Journal of Uncertain Systems
Vol.7, No.3, pp.219-237, 2013
Online at: www.jus.org.uk

A Hybrid Algorithm to Extract Fuzzy Measures for Software Quality Assessment

Xiaojing Wang[1,*], Martine Ceberio[1], Shamsnaz Virani[2]
Angel Garcia[1], Jeremy Cummins[3]

[1] *Computer Science Department, The University of Texas at El Paso, El Paso, Texas 79968-0518*

[2] *Engineering Division, Penn State Great Valley, School of Graduate Professional Studies, Malvern, PA 19355*

[3] *Department of Computer Science and Information Systems*
Youngstown State University, Youngstown, Ohio 44555

Received 2 April 2012; Revised 20 September 2012

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 build upon 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 problem of assessing the quality of software 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. We use non-additive (fuzzy) measures combined with the Choquet integral to solve MCDM problems for they allow to model and aggregate the levels of satisfaction of the several criteria of the problem at hand by considering their relationships. However, in practice, it is difficult to identify such fuzzy measures. An automated process is then necessary and can be used when sample data is available. We propose to automatically assess software by modeling experts' decision process and extracting the fuzzy measure corresponding to their reasoning process: to do this, we use samples of the target experts' decision as seed data to automatically extract the fuzzy measure corresponding to the experts' decision process. In particular, we propose an algorithm to efficiently extract fuzzy measures that is a combination of the Bees algorithm and an interval constraint solver. Our experimental results show that we are able to improve some of the results obtained through previous approaches to automatic software quality assessment that used traditional machine learning techniques.

Keywords: fuzzy measure, software quality assessment, Choquet integral

1 Introduction

Software product quality is crucial as 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. Pre-existing software problems do not stop software spending. Even though there are large amounts of money spent on developing software, the quality of software still remains a mystery. It is only legitimate to ask: What is software quality? And how is it measured?

Pfleeger summarizes software quality assessment in three ways [25]:

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. We rely on existing research and metrics for defining software quality, as presented in Section 2.

*Corresponding author. Email: xwang@utep.edu (X. Wang).

Software quality assessment (SQA) can be seen as a Multi-Criteria Decision-Making (MCDM) problem, that is, the general quality assessment (i.e., final decision of software) is based on a set of metrics (i.e., multiple criteria). This kind of complex decision process is called Multi-Criteria Decision Making (MCDM).

Multi-Criteria Decision Making in Brief. In general, when a decision based on several criteria is not critical, we mentally "average/sort" the satisfaction levels of each of the criteria we have to consider. Usually, the satisfaction level of each criterion is subject to the decision maker's personal preference and some criteria might "weigh" more than others. The overall score of an alternative is then calculated by aggregating the values of satisfaction with weights on each criterion, where each weight indicates how important the corresponding criterion is when compared to the entire set of criteria. This approach is called a weighted-sum approach, and the weights assigned to different sets of criteria in this approach form what is called an "additive measure". However, additive aggregations assume that criteria are independent, which is seldom the case [4]. For instance, quality and price are usually two conflicting and correlated criteria: although high-quality low-price items are sought, high quality usually goes with high price. On the other hand, non-linear approaches also prove to lead to solutions that are not completely relevant [20]. This is why we then turn to non-additive (fuzzy) measures.

For example, if two criteria are strongly dependent, it means that both criteria express, in effect, the same attribute. As a result, when considering the set consisting of these two criteria, we should assign to this set the same weight as to each of these criteria and not double the weight as we would in a weighted sum approach. In general, weights associated to different sets should differ from the sum of the weights associated to individual criteria. In mathematics, non-additive functions that assign values to sets are known as non-additive (fuzzy) measures. It is therefore reasonable to describe the dependence between different criteria by using an appropriate non-additive (**fuzzy**) **measure**.

We then combine values corresponding to different criteria by using a Choquet integral over a fuzzy measure and obtain decisions that take into account the dependence of criteria. However, to make this happen, fuzzy measures need to be determined: they can either be identified by a decision maker/expert or by an automated system that extracts them from sample data. Since human expertise might not always be available and getting accurate fuzzy values (even from an expert) might be tedious [17], we focus here on **extracting fuzzy measures from sample data**.

The sample SQA data that we use is a set of overall preference values (i.e., preferences that would otherwise be obtained after combining criteria satisfaction levels and an appropriate fuzzy measure through Choquet integral) associated with given inputs. Fuzzy measure extraction seeks to determine the fuzzy measure such that the calculated Choquet integral with respect to the fuzzy measure best models the expert's decision, i.e., the corresponding sample data value from expert. This problem is therefore tackled as an optimization problem. Several optimization approaches have been used to extract fuzzy measures from sample data, such as gradient descent algorithms [9], genetic algorithms [5, 34, 36], and the Bees algorithm [35]. More specifically, fuzzy measure extraction constitutes a constrained optimization problem since the optimal solution must also satisfy the monotonicity constraints inherent to the fuzzy measure we aim at determining.

In this article, we propose to use an adaptive hybrid algorithm that combines the Bees algorithm and an interval constraint solver to extract fuzzy measures from SQA. The Bees algorithm has been successfully used in many optimization problems but requires a significant amount of tuning to attain reasonable performance. We adjusted the Bees algorithm to the needs of fuzzy measure extraction problems: in particular, in [35], we showed that using the Bees algorithm to extract fuzzy measures from sample data yields promising performance, with results of similar or better accuracy than those obtained using existing approaches. In this article, we use RealPaver – an interval constraint solver – to shrink the search space, improve the overall performance of the Bees algorithm, and finally provide guarantees that the search has focused on relevant parts of the search space and discarded irrelevant ones.

The article is organized as follows: Section 2 provides background and recalls necessary definitions on SQA, MCDM, fuzzy measures, fuzzy integrals, and fuzzy measure extractions. Section 3 introduces Fuzzy Measure Extraction (FME) as an optimization problem and recalls existing approaches to FME. We show how SQA and MCDM are related in Section 4 and we provide details about our proposed approach in Section 5, describe our experimental strategy, report, and analyze the results in Sections 6 and 7. Finally, we draw conclusions and propose directions for future work in Section 8.

2 Background

2.1 Software Quality Assessment

Software quality is defined as "Conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally developed software" [28]. 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 [16]. Inconsistent expert opinion makes judgment of implicit characteristics such as reusability difficult. Process improvement strategies such as the CMMI have shown to reduce software quality costs [24]. CMMI provides software management with five levels of process capability and maturity. To achieve each level of maturity, management needs to fulfill a set of goals. In fact, CMMI level 4 requires software measurement to be predictive of quality. CMMI level 4 is easily applied for the explicitly stated functional and performance requirements, but it is difficult to apply for implicit characteristics or quality factors. There is no quantitative measurement for quality factors such as reusability. This subjective aspect of software quality results in making software quality management difficult.

Although this process is difficult, many companies depend upon software to acquire a competitive edge. CIOs and IT leaders often assume the roles of change and risk managers in Fortune 500 companies [37]. CIOs and IT leaders are expected to facilitate business problem solving as well as provide technical solutions. These leaders are required to meet market expectations and performance goals through the use of the latest software technologies. The estimation of cost, schedule and quality goals is particularly difficult with relation to software projects. In the widely recognized software engineering textbook [28]. The author says "A software project manager is confronted with a dilemma at the very beginning of a software engineering project. Quantitative estimates and an organized plan are required, but solid information is unavailable" [28].

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 gives three reasons why software project management activities are difficult when compared to other engineering project management activities [29]. The three reasons are

1. "*The product is intangible*" [29]: Other than counting lines of code, there is no way to track the progress of the project because software product cannot be seen or touched.

2. "*There are no standard software processes*" [29]: Since software engineering is a relatively new engineering practice, there is no accepted industry-wide standard software process.

3. "*Large software projects are usually different in some way from the previous projects*" [29]: Software technologies are changing continuously because of this continuous change of technology, and experience transfer from one project to another project is non-existent.

To understand the subjective estimates and the measurements, it is important to understand 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 1 and Table 2.

Table 1: QMOOD model [2]

Quality Factor Definition	Bansiya and Davis's Model
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
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

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 [22].

According to Moody [22], the major problem with the Bansiya and Davis model is the validation process. Validation of the model was performed using a case study and a lab experimental approach which involves gathering expert opinion of software quality. Elicitation of expert opinion is a very well defined process that includes calculation of reliability and inter-rater agreement [15, 3]. Bansiya and Davis [2] appear to have ignored the guidelines on elicitation of expert opinion in the validation process of the model. They did not develop a common set of heuristics or a questionnaire to collect the expert opinion. There was no calculation of agreement among experts on their opinion of a package or a quality factor.

Table 2: QMOOD metric definitions

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.

In addition, a major issue with the model is that the packages used in the study were not very different from each other. The COOL packages used by Bansiya and Davis [2] were developed by different teams, but had the same set of requirements. The methodology developed to calculate quality factors based on the QMOOD set of metrics was restricted to the COOL packages. This allowed the researchers to normalize the data but restricted the prediction capability of the model. Thus, the model could not be used to calculate or compare quality factors for software packages with different sets of requirements. Kitchenham et al. [15] provide a framework of software measurement validation. Their opinion on prediction models in software is that they are based only on empirical evidence and not on theoretical evidence. This restricts the models from developing a causal relationship to quality factors. Because the Bansiya and Davis model was developed as a theoretical model, it implies the causal relationship between design characteristics and quality factors. The validation of the model with data, however, lacked rigor. Etzkorn et al. [7] developed a model to predict reusability of object-oriented code based on theoretical evidence, software metrics and expert opinion. Etzkorn et al. established a relationship between software metrics and reusability; they thus provide a basic methodology to use expert opinion in evaluation of software quality factors. Virani et al. [32] added prediction capability to the Bansiya and Davis model using C4.5 algorithm – a standard machine learning decision tree.

Traditional Machine learning has established itself as an important technique for predicting software product quality because the more a classifier can learn, the better decisions it will make in building a predictive model [19]. In [23], Osbeck et al. used an existing machine learning tool, WEKA [14], to generate a model predicting the difference between five rating classes of software quality. They improved the predication capability by using different ensemble learning techniques, such as boosting, bagging and stacking, combined with different classifiers, including J48, Part, and Random Forest.

In this article, 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.

2.2 Multicriteria Decision Making (MCDM)

Multicriteria decision making (MCDM) is the making of decisions based on multiple attributes (or criteria). Usually, decisions are to be made over a set of alternatives (or consequences), based on a finite set of n criteria (corresponding to the attributes that matter in the items we have to decide on; e.g., cost, size). The objective of MCDM is to make sense of a preference relation $\succeq$ on the set of alternatives / consequences (usually implicit) in order to elicit the best alternative. In what follows, we go over essential notions about MCDM.

Each criterion i is associated to a set X_i of possible values of the corresponding attribute. There exists a preference relation $\succeq_i$ on each set X_i, such that for all $x_i, y_i \in X_i$, $x_i \succeq_i y_i$ means that x_i is preferred to y_i. We assume that for each criterion i, there exists a real-valued function $u_i : X_i \to R$ such that for all x_i, $y_i \in X_i$:

$$x_i \succeq_i y_i \iff u_i(x_i) \geq u_i(y_i)$$

Function u_i is called the i-th monodimensional utility function [21] and scales the values of all attributes onto a common (real) scale.

In turn, the set of alternatives (or consequences) X can be represented as a multidimensional space, where $X \subseteq X_1 \times \cdots \times X_n$. Let us note that X is likely to be only a subset of $X_1 \times \cdots \times X_n$ as this Cartesian product is likely to define some alternatives that are not feasible; e.g., if cost and quality are two attributes of the alternatives we have to decide upon, and their respective sets of possible values are $\{\texttt{low}, \texttt{average}, \texttt{high}\}$ and $\{\texttt{low}, \texttt{average}, \texttt{high}\}$, then there might not exist any alternative $x \in X$ such that $x = (\texttt{low}, \texttt{high})$. Let us also note here that each alternative $x \in X$ can then be described by the tuple of the values it takes in each of the criteria; in particular, $x \in X$ is described by $x = (x_1, \ldots, x_n)$ where $\forall i \in \{1, \ldots, n\}$, $x_i \in X_i$.

Preference relations on the set of alternatives are not as straightforward to define as they are on the sets of values of the attributes. Preferences $\succeq$ over X are usually obtained from combining the preferences at the level of the attributes; in other words, $\succeq$ is defined from the $\succeq_i$'s, $i \in \{1, \ldots, n\}$. In particular, the utility functions are aggregated into a "global" preference (or utility) value $u(x)$. The best alternative (or consequence) is the $x \in X$ that yields the highest value of u. The way utility functions are combined is referred to as the aggregation operator. The common aggregation operator being used is the weighted sum; i.e.,

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

where:

- the w_i's, for $i \in \{1, \ldots, n\}$, are the weights associated to each criterion, representing the importance of each criterion and satifying the following constraint: $\sum_{i=1}^{n} w_i = 1$, and
- $u_i(x_i)$ represents the level of satisfaction assigned to value $x_i \in X_i$ of the i^{th} attribute.

Although this approach is simple, easy to use, and has 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 [20], we choose to use non-additive approaches, i.e., fuzzy measures and integrals [4].

2.3 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 [5]. In what follows, we consider a finite set of criteria $A = \{1, \ldots, n\}$.

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 $\mu : \mathcal{P}(A) \to [0, 1]$ satisfying the following conditions:*

(1) $\mu(\emptyset) = 0$,

(2) $\mu(A) = 1$,

(3) if X, $Y \subseteq A$ and $X \subseteq Y$, then $\mu(X) \leq \mu(Y)$.

In Multi-Criteria Decision Making, fuzzy measures are used to show the importance of each subset of criteria w.r.t. others and to represent how each subset of criteria interacts with others. Fuzzy measures are expensive to determine: for a set of n criteria, the corresponding fuzzy measure has 2^n values (as there are 2^n subsets of A).

In order to show how different fuzzy measures are from additive approaches, let us assume that C_1 and C_2 are disjoint subsets of criteria. Then, in general, the measure value of $(C_1 \cup C_2)$ is not the same as the sum of the individual measure values associated to C_1 and C_2, that is:

$$\mu(C_1 \cup C_2) \neq \mu(C_1) + \mu(C_2).$$

In particular, in the cases we consider, the total measure value $\mu(A) = 1$ associated to the set A of all the criteria is, in general, smaller than the sum of the measure values

$$\mu(\{1\}) + \mu(\{2\}) + \cdots + \mu(\{n\})$$

associated to all criteria. This is possible with the use of fuzzy measures but not with any additive approach.

Let us consider x, an element of the set of alternatives X, defined by $(x_1, \ldots, x_n)$. Now, just as we needed to do so earlier, we still need to aggregate the scores $u_i(x_i)$ by using an appropriate fuzzy measure. We can no longer add these scores with the measure values $\mu(\{i\})$ corresponding to individual criteria, indicating the weight of each individual criteria, because the sum of these values is, in general, larger than 1, so we need to decrease the measure values assigned to different criteria.

One possible approach is to use the most optimistic combination; after transforming the utility functions in such a way that the ranges of all u_i's are the same, we sort the values x_i in increasing order of their utility value $u_i(x_i)$: $x_1 \leq x_2 \leq \cdots \leq x_n$, and then assign as large a measure value as possible to x_i corresponding to larger (more optimistic) values $u_i(x_i)$. We start with the value x_n with the largest utility and we associate it with the full measure value $\mu(\{n\})$. For the next best value x_{n-1}, we select the largest measure value for which the group consisting of the two best criteria is assigned its largest possible measure value $\mu(\{n-1, n\})$. Since we already know the measure value $\mu(\{n\})$ assigned to the n^{th} criterion, we can thus find the measure value to be assigned to the next best criterion as the difference $\mu(\{n-1, n\}) - \mu(\{n\})$. Similarly, we find the measure value for the $(n-2)^{\text{nd}}$ criterion from the condition that the three best criteria gets assigned the largest possible measure value $m(\{n-2, n-1, n\})$. This means that we assign, to this criterion, the measure value equal to the difference $\mu(\{n-2, n-1, n\}) - \mu(\{n-1, n\})$. In general, to the i^{th} criterion, we assign the measure value $\mu(\{i, i+1, \ldots, n\}) - \mu(\{i, \ldots, n\})$. With these measure values, we get the following combination:

$$\sum_{i=1}^{n} u_i(x_i) \cdot (\mu(\{i, i+1, \ldots, n\}) - \mu(\{i+1, \ldots, n\})). \tag{1}$$

This combination is known as the Choquet integral. Choquet integrals with respect to fuzzy measures are actively used in Multi-Criteria Decision Making [11].

In practice, two main integrals can be calculated over fuzzy measures: the Sugeno and the Choquet integrals. Although they are structurally similar, they are different in nature [10]: 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 [21]: 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, which we formally define hereafter.

Definition *Let μ be a fuzzy measure on a set 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 $f(\sigma(1)) \leq \cdots \leq f(\sigma(n))$, $A_{(i)} = \{\sigma(i), \ldots, \sigma(n)\}$ and $f(\sigma(0)) = 0$, by convention.

Note: the above formula is equivalent to that of Equation 1 where $f(\sigma(i)) = u_{\sigma(i)}$.

2.4 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 to making fuzzy measure identification easier for the decision makers have been made in [4, 31].

- In [4], the authors attempt to make this task easier by only requiring the decision maker to give an interval of importance for each interaction.
- In [31], 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 [31], the author discusses user specified weights mixed with an interaction index denoted λ or ξ. This algorithm is applied using an online aggregation application [30].

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 values of a fuzzy measure for a problem with n criteria expert identification: it would be too time consuming anyway to be practical [9]. This is where fuzzy measure extraction comes into play.

In what follows, we focus on the problem of extracting fuzzy measures from sample data.

3 Fuzzy Measure Extraction (FME) and Optimization

3.1 FME and Optimization

For lack of an expert to provide all values of the fuzzy measure, we need to be able to determine fuzzy measures automatically. One way to do so is to use seed decision data to inform us of the preferences / decision-making process of the target expert(s): we use sample data from prior decisions.

Let us take a look at the following situation: We consider an MCDM problem over a set X of alternatives, each with n attributes, and we assume that we are given a sample data set of size m.

If we knew the fuzzy measure $\tilde{\mu}$ corresponding to our problem, we would be able to compute preference values $\tilde{y}_j$ as for alternative j in the set of our alternatives X:

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

where $f(i,j)$ corresponds to the utility value of alternative j for criterion i.

However, with the sample data we have access to, we only have access to the decision 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, 2^n values of the fuzzy measure.

Optimization. We determine μ such that the corresponding computed Choquet integral $\tilde{y}_j$ is as close to the decision values y_j of the sample data as possible. We do so by considering a least square approach and, as a result, we aim at minimizing the following sum (and getting as close to 0 as possible) [12]:

$$e = \sum_{j=0}^{m}(y_j - \tilde{y}_j)^2 \tag{2}$$

which can be rewritten as:

$$e = \sum_{j=0}^{m}\left(y_j - \sum_{i=1}^{n}(f(\sigma(i,j)) - f(\sigma(i-1,j)))\mu(A_{(i)})\right)^2. \tag{3}$$

When $e = 0$, the identified fuzzy measure μ is the exact solution of the problem: this is the ideal case. In most cases, we put up with "only" reaching an approximate optimal solution, that is, with $e \neq 0$ but close to

0. The reason for such a weaker expectation is that the sample data might not be fully consistent with one fuzzy measure.

In any case, extracting a fuzzy measure is cast down to solving an optimization problem. This optimization problem is actually a constrained optimization problem since the values of μ that the minimization process seeks must satisfy the monotonicity properties that characterize a fuzzy measure (as seen in Section 2).

Constrained optimization. Rather than minimizing (or maximizing) an objective function over a given search space, the FME problem also targets constraints that need to be satisfied. In other word, a solution to such a problem is defined by any element of the search space, among only those that satisfy all constraints, that minimizes the objective function. What this means is that the search for the optimum is only conducted in the feasible space of the constraints (elements of the search space that meet the constraints): constraints are requirements that are imposed that must be met; when a candidate meets the constraints it then can be considered a possible solution.

So, in the case of fuzzy measure extraction, fuzzy measures must be monotonic and must always take values between 0 and 1. These are considered the constraints. For the problem with n attributes, the number of monotonicity constraints is

$$\sum_{k=1}^{n-2} \binom{n}{k}(n-2).$$

The fuzzy measure extraction problem is an optimization problem subject to constraints (monotonicity), which can be reformulated as follows:

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

subject to:

$$\mu(A_{(x)}) \leq \mu(A_{(y)}), \text{ if } A_{(x)}, A_{(y)} \subseteq A \text{ and } A_{(x)} \subseteq A_{(y)}$$
$$0 \leq \mu(A_{(x)}) \leq 1, \ \forall A_{(x)} \in \mathcal{P}(A)$$

where A is a set of attributes, $A_{(i)} = \{\sigma(i), \ldots, \sigma(n)\}$, and $\mathcal{P}(A)$ is the power set of A.

3.2 Optimization Techniques for FME: A Brief Review

Several optimization approaches have been proposed to extract fuzzy measures. We briefly go over the main approaches in what follows:

Genetic algorithms have been successfully used to solve a number of optimization problems, including fuzzy measure extraction in [5, 34, 36]. Although they show promise for extracting fuzzy measures, they might fall into a local optimum. While mutations are part of genetic algorithms to try to avoid falling in local minima, they do not totally prevent it (especially if there are local optima that are in distant locations but have values close to the global optimum).

Gradient descent algorithms were also proposed for FME, see [9], taking advantage of the lattice structure of the coefficients of the fuzzy measure. Such approach can quickly and accurately reach a local optimum if the initial values are properly selected. However, the monotonicity constraints need to be checked at every iteration. This algorithm was improved on in [1] and the experiments conducted at that time showed that a gradient descent approach could easily overperform a genetic algorithm approach.

A neural network approach for FME was proposed in [33]: the relation among inputs, fuzzy measures, the output (Chouqet integral) were described by a neural network. However, such search easily falls in a local minimum.

The Bees algorithm proposed in [26], was also used for FME in [35]. It 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. Although this algorithm provided good results for FME, there was still not enough evidence to prove that the algorithm does not fall into a local optimum.

Besides these, many optimization techniques exist, such as for instance Harmony Search [8], Particle Swarm Optimization [27], Simulated Annealing [6].

Limitations of previous approaches. Although previous attempts have been shown to extract fuzzy measures successfully from sample data, they share the following limitations.

(1) 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.

(2) Uncertainty might be part of the model to solve. It is reasonable for experts to provide data in ranges instead of precise values. Using intervals allows to take uncertainty into account but none of the above techniques handled intervals/uncertainty.

(3) When dealing with problems defined on real numbers, the actual computations will round each real number to the most "relevant" floating-point number. Rounding errors can lead the returned result to be dramatically different from the original expected solution.

In what follows, we first motivate why MCDM is related to Software Quality Assessment (SQA) and how Fuzzy Measure Extraction can help automate SQA. We then provide details about the proposed optimization technique for Fuzzy Measure Extraction (FME).

4 Fuzzy Measures Extraction for Software Quality Assessment

As hinted earlier, Software Quality Assessment can be seen as a Multi-Criteria Decision-Making problem. The 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 then 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 2, determining μ consists in solving the following problem:

$$\boxed{\begin{array}{l}\min e = \sum_{A_i} \left\{\text{SQA}(A_i) - \text{Choquet}(\mu, \text{metrics}(A_i))\right\}^2 \\ \text{s.t.} \\ \quad \mu \text{ is a fuzzy measure}\end{array}}$$

5 Our Hybrid Approach to Fuzzy Measure Extraction

The work presented in this article addresses the problem presented in Section 4. Our approach aims at addressing some of the limitations of existing approaches to Fuzzy Measure Extraction, namely, the lack of guarantee that the returned solution is indeed an optimum of the objective function, the rounding errors due to floating-point computations, and the lack of support for uncertainty.

Our approach consists in pairing our previous Bees algorithm for FME with an interval constraint solver, RealPaver [13]. Using RealPaver allows us to focus the Bees' search on relevant and smaller areas of the search space by pruning areas that do not match the currently found optimum. RealPaver is repeatedly requested to solve the following continuous and non-linear constraint problems:

$$\boxed{\begin{array}{l}e = \sum_{A_i} \left\{\text{SQA}(A_i) - \text{Choquet}(\mu, \text{metrics}(A_i))\right\}^2 \leq f^* \\ \mu \text{ is a fuzzy measure}\end{array}} \quad (4)$$

where f^* is the best value our Bees algorithm has been able to identify at the time RealPaver is solicited. RealPaver then helps discard parts of the search space that do not satisfy the above constraints. In particular, it focuses the search on areas of the search space that are more likely to yield a better optimum.

Figure 1 illustrates the pairing of the Bees algorithm with RealPaver: the Bees algorithm first performs an exploration of the search space and converges on some areas of it (as denoted by the black dots in the image on the left hand-side). From this exploration, the Bees algorithm is already able to provide an upper bound for the value of the objective function (which is, and which we call, f^*). A constraint solving problem of the form shown in Equation 4 is then sent to Realpaver (right hand-side of Fig. 1), which in turns will reduce the search space by shrinking it: i.e., non-feasible (w.r.t. considered constraints) parts of the search space that lie on the outskirt of it are discarded. The reduced search space is then sent back to the Bees algorithm for resuming of the former search and further exploration.

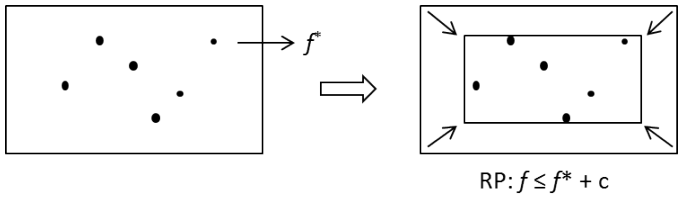

Figure 1: Using RealPaver to shrink the search space

Figure 2 illustrates a situation that can happen during the pairing of Bees and RealPaver: although it is sought that RealPaver will shrink the search space, it might happen that it does not. This means that shrinking the search space (outside-in) would potentially remove an element of the search space that is seen as a feasible element for the constraints. As a result, in such a situation, since we still aim to provide some focus to the Bees search by narrowing the search space, we then split the search space and recall RealPaver of each of the two halves: if shrinking at the outskirt of the original search space was not possible, it might be possible to discard parts of the search space within the original space.

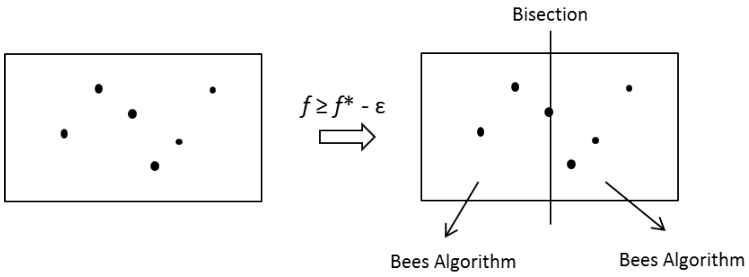

Figure 2: Using RealPaver to bisect the search space

Overall, our approach considered three modes of interaction of Bees and RealPaver, which we called Hybrid1, Hybrid2, and Hybrid3. They are described below:

1. Hybrid1: both RealPaver and the Bees algorithm are called only once. RealPaver is called at the start of the process to shrink the search space based on the fuzzy measure constraints (monotonity and range), and it is followed by the Bees algorithm that then only considers the resulting search space for exploration.

2. Hybrid2: both RealPaver and the Bees algorithm are called multiple times. They alternate until a given accuracy of the solution is reached. No bisection is allowed (i.e., the scenario of Figure 2 is not allowed).

3. Hybrid3: this version is similar to Hybrid2 except that it allows bisection. In this case, RealPaver and our Bees algorithm are invoked alternatively until there is no more reduction of the search space: at this point, the search space is split and the search is resumed on each of the halves, as shown in Figure 2 – the no-bisection mode is also restaured until the search is stalled again. The process ends when a given accuracy of the solution has been reached.

In what follows, we present the experiments we conducted, we report our results and analyze them.

6 Experiments Results on Toy Examples

6.1 Testing Methodology

The goal of our experiments on toy examples is to show that our hybrid approach allows to improve the quality of the extracted fuzzy measure. In order to show this, we designed a fuzzy measure μ, presented in Table 3, and ran our algorithm to see how well it was able to reconstruct it. Besides, in order to be able to compare the performance of the adaptive hybrid algorithm against existing approaches, we followed the same procedure as in [9] and [5], as described hereafter.

Table 3: Fuzzy measure to be identified

A	$\mu(A)$	**A**	$\mu(A)$	**A**	$\mu(A)$
$\{1\}$	0.1	$\{1,2\}$	0.3	$\{1,2,3\}$	0.5
$\{2\}$	0.2105	$\{1,3\}$	0.3235	$\{1,2,4\}$	0.8667
$\{3\}$	0.2353	$\{1,4\}$	0.7333	$\{1,3,4\}$	0.8824
$\{4\}$	0.6667	$\{2,3\}$	0.4211	$\{2,3,4\}$	0.9474
		$\{2,4\}$	0.8070		
		$\{3,4\}$	0.8235		

The input-output system contains 120 sample data with $Y = f_c(X) + n$, where Y is the vector of the system outputs, X is a 4-tuple input (x_1, x_2, x_3, x_4) with $x_i \in \{0, 1\}$, $f_c(X)$ is the computed Choquet integral over X, and n is a centered gaussian noise. In the same manner as it was done in [9] and [5], we also checked for 5 different variances, $\sigma^2 = 0.0$, 0.00096, 0.00125, 0.00625, and 0.0125, respectively. The tests for each variance were run 10 times, and the average result of these 10 tests was calculated as the result for each variance. Our algorithm was executed on an Intel Xeon e5540 @2.53GHz machine. The parameters used for the algorithm are presented in Table 1. In order to find the optimal solution (most fitting fuzzy measure), we were interested in finding the minimum of the least square error E, where

$$E = \frac{e}{m} = \frac{1}{120} \sum_{i=1}^{120} (y_i - \tilde{y_i})^2.$$

6.2 Quality of Solutions

Table 4 shows our experimental results for 1000 iterations. We compare them with the results of the Bees algorithm in the same table. The comparison of the Bees algorithm with other optimization algorithms is available in [35]. We observe that the mean square errors using hybrid approaches are slightly less than those obtained when using the Bees algorithm alone, which means that our hybrid approaches improve the quality of the extracted fuzzy measure. The final obtained fuzzy measures using Hybrid3 are reported in Table 5, and we can observe that they are very close to what we expected to get.

Figures 3 to 7 present the comparison of the convergence diagram for samples with different noise. We observe that with the same number of iteration for the Bees algorithm (comparing Bees and Hybrid1), using RealPaver, even when it is allowed to shrink the search space only once, helps to improve the performance. However, with the same total number of iteration (iterations for each Bees call multiplies the number of Bees calls), reducing the iteration for each Bees call and increasing the number of RealPaver calls to keep shrinking the search space does not seem to significantly improve the performance of the search (in comparison with Hybrid1 and Hybrid2). Moreover, using RealPaver to bisect the search space (Hybrid3) improves the performance when the total number of iterations is increased and there is no improvement of using a shrink-Bees process.

Table 4: Comparison with the Bees algorithm (Total number of iteration = 1000)

σ^2	Bees Algorithm	Hybrid1	Hybrid1	Hybrid3
0.0	2.54E-7	1.38E-7	1.59E-7	2.56E-8
0.00096	0.00070043	0.00070056	0.00070043	0.00070042
0.00125	0.0012948	0.0012945	0.0012944	0.0012943
0.00625	0.0055944	0.0055943	0.0055945	0.0055943
0.01250	0.0143783	0.0143775	0.0143771	0.0143777

Table 5: Obtained fuzzy measures using Hybrid3

μ, σ^2	0.0	0.00096	0.00125	0.00625	0.0125
$\{1\}$	0.10002294	0.1056434	0.08216595	0.046254	0.1380325
$\{2\}$	0.2105267	0.1843541	0.1763852	0.1611654	0.1295627
$\{3\}$	0.23539	0.224232	0.246674	0.2772939	0.1619649
$\{4\}$	0.6665281	0.6694569	0.677733	0.5052198	0.6673205
$\{1,2\}$	0.3000861	0.3130654	0.311589	0.2889373	0.2776955
$\{1,3\}$	0.3232191	0.31432	0.3844483	0.2815279	0.5913198
$\{1,4\}$	0.7332572	0.6781726	0.6827574	0.8223756	0.6845643
$\{2,3\}$	0.4211996	0.4566696	0.3972005	0.4599749	0.3210012
$\{2,4\}$	0.8068813	0.7907318	0.8086925	0.8476835	0.8375995
$\{3,4\}$	0.8234168	0.8587281	0.8081816	0.8437407	0.827119
$\{1,2,3\}$	0.4999835	0.5025472	0.5008131	0.574426	0.591341
$\{1,2,4\}$	0.8666455	0.8929275	0.8087053	0.9582186	0.9380317
$\{1,3,4\}$	0.8824617	0.8755151	0.8770698	0.8771552	0.8783669
$\{1,3,4\}$	0.9478859	0.9702236	0.9741276	0.9379999	0.9900401

7 Experiments and Results on SQA

7.1 Testing Methodology

In the previous section, we have shown that hybrid approaches help to improve the quality of the extracted fuzzy measure. We now aim at assessing the performance of our approach in automatically predicting software quality. The metrics we use are the Quality Model for Object-Oriented Design (QMOOD) metrics, as defined in [2] and used by Virani et al [32]. We focus on the metrics and the quality factors related to QMOOD model, as in [2], as defined in Table 1.

The data at our disposal to run our tests comes from 31 software packages and is composed of 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 criteria and for the total quality. Note that the data is the same data set as used in [23], in which the authors used WEKA, a machine learning tool to model and predict SQA with the same rating classes.

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.

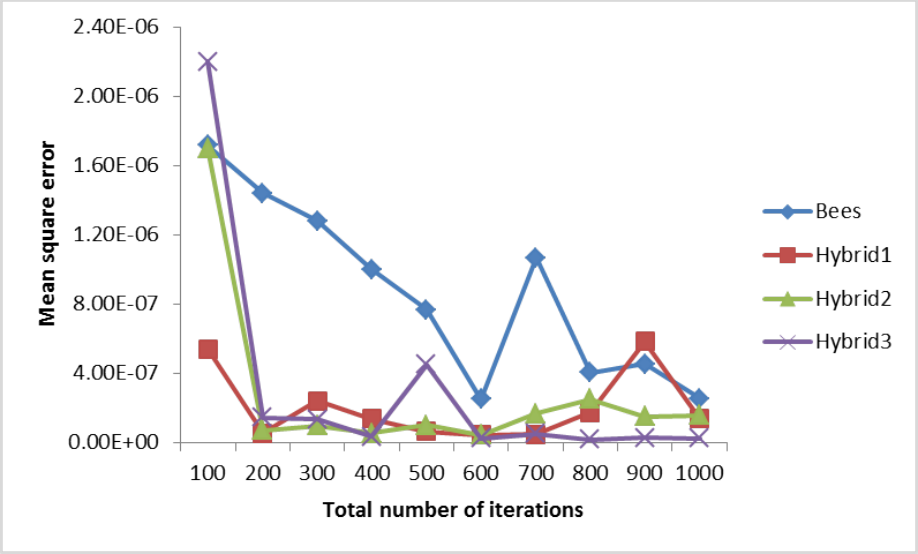

Figure 3: Convergence based on number of iterations ($\sigma^2 = 0.0$)

7.2 Experimental Results

We applied the same 3 configurations on SQA data: Hybrid1, Hybrid2, Hybrid3, together with the Bees algorithm. Each configuration was run 10 times for each property factor, and we calculated the average as the final results. To be able to compare against the performance of the Bees algorithm, each Bees algorithm in Hybrid1, Hybrid2 and Hybrid3 was run 1000 times, and the overall number of iterations in Hybrid2 and Hybrid3 was 10000.

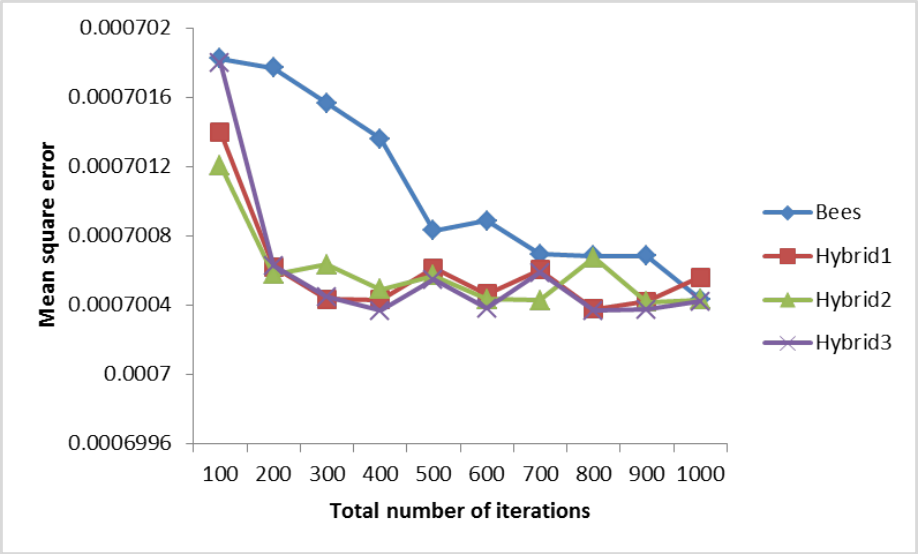

Figure 4: Convergence based on number of iterations ($\sigma^2 = 0.00096$)

Values of e, as defined in Section 3, are reported in Table 6. We can observe that Hybrid3 is slightly more efficient than Hybrid1 and Hybrid2. Overall, Hybrid3 is as efficient as the Bees algorithm alone, which indicates that, in general, the Bees algorithm does not fall into a local minima for the tests we ran: the advantage of Hybrid3 over Bees is that Hybrid3 will guarantee that the search is indeed global, as opposed to the Bees, which will not come with any guarantee.

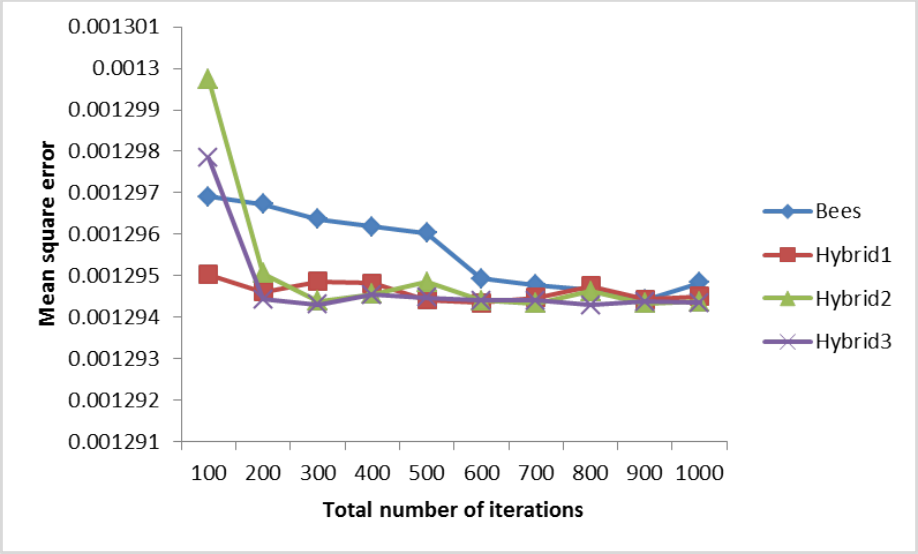

Figure 5: Convergence based on number of iterations ($\sigma^2 = 0.00125$)

In addition, let us note that, within 1000 iterations, the Bees algorithm was not able to find the optimal fuzzy measures for Effectiveness and Understandability that satisfied all constraints.

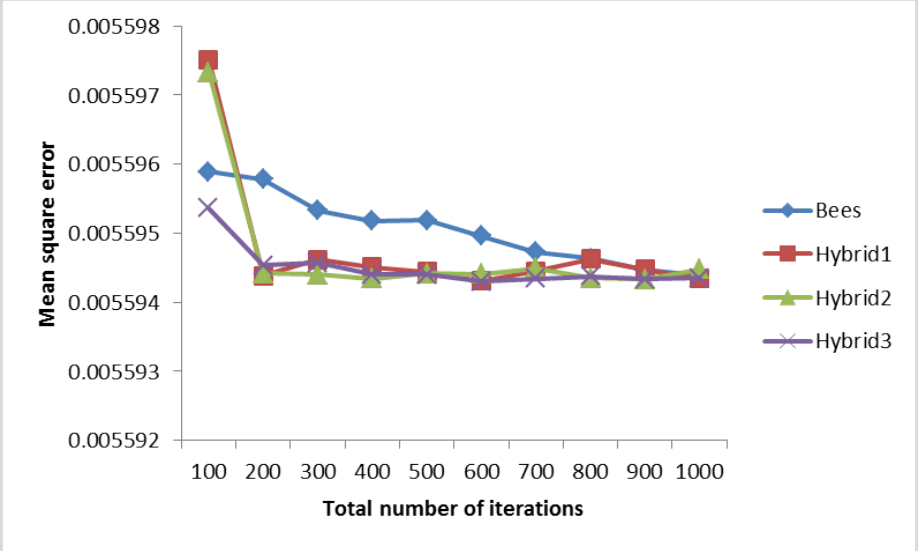

Figure 6: Convergence based on number of iterations ($\sigma^2 = 0.00625$)

7.3 Discussion of Quality Assessment

We used the obtained fuzzy measures to recreate the experts' decision (our seed data). We compared the evaluations we obtained to the original experts' decision and assessed the accuracy of our approach using 3 different evaluation processes.

As we mentioned earlier, the sample data consists of 31 software packages and 2330 samples. For reusability, these 2330 samples can be grouped into 264 groups by input values (DCC, CAM, CIS and DSC). In each group, with the same input values, experts' decision are distributed among five rating classes (bad, poor, fair, good, excellent). We computed the average assessment μ and the standard deviation σ over the known expert's decisions for each group.

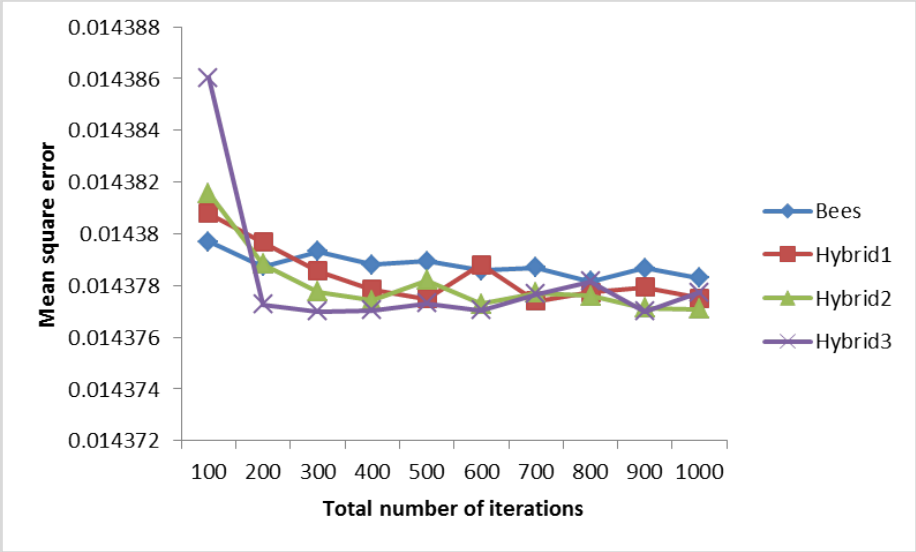

Figure 7: Convergence based on number of iterations ($\sigma^2 = 0.01250$)

1. Eval1: For each group, we calculated how close the predicted decisions (calculated using Choquet integral) are to the known experts' decisions using Gaussian distribution

$$f = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}.$$

   We compute a percentage of accuracy for each predicted decision as follows: the closer the predicted decision is to the average assessment, the higher the accuracy is.

2. Eval2: We used the same average as for Eval1 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. Every predicted decision between $\mu - \sigma$ and $\mu + \sigma$ is considered to be an accurate decision.

3. Eval3: Instead of performing the assessment based on the average, we assume that the maximum number of votes for one rating would correspond to the 100% accuracy, and the accuracy for the other possible ratings depends on the number of votes for it (as compared to the 100% accuracy). We then 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 closer the predicted decision is to the majority rating, the higher the accuracy is.

Table 6: Comparison with the other algorithms

Quality Factor	Bees Algorithm	Hybrid1	Hybrid2	Hybrid3
Reusability	0.076735	0.076753	0.076736	**0.076713**
Flexibility	0.098318	0.099891	0.097946	**0.097942**
Extendibility	0.104124	0.104836	0.104142	**0.104117**
Functionality	0.072958	0.073175	0.072948	**0.072918**

The overall evaluation results for all quality factors using Hybrid3 are in Table 7; we also list and compare results obtained using the machine learning approach used in [23].

The accuracy we reached using either Eval1 and Eval2 were similar. Considering the same piece of software, experts often disagree on the decisions to make, sometimes radically. As a result, the reported accuracy of our approach using this evaluation may be affected. We observe, in particular, that when allowing some uncertainty (also seen as flexibility), the reported accuracy of our approach improved.

We then considered the distribution of all decisions and evaluated whether the measured quality matched the majority decisions (Eval3). Although the accuracy reported when using Eval3 does not match that of Eval1 and Eval2, it still comes close to that obtained through machine learning and we believe that Eval3 is more relevant to group decision than the other two.

Table 7: Accuracy using Hybrid3

Quality Factor	Traditional Machine learning approach [23]	Eval1 (Average)	Eval2 ($\mu \pm \sigma$)	Eval3
Reusability	62.08%	76.91%	79.17%	67.80%
Flexibility	62.12%	73.52%	74.33%	65.97%
Extendibility	63.84%	75.26%	76.65%	69.88%
Functionality	66.14%	61.71%	57.68%	58.99%

8 Conclusion and Future Work

In this article, we proposed a multi-criteria decision-making-based approach to Software Quality Assessment (SQA). By viewing this assessment problem as a multi-criteria decision making (MCDM) problem, we were able to use fuzzy measures to model software experts' decision making processes 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 hybrid algorithms we implemented provide good and reasonably fast results for fuzzy measure extraction, we believe we can improve them by designing better interaction modes and possibly involving a third solver in the process. Moreover, the experts' 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, experts' 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 lower accuracy. In order to better fit the experts' opinions, in the future, we will use an interval end-points approach [18] and may use non-linear utility functions. Finally, we plan to study and quantify 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?

References

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

[2] Bansiya, J., and C. Davis, A hierarchical model for object-oriented design quality assessment, *IEEE Transactions on Software Engineering*, vol.28, no.1, pp.4–17, 2002.

[3] Brewerton, P.M., and L.J. Millward, *Organizational Research Methods: A Guide for Students and Researchers*, Sage Publications, 2001.

[4] Ceberio, M., and F. Modave, An interval-valued, 2-additive Choquet integral for multi-criteria decision making, *Proceedings of the 10th Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2004.

[5] Combarro, E.F., and P. Miranda, Identification of fuzzy measures from sample data with genetic algorithms, *Computers & Operations Research*, vol.33, no.10, pp.3046–3066, 2006.

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

[7] Etzkorn, L.H., W.E.J. Hughes, and C.G. Davis, Automated reusability quality analysis of OO legacy software, *Information & Software Technology*, vol.43, no.5, pp.295–308, 2001.

[8] Geem, Z.W., J.H. Kim, and G.V. Loganathan, A new heuristic optimization algorithm: harmony search, *Simulation*, vol.76, no.2, pp.60–68, 2001.

[9] Grabisch, M., A new algorithm for identifying fuzzy measures and its application to pattern recognition, *Proceedings of 4th IEEE International Conference on Fuzzy Systems*, vol.1, pp.145–150, 1995.

[10] Grabisch, M., The application of fuzzy integrals in multicriteria decision making, *European Journal of Operational Research*, vol.89, no.3, pp.445–456, 1996.

[11] Grabisch, M., I. Kojadinovic, and P. Meyer, A review of methods for capacity identification in Choquet integral based multi-attribute utility theory applications of the Kappalab R Package, *European Journal of Operational Research*, vol.186, pp.766–785, 2008.

[12] Grabisch, M., H.T. Nguyen, and E.A. Walker, *Fundamentals of Uncertainty Calculi with Applications to Fuzzy Inference*, Kluwer Academic Publishers, Norwell, MA, 1994.

[13] Granvilliers, L., and F. Benhamou, RealPaver: an interval solver using constraint satisfaction techniques, *ACM Transactions on Mathematical Software*, vol.32, no.1, pp.138–156, 2006.

[14] Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, The WEKA data mining software: an update, *SIGKDD Explorations*, vol.11, no.1, pp.10–18, 2009.

[15] Kitchenham, B., S. Pfleeger, and N. Fenton, Towards a framework for software measurement validation, *IEEE Transactions on Software Engineering*, vol.21, no.12, pp.929–944, 1995.

[16] Lorenz, M., and J. Kidd, *Object-oriented Software Metrics: A Practical Guide*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994,

[17] Magoc, T., and V. Kreinovich, How to relate fuzzy and OWA estimates, *Proceedings of North American Fuzzy Information Processing Society*, 2010.

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

[19] Mitchell, T.M., *Machine Learning*, 1st Edition, McGraw-Hill, New York, 1997.

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

[21] Modave, F., and P.W. Eklund, A measurement theory perspective for MCDM, *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, pp.1068–1071, 2001.

[22] Moody, D.L., Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions, *Data & Knowledge Engineering*, vol.55, no.3, pp.243–276, 2005.

[23] Osbeck, J., S. Virani, O. Fuentes, and P. Roden, Investigation of automatic prediction of software quality, *Proceedings of North American Fuzzy Information Processing Society*, 2011.

[24] Paulk, M.C., C.V. Weber, B. Curtis, and M.B. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley Pub. Co, 1995,

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

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

[27] Poli, R., J. Kennedy, and T. Blackwell, Particle swam optimization, *Swarm Intelligence*, vol.1, no.1, pp.33–57, 2007.

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

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

[30] Takahagi, E., 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`.

[31] Takahagi, E., A fuzzy measure identification method by diamond pairwise comparisons and ϕ_s transformation, *Fuzzy Optimization and Decision Making*, vol.7, no.3, pp.219–232, 2008.

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

[33] Wang, J., and Z. Wang, Using neural networks to determine Sugeno measures by statistics, *Neural Networks*, vol.10, no.1, pp.183–195, 1997.

[34] Wang, W., Z. Wang, and G.J. Klir, Genetic algorithms for determining fuzzy measures from data, *Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology*, vol.6, no.2, pp.171–183, 1998.

[35] Wang, X., J. Cummins, and M. Ceberio, The Bees algorithm to extract fuzzy measures for sample data, *Proceedings of North American Fuzzy Information Processing Society*, 2011.

[36] Wang, Z., 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*, vol.102, no.3, pp.463–469, 1999.

[37] Weiss, J.W., and D. Anderson Jr, CIOs and IT professionals as change agents, risk and stakeholder managers: a field study, *Proceedings of the 36th Hawaii International Conference on System Sciences*, vol.8, pp.249.3, 2003.