Indian Journal of Science and Technology

Estimation and Evaluation of Change in Software Quality at a Particular Stage of Software Development

Ekbal Rashid[1*], Srikanta Patnayak[2] and Vandana Bhattacherjee[3]

[1] Department of CS & E, CIT Tatisilwai, Ranchi,Jharkhand, India; ekbalrashid2004@yahoo.com
[2] Department of CS & E, SOA University, Bhubaneswar, Orissa, India; Patnaik_srikanta@yahoo.com
[3] Department of CS & E, B.I.T. Mesra, Ranchi, Jharkhand, India; vbhattacherjee@ieee.org

Abstract

This paper suggests a method of comparing the actual rate of software development with the projected or targeted rate. A working function may be devised from past experiences and results that would give the projected or the expected rate of software development. Then using the methods given here, the actual rate of software development can be calculated at a particular stage of work and the required comparisons can be made. On the basis of the results of comparisons made in this manner, decisions can be taken to improve the quality by increasing the quantity or quality of manpower in order to achieve the quality target within the stipulated time. In order to obtain the graphical representation of data, we have used Microsoft office 2007 graphical chart. Which facilitate easy simulation of change in software quality at a particular stage of software development.

Keywords: Project Development Time, Software Development, Accelerated and Retarded Variation.

1. Introduction

It is very common to see large projects being undertaken nowadays. The software being developed in such projects go through many phases of development and can be very complicated in terms of quality assessment. There will always be a concern for proper quality and effective cost estimation of such software. This can be rather tricky as the project being a large one may cover several unknown and unseen factors that might previously be very difficult to judge. The pertinent question here is how can we judge in case of such a large project whether the progress that is being made at a particular stage of software development is really up to the mark. How can we be assured that the rate of development of quality of the software at a particular stage is actually satisfactory enough? This analysis becomes more important because there is every possibility of losing out in the midst of development work without proper scientific planning and evaluation methods. It may so happen that the subjective understanding of the progress in development may lead to unexpected results. This will surely affect the ultimate cost and quality of the software. At the same time one can easily understand that the result of the efforts may be totally disastrous. More the scientific approach in this regard, better is the possibility of achieving expected results. It would be best if a proper mathematical model is available to assess and compare the rate of development of software at a particular stage of development. With properly defined steps and methods, it would be easy to draw correct and objective conclusions. In this method subjective analysis will be replaced by proper scientific assessment. This paper suggests proper the steps and the

*Corresponding author:
Ekbal Rashid (ekbalrashid2004@yahoo.com)

proper methods to be followed for this kind of activity. The novelty in this approach is the use of differential calculus to obtain the correct quantitative analysis. The quantitative analysis itself is a form of qualitative analysis with the help of which proper decisions may be taken to make up for any losses or lacunas in the development work. Simultaneously there may also be the possibility of over usage of resources, that is, resources that are being used, but, are not really required. Due to lack of understanding about the correct rate of development at various stages, organizations may tend to over use their resources, when then could do with lesser options. The assessment of quality at different stages with the help of the methods given in this paper and their proper records would create an important statistical knowledge base which would result in better understanding of the life cycle of any typical software type. Division of work and allocating the number of workers per module will be more easier than before.

The rest of the paper is organized as follows: section 2 gives brief overview of the various related work, section 3 describes the significance, section 4 describes the methodology. In section 5 we describe the change in quality in detail, Section 6 presents the accelerated and retarded variation as a results and conclusion has been presented in section 7.

2. Related Work and Motivation

There are already several well defined attributes for quality testing. The most prominent among them are: auditability, compatibility, completeness, consistency, correctness, feasibility, modularity, predictability, robustness, structuredness, testability, traceability, understandability, verifiability, etc. Some of the well defined metrics to be measured in different stages of software development are defect metrics and maintainability metrics. The intrinsic product quality is generally measured by identifying the number of 'bugs' in the software or by measuring how long the software can run before encountering a 'crash'. In operational definitions the two are termed as defect density rate and the mean time to failure (MTTF). The two metrics are correlated but are different enough to merit close attention. First, one measures the time between failures, the other measures the defects relative to the software size (lines of code, function points, etc.) [1]. This brings back the discussion on the differences between the terms error, fault, defect and failure. According to the IEEE/ ANSI standard:

An error is a human mistake that results in incorrect software.

The resulting fault is an accidental condition that causes a unit of the system to fail to function as required.

A defect is an anomaly in a product.

A failure occurs when a functional unit of a software-related system can no longer perform its required function or cannot perform it within specified limits [2].

However as pointed out by noted authorities on the subject, there is not much difference to be made between fault and defect and one can observe that these terms are used interchangeably in the industry. Over and above it has been felt that the terms defect or fault may be used with the end product while the term error may be used during the development process. This means that the term error refers to the mistake made by the developer and may be considered to be unintended or accidental. On the contrary, the terms defect or fault may refer to some anomaly or inadequacy in the entire software itself which points towards the fact that the designer was unable to foresee potential problems in advance.

So the term defect density deals with the end product and so does the term MTTF. However there should be a term to determine the quality at each stage of development of the software. In such cases there can be a term used often called code quality which is normally referred to as the ratio of the number of lines of code to the number of defects in the program. When there is a mention about using the number of lines of code, it becomes incumbent to mention the different viewpoints regarding the issue of counting lines of code or LOC as it is abbreviated.

The following can be taken as an authoritative definition of LOC:

"A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements" [3]. This was the definition from Conte. Now another established authority on the subject, Bohem says the LOC counting method counts lines as physical lines and includes executable lines, data definitions, and comments [4]. There is also the issue of counting the physical and the logical LOC. It is widely opined that the counting of logical LOC is a better choice for quality data. However there are several problems in this field as programming is done in different languages and so it is difficult to strike

uniformity. While assembly language may need much more lines of code as compared to high level languages, an OOL may have total different paradigms for coding as compared to structured programming language. There is the idea of converting the code into equivalent lines of assembly code. For this the LOC counted on the basis of some agreed standard is multiplied with some coefficient or some ratio to get the normalized value. For this purpose the table of conversion values published by Jones in 1986 is the most popular in industry today. In addition to this there may be other standards that can be adopted by different organizations from time to time. Whatever be the set of standards adopted, it is necessary to specify them and then do the remaining calculation. Rashid et. al emphasized on the importance of software quality estimation [5]. In [6] Rashid et. al has given Enhancing the accuracy of case-based estimation model through Early Prediction of Error Patterns.

3. Significance

The significance of this study is manifold. The following can be shortlisted as some important points:

I. The greatest problem faced by analysts while dealing with lines of code is that they are not able to compare the coding done in different programming languages. Once a common parameter or coefficient is set, and the code from any programming language can be converted into its assembly equivalent or into any other form that can make it comparable to code of other languages in any standard form, the problem faced with the issue of LOC becomes solved. This will make the work of analysis straightforward and may remove the different complicacies involved in the process.

II. The improvement of code quality with respect to the time given for development work is yet another parameter which, if assessed and measured correctly can not only decide the proper cost of the software but also ascertain the value of the developer. Besides as the development work progresses, if we are able to determine the rate of improvement of code quality, then we can also have an alternative assessment of the quality of the final product. This will be a novel idea in the realm of software engineering.

III. The introduction of a new attribute of software quality can help one understand the software better. After all what is the significance of any quality attribute?. The significance lies in the fact that more the number of attributes, the better we can understand the issue. Hence it is better to introduce more attributes so that the issue of software quality can be understood meticulously and with greater effect. Here it also needs mention that a new attribute can bring into effect understanding of newer terms which on the whole can enrich the literature of the subject.

IV. As already mentioned earlier, when we are assessing the quality of both the development phase of the software as well as the quality of the end product, we are establishing a dialectical relationship between the two. For it becomes very obvious that if the rate of improvement of the software is higher, the end product has to be of a much higher quality. Conversely, if the quality of the end product is high, it means that the rate of improvement of the quality of the software must have been quite significant. So goes the dialectical relationship.

V. So this work is significant not only because it introduces additional methods for the quality assessment of the end product, but it is also important because while trying to calculate the rate of improvement of code quality with respect to development time, we can predict the quality of the end product at any particular stage of software development. This means that we are working towards a specific goal in terms of achieving the software quality. The assessment of the end product must not vary in great detail from the prediction made in this manner. This is totally a new concept in the field of software engineering.

Looking from the other way, once we have a final product we can also guess what should be the improvement rate of the software product at any particular stage given the fact as it is also mentioned in point 4 that the two are related dialectically to each other and as mentioned in point 5, we can go with the bottom up approach, and software development like all other natural laws loves symmetry, implying that when we can go bottom up, we can also have a top to down approach with things.

4. Methodology

As software develops from an initial stage to higher stages, it undergoes several changes. A proper planning can take it to improved levels and higher quality. However this

continuous improvement of quality and standard cannot be achieved without a conscious and diligent effort. Moreover hard work and pious wishes cannot bring about the improvement of quality. This has brought about the theories of software engineering and has largely transformed a practice that was much more like an art to an activity that has closer relationship towards scientific methodology. This has brought about the recepies needed to bring out the product in right time and with the right quality. Although many methods and means already exist and continuous research is available in this field, things are not such that we have had enough and that in spite of all the knowledge in this domain things are not in wanting.

Rather more the amount of research being performed in this area, more are things getting better, clearer and precise. This rightly reflects the philosophical fact that truth is indeed concrete, particular and precise. Therefore this paper has also worked upon the different phases of software development in order to ascertain the improvement of quality at each level of development.

This means that during the development of the software, we need to be sure that the software being developed is actually going in the right direction. And this determination must not be based on just subjective formulations but on objective figures and facts. It has to be established on specific models and metrics. There should be measured parameters that can serve as indicators of growth or decay. These are the factors that need to be defined, understood, and established with a view to achieving the required goal.

With this objective of studying and estimating the improvement of quality of software during the development phase the following are some of the different terms and definitions related to the same:

LOC: The lines of code continue to be the determining metric in most of the quality and cost measurement. Now the method in which the LOC count has to be taken is also not yet standardized. However if we assume, for the sake of simplicity, that a standardization has been achieved, then on the basis of those set standards, after taking the LOC count, we can normalize the same as below:

Let the number of coded lines in assembly language be n.

Let the number of coded lines in any particular language be N. Now it is obvious that more the number of coded lines in any particular high level language, more will be the number of coded lines for the corresponding program in assembly. Hence we can say that,

$$n \propto N$$

Meaning that as the number of lines in the assembly level language is directly proportional to the number of lines in the concerned high level language.

Now let the standard coefficient for the particular high level language be k.

This coefficient will be the one decided according to the IEEE/ANSI standard. Then the final normalized equivalent lines of code in the assembly will be given by:

$$n = kN$$

where n now denotes the normalized value of the equivalent lines of code in the assembly language. Now this normalized value for the number of lines of code can be used to calculate the cost of the software or the quality of the software as required.

This standard LOC count after being normalized serves as the primary metric.

Development time: The development time can also be a confusing factor. With experienced personnel, the development time is sure to be on the lower side. However if the staff is not equipped enough, we are going to have a prolonged development time with respect to any particular work. The development time can be calculated on the basis of man hours, or working days or months. This largely depends upon the nature of the organization and the type of software being developed. At this juncture we can only say that lesser the unit of development time, greater is the accuracy of estimation of the quality of product.

Errors: The number of errors is also dependent on the person or the team of persons developing the software. If the developers are experienced there are lesser chances of errors. There may be exceptions to this. However this is the normal trend. Similarly the errors may increase if the staff is relatively inexperienced. Moreover there is also the issue of debugging. It becomes very difficult to debug the software if the staff is not experienced. On the other hand experienced and skilled developers can debug programs more efficiently.

This takes us towards the calculation of the rate of change of software quality. The rate of change of software quality can be calculated in two ways: With respect to time and With respect to LOC.

The rate of change of software quality with respect to time can be defined as the following:

The average change rate in software quality with respect to time is the change in the software quality per unit development time.

Let the initial software quality be q_i and the final software quality be q_f. Then the change in software quality becomes $q_f - q_i$.

The rate of change of software quality with respect to LOC can be defined as the following:

The average change rate in software quality with respect to LOC is the change in the software quality per line of code where the total lines of code have been taken in the normalized form.

Both these concepts are further mathematically formulated in the next section.

5. The Variation

In the previous section the concept of average rate of change of quality with respect to time and LOC has been explained. Since this parameter is expected to play a pivotal role in the estimation of software quality, it becomes incumbent to associate a name to it. The name that seems to be fit for the concept is *variation.*

The average rate of change of software quality can be termed as *average variation of software.* From the previous chapter itself we can reformulate as under:

1. With respect to development time:

$$\textit{Average Variation} = \frac{q_f - q_i}{\textit{development time}}$$

2. With respect to LOC:

$$\textit{Average Variation} = \frac{q_f - q_i}{L_f - L_i}$$

Where the symbols have the following meanings:

q_f = final quality
q_i = initial quality
L_f = final normalized LOC
L_i = initial normalized LOC

When we think in terms of the software quality, the average variation geometrically represents the slope of the secant joining two points in the graph of quality versus time. The graph in Figure 1 is the case of particular software with some imaginary data showing how an increase in quality of software looks geometrically when plotted against time.

Now when we calculate the average variation between two stages of development of the software, namely P and Q, what we get is the slope of the line PQ. The line PQ is shown in Figure 2. It joins the points P and Q of the graph. This forms the basis on which the concept of instantaneous variation is to be developed. Here it needs to be noted that this average variation is between time 3 unit to 7 unit. Had the development been as calculated in the average variation, it would have reached the same level in the particular stage Q.

However it becomes more useful if an instantaneous rate is calculated. That actually enables the developer or anyone concerned to understand whether the software is being developed in the right direction and in proper pace at any particular stage, both from the viewpoint of time and also from the viewpoint of LOC. We have seen earlier that the average rate of change of software quality can be termed as *average variation of software.* On the either hand, the instantaneous rate of change of software quality can be termed as *instantaneous variation* or simply *variation***.**

Now let us calculate the instantaneous variation at a particular stage, say time = 5 units. This can be done by calculating the average variation over different time

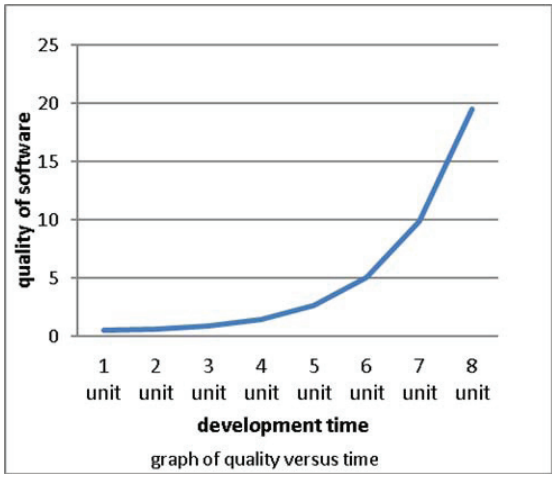

Figure 1. Graph of quality versus time.

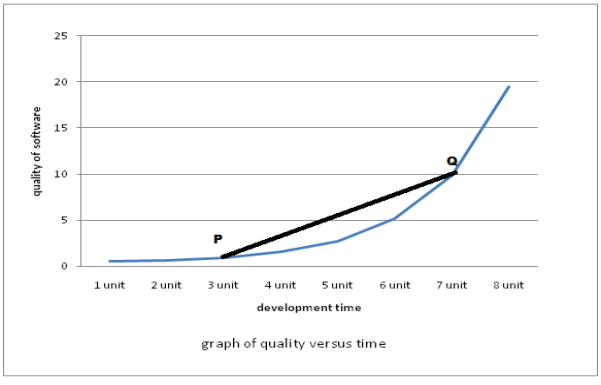

Figure 2. Graph of quality versus time.

periods and gradually reducing the time intervals in such a way that it nearly becomes zero at the stage of time = 5 units. This is shown in Figure 3.

It can be easily seen in the graph that the time interval is gradually being reduced towards the point at which we are going to calculate the instantaneous variation. At each interval the slope of the line joining the two points on the curve gives the average variation in that interval. In the above figure, the slopes of corresponding lines give the average variation between P_1Q_1 and P_2Q_2 respectively. Now as the time interval becomes vanishingly small at the stage where time = 5 units, the secant becomes a tangent and then the average variation at that point becomes the slope of the tangent at that point of the curve. This average variation given by the slope of the tangent at that point is the instantaneous variation at that particular stage of software development. This is shown in Figure 4.

This means the calculation of instantaneous variation at any particular stage of software development is simply the calculation of the slope of the tangent to the quality versus development time curve at that particular stage. From differential calculus we know that the slope of any tangent to a curve at a particular point is given by the first derivative of the function representing the curve at that particular point. Thus we can state the following expression for instantaneous variation or simple *variation:*

$$[variation]_{x=}\left[\frac{dq}{dt}\right]t = x$$

where, $[variation]_x$ stands for variation at x.

As mentioned earlier, the quality can always be expressed as a function of development time. The function may vary from software to software and even from one organization to another. Each organization will be having its own set of attributes and methods to ascertain the relationship between the quality and the development time of the software. However once this relation between the development time and quality is established, we can go forward to simply finding the derivative at a particular point in time to get the expected instantaneous variation at that particular stage.

Likewise we can also calculate the instantaneous variation with respect to the LOC. Only in this case we have to express quality as a function of the LOC. The graph of quality versus LOC has LOC count on the horizontal axis. The tangent to the curve at a particular value of LOC count gives the corresponding variation. The related graph will look much similar to the earlier graph. Figure 5 illustrates the point. This is the graph of quality versus LOC. The curvature of the graph is different and it will be explained in the next chapter.

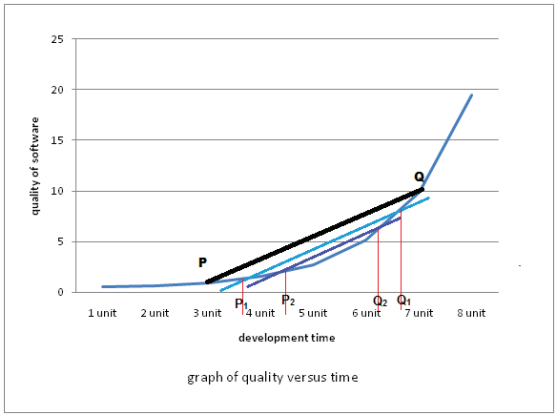

Figure 3. Graph of quality versus time.

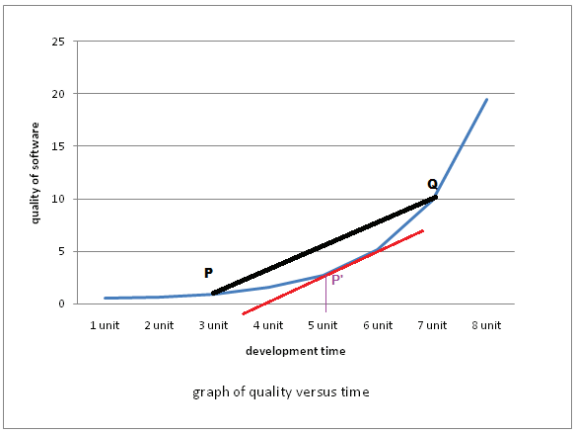

Figure 4. Graph of quality versus time.

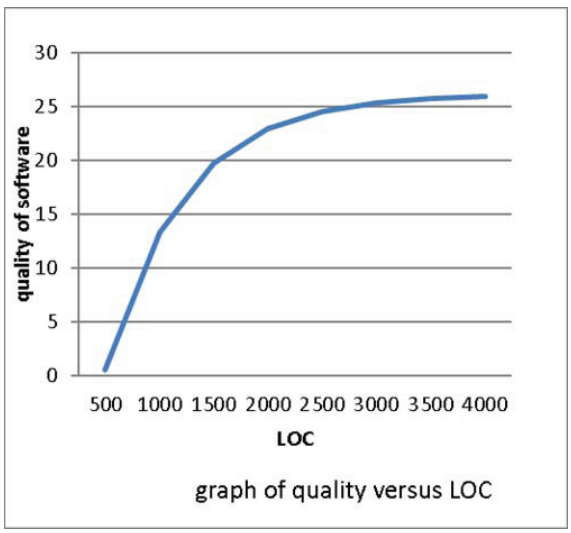

Figure 5. Graph of quality versus LOC.

We can see in Figure 6 that PQ will give us the average variation between the intervals of LOC = 500 to LOC = 3500. The slope of the tangent at P' will give the instantaneous variation when the LOC is exactly 2000.

We can calculate the variation with respect to the LOC in the same manner using differential calculus as we had done in the case of calculating variation with respect to development time. We get the expression as follows:

$$[variation]_{x=}\left[\frac{dq}{dL}\right]L = x$$

where,

q = quality (to be expressed as a function of LOC)

L = normalized LOC

$[variation]_x$ *= instantaneous variation when LOC = x*

$\left[\frac{dq}{dL}\right]L = x$ *= first derivative of the function quality w.r.t.*

LOC at the point where LOC = x

The calculation of variation has deep significance. It needs to be stressed at this juncture that the term variation used here is actually the variation of quality. Using the method shown here, we can calculate this variation in both space and time dimensions. When we are dealing with LOC, we are actually confronted with the size of the software. Then the calculation of variation deals with space dimensions.

Similarly, when we are dealing with development time, we are concerned with understanding the variation of quality in time dimension. The concept of variation of quality in both space and time for the software gives a proper scientific metric that can be understood concretely and compared to accepted standards. Figure 7 showss the model for calculation of variation.

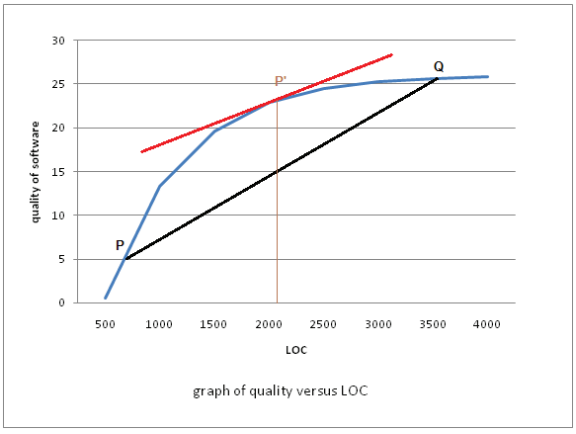

Figure 6. Graph of quality versus LOC.

6. Accelerated and Retarded Variations

The variation as calculated in the above manner may be uniform, accelerated or retarded. First an overview of these terms is given as under:

1. Uniform variation: When the change of quality of software remains the same over equal intervals of development time or over equal increase in LOC, the variation can be said to be uniform. If the software quality variation is uniform, there exists a linear relationship between the quality and development time or between the quality and the LOC. The graph drawn will invariably be a straight line as shown in Figure 8.

The same kind of graph can be seen if the horizontal component is development time. One can easily see that the ration between the quality and the LOC or the development time, as the case is always remains a constant.

2. Accelerated variation: When the increase in quality increases over equal intervals of time, we may call the situation a case of accelerated variation. The graph of an accelerated variation will be an overall concave curve with the slope of the curve increasing over increase in

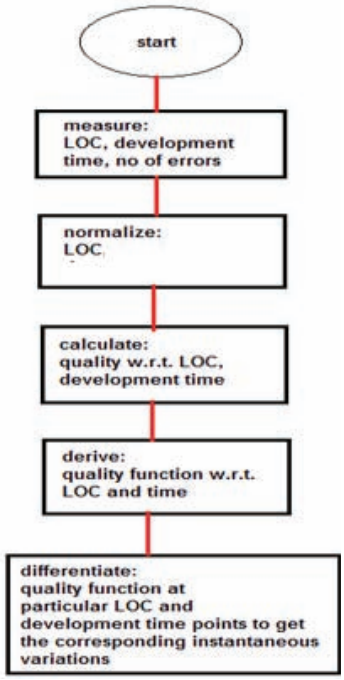

Figure 7. Calculation of variation.

LOC or development time. The graph for an accelerated variation would look something like in Figure 9.

It can be easily seen that the variation between the two stages P and Q and the variation between the two stages Q and R is not the same. The quality changes at different rates between P and Q and between different rates between Q and R. And it can also be easily seen that the variation is greater between Q and R than between P and Q. Thus such a condition when the variation increases with successive stages of LOC growth or equal intervals of development time are called accelerated variation. The curve for an accelerated variation is a concave curve as shown in the figure. Thus if the graph of quality versus LOC or quality versus development time is a concave curve, we may conclude that on the whole it is an accelerated variation.

3. Retarded variation: When the variation over successive equal intervals of development time or over successive stages of LOC growth goes on diminishing we get the case of a retarded variation. The graph showing a retarded variation will be a convex graph with a mountain in it as shown in the Figure 10.

Again it is obvious that the change of quality between P and Q is not the same as the change of quality between Q and R. From the graph we can deduce that the variation is greater from P to Q that the variation between Q and R. If we compare this case with the earlier case, the difference is quite clear. In the earlier graph, the slope between P and Q is less that the slope between Q and R. While in this graph, the slope between P and Q is greater that the slope between Q and R. The rate of change of quality becomes slower in this case. Thus we say that the average variation on the whole is gradually decreasing as through the life cycle of the software. This is a case of retarded variation. It is obvious from the above figure that on an average the graph of a retarded variation shows a convex curve. Thus if the graph of quality versus LOC or the graph of quality versus development time is a convex curve we can easily conclude that the variation is a retarded variation.

However the average nature of variation is not as significant as the instantaneous nature of variations. This is because at a particular stage of software development, the developer may be interested in finding out whether the change in software quality at that particular stage is going according to the set plan or not. For this it is necessary to

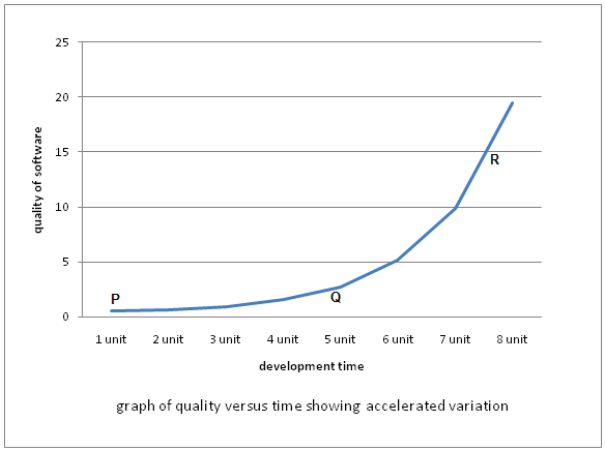

Figure 9. Graph of quality versus time showing accelerated variation.

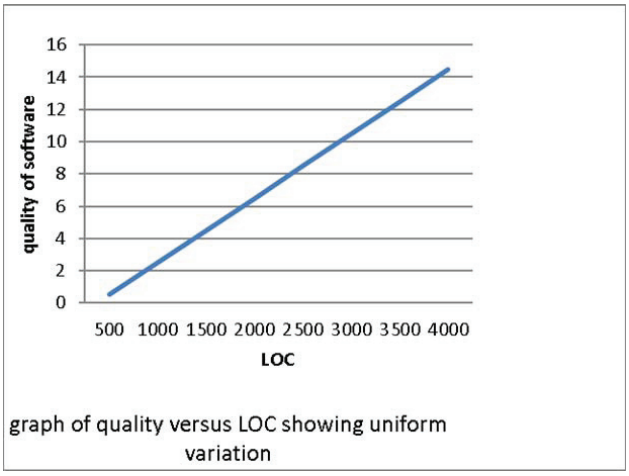

Figure 8. Uniform variation.

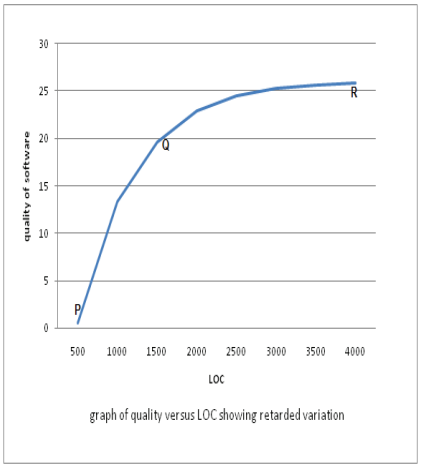

Figure 10. Graph of quality versus LOC showing retarted variation.

determine whether the variation at any particular instance of development time or at any particular instance of code development is accelerated or retarded or uniform. As we have seen earlier that the curve for an accelerated variation is a concave curve and the curve for a retarded variation is a convex curve. So if we just test the curvature of the graph at any particular instance of time, we shall be able to determine what the nature of variation is at that particular instance. Testing the curvature can be done according to the principles of differential calculus where we find the second derivative of the quality function with respect to the LOC or the development time as the case is. Now from differential calculus, if the second derivative at a particular point is positive, we have a concave curve at that point. Simultaneously, if the second derivative at a point is negative, we have a convex curve at that point. Mathematically stating we have the following:

If $\frac{d^2q}{dt^2} < 0$ or if $\frac{d^2q}{dL^2} < 0$ then at t = c or L = c respectively, then at the particular stage c of software development, the variation curve is convex, showing a retardation in the variation. In the same way, if $\frac{d^2q}{dt^2} > 0$ or if $\frac{d^2q}{dL^2} > 0$ then at t = c or L = c respectively, then at the particular stage c of software development, the variation curve is concave, showing an acceleration in the variation. There is another method of determining whether the variation at any particular stage of software development is accelerated or retarded. That is by using the method of increasing and decreasing functions. If at any particular point the graph of quality versus LOC or the graph of quality versus development time gives the indication of an increasing function, we can say that at that particular stage, there is accelerated variation. At the same time if at any particular stage of software development, the graph gives the indication of a decreasing function, we can conclude that at that particular stage of software development, the variation is retarded. We can decide whether the function is increasing or decreasing at any particular point by using the first derivative test. For this at a particular point first we find the instantaneous variation. Then we determine the variation for a point just below and just above the point in consideration. If the value of the first derivative just below the consideration point is less that the first derivative at the consideration point and the first derivative at point above the consideration point is just greater than the value of the first derivative at the considered point then the conclusion will be that the function is increasing as it is passing through the considered point. On the contrary if the first derivative at a point below is greater than the first derivative at the considered point and the first derivative at a point above the is just greater than the first derivative at the point of consideration, we may conclude that the function is a decreasing function and thus it will follow that at that particular stage of software development, the variation is retarded.

A better understanding of the entire concept can be achieved from Figures 11 and 12.

As can be seen from the above graph that the slope of the tangent at Q (the stage where we wish to determine the nature of variation) has a variation greater than that at A but less than that at B. Hence this is a case of accelerated variation at Q.

Similarly in the Figure 12, it can be seen that the first derivative at Q is less than the first derivative at A but is greater than the first derivative at B. This is

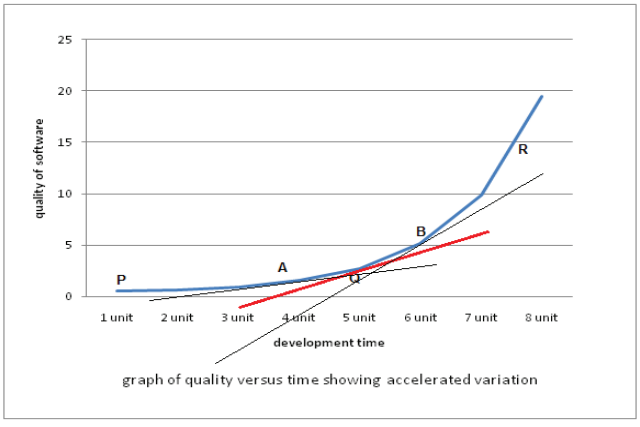

Figure 11. Graph of quality versus time showing accelerated variation.

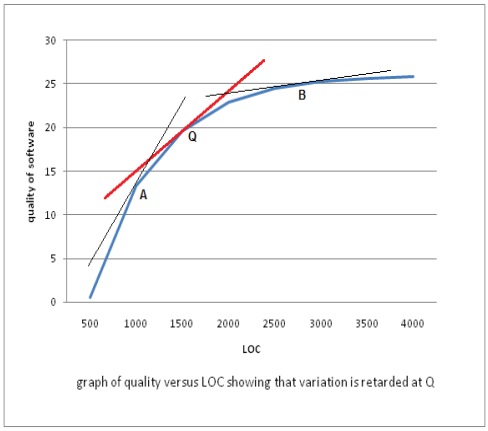

Figure 12. Graph of quality versus LOC showing that variation is retarded at Q.

very obvious from the slopes of the tangents drawn at those particular points. This points towards the fact that the function of quality with respect to the LOC is decreasing at the particular point Q. Thus we can conclude from this case that the software is undergoing a retarded variation at Q.

7. Conclusion

The paper has presented a new method of estimating and evaluating the changes in quality through particular stages of software development. By measuring the changes in software development one can understand if the software development process is going through the expected phases. At the same time we can estimate the probable quality of the final software. The method of predicting the quality of the final software is something that has not been dealt in the present paper; nevertheless, it is a part of the future work. The estimation of changes in quality helps us to properly measure, compare and take concrete decisions regarding quality during the course of software development. The evaluation of the changes in quality gives the developer the ability to forecast the next stage or make adequate plans for the next phase of development. The method is mathematically motivated and its standards can be set up either by an international body or by any organization as per the requirement. This is not only a development in the realm of software development, but also is an added treasure in the realm of applied mathematics. The use of graphical methods and other principles of differential calculus are clear pointers in that direction. There may be dispute about the method in which the various parameters are being calculated. Through repeated discussions and exchanges among experts, better methods of measurement of different parameters may evolve. Still, the method to decide the variation and the nature of variation will remain as enunciated in the paper.

8. References

1. Kan S H (2002). Metrics and Models in Software Quality Engineering, 2nd Edn., Addison Wesley.
2. IEEE/American National Standards Institute (ANSI) standard (982.2).
3. Conte S D et al. (1986). Software Engineering Metrics and Models, Benjamin-Cummings Publishing Co., Inc, USA.
4. Bohem (1981). Software Engineering Economics, 1st Edn., Prentice Hall.
5. Rashid E, Patnaik S et al. (2012). A survey in the area of machine learning and its application for software quality estimation, CM SIGSOFT Software Engineering Notes, vol 37, No. 5, 1–7.
6. Rashid E E, Patnaik S et al. (2013). Enhancing the accuracy of case-based estimation model through early prediction of error patterns, International Symposium on Computational and Business Intelligence (ISCBI 2013), DOI 10.1109/ISCBI.2013.