Fuzzy Measure Extraction for Software Quality Assessment as a Multi-Criteria Decision-Making Problem

Xiaojing Wang*, Martine Ceberio†, Shamsnaz Virani‡, Christian Del Hoyo§, Luis Gutierrez§
Computer Science Department
The University of Texas at El Paso, El Paso, Texas 79968-0518
* xwang@utep.edu, † mceberio@utep.edu, § cdelhoyo@miners.utep.edu, ¶ lcgutierrez@miners.utep.edu
‡ Engineering Division, Penn State Great Valley
School of Graduate Professional Studies, Malvern, PA 19355
Email: ssv1@psu.edu

***Abstract*—Being able to assess software quality is essential as software is ubiquitous in every aspect of our day-to-day lives. In this paper, we rely on existing research and metrics for defining software quality and propose a way to automatically assess software quality based on these metrics. In particular, we show that the software quality assessment problem can be viewed as a multi-criteria decision-making (MCDM) problem. In Multi-Criteria Decision Making (MCDM), decisions are based on several criteria that are usually conflicting and non-homogenously satisfied. Non-additive (fuzzy) measures along with the Choquet integral can be used: they model and aggregate the levels of satisfaction of these criteria by considering their relationships. However, in practice, it is difficult to identify such fuzzy measures. An automated process is necessary and can be used when sample data is available. We propose to automatically assess software by modeling experts' decision process: to do this we automatically extract the corresponding fuzzy measure from samples of the target experts' decision. We were able to improve previous approaches to automatic software quality assessment that used machine learning techniques.**

I. Introduction

Pfleeger summarizes software quality assessment in three ways [16]:

1) The quality of product;
2) The quality of process; and
3) The quality in the context of the business environment.

This paper focuses on software product quality assessment based on direct measurements of code properties. Software product quality is important because software is present in every aspect of normal day-to-day life. Software problems such as server breakdowns, software crashes, and data leaks have become common occurrences. Preexisting software problems do not stop software spending. Even though there are large amounts of money spent on developing software, the quality of this software still remains a mystery. The obvious questions therefore are: what is software quality and how is it measured?

In this paper, we rely on existing research and metrics for defining software quality, as presented in Section II. Once metrics adopted, it is relatively easy to decide, for each metrics, which piece of software is better than the others based on the measure / satisfaction level related to the corresponding metrics. However, even so, the problem of assessing software quality based on several metrics remains since it constitutes a complex decision involving several criteria (based on metrics such as length of code, number of classes, inter-dependence of classes).

Experts are used to assess software quality, but it is desired that this process could be automated so as to ensure the consistency of the assessments as well as its timeliness (e.g., on-the-fly assessment). Such a task has been tackled previously by Fuentes et al. [15] using a machine learning approach. In this article, we show that the software quality assessment problem can be viewed as a multi-criteria decision-making (MCDM) problem, and we propose to extract experts' decision process using Fuzzy Measure Extraction for MCDM.

In what follows, we start by reviewing the state of the art in Software Quality and Software Quality Assessment (Section II). We then provide information about theoretical topics that are central to our MCDM approach (Section III) and we show how SQA and MCDM are related, with details about our specific approach (Section IV). We tested our approach: our strategy along with our experimental results and analysis are provided in Section V. Finally, we conclude and draw directions for future work in Section VI.

II. Software Quality Assessment (SQA): Definitions, Current Status, and Existing Techniques

A. SQA: definitions and current status

Pressman defined software quality as "Conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally developed software" [19].

This definition addresses two aspects of software quality. The first is conformance to explicitly stated functional and performance requirements and explicitly documented development standards. These can be found in the requirements document developed between the customer and client. This requirement can be measured by counting the number of columns and comparing the data type to that stated in the requirements data. The performance requirements are also measurable. For example, the time a software product takes to complete a given task is a measure of performance. Functional and performance requirements and explicitly documented development standards are thus measurable.

The second aspect of software quality addressed in this definition is the implicit characteristics of all professionally developed software. These are characteristics such as reusability or flexibility. These implicit characteristics are also known as quality factors. Most software engineers, programmers and managers believe that software quality factors are best judged by experts. However, research has shown that expert judgments are often inconsistent and subjective. For example, experts such as Lorenz and Kidd considered multiple inheritances (software property) as a sign of bad quality code, but multiple inheritance is widely accepted in the programming community ([10]). Inconsistent expert opinion makes judgment of implicit characteristics such as reusability difficult. There is no quantitative measurement for quality factors such as reusability. This subjective aspect of software quality results in making software quality management difficult.

Solid information regarding the quality of the software product is difficult to estimate. There is currently no tool or process that uses quantitative information to calculate quality factors for software products. This lack of solid information creates problems with software project management.

Sommerville elaborates that software quality management is difficult because of the two different aspects of software quality [20]. The explicit aspect of software quality is to some extent measurable, but the implicit aspect of software quality is solely based on expert opinion which is frequently inconsistent. There is a need to directly measure the implicit aspect of software quality.

To understand the subjective estimates and the measurements, it is important to understand the software quality research.

Theoretical models define several quality factors such as reusability and flexibility but do not quantify them. Predictive models are models based on statistical techniques that predict characteristics such as fault density or fault proneness using direct measurements from code (product metrics). Predictive models predict the faults but have no theoretical evidence to support causality. One solution to remedy this problem is to add prediction capability to a theoretical model. Quality factors defined in a theoretical model are not measurable and hence cannot be predicted. One theoretical software quality model that defined the quality factors and linked all of them to measurable metrics in object-oriented software was Bansiya and Davis's model [2]. Bansiya and Davis' model is more complete than other theoretical software quality models, so it was chosen for analysis here. Bansiya and Davis's model defines the quality factors and links them to QMOOD set of metrics defined in Table I.

Although the Bansiya and Davis model provides a solid explanation for the design quality of object-oriented design, it presents some limitations. The major problems with the Bansiya and Davis model are their validation process, data used in the research and lack of prediction capability.

B. SQA: Existing Techniques

Machine learning is an important aspect in predicting software product quality because the more a classifier can learn, the better decisions it will make in building a predictive model [12]. Osbeck et al improved the prediction capability using J48, Part, and Random Forest, and the ensemble learning techniques examined were boosting, bagging, and stacking [15].

In our work, we show that SQA can be seen as a Multi-Criteria Decision-Making Problem problem and solved accordingly, and that, therefore, software product quality can be predicted by using fuzzy measures and Choquet integral.

III. Theoretical Background

In this section, we introduce important theoretical concepts necessary to later understand how SQA can be viewed as a Multi-Criteria Decision Making (MCDM) problem as well as why Fuzzy Measure Extraction (FME) needs to be carried out.

A. Multicriteria Decision Making (MCDM)

Multicriteria decision making (MCDM) is the making of decisions based on multiple attributes (or criteria). Usually, it consists of a set of consequences, a finite set of n criteria (or attributes), and a preference relation $\succeq$ on the set of consequences.

The set of consequences X is a multidimensional space, where $X \subseteq X_1 \times \cdots \times X_n$, and each X_i represents a set of values of attribute i, where $i \in \{1, \cdots, n\}$. For each criterion (or attribute), there is a preference relation $\succeq_i$ on each space X_i, such that for $x_i, y_i \in X_i$, $x_i \succeq_i y_i$ means that x_i is preferred to y_i. Then, the preference relation of a consequence for all criteria can be combined, using an aggregation operator, into a global value such that the final level of satisfaction of the consequences follows the partial preferences. A preference over the set of consequences X will be denoted as: $\forall x, y \in X$, $x \succeq y$ or $y \succeq x$.

TABLE I
QMOOD MODEL [2]

Quality Factor Definition	Bansiya and Davis's Model	Metric Definition (QMOOD metric)
Reusability Reflects the presence of object oriented design characteristics that allow a design to be reapplied to a new problem without significant effort.	-0.25 * Coupling + 0.25 * Cohesion + 0.5*Messaging +0.5* Design Size	**Design Size (DSC)** A measure of number of classes used in the design. **Hierarchies (NOH)** Hierarchies are used to represent different generalization-specialization aspects of the design. **Abstraction (ANA)** A measure of generalization-specialization aspect of design. **Encapsulation (DAM)** Defined as the enclosing of data and behavior within a single construct. **Coupling (DCC)** Defines the inter dependency of an object on other objects in a design. **Cohesion (CAM)** Accesses the relatedness of methods and attributes in a class. **Composition (MOA)** Measures the "part-of", "has", "consists-of", or "part-whole" relationships, which are aggregation relationships in object oriented design. **Inheritance (MFA)** A measure of the "is-a" relationship between classes. **Polymorphism (NOP)** It is a measure of services that are dynamically determined at run-time in an object. **Messaging (CIS)** A count of number of public methods those are available as services to other classes. **Complexity (NOM)** A measure of the degree of difficulty in understanding and comprehening the internal and external structure of classes and their relationships.
Flexibility Characteristics that allow the incorporation of changes in a design. The ability of a design to be adapted to provide functionality related capabilities.	0.25 * Encapsulation - 0.25 * Coupling + 0.5 * Composition + 0.5 * Polymorphism	
Understandability The properties of designs that enable it to be easily learned and comprehended. This directly relates to the complexity of design structure.	-0.33 * Abstraction + 0.33 * Encapsulation - 0.33 * Coupling + 0.33 * Cohesion -0.33 * Polymorphism - 0.33 * Complexity - 0.33 * Design Size	
Functionality The responsibility assigned to the classes of a design, which are made available by classes through their public interfaces.	0.12 * Cohesion + 0.22 * Polymorphism + 0.22 * Messaging + 0.22 * Design Size + 0.22 * Hierarchies	
Extendibility Refers to their presence and usage of properties in an existing design that allow for the incorporation of new requirements in the design.	0.5 * Abstraction - 0.5 * Coupling + 0.5 * Inheritance +0.5 * Polymorphism	
Effectiveness This refers to the designs ability to achieve the desired functionality and behavior using object oriented design concepts and techniques.	0.2 * Abstraction + 0.2 * Encapsulation + 0.2*Composition + 0.2 * Inheritance + 0.2 * Polymorphism	

The common aggregation operator being used is a weighted sum; i.e.,

$$u(x) = \sum_{i=1}^{n} w_i u_i(x_i),$$

where w_i is the weight of each criterion, representing the importance of each criterion, $\sum_{i=1}^{n} w_i = 1$, and $u_i(x_i)$ represents the level of satisfaction assigned to alternative $x_i \in X_i$. The best consequence ($x \in X$) is the one with the highest value of u. Although this approach is simple, easy to use, and low complexity, using an additive aggregation operator assumes that all criteria are independent, which, in practice, is seldom the case: often, decisions are based on several conflicting criteria and using linear additive aggregation will lead to possibly very counterintuitive decisions. Non-linear approaches also prove to lead to solutions that are not completely relevant. Therefore, using additive approach is often not good: based on our previous work [13], we choose to use non-additive approaches, i.e., fuzzy measures and integrals [3].

B. *Fuzzy measures and integrals*

Fuzzy measures are non-additive measures. They can be used to represent the degree of interaction of each subset of criteria [4]. In what follows, we consider a finite set of criteria $A = \{1, \cdots, n\}$.

Definition Let A be a finite set and $\mathcal{P}(A)$ the power set of A. A fuzzy measure (or a non-additive measure) defined on A is a set function $\mu : \mathcal{P} \to [0, 1]$ satisfying the following axioms:

(1) $\mu(\emptyset) = 0$
(2) $\mu(A) = 1$
(3) if X, $Y \subseteq A$ and $X \subseteq Y$, then $\mu(X) \leq \mu(Y)$

The fuzzy measures are used to show the importance of each subset and how each subset of criteria interacts with others. Fuzzy measures are expensive to determine: for a set defined over n criteria, 2^n values of a fuzzy measure are needed because there are 2^n subsets of A.

Two main integrals can be used to combine fuzzy measures: the Sugeno and the Choquet integrals. Although they are structurally similar, they are different in nature [8]: the Sugeno integral is based on non-linear operators and the Choquet integral is usually based on linear operators. The applications of Sugeno and Choquet integrals are also very different [14]: the Choquet integral is generally used in quantitative measurements, and a MCDM problem usually uses a Choquet integral as a representation function. In this article, we focus on the Choquet integral.

Definition Let μ be a fuzzy measure on A. The Choquet integral of a function $f : A \to R$ with respect to μ is defined by:

$$(C)\int_A f d\mu = \sum_{i=1}^{n}(f(\sigma(i)) - f(\sigma(i-1)))\mu(A_{(i)})$$

where σ is a permutation of the indices in order to have $f(\sigma(1)) \leq \cdots \leq f(\sigma(n))$, $A_{(i)} = \{\sigma(i), \dots, \sigma(n)\}$ and $f(\sigma(0)) = 0$, by convention.

C. Determining Fuzzy Measures

In MCDM, we would expect the decision maker to be more than likely to give the values of the fuzzy measure, but in most circumstances this is not the case. Attempts of making fuzzy measure identification easier for the decision makers have been made in [3], [22].

- In [3], the authors attempt to make this task easier by only requiring the decision maker to give an interval of importance for each interaction.
- In [22], the author suggests a diamond pair-wise comparison, where the decision maker only must identify the interaction of 2 criteria using a labeled diamond. From there, the algorithm evaluates the values of the numeric weights.
- In [22], the author discusses user specified weights mixed with an interaction index denoted λ or ξ. This algorithm is applied using an online aggregation application [21].

However, in most cases, the decision maker either does not understand the interactions well enough to provide a good value of each fuzzy measure, or does not have constant access to an expert who may give all values of the fuzzy measures. In addition, since there are $2^n - 2$ values of a fuzzy measure for a problem with n criteria expert identification: it would be too time consuming anyway to be practical [7]. This is where fuzzy measure extraction comes into play.

D. Fuzzy Measure Extraction (FME) and Optimization

For lack of an expert to provide all values of the fuzzy measure, we need seed data to give us an idea of the preferences / decision-making process: we use sample data. Extracting fuzzy measure is performed starting from such seed data.

Let's take a look at the following situation: Our MCDM problem involves n attributes, and m sample data. If we knew the fuzzy measure $\tilde{\mu}$, we would be able to compute preference values $\tilde{y}_j$ as $(C)\int_A f d\tilde{\mu} = \sum_{i=1}^{n}(f(\sigma(i)) - f(\sigma(i-1)))\tilde{\mu}(A_{(i)})$, where f is a utility function defined on X.

However, with the sample data, we only have access to the preference values of a subset of X. In order to have access to preference values of other alternatives in X, we need to determine μ, which is, the $2^n - 2$ values of the fuzzy measure. We are going to determine μ such that the corresponding computed Choquet integral is as close to the preference values of the sample data as possible. As a result, we aim at minimizing the following sum (and getting as close to 0 as possible) [9]:

$$e = \sum_{j=0}^{m}\left(\tilde{y}_j - \sum_{i=1}^{n}(f(\sigma(i,j)) - f(\sigma(i-1,j)))\mu(A_{(i)})\right)^2 \tag{1}$$

Moreover, the optimal solution must satisfy constraints: fuzzy measures must be monotonic and must always be between 0 and 1.

Therefore, fuzzy measure extraction is a constrained optimization problem, and the candidate solutions must be evaluated to make sure they fit the constraints.

Several optimization approaches have been proposed to extract fuzzy measures, including Gradient Descent methods [1], Genetic Algorithms [4], [27], and Neural Networks [24]. Besides these, many optimization techniques exist, such as for instance Harmony Search [6], Particle Swarm Optimization [18], Simulated Annealing [5].

However, the main drawbacks in these techniques are that the returned solution (found minimum of the objective function) might just be a local minimum, or even worse, a good value. There is no guarantee that it would be the global minimum at all.

In previous work [26], we proposed to use a tuned version of the Bees Algorithm [17] to extract fuzzy measures. The results show promise of the approach, and although the results were not guaranteed to be global (same drawback as pointed out of the other approaches), they were consistently better than best approaches before this one. In this work on SQA, we used the Bees Algorithm to extract fuzzy measures that best model experts' decision processes.

In what follows, we first motivate why MCDM is related to SQA and how Fuzzy Measure Extraction can help automate SQA. We then provide details about the specific optimization technique we use for Fuzzy Measure Extraction.

IV. Fuzzy Measures Extraction for Software Quality Assessment using the Bees Algorithm

As hinted earlier, Software Quality Assessment can be seen as a Multi-Criteria Decision-Making problem. The straightforward reason for that is the following: the general quality assessment (i.e., final decision of software) is based on a set of metrics (i.e., multiple criteria).

As a result, based on what we presented about fuzzy measures and Choquet integrals, if we can find an appropriate fuzzy measure μ, assessing the quality of software can be expressed as follows:

$$\boxed{\text{SQA}(\text{software } A) = \text{Choquet}(\mu, \text{metrics values}(A))}$$

Based on the above formula, we focus on determining μ, which can be viewed as a quantitative model for the expert's decision-making process. It is obtained from seed data, namely sample data of experts' decisions with respect to known pieces of software. As a result, in the above 3-body problem, two of the components are known: the expert's decision and the set of metrics values, and we aim at determining the matching μ.

In practice, we have access to a number of such expert(s)' decision values along with the corresponding sets of metrics values for different pieces of software A_i. As pointed out earlier, in Section III, determining μ consists of solving the following problem:

$$\boxed{\begin{array}{l}\min\ e = \sum_{A_i} (\text{SQA}(A_i) - \text{Choquet}(\mu, \text{metrics}(A_i)))^2 \\ \text{s.t. } \mu \text{ satisfies monotonicity constraints}\end{array}}$$

A. Our approach: the Bees Algorithm

The Bees optimization Algorithm, proposed in [17], uses bees' natural food foraging habits as a model for the exploration of the search space. The Bees Algorithm combines a local and "global" search that are both based on bees natural foraging habits. It roughly unwinds as follows:

1) First a number of "scout bees" are randomly sent out.
2) The patches of "nectar" (elements of the search space / candidate values for the fuzzy measure) are then ranked according to evaluated fitness. More bees are dispatched to look in neighboring areas of good patches of "nectar".
3) At each iteration, a number of "scout bees" are kept to explore other areas in hope of better patches of "nectar": this keeps the algorithm searching "globally".
4) When a new patch is found, its fitness is evaluated and compared against previously explored patches and a proportional number of "bees" is sent to it. The dispatched "bees" perform a local search by moving in a random direction from the patch of "nectar".
5) If a local search "bee" finds a better patch of "nectar", the location from where it was dispatched is moved to the new location [17]. The Bees Algorithm performs local search by sending an amount of "bees" that is proportional to the patch's fitness.

It is believed that the best ranked "patch", when a stopping criterion is met, is the optimal solution, although there is no theoretical guarantee for it. In [17], the Bees algorithm, in most cases, was faster than a number of other algorithms (including genetic algorithms, simplex method, stochastic simulated annealing), and returned a solution within .1% of the perfect solution every time run (in particular up to 207 times faster than the Genetic Algorithm on the benchmarks used). On their test cases, the Bees algorithm was also always able to reach the global optimum and ignore local optima.

V. Experiments

A. What do we want to show and how?

Through our experiments, we aim at quantifying the performance of our approach in automatically predict software quality. As hinted before, the algorithm we use to extract fuzzy measures is the Bees algorithm, as it showed promise in previous work [26].

The metrics we use are Quality Model for Object-Oriented Design (QMOOD) metrics, as defined in [2] and used by Virani et al [23]. In this work, we focused on the metrics and the quality factors related to QMOOD model, as in [2]. Definitions for each of the quality factors and metrics used in QMOOD model are provided in Table I.

The data at our disposal to run our tests is from 31 software packages and 2330 samples. The rating options are bad, poor, good, fair, and excellent. Each package was rated individually by a group of experts for each of the metrics and for the total quality. Note that the data is the same data set as used in [15], in which the authors proposed a machine learning techniques to predict SQA.

Using the above-described data, our goal is to show that: (1) our approach allows to accurately recreate decisions (data) used to determine the reasoning process (fuzzy measure); and that (2) it also works to predict decisions (data) that were not included in determining the reasoning process (fuzzy measure) but that were available to us.

We were also interested in addressing the following questions:

1) Are experts consistent when they assess software? Focusing on one expert at a time, we wanted to know how well we would be able to reconstruct each expert's decision-making process. Again, this was highly dependent on each expert's original ability to make consistent decisions.
2) Do experts agree on software assessment? or are we able to create a "super expert"'s decision process using our approach? Looking at one software package at a time, across experts, we wanted to get a sense of common decision-making process among the experts. We wanted to see how closely we would be able to reconstruct the decisions, which is very tightly coupled with a sense of agreement across the experts.

B. Experimental Results

1) Reconstructing the data: **Using all data and all experts.** The 2330 samples we have at our disposal include 31 software packages and are evaluated by 78 experts. We first extracted a fuzzy measure from all samples that fits them all the best. In Table II, we report the accuracy reached when determining the target fuzzy measure; i.e., the sum of the differences between the reconstructed data and the original data for all the data.In particular, we provide this information for different iteration counts of our Bees algorithm: we can observe that the quality is stabilized already after 100 iterations.

TABLE II
OPTIMAL FUZZY MEASURE FOR ALL SAMPLES

Iterations	**e**
100	0.07675
1000	0.07671
10000	0.07670

Although the results are stable after 100 iterations, it is about 10 times worse than the results of the toy examples in [26]. This is because 78 experts were involved the SQA evaluation and the decisions are not consistent among different experts. Even for the same expert, the decisions may not be consistent. This is the object of our next experiments.

Focusing on one expert at a time. We ran experiments to see whether experts were consistent in their decision process; that is, for different software packages, whether the values of the metrics are similar, then each expert's decision of the quality should be very close. Table III shows the results for each expert.

TABLE III
TESTING RESULTS FOR EACH EXPERT

e	**# of experts**
< 0.01	25
$[0.01, 0.035)$	25
$[0.035, 0.075)$	16
≥ 0.075	12
min	1.68E-05
max	0.4398

We find that most individual expert's decision processes (64%) are consistent, while some experts' decision shows big discrepancies from one sample or package assessment to another.

For example, one expert evaluated 5 samples for 1 software package, and the result is 0.4398. We check the sample data and find no matter how difference the value of the metrics are, the expert always makes the same decision, which means the expert's opinion on the SQA is not consistent.

Focusing on one software package at a time. We tested to see whether the evaluation from different experts for the same software package is consistent, that is, for the same software package, since evaluated by different experts, the discrepancy of the experts' decision should be larger than the previous results. Figure 1 shows the results for each software package.

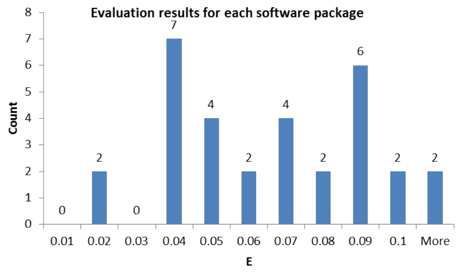

Fig. 1. Evaluation Results for each Software package

Since each software package was evaluated by at least 6 experts, and each expert has a different point of view for each Software package, it is reasonable that the results for each Software package is worse (less consistent) that the results for each expert.

2) Predicting the data: As mentioned earlier, the goal of this research is to test how well fuzzy measure extraction can help predict software quality. We test it by first extracting fuzzy measure from part of the sample data (randomly selected) and then using the extracted fuzzy measure to predict the software quality for the rest samples to see if the predicted software quality matches the original software quality. Testing results are in table IV.

TABLE IV
TESTING RESULTS FOR RANDOMLY SELECTED PARTIAL SAMPLES (ITERATION = 1000)

Sample numbers	**e**
500	0.07904
1000	0.07862
2330	0.07671

C. Analysis of our results

What we have shown in above is how well the Bees algorithm used to extract fuzzy measure. Now we want to test how well the extracted fuzzy measure can help predict Software quality.

We compared the evaluations we obtained to the original experts' decision and assessed the accuracy of our approach using 3 different evaluation processes.

1) Eval1: We computed the average assessment over the known experts' decisions and considered this average the target evaluation. Anything else would just be considered wrong.
2) Eval2: We used the same average as before but allowed for some flexibility by accepting any evaluation within σ (standard deviation) of the target average. This evaluation accounted for the uncertainty of the experts' decisions.
3) Eval3: We mapped the number of experts decisions from 0% to 100% based on the maximum number of votes for one rating (which would be the one to

receive 100% accuracy) and interpolated all other possible ratings (numerical values in between posted ratings) to determine their accuracy.

The overall evaluation results for all quality factors are in table V, and we also list the results using machine learning approach to compare with.

TABLE V
Accuracy using Hybrid2

Quality Factor	Machine learning approach [15]	Eval 1 (Average)	Eval 2 ($\mu \pm \sigma$)	Eval 3
Reusability	69.91%	41.67%	79.55%	72.34%
Flexibility	75.39%	47.89%	75.71%	66.34%
Extendibility	70.37%	42.80%	77.43%	70.91%
Functionality	78.54%	27.39%	42.74%	59.50%

VI. Conclusion and Future Work

In this article, we proposed an multi-criteria decision-making-based approach to software quality assessment. By viewing this assessment problem as a multi-criteria decision making (MCDM) problem, we were able to use fuzzy measures to model software expert's decision making process and help predict/evaluate software quality. We were able to show that our approach (specifically fuzzy measure extraction based on experts' decisions data) helps to predict/evaluate software quality with consistently over 60% accuracy, which is as accurate as previous approaches to SQA conducted using machine learning techniques.

Our current approach can be further improved as follows. Although the Bees algorithm we implemented provides good and reasonably fast results for fuzzy measure extraction, we believe we can improve it by combining it with another solver. Moreover, the expert's opinions (data used to extract a decision process model) usually are linguistic values; for example, Excellent, Good, Fair, Poor, and Bad. These words have different meanings to different experts, and therefore, expert's linguistic ratings are uncertain and their interpretation should not be uniform. In particular, using a continuous utility function that assigns a precise value to each of these evaluation results would result in losing accuracy. In order to better fit the expert's onions, in the future, we will use an interval end-points approach [11] and may use non-linear utility functions. Finally, we need to study the amount of data that is necessary to extract meaningful and accurate decision process models: for instance, what is the impact of a reduced sample data set on the quality of the decisions? What is the critical number of data w.r.t. the number of criteria for instance?

References

[1] S. H. Alavi, J. Jassbi, P. J. A. Serra, and R. A. Ribeiro. Defining fuzzy measures: A comparative study with genetic and gradient descent algorithms. In *Intelligent Engineering Systems and Computational Cybernetics*, pages 427–437. Springer Netherlands, 2009.

[2] J. Bansiya and C. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering.*

[3] M. Ceberio and F. Modave. An interval-valued, 2-additive choquet integral for multi-criteria decision making. In *Proceedings of the 10th Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'04)*, Perugia, Italy, July 2004.

[4] E. F. Combarro and P. Miranda. Identification of fuzzy measures from sample data with genetic algorithms. *Computers & Operations Research*, 33(10):3046–3066, 2006.

[5] L. Davis. *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, 1987.

[6] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60–68, 2001.

[7] M. Grabisch. A new algorithm for identifying fuzzy measures and its application to pattern recognition. In *Proceedings of 4th IEEE International Conference on Fuzzy Systems*, Yokohama, Japan, March 1995.

[8] M. Grabisch. The application of fuzzy integrals in multicriteria decision making. *European Journal Of Operational Research*, 89(3):445–456, 1996.

[9] M. Grabisch, H.T. Nguyen, and E. A. Walker. *Fundamentals of uncertainty calculi with applications to fuzzy inference.* Kluwer Academic Publishers, Norwell, MA, 1994.

[10] M. Lorenz and J. Kidd. *Object-oriented software metrics: a practical guide.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[11] J. Mendel. Computing with words and its relationships with fuzzistics. *Information Sciences*, 177:988–1006, 2007.

[12] T. M. Mitchell. *Machine Learning.* McGraw-Hill, New York, first edition, 1997.

[13] F. Modave, M. Ceberio, and V. Kreinovich. Choquet integrals and owa criteria as a natural (and optimal) next step after linear aggregation: A new general justification. In *Proceedings of MICAI'2008*, pages 741–753, 2008.

[14] F. Modave and P. W. Eklund. A measurement theory perspective for mcdm. In *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, pages 1068–1071, Melbourne, Australia, 2001.

[15] J. Osbeck, S. Virani, O. Fuentes, and P. Roden. Investigation of automatic prediction of software quality. In *North American Fuzzy Information Processing Society (NAFIPS'2011)*, El Paso, TX, March 2011.

[16] Pfleeger. *Software Engineering Theory and Practice.* Prentice Hall, 2001.

[17] D. Pham, A. Ghanbarzadeha, E. Koc, S.Otri, S. Rahim, and M. Zaidi. The bees algorithm-a novel tool for complex optimization problems. In *Proceedings of 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, pages 454–459, 2006.

[18] R. Poli, J. Kennedy, and T. Blackwell. Particle swam optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[19] R. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 2005.

[20] I. Sommerville. *Software Engineering.* Addison Wesley Publishing Company, Harlow, England, 2004.

[21] E. Takahagi. Usage: Fuzzy measure-choquet integral calculation system (λ fuzzy measure and sensitivity analysis). http://www.isc.senshu-u.ac.jp/~thc0456/Efuzzyweb/mant2/mant2.html.

[22] E. Takahagi. A fuzzy measure identification method by diamond pairwise comparisons and ϕ_s transformation. *Fuzzy Optimization and Decision Making*, 7(3):219–232, 2008.

[23] S. S. Virani, S. Messimer, P. Roden, and L. Etzkorn. Software quality management tool for engineering managers. In *Proceedings of the Industrial Engineering Research Conference*, pages 1401–1406, Vancouver, Canada, 2008.

[24] J. Wang and Z. Wang. Using neural networks to determine sugeno measures by statistics. *Neural Networks*, 10(1):183–195, 1997.

[25] X. Wang, J. Cummins, and M. Ceberio. The bees algorithm to extract fuzzy measures for sample data. In *North American Fuzzy Information Processing Society (NAFIPS'2011)*, El Paso, TX, March 2011.

[26] Z. Wang, K. Leung, and J. Wang. A genetic algorithm for determing nonadditive set functions in information fusion. *Fuzzy Sets and Systems - Special issue on fuzzy measures and integrals*, 102(3):463–469, 1999.