Interval-based Algorithms to Extract Fuzzy Measures for Software Quality Assessment

Xiaojing Wang*, Angel F. Garcia Contreras†, Martine Ceberio‡,
Christian Del Hoyo§, and Luis C. Gutierrez¶, Shamsnaz Virani‖
Computer Science Department
The University of Texas at El Paso
El Paso, Texas 79968-0518
* xwang@utep.edu, † afgarciacontreras@miners.utep.edu, ‡ mceberio@utep.edu,
§ cdelhoyo@miners.utep.edu, ¶ lcgutierrez@miners.utep.edu, and ‖ ssv1@psu.edu

***Abstract*—In this paper, we consider the problem of automatically assessing sofware quality. We show that we can look at this problem, called Software Quality Assessment (SQA), as a multicriteria decision-making problem. Indeed, just like software is assessed along different criteria, Multi-Criteria Decision Making (MCDM) is about decisions that 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 to model and aggregate the levels of satisfaction of these criteria by considering their relationships. However, in practice, fuzzy measures are difficult to identify. An automated process is necessary and possible when sample data is available. Several optimization approaches have been proposed to extract fuzzy measures from sample data; e.g., genetic algorithms, gradient descent algorithms, and the Bees algorithm, all local search techniques. In this article, we propose a hybrid approach, combining the Bees algorithm and an interval constraint solver, resulting in a focused search expected to be less prone to falling into local results. Our approach, when tested on SQA decision data, shows promise and compares well to previous approaches to SQA that were using machine learning techniques.**

I. 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.

Software quality assessment (SQA) is categorized in three ways [15]:

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

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).

In general, when a decision is not critical, we mentally "average/sort" criteria along with their satisfaction levels. Usually, the satisfaction level of each criterion is subject to the decision maker's personal preference. The overall score of an alternative is then calculated by aggregating the values of satisfaction with weights on each criterion, where the weight indicates how importance each criterion is over the entire set of criteria. This approach is called a weighted sum approach, and the weight assigned to different sets of criteria in this approach forms an "additive measure". However, such additive aggregation assumes that criteria are independent, which is seldom the case [3]. Non-linear approaches also prove to lead to solutions that are not completely relevant [12]. 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, such non-additive functions assigning numbers 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.

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 [10], we focus here on extracting fuzzy measures from sample data.

The sample 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 (i.e., items that we need to decide on, such as cars). The SQA data we used in this work are based on direct measurements of code properties that follow QMOOD set of metrics [2].

Fuzzy measure extraction seeks to determine the fuzzy measure that, when combined in a Choquet integral, returns a value that 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

978-1-4673-2338-3/12/$31.00 ©2012 IEEE

sample data, such as gradient descent algorithms [6], genetic algorithms [4], [24], [26], and the Bees algorithm [25]. 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 identify fuzzy measures from sample data. The Bees algorithm has been successfully used in many optimization problems but requires a significant amount of tuning to attain reasonable performance. We adjusted it to the needs of fuzzy measure extraction problems: in [25], we showed that using the Bees algorithm to extract fuzzy measures from sample data provides better performance with results of similar or better accuracy than existing approaches. In this article, we use an interval constraint solver to shrink the search space to focus the Bees search to improve the overall performance and provide a certificate that we explore the right parts of the search space.

The article is organized as follows: Section II provides background and recalls necessary definitions on SQA, MCDM, fuzzy measures, fuzzy integrals, and fuzzy measure extractions. Section III introduces Fuzzy Measure Extraction (FME) as an optimization problem and recalls existing approaches to FME. We present our approach in Section IV, describe our experimental strategy, report, and analyze the results in Section V. Finally, we draw conclusions and propose directions for future work in Section VI.

II. Background

A. 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" [17]. 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 kind of software quality is to some extent measurable [18]. 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. The implicit aspect of software quality is solely based on expert opinion which is frequently inconsistent and subjective [18]. There is a need to directly measure the implicit aspect of software quality.

There are two main type of software quality models: theoretical models and predictive models. 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 of the theoretical software quality models that defined the quality factors and linked all of them to measurable metrics in object-oriented software was proposed by Bansiya and Davis [2], and this model defines six quality factors and linked them to QMOOD set of metrics defined in Table I.

Although the Bansiya and Davis's 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 [5].

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 [11]. Osbeck et al improved the prediction capability using J48, Part, and Random Forest, and the ensemble learning techniques examined were boosting, bagging, and stacking [14].

In our work, we show that SQA can be seen as a MCDM problem, and software product quality can be predicted by using fuzzy measures and Choquet integral.

B. Multicriteria Decision Making

Multicriteria decision making (MCDM) is the making of decisions based on multiple criteria (or attributes). In general, it consists of:

- X is the set of consequences;
- $A = \{1, \cdots, n\}$ is the (finite) set of n criteria (or attributes); and
- $\succeq$ is a preference relation 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 criterion i, where $i \in A$. For each $i \in A$, 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. And there is a global preference relation $\succeq$ on X.

Note: The reason why X can be a proper subset of $X_1 \times \cdots \times X_n$ is because not all combinations of all criteria values necessarily exist: each n-tuple of $X_1 \times \cdots \times X_n$ represents a possible instance / an alternative to pick from, all of which are not necessarily possible. For instance, consider the case of cars: one criterion being the price, another being the year of make. It is unlikely that the lowest value of the price criterion can match any high value of the year of make; *i.e., there is likely no recent car that is very cheap*.

Then an aggregation operator that "combines" the monodimensional preferences needs to be used to represent the global preference, i.e., a preference over the set of consequences X: $\forall x, y \in X$, $x \succeq y$ 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	

As mentioned in introduction, 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 represents the level of "satisfaction" of criterion i. The best alternative is the one with the highest value of u. However simple, easy, and low-complexity this approach is, using an additive aggregation operator assumes that all the criteria are independent, which 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. Based on our previous work [12], we choose to use non-additive approaches, i.e., fuzzy measures and integrals [3].

C. 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].

Definition 1. *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 $\mu(X) \leq \mu(Y)$*

Once a fuzzy measure is identified, two main integrals can be used as "aggregation" operators: the Sugeno and the Choquet integrals. Although structurally similar, they are different in nature [7]: 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 [13]: 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 2. *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 $f(\sigma(1)) \leq \cdots \leq f(\sigma(n))$, $A_{(i)} = \{\sigma(i), \ldots, \sigma(n)\}$ and $f(\sigma(0)) = 0$, by convention.

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.

D. Determining Fuzzy Measures

Although we would expect decision makers to be likely to provide the values of the fuzzy measure, in most circumstances this is not the case. Attempts at making fuzzy measure identification easier for the decision makers have been made in [3], [20], [21].

- 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 [21], 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 [20], the author discusses user specified weights mixed with an interaction index denoted λ or ξ. This algorithm is applied using an online aggregation application [19].

However, in most cases, the decision maker does not understand the interactions well enough to be able to provide relevant values of the fuzzy measure. This is where fuzzy measure extraction comes into play.

III. Fuzzy Measure Extraction (FME) and Optimization

A. Relation Between our Problem 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 / expert's opinions: we use sample data. Our objective is to determine a fuzzy measure that returns the closest values to our seed data (the expert's known decisions).

Let us take a look at the following situation:
Our MCDM problem involves n criteria, and we have m sample data. It means that we have access to the following: m expert's decision values $\tilde{y}_j$, $j \in \{1, \cdots, m\}$, corresponding to m alternative items (in our case, software pieces). As a result, if we computed the perfect corresponding fuzzy measure, denoted by $\tilde{\mu}$, we would have, $\forall j \in \{1, ..., m\}$:

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

where f is a utility function defined on X.

In practice however, for lack of consistency in sample decision data, we "only" aim at getting as close to seed data as possible. As a result, we aim at minimizing the following sum (and getting the "error" e as close to 0 as possible) [8]:

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

In addition to minimizing e, constraints need to be satisfied, which ensure that we obtain a fuzzy measure. Fuzzy measures must be monotonic (1) and their values must be between 0 and 1 (2). (1) defines the constraints of the FME problem; (2) defines the search space. For a problem with n criteria, the number of monotonicity constraints is $\sum_{k=1}^{n-2}\binom{n}{k} * (n-2)$. As a result, extracting a fuzzy measure is cast down to solving a constrained optimization problem. When $e = 0$, the identified fuzzy measure μ is the exact solution of the problem: this is the ideal case. In most cases, the sample data might not be fully consistent with one fuzzy measure, i.e., human decisions are not always consistent, and we "only" reach an approximate optimal solution, that is, with $e \neq 0$ but close to 0.

B. Optimization Techniques used for FME

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 [4], [24], and [26]. 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 [6], 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 [23]: the calculation of the Choquet integral was described by a neural network. However, such search easily falls in a local minimum.

The Bees algorithm, proposed in [16], was also used for FME in [25]. 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.

IV. Identifying Fuzzy Measures Using Interval-based Algorithms

Although previous attempts have been shown to extract fuzzy measures successfully from sample data, they have limitations. In particular, the returned solution (found minimum of the objective function) might just be a local minimum, or even worse, a good value. Moreover, 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 this kind of uncertainty into account. Furthermore,

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.

The work presented in this article addresses the above-mentioned issues: it focuses the search on relevant areas of the search space by pruning areas that do not match the currently found optimum, it can process interval data, and will not be prone to rounding errors. We propose an adaptive hybrid interval-based approach that combines the Bees algorithm (a global local search algorithm) with an interval constraint solver.

We use RealPaver [9], a complete, interval-based, continuous constraint solver. In the context of our hybrid solver, RealPaver is used to reliably discard parts of the search space that do not contain solutions, in order to help the Bees algorithm focus its search on feasible regions, see Figure 1.

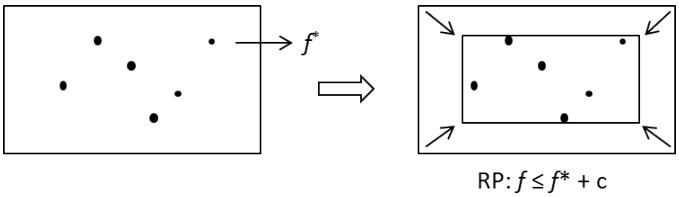

Fig. 1. Using RealPaver to shrink the search space

V. Experiments and Results

A. Testing Methodology

We 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 [22]. We focus on the metrics and the quality factors related to QMOOD model, as in [2], as defined in Table I.

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 [14], 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.

B. Experimental Results

When using our hybrid algorithm, we considered 2 configurations:

- Hybrid1 – Both RealPaver and the Bees algorithm are called only once. RealPaver is called in the beginning to shrink the search space first, then the Bees algorithm only is run.
- Hybrid2 – Both RealPaver and the Bees algorithm are called multiple times. The Bees algorithm is called after each run of RealPaver until a given accuracy is reached.

Each configuration is run 10 times on each sample decision data, and we calculate the average as the final results. To be able to compare against the performance of the Bees algorithm, each Bees algorithm in Hybrid1 and Hybrid2 is run 1000 times, and the overall number of iterations in Hybrid2 is 5000.

Values of e, as defined in Section III, are reported in Table II. We can observe that Hybrid2 is slightly more efficient than Hybrid1. Overall, Hybrid2 is as efficient as Bees alone, which indicates that, in general, the Bees algorithm does not fall into a local minimum: the advantage of Hybrid2 over Bees is that Hybrid2 will guarantee the global search, as opposed to the Bees, which will not come with any guarantee. In addition, let us note that, within 1000 iterations, the Bees algorithm could not find the optimal fuzzy measures for Effectiveness and Understandability that satisfied all constraints.

TABLE II
Comparison with the other algorithms

Quality Factor	Bees Algorithm	Hybrid1	Hybrid2
Reusability	0.076735	0.076753	0.076725
Flexibility	0.098318	0.099890	0.098316
Extendibility	0.104124	0.104836	0.104063
Functionality	0.072958	0.073175	0.072967

C. Discussion of quality assessment

We then 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.

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 III, we also list the results using machine learning approach to compare with.

The accuracy using Eval1 is low. With the same group of inputs, the experts' decisions are different; sometimes, they may even draw exactly opposite decisions. As a result, the reported accuracy of our approach using this evaluation may be affected. We observe, in particular, that when allowing some uncertainty (Eval2, also seen as flexibility), the reported accuracy of our approach significantly improved.

We then consider the distribution of all decisions and evaluate if the measured quality matches the majority decisions (Eval3). Although the accuracy reported when using Eval3 does not match that of 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 III
ACCURACY USING HYBRID2

Quality Factor	Machine learning approach [14]	Eval 1 (Average)	Eval 2 ($\mu \pm \sigma$)	Eval 3
Reusability	69.91%	42.05%	79.17%	72.12%
Flexibility	75.39%	47.13%	74.71%	66.62%
Extendibility	70.37%	42.80%	77.04%	71.02%
Functionality	78.54%	33.61%	57.68%	64.02%

VI. CONCLUSION AND FUTURE WORK

In this article, we proposed a hybrid interval-based algorithm that combines Bees with an interval constraint solver that narrows down the search space to focus on areas that are likely to contain better values of the objective function. We tested our approach in the context of software quality assessment and compared our results with those of previous work using machine learning. We observed that our approach yielded similar results overall, depending on the evaluation rules.

We need to explore further the definition of a more relevant evaluation process and conduct more comparisons with the machine learning approach. We might consider combining them. We also plan to use our FME approach to help model non-expert decision-making process, helping understand which information factors contribute to changing decisions' outcomes.

ACKNOWLEDGMENT

The work presented here was partially supported by NSF grant CCF No. 0953339.

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*, 28(1):4–17, 2002.

[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] Moody D. L. Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, 55(3):243–276, 2005.

[6] 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*, volume 1, pages 145–150, Yokohama, Japan, March 1995.

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

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

[9] L. Granvilliers and F. Benhamou. Realpaver: An interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):138–156, 2006.

[10] T. Magoč and V. Kreinovich. How to relate fuzzy and owa estimates. In *Proceedings of North American Fuzzy Information Processing Society (NAFIPS'2010)*, Toronto, Canada, July 2010.

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

[12] 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.

[13] 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.

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

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

[16] 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.

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

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

[19] 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.

[20] E. Takahagi. On identification methods of λ -fuzzy measures using weights and λ. *Japanese Journal of Fuzzy Sets and Systems*, 12(5):665–676, 2000.

[21] 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.

[22] 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.

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

[24] W. Wang, Z. Wang, and G. J. Klir. Genetic algorithms for determining fuzzy measures from data. *Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology*, 6(2):171–183, 1998.

[25] X. Wang, J. Cummins, and M. Ceberio. The Bees algorithm to extract fuzzy measures for sample data. In *Proceedings of 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*, 102(3):463–469, 1999.