New mathematical model for software quality prediction of component-based software using shuffled frog-leaping algorithm

Deepak Panwar*

Amity School of Engineering & Technology,
Amity University Rajasthan,
Jaipur 303002, India
Email: deepakpanwar03@gmail.com
*Corresponding author

Pradeep Tomar

Department of Computer Science & Engineering,
School of Information & Communication Technology,
Gautam Buddha University,
Gautam Buddha Nagar, Greater Noida 201308, India
Email: parry.tomar@gmail.com

Abstract: Customer satisfaction and profit making are the two motives that define software quality; therefore, software industry uses new technologies like component-based software engineering, re-engineering, etc., to make their software production more profitable. The proposed mathematical model is executed under ISO/IEC 9126 quality assurance model and justifies the definition of software quality given by IEEE 1061(1998). The model calculates the degree of stakeholder satisfaction (Q) by combining the quality attributes and it is validated using Shuffled Frog-Leaping Algorithm (SFLA) which improved the result by 2.46%.

Keywords: software quality prediction; ISO/IEC 9126; shuffled frog-leaping algorithm; component-based software development.

Reference to this paper should be made as follows: Panwar, D. and Tomar, P. (2017) 'New mathematical model for software quality prediction of component-based software using shuffled frog-leaping algorithm', *Int. J. Computer Applications in Technology*, Vol. 55, No. 4, pp.266–275.

Biographical notes: Deepak Panwar is an Assistant Professor in Amity University Rajasthan, India. He obtained his Bachelor degree and Master degree in Computer Science & Engineering from Gautam Buddha University, India, in 2009 and 2011, respectively. His main research interest is in the area of software quality assurance and computational intelligence.

Pradeep Tomar is an Assistant Professor in Gautam Buddha University, India. He obtained his PhD in Computer Science & Engineering from Maharishi Dayanand University, Rohtak, Haryana, India. His main research interest is in the area of component-based software engineering and computational intelligence.

1 Introduction

Every organisation wants to run with profit only and it is a common trend to acquire the new technologies, methods, and models to enhance the quality of the software product. Software quality is directly proportional to the value of the product and the profit of the organisation. As specified by Sommerville (1982), 'software quality means the satisfaction of the stakeholders'. Producing software using Open Source Software (OSS) and Commercial off-the-shelf (COTS) components is extremely helpful to increase the quality of the software product to make it valuable for the market and to enhance its scope.

The idea of Component-Based Development (CBD) given by 'Douglas McIlroy' in a conference was about the software crisis. But after that another researcher 'Brad Cox' defined the concept of software component. Firstly, IBM used this concept in 1990s. However, these days it is very common to produce software using components, therefore CBD is very effective to increase the profit of the producer

of Component-Based Software (CBS). Component-based approach is also useful to produce complex software using COTS.

But this study suggests some hidden objectives of the CBD and these are:

1 OSS can also play an important role in CBD because developers can use an OSS as a component to develop a component-based system such as hybrid re-engineering.

2 If the developer is able to enhance the quality of a component, then it would be an easy task for the improvement in quality of software which uses that component. It means when the developer uses the COTS or OSS or both, then there should be some method of quality assurance for the respective component.

3 Testing is difficult with COTS because the code is not available to find the source of defects but this condition arrives only when the developer is working with COTS. In the case of OSS the developing team would be able to detect the source of defect or fault because the code is available after reverse engineering could be available on it.

This study also presents a novel mathematical model for software quality prediction and it is new because in past the researchers did their research about software quality prediction methods by taking one or two quality attributes without any base while the proposed work has a base model ISO/IEC 9126 with a number of quality attributes.

The remainder of the paper is organised as follows: Section 2 describes the difference between COTS and OSS but how they work together; Section 3 includes ISO/IEC 9126 software quality assurance model; Section 4 gives the knowledge about existing methods for software quality measurement; Section 5 describes the basic concept of SFLA; Section 6 presents software quality prediction problem; Section 7 is about proposed model and modelling of software quality prediction according to it; Section 8 represents experimental validation and last Section 9 provides the conclusion.

2 COTS and OSS

The study of literature shows that the COTS and OSS are the essential parts for CBD. In current scenario the software production industry wants to generate more revenue to get more profit leading to the evolution of technologies. CBD plays an important role to make quality software in less time but there are some limitations for both COTS and OSS in CBD (Sparling, 2000):

- 'A component is a language neutral, independently implemented package of software services delivered in an encapsulated and replaceable container accessed via one or more given interface'.
- 'A software component is a physical package of executable software with a well-defined and published interface'.
- 'A Software Component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition'.

All the definitions show a component like a closed source and that is the main reason regarding the inappropriate behaviour of some traditional metrics towards the CBD. For example, Halstead Software Science (HSS) equations cannot be applied to the CBD directly like equation (1), owing to the improper knowledge of actual number of operators and operands in code.

$$Fault(B) = V / S_0 \tag{1}$$

where:

V = volume in terms of operator and operands.

S_0 = discrimination constant.

It is a static equation and this is the main disadvantage for software engineers. It states that 'Fault in a program is a function of its volume'. It also shows a dependency of a number of faults detected directly to the volume of the program, therefore the old method has a huge variation in observation of the faults in software product and due to this reason it is necessary task to make it dynamic.

The developer is also unaware about the number of lines of code on which many traditional metrics work, although these definitions are applicable only for COTS but not on OSS. If a developer wants to eliminate the difficulties about the evaluation of COTS component to make a quality product, then there is a need to focus on the internal and external interfaces of the component and to reduce the testing cost developer should make it testable as pointed by Gill and Tomar (2011). Component testing is the main factor in CBSD because it supports the productivity and quality. Software testing principle said that 'Early testing should be executed' to reduce the cost and time of software development. So it is necessary to make a software component testable to achieve the quality. There is a set of program characteristics that lead to testable software, including operability, observability, controllability and understand-ability. A developer can make a component testable with a set of built-in interfaces with some features. Test interface and test architecture model are the main features which are used to interact with the test suite to choose the functional test cases. These are also helpful to interact with test driver and test report interface to provide the recorded test results. Making a component testable is not an easy task; there are lots of questions to design a testable component like:

- How could a developer be able to design the testable component architecture?
- How it would be systematic?
- How does a designer design a component which would be helpful for improving the testing ability?

As the researchers said the testable component should be able to satisfy some parameters which are important for its production.

As each component should be according to its requirements it is mandatory that it should be tested in an isolated environment. When a developer makes a test plan for the component then it would be easy to specify the test strategy.

External interface should be well defined because the component is the combination of two operational modes.

- Test mode
- Normal mode

In the first mode, the functions of the component are executed with the help of test interfaces while in another it is executed in normal mode and these modes are helpful to setting up a test case. The drivers used in these modes are functional test drivers, condition set-up drivers, case-oriented test drivers and generic test drivers. These test drivers are used to initiate the external interface. Internal interface should also be tested because test function can be set up dynamically and each test case should be executed with a test record for every test run.

Use of OSS is more profitable rather than traditional closed source software. It is more profitable for the user and developer because:

- It is based on knowledge sharing approach.
- It is more flexible and it provides the ability for the solution to adopt possible future changes in requirements and to gain high flexibility with low coupling.
- Code is available for OSS so it is ready to reuse, therefore a developer can use it to make a new product by performing white-box testing on it before executing the black-box testing approach, as is used for the COTS.
- Developer can reuse existing functionalities so it is easy to use the functionality of OSS with or without modification for new software development.
- OSS is also used to increase the productivity because increase in performance due to availability of code.

It is impossible to be ideal for anything in this world so OSS also has some limitations like quality of a product which is generated with the help of OSS is totally dependent upon code review and data testing. Sometimes this is inconvenient for the developer due to some reasons, for example:

- Review of large projects.
- Lack of tools and methods for the development and quality assurance.

3 ISO/IEC 9126 quality assurance model

International standard ISO/IEC 9126 was prepared by joint technical committee ISO/IEC JTC 1 Information Technology. It was necessary to make a framework for the evaluation of software quality. ISO/IEC 9126 is that framework which provides the facility to compile software with respect to its objectivity. ISO/IEC 9126 is the combination of six different quality attributes or characteristics with minimum overlapping and these are as follows (ISO/IEC, 2001a, 2001b):

Reliability: It is the phenomenon to perform failure free operation in specified time with specified conditions, so this characteristic concerns the maturity of software, and means the frequency of failures. It also talks about the fault tolerance and recoverability for evaluating the software systems capability to re-establish an acceptable level of performance.

Usability: This characteristic gives the knowledge about the understandability, learnability and operability of the software system and its functions. Usability also works like a property of a system.

Efficiency: It is totally dependent upon the behaviour whether it is time behaviour or resource behaviour. Time behaviour is a set of measurements which is useful for the computer response time prediction. Response time means the essential time given to a task to execute. It estimates the design, transaction path of complete modules and complete system during testing phase.

Portability: It characterises the ability of a system to change according to the modified specifications or new specifications. It also characterises the effort required to install the system. If a developer needs to change a software component then portability characterises the play and plug aspects of the software component.

Maintainability: If quality assurance team desires to analyse the cause of failure and the effort to change a system then the maintainability provides these facilities. Testing and suitability also arrive under this quality attribute.

Reusability: This newly added characteristic gives the knowledge about the interconnections between the program units and the measure of how well modules fit together. If the developer desires to check the ability of software to change according to the new system requirements then it is good to work with this characteristic. But the most important thing is that the ISO/IEC 9126 provides the metrics and methods for the measurements of quality attributes and sub-attributes.

On the other hand, ISO/IEC 9126 could not be applied to CBD in a proper manner because CBD is totally dependent upon reusability of code and there is no characteristic or attribute available in this quality assurance model for the quality assessment of a reusable component. So firstly there should be a new characteristic to solve this problem in ISO/IEC 9126 for CBD quality assessment while developer is using COTS or OSS or both (ISO/IEC, 2002). Whenever an organisation or development team of software system wishes to use this software quality framework for the development of software system using CBD techniques then the use of reusability characteristic would be compulsory for the software development. This study suggests the new characteristic reusability with its sub-attribute for the quality assessment of CBS.

4 Software quality prediction methods

Software development market has many existing models for OSS quality assurance but all of them are bound and limited to some extent, which means all the present quality assurance methods are capable to predict one, two or three quality attributes using HSS, fuzzy optimisation, neuro-fuzzy optimisation technique, etc. (Wu, 2011). Whenever a developer uses the HSS then the maximum number of faults can be detected in early phase of software development life cycle (SDLC) as pointed by Panwar and Tomar (2011) the software development team checked the impact of change in requirements, design and code. According to Table 1, the factors which have any type of effect on quality attribute, whether in a positive way or in a negative, should be calculated. The factors which have adverse effect are taken as numerator and the factors which affect positively are taken as denominator. And after that the quality assurance team calculates the variables for reusability and reliability shown in equation (2).

$$B = S_{RE} * S_{REU}\left(\frac{V}{S_0}\right) \tag{2}$$

This equation is helpful to detect the number of faults in early phases of SDLC using OSS. According to HSS the volume of OSS component can be detected and in case of any change in customer requirements it is necessary to calculate the volume and faults again.

It is a simple method which could be applicable on simple programs manually. Next method is prediction of software quality using fuzzy logic (Singh et al., 2007) pointed that the maintenance affects the cost of software; this study was based on the analysis of major factors that can affect the maintenance and the factors were documented document quality, understandability, cyclomatic complexity, etc., as shown in Figure 1. Quality assurance used a fuzzy model for the prediction of software maintenance. The given model states that the fuzzy logic is a fascinating area of research because it does a great job of trade-off between precision and significance. According to this fuzzy model developer uses the maintenance factors to predict the quality of OSS by a trained fuzzy system because fuzzy set theory has a good accuracy rate for software quality assessment (Michalmay, 2005). The related work on various aspects of software engineering can be consulted from Yi et al. (2016), Gupta et al. (2015a), Srivastava et al. (2015), and Gupta et al. (2015b).

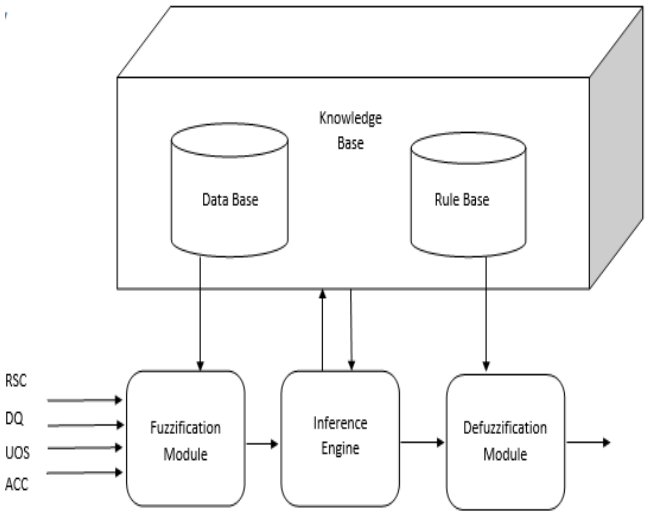
The existing model considers the inputs and outputs of maintainability as in rule base and then fuzzes the input data. Fuzzification integrates all the inputs and converts them into a single output and all these done according to the rule base (Aggarwal and Singh, 2008).

The inputs and outputs are classified using trimf membership function. This method is also applied to the other quality attributes like reusability, etc., but it works with some limitations like – it works for single quality attribute (Baisoh and Liedtke, 1997).

Table 1 Impact on software quality attributes

S. no.	*Factors for analysing quality attributes*	*Effect*	*Impacts on reusability with change in different phases*	*Impacts on reliability with change in different phases*
1.	Requirement	Change	▲	▲
		No change	▼	▼
2.	Design	Change	▲	▲
		No change	▼	▼
3.	Code	change	▲	▼
		No change	▼	Constant
4.	Component complexity	Increase	▼	▼
		Decrease	▲	▲
5.	Software complexity	Increase	▼	▼
		Decrease	▲	▲

Figure 1 Factors that can affect the maintenance



5 Shuffled frog leaping (SFL)

It is a meta-heuristic technique and it has been developed for solving combinatorial optimisation problems. It is used in proposed work to achieve higher degree of customer satisfaction (Quality Software). But before the discussion on proposed work there is a need to understand what the SFL is. As pointed out by Eusuff et al. (2006), SFL is based on the behaviour of random numbers of frogs which means on a random population of frogs. In this technique the behaviour is observed, imitated and modelled and then the population is partitioned into memeplexes (Otte et al., 2008). SFL is already a successful technique for various optimisation problems such as water resource optimisation, travelling salesman problem, etc. It is the combination of the benefits of Genetic Algorithm and Particle Swarm Optimisation (Rajpurohit et al., 2016).

SFL has various steps as shown in Figure 2. In SFL algorithm a random number of frogs or a population (P) is defined and then divided into subset called memeplexes. Each frog performs local search and can be influenced by the other frog to evolve within the memeplex and this evolution is called memetic evolution. SFL can be described as after a particular number of evolution steps all the subsets are forced to mix together and new subsets are formed through a shuffling process. To satisfy the convergence criteria the process of local search and shuffling remains in continuity (Gill and Grover, 2003).

The various steps are as follows:

1 It involves a random population of frogs (P) which is made by a group of effective frogs.

2 P is defined according to the value of their fitness and then it is divided into (m) memeplexes.

3 Frog i is expressed as $x_i = (x_{i1}, x_{i2}, \ldots, x_{ik})$, where k means the number of variables.

4 Best fitness valued frog is identified by X_b and the frog which has the worst fitness value is identified by X_w within a memeplex.

5 And the global best fitness value is defined by X_g.

6 Following equations are used to improve the worst fitness valued frog, as shown in Figure 3 (Eusuff et al., 2006).

$$D_i = (r)(X_b - X_w) \tag{3}$$

$$X_{new\,w} = X_{old\,w} + D_i \left(-D_{max} \le D_i \le D_{max}\right) \tag{4}$$

where:

D_i = Change in frog position

D_{max} = maximum change in frog position

If this method gives a better solution in comparison to worst frog then it replaces the worst frog otherwise equations (3) and (4) are repeated to global fitness value X_g and when there is no further improvement with this case then X_w will be produced randomly (Gill and Grover, 2004).

Figure 2 Shuffled frog-leaping algorithm process

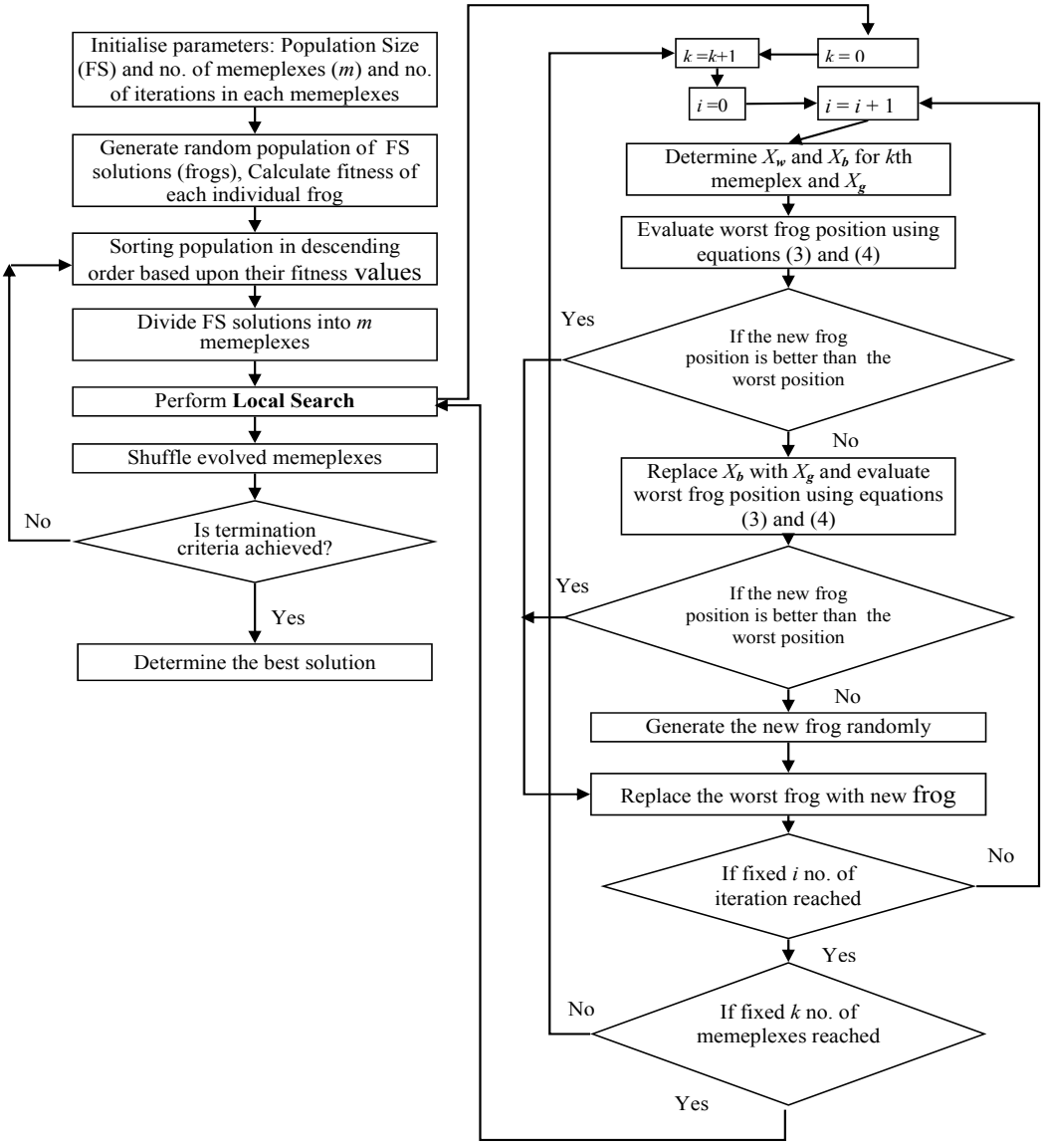
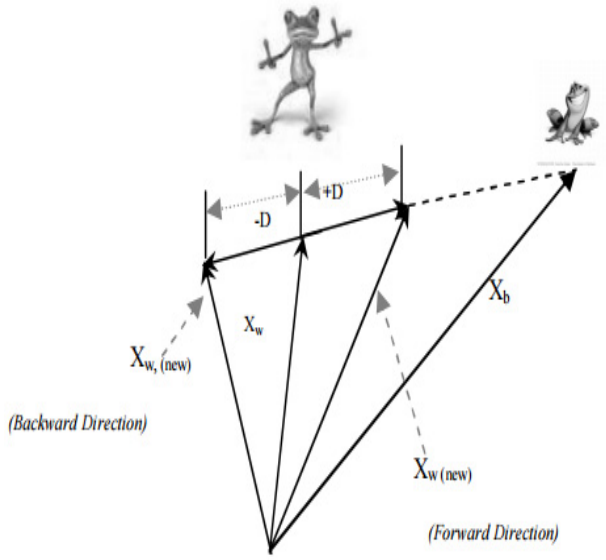


Figure 3 Shuffled frog leaping



6 Defining software quality prediction problem

Software quality prediction is a measure of the fulfilment criteria of customer requirements in the form of different quality attributes. Measurement of attributes is dependent on some basic characteristics like metric name which is useful to match the metric with its exact value, characteristics like purpose of the metric (ISO/IEC, 2001b), method of application, measurement formula and most important interpretation of the measured value of that particular matric. But the software quality means is to meet some performance goal according to the customer requirements. Researchers define the degree of customer satisfaction in the form of a variable Q (Raghunathan et al., 2005).

Degree of customer satisfaction measures how well the customer expectations are met to the actual output of a product or a service provided by the software development organisation to the customer. Fulfilment of expectations is totally dependent upon quality attributes and it signifies that the quality attributes should be optimised with respect to the customer requirement specifications, for example when the value of functional compliance is near to 1 then it is good, so if the researcher works to get quality software then this attribute should be near to 1. If the value of accuracy expectation falls then it is good. Case with knapsack problem, the priority given to the quality attribute according to the customer requirements represents its weight and a random number of items numbered from 1 to n each with a weight W_{ij} as shown in equation (6). The 0/1 knapsack problem comes under NP-complete problem and in this paper SFLA is used for solving this hard combinatorial optimisation problem of software quality prediction. When M number of quality attributes according to any software quality assurance model then researchers compare these attributes with 0/1 knapsack problem as shown in Table 2.

Table 2 Relationship between knapsack and software quality prediction problem

Item no.	*0/1 knapsack problem*		*Software quality attributes*	
	Price	*Weight*	*Priority*	*Value of attribute*
1.	5	3 kg	0.10	0.79325
2.	25	2 kg	0.17	0.9325
3.	10	7 kg	0.19	0.01234
4.	3	10 kg	0.13	0.11223

7 Proposed mathematical model

In this section of paper the proposed mathematical model is discussed for the assessment of software quality for CBS. Software quality means the degree of stakeholder's satisfaction. It consists of many quality attributes defined by researchers through different quality assurance models like as McCall quality model, ISO/IEC 9126, etc. Proposed work has its dependency on ISO/IEC 9126 quality assurance model. It consists of six main attributes as already mentioned in Section.3. But ISO/IEC 9126 does not make any transparency in case of reusability attribute and its sub-attributes. When developer wants to go for the quality prediction for CBD then it is necessary to consider the reusability factor. So the proposed mathematical model has its dependency on seven qualities for quality prediction of CBS. In proposed model researcher defined Q as a quality degree measure in customer satisfaction.

$$Q = \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} K_{ij} W_{ij} \tag{5}$$

where:

Q = quality measure degree in customer satisfaction

$K_{ij} = R_i S_j$

R_i = priority given to the main attributes

S_j = priority given to the sub-attribute

{Before applying SFL the normalisation for S_j is 0.1 and scale of S_j is between (0 – 1)}

W_{ij} = calculated value of quality attributes according to formulas defined in ISO/IEC 9126.

After taking equation number for modified ISO/IEC 9126, we get:

$$Q = \sum_{i=1}^{i=7} \sum_{j=1}^{j=5} K_{ij} W \tag{6}$$

$$\begin{aligned} Q = {} & R\left(S_1 W_{11} + S_2 W_{12} + S_3 W_{13} + S_4 W_{14} + S_5 W_5\right)] \\ & + \left[R_2\left(S_1 W_{21} + S_2 W_{22} + S_3 W_{23} + S_4 W_{24} + S_5 W_{25}\right)\right] \\ & + \cdots + \left[R_7\left(S_1 W_{71} + S_1 W_{72}\right.\right. \\ & \left.\left. + S_1 W_{73} + S_1 W_{74} + S_1 W_{75}\right)\right] \end{aligned} \tag{7}$$

Some attributes have five sub-attributes but some have less than five so in that case, value of extra sub-attribute should be equal to zero. So it does not have any effect on Q.

8 Modelling software quality prediction using mathematical model with SFLA

1 The SFLA is dependent upon the fitness value of frogs.

2 The shuffling process is carried out with the change in position of frog and it depends upon the worst valued and best valued frog within a memeplex.

3 Normalisation of quality attributes brings them on a common platform and means it is the equally distributed priority value to the quality sub-attributes.

4 But as the change in positions of frog. We got the new positions. Researchers also get the new priority value for sub-attributes which are behaving like memeplexes.

8.1 Example validation

This study used a management system with the name 'E-out-pass Management' made by the faculty of Computer Science and Engineering of Amity University Rajasthan. This system derives many of its features from the modern – email system. This given system is easy to use and makes the process of obtaining the out-pass/gate pass in a synchronised way. This project contains five modules, i.e. administration, student, mentor, warden and security. The admin has the sole right to provide rights to each genre. The project's aim is to improve the process of generating out-pass in a secure manner at school, colleges and various private institutions. The knowledge about the requirements of this project is very necessary to discuss because the prioritisation values are depending upon them.

Core system functionalities are:

- Admin should be able to register for faculty and warden.
- Student can register for himself.
- Student registers for the out-pass and request is sent to mentor.
- The mentor can accept or deny the request. If accepted, the request is forwarded to the warden. Email is sent to student and warden regarding the out-pass status.
- The warden can accept or deny the request. If accepted, the student can generate his out-pass. Email is sent to student and warden regarding the out-pass status.

Some important and fundamental requirements are as follows with respect to different modules.

Admin module

- Admin should be able to register for faculty and warden.
- Admin should be able to add hostel number, wing, room number, etc.

Student module

- Student should be able to register himself.
- Student should be able to register for out-pass and request should be sent to his mentor.
- Student should be able to generate his out-pass in pdf form or in printable form his request gets accepted.

Mentor module

- Mentor should be able to update student information.
- Mentor should be able to accept the student out-pass request.
- If mentor accepts student request, the request is forwarded to the warden.
- If mentor denies the student request, the request is deleted.
- Student out-pass status mail is sent to both warden and student after mentor accepts or denies the out-pass request.

Warden module

- Warden should be able to accept the student out-pass request forwarded by mentor.
- If warden accepts student request, the student is able to generate out-pass, but if warden denies student request, the request is deleted.
- Student out-pass status mail is sent to both mentor and student after warden accepts or denies the out-pass request.

These are the basic requirements of project for the validation of our mathematical model using SFL. It is important to describe the basic requirements because calculated values of quality attributes and sub-attributes are totally dependent upon the customer requirements.

The values of quality attributes and sub-attributes are calculated with the help of metrics defined in ISO/IEC 9126. Calculation is dependent upon particular metric scale type, size types and count types. Calculated values are shown in Table 3.

The ordering of quality attributes is totally dependent upon the customer requirements. So the priority of seven different attributes are according to the requirements of project 'E-out-pass Management' as also been discussed in previous section.

Table 3 Software quality attributes primary values, values after using mathematical model and values after applying SFL

S. no.	*Quality attributes according ISO/IEC-9126*	*Sub-attributes*	*Measured value*	*Priority value for quality attributes according to the customer*	*Normalisation coefficient before applying SFL*	*Scaled value of sub-attributes*	*Normalisation coefficient after applying SFL*	*Improved scaled value*
1.	Reliability	Maturity	0.771506	0.9	0.1	0.0694355	0.199	0.1381760
		Fault tolerance	0.948212		0.1	0.0853391	0.195	0.1664112
		Recoverability	0.123634		0.1	0.0111271	0.181	0.0201399
		Reliability compliance	0.296782		0.1	0.0267104	0.184	0.0491471
2.	Reusability	Correctness	0.518987	0.3	0.1	0.0155696	0.106	0.0165037
		Extensibility	0.277653		0.1	0.0083296	0.103	0.0085794
		Reusability compliance	0.923701		0.1	0.0277110	0.101	0.0280884
3.	Maintainability	Analysability	0.782193	0.6	0.1	0.0469316	0.159	0.0746212
		Changeability	0.895292		0.1	0.0537175	0.121	0.0649982
		Stability	0.373293		0.1	0.0223976	0.155	0.0347162
		Testability	0.221397		0.1	0.0132838	0.170	0.0225824
		Maintainability compliance	0.384490		0.1	0.0230694	0.113	0.0260684
4.	Functionality	Suitability	0.486380	0.7	0.1	0.0340466	0.124	0.0422177
		Accuracy	0.756264		0.1	0.0529385	0.151	0.0799371
		Interoperability	0.598944		0.1	0.0419261	0.179	0.0750476
		Security	0.193677		0.1	0.0135574	0.175	0.0237254
		Functionality compliance	0.075170		0.1	0.0052619	0.148	0.0077876
5.	Usability	Understandability	0.602931	0.4	0.1	0.0241172	0.166	0.0400346
		Learnability	0.545237		0.1	0.0218095	0.163	0.0355494
		Operability	0.096077		0.1	0.0038431	0.143	0.0054956
		Attractiveness	0.953120		0.1	0.0381248	0.111	0.0423185
		Usability compliance	0.948011		0.1	0.0379204	0.108	0.0409540
6.	Portability	Adaptability	0.601954	0.5	0.1	0.0300977	0.135	0.0406319
		Insatiability	0.110660		0.1	0.0055330	0.132	0.0073035
		Coexistence	0.812503		0.1	0.0406252	0.130	0.0528127
		Replaceability	0.726915		0.1	0.0363458	0.127	0.0461591
		Portability compliance	0.832286		0.1	0.0416143	0.116	0.0482725
7.	Efficiency	Time behaviour	0.149923	0.8	0.1	0.0119938	0.194	0.023268
		Resource utilisation	0.224257		0.1	0.0179406	0.190	0.0340870
		Efficiency compliance	0.726557		0.1	0.0581246	0.187	0.1086929

The quality attributes which have the highest impact upon the quality (requirements) are present with highest priority as shown in Table 3. After that when researcher used SFLA for the optimisation then it is necessary to use normalisation coefficient first to bring all the attributes at a common platform.

Software prediction problem is an NP-complete problem like 0/1 knapsack and SFL technique is an optimisation algorithm for the solution of combinatorial problem so the researcher used it to get the best solution with the help of given mathematical model. When the values of quality attributes are available then researcher applied the mathematical model as shown in equation (6) to get the value of Q called Q_{old} with a common normalised value, i.e. 0.1. It is an initial value in terms of degree of customer satisfaction. But after applying the SFL researcher got the improved value for Q means more customer satisfaction. Using equations (3) and (4) SFL has been applied to change the position of the sub-attributes (frogs) with the help of

improved normalisation coefficient (priority) because all the attributes and sub-attributes are behaving like the memeplexes and sub-memeplexes as shown in Figure 2. Applying SFL we got a new value for Q, i.e. Q_{new}, and after the comparison of Q_{old} and Q_{new} we got a better result in terms of degree of customer satisfaction which is improved by 2.46% as shown in Figures 4, 5 and 6. So it is good to use to optimise the result using computational intelligence for the proposed mathematical model of software quality prediction.

9 Discussion

This paper presents a new mathematical model for software quality prediction of CBS by introducing a new term: the degree of customer satisfaction as a unit of software quality. Software quality prediction problem is a combinatorial problem which acts like an NP-complete problem. So it becomes necessary to use an appropriate stochastic algorithm to solve this type of problem. SFL played a significant role to find an optimal solution for this type of combinatorial problem because software quality attributes and sub-attributes work like the elements of memeplexes and sub-memeplexes defined in SFLA. So the proposed mathematical model in this study acts like an initial important step to predict the software quality. Researcher taken the quality attributes with a neutral impact on each other as shown in proposed mathematical model validation results using SFLA like the iterations increase the degree of customer satisfaction achieve the convergence in a positive manner; however, it is also possible that a particular attribute with increasing value may affect the other subsequent attribute in an adverse or in a constructive manner. In future the researchers could choose other meta-heuristic approach through which the proposed model could be better for quality prediction of software.

Figure 4 Comparisons of software quality attributes after and before applying SFL

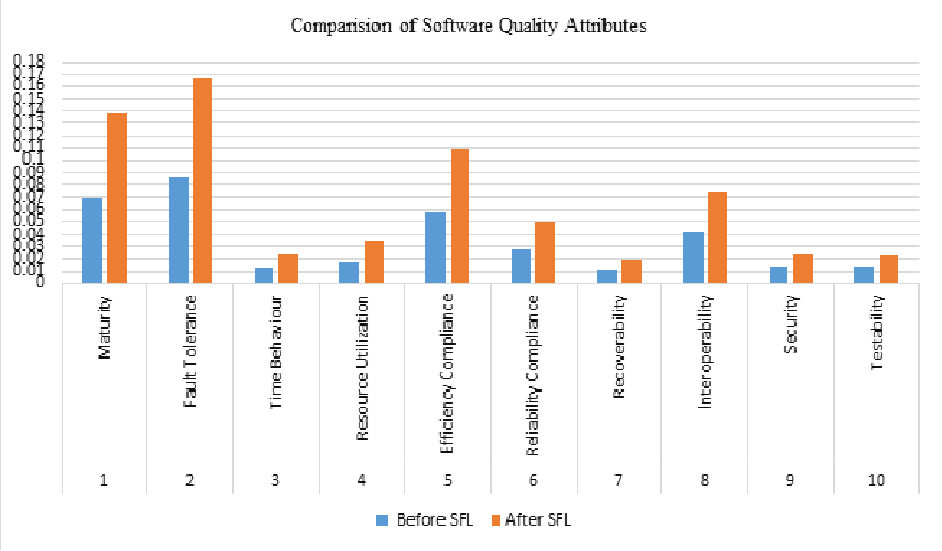


Figure 5 Comparisons of software quality attributes after and before applying SFL

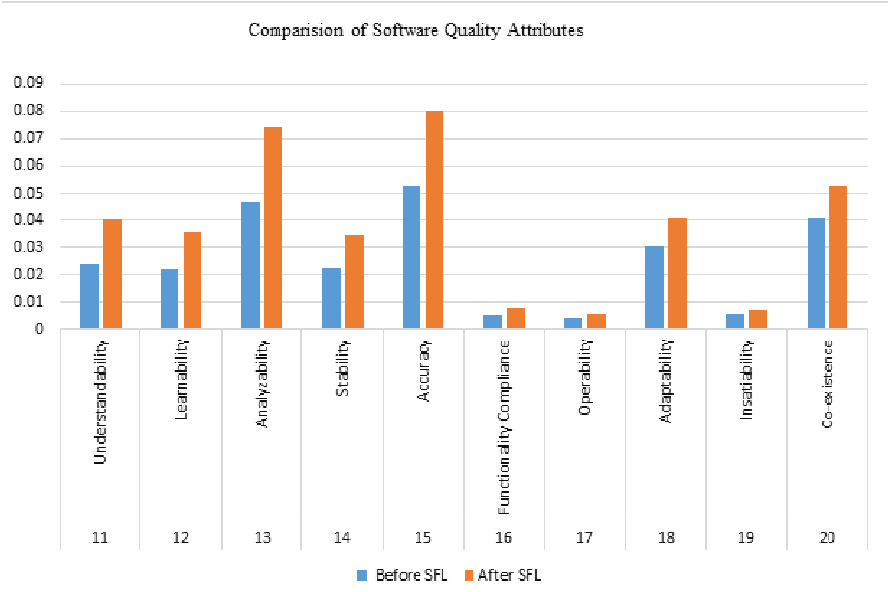
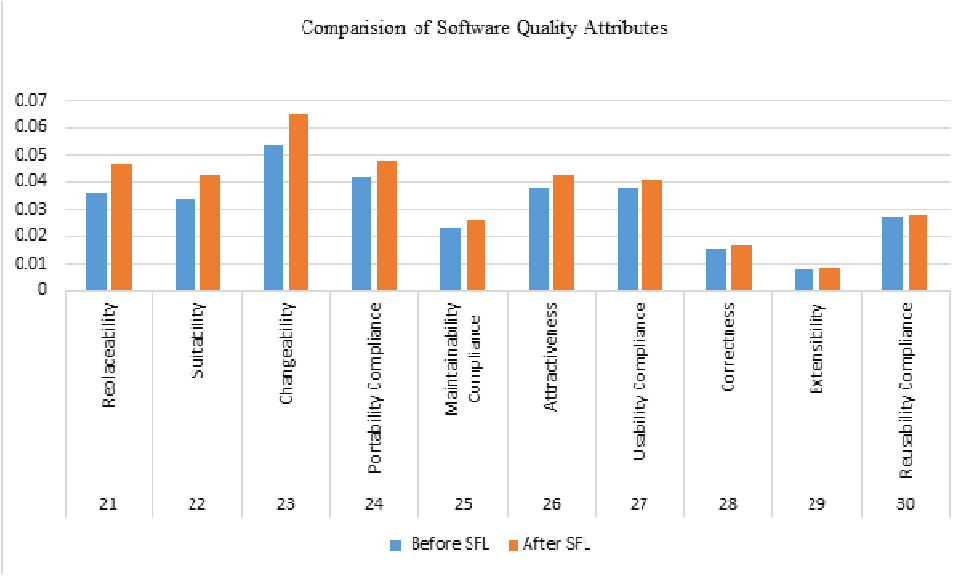


Figure 6 Comparisons of software quality attributes after and before applying SFL



References

Aggarwal, K.K. and Singh, Y. (2008) *Software Engineering: Program, Documentation and Operating Procedure*, 3rd ed., New Age International Publication, New Delhi.

Baisoh, E. and Liedtke, T. (1997) 'Comparison of conventional approaches and soft-computing approaches for software quality prediction', *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, Piscataway, NJ, pp.1045–1049.

Eusuff, M., Lansey, K. and Pasha, F. (2006) 'Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization', *Engineering Optimization*, Vol. 38, No. 2, pp.129–154.

Gill, N.S. and Grover, P.S. (2003) 'Component-based measurement: few useful guidelines', *ACM SIGSOFT Software Engineering Notes*, Vol. 28, pp.1–4.

Gill, N.S. and Grover, P. (2004) 'Few important considerations for driving interface complexity metric for component-based development', *ACM SIGSOFT Software Engineering Notes*, Vol. 29, pp.1–4.

Gill, N.S. and Tomar, P. (2011) 'New & innovative process to construct testable component with systematic approach', *ACM SIGSOFT Software Engineering Notes*, Vol. 36, pp.1–4.

Gupta, V., Chauhan, D.S. and Dutta, K. (2015a) 'Hybrid regression testing technique: based on requirement priorities, faults & modification history', *International Journal of Computer Applications in Technology*, Vol. 51, No. 4, pp.352–365.

Gupta, C., Srivastav, M. and Gupta, V. (2015b) 'Software change impact analysis: an approach to different type of change to minimize regression test selection', *International Journal of Computer Applications in Technology*, Vol. 51, No. 4, pp.366–375.

ISO/IEC (2001a) *ISO/IEC 9126-1 Software Engineering-Product Quality-Part-1: Quality Model*, International Organization for Standardization, Geneva.

ISO/IEC (2001b) *ISO/IEC 9126-2 Software Engineering-Product Quality-Part-2: External Metrics*, International Organization for Standardization, Geneva.

ISO/IEC (2002) *ISO/IEC 9126-1 Software Engineering-Product Quality-Part-3: Internal Metrics*, International Organization for Standardization, Geneva.

Michalmay, M. (2005) 'Quality practices and problems in free software projects', *Proceedings of the First International Conference on Open Source System*, Genoa, Italy, pp.24–28.

Otte, T., Moreton, R. and Knoell, H.D. (2008) 'Applied quality assurance methods under the open source development model', *32nd Annual IEEE International Computer Software and Applications, 2008. COMPSAC '08*, IEEE, Turku, Finland.

Panwar, D. and Tomar, P. (2011) 'New method to find the maximum number of faults by analysing reliability and reusability in component-based software', *2011 3rd International Conference on Trendz in Information Sciences and Computing (TISC)*, IEEE, Chennai, India, pp.164–168.

Raghunathan, S., Prasad, A., Mishra, B.K. and Chang, H. (2005) 'Open source versus closed source: software quality in monopoly and competitive markets', *IEEE Transactions on Systems, Man and Cybernetics: Part A: System and Humans*, Vol. 35, No. 6, pp.903–918.

Rajpurohit, J., Sharma, T.K. and Nagar, A.K. (2016) 'Shuffled frog leaping algorithm with adaptive exploration', Proceedings of Fifth International Conference on Soft Computing for Problem Solving *Volume 436 of the Series* Advances in Intelligent Systems and Computing, Springer, Singapore, pp.595–603.

Singh, Y., Kumar, P. and Sangwan, O.P. (2007) 'A review of studies on machine learning techniques', *International Journal of Computer Science and Security*, Vol. 1, pp.70–84.

Sommerville, I. (1982) *Software Engineering*, 6th ed., Addison-Wesly, Harlow, England.

Sparling, M. (2000) 'Lessons learned through six years of component-based development', *Communication of the ACM*, Vol. 43, pp.47–53.

Srivastava, P.R., Pradyot, K., Sharma, D. and Gouthami, K.P. (2015) 'Favourable test sequence generation in state-based testing using bat algorithm', *International Journal of Computer Applications in Technology*, Vol. 51, No. 4, pp.334–343.

Wu, B.H. (2011) 'An evolutionary approach to evaluate the quality of software system', *Fourth International Workshop on Advanced Computation Intelligence*, IEEE, Wuhan, China.

Yi, W., Li, X. and Pan, B. (2016) 'Solving flexible job scheduling using effective memetic algorithm', *International Journal of Computer Applications in Technology*, Vol. 53, No. 2, pp.157–163.