University of Sindh Journal of Information and Communication Technology (USJICT)
Volume 6, Issue 2, July 2022
ISSN-E: 2523-1235, ISSN-P: 2521-5582
Website: http://sujo.usindh.edu.pk/index.php/USJICT/


# Impact of Software Metrics on Software Quality using McCall Quality Model: In-Depth Analysis

Sana Fatima, Zainab Fatima, Muhammad Abdullah Hayat, Muhammad Hamza Shahab, Muhammad Khurram Meraj, Rana M. Ibrahim, Syed Muhammad Muneeb
Software Engineering Department, NED University of Engineering and Technology, Karachi, Pakistan
Sana.fatima@cloud.neduet.edu.pk, Zainab.ned@gmail.com, malik_muhammad222@outlook.com, hamzashahab1610@gmail.com, khurrammeraj17@gmail.com, rmibrahim00@gmail.com, s.muneeb2k@gmail.com

***Abstract:*** Software metrics plays a very vital role in life cycle of software development. Rapid software development techniques and tools have made it very complex to fully control the quality of a software. Software metrics are required to make sure that the quality of software is fully under control. Many software metrics have already been developed and applied to control the quality of software products. Software metrics is the measurement of quality in which performance is measured against quality standards to check whether they are according to the expectations. Quality metrics are also used to determine customer requirements into acceptable performance measures. This paper discusses the concepts of software quality, quality factor model, mapping according to McCall Quality Model & the quality metrics. The act of applying software quality measurements to functional components and to keep up with factors is a mind-boggling task. Effective software quality affirmation is exceptionally reliant upon quality methods. Future examination is needed to expand out and work on the approach to widen measurements that have been accepted on one venture, utilizing our rules, legitimate proportions of value on future software project. This paper also dives deep into the impacts of the various software metrics over different quality factors and explains the relationship between them.

**Keywords**: Software Metrics, Software Quality Factors, Software Quality Model, Software Quality Assurance.

## I. Introduction

Software quality elements play a vital on the SQA activities to achieve quality goal. Quality metric is quantitative proportion of the degree to a framework, segment, or process. Software quality factors are doing significant role for acknowledgment in associations endeavor to further develop venture quality. The measurements are the quantitative proportions of how much quality measures in a given property that influences its built quality. To improve software quality, a detailed knowledge is required regarding the most frequently occurring code smells, most used refactoring technique and the software metrics that have a direct influence on them [1].

SQA is a conventional interaction for assessing and recording the attributes of the items created during every phase of the product improvement lifecycle. The Software metrics are necessary in the whole SDLC [2]. Software metrics indulges in the measurement of development process of software product and support in processing models. Using it we can increase range of information regarding quality of the product that is provided to the user, estimation of cost, software project progress, and software system complexity [3].

Meanwhile there is a proper map to the requirements and factors needed in a document according to McCall Quality Model [4]. They types are documented which helps to achieve metrics. The impact of factors show that we have found in order to attain software quality goal. We have quoted formulas to calculate different elements related to quality. We have mapped the factors according to the quality criteria.

The observations and the results concluded from the various mappings and discussions are sophistically illustrated in the form of tables which describe the impact of various software criteria in different lifecycle phases and the effect of software criteria of various software quality factors.

## II. Literature Review

The first concept of software metrics was based on the lines of code (LOC). The software metrics were used as the measure of program quality as well as the programmer's productivity. In 1961, software quality metrics were published for the first time, i.e., several defects per KLOC. But the obvious drawbacks were observed in the mid-1970s of this approach because the other aspects i.e., size, complexity, and functionality were not considered in software metrics [5].

Researches have been made using different aspects of the software to find the relation between software quality and software metrics, and how software metrics affect its quality. In 2019, research was made to find a relation between software metrics and complexity resulting in the variation of software quality. After observing results of various metrics on software quality and complexity, it was concluded that source code metrics is the best possible way to predict the build failure or success of software product. Also, the technique of data stream mining technique is the best way (yet) to ensure the quality of a software product. The impact

of different complexity metrics on software quality was calculated using various techniques. To increase the quality of the software product reliability, complexity, defect removal efficiency metrics, etc. help in increasing the quality & customer satisfaction model index is best for increasing customer satisfaction which leads to the quality of the software product [6].

To measure the quality of intermediate deliverables during software development, different rubrics are used which measure the quality of software according to requirements, design, and coding, this unit of measuring is called Software metrics [7]. To achieve the quality standards only external factors, matter but to achieve these external factors, the key role is played by internal factors. Different kinds of metrics include dynamic metric, object-oriented design metric, sub-factor metric, and structural metric for process model. This research paper suggest that understandability can be an easy approach for measuring. Many measures help to estimate different quality factors. Also that measure used in different studies shows that dynamic metric, source code and metric relating to documentation are mainly used in measuring the quality of the software. [8]

## III. Methodology And Discussion

### *A. Software Quality*

Software Quality is a measure to which the software works according to the requirements described in the document. It is a degree to which the software can perform the required tasks without any moderation [9]. There are many ways to measure the quality of the software these include a combination of different quality factors according to different quality factor models

### *B. Quality Metrics and Its Types*

Quality metrics are the main components in efficient software project quality management. It is the measurement of quality in which performance is measured against quality standards to check whether they are according to the expectations. Quality metrics are also used to determine customer requirements into acceptable performance measures. They are also used to evaluate and analyze software products and processes. [10]

Software metrics can be divided into three types [11]:

#### *1) Product Metrics*

Product metrics are used to measure the quality of the final product. The product metrics are the most extensive metrics from the three categories. Since its based on product, is measured at end of development phase [12]. It deals with different aspects of the product such as:

- Size (LOC)
- Complexity
- Reliability
- Portability

#### *2) Process Metrics*

Process metrics are used to increase the quality of software development and maintenance by considering many factors, such as:

- Time required to complete a product using a process.
- The effort required in a process.
- Defects found in a process.

#### *3) Project Metrics*

Project metrics are used to monitor the project progress and status so that the software development plan can be optimized.

- Few examples of project metrics are:
- Scheduling of a project.
- Cost estimation.
- Resources used in a project.

Software quality metrics mainly target the quality aspects of product, process, and project. They are further divided into three categories:

#### *1) Product Quality Metrics*

Product quality metrics deal with the maintenance of the quality of a product and its features. The true value of product metrics comes from their association with measures of important external quality attribute [13] s as it deals with various aspects few of them are:

- Customer satisfaction.
- Defect density.
- Meantime to failure.
- Customer’s problems.

#### *2) In-Process Quality Metrics*

In-process quality metrics mainly deal with the tracking of defects and errors during standard machine testing. They are less formally defined than end products and they vary among different developers. A few of its aspects are:

- Integration testing.
- Defect arrival.
- Defect removal pattern in a phase.
- Effectiveness of defect removal methods.

#### *3) Software Maintenance Metrics*

Software maintenance metrics are used during the phase of maintenance to ensure the quality and verifying that the software developed is according to the customer’s requirements. It includes:

- Fixing backlog and backlog management index.
- Making sure that response time (to any defects reported) is minimum.
- Fixing quality.

### *C. Mapping McCall Quality Model to Criteria and its Metrics*

The table 1 given below shows the relationship between various criteria for different factors. These quality factors are obtained using McCall's quality model. The main criteria in the McCall model are correctness, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, interoperability. [14] The criteria are just the simplification of the quality factor which makes it more measurable, simple to understand and specific [15] Achieving these criteria means that the quality factors have been met. Like, the correctness of software depends on completeness, consistency, etc. These quality criteria show that for every quality factor there are multiple attributes of the software product to depend on and these factors are used to define the quality of the software.

Following are the quality criteria factors mentioned below [16]:

*1) Completeness*

Attributes that require the module to provide the complete functionality of required tasks [17].

*2) Consistency*

Attributes that require the software to have a uniform design and functionality.

*3) Operability*

Attributes that are concerned with the correct functioning of the operations of the software.

*4) Conciseness*

Attributes that require the implementation of functions with minimum number lines of code.

*5) Efficiency*

Attributes that deal with minimizing the execution time and storage requirements.

*6) Augment Ability*

Attributes of the software that can be easily extended for further development.

*7) Security*

Attributes that are responsible for securing the software and correcting the known issues [18].

*8) Accuracy*

Attributes of the quality that are concerned with correct results in outputs.

*9) Modularity*

Attributes that deal with the structure of different components of software to be used in further development.

*10) Simplicity*

Attributes of software that provide easy understanding without any complexity.

*11) Training*

Attributes that help users to transition between the modules easily.

*12) Software Independence*

Attributes that determine the ability of software to deal with other software

*13) Generality*

Attributes that allow the software component to perform general functions without many moderations.

*14) Self-Documentation*

Attributes of the software that do not need any extra documentation and can be used easily with other modules.

*15) Data commonality*

Attributes that deals with correct data representation.

TABLE I. MAPPING MCCALL QUALITY MODEL TO CRITERIA AND ITS METRICS

| Quality Factors | Quality Criteria | Quality Metrics |
|---|---|---|
| **Correctness** | Completeness | ● Completeness checklist |
| | Consistency | ● Data consistency<br>● Procedure consistency |
| | Operability | ● User output communicativeness<br>● Operability checklist<br>● User input communicativeness |
| **Efficiency** | Conciseness | ● LOC metrics<br>● Conciseness Efficiency |
| | Efficiency | ● Storage effectiveness measure<br>● Processing effectiveness measure<br>● Communication effectiveness measure<br>● Data usage effectiveness<br>● Measure |
| | Operability | ● User output communicativeness<br>● Operability checklist<br>● User input communicativeness |
| **Integrity** | Augment ability | ● Channel extensibility<br>● Data storage expansion<br>● Design extensibility<br>● Computation extensibility |

| | | |
|---|---|---|
| | Security | • Security metrics |
| **Reliability** | Consistency | • Data consistency<br>• Procedure consistency |
| | Accuracy | • Accuracy checklist |
| | Modularity | • Modular design |
| | Simplicity | • Data and control flow complexity<br>• Design structure<br>• Structured language<br>• Halstead's level of difficulty measure<br>• Coding simplicity |
| **Usability** | Training | • Training checklist |
| | Operability | • User output communicativeness<br>• Operability checklist<br>• User input communicativeness |
| **Maintainability** | Conciseness | • LOC metrics<br>• Conciseness Efficiency |
| | Software independence | • Database independence<br>• Data structure<br>• Database management<br>• Microcode independence<br>• Database implementation<br>• Architecture standardization<br>• Function independence |
| | Consistency | • Data consistency<br>• Procedure consistency |
| | Modularity | • Modular design |
| **Portability** | Simplicity | • Data and control flow complexity<br>• Design structure<br>• Structured language<br>• Halstead's level of difficulty measure<br>• Coding simplicity |
| | Consistency | • Data consistency<br>• Procedure consistency |
| | Generality | • Unit referencing<br>• Unit implementation |
| | Self-Documentation | • Quantity of comments<br>• Descriptiveness of language<br>• Effectiveness of comments |
| | Modularity | • Modular design |
| **Reusability** | Self-documentation | • Quantity of comments<br>• Descriptiveness of language<br>• Effectiveness of comments<br>• Database independence |
| | Modularity | • Modular design |
| | Generality | • Unit referencing<br>• Unit implementation |
| | Software Independence | • Data structure<br>• Database management<br>• Microcode independence<br>• Database implementation<br>• Architecture standardization<br>• Function independence |
| **Testability** | Augment ability | • Channel extensibility<br>• Data storage expansion<br>• Design extensibility<br>• Computation extensibility |
| | Modularity | • Modular design |
| | Self-documentation | • Quantity of comments<br>• Descriptiveness of language<br>• Effectiveness of comments |
| | Simplicity | • Data and control flow complexity<br>• Design structure<br>• Structured language<br>• Halstead's level of difficulty measure<br>• Coding simplicity |
| **Interoperability** | Data commonality | • Structured language<br>• Design structure<br>• Data commonality checklist |
| **Flexibility** | Self-documentation | • Quantity of comments<br>• Descriptiveness of language<br>• Effectiveness of comments |
| | Generality | • Unit referencing<br>• Unit implementation |
| | Modularity | • Modular design |

| | | |
|---|---|---|
| | Software Independence | • Data structure<br>• Database management<br>• Microcode independence<br>• Database implementation<br>• Architecture standardization<br>• Function independence |

The table 1 above gives the relationship between quality criteria and quality metrics. This shows that every quality criterion is related to one or more quality metrics. Like efficiency criteria of software is related to processing effective measure, storage effectiveness measure, etc.

*C. Measuring Quality Metrics*

*1) Completeness Checklist*

The completeness of the checklist is measured by the following factors:

- Unambiguous references (input, function, output)
- The data reference is defined either from computed or are obtained from an external source
- Usage of all the defined functions
- Definition of all the referenced functions
- For each decision point, all the conditions and processing are defined
- All sequence parameters are defined and agreed upon
- Resolution of all the problem reports
- The requirements are aligned with the designs
- Code is in alignment with the design The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*2) Data Consistency Measure*

The measure of data consistency depends on the following factors:

- Representation of standard data usage
- Unit consistency
- Data type consistency
- Consistent global definitions
- Naming conventions

Where each of the above parameters is calculated by the following formula:

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

The system metric value is obtained as:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*3) Procedure Consistency Measure*

The procedure consistency metric is measured by the following elements:

- I/O conventions
- Standard design representation
- Error handling conventions
- Calling sequence conventions

Where each of the above parameters is calculated by the following formula:

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*4) Operability Checklist*

This metric is measured by several parameters:

- It is described by all the operation steps
- Whether the appropriate description is made to the operator for all errors, conditions, and responses?
- Necessary provisions are made for the operator to obtain status, save, modify, interrupt and continue processing
- There should be a reasonable number of operator actions which is calculated as:

$$1 - \frac{time\ for\ operators}{total\ time\ for\ job}$$

- Description of job setup and tear down procedures
- Maintenance of hard copy log of interactions
- The responses standard and the consistent operator messages:

The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*5) Storage Effectiveness Measure*

It is measured by the following elements:

- Allocation of storage requirements to design
- Usage of virtual store facilities
- Common data is defined only once

$$1 - \frac{\#\ of\ variable\ defined\ more\ than\ once}{total\ \#\ of\ variables}$$

- Program segmentation

$$1 - \frac{maximum\ segment\ length}{total\ program\ length}$$

- Data segmentation

$$1 - \frac{amount\ of\ unused\ data}{total\ amount\ of\ data}$$

- Utilization of dynamic memory management
- Usage of data packing
- Free of nonfunctional code

$$1 - \frac{\#\ of\ non-functional\ statements}{total\ \#\ of\ statements}$$

- No code duplication

$$1 - \frac{\#\ duplicate\ \ statements}{total\ \#\ of\ statements}$$

- Storage optimizing compiler/assembly language used
- Data elements should be free from redundancy

$$1 - \frac{\#\ of\ redundant\ data\ elements}{total\ \#\ data\ elements}$$

The module metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

The system metric value is obtained as follows:

$$\frac{sum\ of\ storage\ efficiency\ measures\ for\ each}{number\ of\ modules}$$

*6) Training Checklist*

This metric is the collective measure of the following applicable elements:

- Development of lesson plans and training material provided to the operators, maintainers, and end-users
- Realistic simulated exercises
- Availability of diagnostic information and sufficient help online
- The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*7) Software Independence Measure*

This metric is measured by calculating the following factors:

- Dependency of software utility programs

$$1 - \frac{\#\ of\ programs\ -\ utility\ programs}{total\ \#\ of\ programs}$$

- Software library routines dependency

$$1 - \frac{\#\ of\ library\ routine\ used}{total\ \#\ modules}$$

- Usage of a common and standard subset of the language

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

- Free from operating system references

$$1 - \frac{\#\ of\ modules\ with\ OS\ References}{total\ \#\ modules}$$

The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*8) Conciseness Efficiency*

It is measured by:

$$V^* = (n1 + n2^*) \log_2(n1 + n2^*)$$

V* is confines efficiency

n1 is number of unique operators

$n2^*$ is minimal set of operands

*9) Accuracy Checklist*

It is measured by following steps.

- Is error analysis performed to a module?
- Is there a conclusive statement of requirement for output, input, constants, and processing accuracy?
- Is there an abundance of numerical methods?
- Is the execution of outputs within tolerance? The metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*10) Modular Design*

It is measured by:

$$\frac{expected\ number\ of\ modules\ changed}{total\ number\ of\ modules}$$

*11) Data and Control Flow Complexity*

Complexity is measured by using Cyclomatic complexity V (G) as proposed by McCable.

$$V(G) = edges - nodes - 2 * (connected\ components)$$

The system metric value is obtained as follows:

$$\frac{sum\ of\ complexity\ of\ modules}{number\ of\ modules}$$

*12) Design Structure Measure*

It is measured by the following steps:

- Designed organized in a top-down fashion
- There are no duplicate functions
- Modularity of model
- The module is not depending on the processing of previous steps

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

- The module has only one entrance and exit point.

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

- No global values

The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*13) Structured Language*

- Is Structured language used or not?

The system metric value is obtained as follows:

$$\frac{sum\ of\ modules\ scores}{number\ of\ modules}$$

*14) Halstead's Level of Difficulty Measure*

It is measured by :

$$\frac{unique\ operators}{2} + \frac{total\ operands}{unique\ operators}$$

*15) Coding Simplicity*

It is calculated by:

- Number of nesting level

$$\frac{1}{nesting\ levels}$$

- Number of branches

$$1 - \frac{branches}{\#\ of\ executables\ statements}$$

- The naming of variables being unique
- Is module self-modifying
- Variable density

$$1 - \frac{variables}{\#\ of\ executables\ statements}$$

- Number of jumps or go to statements

$$1 - \frac{\#\ of\ GOTO\ statements}{\#\ of\ executables\ statements}$$

The module metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

The system metric value is obtained as follows:

$$\frac{sum\ of\ simplicity\ of\ each\ module}{number\ of\ modules}$$

*16) Unit Referencing*

It is measured by the extent to which modules are referenced by other modules

The system metric value is obtained as follows:

$$\frac{number\ of\ common\ modules}{number\ of\ modules}$$

*17) Unit Implementation*

It is measured by the following factors:

- Is processing data value or volume-limited?
- Are machine-dependent and application functions mixed in the same module?
- Are I/O, Processing functions mixed in a single module?

Each of the above parameters is calculated by the following formula:

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

*18) Quantity of Comments*

It is measured by:

$$1 - \frac{\#\ of\ comments}{total\ \#\ of lines}$$

*19) Descriptiveness of Language*

This is measured by the following steps:

- Are variables the physical name of the function?
- Is one line containing one statement?
- Is code logically indented or blocked?
- Are keywords used as variables names?
- Is high-level language code used?

Each of the above parameters is calculated by the following formula:

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

*20) Effectiveness of Comments*

It is measured by:

- Do modules have proper comments existing of the module name, author, purpose, functions, inputs, outputs, references, etc.?
- Do comments just repeat what function does?
- Is machine code commented?
- Is an attribute of all variables commented? Each of the above parameters is calculated by the following formula:

$$1 - \frac{\#\ modules\ violate\ rule}{total\ \#\ of\ modules}$$

The system metric value is obtained as follows:

$$\frac{sum\ of\ scores\ of\ applicable\ elements}{number\ of\ applicable\ elements}$$

## IV. Observation And Results *(Heading 4)*

After the in-depth analysis, we have derived following observations. Table 2 provides the overview of each of the quality metric, group as quality criteria over multiple phases of software development. The seven phases of SDLC which are requirement gathering, design, Development, testing, operation and transition [19]. These 7 are grouped in 3 phases as Development which includes 3 initial phase, Evaluation phase based on testing, and operation which involves implantation and post implementation phases. [20]

TABLE II. Impact of Criteria on Lifecycle Phases

Where:

R = Requirement Analysis O = Operation

D =Design M = Maintenance

| Lifecycle phase / Criteria | Development | | | Evaluation | Operation | | |
|---|---|---|---|---|---|---|---|
| | R | D | C | S | O | M | T |
| Completeness | ● | ● | ● | × | × | | |
| Consistency | | ● | ● | × | × | | |
| Operability | ● | ● | ● | × | × | | |
| Conciseness | | | ● | | | × | × |
| Efficiency | | ● | ● | | × | | |
| Accuracy | ● | ● | ● | × | × | | |
| Modularity | | ● | ● | × | × | × | × |
| Training | | ● | ● | | × | | |
| Simplicity | | ● | ● | × | × | × | × |
| Software independence | | ● | ● | | | × | × |
| Generality | | ● | ● | | | × | × |
| Self-Documentation | | | ● | × | | × | × |
| Data commonality | ● | ● | ● | | | | × |

C = Code and Debug T = Transition

S = System Test.

● where criteria should be measure

× where impact of poor quality is realized

The above table 2 emphasis on the implementation or adoption of quality matrices in every phase of SDLC. Table 2 explains that which metric has to measure in which phase and if we do not measure or apply any metric, it will affect the product in later phases of SDLC.

Taking example of conciseness, if we do not check concusses and apply matrices in the coding phase, it will impact us in the maintainability and transition of product. Conciseness impact the size of the product code or SLOC, which if not minimized will result late run maintenance phase as the more SLOC, more hard it will be to deal with its maintenance.

Completeness and operability matrices must be applied in every phase of development. If failed to apply or meet certain requirements, the product will not pass any evaluation or it will fail in operation.

All the matrices must be met and measured in the respective phase as mentioned above. But not all matrices can be measured and met or improved simultaneously. There are always certain trade-offs which we have to adopt depending on the product and scenarios. The relation between these metrics which are mapped to quality factors is shown below

TABLE III. EFFECT OF CRITERIA ON SOFTWARE QUALITY FACTORS

| | Correctness | Reliability | Efficiency | Integrity | Usability | Maintainabilit | Testability | Flexibility | Portability | Reusability | Interoperabili |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Completeness | ✕ | ✕ | | | ✕ | | | | | | |
| Consistency | ✕ | ✕ | | | | ✕ | ✕ | ✕ | | ✕ | |
| Operability | | | ● | | ✕ | | | | | ✕ | |
| Conciseness | ✕ | | ✕ | | | ✕ | ✕ | | | | |
| Efficiency | | | ✕ | | | | ● | | ● | | |
| Accuracy | | ✕ | ● | | ✕ | | | | | | |
| Modularity | | | ● | | | ✕ | ✕ | ✕ | ✕ | ✕ | |
| Training | | | | | ✕ | | | | | ✕ | |
| Simplicity | ✕ | ✕ | ✕ | | | ✕ | ✕ | ✕ | ✕ | ✕ | |
| Software independence | | | ● | | | | | ✕ | ✕ | ✕ | ✕ |
| Generality | | ● | ● | ● | | | | ✕ | | ✕ | ✕ |
| Self-Documentation | | | ● | | | ✕ | ✕ | ✕ | ✕ | ✕ | |
| Data commonality | | | | ● | | | | | | ✕ | ✕ |

Where:

● Negative effect on the quality factor

✕ Positive effect on the quality factor

Referring to Table 3, we determine the effect of various criteria over the different software quality factors.

The completeness of the project positively impacts the quality of the software enhancing its correctness, reliability, and usability. However, it doesn't have much impact on the other quality factors.

Taking the consistency of the project into consideration, the more consistent the project development practices are, the more software improves in terms of correctness, reliability, maintainability, testability, flexibility, and reusability.

The operability criteria of the software adversely impact the efficiency of the software. The more operations there are in software the less efficient it gets. However, operability helps in enhancing the usability and reusability of the software.

The conciseness of the software improves the correctness, efficiency, maintainability, and testability of the software.

As for the efficiency criteria, the more efficient a software is, the harder it gets to test the software and maintain its portability thus negatively affecting the testability and portability of the software.

The accuracy criteria determine the reliability and usability of a software project and are in a direct relationship with them however, it impacts badly on the efficiency of the software since only one aspect of the software can be maintained at a time.

The increased modularity of the software harms the efficiency of the software. If software comprises sophisticated and smaller modules it becomes easier to maintain, test, and reuse the software and also has a positive impact on the software flexibility and portability.

The training criteria are in a direct relationship with the usability and reusability of the software, the more training given to a user the more the software usability increases.

The simplicity of a project helps improve the correctness, reliability, efficiency, maintainability, testability, flexibility, portability, and reusability of the software.

The independence of software over other modules helps to increase the flexibility, portability, reusability, and interoperability of the software. On the other hand, it makes the software less efficient.

The generality criteria reduce a software's reliability, efficiency, and integrity of software while increasing the flexibility, reusability, and interoperability of the system.

Self-documentation is a criterion that adversely affects the software's efficiency while increasing its maintainability, testability, flexibility, portability, and reusability.

Finally, data commonality is a basic criterion that helps in enhancing the reusability and interoperability of the software while reducing its integrity.

## V. CONCLUSION

The primary focus of this research was to study the impact of quality measures and formulas in achieving better software quality. The research was carried out by comparing the software quality factors described in McCall model with the quality criterion. The findings of this research constitutes that many of the quality metrics have a positive impact on the lifecycle of the software, this lifecycle includes different phases like development, testing, operations. The software metrics are measured in these different phases according to their impact on the overall software.

The types of metrics are defined in detail which further help in the calculations and measurements. While measuring the software metrics certain trade-offs have to adopted considering the product and its working environment. This research also discusses the impact of individual criterion on the factors, in order to show a proper view and to get knowledge about the results and experiments used for quality goal.

## ACKNOWLEDGMENT

We express our gratitude to the people who participated in the surveys and interviews conducted for this research and for giving their valuable time for sharing their experiences

with us. We managed to do our research through their valuable contribution. We would also like to express our gratitude to our supervisor **Miss Sana Fatima** for her recommendations and guidance since the very beginning and her never ending support.

## REFERENCES

[1] M. Agnihotri and A. Chug, "A Systematic Literature Survey of Software Metrics, Code Smells and Refactoring Techniques," Journal of Information Processing Systems, vol. 16, no. 4, pp. 915-934, 2020.

[2] N. Fenton and J. Bieman, Software metrics: a rigorous and practical approach, CRC Press, 1991.

[3] K. J. Padmin, H. M. N. D. Bandara and I. Perera, "Use of Software Metrics in Agile Software Development Process," in MERCon 2015 - Moratuwa Engineering Research Conference, 2015.

[4] R. Fitzpatrick, "Software quality: definitions and strategic issues," Dublin, 1996.

[5] N. E. Fenton and M. Neil, "Software metrics: successes, failures and new directions," Journal of Systems and Software, 1999.

[6] J. Rashid, T. Mahmood and M. W. Nasir, "A Study on Software Metrics and its Impact on software quality.," Technical Journal, University of Engineering and Technology (UET) Taxila, Pakistan, vol. 24, no. 1, pp. 1-14, 2019.

[7] N. U. Eisty, G. K. Thiruvathukal and J. C. Carver, "A Survey of Software Metric Use in Research Software Development," in IEEE 14th International Conference on eScience (e-Science), Amsterdam, 2018.

[8] S. Reyaz and D. Ranjan, "STUDY OF VARIOUS QUALITY METRICS SUITABLE FOR THE OBJECT ORIENTED ENVIRONMENT," International Journal of Technical Research and Applications, vol. 6, no. 2, pp. 92-97, 2018.

[9] F. N. Colakoglu, A. Yazici and A. Mishra, "Software Product Quality Metrics: A Systematic Mapping Study," in IEEE Access, vol. 9, pp. 44647-44670, 2021

[10] M. Maddox and S. Walker, "Agile Software Quality Metrics," 2021 IEEE MetroCon, 2021, pp. 1-3

[11] M.-C. Lee, "Software Quality Factors and Software Quality," British Journal of Applied Science & Technology, vol. 4, no. 21, pp. 3070 -3095, 2014.

[12] T. Mladenova, "Software Quality Metrics – Research, Analysis and Recommendation," in International Conference Automatics and Informatics (ICAI), 2020.

[13] T. R. Vanitha N., "A Report on the Analysis of Metrics and Measures-A Literature Study," International Journal of Computer Science and Information Technology, vol. 5, p. 5, 2014.

[14] D. A. Wahab, E. B. Setiawan and R. Wahdiniwaty, "Comparative Analysis of Software Quality Model In The Selection of Marketplace E-Commerce," in 2018 International Conference on Information Technology Systems and Innovation (ICITSI), Bandung, 2018.

[15] D. Singh and N. B. Kassie, "User's Perspective of Software Quality," in 2nd International conference on Electronics, Communication and Aerospace Technology, 2018.

[16] S. Chouksey, S. Gupta, A. Pandey and S. Rao, "Software Quality Metrices," International Journal of Engineering Research & Technology (IJERT), vol. 6, no. 4, pp. 642-647, 2017.

[17] Birhanu, Ermiyas, "Analysis of Software Quality Using Software Metrics," *International Journal on Computational Science & Applications*, vol. 8, pp. 11-20, 2018.

[18] H. Mumtaz, M. Alshayeb, S. Mahmood, en M. Niazi, "An empirical study to improve software security through the application of code refactoring", *Information and Software Technology*, vol 96, pp. 112–125, 2018.

[19] O. J. Okesola, A. A. Adebiyi, A. A. Owoade, O. Adeaga, O. Adeyemi, en I. Odun-Ayo, "Software Requirement in Iterative SDLC Model", in *Intelligent Algorithms in Software Engineering*, pp. 26–34, 2020.

[20] N. B. Ruparelia, "Software Development Lifecycle Models," SIGSOFT Softw. Eng. Notes, vol. 35, no. 3, p. 8– 13, 2010.