factors Underlying Software Maintenance Problems, CIRAD, Inc., December 1971.
08	Research Toward Ways of Improving Software Maintenance, CIRAD, Inc., January 1973.
09	Software Portability by Warren, J., Stanford University Digital Systems Laboratory, Technical Note No. 48, September 1974.
10	"DOD Defense System Software Management Program", DeRoze, B. C., Abridged Proceedings from the Software Management Conference, 1976 (obtainable through Los Angeles Section AIAA).
11	"Software Requirements Analysis and Validation", Kossiakoff, A., and T. P. Sleight, ibid.
12	"DOD Common High Order Language (HOL) Program", Whitaker, W. A., ibid.
13	"Software Reliability/Quality Assurance Practices", Light, W., ibid.
14	Characteristics of Software Quality, Boehm, B. W., J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod, M. J. Merritt, TRW Software Series TRW-SS-73-09, December 1973.
15	Software Reliability Study (Final Technical Report), Thayer, T. A., et al, TRW Report No. 76-2260.1.9-5, March 1976.
16	"Programming with Abstract Data Types", Liskov, B. H., and S. N. Zilles, ACM SIGPLAN Notices, April 1974, pp. 50-59.
17	"Automation of System Building," Teichroew, D. and H. Sayari, Datamation, August 1971, pp. 25, 30.
18	"A Requirements Engineering Methodology for Real-Time Processing Requirements," Alford, M. W., Jr., Proceedings, IEEE-ACM Second International Conference on Software Engineering, October 1976.
19	"An Extendable Approach to Computer-Aided Software Requirements Engineering," Bell, T. E., andD. C. Bixler, ibid.
20	"Some Experiences with Automated Aids to the Design of Large-Scale Software," Boehm, B. W., R. K. McClean, and D. B. Urfrig, IEEETrans, Software Engineering, March 1975, pp. 1Z5,133.
21	“Mathematical Foundations of Structured Programming” by Mills, H. D., IBM-FSD Report-72-6012, 1972.
22	“Proceedings of the AIIE Conference on Software”, Brown, J. R., Washington, D.C., July 19-21, 1976.
23	“Design and Code Inspections and Process Control in the Development of Programs,” Fagan, M. E., IBM- TR-21-572, December 1974.

About the Author:
SaFi Khan;
MS in Computer Sciences;
(Software Engineering);
Virtual University of Pakistan.