Software Engineering
2021; 9(2): 36-44
http://www.sciencepublishinggroup.com/j/se
doi: 10.11648/j.se.20210902.11
ISSN: 2376-8029 (Print); ISSN: 2376-8037 (Online)

Quantifying Software Quality in Agile Development Environment

Ikerionwu Charles*, Nwandu Ikenna Caesar

Department of Software Engineering, Federal University of Technology, Owerri, Nigeria

Email address:
charles.ikerionwu@futo.edu.ng (I. Charles)
*Corresponding author

To cite this article:
Ikerionwu Charles, Nwandu Ikenna Caesar. Quantifying Software Quality in Agile Development Environment. *Software Engineering*. Vol. 9, No. 2, 2021, pp. 36-44. doi: 10.11648/j.se.20210902.11

Received: June 30, 2021; **Accepted**: July 20, 2021; **Published**: August 24, 2021

Abstract: Due to required efforts and the challenges involved in understanding the quantification of software quality, researchers have chosen varying quality attributes to describe the quantification of software quality. The degree of software quality is achieved from the standards and quality attributes at each development process: the adherence of software engineering principles towards realizing a product of good quality. In agile environment, the software engineering process ensures that qualities of interest are built-in and to produce software product with an acceptable level of quality. Thus, this study is aimed at quantifying six related software quality attributes. The specific objectives include identifying the software quality attributes, the design of the algorithm for measurement metrics, and to perform relational analytics of each attribute with respect to the software quality. The methodology followed an exploratory evaluation of measurement and metrics and their role in quantifying software quality in agile development environment. The study adopted existing metrics to quantify software quality attributes. Twelve opensource software projects were tested for 6 specific quality attributes and each result is quantified and presented. Results show that software number 2 (SW2) has a maintainability value of 6 minutes, 50% availability, and 0.62 reliability values. It implies that a high value of maintainability does not translate to high reliability. These values establish the relationship between attributes and enhances developers and users' understanding of the software quality and its attributes.

Keywords: Software Quality, Measurement, Metrics, Attributes, Quantification, Agile

1. Introduction

The agile framework emphasised on continuous delivery of quality applications through object-oriented modelling and universal modelling language but silent on the process of quantifying the quality of the ensuing software. The quality of software depends on the cumulative development iterations within a process, from which it evolved [1]. Software development process indicates that the software quality could not be engineered within a single iteration in a process. It implies that building-in quality is not a stopgap approach. Many researchers, over the years, commonly suggest that, to build-in acceptable level of quality in software, each development process must adhere to its standards and quality attributes [2, 3]. This implies that established software quality control inputs should be performed in the engineering process to ensure that the final software product has an acceptable level of quality. Throughout the development process, an objective assessment to determine whether quality requirements are being met should be performed. In so doing, a quantitative assessment of quality could provide the basis for decisions regarding the software's fitness for use. Table 1 presents quality control activities administered during each phase of the development processes.

While emphasising on the quality of the product, [4] posits that, "software program that repeatedly and frequently fails to perform expectedly, it matters little whether other quality measures at different levels met the respective standards". In an Agile environment, it is on this premise that software engineers and their customers need to device a consistent pattern or mechanism to communicate the purpose of the

ensuing system, constrains that must be addressed, the risks to be managed, the design and implementation strategy [5]. These, in effect could improve the overall software quality.

Determining software quality includes adhering to quality standards at requirements specifications, software project management, software design, quality assurance, and testing [6]. At each level, quality is calculated through a quantitative measure of the degree to which it possesses a given quality attribute (IEEE software quality). Thus, the reliability of a software product is summed up from the overall quality. In addition, to measure the quality of the software product, [7] suggest four-step procedures: define the quality to be considered, state the thresholds for the quality metrics, list information for the measurement, and finally, measure and evaluate the quality metrics based on the thresholds. These steps further confirm that the requirements are complete and consistent, design is standardized, and consistent, and key appropriate developmental tools are used.

Table 1. *Quality control activities in agile development process.*

S/no.	Development Process	Quality control
1	Requirement specification	Clear, unambiguous, consistent, and easy to understand.
		Clear definition of components.
2	Architecture design	Clear procedural structure of the system.
		Clear relationship between components.
		Minimal lines of code (LOC)
3	Programming (coding)	Less complex system
		High level of Structure with software specification
		Design a test plan
4	Testing	Development testing
		Unit, components, and system testing.
		Review test plan
5	Validation	Validate with user and system requirements, respectively.
		COCOMO
6	Project management	Risk Management
		Meets customer expectation
		Coherent and well-functioning development team.

As the need for software products to solve our increasing problem increases, the development process becomes complex, and software engineers strive to provide clients with software products with an acceptable level of quality [8]. To achieve clients' demand in the prevailing complexities, software engineers are always looking for ways to improving the engineering process and the quality attributes. These improvements are better understood when they are measured in numbers and in effect, [9] affirmed the assertion that software quality measurement is an essential component of software engineering.

However, in practice, quantification of software quality has fallen short of detailed explanation especially in organisations where it has not been a norm from the inception [5]. To reduce this ambiguity and improve on the quality, [5] is of the view that stakeholders should clearly define system requirements and present understandable attributes. These attributes must be quantifiable so that progress towards the goal of the system can be evaluated competitively. With quality in focus, communication among stakeholders at different levels such as – specification, design, project management, and process must be continuous. In conclusion, [5] suggests that effective communication amongst stakeholders at all levels could lead to successful projects.

Asthana A. and Olivieri J. focused on quantifying the reliability (a single attribute of software quality) of software application at the time of product shipment by adopting a model that combined the product and development process parameters to determine the level of the reliability [10]. In addition to the system approach to identify software product goodness, the study included the display of measurable targets for each identified metric index in a dashboard during the development process. In a related research, [11] implored static metric approach to record the reusability (another software quality component) of software application. The study used an open dataset from the Github and assessed the number of reuses for each component based on five different properties: complexity, cohesion, coupling, inheritance, documentation, and the size of the system. The findings suggest that the developed static model could effectively predict the reusability as appraised by the software developers. In all these research efforts, the focus has been the emphases on one attribute of the software quality, but this study has made an aggregate of quantifying software quality attributes and establish each attributes effect on the overall software quality.

Therefore, this study is aimed at quantifying the related attributes that contribute to the software quality. Further, it identifies specific objectives that would lead to realising the aim of the study: identify software quality attributes, design an algorithm for measurement metrics, and perform relational analytics of each attribute with respect of the software quality. The rest of the study is structured as follows: section 2 presents the correlation between measurement and metrics; section 3 presents the related works to this study; section 4 is the methodology; and section 5 presents the results of the study and the conclusion drawn from them.

2. Correlation Between Measurement and Metrics

Agile software development method has demonstrated software quality determining features and emerged as a preferable method over some of the established orthodox development methods [12]. Key characteristics is the emphasis on the products over individuals, and a clear approach to quality quantification – where measurement and metrics play very important roles. [9] posit that "measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way to describe them according to clearly stated standards". Specifically, measurement captures the information about specific attributes in numbers. Both practitioners and academicians have shown that quantification of software quality attributes are calculated through metric and measurement. Software metrics are functions (formulas) used to compute the values, while measurements are the numbers calculated using the function.

Table 2 shows a list of six different software quality attributes and their standard measurements. Measurement is specific and targeted towards specific software attributes. The process of measurement puts the product's performance under check towards achieving the goals and objectives of the product. [13] defines attribute as, "a measurable physical or abstract property of an entity". Further, an attribute underlines the characteristics of interest inherent in a software entity and when measured, specific attributes help to distinguish one entity from another. The standard of measure that ascertains how much a software system possesses the distinguishing characteristics is termed metrics. A metric is a measurement function aimed at quantifying a software property/characteristic. This is a clear indication that software metrics is all about measurement which in turn involves numbers. We, therefore, posit that the relationship between metric and measurement lies or rather converge in the quantification process of software quality. This is in line with the [13], which states that, "each quality factor is a direct metric that serves as a quantitative representation of a quality factor." Similarly, [14] agreed that "the quantification of software characteristics is affected using software metrics". This close tie between measurement and metric allows for the classification of quality measurements into three classes:

Product metrics: These are predictor metrics used to quantify internal attributes of a software product [3]. For example, the size of the system – measured in lines of code (LOC), complexity of the software product (depth and quantity of routine in a program), the number of modules associated in the system, design features and cumulative quality. Product metrics is further categorised into two specific classes: dynamic and static metrics. Dynamic metrics are measurements made in programs in execution. They are the class of software metrics that presents the dynamic behaviour of a program in execution [15]. For example, the time taken to compute a given task, the number of bugs reported in a predetermined period etc. According to [16], static metrics are measurements inferred from the software to express its characteristics, which include size of the software (expressed in lines of code [LOC]), cyclomatic complexity, fog-index etc.

Process metrics: It measures the progress of the software development process as well as the various characteristics of the inserted techniques such as the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software. This implies that process metrics provides an organisation and the development team a strategic review of the entire development process to ascertain the effectiveness of the adopted process.

Project metrics: It quantifies the management of the software development process purposely to guide the manager and the development team in assessment of the process resources and its impact. Through project metrics, expected period of delivery, costs estimation, project schedule, man-hour and all deliverables are calculated. However, [4] suggests that metrics collected from earlier projects should be used as a benchmark to calculate effort and time duration of the current project.

3. Related Works

Quality measurement and evaluation are key aspects in software products delivered to users, and in effect, researchers have adopted diverse methods of quantifying software quality. As a result, ISO/IEC JTC1/SC7 developed series of ISO/IEC 25000 standards which are designed to standardize the quality measurement and evaluation of software products. Consequently, [17] critically analysed and adopted the methodological system within the ISO/IEC 25000 series to quantify software products based on the defined benchmark and preference as key requirements in the missing weighting method in the methodological system. Some previous research efforts are known to focus on a single software quality attribute to quantify the software product. For example, maintainability- a feature of code metrics and reusability is evaluated to ascertain the level compliance to established standards. [18] considered a set of software metrics and reference benchmarks to evaluate maintainability, as a quality attribute using a declarative Query/View/Transformation (QVT) Relations language. Results suggest that, from the onset, the implementation of the model at the architectural level would improve the software maintainability.

Software measurement activity usually considers certain restricting factors, such as the characteristics of a product user (e.g., experience, age, gender, etc.), the type of tasks being performed by the user, the environmental study (ranging from controlled laboratory conditions to largely unstructured field studies), as well as the nature of the evaluation object, which can be a paper prototype, a software mock-up, a partially functional prototype or an

accomplished system [19]. [20] studied several evaluation schemes for workflow products but majored on web-based products with multiple users' access and concluded that five factors are requisites for a successful evaluation. These include a) accuracy, which presents the author's profile; b) authority - provides specific authors' credentials and link to the published documents; c) objectivity – discusses the authors' opinion; d) currency, discusses the process of software update and e) coverage that indicates the cost of the software. Determination of software capability or usability informs achieving users' requirements and goals; and through evaluation method, the software quality could be quantified. Earlier efforts have shown the adoption of different methods to determine usability as an attribute of software quality and through evaluation, quality is quantified [21]. Further evaluation to determine software capability has been narrowed down by ISO/IEC25000:2011 as the "extent to which a product can be used by specific users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". [21] concluded that usability, as presented in extant literature shows that usability in inbuild at the design -phase of software development life cycle and accounts for 71% of the reviewed work. Both individual users and industries have shown preference to applications that are easily maintainable and reliable, however, they have faced challenges with the choice of available metric frameworks. The existing gap in maintainability evaluation metrics spans across false – positive and high level of complexity in the computing and quantifying the maintainability attribute. Thus, [22] developed a comprehensive cloud-based automated infrastructure framework to address the identified gaps. Due to its efficacy, the framework named SQUAAD has been used in empirical studies and government related parastatals.

Misra, S., Akman designed a framework meant to analyse whether software metric qualifies as a measure of quality from different perspectives [23]. The study adopted the framework by using cognitive functional size measure (CFS) for the purpose of evaluation and validation of software complexity measurement. The findings suggest that the framework is a better way to effectively present the software parameters required to evaluate and validate the complexity measurement. To demonstrate the interaction amongst components that define the software quality, [24] developed quantitative evaluations by considering interactions among these components in a multi-criteria decision-making (MCDM) problem. The evaluation process adopted the aggregator method of arithmetic mean (AM) and weighted arithmetic mean (WAM). Findings emanating from the study were ranked in six main quality attributes as identified by the ISO 25000 standard.

Table 2. *Software quality attributes and standard measurements.*

Attributes	Standard measures	Citation
Maintainability	Depth of inheritance tree. Ease of modification and mean-time to debugging. Component independence. Measure of how hard or easy it is to maintain a software	Somerville (2016) Fenton and Bieman (2014) Lu et al. (2016)
Usability	Length of users' manual. Number of error messages. Capability to be understood, learned, and used (ISO/IEC 9126-1, 2000)	Abran et al. (2003) Khan et al. (2018)
Reliability	Cyclomatic complexity Programs' lines of code	Ikerionwu (2010) Yamada (2014) Fenton and Bieman (2014)
Reusability	Amount and frequency of reuse. Portability, adaptability- level with ease of platform independency Flexibility, modularity, and understandability Cohesion and documentation	Hristov et al. (2012) Ampatzoglou (2018) Gui and Scott (2009)
Efficiency	Device efficiency Accessibility Correctness	Sitaraman and Weide (2014) Watro (2014)
Testability	The extent a unit or module support its testing Modularity Ease of detecting cause of failures	Garousi et al. (2018) Huda et al. (2015) Khan et al. (2016)

4. Methodology

We considered each of the identified quality attributes and independently used available functions (metrics) to derive its measurement. There are 12 software applications, referred to as "software projects" and denoted as SW, which are subjected to quality quantification. These software projects are identified as: SW1……SW12. Therefore, in this section, quality attributes are identified, quality metrics are explained, measurements are derived, and each attribute is calculated and quantified. These values are contained in table 5.

It is essential to represent the attributes of software products in numeric or symbolic terms for the purpose of quality quantification. This is achievable by employing relevant software metrics in the process of measuring the quality attributes of these software products. This section discusses the quality attributes and corresponding metrics applied to our project models (as summarized in table 3) and their outcomes (as shown in table 4 and figure 1).

Table 3. *Software quality attribute metrics.*

Quality Attribute	Metric
Maintainability	MTTR= $\frac{total\ downtime}{number\ of\ outages}$
Availability	Availability = $\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$ *100%
Reliability	R= $\frac{1}{MTTF}$
Portability	Portability = (Number of successful ports) / (Total number of ports) * 100
Testability	Tm=k *VC and Tc=min (Tm)
Reusability	Rc_c= w_1.mai + w_2.ada + w_3.doc + w_4.com + w_5.ava

Each of these metrics is applied to calculate the numeric values of the six quality attributes of the twelve software projects listed in table 5. These attributes are maintainability, availability, reliability, portability, testability, and reusability.

4.1. Maintainability

Maintainability is an attribute of software quality and has been described as the level of ease with which a software can be maintained or modified [25]. Such maintenance includes debugging, modification, and extension of functionality to adapt to new environment. The possibility of maintenance and modification depends on the extent to which a software is readable, understandable and its extensibility (RUE).

In the expression i, we establish the relationships in terms of proportionality between these components - RUE:

if
{
Maintainability=a
Constant = k
Readability=b
Understandability=c
Extensibility=x
then,

$$a \propto k(bcx) \tag{1}$$

Equation 1 denotes that a high degree of maintainability implies a high degree of RUE attributes. The maintainability of software product is measured as the mean time to repair (MTTR).

$$\text{MTTR}= \frac{total\ downtime}{number\ of\ outages} \tag{2}$$

We defined a maintainability period of 60 days where different application software is used and performed varying measurements using different operating systems that include software (MacOS, Windows, and Linux) and hardware ports. The calculated MTTR is presented in table 5.

4.2. Availability

Availability of software implies its readiness to be put in use. Availability quantifies the likelihood of the software to be in use over a specific period. A software's availability is proportional to its reliability. That is why it is usually considered as a measure of software reliability. It is measured as a relation of the software's mean time to failure (MTTF) and its mean time to repair (MTTR):

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} *100\% \tag{3}$$

$$\text{where MTTF} = \sum_{i=1}^{ne} \frac{t_{i+1} - t_i}{n_e - 1} \tag{4}$$

t_i - t_{i+1} = Execution time,
n_e is the number of failed executions due to error.

4.3. Reliability

The reliability of software is concerned with the probability of the software system to perform a function correctly for a specified number of inputs within a specified time interval. Reliability measures the probable non-failure of a software system to execute its intended functions over a specified period. The reliability of a software is much inclined to the maturity, fault tolerance and recoverability of the software. It is usually measured as the inverse of the software's mean time to failure (MTTF). The calculated measurement is recorded in table 5 under reliability column.

$$\text{R}= \frac{1}{\text{MTTF}} \tag{5}$$

4.4. Portability

The portability of software focuses on the ability of the software to operate successfully in all platforms - i.e., working in different environments. This is in line with ISO/IEEE 24765, which presents portability as the ease with which a system or component can be transferred from one hardware or software environment to another. Both the hardware and software requirements are considered non-functional requirements. The measure of portability encompasses installability, adaptability, replaceability, and compatibility, but in this study, the researchers limited their scope within the overall measure of portability at the implementation level. We used an open-source application software to model its portability. The algorithm towards the measure of portability involves the following steps:

i. Record total number of ports (e.g., browser and version, operating system and version, programming language, processor make and speed, software modules (units) etc.) available within the environment.
ii. Identify the total number of successful ports within the environment to measure the portability.

iii. Apply the metric to measure the portability.
iv. Repeat steps i to iii for a different application software.
v. Repeat steps i to iii for a different application software in a different environment (operating system).
vi. End.

In table 4, twelve software projects were adopted to test and quantify their respective portability values. Using the successful and total ports in table 4, the values of portability are derived and presented in table 5.

Portability = (Number of successful ports) / (Total number of ports) * 100

Respectively, the ports include:

Table 4. *Portability ports for the 12 projects.*

Successful Ports	22	15	10	12	30	8	10	16	18	7	8	13
Total ports	25	20	11	20	30	14	12	17	18	12	9	15

The successful ports record the total number of ports available from the operating system and hardware system that are compatible with the application software being tasted for portability. The second row is the total number of ports available within the software and hardware environments which are available for use. When the SW is plugged-in, some of these ports were incompatible with the environment and the compatible ones are recorded as successful ports. Thus, these numbers are used in calculating the numeric value of portability as one of the quality attributes.

For instance, to calculate the portability value of SW1, we adopt the portability metric:

$$\text{portability} = \frac{22}{25} * 100 \quad (6)$$

$$= 88$$

Table 5. *Results of Metrics Application on Model Projects.*

Projects	Maintainability (minutes)	Availability (%)	Reliability	Testability	Portability	Reusability
SW1	1	75	1	Observability Simplicity Modularity Stability (4)	88	2.725
SW2	6	50	0.62	Modularity Stability (2)	75	3.55
SW3	1	70	1	Observability Modularity (2)	90.9	2.91
SW4	3	50	0.63	Observability Stability (2)	60	2.85
SW5	1	73	1	Observability Simplicity Modularity (3)	100	2.319
SW6	4	50	0.66	Observability Modularity (2)	57.1	3.25
SW7	3	70	0.95	Observability Simplicity Modularity (3)	83.3	3.11
SW8	6	75	1	Observability Simplicity Modularity (3)	94.1	3.525
SW9	1	75	1	Observability Simplicity Modularity (3)	100	2.925
SW10	3	50	0.56	Modularity Stability (2)	58.3	2.95
SW11	2	77	0.98	Observability Simplicity Modularity (3)	88.8	2.754
SW12	3	70	0.96	Observability Simplicity Modularity Stability (4)	86.7	2.81

In table 5, there are twelve software projects (SW1 …. SW12) that are tested for software quality under the following attributes: maintainability, availability, reliability, testability, portability, and reusability. The calculated values using respective software quality metrics are presented in table 5.

4.5. Testability

The identification and quantification of testability of the software project is based on five key testability metrics - observability, simplicity, modularity, and stability. Each of these is identified as one metric and appropriately indicated in table 5. For clarity, four (4) is the highest assigned number and shows that the software possesses all the four metrics - testability metrics:

4 indicates the software possesses four of the testability metrics.
3 indicates three metrics.
2 shows the software contains two metrics and,
1 indicates only one metric.

i. Observability: What is observable is testable. This focuses on the observable states and features affecting the output of the software. [26] argues that "software behaviour states is the ability of the test system to observe the outputs of the states and to determine which input triggers a particular output".
ii. Simplicity: The simplicity metric measures the ease of performing a test on any software. It implies that little efforts are required in testing the software product. These eases are dependent on functional, structural and code simplicity.
iii. Modularity: Modularity is a key metric found in object-oriented design and it has made software developed with object-oriented programming language easier to maintain. Each module could be decoupled, edited based on client requirements, maintained, or reused in another development [16].
iv. Stability: A stable software has fewer number of changes, which makes it easier to test. Stability is attained when the software does not require frequent alteration, but when it is inevitable, such changes should be measured and communicated clearly.

Empirically, testability is usually obtained using the following relations:

$$\text{Testability of method (Tm)} = k * VC, \quad (7)$$

where VC = Visibility

$$\text{Component} = {Possible\ Output}/{Possible\ Input} \quad (8)$$

Testability of the class (Tc) = min (Tm)

4.6. Reusability

Software reusability describes the ease with which software components can be used in different modules to develop new applications. In other words, if some components in a particular software are used to develop a different system with different functionality, it is said to be reusable. Reusability usually results in realizing high quality products. It is measured as:

$$Rc_c = w_1.mai + w_2.ada + w_3.doc + w_4.com + w_5.ava \quad (9)$$

Where:

$w_1 – w_5$ are weights and the rest are composite metrics for the attributes from the reusability measurement model. The values of the weights determine the importance of each characteristic of the component for its reusability and are determined empirically or through expert opinion. According to [27], the sum of the components of interest is equal to 1.

mai= maintainability (adjustability to higher versions of software)

ada= adaptability (programming language, adapters, appropriate methods, and interfaces)

doc= documentation (amount, quality, completeness, existence of legal terms and conditions)

com= complexity (size, coupling, cohesion, amount and complexity of methods and parameters)

ava= availability (instant, upon search, upon request, unavailable)

5. Results and Discussion

The values in table 5 are plotted in a bar chart to vividly express the relationships between different quality attributes. It is observed that through quantification, software quality when broken down to varying quality attributes indicate the lack of software product with 100% quality. The attributes vary, but the software when used provides an acceptable level of quality to meet the user requirements. For example, SW2 has the following values of quality attributes: maintainability= 2 minutes; availability = 50%; reliability = 0.62; testability = 2; portability=75; reusability=3.55.

The high reliability value of 3.55 in SW2 is translated from the modularity feature in testability attribute. This is in line with [25] assertion that modularity is a quality attribute that is essential for software reuse. Similarly, SW2 has a reliability value of 0.62 but an optimum availability value of 50%.

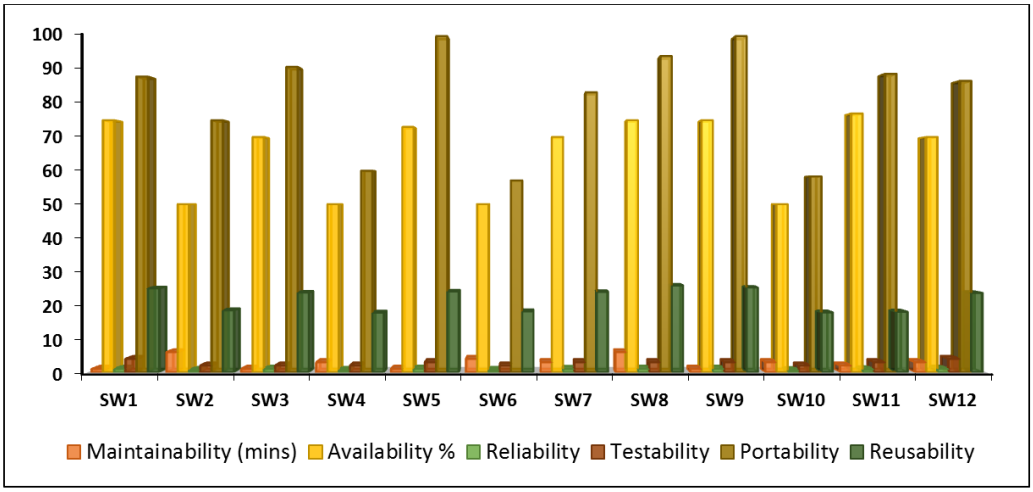

Figure 1. *Metric values of quality attributes.*

The numbers plotted in figure 1 were derived from tables 4 and 5 to quantify each identified quality attribute. Overall, there is high level of portability without a direct effect on the overall quality of the software system. Noticeable effect is the direct relationship between reliability and portability, which are directly proportional to each other. A high value of maintainability does not translate to high reliability. Similarly, maintainability is inversely proportional to availability, i.e., when software takes longer time to repair, the period of availability is significantly reduced. It is observed that the quantification of these quality attributes provides easy understanding of software quality for the software users, software project managers and developers.

6. Conclusion

Software quality is subjective, i.e., what level of user's requirements did it satisfy, and would be better understood when the concept of quality is quantified. By expressing software quality in numbers derived from metrics and measurements, both developers and users could easily comprehend the level of quality possessed by a software product. In this study, using open-source software (SW1, SW2, SW3, SW4, SW5, ……., SW12), 6 quality attributes were identified, examined and measured by adopting specific function (metric) to express the quality in numeric value.

These software measurement activities were identified to have explicit impact on performing software quality benchmarking, establishing software quality assurance as well as setting software quality objectives. Concisely, a user could easily indicate the level of software quality when each quality attribute is expressed in numbers. Through calculated quality numeric values, the study established specific relationship between each quality attribute and how it affects the overall quality of the software system. Earlier research efforts have focused on quantification of a single quality attribute [28-31], but this study demonstrated the quantification of key software quality attributes that summarises the quality of the software. This paper, therefore, laid emphasis on the characteristics of software products that can unleash significant information about the quality of the products. Furthermore, an overview of software attributes measurement together with relevant quality evaluation metrics were made. It presented the significant correlation of the process of measurement and metric usage in quantifying software quality. Further research could focus on how quality is built-in at every level of the software engineering process using agile methodology.

References

[1] Ikerionwu, C., Foley, R., & Gray, E. (2014). Improving software quality in the service process industry using agility with software reusable components as software product line: An empirical study of Indian service providers. *International Journal of Advances in Engineering & Technology*, *7* (3), 701.

[2] Boehm, B., Lane, J. A., Koolmanojwong, S., & Turner, R. (2014). *The incremental commitment spiral model: Principles and practices for successful systems and software*. Addison-Wesley Professional.

[3] Sommerville, I. (2015). Software engineering. 10th. In *Book Software Engineering. 10th, Series Software Engineering*. Addison-Wesley.

[4] Pressman, R. S. (2015). Software Engineering: A Practitioner's Approach, 2000.

[5] Gilb, T. (2005). Competitive engineering: a handbook for systems engineering, requirements engineering, and software engineering using Planguage. Elsevier.

[6] Gorla, N., Somers, T. M., & Wong, B. (2010). Organizational impact of system quality, information quality, and service quality. *The Journal of Strategic Information Systems*, *19* (3), 207-228.

[7] Nakai, H., Tsuda, N., Honda, K., Washizaki, H. and Fukazawa, Y. (2016). A SQuaRE-based Software Quality Evaluation Framework and its Case Study. IEEE International Conference on Software Quality, Reliability & Security. 1-4.

[8] Ikerionwu, C., Gray, E., & Foley, R. (2013, September). Embedded software reusable components in agile framework: The puzzle link between an outsourcing client and a service provider. In *Quality comes of age, The BCS Quality Special Group's Annual 21st Software Quality Management (SQM) Conference, ISBN–987-0-9563140-8-6* (pp. 63-78).

[9] Fenton, N. & Bieman, J. (2014). *Software metrics: a rigorous and practical approach*. pp. 22-34 CRC press.

[10] Asthana, A. & Olivieri, J. (2009). "Quantifying software reliability and readiness," 2009 IEEE International Workshop Technical Committee on Communications Quality and Reliability, Naples, FL, 2009, pp. 1-6, doi: 10.1109/CQR.2009.5137352.

[11] Papamichail, M. D., Diamantopoulos, T., & Symeonidis, A. L. (2019). Measuring the reusability of software components using static analysis metrics and reuse rate information. *Journal of Systems and Software*, *158*, 110423.

[12] Kupiainen, E., Mäntylä, M. V., & Itkonen, J. (2015). Using metrics in Agile and Lean Software Development–A systematic literature review of industrial studies. *Information and Software Technology*, *62*, 143-163.

[13] IEEE, "IEEE Std. 1061-1998, Standard for a Software Quality Metrics Methodology, revision." Piscataway, NJ: IEEE Standards Dept., 1998.

[14] Tahir, A. (2015). A Study on Software Testability and the Quality of Testing In Object-Oriented Systems. University of Otago.

[15] Gunnalan, R., Shereshevsky, M and Ammar, A. (2005). Pseudo dynamic metrics [software metrics], The 3rd ACS/IEEE International Conference on Computer Systems and Applications, Cairo, pp. 117.

[16] Ikerionwu, C. (2010). Cyclomatic Complexity as a Software Metric. *International Journal of Academic Research*, *2* (3).

[17] Liu, X., Zhang, Y., Yu, X., & Liu, Z. (2018, June). A software quality quantifying method based on preference and benchmark data. In *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (pp. 375-379). IEEE.

[18] Kapová, L., Goldschmidt, T., Becker, S., & Henss, J. (2010, June). Evaluating maintainability with code metrics for model-to-model transformations. In *International Conference on the Quality of Software Architectures* (pp. 151-166). Springer, Berlin, Heidelberg.

[19] Gediga, G., Hamborg, K. and Düntsch, I. (2015). Evaluation of Software Systems, Institut für Evaluation und Marktanalysen Brinkstr. 19, 49143 Jeggen, Germany, pp 1-5.

[20] Stoilova, K. and Stoilov, T. (2005). Software Evaluation Approach. Institute of Computer and Communication Systems – Bulgarian Academy of Sciences, pp 1-6.

[21] Sagar, K., & Saha, A. (2017). A systematic review of software usability studies. *International Journal of Information Technology*, 1-24.

[22] Behnamghader P., & Boehm B. (2019) Towards Better Understanding of Software Maintainability Evolution. In: Adams S., Beling P., Lambert J., Scherer W., Fleming C. (eds) Systems Engineering in Context. Springer, Cham. https://doi.org/10.1007/978-3-030-00114-8_47

[23] Misra, S., Akman, I. and Colomo-Palacios, R. (2013). A Framework for Evaluation and Validation of Software Complexity Measures, Department of Computer Engineering, Atilim University, Ankara, Turkey, 1-27.

[24] Alashqar, A. M., Elfetouh, A. A. and El-Bakry, H. M. (2015). ISO 9126 Based Software Quality Evaluation Using Choquet Integral. International Journal of Software Engineering & Applications (IJSEA), 6 (1), 111-121.

[25] Boehm, B. (2017, January). Evaluating Human-Assessed Software Maintainability Metrics. In *Software Engineering and Methodology for Emerging Domains: 15th National Software Application Conference, NASAC 2016, Kunming, Yunnan, November 3–5, 2016, Proceedings* (Vol. 675, p. 120). Springer.

[26] Liu, P. (2017). Testability Metrics for Software Behavioral Models. *International Journal of Performability Engineering*, *13* (8).

[27] Hristov, D., Hummel, O., Huq, M., & Janjic, W. (2012). Structuring software reusability metrics for component-based software development. In *Proceedings of Int. Conference on Software Engineering Advances (ICSEA)* (Vol. 226).

[28] Ardito, L., Coppola, R., Barbato, L., & Verga, D. (2020). A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review. *Scientific Programming*, *2020*.

[29] Kang, H. G., Lee, S. H., Lee, S. J., Chu, T. L., Varuttamaseni, A., Yue, M., ... & Li, M. (2018). Development of a Bayesian belief network model for software reliability quantification of digital protection systems in nuclear power plants. *Annals of Nuclear Energy*, *120*, 62-73.

[30] Rizvi, S. W. A., Singh, V. K., & Khan, R. A. (2016). Fuzzy logic based software reliability quantification framework: early stage perspective (FL SRQF). *Procedia Computer Science*, *89*, 359-368.

[31] Chen, C., Alfayez, R., Srisopha, K., Shi, L., & Boehm, B. (2016, November). Evaluating human-assessed software maintainability metrics. In *National Software Application Conference* (pp. 120-132). Springer, Singapore.