Optimal Software Release Using Time and Cost Benefits via Fuzzy Multi-Criteria and Fault Tolerance

Praveen Ranjan Srivastava*

Abstract—As we know every software development process is pretty large and consists of different modules. This raises the idea of prioritizing different software modules so that important modules can be tested by preference. In the software testing process, it is not possible to test each and every module regressively, which is due to time and cost constraints. To deal with these constraints, this paper proposes an approach that is based on the fuzzy multi-criteria approach for prioritizing several software modules and calculates optimal time and cost for software testing by using fuzzy logic and the fault tolerance approach.

Keywords—Software Testing, Fuzzy Multi-Criteria Approach, Fuzzy Logic, Fuzzy Rules Based, Confidence, Centre of Gravity, Fault Tolerance, Kilo Line of Code (KLOC), Software Development Effort (SDE), Software Test Effort (STE), Decision Makers (DM)

1. INTRODUCTION

Testing is an important and critical part of the software development process because it is a necessary process and consumes almost one third of the time and cost of a project [1]. The quality of the delivered product strictly depends on process. The effective software testing ensures that the software meets the quality standards and the customer requirements [2]. We know that all of the modern software applications are so complex in nature and are run in such an interdependent vicinity that complete testing can never be done.

Testing is a process that used to able to predict the quality of developed software products and it is able to identify the correct and error free product [1]. With that in mind, testing can never completely establish the correctness of computer software. Software testing is a strenuous and expensive process. Research has shown that at least 50% of the total software cost is comprised by testing activities [2-5].

Testing is not limited to the detection of “bugs” in the software. It also increases confidence in the software's proper functioning and assists with the evaluation of functional and nonfunctional properties. Testing related activities encompass the entire development process and may consume a large part of the effort required for producing software. Software testing remains the...
primary technique used to gain consumers confidence in the software. The process of testing any software system is an enormous task, which is both time-consuming and costly [2]. Software testing and software release is a big challenge for software industry thus; the question of how much to test a software and release is an important question for our software society [6]. The impact on the software industry due to lack of robust, standardized testing technology are things like increased failures due to poor quality, increased software development costs, inefficient testing, and increased market transaction costs etc. That’s why a tester should find out what the modules with greater importance are so that they can be tested first and assign more effort. It is impractical to test the software till all the bugs are removed. The tester should also be aware of the optimal testing time and the costs required to test the modules. When it is not possible to remove all the bugs with limited resources, then we have to accept limited faults in the software.

If the software components are not prioritized for testing, it might happen that the most critical components may not get tested within the prescribed optimal time. To overcome this drawback and to maximize the testing of critical components, we have introduced the concept of a fuzzy multi-criteria approach, fault tolerance and percentage stringency. This paper also considers faults along with the time and cost. The aim of this paper is to classify different modules into five precedence categories and to find out whether the software is ready to be released in the market after testing software for the given time and within the specified cost range.

This paper is organized as follows: in Section 2, an overview on background work is presented. Whereas, Section 3 describe the fuzzy multi-criteria approach. Section 4 essentially illustrates the software development effort and software testing effort. Section 5 presents an introduction to the testing effort, time, and fuzzy logic. In Section 6 an approach that uses proposed a prioritization technique with an optimizing cost and time technique are discussed. Section 7 essentially illustrates the proposed approach with the case study and finally, Section 8 presents conclusions and future work.

2. RELATED WORK

Software quality refers to the ability related properties of software products that meet user’s needs. According to its standards, the software quality evaluation is the software development process throughout the software quality and continuous measurement process. It reveals the current state, estimates the follow-up trend of software quality, and provides precise control of it [7]. Software quality can be evaluated quantitatively and manage qualititatively [8].

Software quality attributes are the benchmarks that describe a system’s expected behavior within the environment for which it was built. Software quality attributes provide visibility to management so that the software products and processes in the project life cycle conform to the specified requirements and established plans [9]. Various quality metrics and factors are given by [10], but it is hard to evaluate total software quality.

Measuring the quality of a software product is really quite a challenging task. A limited understanding of the relationships among the various software quality factors poses a challenge in the estimation of total software quality [11].

In a software quality model that took into account the three perspectives of a manager, user, and developer in calculating the total quality of a software product, we extended the methodology and assigned weights to each of the attributes and accommodated the three perspectives as
three different decision makers. Generally, a decision-making problem is defined by the presence of alternatives. The traditional approach consists in using only one criterion in order to select alternatives. However, using a single criterion is not sufficient when the consequences of the alternatives to be analyzed are important [12]. The goal of the Multi-criteria Decision making method consists in defining priorities from among different alternatives relating to actions, scenarios, and projects. In contrast to a mono criterion approach, Multi-criteria methods allow a more in-depth analysis of the problem because they consider different aspects in totality, which makes the analysis even more complicated.

There are several other instances where researchers have used fuzzy logic, neural networks, and other models to determine the quality of a software product. Previously [13] presented a new assessment method to obtain the integrated software quality for evaluating user satisfaction by using the fuzzy set theory, which is based on the sample quality model, with a single evaluator. Bo Yang et al. [14] had proposed a software quality prediction model that was based on the fuzzy of a neural network, which helps in identifying design errors in software products in the early stages of the lifecycle of the software.

There is an enhanced interest in developing new approaches to multi-criteria decision making where the criteria of interest and their relative importance are expressed by qualitative terms (such as low, medium, high, etc.). The imprecision that is inherent in qualitative descriptions of criterion values and criteria weights can be formalized using fuzzy logic and fuzzy set theory [15] [16] modeled objectives and attributes together, in order to form the decision space, which is represented by a fuzzy set whose membership function is the degree to which each alternative is a solution. Baas and Kwakernak [17] introduced fuzzy concepts in ranking, assuming that criteria values and the relative importance of criteria were fuzzy numbers. They [17] extended the classical weighted average rating method to handle fuzzy numbers. The final evaluation of alternatives was given by their membership functions by [17, 18] the employed algebraic operations of fuzzy numbers that are based on the extension principle. Carlsson C. and Fuller R. [19] presented a survey of fuzzy multi-criteria decision-making methods that places an emphasis on fuzzy relations between interdependent criteria [20]. A new elicitation method for assigning criteria importance based on linguistic variables was introduced by Ribeiro [21]. He introduced a new pair wise preference approach, which permitted for a homogeneous treatment of different kinds of criteria evaluations [21]. The fuzzy approach was also used for the modeling of the decision maker's preference structure, which consists of additional information that can be taken into account in the analysis (e.g., the decision maker's attitude toward the difference between the criterion values of two alternatives) [22].

The majority of the proposed approaches extend the classical simple additive weighting method by considering the linguistic values for criteria and weights. The approach described in this paper is based on measures of beliefs that one fuzzy rating is better/worse than the others are. The final rating of each alternative is given as a crisp value and therefore no sophisticated ranking procedure is required. In another work, the software requirement must be specified by a quality parameter that uses the fuzzy approach [23].

We use the fuzzy multi-criteria approach to deal with the fuzziness that is known to exist in software quality attributes. The method is suitable for making decisions in a fuzzy environment. To deal with uncertainty in the form of the fuzziness of the total quality of a software product, the important weights of various criteria and the ratings of qualitative criteria are considered as linguistic variables in this paper. Fuzzy logic is then applied to each of the criteria and using the
multi-criteria methodology, the total software quality or the best software proposal out of a set of proposals can be found.

Until now, the background was related to ranking the software. Now there is need after ranking the software, you must release whole component, for this we will discuss the various research works on test effort estimation and the release issues for software or component. Software development is very important and necessary in today’s world in any field of applied science and technology. However, optimal software strategy should be in such a way that it can best satisfy the constraints like cost, time, efforts, and other resources. This can be achieved by formulating a systematic software development lifecycle along with the Software Testing Effort to ensure that the above constraints can be fully satisfied before the software is released. Test engineering managers use many different methods to estimate and schedule their test engineering efforts. Different organizations use different methods depending on the type of projects, the inherent risks in the project, and the technologies involved, etc. There has been a lot of research on Software Testing but research on Software Testing Effort estimation has not received enough attention. Estimating the cost and duration of the software testing effort is a major challenge.

Currently, an early prediction of STE is based on testing metrics [24], which generally overestimate the efforts depending on the expertise of the software testing team that is developing the software. M. H. Halstead developed a set of metrics to measure a program module's complexity directly from the source code, with an emphasis on computational complexity [25]. The measures were developed as a means to directly determine the quantitative indication of the complexity from the operators and operands in the module. This study evaluates the overall software testing efforts that are to be allocated to a particular module. However, it has been observed that the study overestimates the analysis of software testing efforts because a significant complexity measure increase may be the sign of a brittle or high-risk module. Using the cognitive information complexity [26] method to estimate the testing effort, which in this case is the time required to comprehend the software, is proportional to its cognitive complexity. This work has made use of an existing simple cognitive metric that includes all the important parameters of the software required for estimating the test effort. This approach is able to demonstrate that the often-used cyclometric number [27] test effort increases with the increase in the software complexity. Jones Capers [28] estimated that the number of test cases could be determined by the function point’s estimate for the corresponding effort. Nagappan N. [29] proposes a metric suite model for the analysis of the software testing effort, which is based on historical data from comparable projects. Nagappan N[29]doesn’t propose any method to provide a testing effort. In another study on Test Point Analysis, a method for estimating the effort has been emphasized to perform all functional test activities based on use case points [30]. This model estimates the effort required for all test activities together, such as defining, implementing, and executing all the tests. Jørgensen [31] mentioned that effort estimation is based upon factors like human value, project data, project behavior, drivers, and other resources, but he does not explain how these values help in effort estimation. Aranha and Borba [32] propose a model for the test effort that is based on the test specifications. The main drawbacks for these model test cases are pre defined.

Xu S. and Rajlich V. [33] have presented a classifying program of debugging activities that are based on the cognitive activities. Their proposals are subjective and are not supported by any empirical results. Dawson [34] proposes a software test effort that is based on the neural network concept, but for the main drawback of the training data set and their effectiveness Xishi Huang et. al. [35] suggested to improved the cocomo model by using the fuzzy theory. Halstead
software science [36] suggests that an attempt has been made to estimate the software testing efforts with the help of code input. Recently Srivastava et al. [37] proposed a software testing effort that uses the neural network, but the main drawback was training data and cost drivers. They tried to optimize the software test effort based on Halstead’s software science using effective test drivers [38]. Srivastava Praveen Ranjan [39] discussed test effort and its calculation based upon evaluator knowledge. The next paragraph discusses work related to software release issues.

Extensive work has been carried out by a number of researchers in the area of optimal software time estimation. McDaid and Wilson [40], who gave three plans to settle the question of optimal time [40]. Software Testing stopping criteria based on the Stochastic and Economic Model proposed by Dalal et al. [41]. Musa and Ackerman [42] used the concept of reliability to make the decision for release. Ehrlich et al. [43], tried to find out the cost of a stop test decision. One of the most suitable models for the problem of determining optimal cost and time was proposed by Goel and Okumoto [44-46]. This was based on the non-homogenous poisson process model for determining the optimal cost and time for software. Hongyu Zhang [47] has shown that 80% of the faults in software are in 20% of the modules. So there is a need to identify those 20% modules that can be prioritized during the Non-Homogenous Poisson Process Based Model for determining how long to test the module. The model utilizes the concept of component prioritization based on cumulative priorities. Prioritizing the software modules ensures that the components are tested based on the requirements of the customer and the development team. In the component prioritization schema for achieving maximum time and cost benefits from software testing, they propose a prioritization scheme to rank software modules based on customer requirements, code complexity, function usage, requirements vulnerability, and reusability [49-52].

Testing is potentially endless and at some point, we have to stop testing and release the software. The question is when. Realistically, testing is a trade-off between budget, time, and quality. Profit models drive it. The pessimistic and, unfortunately, most often used approach is to stop testing whenever some, or any of the allocated resources (e.g., time, budget, or test cases are exhausted). The optimistic stopping rule is to stop testing when either reliability meets the requirement, or when the benefit from continuing testing cannot justify the testing cost [53].

From the above discussion, we found that ranking a module, estimating the testing effort, and releasing the model of any type of software in single platform is quite extensive and complex phenomena. That is why this research paper is looking for a solution on the basis of a prioritization scheme to rank the modules of a software project based on the fuzzy multi criteria approach and to find out whether the software is ready to be released in the market after testing the software for the given time within the specified cost. This paper also considers the fault tolerance along with time and cost.

### 3. THE FUZZY MULTI-CRITERIA APPROACH

Fuzzy multi-criteria decision-making is a systematic and formal decision model [54, 55]. As such, most of the real-life decision problems are characterized by the presence of interactive criteria. The interactions between the criteria must be considered to provide intelligent decision support to the decision maker. Traditional additive measures such as the weighted average are inadequate for modeling the interaction between the criteria. Thus, the fuzzy multi criteria deci-
Optimal Software Release Using Time and Cost Benefits via Fuzzy Multi-Criteria and Fault Tolerance

Fuzzy logic [56] implements human experiences and preferences via membership functions and fuzzy rules [57]. Fuzzy membership functions can have different shapes depending on the designer's preference and/or experience. The fuzzy rules, which describe relationships at a high level (in a linguistic sense), are typically written as antecedent consequent pairs of “IF-THEN” statements. Triangular fuzzy numbers are used based on arithmetic operations to express the decision maker’s evaluation on alternatives with respect to each criterion.

First of all, the Decision Maker (DM) requires knowledge about the consequences of the decisions. In order to predict the behavior of the system the problem has to be described by a model. It is a crucial task to analyze and identify the relevant system components and their interactions, the internal structure, and dynamics, as well as external interfaces and influential factors. Multi Objective Decision Making should be a process of steady collaboration between analysts and decision makers. On the one hand, through interaction and communication the decision maker can bring in his preferences, the analyst on the other hand may support the DM in gaining better understanding for the process [12].

4. SOFTWARE DEVELOPMENT EFFORT AND SOFTWARE TESTING EFFORT

This is the process of predicting the most realistic use of effort required to develop or maintain software based on incomplete, uncertain, and/or noisy input [58]. Effort estimates may be used as input to project plans, iteration plans, budgets, investment analyses, pricing processes, and bidding rounds [1]. There are several models for estimating the Software Development Effort (SDE). Some of them are the Constructive Cost Model (COCOMO), Function point analysis, and so on [1],[59]. The COCOMO is an algorithmic software cost estimation model that was developed by Barry Boehm in 1981. This model uses a basic regression formula with parameters that are derived from historical project data and current project characteristics. The COCOMO model is the most widely used model for estimating the software development effort [60]. It is estimated as being 40 to 50 percent of the development effort [5]. The fundamental premise here is that test-engineering efforts are dependent on the time and/or cost of SDE. First, development effort is estimated using the technique discussed above and the next step is using some heuristic to predict a value next to it for testing. This varies widely and is usually based on previous experiences. However, there is no standard procedure to determine an accurate value. Therefore, there is a need to provide an approximation of the testing effort that acts as a valuable tool for the developers and practitioners.

5. INTRODUCTION OF SOFTWARE EFFORT, COST TIME, AND FUZZY LOGIC

Delivery of an error free product (e.g., software that is within budget and schedule) is the top most priority [59] of any organization. This in turn calls for reasonably good estimates on effort, time, and cost, and of course quality, which all effect schedule and effort. Software cost and effort estimation will never be an exact science [61]. Too many variables like humans, technical
factors, the environment, and political factors can affect the ultimate cost of software and effort applied to develop it. Software Cost estimation (e.g., the task to calculate the value of the effort and the schedule required to develop and maintain a software system) is one of the most critical activities in managing software projects. Among the several model CONstructive COST MoDEL (COCOMO) is the most widely used model for estimating the software development effort [51]. It is an algorithmic software cost estimation model that was developed by Barry Boehm in 1981. This model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics. It is a type of algorithmic model where input is the project size and a few evaluations of other parameters [1]. COCOMO is one of the best-documented cost-estimation models, which is based on line of code (LOC). The first level, Basic COCOMO, is good for a quick, early, and rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (Cost Drivers) [1]. Intermediate COCOMO [58-60] computes software development efforts as functions of program size and a set of “cost drivers” that include a subjective assessment of the product, hardware, personnel, and project attributes, and each main attribute with a number of subsidiaries attributes.

Each of these attributes receives a rating on a six-point scale that ranges from “very low” to “extra high” (in importance or value). The product of all effort multipliers results in an effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4. The Software Development effort is calculated by using the formula [58, 59].

Typical values for EAF range from 0.9 to 1.4. The Software Development effort calculates by using the formula below [58, 59].

\[
E = a_i \times (KLOC)^{b_i} \times EAF
\]

where E is the effort applied in person-months, KLOC is the estimated number of thousands of delivered lines of code for the project, and EAF is the factor calculated above. The coefficient \(a_i\) and the exponent \(b_i\) depend on the structure of the organization. Table 1 below shows the type of organization and the corresponding values of coefficients.

Software Testing is an empirical investigation that is conducted to provide stakeholders with information about the quality of the product or service under the test, with respect to the context in which it is intended to operate. Software Testing also provides an objective and independent view of the software to allow the business to appreciate and understand the risks at the implementation of the software. It can also be stated as the process of validating and verifying that a software program/application/product meets the business and technical requirements [62]. Testing is a very major part in the software development cycle. A huge amount of effort is invested in testing. The computation of cost and time effort is not an easy task because it depends upon the exact value of SDE and test effort drivers etc. There are two main disadvantages for calculating development effort when using the above COCOMO model. First, the model makes an assumption on the form of the function consisting of some of the known factors, and is represented by: \(E = a_i \times (KLOC)^{b_i} \times EAF\). The coefficient \(a_i\) and the exponent \(b_i\) depend on the structure of the organization. To evaluate the attributes, certain linguistic values are assigned such as “very low,” “complex,” “important,” and “essential.” When dealing with linguistic values, imprecision, uncertainty, and partial truth are inevitable. However, until now, the factors that are taken into consideration for software measurement are numbers or classical intervals for representing these
linguistic values. Furthermore, such transformation and representation does not copy the way in which human beings take into mind these linguistic values and as a result cannot deal with imprecision and uncertainty. To overcome these limitations, this paper suggests the application of fuzzy logic concepts, which have been described in the subsequent paragraphs.

5.1 FUZZY LOGIC

Fuzzy logic is basically a multi-valued logic that allows intermediate values to be defined between conventional evaluations like yes/no, true/false, black/white, etc. Notions like rather warm or pretty cold can be formulated mathematically and processed by computers [65]. The broadly, problems dealing with fuzzy logic concepts can be divided into the following three steps:

1. Fuzzification of data: in this step, the fuzzification process is carried out by developing membership functions that have been generated from different input sources.
2. Development of the Fuzzy Rule Base: the fuzzy rule base is usually constructed from the experience of the decision maker. This phase comprises of applying the fuzzy rule base over the fuzzy input and arriving at the fuzzy output.
3. Defuzzification of Output Data: this converts the fuzzy output into crisp output. There are several ways to achieve defuzzification. Some of them are the mean-of-maxima method, the center of gravity method, the modified center of gravity method, and the height method, etc. In this paper, the center of gravity technique has been used for defuzzification.

6. PROPOSED APPROACH

The component based prioritization technique for optimizing the time and cost using the fuzzy multi-criteria approach and fault tolerance have a series of steps, which are shown in Fig. 1.

<table>
<thead>
<tr>
<th>Type of organization</th>
<th>Development environment</th>
<th>a_i</th>
<th>b_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>Familiar</td>
<td>3.2</td>
<td>1.05</td>
</tr>
<tr>
<td>Semidetached</td>
<td>Medium</td>
<td>3.0</td>
<td>1.12</td>
</tr>
<tr>
<td>Embedded</td>
<td>Complex</td>
<td>2.8</td>
<td>1.20</td>
</tr>
</tbody>
</table>

Table 1. Coefficient for COCOMO [Boehm Barry, 1981]

Fig. 1. The various steps of our proposed approach
The presented approach is based on the prioritization of different modules in the large software project and is performed based on the qualitative criteria, which are often imprecisely defined for the decision-makers. It is therefore difficult to clearly identify the “best” among them. Therefore, this uncertainty in the form of impreciseness in prioritizing the modules in software products adopts a fuzzy decision-making method by deriving membership functions and a fuzzy preference relation matrix [54]. The main purpose for the prioritization of the modules is to make the best use of the software testing for optimizing software testing time and cost. Here we are considering the attributes for prioritizing the modules such as component rank, kilo lines of code, cyclometric complexity, customer priority, and the average of the KLOC of all the modules in the project [48].

These factors may be considered as different criteria to select the best alternative from among

---

**Fig. 2. Various steps of the fuzzy based selection process**

1. Identify the selection criteria, its attribute, and possible alternatives for each criterion.
2. Compute the linguistic variables for the important weight of each criterion.
3. The decisions maker’s choice for the important weight for each criterion.
4. Evaluate the fuzzy weight for each criterion.
5. Compute the linguistic variable for ratings.
6. The decisions maker’s rating choice for alternatives depending on the criteria.
7. Derive the fuzzy decisions matrix.
8. Derive the fuzzy normalized decision matrix.
9. Evaluate the fuzzy evaluation values of the three alternatives.
10. Evaluate the fuzzy preference relation matrix.
11. Evaluate the fuzzy strict preference relation matrix.
12. Compute the non-dominated degree of each alternative.
13. Rank the alternative depending on the non-dominated degree.
the given alternatives using fuzzy based multi-criteria analysis. The method is suitable for making a decision in a fuzzy environment. To deal with uncertainty in the form of the fuzziness of the selection process of modules, the important weights of various criteria and the ratings of qualitative criteria are considered as linguistic variables in this paper. These linguistic variables can be expressed in triangular fuzzy numbers, which is defined by a triplet \((n_1, n_2, n_3)\). The membership function for such triangular fuzzy number has been calculated accordingly [23, 66].

By expressing the important weight and ratings of each criterion in terms of linguistic variables, different alternatives have been evaluated for different criteria depending on the choices of decision makers using fuzzy logic concepts. The series of steps for are shown in Fig. 2 [23, 66].

We know that the performance of a software system largely depends upon testing time. A longer testing phase leads to enhanced performance and also the cost of fixing a default during operation is generally much more than during testing. However, the time spent in testing delays the product release, which leads to additional costs. Hence, the objective is to determine the optimal release time to minimize the cost by reducing the testing time.

After prioritizing the module, we have to optimize our time and cost so that we can ship our project as early as possible. In this paper an effort has been made to compute the approximate value of the Software Testing Effort (STE) by using the intermediate COCOMO model and confidence level of the project manager. The SDE value is estimated by integrating the confidence level with the COCOMO Model by using fuzzy logic [67].

**Steps for Estimating the Software Development Effort (SDE) [67]:**

An algorithm is constructed, as given below, in order to assess the STE along with the block diagram, as shown in Fig. 3 and Fig. 4.

Step 1. Prepare the Software Requirement Specification (SRS) for any project/module.

Step 2. Estimate the Kilo Line of the Source Code (KLOC) for a project/module and the type of organization.

Step 3. Calculate the confidence (C) value of the project manager based upon the Experience

![The flow chart for the Calculation of the Software Testing Effort](image-url)

Fig. 3. The flow chart for the Calculation of the Software Testing Effort
Praveen Ranjan Srivastava

3.1. Take the number of years of experience and the number of similar projects done by the project manager as the crisp input.

3.2. Fuzzify this crisp input using the membership functions.

3.3. Apply the fuzzy rule base on this fuzzy input to evaluate the fuzzy output.

3.4. Defuzzify this fuzzy output to get the confidence value.

Fig. 4. Flow chart for the calculation of the confidence value

and Familiarity of the project. Do so by using the fuzzy rule base, the max-min composition, and defuzzification.

Step 4. Calculate the approximate value of the line of the source code (e.g., KLOC"= KLOC*C.)

Step 5. Calculate the approximate Software Development Effort (E’) using the COCOMO formula and KLOC’.

**Approximate Estimation of the Software Development Effort (SDE) [67]**

The estimation of the SDE with the help of the COCOMO model depends upon the KLOC, which is generally estimated by the senior project manager on the basis of his/her ability. This section mainly deals with two main factors, which evaluate the confidence value.

1. Experience: the experience of the project manager, who gave an estimation of the LOC, plays a major role. The experience of a project manager can be quantified by the number of years the project manager has been working.
2. Familiarity: the familiarity of the project manager with the current project also plays a significant role. The number of similar projects done earlier can quantify the familiarity.

Thus, the confidence value of a project manager is dependent upon the KLOC derived from the experience and the familiarity of the software project manager. Both of the factors are uncertain and can be expressed in qualitative terms like more, less low, high, etc. Therefore there is a need to apply a fuzzy based approach to quantify these qualitative terms by deriving suitable membership functions to evaluate the approximate value of the SDE.

These membership functions can be of different types such as triangular membership functions, linear membership functions, etc. depending upon the experience of the decision maker/project manager. Using the crisp input as the number of years of experience of a project manager and a qualitative term such as low, medium, or high, membership functions can be generated by fuzzifying it as shown in Fig. 5.

Similarly, by using the crisp input as the number of similar projects done by a project man-
ager and a qualitative term of familiarity such as low, medium, or high, membership functions can be generated by fuzzifying it as shown in Fig. 6.

After generating the membership values of experience and familiarity, the fuzzy rule base is constructed to arrive at the fuzzy output as given in Table 2 where LC stands for low confidence, MC for medium confidence, and HC for high confidence [67].

From Table 2, the fuzzy rules can be stated as follows:
- If the experience is low and the familiarity is low, the confidence is low.
- If the experience is high and the familiarity is high, then the confidence is high.
- If the experience is medium and the familiarity is high, the confidence is high.

After applying the fuzzy rule base and max-min composition [68, 69], the membership values
As the output of confidence values obtained by this process is fuzzy, it has to be defuzzified using the center of gravity to arrive at the crisp output.

After achieving the crisp output of the confidence value (say C varying between 0.7 and 1 as shown in Fig. 7), the estimated number of lines of code (KLOC) is modified as:

\[ KLOC' = KLOC \times C. \]  

(2)

Once a new project estimation size (KLOC’) with full confidence is estimated, an intermediate COCOMO model can be applied to calculate the approximate Software Development Effort (say E’):

\[ E' = a_i \times (KLOC')^{b_i} \times EAF \]  

(3)

where \(a_i\) and \(b_i\) are derived from Table 1.

It may be noted that the development effort (E’) that is estimated using equation (3) is always lesser than the development effort (E) estimated from equation (1).

After evaluating the Software Development Effort (SDE), the percentage of the software development effort that is supposed to be invested in testing should be determined. This percent-

---

**Table 2. Fuzzy rule base for determining Confidence values**

<table>
<thead>
<tr>
<th>Familiarity</th>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>M</td>
<td>LC</td>
</tr>
<tr>
<td>H</td>
<td>MC</td>
</tr>
</tbody>
</table>

---

**Fig. 7. Membership function for Calculating confidence**
age is purely based on heuristics and there is no standard procedure for arriving at this percentage, as the process of software testing is a complicated task. However, it may vary around 50% as discussed in the previous section. The stringency values are uncertain. Hence, in the subsequent section, an attempt has been made to estimate the true value of percentage by using fuzzy logic.

After prioritizing the modules in 5 categories (as in Table 3) and calculating the software development effort, this paper suggests finding out the maximum allowable cost and time for the testing. Then it advises to start testing the software to calculate the actual time and actual cost for each priority category. The deviation from optimal testing time and optimal cost can be calculated from equation (4) and equation (5) [48-52].

\[
\alpha = \frac{(T_a - T^*)}{T^*} \quad (4)
\]

Where \(\alpha\) = deviation from optimal time
\(T_a\) = actual testing time
\(T^*\) = optimal testing time

And
\[
\beta = \frac{(C_a - C_o)}{C_o} \quad (5)
\]

Where \(\beta\) = deviation from optimal cost
\(C_a\) = actual testing cost
\(C_o\) = optimal testing cost

Then we start testing and note down the actual time, cost, and percentage of faults obtained for each category. Note that these values are obtained after complete testing of the entire individual category has taken place. Afterward we find out the deviation from the optimal testing time and the optimal cost from equation (4) and (5).

Since, time and cost might have different weights depending upon organizational needs; we modify the formula for calculating \(\delta\). The new calculations for determining \(\delta\) are given by:

\[
\delta = m \times \alpha + n \times \beta \quad (6)
\]

where \(m\) and \(n\) are constants that are determined by organization in such a manner that their sum turns out to be 1. These constants are useful in giving different weights to cost and time.

Afterwards it cumulatively calculates the limiting factor \(\delta\) to determine whether further software testing is required. Since the weight of cost and time can’t be the same [50, 51]. For this purpose we have used different stringency values for both of them. The deviation value varies for different organizations. A sample of stringency values is given in Table 3.

Let \(T\) and \(C\) be the total time and cost available to release the software. Our aim is to test all the modules within \(T\) and \(C\). But if we are not able to do this then at least the Very High, High, and Medium ranged modules should be tested. We set the fault tolerance = 0 for the first time testing of all the modules of a particular category (e.g., Very High) and find out the actual time and cost for testing.

Table 3 is used to compute the maximum value of \(\delta\) for a given category. For example, with the sample stringency values given in Table 3, \(\delta_{\text{max}}\) for the Very High category will be \(m (0.25)\)
If the values of m and n are 0.7 and 0.3 respectively then $\delta_{\text{max}}$ for the Very High category is $0.7 \times 0.25 + 0.3 \times 0.20 = 0.23$. Thus we can calculate the $\delta_{\text{max}}$ for all the categories. A summary is given in Table 4 (with $m = 0.7$ and $n = 0.3$).

If the $\delta$ values obtained fall inside the $\delta_{\text{max}}$ limits, we understand that this category had faults within the estimated fault limit so we move further to test lesser priority modules. But, if the $\delta$ values exceed the estimated boundary, we calculate the faults obtained in the testing to compare it with the fault tolerance. Now there can be 2 cases:

**Case I. actual faults $\leq$ fault tolerance**

In this situation, we suggest proceed to the lesser priority components. If the faults are within the tolerance range, then these remaining faults can be corrected at the maintenance time. Sometimes it may also happen that there are no faults in the tested category but still the observed testing time is higher. This is because of bad testing policy, the inexperience of the tester, or some other environmental factor.

**Case II. actual faults $>$ fault tolerance**

Here, we have to debug the modules based on the faults detected till either the observed faults are within the tolerance or the resources are finish. After testing each time, we increase the fault tolerance a little bit. It prevents us from getting stuck in an infinite loop of testing the same category again and again. We are following this approach because we want to test as many modules within a limited amount of software release time. However, our aim is to debug severe bugs residing in the software.

Initially we assume zero fault tolerance for the first iteration. Then we calculate a new fault tolerance for each iteration, as given by equation (7):

$$\text{New fault tolerance} = \min (\max\_\text{tolerance}, \% \text{faults obtained in last iteration} - \text{min\_improvement}) \quad (7)$$

Where the max\_tolerance is the maximum fault tolerance variable whose value increases by 2%
and the min_improvement is the minimum fault improvement variable whose value increases linearly after each iteration. The maximum value of the max_tolerance is 10%. After each iteration we are increasing the value of the max_tolerance because our resources are becoming more and more crucial. On the other hand, we increase the min_improvement because it ensures fewer faults than the previous iteration. The value of variables varies from organization to organization depending upon their needs.

We repeatedly test a particular category and calculate a new time and costs until the errors come within the fault tolerance limit, each time exceeding the fault tolerance. If the number of faults obtained in several consecutive iterations are the same (e.g., we are not getting any further improvement even after many iterations), than management can decide to transfer this module to some other tester. At the end of T and C, we should be able to test Very High, High, and Moderate modules. If we are stuck in a particular category for a very large amount of time and are not able to finish these 3 categories, then we report to the managers that the software is error-prone and that it is entirely their risk in launching the software into the market. It should be noted that even if we are not able to test the Low and Very Low categories we prefer to launch the software.

Hence, this is the proposed approach by which we can prioritize the modules in the software product and to find out the actual cost of allowable time and cost for testing.

The above method is illustrated in the next section on real time software.

7. CASE STUDY

The main objective of presenting this case study is to incorporate the above-mentioned approach to prioritize the modules and to optimize the testing time and effort with the help of fuzzy logic during the development of software for a real project.

Case Study (In Details):

We are considering a in house developed project, title as course management system (internally done), which consists of five different modules. The above-mentioned approach is finally applied for five modules in a project to produce a good software product with optimal software testing time and cost.

In course management system we have five alternatives (namely M1, M2, M3, M4, and M5) as modules to make a good software quality product.

As a policy maker we have to take the opinion of three experts from a company (namely DM1, DM2, and DM3) for different factors to be considered for the best selection depending upon the relative importance given by these experts. These factors are dependent on prioritizing software modules [50-52]. For the sake of giving an example, we are considering such as the component rank (C1), kilo lines of code (C2), cyclometric complexity (C3), customer priority (C4), and the average KLOC of all the modules in the project (C5)[66].

Keeping with the various steps of the fuzzy based selection process described in Fig. 1, the evaluation was carried out to perform analysis for choosing the best modules from amongst the available alternatives. In this study a total of five important selection criteria, its attributes, and all five possible alternatives have been identified and the linguistic variables for assigning important weights to each criterion (like very low [VL], low [L], medium [M], high [H], and very high [VH]) have been defined in the form of a triplet of triangular fuzzy numbers. The scale of
these linguistic variables is between the values of 0.0 to 1.0.

The weights of all linguistic variables are shown below in Table 5.

These variables are then used to assess the importance of each criterion by the experts DM1, DM2, and DM3 and are depicted in Table 6.

The fuzzy weight of each criterion is then calculated using the assigned weight of the corresponding linguistic variable and the important weight of each attribute as given in Table 6. The fuzzy weight of any jth criteria can be calculated as:

$$w_{j \text{-fuzzy}} = \frac{w_{j}^{1} + w_{j}^{2} + \ldots + w_{j}^{k}}{k}$$

where k is the number of experts and $w_{j}^{k}$ is the weight assigned by the kth expert for jth criteria. For example, in Table 6, for the criterion related to the Component rank (C1), there are different opinions from three experts (e.g., H, VH, and H), which correspond to the triangular membership value weight as (0.5, 0.7, 0.9), (0.7, 0.9, 1.0), and (0.5, 0.7, 0.9) respectively. Therefore, the fuzzy weight of any jth criteria (say related to the component rank) can be calculated as:

$$\left(\frac{0.5 + 0.7 + 0.5}{3}, \frac{0.7 + 0.9 + 0.7}{3}, \frac{0.9 + 1.0 + 0.9}{3}\right)$$

e.g., $W_{j \text{-fuzzy}} = (0.56, 0.76, 0.93)$

Similarly, other values have been calculated and entered into Table 7.

The linguistic variables for assigning ratings for each criterion (like very poor [VP], poor [P], fair [F], good [G], and very good [VG]) have also been classified in the form of a triplet of triangular fuzzy numbers. The scale for these linguistic variables lies between the values of 1 to 10.

Table 5. The weights of the linguistic variable of each criterion

<table>
<thead>
<tr>
<th>Linguistic Variable</th>
<th>Corresponding Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL</td>
<td>(0.0, 0.1, 0.3)</td>
</tr>
<tr>
<td>L</td>
<td>(0.1, 0.3, 0.5)</td>
</tr>
<tr>
<td>M</td>
<td>(0.3, 0.5, 0.7)</td>
</tr>
<tr>
<td>H</td>
<td>(0.5, 0.7, 0.9)</td>
</tr>
<tr>
<td>VH</td>
<td>(0.7, 0.9, 1.0)</td>
</tr>
</tbody>
</table>

Table 6. The important weight of each attribute

<table>
<thead>
<tr>
<th>SNo</th>
<th>Attributes</th>
<th>DM1</th>
<th>DM2</th>
<th>DM3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CR (C1)</td>
<td>H</td>
<td>VH</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>KLOC (C2)</td>
<td>VH</td>
<td>VH</td>
<td>H</td>
</tr>
<tr>
<td>3</td>
<td>CC (C3)</td>
<td>H</td>
<td>VH</td>
<td>L</td>
</tr>
<tr>
<td>4</td>
<td>CP (C4)</td>
<td>M</td>
<td>VH</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>KLOC_{avg} (C5)</td>
<td>M</td>
<td>H</td>
<td>M</td>
</tr>
</tbody>
</table>
The weight of these entire linguistic variables for assigning a rating for each criterion is shown in Table 8.

These variables are then used to assess the ratings of each criterion by the experts DM1, DM2, and DM3 for all five possible alternatives that are presented in Table 9. The systematic procedure to evaluate the local and normalized fuzzy decision matrix, the fuzzy preference relation matrix, and finally ranking the possible alternatives is described below:

1. The fuzzy decision matrix is derived by using the defined ratings of each criterion with respect to each of the possible alternatives modules. Each element of this matrix, say $r_{ij}$, is the fuzzy rating of any alternative $A_i$ with respect to any $j$th criteria assigned by the experts and can be calculated as:

$$r_{ij} = \frac{r_{ij}^1 + r_{ij}^2 + \ldots + r_{ij}^k}{k}$$

where $k$ is the number of experts and $r_{ij}^k$ is the weight assigned by the $k$th expert for the $i$th alternative with respect to the $j$th criteria. For example, in Table 9, for the fifth criteria related to the KLOC avg. (C5), there are different opinions from three experts (e.g., good, fair, and good)

<table>
<thead>
<tr>
<th>S.N</th>
<th>Attributes</th>
<th>DM1</th>
<th>DM2</th>
<th>DM3</th>
<th>DM4</th>
<th>DM5</th>
<th>DM6</th>
<th>DM7</th>
<th>DM8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CR (C1)</td>
<td>G</td>
<td>G</td>
<td>VG</td>
<td>VG</td>
<td>VG</td>
<td>F</td>
<td>VG</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>KLOC (C2)</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>VG</td>
<td>F</td>
<td>VG</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>3</td>
<td>CC (C3)</td>
<td>F</td>
<td>F</td>
<td>G</td>
<td>VG</td>
<td>F</td>
<td>VG</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>CP (C4)</td>
<td>F</td>
<td>P</td>
<td>P</td>
<td>G</td>
<td>VG</td>
<td>F</td>
<td>VP</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>KLOC avg (C5)</td>
<td>G</td>
<td>F</td>
<td>G</td>
<td>F</td>
<td>F</td>
<td>G</td>
<td>G</td>
<td>VG</td>
</tr>
</tbody>
</table>

The weight of these entire linguistic variables for assigning a rating for each criterion is shown in Table 8.

These variables are then used to assess the ratings of each criterion by the experts DM1, DM2, and DM3 for all five possible alternatives that are presented in Table 9. The systematic procedure to evaluate the local and normalized fuzzy decision matrix, the fuzzy preference relation matrix, and finally ranking the possible alternatives is described below:

1. The fuzzy decision matrix is derived by using the defined ratings of each criterion with respect to each of the possible alternatives modules. Each element of this matrix, say $r_{ij}$, is the fuzzy rating of any alternative $A_i$ with respect to any $j$th criteria assigned by the experts and can be calculated as:

$$r_{ij}^\text{fuzzy} = \frac{r_{ij}^1 + r_{ij}^2 + \ldots + r_{ij}^k}{k}$$

where $k$ is the number of experts and $r_{ij}^k$ is the weight assigned by the $k$th expert for the $i$th alternative with respect to the $j$th criteria. For example, in Table 9, for the fifth criteria related to the KLOC avg. (C5), there are different opinions from three experts (e.g., good, fair, and good)
for alternative 1 (e.g., A₁), which corresponds to the triangular membership value rating as (5.0, 7.0, 9.0), (3.0, 5.0, 7.0), and (5.0, 7.0, 9.0) respectively. Therefore the fuzzy rating of any jth criteria (say related to probability) can be calculated as:

\[
r_{ij-\text{fuzzy}} = r_{15-\text{fuzzy}} = \left( \frac{5.0 + 3.0 + 5.0}{3}, \frac{7.0 + 5.0 + 7.0}{3}, \frac{9.0 + 7.0 + 9.0}{3} \right)\]

\[
r_{15-\text{fuzzy}} = (4.33, 6.33, 8.33)\]

Similarly, the other entries of this matrix are evaluated as such.

(2) For transforming all the criteria evaluation into a common scale, a global or normalized fuzzy decision matrix is derived as calculated in Table 10. Any entry into this global matrix, say:

\[
g_{ij-\text{fuzzy}} = \left( \frac{4.33}{10}, \frac{6.33}{10}, \frac{8.33}{10} \right)\]

where the numerator values of this entry are taken from the fuzzy decision matrix evaluated above (say 4.33, 6.33, 8.33) and the denominator value is the maximum weighted value that is available in the fuzzy decision matrix derived in step (1) for any alternative with respect to any of the criteria. In this case it is given as 5.6, 7.63, 9.3 for the first alternative with respect to Criteria 1 and hence the denominator value will be 10. Similarly, other entries for the normalized fuzzy decision matrix are calculated and tabulated in Table 10.

(3) Using the different importance levels of each criterion for the given alternative and the elements of the fuzzy global decision matrix, the fuzzy evaluation value of each alternative is calculated for all criteria j = 1, 2 … n. It can be expressed as:

\[
E_j = \sum_{j=1}^{n} g_{ij-\text{fuzzy}} \times w_{j-\text{fuzzy}}
\]

For example, the final fuzzy evaluation value in terms of the triangular membership function for alternative 1, \( E_1 = (E_{11}, E_{12}, E_{13}) \) where \( E_{11} \) is the value of the final fuzzy evaluation for alternative 1 with respect to the first value of the triangular fuzzy number (e.g., the value corresponding to the membership function value of zero), \( E_{12} \) is the value of the final fuzzy evaluation for alternative 1 with respect to the second value of the triangular fuzzy number (e.g., the value corresponding to the membership function value of one), and \( E_{13} \) is the value of the final fuzzy evaluation for alternative 1 with respect to the third value of the triangular fuzzy number (e.g., the value corresponding to the membership function value of two).

Table 10. The Fuzzy Normalized Matrix for every criteria/attributes of each module

<table>
<thead>
<tr>
<th>S.N</th>
<th>Attributes</th>
<th>M₁</th>
<th>M₂</th>
<th>M₃</th>
<th>M₄</th>
<th>M₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CR (C₁)</td>
<td>(0.56,0.76,0.93)</td>
<td>(0.70,0.90,1.0)</td>
<td>(0.50,0.70,0.86)</td>
<td>(0.63,0.83,0.96)</td>
<td>(0.50,0.70,0.90)</td>
</tr>
<tr>
<td>2</td>
<td>KLOC (C₂)</td>
<td>(0.50,0.70,0.90)</td>
<td>(0.56,0.76,0.9)</td>
<td>(0.50,0.70,0.90)</td>
<td>(0.36,0.56,0.76)</td>
<td>(0.30,0.50,0.70)</td>
</tr>
<tr>
<td>3</td>
<td>CC (C₃)</td>
<td>(0.36,0.56,0.76)</td>
<td>(0.50,0.70,0.86)</td>
<td>(0.63,0.83,0.96)</td>
<td>(0.43,0.63,0.83)</td>
<td>(0.50,0.70,0.86)</td>
</tr>
<tr>
<td>4</td>
<td>CP (C₄)</td>
<td>(0.16,0.36,0.56)</td>
<td>(0.50,0.70,0.86)</td>
<td>(0.13,0.30,0.50)</td>
<td>(0.50,0.70,0.86)</td>
<td>(0.56,0.76,0.93)</td>
</tr>
<tr>
<td>5</td>
<td>KLOC avg (C₅)</td>
<td>(0.43,0.63,0.83)</td>
<td>(0.30,0.50,0.70)</td>
<td>(0.50,0.70,0.90)</td>
<td>(0.50,0.70,0.86)</td>
<td>(0.50,0.70,0.86)</td>
</tr>
</tbody>
</table>
responding to the membership function value of 1), $E_{13}$ is the value of the final fuzzy evaluation for alternative 1 with respect to the third value of the triangular fuzzy number (e.g., the value corresponding to the membership function value of zero). The value of these fuzzy evaluations can be calculated by the following formula:

$$E_{11} = (0.56 \times 0.56) + (0.63 \times 0.50) + (0.43 \times 0.36) + (0.5 \times 0.16) + (0.36 \times 0.43) = 1.02$$

Similarly, $E_{12} = 2.12$ and $E_{13} = 3.45$ and therefore, the final fuzzy evaluation value for alternative 1 is $E_1 = (1.02, 2.12, 3.45)$. In the same way, the final fuzzy evaluation value for alternatives 2, 3, 4, and 5 are calculated respectively as:

$$E_2 = (1.32, 2.52, 3.75)$$

$$E_3 = (1.11, 2.24, 3.61)$$

$$E_4 = (1.19, 2.37, 3.68)$$

$$E_5 = (1.14, 2.31, 3.65)$$

(4) The fuzzy differences between the upper and lower values for all possibly occurring combinations have been calculated and presented below:

$$Z_{12l} = E_{1l} - E_{2u} = (1.02 - 3.75) = -2.73$$

$$Z_{12u} = E_{1u} - E_{2l} = (3.45 - 1.32) = 2.13$$

$$Z_{12} = [Z_{12l}, Z_{12u}] = [-2.73, 2.13]$$

(5) The fuzzy preference relation matrix is calculated in the following ways: the lower and upper values that correspond to the zero membership function (e.g., $Z_{ijl}$ and $Z_{iju}$) are already calculated in the above step. Assuming the triangular membership function with the equilateral triangle, the value of $Z_{ij}$ corresponding to the membership value of 1 can be calculated as:

$$Z_{ij}^1 = \frac{Z_{ijl} + Z_{iju}}{2}$$

For example:

$$Z_{12}^1 = \frac{-2.73 + 2.13}{2}$$

Therefore the coordinates of the vertices of the triangular membership function curve corresponding to $z_{12}$ will be:

$$Z_{12t}, \mu (Z_{12t}) \text{ at vertex } 1 = (-2.73, 0) \text{ and}$$

$$Z_{12t}^1, \mu (Z_{12t}^1) \text{ at vertex } 2 = (-0.30, 1)$$

$$Z_{12u}, \mu (Z_{12u}) \text{ at vertex } 3 = (2.13, 0)$$

respectively. Now taking the region for $S_1$ for $Z_{12} > 0$, the value of the membership function $\mu (Z_{12})$ can be calculated corresponding to the zero value of $Z_{12}$. Using the linear interpolation, for
\( Z_{12} = 0 \) is calculated as shown below.

As we have three coordinates \((x_0, y_0), (x, y), \) and \((x_1, y_1)\) they have the values of \((-0.30, 1), (0, y)\) and \((2.13, 0)\) respectively.

\[
y = \frac{y_0 + (x - x_0)(y_1 - y_0)}{(x_1 - x_0)}
\]

By using the above equation we calculated \( \mu(Z_{12}) = 0.87. \) If \( Z_{ij} > 0, \) then alternative \( M_i \) is absolutely preferred to \( M_j. \) If \( Z_{ij} < 0, \) then alternative \( M_i \) is not preferred to \( M_j. \) If \( Z_{ij} < 0 \) and \( Z_{ji} > 0, \) the degree of preference of alternative \( M_i \) over alternative \( M_j \) can be obtained by introducing the term \( p_{ij}. \) This term \( p_{ij} \) may be expressed as membership function \( \mu_{zi}(x) \) and the fuzzy preference relation matrix \( (PR) \) may thus be expressed as:

\[
PR = \begin{bmatrix}
pr_{11} & pr_{12} & pr_{13} & pr_{14} & pr_{15} \\
pr_{21} & pr_{22} & pr_{23} & pr_{24} & pr_{25} \\
pr_{31} & pr_{32} & pr_{33} & pr_{34} & pr_{35} \\
pr_{41} & pr_{42} & pr_{43} & pr_{44} & pr_{45} \\
pr_{51} & pr_{52} & pr_{53} & pr_{54} & pr_{55}
\end{bmatrix}
\]

where \( pr_{11} = pr_{22} = pr_{33} = 0.5 \) and other entries of the matrix when \( i \neq j \) are calculated as follows:

\[
p_{ij} = \frac{\text{Area covered under the triangular membership function curve varying from 0 to } z_{iju}}{\text{Total area covered under the triangular membership function curve varying from } z_{ijl} \text{ to } z_{iju}}
\]

So now, we can plot the triangular membership function curve for \( \mu(Z_{12}). \) Calculate the areas covered under the triangular membership function varying from 0 to \( Z_{12} \), which is denoted by \( S_1. \) The total area covered under triangular membership function curve from \( Z_{12l} \) to \( Z_{12u} \) is denoted by \( S_2. \) The calculation for \( pr_{12} \) is shown below.

The coordinates for \( S_1 \) are \((0, 0.94)\) and \((2.66, 0)\). Hence, by using linear interpolation we have: \( y = 0.87 - 0.41x \)

Similarly for \( S_2 \) we have coordinates \((-3.02, 0)\) and \((-0.18, 1)\). Hence, by using linear interpolation we have: \( y = 0.41x + 1.12 \)
To determine the most suitable alternative from among the given alternatives, it is necessary to evaluate the degree of strict dominance of alternative $M_i$ over alternative $M_j$, which is given as follows:

$$PRS_{ij} = P_{ij} - P_{ji}$$

0 otherwise

And therefore the fuzzy strict preference relation matrix is calculated from the final value fuzzy preference relation matrix as shown below:

The calculated fuzzy strict preference relation matrix is as follows:

(7) The non-dominated degree (NDD) of each alternative $M_i$ (for $i = 1, 2, 3, 4, 5$) is evaluated from the fuzzy strict preference relation matrix, which was derived above, using the expression:

$$NDD (A_i) = 1 - \max_{j \in A_i} (PRS_{ji})$$

Therefore, the non-dominated degree (NDD) of each alternative $M_1$ would be equal to $1 - \max (0, 0.66, 0.14, 0.24, 0.18) = 0.34$. Similarly, it would be calculated as $1.0, 0.78, 0.86$ and $0.82$ for alternatives $M_2, M_3, M_4$ and $M_5$ respectively.

(8) As per the above calculation, the alternative $M_2$ has the highest non-dominated degree than the other two alternatives. Therefore alternative $M_2$ has the highest rank as $r(M_2) = 1$.

(9) Deleting the alternative $M_2$ from the fuzzy strict preference relation matrix by deleting the corresponding row and column from the matrix, the resulting fuzzy strict preference relation matrix will be given as:
Which further gives a non-dominated degree of alternatives $M_1$, $M_3$, $M_4$, and $M_5$ as 0.76, 0.9, 1.0, and 0.96 respectively and therefore alternative $M_4$ will have a higher rank than other alternatives. Finally we have $(M_2 > M_1 > M_3 > M_5 > M_4)$ and alternative 2 is the best alternative compared to others alternatives. Hence, we can say that the second module is the most important, while the first module is the least important.

Now after prioritizing the modules we have to apply the steps for estimating the software testing effort.

**Step 1:** Prepare the Software Requirement Specification/Solution (SRS) for the software.

**Step 2:** Estimate the line of the source code (KLOC) for the above module, as well as the identified type of organization. On the basis of the project manager's ability, the number of lines of code estimated is 400. This project is a simple project, so as per the COCOMO guideline, assume that the type of organization is organic, and from Table 1 take the variable in respect to the organization for calculation purposes.

**Step 3:** In this step calculate the confidence value ($C$) for the project manager.

**Step 3.1** Suppose that a person who has 8 years of experience and has done 4 similar modules gives this estimation. Now calculate the membership grades for low, medium, and high for experience and familiarity using the membership functions given in Fig. 9 and Fig. 10 respectively [67].

Using the crisp input as the number of years of experience (8 year) of a project manager and a
The membership functions can be generated by fuzzifying it as shown in Fig. 9 as \( \{0.375, 0.75, 0.625\} \). Similarly, by using the crisp input as the number of similar projects done (4) by the project manager and a qualitative term of familiarity such as low, medium, and high, membership functions can be generated by fuzzifying it as shown in Figure 10 as \( \{0.25, 0.5, 0.75\} \). The results are shown in Table 11.

**Step 3.3** After generating the membership values of experience and familiarity, the fuzzy rule base is constructed to arrive at the fuzzy output as shown in the matrix below [67].

<table>
<thead>
<tr>
<th></th>
<th>LOW</th>
<th>MEDIUM</th>
<th>HIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPERIENCE</td>
<td>0.375</td>
<td>0.75</td>
<td>0.625</td>
</tr>
<tr>
<td>FAMILIARITY</td>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
</tr>
</tbody>
</table>

The upper-most row contains the membership values of the low, medium, and high for the number of years of experience. The left most columns comprise the membership values for the low, medium, and high of the number of similar projects done. The values of the cells of the matrix are computed using the max-min composition as follows:

The value of the cell \((i, j) = \min (i\text{th value of the upper-most row, } j\text{th value of the left-most column})\).

Now applying the max-min composition to the above confidence values, arrive at the values for low, medium, and high confidence as follows:

The value of \((1, 1)\) of the matrix above is \(0.25, 0.375\). The value of \((1, 2)\) of the matrix above is \(0.25, 0.75\). Similarly, compute the values for the other elements in the matrix.

Now, referring to Table 2, we have three values for the low confidence (LC), which are the...
values of the elements at cells (1,1), (1,2), and (2,1). Similarly, we have four values for the medium confidence (MC), which are the values of the elements at (2,2), (2,3), (3,1), and (1,3) and two values for the high confidence (HC), which are the values of the elements at the cells (3,2) and (3,3).

Now, take the maximum of the three values of the LC to get the final value for the low confidence (LC). Hence LC = max (0.25, 0.375, 0.25) = 0.375.

Similarly, take the maximum of the four values for calculating the value of the normal confidence (MC). Hence MC = max (0.25, 0.5, 0.5, 0.375) = 0.5.

Similarly, take the maximum of the two values for calculating the value of the high confidence (HC). Hence HC = max (0.75, 0.625) = 0.75.

Finally the values of the low, medium, and high confidence are (0.375, 0.5, 0.75) respectively.

**Step 3.4** After applying the fuzzy rule base and max-min composition, the membership values for low, medium, and high confidence are obtained as shown in Fig. 11 with corresponding region areas. As the output of the confidence values obtained by this process is fuzzy, it has to be defuzzified using center of gravity to arrive at the crisp output. For the low confidence, the areas to be considered are R₁, R₂, and R₃. For medium confidence the areas to be considered are regions R₂, R₃, R₄, and R₅, and finally for high confidence the areas to be considered are regions R₃, R₅, and R₆ as shown in Fig. 11.

For low confidence, the areas to be considered are R₁, R₂ and R₃. This is a trapezium as shown in Fig. 12. The shaded area represents the low confidence region. The area of the trapezium is 0.34218.

The center of gravity of the same area is calculated using the standard method. Observe that this area is composed of a rectangle and a triangle. The center of gravity of the triangle is \((0.875+0.875+0.95)/3=0.9\) and the center of gravity of the rectangle is \((0.7+0.875)/2=0.7875\). The center of gravity of this area is \((0.9+0.7875)/2=0.84375\), which lies on the line \(x = 0.84375\). Hence the center of gravity and the area of the low confidence are \((0.84375, 0.34218)\).

For the Medium confidence, the areas to be considered are regions R₂, R₃, R₄, and R₅ and shaded the area represents the Medium confidence, as shown in Fig. 13.

![Fig. 11. Defuzzification using the Center of Gravity Technique](image-url)
The center of gravity and the area of the medium confidence are \((0.85, 0.075)\) [67].

For the High confidence, the areas to be considered are regions \(R_3, R_5,\) and \(R_6\), which are shown in Fig. 14. The center of gravity and the area of the High confidence are \((0.9072, 0.12652)\).

The final confidence value \((C)\) is:

\[
C = \frac{0.84375 \times 0.34218 + 0.85 \times 0.075 + 0.9072 \times 0.12652}{0.34218 + 0.075 + 0.12652}
\]

\[= 0.8592\]

which is the center of gravity of all regions, as shown in Fig. 15.

Step 4: The modified number of the lines of code \((\text{KLOC}') = \text{KLOC} \times C\)
The total effort will be calculated with the help of the Intermediate COCOMO model in the above-mentioned equation 3.

\[ E_i = 3.2 \times (343.68)^{1.05} = 1472.69 \]

Out of 14 cost drivers, 12 cost drivers are normal. However, the value of two-cost drivers is highly capable and the very high experience in programming language are 0.82 and 1.14 respectively.

Hence, \( E_{AF} = 0.82 \times 1.14 \times 1.0 = 0.9348 \)

The total effort \( (E_t) \) will be \( E_T = E_i \times E_{AF} \)

\[ = 1472.69 \times 0.9348 \]
\[ = 1377 \text{ PM} \]
As before, we are demonstrating that research has shown that at least 50% of the total software cost is comprised for testing activities. Hence, we have around a 688 PM effort for testing activities. The duration for the completion of project is:

\[
D = c \times (E_1)^d \\
= 2.5 \times (688)^{0.35} \\
= 29.93 = 30 \text{ Months}
\]

Now we have total cost and time available to release the software, which is 688 PM and 30 months respectively. The project manager divides this total time and cost on the basis of his own experience among the modules. As we know in the above discussion modules the priority is given in the following way: \(M_2 > M_4 > M_5 > M_3 > M_1\).

Suppose the project manager divides the total cost into 250, 185, 125, 90, and 38 for \(M_2, M_4, M_5, M_3, \) and \(M_1\) respectively.

As the manager distributes this cost among the modules that have uncertainty in their value so we are again repeating the procedure to calculate the confidence value in the same manner as discussed above.

The confidence value for the project manager comes out to be 0.8457. So we will multiply this confidence value with all the divided cost for the particular module, which provides us with the optimal testing cost for each module.

Similarly, the total time is divided into 10, 8, 6, 4, and 2 for \(M_2, M_4, M_5, M_3\) and \(M_1\) respectively. These values are also uncertain, so we have to multiply with the confidence value of the project manager, which provides us with the optimal time for a particular module. The optimal cost and time for a particular module are listed below in Table 12.

The maximum value of cost and time(using percentage stringency) are listed in Table 13 below:

Now we have a total of 582 cost units and 25 time units(as per Table 12) to test the software. As we have also calculated the value of \(\delta_{\text{max}}\) for every module, which is listed in the proposed approach section, we calculate the highest priority category modules first. After testing this category, we obtained the actual time, cost, and percentage fault values as 9.5, 220, and 2% respectively. Since here \(\delta < \delta_{\text{max}}\) we moved to the next priority modules (e.g., the high priority modules), in spite of the faults exceeding the tolerance limit.

Now we received the time, cost, and percentage fault values as 9.0, 188, and 3% respectively. Here \(\delta = (9.0 - 6.76) / 6.76 \times 0.7 + (188 - 156.45) / 156.45 \times 0.3 = 0.29\) from (5), (6) and (11). Since \(\delta > \delta_{\text{max}}\) as the \(\delta_{\text{max}}\) value for the high priority modules is 0.20 from Table 4, now we would consider the percentage of faults obtained. Since this percentage fault is outside the fault toler-
ance (for the first iteration it is 0), we had to test this category again with an increased fault tolerance of 2% (if max_tolerance = 2% and min_improvement = 1% then from equation (7) new tolerance = min [2, 3-1]). The next time, we got the time, cost, and percentage faults as 7.0, 159, and 1% respectively. Again, here $\delta < \delta_{\text{max}}$, so we moved to a lesser priority (e.g., the medium priority modules).

This time we got the time, cost, and percentage faults as 7, 122 and 0% respectively. Since $\delta > \delta_{\text{max}}$ in this case, we had to look into percentage faults. Since these faults are within the tolerance, we did not look into this priority again. By that time, our resources (time and cost) were over. Therefore, we checked which priority modules had been tested enough. Since we could test the highest, high, and medium priorities, the software was ready to be launched.

We tested the proposed approach on various real time software and the results we found are very encouraging.

**Analysis for effort estimation only:**

Since the idea of this paper is new, unique, and novel the module prioritization, optimal testing effort, and the optimization of the software release policy were combined into a single platform (e.g., it is hybrid in nature). The approach is very new, this is why only a comparison is possible with respect to a software test effort estimation.

From Table 14 we can see that there is a variation between the estimated values and the actual efforts. The conventional method for testing the effort estimation also shows a variation between the real cost and the estimated cost, but the difference between the fuzzy based approach and actual effort is very little and hence can be reliably used for test effort estimation. From the graph shown in Fig. 16, which is given below, we have illustrated the comparisons of efforts. Using fuzzy process on software testing and the test drivers testing efforts can be calculated with high precision and the results can then obtained with less deviations.

### Table 13. Maximum time and cost

<table>
<thead>
<tr>
<th>MODULE</th>
<th>MAXIMUM TIME</th>
<th>MAXIMUM COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>M₂</td>
<td>10.14</td>
<td>264.27</td>
</tr>
<tr>
<td>M₄</td>
<td>7.97</td>
<td>190.86</td>
</tr>
<tr>
<td>M₅</td>
<td>5.75</td>
<td>121.56</td>
</tr>
<tr>
<td>M₃</td>
<td>3.65</td>
<td>83.72</td>
</tr>
<tr>
<td>M₁</td>
<td>1.75</td>
<td>33.73</td>
</tr>
</tbody>
</table>

### Table 14. Comparison of the test effort on few pieces of project data

<table>
<thead>
<tr>
<th>Project number</th>
<th>Effort in Person Months (PM)</th>
<th>Actual efforts</th>
<th>Estimated efforts based on COCOMO</th>
<th>Actual testing efforts</th>
<th>Conventional testing efforts estimation</th>
<th>Fuzzy based testing effort estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
<td>82.00</td>
<td>67.30</td>
<td>38.00</td>
<td>28.69</td>
<td>35.01</td>
</tr>
<tr>
<td>2.</td>
<td></td>
<td>309.80</td>
<td>280.09</td>
<td>145.41</td>
<td>123</td>
<td>141.07</td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td>580.06</td>
<td>516.8</td>
<td>256.00</td>
<td>229.88</td>
<td>253.22</td>
</tr>
<tr>
<td>4.</td>
<td></td>
<td>867.45</td>
<td>796.05</td>
<td>430.58</td>
<td>356.67</td>
<td>410.78</td>
</tr>
<tr>
<td>5.</td>
<td></td>
<td>1026.02</td>
<td>978.89</td>
<td>497.67</td>
<td>438.5</td>
<td>480.87</td>
</tr>
</tbody>
</table>
8. CONCLUSION

This paper illustrates how we can prioritize the software modules in order to test the important modules primarily with the help of the fuzzy multi-criteria approach and the fuzzy rule based system. The advantage of this approach is that it optimizes the testing costs and time for individual components.

By prioritizing, the software modules with the fuzzy multi-criteria approach made sure that critical errors do not pass undetected during testing. Thus, the exact time when the software can be released can be predicted using the above proposed model. After allowing a little deviation from the values and accepting low risk faults in the system, we can test the software effectively even if we have limited resources available to us. The fault tolerance concept assists us in testing the software within a given time and cost. At the end of the resources, we can also find out if we have tested enough or if there is a further need for testing the important modules. This facilitates us in being able to make a decision about whether the software product is ready to be released in the market or not.

In this paper, a simple and efficient method has been proposed to deal with the multi-criteria ranking of alternatives in the presence of uncertainty. It has been shown that the concept of fuzziness is worth considering in the ranking of modules in the cases where criteria values and criteria weights are vague linguistic terms. By using discrete fuzzy sets to describe vague linguistic terms, the handling of uncertainty in the proposed ranking method is computationally attractive.

REFERENCES


Optimal Software Release Using Time and Cost Benefits via Fuzzy Multi-Criteria and Fault Tolerance


Praveen Ranjan Srivastava
Praveen Ranjan Srivastava is working under the Software Engineering and Testing Research Group in the Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani at the Pilani Campus in Rajasthan, India. He is currently doing research in the area of software testing. His research areas are software testing, quality assurance, quality attributes ranking, testing effort, software release, test data generation, test effort estimation, agent oriented software testing, and metaheuristic approaches. He has published more than 75 research papers in various leading international journals and conferences in the area of software engineering and testing. He has been actively involved in reviewing various research papers that he has submitted in his field for different leading journals and various international and national level conferences. He has received various funds from different agencies like Microsoft, IBM, Google, DST, CSIR etc. He can be contacted at: praveenrsrivastava@gmail.com.