An Analysis of the Software Quality Metrics from Consumer Satisfaction Perspective

Najla Mohammed A Althuniyan¹, Maygualida Sanchez Flores², Maedeh Pishvaei, Hussein Altabrawee²

Department of Computer Science, Computer & Engineering College
California State University - Fullerton

Abstract
Software products are related to every aspect of our modern life just like the air that we breathe. Every system in our world today depends on computers which depend on the software products. So the quality of our life related to the quality of software products. From that, delivering high quality software products is a necessary need but what the software quality is and how it can be measured and improved.

Software quality can be defined and classified into two levels. The first level is the intrinsic product quality or the conformance to the costumer’s requirements and that can be expressed in two ways, the defect rate and the reliability. The second level, which is the broader, consists of the product quality, the process quality, and the customer satisfaction.

Improving the software quality during the development depends on many factors. For instance, the models of the development process, the methods and the approaches in the process, the tools and the technologies. In addition, we need measures of the characteristics and quality parameters of the development process as well as metrics and models to ensure that the process will achieve the product’s quality objectives.

There are three categories of software metrics: product metrics, process metrics, and project metrics. The software quality metrics are software metrics that related to the quality of the product, process, and project. In fact, the software quality metrics are categorized into end-product quality metrics, in-process quality metrics, and the maintenance process quality metrics.(Kan, 2003)

Introduction
Software products are related to every aspect of our modern life just like the air that we breathe. Almost every system in our world today depends on computers which depend on the software products. So the quality of our life is related to the quality of software products. From that, delivering high quality software products is a necessary need, but what the software quality is and how it can be measured and improved.

Quality in general can be viewed from two different perspectives, the popular perspective and the professional perspective. The popular perspective of quality defines it as an intangible trait. It can be felt
and judged, but cannot be measured as the saying “I know it when I see it”. This perspective implies that the people perceive and interpret quality in different ways. Another popular perspective defines it as luxury, class, and taste for example expensive and more complex products are offering a higher level of quality than their simpler equivalents.

In contrast, the professional perspective of quality believes that quality should be defined, measured, monitored, managed, and improved. From professional perspective Crosby (1979) defines quality as “conformance to requirements” that implies that requirements must be clearly identified and the final product must meet those requirements. Another definition was by Juran and Gryna (1970). They define it as the “fitness for use” and that means take customers' requirements and expectations into account. The fact that different customers may use the products in different ways implies that the products have to possess multiple elements, quality characteristics, of fitness of use. One of the quality characteristics is the quality of conformance to the intent design. Both of the definitions are essentially similar. The difference is that the fitness of use focuses more on customer’s requirements and expectations.

From a concept’s high-level definition to a product’s operational definition, many steps are involved and each of which may be vulnerable to shortcomings. An example of these steps are gathering and analyzing the requirements, writing the specifications, and developing the product. At each step an error may occur. From the customer's point of view, satisfaction after the purchase of a product is the validation of quality. From that we can say that the definition of quality has two levels of concept. The first one is the intrinsic product quality, often operationally limited to the product’s defect rate and reliability. The broader level include product quality, process quality, and customer satisfaction.

In software, the basic of product quality is commonly recognized as lack of bugs in the product and this is usually expressed in two ways, defects rate and reliability. Customer satisfaction is usually measured by percent satisfied or non-satisfied from customer satisfaction surveys. In addition, satisfaction toward specific attributes is measured as well for example IBM satisfaction attributes are capability, usability, performance, reliability, installability, maintainability, documentation, service, and overall. From that we can say that software quality is the conformance to the customer’s requirement. In addition, another aspect of software quality is the software process quality.

Improving the software quality during the development depends on many factors. For instance, the models of the development process, the methods and the approaches in the process, the tools and the technologies. In addition, we need measures of the characteristics and quality parameters of the development process as well as metrics and models to ensure that the process will achieve the product’s quality objectives.

There are three categories of software metrics: product metrics, process metrics, and project metrics. The software quality metrics are software metrics that related to the quality of the product, process, and project. In fact, the software quality metrics are categorized into end-product quality metrics, in-process quality metrics, and the maintenance process quality metrics. (Kan, 2003)
Objectives of this study

Product Quality Metrics

The product quality metrics address the two aspects of the quality definition, the intrinsic level and the customer satisfaction. The intrinsic quality is usually measured by the number of bugs (the defects rate) or by how long the software can run before crashing (mean time to failure). The MTTF is mostly used with the critical systems such as the airline traffic system while the defects density is used in many commercial software systems. These two metrics are correlated but are different because the MTTF measures the time between failures in contrast, the defect rate measures the relatively to the software size. In addition, the MTTF metric is more difficult to implement and gathering the data about time between failures is very expensive and required high level of accuracy.

The Defect Density Metric

The defect density metric is favorite method to many commercial software development organizations. That is because the defect rate of software or the expected number of defects over a certain amount of time is important for cost and resource estimates of the maintenance of the software. Simply, the defect density is the number of defects during a specific time frame over the size of the software. Because failures are defects materialized, we can use the number of the unique causes of observed failures to approximate the number of defects. The size of the software usually measured in thousand lines of code (KLOC) or in the number of functional points. About the time frame, many definitions are used ranging from one year to many years after the release of the software.

Line of Code Metric

Introduction

The oldest metric for software projects is Lines of Code (LOC). It was first introduced in 1960s and was used for economic, productivity, and quality studies (Jones, 2008). The economics of software applications were measured using “dollars per LOC.” Productivity was measured in terms of “lines of code per time unit.” Quality was measured in terms of “defects per KLOC” where “K” was the symbol for 1000 lines of code. The LOC metric was reasonably effective for all three purposes because physical line was the same as one instruction (for example in languages such as Assembly and FORTRAN).

At the early age of software industry, measuring lines of the code was an effective tool for economic, productivity and quality analysis. But after the evolution of programming languages and arrival of high level languages, LOC metric wasn’t useful as before. Differences between physical lines and instruction statements (or logical lines of code) and differences among languages caused a lot of ambiguity in counting LOCs.

In following, first we will mention briefly about different LOC counting methods. Then we will discuss about differences of these method, advantage and disadvantages.
Physical lines

This metric counts the physical lines, including the empty (or whitespace) lines, and comments. But this method has a major defect. The problem is with improper line counts, it is possible that a programmer appears really productive by hitting the Enter key, or alternatively, pretending that tighter code is written, by deleting all comments.

Physical lines of code

This type of a metric counts the lines but excludes empty lines and comments. This is sometimes referred to as the source lines of code (sLOC) metric.

Logical lines

A logical line covers one or more physical lines. Two or more physical lines can be joined as one logical line, for example in languages such as BASIC, PASCAL, and C. The logical line metric counts a joined line just once regardless of how many physical lines there are in it.

The logical lines of code metric (LLOC) has both advantages and disadvantages. It is a simple measure, easy to understand, and widely used. It can be used to measure productivity, although you need to be cautious, because programming style can have an impact on the values. You can also estimate the number of defects per 1000 LLOC.

Logical lines of code

A logical line of code is one that contains actual source code. An empty line or a comment line is not counted in LLOC.

Line of code Analysis

Each physical and logical counting has advantage and disadvantages (Jones, 2000). In general, logical statements can be considered as a more rational choice for measuring the quality of data.

In addition, in any publication of quality, the author should mention that LOC counting method is based on physical LOC or logical LOC.

In 1970, IBM encountered first known problem with LOC metrics, because many IBM publication groups exceeded their budgets for that year. The reason was that technical publication group budgets had been based on 10% of the budget for programming. Although the publication projects based on assembly language did not overrun their budgets, manuals for the projects coded in PL/S (a derivative of PL/I) had major overruns. It was discovered that the main cause of the problem was reducing only PL/S coding effort by half, with the same size of technical manuals.

Giving a formal mathematical definition to language “levels” was considered as an initial solution for this problem. The definition of “level” was the number of statements in basic assembly language needed to equal the functionality of 1 statement in a higher-level language. Thus COBOL was a “level 3” language because it took 3 basic assembly statements to equal 1 COBOL statement.

For several years before function points were invented, IBM used “equivalent assembly statements” as the basis for estimating non-code work such as user manuals. Thus instead of basing a publication budget on 10% of the effort for writing a program in PL/S, the budget would be based on 10%
of the effort if the code were basic assembly language. This method was crude but reasonably effective. However neither customers nor IBM executives were comfortable with the need to convert the sizes of modern languages into the size of an antique language. Therefore a better form of metric was felt to be necessary and resulted in to the using of function points as a metric.

Some companies use straight LOC and some other use normalized LOC (normalized to Assembler-equivalent LOC). In the latter case, the standards should include the conversion ratios from high level language to Assembler by using straight LOC count.

Comparing the size and defect rate between different languages using LOC metric, is very difficult. So it is better to compare any language against itself and study the improvement of it over the time.

In summary, LOC method has more disadvantages that advantages (Morozoff, 2010). The reasons why it might be considered as measurement tool is that LOC is a physical entity and it can be counted manually and visualized. So it can express the size of software by using some small utilities.

However there are many disadvantages for using LOC method to measure the size of the program. First of all, the utility made to count the lines of one language can’t be used for another language.

Second, there are severe problems for studying the productivity of a project using LOC. Efficient design is needed to provide more productivity for the software and this means fewer LOC. Thus LOC counting can be misleading in studying the productivity of the software. In addition, experienced and professional developers can develop software with same functionality and less code. Therefore LOC can’t be used as productivity measurement.

Third, consider two programs with same function points written in two different languages (like C and Visual Basic). Due to the differences of the programming languages, the LOC would be definitely different for two applications and can’t be used as a measurement tool.

Forth, many programming languages have provided new GUI tools that help programmers to write fewer LOC with more functionality. But the same task in one language may take more or less LOC in another language because the Code that is automatically generated by a GUI tool is not usually taken into consideration when using LOC methods of measurement.

Fifth, if we want to measure the productivity of a programmer by the lines of the code of his program, we may provide an incentive for him to write a more verbose code result in to the more complexity. This extra complexity is exactly what we don’t want because it makes it hard to manage the software and will increase the cost of maintenance and bug fixing.

Function Point Metric Overview

The function point story has begun in the mid of 1970 when IBM formed a research team to explore software measurement and metrics. After more than one year, Allan Albrecht who was an engineer in the team, and his team formulated the first version of the function point. The reasons behind the research mission were the growing importance of software in IBM and the problems with the line of code method.
IBM has used the function point method for several years. In October 1979, Albrecht introduced the function point metric in a paper called “Measuring Application Development Productivity” at a conference in Monterey, California. By 1984, the function point usage had become widespread that a non-profit association was created and called “The International Function Point Users Group”. (Garmus & Herron, 2001, p. xv)

**Identifying Function Point**

The functional point is a synthetic method that provides a methodology to measure the relative size of software project, application, and subsystems. It is based on the functionality and the value that is delivered to the end user. The functional point method can be used to measure the productivity of a team or a single developer as well as the quality of software products. Function point analysis centers on its ability to measure the size of any software deliverable in logical, user-oriented terms rather than counting lines of code. (Garmus, 2005)

The logical user data groups maintained with the application must be identified when identifying the function point. The External inputs must add, populate, revise, update, or change the data stored. In addition, the groups have to be used separately within the application in order to provide external outputs to external inquiries. There are five functional elements that are estimated based on their complexity and used to find the function point count value. The following are the five elements. (Garmus, 2005)

1- **External Inputs**
   Any data that is entering into a system for example the logical transaction inputs or system feeds. Another example, the data that is received from outside the application boundary that provides control functions or maintains of the internal logical files. Also, the data that is required unique processing requirements is external input.

2- **External Outputs**
   Any data that leaves that application such as the data that sent to another application or service, printed information, and the screen displays that provided to the user. The user-requested information that is retrievals from any combination of the internal logical files and the external interfaces is exception. These retrievals counted as external inquiries if they do not contain derived or calculated information.

3- **External Inquiries**
   The operator trigger combination that retrieves stored data without logical or mathematical calculation and without update the internal logical files are external inquiries.

4- **Internal Logical Files**
   Any data that is manufactured and stored within the system such as logical groups of user-defined data is considered as internal logical files.

5- **External Interfaces**
   Any data that is maintained within a different system but is necessary to satisfy a particular process requirement such as interfaces to other systems is considered as external interfaces.
Calculating Function Point

The process of calculating function point for software can be summarized in the following seven steps.(Garmus & Herron, 2001, p. 84)

1- Determine the type of function point count (development project count, Enhancements project count, or Application count).
2- Identify the counting scope and the application boundary.
3- Identify all data functions (internal logical files and external interface files) and their complexity.
4- Identify all transactions functions (external inputs, external outputs, and external inquiries) and their complexity.
5- Determine the unadjusted function point count.
6- Determine the value of the adjustment factor that is based on 14 general system characteristics.
7- Calculate the adjusted function point count.

The unadjusted function point is calculated by the following equation.(Kan, 2003, p. 94)

\[
\text{UFP} = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} x_{ij}
\]

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

The value of the adjustment factor (VAF) is depended on the general system characteristics (GSCs). The fourteen GSCs are rated from zero to five based on their degree of influence (DI) on the application. The following are the GSCs.(Garmus & Herron, 2001, p. 146)

1- Data communication
2- Distributed data processing
3- Performance
4- Heavily used configuration
5- Transaction rate
6- Online data entry
7- End user efficiency
8- On-line update
9- Complex processing
10- Reusability
11- Installation ease
12- Operation ease
13- Multiple sites
14- Facilitate change

Then the total degree of influence (TDI) is calculated by summing the DI of the 14 GSCs. Then the value adjustment factor (VAF) is calculated by the following equation.

\[
VAF = (TDI \times 0.01) + 0.65
\]

Then the application function point (AFP) is calculated by multiplying the unadjusted function point count by the value adjustment factor (VAF). (Garmus, 2005)

\[
AFP = UFP \times VAF
\]

**Function Point barriers**

There are two important issues that must be addressed regarding the function point metrics. The first one is the expensive cost of manually counting the function point by certified experts. The second one is the comparatively slow speed of the function point counting. Both of them are the major economic barriers of the widespread adoption of the function point metrics. These issues are applicable for different forms of function point such as IFPUG, COSMIC, Mark II, and NESMA. Manual counting implies analyzing and enumerating function points from the requirements and the specification by certified counter. Its accuracy is good but the costs are high. For example, a major ERP package such as SAP or Oracle is in the range of 275,000 function points. It would take up to six months in order to calculate manually the size of such system by team of certified counters. In addition, it would cost more than half a million dollars. (Jones, 2008, p. 78-81)

**Customer Problems Metric**

When a customer or end user uses a final software project, during deployment phase, there are inevitable problems that can come up. These problems are not only software defects. They can be usability related problems, or even user error. For instance, that happens when some procedure or task is wrongly described in software manuals or documentation. These kinds of problem are called non-defect-oriented problems. The union of the non-defect-oriented problems with the defect problems is the total plot of the software from the customers’ viewpoint. Knowing a metric in order to determine how many problems the customer will be facing is a crucial part of the software quality metrics because this can affect other metrics such as customer satisfaction metric.

Commonly, the problem metrics is stated in terms of problems per use month (PUM).

\[
PUM = \frac{TPC}{TNL}
\]

With,
TPC = Total problems that customers reported (true defects and non-defect-oriented problems) for a time period.

TNL = Total number of license-months of the software during the period.

= Number of install licenses of the software × Number of months in the calculation period.

Usually, a PUM is calculated each month after the software is released to the customer and at the end of the year an average is calculated with these values. The aim of this metric is related to usage because in the last quotient is not mentioned code line or function points. Following, there are three approaches, which lead a low PUM:

1. Enriching the development process and decrease the product defects.
2. Improving aspects of the product like usability or documentation in order to decrease the non-defect-oriented problems.
3. Looking forward to enlarge the number of installed licenses of the product or sold products.

As the defect rate metric as the customer problems metric both look to measure the qualities of a certain product, their main differences are stated in the table 4.1. Defects associate to source instructions or function points while problems concern to running of the product.

<table>
<thead>
<tr>
<th>Defect Rate</th>
<th>PUM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Numerator</strong></td>
<td>Valid and unique product defects</td>
</tr>
<tr>
<td><strong>Denominator</strong></td>
<td>Size of product: KLOC function point</td>
</tr>
<tr>
<td><strong>Measurement perspective</strong></td>
<td>Producer</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>Intrinsic product quality</td>
</tr>
</tbody>
</table>

TABLE 4.1 Defect Rate and Customer Problems Metrics
Even though each metric differed from the other, they both can be used in the same project. The customer problems metrics can be an intermediate metric between the defects measurement and customer satisfaction because this metric can improve part of usage such as documentation and non-defect problems and help to obtain a good value in the customer satisfaction metric. In Figure 4.1 the scope of these three metrics is represented by a Venn diagram. (Kan, 2003, p. 97-99)

![Venn diagram showing the scopes of three quality metrics: Customer Satisfaction, Defects, and Problems.](image)

**FIGURE 4.1 Scopes of Three Quality Metrics**

### Customer satisfaction metrics

Customer satisfaction metrics helps institutions focus on its customers and should push service owners, customer-meeting staff, policy, strategy, and research staff, as well as senior management, to find out how to improve the customer experience. However, focusing on measurement is the wrong direction to establish customer satisfaction matrices. This process is not about data collection; it is about improving customer satisfaction and making the right decisions. So, the most challenging part in this process is how to collect customers’ thoughts and get better results for it, not just collecting customer data. Institutions, which used to create customer satisfaction metrics and use it for improving quality, have a strong relationship with their customers. Also, they have their own measured levels and criteria of customer satisfaction. This allows them to measure up their customer expectations for the incoming products (Kan, 2003).

### Identifying Customer Satisfaction Process

Customer Satisfaction Process is not a set of surveys or a set of “complete the blanks” actions in response to customer satisfaction surveys. Customer Satisfaction Process is: end-to-end, planned, comprehensive, coordinated, managed and well understood set of activities and interactions designed to
achieve the highest possible of customer satisfaction. Also it is actions that initiated both ahead of “Customer Satisfaction Surveys” and as a response to “Customer Satisfaction Surveys”.

Developing a matrix is a process that must answer these questions

- Who are its customers?
- What type of survey should be administered to them?
- How will satisfaction be measured across the organization?

Understanding the Customer is the main role before and during establishing any matrix. Each matrix is designed especially for its customers and purpose with some standardization. So, the customer satisfaction must be measured for every function and every service of an institution. There are usually many connections and shared features among the services and products within an institution. For example, a bank provides a variety of different services inside it and, in addition, a number of services are done outside the bank, such as ATM machines.

In such a complex bank business, the services get classified into features and main functions, such as accounts and transactions. Thus, the first step is to identify the customers, both internal and external, for each of all functions throughout the bank business.

VOC, the voice of the customer is the important step to be determined. The VOC is essential to select the aspects of a product or service of the survey that is going to be built to measure customer satisfaction.

Customers’ feedback is very important to create and construct a customer satisfaction matrix. There are some scales rating used to get the first customers feedback. The common scale rating survey that has been used widely is measuring the customer survey data with the five-pointscale:

- Very satisfied
- Satisfied
- Neutral
- Dissatisfied
- Very dissatisfied

These points will measure different attributes for the provided services or products. For example, capability, functionality, usability, performance, reliability, installability, maintainability, documentation/information, service, and overall. After the result is collected, different kind of matrices can be created in a way to help improve a product and/ or service and make right decisions. For example, number of satisfied customer only. This may mean customers who are satisfied and completely satisfied. Usually, the second created metrics represents percentage of the collected data that belongs to the same group. These percentage metrics help the companies to form the net satisfaction index (NSI) to expand comparisons across a product. The NSI takes range from 0% to 100%. The value of the NSI represents an indicator to a product; how it was done.

Servqual

There are different methodologies to create customer satisfaction matrices. The common methodology that has been used id Servqual. Servqual or RATER is a quality framework that has been
widely used to understand customer needs and find the important factors for measuring customer satisfaction (developed in the 1980s by Valarie A. Zeithaml, A. Parasuraman and Leonard L. Berry). The framework has been known as the RATER model. It uses five dimensions as below:

- **Reliability** – A company’s ability to deliver the services and product as promised.
- **Responsiveness** – The willingness to provide prompt services and products to their clients.
- **Assurance** – The knowledge, qualification of institution staff and their ability to transfer trust and confidence.
- **Tangibles** – Physical facilities, equipment and appearances that the institution has impressed their customers.
- **Empathy** – The customer services, communication and understand customer concerns.

The Servqual dimensions for the high-level survey categories in a questionnaire work on a framework by assigning an importance weight to each element of the survey. The importance weight, combined with the customer rating, helps an institution identify areas for improvement (Rahaju&Dewi, 2011).

**The Kano Model**

The other method that is used to construct a customer satisfaction matrix is The Kano Model of Customer (Consumer) Satisfaction. This method works by classifying product attributes based on how they are understood by customers and their effect on customer satisfaction. These classifications help make decision in a way that this product or services is good or good enough for customers.

The Kano Model of Customer satisfaction divides product attributes into three main categories: threshold, performance, and excitement. A competitive product meets basic attributes, maximizes performances attributes, and includes as many “excitement” attributes as possible at a cost the market can bear.

Threshold (or basic) attributes are attributes that a product must have to meet customers’ expectations. If some of these attributes do not exist in that product or service, the product or service will be a leave the market soon due to customers” dissatisfaction.

The second class of the Kano model is Performance Attributes. The performance attributes which generally make a product or service better and will improve customer satisfaction. However, not having this attributes will reduce customers' satisfaction. The new concept of the coming product and service will be built based on the weighted performance attributes.

The last class of this model is excitement attributes. This is about the high levels of customer satisfaction. However, not having this attributes will not reduce customers’ satisfaction. Usually, excitement attributes create a good environment for competitive marketplace to match the latest services or products.

According to the Kano model, some products have other attribute that are not related to the Kano model classifications, because it has no consequence on their customers.
This model helps companies to brainstorm all features and attributes for their product. Also, it helps to classify their attributes based on their features and attributes. Building a customer satisfaction matrix using the Kano model helps to meet the product or service objectives and goals (Witell, 2011).

**Conclusion**

Quality in any product in general is target number one for any customer. It can be viewed from two different aspects: the popular aspects and the professional aspects. Engineers should focus on the two aspects to satisfy customers and users. Using product quality metrics will help improving customer and user satisfactions by addressing quality definition, the intrinsic level and the customer satisfaction and produce a high quality software product.

**References**