International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Design Quality Metrics on the Package Maintainability and Reliability of Open Source Software

Madhwaraj Kango Gopal[1]*

Vignan's Foundation for Science, Technology & Research University, Vadlamudi, Guntur District, India
* Corresponding author's Email: madhwarajkg@gmail.com

Abstract: Software metrics play an important role in the planning and control of software development. The quality of the software is the degree to which a finished software product proves to be efficient to its specification. Recently, Object-Oriented (OO) methodology has emerged as a major approach for the software development in both scientific and business applications. As the necessity for productive software is increasing, the OO design technique for constructing software is proving to be a powerful method for developing efficient software systems. Generally, the software used by the organizations and the individuals is the one that is owned by the organization which develops it. But open source software is the one which is available free for the user and can be altered based on their needs. In such kind of the software, the maintainability and reliability play a major role regarding the quality of the software developed. In this paper, we have empirically analyzed the various design metrics of different versions of software using JDepend tool and their effect on maintainability is tested. Further, the reliability of the software is measured using Rayleigh's model. The relationships between maintainability and reliability is found out by checking how these metrics influence the quality of software.

Keywords: Quality metrics, Package maintenance, Open source software, Object-oriented methodology.

1. Introduction

The objective of software engineering is to improve the methods and devices required to develop high-quality applications to make them steady and maintainable. Developers and managers employ numerous metrics in order to evaluate and develop the quality of an application during the growth process [1]. Different business and technical intentions such as shorter development cycles, lower development costs, improved product quality, and access to source code, more and more software developers and companies are basing their software products on open source elements [2]. During software advancement, Structural organization of software has the main authority on the locality of changes. One of the significant kinds of such changes is those distressed with extending and altering the executed functionality [3]. Object Oriented Design metrics are supportive in recognizing defective design at an early stage of software improvement. The steadiness of the software device is a necessary characteristic that includes much of the competence and presentation of it [4]. Object-oriented technology is built upon a sound engineering foundation, whose components are jointly called the object model. As a result, the object model is constructive for understanding problems, communicating with application experts and modeling complex enterprises into a software plan. This technology moreover assists to sponsor software reusability, maintainability, dependability and performance [5].

Open Source Software has been described as the software whose license shall not limit any party from selling or giving away the software as an element of a collective software distribution having programs from numerous dissimilar sources [6]. The word "open source" often refers to software development practice that relies on the involvements of geographically separated developers via the Internet [7]. Maintainability refers to the level of attempt necessary to develop a software unit over time so as to accurate imperfections and to manage with novel necessities or alters in its environment [8, 9]. With the contribution of several individuals and organizations, Open source software is improved. In order to attain high-quality software and evade faults, an organization that employs and customizes Open source software must cautiously administer the development process [10].

In addition, maintaining software regularly means evolving software, and adapting existing code is a huge part of new software engineering [11]. The mechanism to promote the software is called open source which is related to the complete access of the software. For making high-quality applications, well-known objectives of software engineering are to improve and employ methods and devices. The requests that have high quality and modularity are more constant and maintainable [12, 13].The maintainability of a software system can considerably impact software costs. This implies that it is significant to be able to predict a software system's maintainability so to successfully cope costs [14]. As open source software has significant economic impact and is ever more applied in mission-critical actual world applications, several organizations would like to contain at hand object measures considering the quality of the development process and the consequent product [15].

For the maintainability of the software, different techniques have been applied. A few of the techniques are Standardized Code Quality Benchmarking for Improving Software where maintainability employs a standardized measurement procedure based on the ISO/IEC 9126 definition of maintainability and source code metrics [16], Prediction of Reusability of Object Oriented Software Systems by means of Clustering Approach where hybrid K-Means and Decision tree strategy is applied to forecast the reusability value of object-oriented software components based on the metric values [17], Aspect-Oriented Reengineering of an Object-oriented Library in a Short Iteration Agile Process where an appropriate strategy for introducing aspect-oriented refactoring into a re-engineering process [18], Unified Design Quality Metric Tool for Object-Oriented approach together with other principles where the idea of merging design metric devices as a package along with other plan principles like abstractness and steadiness are executed [19], Ontologies and Review on Maintainability Models for Object-Oriented Software System where the software maintainability model with object oriented system and ontologies which offer a knowledge base domain for data structure employed in object-oriented software system is carried out [20].

The main aim of the proposed methodology is to estimate the quality of the open source software. Here, we estimate maintainability and reliability using Adaptive Genetic algorithm which helps to calculate the metrics very accurately.

The remaining document is arranged as follows. Various researches performed in relation to our suggested work are presented in Section II. The design approach and the suggested technique are described in Section III and Section IV. The result and conversation of our suggested method are demonstrated in Section V and lastly, section VI closes our suggested method for software maintainability and dependability check for open source software.

2. Related Work

In the field of OO open source software maintenance, a handful of researches have been made. OO Design Quality Metrics is a general technique applied for software maintenance. A few of the modern researches are stated beneath.

A technique with data mining clustering to hold the assessment of software systems maintainability has been offered by Antonellis *et al.* [21]. As a contribution to their study, they used software measurement information removed from Java source code. Initially, they suggested a two-steps clustering process which assists the evaluation of a system's maintainability and next an in-cluster study in order to revise the growth of every cluster as the system's versions pass by. The procedure was assessed on Apache Geronimo, a J2EE 1.4 open source Application Server. The assessment engages examining numerous editions of this software system in order to evaluate its evolution and maintainability over time.

A multivariate linear model 'Maintainability Estimation Model for Object-Oriented software in Design phase has been progressed by S. W. A. Rizvi and R. A. Khan [22], which calculates approximately the maintainability of class drawings in terms of their understandability and modifiability. As in order to count class drawing's understandability and modifiability further developed two new multivariate models.

Security idea in the field of electronic messages has been suggested by M. Zurini [23]. Dissimilar kinds of spam filters are distinguished along with the attitudes of email study. Inside the open source distribution, the idea of Bayesian spam filter was discussed. The maintainability feature was underlined in terms of spam filtering. The impact of maintainability of the high quality software was the major cause for the cost variation. For creating measurement metrics and assessing the maintainability of the software product, the main tasks of Bayesian CS open source spam filter were examined. A procedural model was explained for increasing the quality of the maintainability procedure of an open source Bayesian spam filter.

A software maintainability model with object oriented system has been suggested by S. K. Dubey *et al.* [24], in which a large number of maintainability model and a dissimilar subset associated to maintainability in the object-oriented system were explained. Therefore, the technique spotlights on the dissimilar variable, methods and datasets are applied and the study employed by different techniques.

A hierarchal quality model has been suggested by R. Shatnawi and W. Li [25] where the consequence of software refactoring on software quality was learned. They offered details of their findings as heuristics that can assist software developers to make further informed conclusions about what refactorings to execute in consider to develop a particular quality issue. On two open-source systems, they authenticate the suggested heuristics in an empirical setting. They discovered that the greater part of refactoring heuristics does develop quality, yet some heuristics do not have an optimistic impact on all software quality factors. Besides, they found that the impact study of refactorings separates software measures into two classes: high and low impacted measures. These classes assist in the endeavor to know the best measures that could be applied to recognize refactoring candidates.

Shaik *et al.* [26] have suggested an Assessment of the present state of the art in Metrics and Object-Oriented Software System Quality was made. Moreover, it encloses short evocative taxonomy of the Object-Oriented Design and Metrics.

Emanuel *et al.* [27] have offered the initial quantitative software metrics to calculate modularity level of Java-based OSS Projects named Modularity Index. By examining modularity attributes such as size, complexity, cohesion, and coupling of Java-based OSS Projects, this software metrics was created. These OSS Projects are chosen as they had been downloaded more than 100K times and deemed to contain the necessary modularity attribute to be thriving. The software metrics associated to modularity in class, package and system level of these projects were removed and examined. The resemblances found are next examined to verify the class quality, package quality, and then united with system architecture measure to create the Modularity Index.

From the existing methods, stability is the major factor that determines the software maintenance. The complexity is made low by using the weighted methods per class procedure. The existing methods fail to maintain the stability. Our proposed method optimally calculates the quality of the software in terms of reliability and maintainability.

3. Design Strategy

3.1 Open Source Software

The Open Source Software is commonly a Free Software as it offers the usage necessary and moreover it differs from normal software which is generally preventive in its usage and can never be adapted more without the consent of the developer. The maintainability of the software along with dependability, constancy, complexity and reusability has been the main problem when software is improved and this plays the key role in the functioning and a long lifetime of the enhanced software [28].

3.2 Object Oriented Concept

Further, generally, the object-oriented technique is applied for improving the open source software and thus the above quality metrics of the software have to be regarded as major information in the object-oriented idea. As of the significance in thriving improvement of software, Object-oriented metrics plays a key role in software quality metrics. The maintainability and dependability are intimately associated with each other. The maintainability of the software formulates it further dependable. The dependability of the software will rise whenever the maintainability of the software is effectual or raised. So we can declare that maintainability and dependability are directly comparative to each other.

4. Proposed Method to Find the Maintainability and Reliability of the Software

4.1 Software Maintainability

Maintainability of the software is the manner by which software can be adapted and it is regarded to be the major software quality feature.

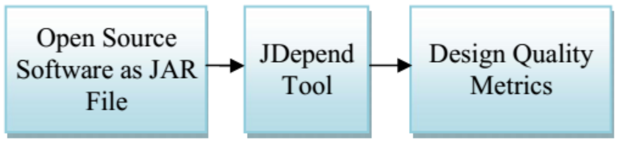

Figure.1 Software Quality Metrics Extraction Process

High steadiness and enhanced reliance are the main features that are regarded as while planning the software packages. Those software packages are regarded to be the improved maintainable software. The reliance of software can be calculated by identifying the different design quality metrics of software which more calculates approximately the maintainability of that software. The three main design quality metrics that are necessary for measuring the maintainability of the software are The Abstractness, Instability, and efferent coupling. These quality metrics from open software are to be removed to calculate the maintainability of the software. In our suggested technique, with the assist of the JDepend device, the above said design quality metrics are removed since the JDepend device permit as to automatically measure the quality of the software in terms of maintainability and dependability. The extortion process of the quality metrics is demonstrated in figure 1. As shown in the figure 1 the open source software are taken as a jar file. In our proposed method we have taken three versions of the MySQL open source software. MySQL-3.0.15, MySQL-5.0.7, and MySQL-5.1.8 are the three versions that we have chosen for our experiment. For these versions, the jar files are taken and it is applied to the JDepend tool which is used for extracting the various quality metrics of the software. The various quality metrics like abstractness, instability and efferent coupling are extracted for each version of the MySQL software and these metrics are grouped into Dependable and Non-dependable packages based on their values extracted. The packages with abstractness (A=0) and instability (I=1) is called non-dependable packages and packages with an efferent coupling (Ce=0) and instability (I=0) are grouped under the dependable packages. The dependable and the Non-Dependable packages in any software are responsible for improving or reducing the maintainability of the software. Only by measuring the count of the Dependable and the Non-Dependable packages in software, the maintainability of the software can be measured. Hence after grouping the packages into Dependable and Non-dependable packages, the count of these packages are analyzed for different versions of open source software and verified with some criteria in order to find the best maintainable software. The different criteria to be satisfied by the software are that

- For a different version of software, the non-dependable package count should be reduced.
- More dependable package must be present and it should be increasing from one version to other.
- Finally, for every version, the dependent package must be dominant over the non-dependent package.

The software that totally suits the above all criteria are regarded to be extremely maintainable. Thus for any open source software, these issues are to be the major deliberation and only that software is to be regarded to be extremely maintainable with improved quality. We further include the optimization of this dependable package in order to get better results. For optimization of packages, we have utilized a Genetic algorithm which is one of the best optimization algorithms.

4.1.1. Optimization of packages using Adaptive Genetic Algorithm

In the Genetic algorithm, initially, chromosomes are generated in which genes are the indices of the software packages. These genes are generated without any repetition within the chromosome and the values of the genes depend on the number of packages in the open source software. Then the chromosomes are subjected to the genetic operators, crossover, and mutation, and hence the new chromosomes are generated. Then the fitness is determined for the newly generated chromosomes. The various steps involved in our proposed GA is given below

Generation of Chromosomes

Initially, generate N_r a number of random chromosomes and the number of genes in each chromosome relies on the number of dependable packages that are present in the software. The generated genes are

$$G_c^{(i)} = \left\{ G_0^{(i)}, G_2^{(i)}, G_3^{(i)}, \cdots, G_{n-1}^{(i)} \right\} \quad (1)$$

$$0 \le j \le N_r - 1, 0 \le c \le n-1$$

where,

n - A number of dependable packages in the software.

$G_c^{(i)}$ - c^{th} The gene of the i^{th} chromosome.

Fitness Function

Evaluate fitness function (it will give the best input that will satisfy the best or actual solution). Higher fitness function means a better solution. Following are the purpose of the fitness function.

- The fitness function is used for Parent selection
- Fitness function measures for convergence

The fitness function is a type of objective function, which is the leading target parameter to the optimized value. The fitness function is calculated by using the following formula.

$$F_a = \sum_{a=0}^{N} ((H_a + W_a)/2)$$

$$F_b = \sum_{b=0}^{N} ((F_a + T_a)/2) \tag{2}$$

$$F = F_a + F_b$$

Here the fitness value of each chromosome is calculated based on the coverage (H) and weight (W) of the genes.

Selection of Optimal Solution

After the process is repeated I_m times, best chromosomes are selected from the obtained group of chromosomes. Here, the best chromosomes are the chromosomes which have maximum fitness. The obtained best chromosome is used to for the maintainability calculation process. For selection, various methods are utilized and in our proposed method we used Roulette-Wheel Selection.

Roulette-Wheel Selection

The *i*th string in the population is chosen with a probability proportional to f_i. The probability for selecting the *i*th string is

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \tag{3}$$

Where n is the population size?

The average fitness of the population is calculated as

$$f = \sum_{i=1}^{n} f_i \tag{4}$$

Here we will check our constraints are satisfied or not. If constraints are satisfied then we will select this output and if not then we will use genetic algorithm operator for obtaining actual output which is described below,

Crossover and Mutation

Among different types of crossovers, the two-point crossover is selected with the crossover rate of C_{rate}. In the two-point crossover, two points are selected on the parent chromosomes using the eq. (5) and (6). The genes in between the two points c_1 and c_2 are interchanged between the parent's

chromosomes and so $N_r/2$ children chromosomes are obtained. The crossover points c_1 and c_2 are determined as follows

$$c_1 = \frac{\left|G_c^{(i)}\right|}{3} \tag{5}$$

$$c_2 = c_1 + \frac{\left|G_c^{(i)}\right|}{2} \tag{6}$$

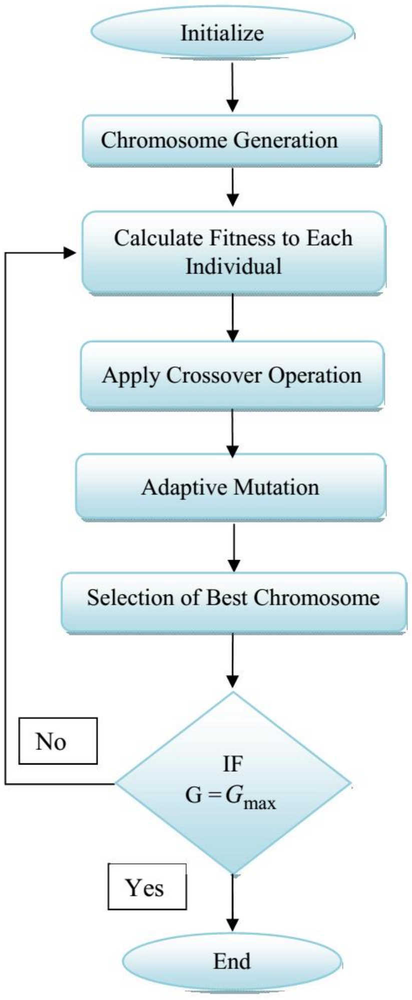
Initialize
Chromosome Generation
Calculate Fitness to Each Individual
Apply Crossover Operation
Adaptive Mutation
Selection of Best Chromosome
No
IF G = G_{max}
Yes
End

Figure.2 Proposed Adaptive Genetic Algorithm Flow Diagram

Now, we have the children chromosomes are stored separately and their corresponding indices from $G_c^{(i)}$ are stored in $G_{c1}^{(i)}$. Then, the mutation is accomplished by replacing N_m number of genes from every chromosome with new genes. The replaced genes are the randomly generated genes without any repetition within the chromosome. Then, chromosomes which are selected for crossover operation, and the chromosomes which are obtained from the mutation are combined, and so the population pool is filled up with the N_r chromosomes. Then, the process is repeated iteratively until it reaches a maximum iteration of I'_m. The final step is the convergence process where it decides when to stop. Convergence step can be defined previously at a given threshold or maximum iteration can be calculated when above steps repeat the same value.

Formerly the dependency of the software are found out, its maintainability can be calculated approximately and improved software can be structured when a software is planned in such a way that improved dependency can be happened which further results in improved maintainability. The maintainability of the software moreover manipulates its dependability to a great extent.

4.1.2 Mathematical Modeling for Making Software Maintainable

In stating the quality of the software, the maintainability of the software acts the main responsibility. As a utility of directly or indirectly calculable features, most maintainability evaluation models explain maintainability. Evaluating maintainability is, on the other hand, a disreputably hard task. Calculating approximately maintainability in a quantitative method is hard and there is no generally admitted measure for it. There are, still, a lot of issues that are agreed upon to influence maintainability. With statistical calculations, most of those features can be calculated approximately or measured. Conversely, the maintainability model we suggested now is based on the cost of the software which relates to maintainability a great extent.

Maintainability is normally said as a function of characteristics as given by,

$$M = F(a_1, a_2, a_3 \ldots\ldots a_n) \qquad (7)$$

The major characteristic that is regarded in our technique is the cost of the software and based on this our approach is planned.

The suggested model is based on two effortless statements:

- When making alterations to a software system with no clearly seeking to develop it, its maintainability will reduce, or at the very slightest it will stay unaffected.
- Executing alterations in a software system with poorer maintainability is costlier.

The cost can relate the maintainability of the software which relates to the initial statement we prepared,

$$\frac{dM(t)}{dt} = -gR(t)\phi(t) \qquad (g > 0) \quad (8)$$

In the above eq, the term $\frac{dM(t)}{dt}$ signifies the rate of change of the maintainability with respect to time and $R(t)\phi(t)$ signifies the line changed in the software and g is unvarying which signifies the maintainability reduction which is caused by changing the line in the code during maintainability.

The related equation is specified as below according to the second statement,

$$\frac{dC(t)}{dt} = z\,\frac{R(t)\phi(t)}{M(t)} \qquad (9)$$

In the above eq, z is the cutback constant and $\frac{dC(t)}{dt}$ is the changes in the cost while changing the code. From the phrase, it is obvious that maintainability and rate of change of the code are inversely related which says that as the maintainability is reduced when the cost invested for the code change raise and vice versa.

So from the eqn.9 it is obvious that for the software to be extremely maintainable the cost invested for the change of the code in the software must be in a particular amount and so the maintainability can be made simpler and effectual.

Currently, by relating the eqn.7 and eqn.8 we enter at a concluding relation which is specified below,

$$M(t_1) = M(t_0)\,\ell^{-\frac{g}{z}\left(C(t_1) - C(t_0)\right)} \qquad (10)$$

We relate the maintainability and the cost at two-time intervals $t_1$and t_0 as illustrated in the eqn.10. The maintainability of the software will be reduced exponentially with the cost of the software is made obvious from the above eq, while changing the codes. This bring to a close that for a software to be extremely maintainable, the cost must be made lesser for changing the codes and this factor offers the result that for software planned with ease of alteration or changes of the code can stay as extremely maintainable. Therefore, while planning software the cost necessary while changing the code has to be regarded as the main factor so that we can plan software which is of high maintainability.

The maintainability of the software can be amplified by knowing the cost necessary for the future. While planning novel software, the coding should be prepared with the evaluation of the cost necessary in the future for repairs of the software. By discerning this cost value one can make the software absolutely maintainable. By comparing the equations described above, a phrase for the computation of the cost of the software can be obtained by taking the combination of the means as specified below,

$$C(t) = -\frac{z}{g}\ln\left|\int_0^t (R(s)\phi(s) + \int_0^t -\frac{g}{M(s)}(R(s)\phi(s)\right| \quad (11)$$

The eqn.11 can be made simpler as specified below,

$$C(t) = -\frac{z}{g}\ln\left|1 - \frac{g}{M(s)}\int_0^t (R(s)\phi(s)\right| \quad (12)$$

The eqn.12 offers the phrase for computing the cost for the future in planning the software. As a result, by calculating the cost necessary for altering the software in the future we can plan software with high maintainability so by enhancing the quality of the software.

4.2 Software Reliability

To a great extend, the quality of the software depends on its dependability. The more dependable one software is, the higher is its quality. In a software product incessant accessibility is necessary and for this, the dependability plays the main responsibility. Normally due to the imperfections that happen during the manufacturing of software, the dependability of the software is influenced. The most important reason for the defects in software is due to the inaccurate execution. Imperfection compacts with estimating the total number of faults in the software. The faults are chiefly generated during the software planning and these faults confirm to be the major reason for determining the dependability of software. For software to be very maintainable, the amount of fault in that software must be calculated approximately properly and should be taken away. In order to estimate the imperfection rate happening in software's, different models have been devised. The Rayleigh model is more appropriate and easier model for locating the dependability of the software and it belongs to the family of Weibull allocation. It assists in recognizing the imperfection rate accessible in the software by calculating the faults present in software at certain time cases. The imperfection rate can be computed by measuring the amount of mistakes and the time intervals which more assists in locating the dependability of the software.

The Rayleigh defect forecast function is as specified below,

$$F(t) = f(K, t_m, t) \quad (13)$$

where,

K is the total defect

t_o is the maximum peak time

t is the actual time

At first, the imperfections in the software are to be distinguished in Rayleigh model. For this, the software is performed and the run-time faults in the software are found out. The time of incidence of the fault is moreover calculated. From this, the peak time and the time variation in every occurrence of a mistake are moreover noted and by means of these measures the Rayleigh function is worked out. For the dissimilar versions of the software the imperfections rates are calculated and by means of these values, the Rayleigh function is located.

Following measuring K and t_o, the Rayleigh function can be signified as specified below,

$$f(t) = K\left[(1/peak)^2 te^{-\left(1/2peak^2\right)t^2}\right] \quad (14)$$

Where,

peak - maximum time value.

The Rayleigh function can be computed by means of the eq (14) once the total number imperfections (*K*), *peak* value and actual time *t* are founded. The Rayleigh function presents the imperfection rate of the software. By discerning the imperfection rate for software its dependability can be calculated. For improved dependable software the imperfection rate must be low. Now by the similar formula, the dependability value for the three versions of the MySQL software are worked out and the version with less imperfection rate offers improved dependability value and the relation with maintainability of the software version is moreover verified.

5. Result and Discussion

As mentioned in the above section, various design quality metrics of the software were extracted using the JDepend tool and these metrics are grouped into non-dependent and dependent packages based on the values of each metrics extracted using JDepend tool. Table 1 shows the percentage of non-dependable and dependable packages for the three versions of the MySQL software.

Table 1. Percentage of Dependable and Non-Dependable Packages

Open Source Software	Non dependent package(A=0) and (I=1) (Np)	Dependent package (Ce=0) and (I=0) (Dp)	Percentage of Non-dependable package (Np/Tp)	Percentage of dependable package (Dp/Tp)	Total Package (Tp)
MySQL-3.0.15	5	12	29.41	70.59	17
MySQL-5.0.7	6	15	28.57	71.43	21
MySQL-5.1.8	6	17	26.09	73.91	23

For different versions of the software, the quality metrics are found out and the non-dependent and dependent packages count are calculated. The packages with Abstractness ‘0’ and Instability value ‘1’ are grouped under the Non-dependent package and its count is calculated and similarly the packages with Efferent coupling ‘0’ and Instability value ‘1’ are grouped under the Dependent package and its count is calculated. The total packages of software are also measured. The ratio between the count of the Nondependable package and the total package is measured which gives percentage value for Non-dependable package and ratio between the count of the dependable package and the total package is measured which gives percentage value for the Dependable package. The percentage values are shown in the below table 1.

Fig 2 shows the plot of percentage values of the three versions of the MySQL software along with a total number of the packages analyzed for measuring the maintainability value.

As shown in the fig, the dependable packages of all the three versions MySQL-3.0.15, MySQL-5.0.7 and MySQL-5.1.8 are dominant when compared to the non-dependent packages and the values are increasing along the newer versions which satisfy the criteria for the software to be highly maintainable.

The experiment shows that the reliability of the software is increased as its version is increased and so the quality of the software also tends to increase with reliability. The reliability of the software is then calculated to find its relation with the maintainability. Table 2 shows the reliability value for the three version of the MySQL software. As shown in the table, the higher version of the software has the improved reliability.

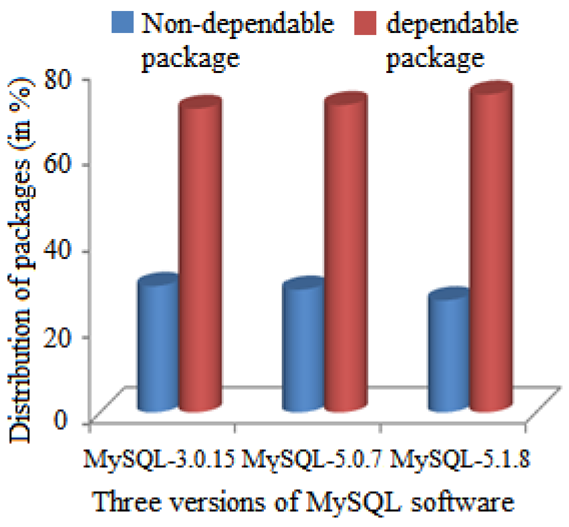

Figure.2 Graphical representation of extracted values from the table

Table 2. Reliability of the software

Open Source Software	Reliability
MySQL-3.0.15	3.853
MySQL-5.0.7	4.214
MySQL-5.1.8	4.214

The reliability value for each higher version is increasing as that of the dependable package percentage. The graph is plotted for various version of the MySQL software with its reliability values and is shown in Fig 3.,

From the Fig 2 and Fig 3, it is clear that as the dependent package value of the software increases along the higher versions of the software, the reliability of the software also improved and MySQL version 5 has the better reliability value when compared with its previous version.

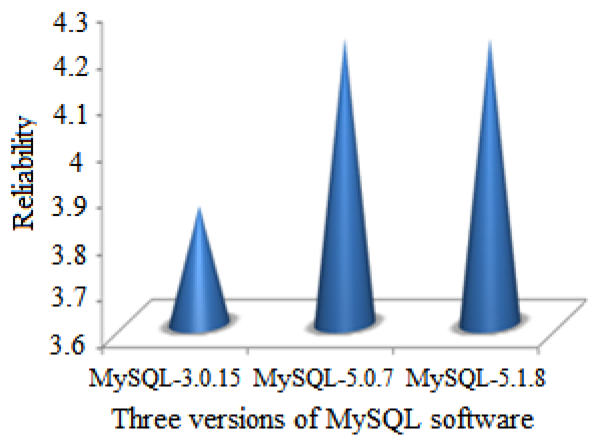

Figure.3 Reliability plot for various version of MySQL software

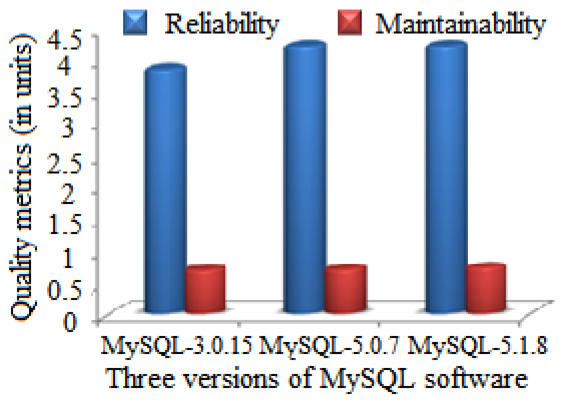

Figure.4 Relation between maintainability and reliability

As the maintainability of the software is influenced more by the values of the dependable package in software which increases through the higher version, it is concluded that the maintainability and reliability of the software are directly proportional to each other. The software with better maintainability is more reliable and provides higher quality as software. The relation between the maintainability and reliability of the software is given in Fig 4 using the table 1 and 2.

From the consequences of our suggested method, it is obvious that the software must be planned in such a manner that the count of the reliable packages of the multiple versions of the software is to be higher when compared to the Non-dependent package count so as to offer improved maintainability and dependability for the software.

Comparative Analysis

In our proposed method we are analyzing the quality of the software in terms of maintainability and reliability.

Table 2. Maintainability Comparison with existing techniques

Actual Maintainability	1
Proposed Method using AGA	0.985
Maintainability calculation using GA	0.951
Existing Method [29]	0.975

Here the maintainability can be measured using AGA which is compared with conventional GA. From the table, we observed that the maintainability can be measure accurately by using our proposed method using Adaptive Genetic Algorithm. Our performance is nearer to the actual maintainability value. So the correlation will be significant at 0.01 level.

6. Conclusion

In this paper, we have studied the relationships among different design metrics and their influence on the maintainability of a software system. Further, we also studied the relationships between the dependability and maintainability of software. Using the quality metrics, three versions of MySQL software were taken and the maintainability of that software version is identified. Later, the dependability of these software versions is computed by means of the Rayleigh's method. The maintainability of a software system would be improved if all the three criteria are contented and the imperfection rate of the software is computed to discover the dependability of the software. The correlation is verified and it is found that when there is enhanced maintainability of the software there is more dependability for that version.

In future work, we will be focusing on measuring other quality metrics such as portability, testability etc… which are proposed in the ISO 9126 standard. A quality index can be developed whereby software companies can be ranked.

Reference

[1] J. A. Dallal, "Mathematical Validation of Object-Oriented Class Cohesion Metrics", *International Journal Of Computers*, Vol. 4, No. 2, 2010.

[2] H. Orsila, J. Geldenhuys, A. Ruokonen and I. Hammouda, "Update Propagation Practices in Highly Reusable Open Source Components", *In.proc.of 20th World Computer Congress on Open Source Software, Milano, Italy*, Vol. 275, pp. 159-170, 2008.

[3] A. Olszak, E. Bouwers, B. N. Jrgensen and J. Visser, "Detection of Seed Methods for Quantification of Feature Confinement", *In proc. of the 50th International Conference on Objects, Models, Components, Patterns*, 2012.

[4] A. Handa, "Software Quality Analysis by Object Oriented Approach in Design Phase Using UML", *International Journal of Computer Technology and Electronics Engineering*, Vol. 1, No. 3, 2012.

[5] S. Babu and R. M. S. Parvathi, "Development of Dynamic Coupling Measurement of Distributed Object Oriented Software Based on Trace Events", *European Journal of Scientific Research*, Vol. 69, No. 4, pp. 527-540, 2012.

[6] R. E. A. Qutaish and M. I. Muhairat, "The Analytical Hierarchy Process as a Tool to Select Open Source Software", *In. proc. of the 8th WSEAS Int. Conference on Software Engineering, Parallel And Distributed Systems*, pp. 39-44, 2009.

[7] N. Mohamed, M. S. A. Seman and R. Hussein, "Open Source Software in Information Technology Education", *In.proc.of International Conference on Information Management and Engineering*, pp. 99-102, 2009.

[8]S. Karus and M. Dumasa, "Predicting the Maintainability of XSL Transformations", *Journal on Science of Computer Programming*, Vol. 76, No. 12, 2011.

[9] T. Goldschmidt, R. Reussner and J. Winzen, "A Case Study Evaluation of Maintainability and Performance of Persistency Techniques", *In. Proc.of the 30th international conference on Software engineering New York, USA*, 2008.

[10] Kebler, Steffen, Alpar and Paul, "Do Best Practice Frameworks Fit Open Source Software Customization”, *In.proc.of 17th European Conference on Information Systems*, pp. 2339-2350, 2009.

[11] R. P. L. Buse and W. R. Weimer, "A Metric for Software Readability", *In.Proc.of the int. symposium on Software testing and analysis, New York*, 2008.

[12] A. Haider and A. Koronios, "Issues of Open Source Software Uptake in Australian Government Agencies", *Journal in Communications of the IBIMA*, Vol. 6, No. 10, pp. 62-66, 2008.

[13] J. A. Dallal, "Software Similarity-Based Functional Cohesion Metric”, *Journal of IET Software*, Vol. 3, No. 1, pp. 46-57, 2009.

[14] M. Riaz, E. Mendes and E. Tempero, "A Systematic Review of Software Maintainability Prediction and Metrics”, *In .proc.of Third Int. Symposium on Empirical Software Engineering and Measurement*, 2009.

[15] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P. J. Adams, I. Samoladas and I. Stamelos, " Evaluating the Quality of Open Source Software", *Electronic Notes in Theoretical Computer Science*, Vol. 233, pp. 5-28, 2009.

[16] R. Baggen, K. Schill and J. Visser, "Standardized Code Quality Benchmarking for Improving Software Maintainability", *In.proc.of the 4th International Workshop on Software Quality and Maintainability , Madrid, Spain*, 2010.

[17]A Shri, P. S. Sandhu, V. Gupta and S. Anand, " Prediction of Reusability of Object Oriented Software Systems using Clustering Approach", *World Academy of Science, Engineering and Technology*, Vol. 67, pp. 853-856, 2010.

[18] A. Riordan, "Aspect-Oriented Reengineering of an Object-oriented Library in a Short Iteration Agile Process", *Informatica*, Vol. 35, pp. 499-511, 2011.

[19] U. S. Poornima, "Unified Design Quality Metric Tool for Object-Oriented Approach including other Principles", *International Journal of Computer Applications*, Vol. 26, No. 7, pp. 0975-8887, 2011.

[20] A. Sharma, S. K. Dubey and A. Rana, "Ontologies and Review on Maintainability Models for Object Oriented Software System", *VSRD International journal of Computer Science and Information Technology*, Vol. 2, No. 1, pp. 23-32, 2012.

[21] P. Antonellis, D. Antoniou, Y. Kanellopoulos, C. Makris, E. Theodoridis, C. Tjortjis and N. Tsirakis, " Clustering for Monitoring Software Systems Maintainability Evolution", *Journal of Electronic Notes in Theoretical Computer Science (ENTCS)* , Vol. 233, 2009.

[22] S. W. A. Rizvi and R. A. Khan, "Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD)", *Journal Of Computing*, Vol. 2, No. 4, pp. 26-32, 2010.

[23] M. Zurini, "Improving Email Host Security using Maintainability of Open Source Bayesian Spam Filters", *Open Source Science Journal*, Vol. 2, No. 3, 2010.

[24] S. K. Dubey, A. Sharma, and A. Rana, "Analysis of Maintainability Models for Object Oriented System", *Int. Journal on Computer Science and Engg.*, Vol. 3, No. 12, pp. 3837-3844, 2011.

[25] R. Shatnawi and W. Li, "An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model", *International Journal of Software Engineering and Its Applications*, Vol. 5, No. 4, 2011.

[26] A. Shaik, C. R. K. Reddy and A. Damodaram, "Object Oriented Software Metrics and Quality Assessment: Current State of the Art", *International Journal of Computer Applications*, Vol. 37, No. 11, pp.0975-8887, 2012.

[27] A. W. R. Emanuel, R. Wardoyo and J. E. Istiyanto, "Modularity Index Metrics for Java-Based Open Source Software Projects", *Int. Journal of Advanced Computer Science and Applications*, Vol. 2, No. 11, 2011.

[28] K.G. Madhwaraj and Chitra Babu, “An Empirical Investigation of the Influence of Object Oriented Design Quality Metrics on the Package Maintainability of Open Source Software”, *Journal of Computer Science and Engineering*, Vol. 9, No. 2, pp. 30-37, 2011

[29] S. W. A. Rizvi and R. A. Khan, “Maintainability Estimation Model for ObjectOriented Software in Design Phase (MEMOOD)”, *Journal of Computing*, Vol. 2, No. 4, 2010.