ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Fuzzy Sets and Systems 145 (2004) 57–80

FUZZY sets and systems

www.elsevier.com/locate/fss

Fuzzy logic-based object-oriented methods to reduce quantization error and contextual bias problems in software development

Francesco Marcelloni[a,*], Mehmet Aksit[b,1]

[a]*Dipartimento di Ingegneria della Informazione, University of Pisa, Via Diotisalvi, 2-56122 Pisa, Italy*
[b]*Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*

Abstract

During the last several years, a considerable number of software development methods have been introduced to produce robust, reusable and adaptable software systems. Methods create software artifacts through the application of a large number of heuristic rules. These rules are generally expressed in two-valued logic. In object-oriented methods, for instance, candidate classes are identified by applying the following intuitive rule: "If an entity in a requirement specification is relevant and can exist autonomously in the application domain, then select it as a class". In this paper, we identify and define two major problems regarding how rules are defined and applied in current methods. First, two-valued logic cannot effectively express the approximate and inexact nature of a typical software development process. Although software engineers can perceive partial relevance of an entity and possibly select the entity as a partial candidate class, they are constrained by two-valued logic to quantize relevance into relevant and irrelevant. Second, the influence of contextual factors on rules is generally not modelled explicitly. We term these problems as quantization error and contextual bias problems, respectively. To reduce these problems, we propose to express heuristic rules using fuzzy logic. We illustrate formally how fuzzy logic-based methodological rules can help in lowering the effects of quantization error and contextual bias problems.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Approximate reasoning; Fuzzy inference systems; Object-oriented methods; Quantization error; Adaptable design models

* Corresponding author. Tel.: +39-50-568678; fax: +39-50-568522.
E-mail addresses: f.marcelloni@iet.unipi.it (F. Marcelloni), aksit@cs.utwente.nl (M. Aksit).
[1] Also for correspondence.

0165-0114/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.fss.2003.10.005

1. Introduction

During the last several years, a considerable number of object-oriented methods [4,10,12,23] have been introduced to produce robust, reusable and adaptable software systems. Object-oriented methods create software artifacts by exploiting object-oriented concepts through the application of a large number of rules. For example, the OMT method [23] introduces rules for identifying and discarding classes, associations, part-of and inheritance relations, state-transition and data-flow diagrams. Basically, these rules are expressed by using two-valued logic. A candidate class is, for instance, generally identified by applying the rule: "If an entity in a requirement specification is relevant and can exist autonomously in the application domain then select it as a candidate class".

We observe the so-called *quantization error* and *contextual bias* problems in how rules are defined and applied in current object-oriented methods. Two-valued logic cannot conveniently express the approximate and inexact nature of software development processes. For example, to identify a class, a software engineer has to determine whether the entity being considered is relevant or not for the application domain. Intuitively, the more the entity is relevant, the more it is a class. The software engineer, however, may perceive different grades of relevance of an entity in the requirement specification and may conclude that the entity partially fulfills the relevance criterion. This conclusion implies the classification of the entity as a partial candidate class. Methodological rules, however, force the software engineer to take abrupt decisions, such as accepting or rejecting the entity as a class. This results in loss of information because the information about the partial relevance of the entity is not modelled and therefore cannot be considered explicitly in the subsequent phases of the development process. Moreover, the validity of a rule may largely depend on contextual factors such as the application domain, changes in user's interest and technological advances. Unless the relevant contextual factors that influence a given rule are defined explicitly, the applicability of that rule cannot be determined and controlled effectively.

In some previous papers [1,18,19,20], we have shown how fuzzy logic can be applied in modelling software development methods. The advantages of using fuzzy logic were explained in an intuitive manner without resorting to any formalism. In this paper, we illustrate the quantization error and contextual bias problems in a formal manner. In the following section, we introduce the definition and the mathematical formulation of the quantization error. In Section 3, we explain the contextual bias problem. In Section 4, we describe how software development methods can be modelled using fuzzy logic. The effect of using fuzzy logic in reducing the quantization error is illustrated in Section 5. Section 6 discusses how fuzzy logic-based methods can be used in modelling the context. Section 7 refers to the related work. Section 8 presents our future work. Finally, Section 9 gives conclusions.

2. The quantization error problem

2.1. Definitions

A software development method may be characterized in terms of three major components: artifacts, heuristics and software process [5]. Classes, attributes, operations, and inheritance and part-of relations are examples of object-oriented artifacts. To identify or eliminate an artifact, and relate an artifact to other artifacts, methods provide heuristics. In most methods, heuristics are defined

informally using textual forms in a natural language (see, for instance, [13,22,23]). Artifacts may have some casual order among each other. The heuristics implicitly express how an artifact is casually related to other artifacts.

Consider the following rule *Candidate Class Identification*, which is used to identify candidate classes by some popular object-oriented methods:

IF an entity in a requirement specification is relevant **AND** can exist autonomously in the application domain **THEN** select it as a candidate class.

Here, *an entity* and *a candidate class* are the two object-oriented artifacts. *Relevant* and *Autonomously* are the *input values* for the first and second conditions, respectively. If the antecedent of the rule is *true*, then the result of this rule is the classification of an entity in a requirement specification as a candidate class. For illustration purposes, in this article we will refer to a number of rules, which are adopted by most of the object-oriented methods.

After identifying candidate classes, redundant classes can be eliminated for instance by using the rule *Redundant Class Elimination*:

IF two candidate classes express the same information **THEN** discard the least descriptive one.

In general, application of a rule *quantizes* a set of object-oriented artifacts into two subsets: *accepted* or *rejected*. Once an artifact has been classified, for instance into the rejected set of a rule, it is not considered anymore by the rules that apply to the accepted set of that rule. For example, after applying the rule *Candidate Class Identification*, if an entity in a requirement specification is not selected as a candidate class, then this entity will not be considered by the rule *Redundant Class Elimination*. Of course, a rejected entity can be considered by another rule, which applies to the entities in a requirement specification. Consider, for example, the rule *Candidate Attribute Identification*:

IF an entity in a requirement specification is relevant **AND** cannot exist autonomously in the application domain, **THEN** identify it as an candidate attribute.

This rule can be applied to the entities in a requirement specification, which are rejected by the rule *Candidate Class Identification*. If all the rules, which are applicable to an entity in a requirement specification, reject that entity, then the entity is practically discarded. Improper classification of artifacts, especially in the early phases of the development process, may irremediably deteriorate the quality of the overall development process. The *quantization process* carried by the methodological rules is therefore crucial to the quality of the final product.

We believe that the *quantization process* as defined by current methods is problematic and generates a high *quantization error*. To make this clear, we refer to the area of digital signal processing because quantization errors have been extensively studied in this field [24]. In digital signal processing, the quantization process assigns the amplitudes of a sampled analog signal to a prescribed number of discrete *quantization levels*. This results in a loss of information because the quantized signal is an approximation of the analog signal. The quantization error is defined as the difference between an analog and the corresponding quantized signal sample. The quantization error is therefore inversely proportional to the number of quantization levels.

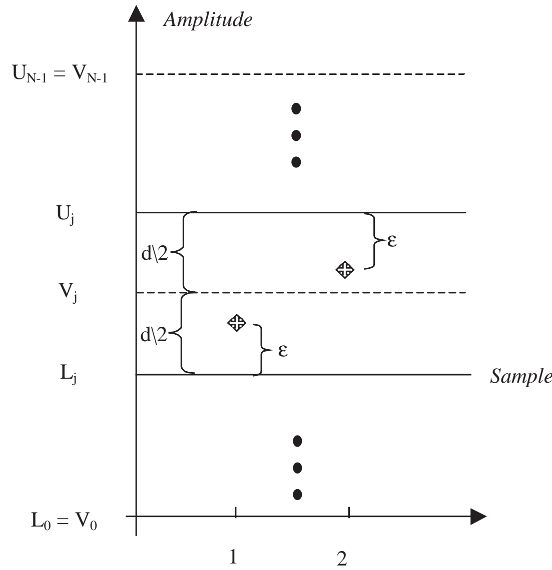

Fig. 1. Quantization error with N quantization levels.

2.2. Formulation of the quantization error

In the following, we will first formulate the error in the quantization process of a sampled signal. Then, we will use these formulas as a basis to calculate quantization errors in object-oriented methods.

The quantization error is directly related to the difference between adjacent discrete amplitude levels. The error can be reduced to any desired grade by choosing the difference between levels small enough. Fig. 1 shows an example of a quantization process. Here, V_j is the amplitude of level j, and U_j and L_j are the upper and lower threshold values of level j, respectively.

Now assume that the amplitude of a signal, which varies in the range of uncertainty between U_j and L_j, is approximated to level j. In digital signal processing, the root mean square value of the quantization error is generally considered as a good index to compare different quantization processes. If the amplitude distribution of the signal is known, the root mean square value of the quantization error generated by quantizing the amplitude to a generic level j can be computed by using the following formula (1) [21]:

$$\bar{\varepsilon}_j = \sqrt{\int_{D_j} (v - V_j)^2 \, pd(v \mid Output = V_j) \, \mathrm{d}v}, \qquad (1)$$

where v is the amplitude of the signal, $D_j = [L_j, U_j]$ is the domain of the signal when the output of the quantization process is V_j, $pd(v|output = V_j)$ is the probability density function of v conditioned upon the event $output\ of\ quantization\ process = V_j$.

Suppose that over a long period of time all possible amplitude values appear the same number of times. Then, $pd(v \mid Output = V_j) = 1/(U_j - L_j)$. Formula (1) then becomes

$$\bar{\varepsilon}_j = \sqrt{\int_{L_j}^{U_j} (v - V_j)^2 \frac{1}{U_j - L_j} \, \mathrm{d}v} = \sqrt{\frac{1}{3 \cdot (U_j - L_j)} \left((U_j - V_j)^3 - (L_j - V_j)^3\right)}. \tag{2}$$

Let us denote the number of quantization levels and the maximum value of the signal as N and V, respectively. If the quantization levels are uniformly separated, then the distance d between two adjacent levels is equal to $V/(N-1)$. For each quantization level j, $U_j - L_j$ is equal to d (except for $j = 0$ and $j = N-1$, where $U_j - L_j$ is equal to $d/2$). Then formula (2) can be reduced to

$$\bar{\varepsilon}_j = \sqrt{\frac{1}{3}\left(\frac{d}{2}\right)^2} = \frac{V}{2(N-1)}\sqrt{\frac{1}{3}}. \tag{3}$$

It is clear from (3) that root mean square value of the quantization error decreases with the increase of the number of quantization levels.

The global root mean square value can be computed as

$$\bar{\varepsilon} = \sqrt{\sum_{j=0}^{N-1} \left(\bar{\varepsilon}_j^2 \, p(Output = V_j)\right)}, \tag{4}$$

where $p(Output = V_j)$ is the probability to have V_j as output of the quantization process.

In two-valued logic-based software development methods, high quantization errors arise from the fact that rules adopt only two quantization levels. For example, the rule *Candidate Class Identification* requires from the software engineer to decide whether an entity in a requirement specification is relevant or not. The software engineer may, however, perceive that an entity partially fulfills the relevance criterion, and may conclude that the entity is *substantially* relevant. Here, the quantization error is the difference between the perception of the software engineer and the "quantization levels" imposed by the two-valued logic-based methodological rules.

If the rule *Candidate Class Identification* is applied to many artifacts, then the quantization error can be considered as a random variable. Let us suppose that the relevance can assume values between 0 and 1. Further, assume that if the relevance value is between 0 and 1/2, and 1/2 and 1, it is approximated to 0 and 1, respectively. Fig. 2 shows the quantization error in determining the relevance of an entity. Here, the Y-axis represents the relevance of an entity and the X-axis indicates the entities being considered. If the distribution of the relevance value is known, the root mean square value of the quantization error generated by classifying the relevance to 0 or 1 can be computed by using formula (1). Assuming that the value of the relevance is uniformly distributed, we can use formula (3) and compute

$$\bar{\varepsilon} = \bar{\varepsilon}_j = \frac{V}{2(N-1)}\sqrt{\frac{1}{3}} = 0.289.$$

As methods are composed of a set of rules, we are particularly interested in calculating the quantization error that affects the result of rules. In calculating the quantization error, we suppose that the truth-value of a condition varies between 0 (*false*) and 1 (*true*), and the truth-value of

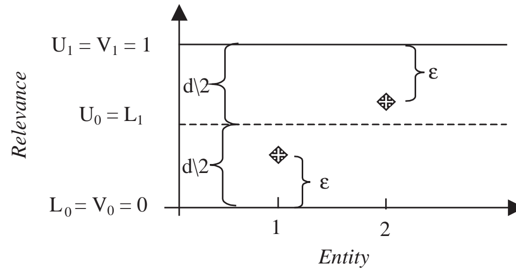

Fig. 2. Quantization error for Relevance.

the consequent increases linearly with the increase of the product among the truth values of the conditions. [2]

Consider a rule whose antecedent is composed of n conditions. Let c_i be the truth value of the condition i and let $pd(c_1, c_2, \ldots, c_n)$ be the joint probability density of $c_1, c_2, \ldots, c_n$. We can calculate the root mean square value of the quantization error when the result is approximated to V_j as

$$\bar{\varepsilon}_j = \sqrt{\int_{D_j} (\underline{c} - V_j)^2 \, pd(\underline{c \,|(Output = V_j)})\underline{dc}}, \tag{5}$$

where D_j indicates the domain on which the truth values of conditions vary when the result of the rule is V_j, and $\underline{c}$, $pd(\underline{c \mid Output = V_j})$ and $\underline{dc}$ denote $c_1 \cdot c_2 \cdot \ldots \cdot c_n$, $pd(c_1|(Output = V_j), c_2|(Output = V_j), \ldots, c_n|(Output = V_j))$ and $dc_1 \cdot dc_2 \cdot \ldots \cdot dc_n$, respectively.

Let us suppose that random variables $c_1, c_2, \ldots, c_n$ are independent. Then, $pd(\underline{c \mid (Output = V_j)}) = \prod_{i=1}^{n} pd(c_i|(Output = V_j))$.

We observe that for most rules the truth-values of conditions and consequent increase linearly with the increase of input values and result, respectively. Further, for simplicity, we assume that over a long period of time all possible truth-values of the conditions appear the same number of times. Then, $pd(c_i|(Output = V_j)) = 1/(U_j^i - L_j^i)$, where $U_j^i - L_j^i$ is the domain of the truth-value of condition i when the output is V_j.

In two-valued logic, the consequent of a rule is *true* if the conditions in the antecedent of the rule are *true*. This means that the truth-value of the consequent is approximated to 1 when all the conditions have the truth-values between $\frac{1}{2}$ and 1. In case $V_j = 1$, $D_1 = D_1^1 \times D_1^2 \times \cdots \times D_1^n$, where $D_1^i = [\frac{1}{2}, 1]$ is the domain of c_i when the output is 1. We can now compute $\bar{\varepsilon}_1$ by applying formula (5). Here, $pd(\underline{c|(Output = 1)}) = 2^n$. We obtain

$$\bar{\varepsilon}_1 = \sqrt{2^n \int_{1/2}^{1} \int_{1/2}^{1} \cdots \int_{1/2}^{1} (\underline{c} - 1)^2 \underline{dc}} = \sqrt{1 + \left(\tfrac{7}{12}\right)^n - 2\left(\tfrac{3}{4}\right)^n}. \tag{6}$$

Formula (6) shows that when the result is classified to 1 (*true*), the root mean square value of the quantization error increases with the increase of the number of conditions and approaches to 1

[2] Our experience with current software development methods shows that this assumption is valid for most methodological rules [1].

when $n \to \infty$. As an example, let us calculate the root mean square value of the quantization error, which affects the result of the rule *Candidate Class Identification* by using formula (6) when an entity is accepted as a candidate class. By substituting $n=2$ in formula (6), we obtain

$$\bar{\varepsilon}_1 = \sqrt{1 + \left(\tfrac{7}{12}\right)^2 - 2\left(\tfrac{3}{4}\right)^2} = 0.464.$$

It follows that, in the hypothesis of uniform distribution of the truth-values of conditions, each entity in the requirement specification identified as a candidate class in average matches only the 53.6 percent of the definition of an ideal candidate class.

Now, we calculate the quantization error in case $V_j = 0$. Theoretically, if the antecedent of a rule is *false*, it is not possible to infer that the consequent of the rule is *false*. It can be often appropriate to make use of the *closed-world assumption*. This assumption means that anything, which cannot be inferred as true from the given facts and the available rules, is *false*. We can easily compute $\bar{\varepsilon}_0$ from formula (5) by considering $D_0 = D - D_1$, where $D = D^1 \times D^2 \times \cdots \times D^n$ and $D^i = [0,1]$. Thus,

$$\bar{\varepsilon}_0 = \sqrt{\int_D \underline{c}^2\, pd(\underline{c} \mid Output = 0)\, d\underline{c} - \int_{D_1} \underline{c}^2\, pd(\underline{c} \mid Output = 0)\, d\underline{c}} = \sqrt{\frac{1}{3^n} \cdot \frac{1 - (\frac{7}{8})^n}{1 - (\frac{1}{2})^n}} \qquad (7)$$

as $pd(\underline{c} \mid Output = 0) = 1/(1 - 1/2^n)$.

Formula (7) shows that, when the result is classified to 0 (*false*), the root mean square value of the quantization error decreases with the increase of the number of conditions in a rule and approaches to 0 when $n \to \infty$.

As an example, let us calculate the root mean square value of the quantization error, which affects the result of the rule *Candidate Class Identification* when an entity is discarded as a candidate class. By substituting $n=2$ in formula (7), we obtain $\bar{\varepsilon}_0 = 0.186$.

Now, we can calculate the global root mean square value of the quantization error, which affects the result of a rule by applying formula (4):

$$\bar{\varepsilon} = \sqrt{\bar{\varepsilon}_1^2 p(Output = 1) + \bar{\varepsilon}_0^2 p(Output = 0)} = \sqrt{\frac{1 - \left(\frac{7}{8}\right)^n}{3^n} + \frac{1 + \left(\frac{7}{12}\right)^n - 2\left(\frac{3}{4}\right)^n}{2^n}}. \qquad (8)$$

For instance, the root mean square value of the quantization error, which affects the result of the rule *Candidate Class Identification*, is $\bar{\varepsilon} = 0.283$.

Also coupling of rules with multiple conditions affects the quantization error. Consider, for example, the rules $Rule_1$ and $Rule_2$ whose antecedents are composed by n and m conditions, respectively. Suppose that $Rule_1$ is coupled to $Rule_2$, i.e., the consequent of $Rule_1$ matches a condition of $Rule_2$. This is like having the antecedent of the rule $Rule_2$ composed by $n + m - 1$ conditions. The root mean square value of the quantization error can still be computed using formulas (6)–(8).

2.3. Evaluation of the accuracy of the formulas

In this section we will evaluate the accuracy of the quantization error formulas from the following five perspectives: effect of quantization errors, validity of the formulas, interpretation of the formulas, effect of quantization policy and possible use of metrics.

2.3.1. Effect of quantization errors

The formulas presented in the previous section determine the root mean square value of the quantization error. This does not mean that the resulting object-oriented model will have the same percentage of error. The measurement of error in the resulting object model requires detailed semantic analysis of the requirement specification and the object model. [3] It is, however, expected that loss of information will eventually cause errors in the resulting object-oriented model.

2.3.2. Validity of the formulas

Formulas (4) and (5) are based on some general assumptions and therefore they are applicable to a large category of rules. Validity of formulas (6)–(8) depends on the assumptions made about the relations between the input values and the truth-values of the conditions, and the output value and the truth-value of the consequent. In addition, assumptions made about the uniform distribution of the input values also affect these formulas. Obviously, other distribution functions different from the uniform distribution could be used leading to different instantiations of formulas (4) and (5). Further, since formulas are based on statistical assumptions, their validity increases after a long run and for applications with a large number of entities. We believe that the formulation of the quantization error is particularly useful in analysing, comparing and for relatively improving methods. For this purpose, within the objective of this article, we consider these assumptions to be acceptable.

2.3.3. Interpretation of the formulas

If the coupling between rules is through the accepted subsets, the root mean square value of the quantization error can be computed by formula (6). This formula shows that the error value increases with the increase in the number of couplings between rules whose antecedents are composed by two or more conditions and approaches to 1 when this number approaches to ∞. If the coupling occurs always through the rejected subsets, the error value can be computed by formula (7). This formula shows that the error value decreases with the increase in the number of couplings between rules whose antecedents are composed by two or more conditions and approaches to 0 when this number approaches to ∞.

In most methods, models are created by classifying artifacts in accepted sets. In this case, the quantization error will increase with the number of couplings. If all the rules reject the entities, then the error will approach to zero, but there will be no model in the end. This shows that it is not meaningful to define object-oriented methods solely based on the quantization error calculations. The objectives of the design must be considered as the main goal. To this aim, methods have to be derived from the software engineer's perception and experience. However, design methods can be improved by reorganizing design rules. For example, it may be better to defer the elimination of an artifact until all the relevant information is collected. In such a way, the elimination is affected by a lower quantization error than an earlier elimination. Another observation is that classifying an entity to a rejected set does not only mean to classify that entity to the level 0, but generally it also means discarding that entity. As a result, the discarded entity cannot be considered anymore by the subsequent rules.

[3] Formal semantic analysis is not the aim of this paper.

2.3.4. Effect of quantization policy

In the quantization process of a sampled signal, if a value is between two quantization levels, a quantization policy must be adopted to classify the signal into one of the adjacent levels. If the difference between these levels is too large, then the policy becomes more crucial to the quality of the quantization process. Since two-valued logic-based object-oriented methods have two quantization levels, the quantization policy should be as accurate and precise as possible.

The quantization policy of a rule is determined by the conditions of the rule, which establish a threshold value to quantize object-oriented artifacts into accepted or rejected sets. It is not always easy to define an ideal threshold value especially in the early phases of software development. In addition, during the interpretation of conditions, the software engineer may be requested to provide some information, which is already quantized. For instance, in the rule *Candidate Class Identification*, the condition "If an entity in a requirement specification is relevant" requires from the software engineer to quantize the relevance of an entity as relevant or irrelevant. The relevance of an entity depends on the intuition and experience of the software engineer. Since both the threshold value and the response of the software engineer may be very subjective, it is a difficult task to define a very accurate and precise quantization policy. In practice, the quantization policy of a rule can be improved by defining fine-grained dedicated rules for the targeted application domain.

2.3.5. Possible use of metrics

Using metrics in the conditions of rules may eliminate the necessity to have input quantized by the software engineer. Most object-oriented metrics aim to measure the artifacts of an object-oriented development process in an objective way. For example, in [6], a number of metrics are defined to quantify a set of design artifacts such as *class coupling* and *operation inheritance*. In general, the metrics of an artifact are computed by using a formula and result in numbers. A rule then must establish a threshold value to determine whether the measured artifact is acceptable or not. The quality of metrics and of the threshold value determines the quality of a quantization policy. A satisfactory metrics-based quantization policy may be too difficult or even impossible to determine. It is very hard to formulate accurate and precise metrics especially for the early phases of software development. Further, even if a satisfactory metrics-based quantization policy exists, the result of a rule remains a two-level quantization.

2.4. Possible effects caused by the quantization error

One of the most dramatic effects caused by the quantization error on the development process is *the early elimination of artifacts*. Each decision taken by a rule is based on the available information up to that phase. For the early phases, there may not be sufficient amount of information available to take abrupt decisions like discarding an entity. Such an abrupt decision must be taken only if there is sufficient evidence that the entity is indeed irrelevant. In most object-oriented methods, however, each identification process is followed by an elimination process. For example, the OMT method [23] proposes a process that includes class identification and elimination, association identification and elimination, and so on. Now, assume that a software engineer discards an entity in the requirement specification because it is considered non-relevant. Let us also suppose that this decision was based on partial information and the discarded entity could have been included as a candidate class, if the software engineer had gathered more information about the structure and operations of that entity.

During the later phases this would be practically impossible because the discarded entity could not be considered further. Early elimination of artifacts in current methods is practically inevitable.

If, at the end of the development process, the software engineer realizes that the resulting object model is not satisfactory, there are two possible options: improving the model by applying subsequent rules and/or by iterating the process. The application of subsequent rules may not adequately improve the model because of the loss of information due to quantization errors. The iteration of the process still suffers from the quantization error problem. Moreover, managing iterations is generally considered as a difficult task.

3. The contextual bias problem

The soundness of the result inferred by applying a rule depends on the validity of the rule and on the accuracy of inputs provided to the rule.

Validity of a rule may be affected by contextual factors such as application domain, changes in user's interest and technological advances. Let us consider, for instance, the rule *Design Modification*, which is extracted from the Coupling Between Object Classes (CBO) metric introduced in [6].

IF the number of classes coupled to a class is larger than τ, **THEN** the design is not modular.

If this rule concludes that the design under evaluation is not modular, then the design should be modified to achieve better reuse and maintenance. An object is said to be coupled to another object if its methods use the methods or instance variables of that object. As objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. Excessive coupling between object classes is detrimental to modular design, prevents reuse and makes maintenance difficult. The success of this rule depends on the choice of the threshold τ. As observed in [2], metrics must be associated with some interpretation to determine the threshold of a rule. For the rule *Design Modification*, it is well known that the coupling between classes is an increasing function of the number of classes in the application [6]. Further, classes responsible for managing interfaces have high CBO values. Thus, the correct choice of τ depends both on the type of application, which determines the number of classes, and on the specific function of the class to which the rule is applied. To obtain valid results, the relation between the threshold τ and the contextual factors has to be correctly modelled. Further, the transition from the number of coupled classes, which identifies a modular design, to the number, which does not, appears gradual rather than abrupt. For example, if we fix $\tau = 10$, a design should be considered modular if the number of coupled classes in that design is 10 or less, and not modular if the number of coupled classes is 11 or higher. The difference of one coupling, however, does not seem to be a discriminating difference.

Results of rules are also affected by the accuracy of input values. Inputs to a large amount of methodological rules, especially in the early phases of the development process, require subjective evaluations from software engineers. The software engineers' experience and skill can affect the accuracy of the inputs. Consider the rule *Candidate Class Identification*. Here, the selection of an entity as a candidate class is based on the software engineer's perception of relevance and autonomy. This perception can be different from software engineer to software engineer and a very

small difference in perception can generate conflicting results. For example, let us assume that the same entity is being considered by two software engineers and one perceives the entity as *slightly* relevant and the other as *substantially* relevant. In current methods, due to the quantization process described in Section 2, it is likely that the first software engineer would quantize the entity as irrelevant and the second one as relevant. Thus, the rule would reject and accept the entity as a candidate class for the first and second software engineer, respectively. Here, the software engineer's perception is the contextual factor which affects the accuracy of the input. As current methods use only two quantization levels, the effects of the different perceptions are exalted in such a way that they may generate contradictory results.

4. Fuzzy logic-based methods

4.1. Introduction

In [19,20], we have showed that artifact types are naturally modelled by graded categories. This means that they show a fuzzy boundary, which can be defined by a linear scale of values between 0 and 1, with 1 at the interior and 0 at the exterior. As a consequence, each artifact can be an instance of more than one artifact type with different grades. For instance, an entity can be an instance of artifact types Attribute and Class at the same time with 0.7 and 0.3 membership degrees, respectively. Each methodological rule determines the membership degree of an artifact with respect to some artifact type. For instance, the rule *Candidate Class Identification* identifies at which extent an entity in the requirement specification is an instance of artifact type Candidate Class. The membership degree depends on the value of the conditions. For example, let us assume Relevance and Autonomy vary in the interval [0,1]. The heuristic, which the rule *Candidate Class Identification* is based on, suggests that the more an entity is relevant and autonomous, the more this entity is a candidate class. This leads, for instance, to compute the membership value to Candidate Class as the product of the values of Relevance and Autonomy. We think, however, that software engineers cannot easily provide numeric input values for this rule. For properties as Relevance and Autonomy, software engineers may better express qualitative evaluations such as *weakly relevant* or *partially dependent*.

As well known in the literature, fuzzy logic provides means to express qualitative, vague and uncertain expressions, and to reason on these expressions [9,28]. Properties of artifacts can be expressed in terms of linguistic variables. A *linguistic variable* is a variable whose values, called *linguistic values*, have the form of phrases or sentences in a natural or artificial language. Syntactically, a linguistic value is a composition of the following atomic terms:

1. *primary terms*, which are labels of specified fuzzy sets in the universe discourse;
2. negation *not* and connectives *and* and *or*;
3. *markers* such as *parentheses*. [4]

[4] For the sake of simplicity, we do not consider modifiers in the atomic terms.

All possible values of a linguistic variable can be generated by a context-free grammar $G=(T,N,P)$, where T and N are the terminal and non-terminal symbols, respectively, and P is the production system. The terminal symbols are the atomic terms. The meaning associated with each possible linguistic value is determined by a semantic rule R, which maps each linguistic expression into an operation on fuzzy sets. Each linguistic variable is characterized by a quintuple $(x, TN(x), U, G, R)$, where x is the name of the variable, $TN(x)$ is the term set of x, that is, the union of terminal and non-terminal symbols of x with each value being a fuzzy set defined on universe U, G is the context free grammar for generating the symbols of x, and R is the semantic rule. The definition of G and R is shared among all linguistic variables except for the primary terms and their meanings. In general, a linguistic variable is completely characterized by defining the universe, the primary terms and their meanings.

To make the interaction between a software engineer and the method as friendly as possible, it is crucial to investigate how many and which primary terms would be meaningful for these linguistic variables. To this aim, we have adopted the following method: we have selected a pool of software engineers [5] and asked them to define the linguistic variables. Then, we have stimulated a revision process within the pool aimed at reaching an agreement. For instance, the pool has concluded that Relevance can be expressed as *weakly*, *slightly*, *fairly*, *substantially* and *strongly relevant* and Autonomy as *dependent*, *partially dependent* and *fully autonomous*. The meaning of the primary terms of Relevance and Autonomy are shown in Figs. 3 and 4. Here, standard piecewise quadratic functions are used to define membership functions. The universes are supposed to vary from 0 to 1. Through the presented method, it is possible to adjust the linguistic values with respect to the software engineer's perception. In addition, the possibility of expressing fine-grained input values increases the number of quantization levels and therefore decreases the quantization error.

4.2. Formulation of fuzzy rules

Linguistic variables allow expressing methodological rules in a natural way and the meaning associated with each linguistic value permits to reason on these rules [28]. A fuzzy rule is typically expressed as **IF** X_1 is A_1 **AND** $\cdots$ **AND** X_N is A_N **THEN** Y is B, or for short $I(A_1 \wedge \cdots \wedge A_N, B)$, where X_i, with $i=1..N$, and Y are linguistic variables defined on the universes U_i and V, respectively, A_i and B are linguistic values of X_i and V, respectively, and I is a fuzzy implication operator. The connective **AND** and the fuzzy implication are implemented as fuzzy relations. For the connective **AND**, $A_1 \wedge \cdots \wedge A_N = \int_{U_1\times\cdots\times U_N} T(\mu_{A_1},\ldots,\mu_{A_N})/u_1\ldots u_N$, with T a triangular norm and μ_{A_i} the membership function associated with the primary term A_i. A *fuzzy implication* is defined for all $t\in T$ and $v\in V$ by $I(A,B)=\int_{T\times V} F(\mu_A(t),\mu_B(v))/(t,v)$, where F may be each function from $[0,1]\times[0,1]$ to $[0,1]$ that satisfies the boundary conditions $F(0,0)=F(0,1)=F(1,1)=1$ and $F(1,0)=0$. Several families of fuzzy implication operators have been proposed in literature. Comparative studies can be found in [15].

[5] A group of students have played the role of software engineers. Some of the experiments have been published in [25].

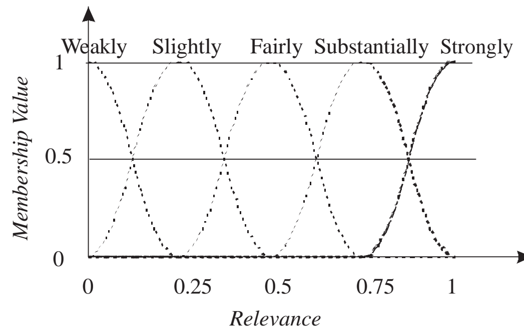

Fig. 3. Linguistic variable Relevance.

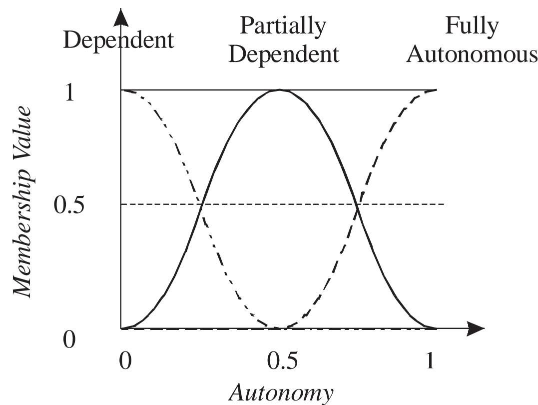

Fig. 4. Linguistic variable Autonomy.

Consider, for example, the fuzzy logic version of the rule *Candidate Class Identification*:

IF an entity in a requirement specification is *relevance value* relevant **AND** can exist *autonomy value* in the application domain, **THEN** it is *membership value* a candidate class.

Here, an entity and a candidate class are the artifact types to be reasoned, Relevance and Autonomy are the properties of artifact type entity, *relevance value* and *autonomy value* indicate the domains of these properties, and *membership value* denotes the domain of membership grades of an entity to the artifact type Candidate Class. These membership grades are expressed by means of linguistic values. The pool of software engineers have agreed on the following primary terms: *weakly*, *slightly*, *fairly*, *substantially*, *strongly*. The meaning of these primary terms is analogous to the one shown in Fig. 3. Each combination of relevance and autonomy values of an entity has to be mapped into one of the five membership values to Candidate Class. The mapping is based on the intuition of what a candidate class is. The more an entity is relevant and autonomous, the more the entity is a candidate class. We adopt a tabular representation. The resulting 15 *sub-rules* are shown in Table 1. Each element of the table, shown in italics, represents the output value of a sub-rule, which is the relevance value of the candidate class being considered. For example, if the relevance and autonomy values of an entity are, respectively, *strongly* and *fully autonomous*, then the candidate class relevance value is *strongly*. Similarly, methodological rules used in current object-oriented methods can be

Table 1
Sub-rules of rule Candidate Class Identification

	Entity: Autonomy		
Candidate class: Membership	Dependent	Partially dependent	Fully autonomous
Weakly	Weakly	Weakly	Weakly
Slightly	Weakly	Slightly	Slightly
Fairly	Weakly	Slightly	Fairly
Substantially	Weakly	Fairly	Substantially
Strongly	Slightly	Fairly	Strongly
Entity: Relevance			

transformed in more intuitive fuzzy logic-based rules. We believe that this transformation is quite natural since software engineers' intuitions are generally fuzzy rather than crisp. Our experimentation in transforming methodological rules of well-known object-oriented methods confirms our claim [1,19].

Given a fuzzy rule **IF** X_1 is A_1 **AND** $\cdots$ **AND** X_N is A_N **THEN** Y is B and a fact X_1 is $\hat{A}_1$ **AND** $\cdots$ **AND** X_N is $\hat{A}_N$, the inference mechanism used to infer a conclusion B' is normally implemented by a generalization of the modus ponens, called *generalized modus ponens* or *compositional rule of inference*. Conclusion B' is computed as $B' = (A'_1 \wedge \cdots \wedge A'_N) \circ I(A_1 \wedge \cdots \wedge A_N, B)$, where the character "$\circ$" denotes the composition operator and I a fuzzy implication operator [27]. The conclusion B' is therefore obtained by first computing the fuzzy sets corresponding to the fact and to the rules, and then composing these fuzzy sets by the composition operator.

A fact can activate more than one fuzzy rule. For example, when the software engineer inputs the relevance and the autonomy values of an entity, these input values activate all the sub-rules of the rule *Candidate Class Identification*. The conclusion inferred by applying the rule will be obtained as aggregation of the conclusions obtained by inferring the sub-rules separately. [6]

Aggregation is generally implemented as a fuzzy intersection or union. The choice of the type of aggregation operation depends on the type of fuzzy implication and composition operators. In general, the criterion adopted in the choice is the fundamental requirement for fuzzy reasoning, i.e., given a fact that matches the antecedent of a rule, the conclusion has to match the consequent of that rule. A detailed analysis on the relationships among types of aggregation, implication and composition operators, and partitions of the input and output spaces for satisfying the fundamental requirement for fuzzy reasoning can be found in [16]. Here, we would like to point out that the partitions shown in Figs. 3 and 4 are suitable partitions. If a software engineer inputs primary terms, the conclusion inferred from the sub-rules is a primary term in its turn. In this case, no computation is needed, but the conclusion can be obtained by using a look-up table. Searching for a label in a small table rather than computing the generalized modus ponens speeds up dramatically the approximate reasoning process. This improvement in performance can be applied to a large amount of rules, particularly in the first phases of the development process when software engineers

[6] For the sake of simplicity, in our system we adopt only the "first infer then aggregate" (FITA) strategy, that is, first we infer a conclusion from each sub-rule, then we aggregate all the conclusions.

typically express input values using primary terms of linguistic variables. For instance, for the rule *Candidate Class Identification*, if the software engineer decides that an entity is strongly relevant and fully autonomous, then the entity can be directly identified as strongly a candidate class.

The linguistic values, which express the membership of an artifact to an artifact type, can be considered as measures of how much an artifact is actually an instance of an artifact type. These measures are particularly useful in selecting the best alternative within a set of possible conflicting artifact types. Two artifact types are conflicting if an artifact cannot be instance of both in the design model [20]. For instance, an entity in the requirement specification cannot be instance of both Class and Attribute. After applying the rules *Candidate Class Identification* and *Candidate Attribute Identification*, the entity could be considered *slightly* a candidate class and *substantially* a candidate attribute. If the resulting model should be released, the entity would be implemented as attribute. Indeed, *substantially* is intuitively larger than *slightly*. In this case, the evaluation of the measures is quite simple.

When a method is applied, several different rules can contribute to determine the membership value of an artifact with respect to an artifact type. This value could correspond to a generic fuzzy set rather than to a linguistic value. In this case, the choice cannot be based on the orderings defined by the meanings associated with the linguistic values. To be able to take a decision, fuzzy sets have to be approximated to crisp values by applying a defuzzification strategy. The defuzzified values are therefore compared and the artifact is instantiated to the artifact type with the highest value. At present, the commonly used strategies are the *mean of maxima* and the *center of area*. The crisp value produced by the mean of maxima strategy represents the mean value of the elements, which belong to the fuzzy set characterizing the conclusion with maximum grade. The center of area strategy produces the center of gravity of the fuzzy set characterizing the conclusion.

In this section, we have presented only one example of a transformation from a heuristic rule expressed in two-valued logic to a rule expressed in fuzzy logic. As the aim of this article is not to introduce a new method, but to formulate and discuss the quantization error and the contextual bias problems, we consider that this example is sufficient. The interested reader can find examples of fuzzy methods in [1,20], and an application of these methods in [1].

5. The quantization error problem in fuzzy logic-based methods

A software engineer can provide both linguistic values and crisp values as input to the methodological rules. In the early phases of the development process, for example when the rule *Candidate Class Identification* is applied, linguistic values are considered more appropriate. In the later phases, for instance when the rule *Design Modification* is applied, crisp values are preferred. For this reason, in the following we will discuss the quantization error in both cases.

5.1. The quantization error in case of linguistic values as input

Consider the membership functions defined by Fig. 3. Here, 0.125, 0.375, 0.625 and 0.875 correspond to the X-axis values of the crossing points of the membership functions. If the software engineer selects a linguistic value such as *slightly*, the actual value can be at any point defined by the membership function *slightly*. To simplify the formulation of the quantization error, however, we

make the following assumption: The software engineer's perception is likely to be restricted to the values between the crossing points. For example, if the software engineer assumes that the actual value is less than 0.125, probably he or she would have used *weakly* instead of *slightly*. Therefore, we assume that the software engineer mentally limits the fuzzy set to the range of values at which the membership function associated with the fuzzy set takes higher values than the other membership functions. This means, for example, that *slightly* is considered to be between 0.125 and 0.375. We would like to stress that this assumption is only taken to formulate the quantization error, and during the application of the fuzzy rules, no such restriction is applied.

To formulate the quantization error for more than 2 quantization levels, we have to determine the threshold values and the amplitudes of the quantization levels. Now, assume that the crossing points are the threshold values. The amplitude of a quantization level can be considered as the defuzzified value of the corresponding fuzzy set. Referring to Fig. 3, if the *mean of maxima* defuzzification strategy is used, then the quantization levels will be 0, 0.25, 0.5, 0.75 and 1. Assuming over a long period of time all possible X values appear the same number of times, by using formula (3) of Section 2, with $V=1$ and $N=5$, the mean value of the quantization error is computed as 0.072.

Now, we calculate the quantization error that affects the result of a fuzzy rule. In case of two-valued logic, it is possible to define formulas such as (6)–(8) for rules with n conditions. In fact, the truth-values of all the conditions have two levels, one at 0 and the other at 1, and a threshold can be assumed to be at 0.5. Further, each combination of the truth-values of the conditions, which generate the result 0 or 1, is predefined.

In case of fuzzy logic, however, the truth-values of conditions can have a different number of levels or levels positioned at different amplitudes. Further, each level in the result of a fuzzy rule can be generated by many different combinations that are not predefined. For this reason, although formulas (4) and (5) continue to be valid, it is not possible to define general formulas as (6)–(8) for fuzzy rules with n conditions. Nevertheless, if the definition of a fuzzy rule is "precisely" known, the quantization error of the rule can be computed.

By taking these considerations into account, we will compute the root mean square value of the quantization error only for the fuzzy rule *Candidate Class Identification* as defined by Table 1. Firstly, we will compute the quantization error for each output level. Then, by using formula (4) we will calculate the global root square mean error. Finally, we will compare this result with the quantization error calculated in Section 2.

In general, each output level can be the result of applying more than a single rule. The root mean square value of a quantization error in case of applying R rules can be computed by using the formula

$$\bar{\varepsilon}_j = \sqrt{\frac{\sum_{i=1}^{R} p(r_{j,i})\bar{\varepsilon}_{j,i}^2}{\sum_{i=1}^{R} p(r_{j,i})}}, \tag{9}$$

where $p(r_{j,i})$ is the probability to apply rule $r_{j,i}$ and $\bar{\varepsilon}_{j,i}$ is the root mean square value of the quantization error generated by classifying the result of the rule i to the level j. The probability $p(r_{j,i})$ is computed by using the following formula:

$$p(r_{j,i}) = \prod_{c=1}^{C_{j,i}} \left(\int_{L_c}^{U_c} pd(c_c)dc \right), \tag{10}$$

where $pd(c_c)$ is the probability density of the truth value c_c of the rule and $C_{j,i}$ is the number of conditions of rule $r_{j,i}$.

The value $\bar{\varepsilon}_{j,i}$ is computed by formula (5) where V_j is the value of the level j of the result of the rule. Let us assume the same hypotheses as in case of 2 quantization levels: (i) the random variables which represent the truth values of the conditions are independent, (ii) the truth values of conditions and consequent increase linearly with the increase of input values and result, respectively, and (iii) over a long period of time all possible truth values of the conditions appear the same number of times.

As an example, we compute $\bar{\varepsilon}_{j,i}$ and $p(r_{j,i})$ for the following sub-rule $\bar{i}$:

IF an entity in a requirement specification is *strongly* relevant **AND** can exist *fully autonomous* in the application domain, **THEN** it is *strongly* a candidate class.

The quantization levels of the result are defined in the same way as the ones of Fig. 3. This means that *strongly* corresponds to the quantization of the result to 1. We apply formula (5) with $D_j = [0.875, 1] \times [0.75, 1]$, $V_j = 1$, $pd(c_1|(Output = 1)) = \frac{1}{(1-0.875)}$ and $pd(c_2|(Output = 1)) = \frac{1}{(1-0.75)}$. We obtain $\varepsilon_{4,\bar{i}} = 0.195$.

As the probability density of the truth values is uniform in the interval $[0, 1]$, $pd(c_1) = pd(c_2) = 1$ and $p(r_{4,\bar{i}}) = (1 - 0.875) \cdot (1 - 0.75) = 0.031$.

By computing $\bar{\varepsilon}_{j,i}$ and $p(r_{j,i})$ in the same way for all the sub-rules in Table 1, we obtain the following root mean square values of the quantization error when the result is classified to the five levels: $\bar{\varepsilon}_0 = 0.066$, $\bar{\varepsilon}_1 = 0.11$, $\bar{\varepsilon}_2 = 0.147$, $\bar{\varepsilon}_3 = 0.125$ and $\bar{\varepsilon}_4 = 0.195$, where levels $0, 1, 2, 3$ and 4 correspond to *weakly*, *slightly*, *fairly*, *substantially* and *strongly*.

By applying formula (4), we obtain $\bar{\varepsilon} = 0.114$. In case of two quantization levels, $\bar{\varepsilon}$ was computed as 0.283. This demonstrates that compared to two-valued logic, fuzzy logic-based rules reduce loss of information considerably.

5.2. The quantization error in case of crisp values as input

Instead of linguistic values, the software engineer may also provide crisp values as input. A crisp value indicates a single point at the X-axis: this means that the input is determined with certainty. Providing a crisp value, however, may be very difficult especially for the early phases of software development process. Even though the software engineer may be allowed to select a crisp value such as the entity in the requirement specification is 0.3 relevant, the reliability of this value is highly questionable. On the other hand, inputs to some rules can be estimated with certainty. For example, consider the rule *Design Modification* introduced in Section 3. This rule can be better expressed by removing the abrupt transition from the number of coupled classes, which identifies a modular design, to the number, which does not. This can be obtained by transforming the two-valued logic rule into the following fuzzy rule:

IF the number of classes coupled to a class is *number of coupled classes*, **THEN** the modularity of the design is *modularity value*.

Table 2 shows the three intuitive sub-rules derived from this rule. The software engineer can precisely determine the crisp input value of the linguistic variable Number of Coupled Classes.

Table 2
Sub-rules of rule Design Modification

	Class: Number of coupled classes		
	Low	Medium	High
Design: Modularity	High	Medium	Low

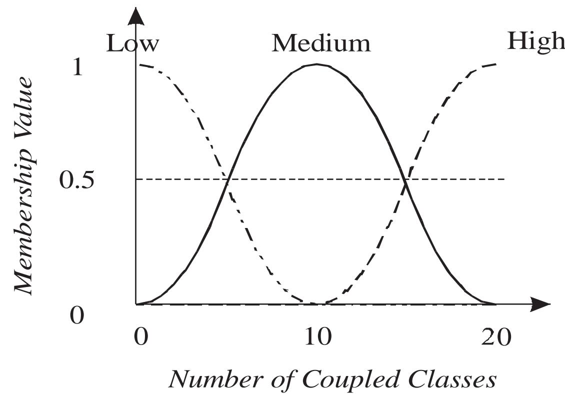

Fig. 5. Linguistic variable Number of coupled classes.

Having a crisp value as input makes it possible to determine the membership degree of the crisp value with respect to the fuzzy set associated with each linguistic value. Consider, for example, Fig. 5 where the meaning of the three linguistic values *low*, *medium* and *high* of the linguistic variable Number of Coupled Classes is defined. [7] If the software engineer provides 5 as an input value, then the membership degree with respect to linguistic values *low* and *medium* is 0.5 and with respect to *high* is 0. Knowing the membership degree is similar to knowing the distance between the actual signal and the amplitude of the quantization level. Practically, this means that the quantization error introduced by the rule can be negligible. This is considered as an additional advantage of using fuzzy sets with respect to disjoint sets.

5.3. Evaluation of the accuracy of the error calculations in fuzzy logic-based methods

In Section 2.3, the accuracy of the quantization error calculations for two-valued logic-based rules was analyzed with respect to the effect of quantization errors, the validity and the interpretation of the formulas, the effect of the quantization policy, and the possible use of metrics. All these evaluations are also valid for the fuzzy logic-based method. In addition to these, the accuracy of the formulas for fuzzy logic-based rules depends on the accuracy of the definitions of the membership functions, the assumptions about the threshold values used in the quantization process and quantization levels, the

[7] In defining these linguistic values we referred to the data collected as Site A in [6].

definition of the fuzzy connectives **AND** and **OR**, the fuzzy implication and composition operators, and aggregation operation [16].

Similar to two-valued logic-based rules, it is not meaningful to define fuzzy logic-based rules for the purpose of reducing quantization errors only. For example, when applying the fuzzy rule *Candidate Class Identification*, in case an entity is *strongly relevant* and *fully autonomous*, selecting that entity as a *substantially relevant* candidate class would have created a lower quantization error than selecting it as a *strongly relevant* candidate class. Experienced software engineers, however, would select the output value as *strongly* when both input values have their highest possible values. Concluding, we consider the formulation of the quantization error mainly useful in analyzing, comparing and for relatively improving methods.

5.4. Advantages in applying fuzzy logic-based methodological rules

In the previous sections, by formalizing the quantization error in methods, we have shown that the fuzzy logic-based method creates less quantization error than the two-valued logic-based approach. In practice creating less quantization error has the following advantages.

First, if fuzzy logic-based rules are applied, none of the artifacts is theoretically eliminated but each artifact is accepted with different acceptance levels. The fuzzy logic-based method can be considered as a *learning process*; a new aspect of the problem being considered is learned after the application of each rule [19]. Obviously, a new aspect can modify the previously gathered property values. The fuzzy logic theory provides techniques to reason and compose the results of the rules. Clearly, software development through learning creates very adaptable and reusable design models.

Further, fuzzy logic allows managing a number of design alternatives and associating a measure with each alternative [20]. Consider, for instance, that during an object-oriented development process, the software engineer judges an entity as a *substantial* candidate class and a *slight* candidate attribute. The concepts of class and attribute are considered as conflicting in object-oriented paradigm. During the overall development process these conflicting alternatives can be maintained, so reducing the loss of information and increasing the quality of the development process. When the final product has to be released, conflicts have to be solved. The meanings univocally associated with the linguistic expressions provide a valid support to conflict resolution. Each meaning can be considered as a measure of each alternative. Conflict resolution can be therefore reduced to select the alternatives with the maximum defuzzified value.

6. The contextual bias problem in fuzzy logic-based methods

As explained in Section 3, contextual factors can affect the soundness of the results of the application of the methodological rules by influencing both the accuracy of the inputs provided to the rules and the validity of the rules themselves. The reduction of the effects on the accuracy of the inputs is obtained as side effect of the diminution of the quantization error. As an example, consider the rule *Candidate Class Identification*. In Section 3, we have presented that a little difference in perception can cause contradictory results. We claimed that if two software engineers consider an entity in the requirement specifications as *slightly* and *substantially relevant*, it is likely that the one would reject and the other would accept the entity as a candidate class. By increasing the number

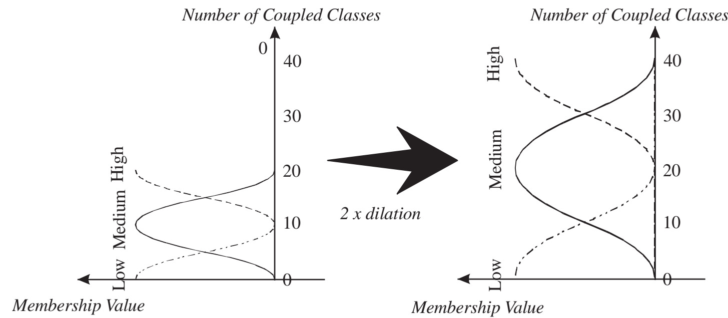

Fig. 6. Adapting to context through dilation.

of quantization levels, software engineers can express their perception more precisely and a possible difference between the input values caused by contextual factors is not exalted.

Modelling the influence of the context can explicitly reduce the effect of contextual factors on the validity of a rule. Let us consider the rule *Design Modification* as defined in Section 3. The validity of this rule depends on the value of the threshold τ. As pointed out in [6], τ increases with the number of classes. Further, τ depends on the kind of class being reasoned. Consider the fuzzy rule *Design Modification* as introduced in Section 5.2. The validity of this rule depends on the meaning associated with the linguistic values *low*, *medium* and *high* of linguistic variable Number of Coupled Classes. This meaning has to be adapted with respect to the number of classes composing the application and the kind of class under consideration. For instance, when the application is composed of a high number of classes, the membership functions associated with the primary terms of Number of Coupled Classes should be dilated so that a larger number of coupled classes can be considered acceptable. The dilation factor depends on the number of classes in the application. Fig. 6 shows a linear dilation function with factor 2.

In general, each type of adaptation can be obtained by translating, compressing and dilating the membership functions associated with each primary term. This adaptation process can be carried out by defining the membership functions on a reference universe of discourse. Then, translation operation is attained by shifting the reference universe along the X-axis; compression and dilation of fuzzy sets are achieved by reducing or expanding the reference universe of discourse. Translation, compression or dilation factors depend on the context. In general, it is difficult to formalize this dependence by analytical functions and therefore heuristic rules have to be adopted. Since rules defining the effect of contextual factors are typically expressed in terms of linguistic expressions, fuzzy logic seems to be appropriate for implementing these rules. The following couple of rules, for instance, can be used to determine the dilation factor used to adapt the meaning of the primary terms of the linguistic variable Number of Coupled Classes with respect to the number of classes in the application and to the type of class being considered:

Rule DL1:

IF the number of classes is *number of classes value*, **THEN** the dilation factor is *dilation factor value*.

Rule DL2:

IF the class is *certainty value* an interface class, **THEN** the dilation factor is *dilation factor value*.

Table 3
Sub-rules of rule DL1

	Class: Number of classes		
	Low	Medium	High
Dilation: Factor	Low	Medium	High

Table 4
Sub-rules of rule DL2

	Class type: certainty		
	Doubtfully	Approximately	Certainly
Dilation: Factor	Low	Medium	High

Here, the primary terms of the linguistic variables Number of Classes and Dilation Factor are *low*, *medium* and *high*; the primary terms of the linguistic variable Certainty are *doubtfully*, *approximately* and *certainly*. Tables 3 and 4 define the two rules, respectively. The dilation factor is obtained by defuzzifying the conclusion obtained by applying the two rules and by aggregating the results.

7. Related work

To the best of our knowledge, only a very few publications have combined fuzzy techniques with software models and methods. Concerning object-oriented models, fuzzy theory has been applied to the notion of objects to create the so-called fuzzy objects [11,26]. Fuzzy objects can have attributes that contain fuzzy sets as values and there may be a *partial inheritance* relation between classes. For example, class ToyVehicle can be defined to inherit the property Cost with a grade of 0.9 from class Toy and with a grade of 0.3 from class Vehicle. In class ToyVehicle the property Cost is initialized to *Low*, whereas in class Vehicle it is initialized to *High*. By using the fuzzy union of the fuzzy sets determined by *Low* and *High*, the value of property Cost of class ToyCar can be obtained. Fuzzy objects have also proved to be very useful to deal with uncertain information in relational databases [26]. Our approach focuses on applying fuzzy logic to software methods for reducing the quantization error, and enhancing adaptability and reusability of design models. The previous related work mainly focuses on applying fuzzy theory to object modelling.

Benedicenti et al. [3] exploit fuzzy logic for coping with unreliable data in a business process modelling method. Fuzzy logic is employed to attribute resources to activities, determine activity cost driver and resource (per activity) cost driver.

Liu and Yen [17] propose a systematic approach for specifying and analysing imprecise requirements. The constraint imposed by an imprecise requirement R is represented as a satisfaction function that maps an element of R's domain D to a number in [0,1]. In practice, the satisfaction function defines a fuzzy subset of D that satisfies the imprecise requirement. The different impact of satisfying a requirement on the satisfaction degree of another requirement introduces four types of relationships

between requirements: conflicting, cooperative, mutually exclusive and irrelevant. For instance, two imprecise requirements are said to be conflicting with each other if an increase in the satisfaction degree of one requirement decreases the satisfaction degree of the other. Knowledge-based techniques developed on these relations allow assessing the impact of requirement changes and inferring relationships between requirements in order to detect implicit conflicts. Detected conflicts are resolved by favouring satisfaction of requirements with high priority.

Cîmpman and Oquendo propose to use fuzzy logic to monitor software processes [7,8,9]. The monitoring process focuses on the detection of deviations between the actual enacting process and the process enactment plan. The level of deviation is computed for different aspects of the process like progress, cost, structure (order between activities), etc., and varies from total conformance to no conformance at all. Fuzzy logic is used to represent possible imprecise and uncertain information handled by the monitoring system, and to reason on it. Similarly to our approach, the use of fuzzy logic is justified by its ability to naturally represent uncertain and imprecise information, and to constitute a good framework for approximate reasoning. Our approach, however, focuses on managing software artifacts, whereas the monitoring system focuses on handling deviations of software processes. The two approaches are defined at two different abstraction levels.

8. Future work

In addition to reducing the quantization error and contextual bias problems, the fuzzy logic-based approach opens a set of interesting perspectives to software development. In the following, we will present some of these perspectives and briefly indicate our related research.

- *Accumulative software development*: Most current life-cycle models, such as the water-fall or the spiral models [14] are inherently based on two-valued logic. Adopting fuzzy logic reasoning brings new perspectives to modelling software life-cycle. Basically, a fuzzy logic-based method implements an accumulative learning process; at each step of the development process, a new aspect of the software being developed can be learned. This naturally results in a very adaptable and reusable design model. We believe that fuzzy logic-based methods can shift today's emphasis of producing reusable code to creating reusable and adaptable design models.
- *Design documentation*: In fuzzy logic-based methods, each artifact type is characterized by a set of properties whose values determine the membership degree of an artifact to the artifact type. Property values are modified through the application of rules which are implemented as fuzzy operations. These operations can be stored as history information. Fuzzy logic-based object models, therefore, naturally document the complete software development history; applied rules, the assumptions made by the software engineers, the contextual information, they all can be fully traced. More importantly, this way of documenting design information is fully integrated with the object model, since the concepts that constitute the object model are created through the application of these rules. We will be carrying out assessment studies on how this self-documenting design expertise can be utilized in a commercial organization.
- *Active methods*: While developing software systems, decisions made about the validity of certain artifacts may change. Similarly, lack of modelling techniques and methods for dedicated applications may force the designer to take decisions towards a wrong direction. For example, assumptions made on processing speed and heuristics on dynamic behaviour of a system may not

remain valid. Changing the property value of an artifact may occur during the software development or even during the operation phase. Since artifact types are largely dependent on each other, changing the property values of a certain artifact may influence other related artifacts. For large software systems, it can be very tedious to monitor and validate all the dependencies manually. Fuzzy logic-based methods can maintain knowledge about the process and context of the software development activity so that they can assist software engineers specifically. The fuzzy logic-based method automatically updates the related linguistic values because it is implemented as a fuzzy logic reasoning system. Obviously, here cyclic dependencies must be avoided [20]. We are currently experimenting with the usefulness of active methods.

9. Conclusions

In previous papers [1,19,20], we have introduced fuzzy logic-based methods and have given a number of examples to illustrate their applicability. The contribution of this article has been to formally introduce the so-called quantization error and contextual bias problems, which affect current software development methods based on two-valued logic, and to show how fuzzy logic-based methods can reduce these problems. Fuzzy logic can capture the software engineer's perception more appropriately than two-valued logic thanks to its ability to compute with real-word linguistic expressions. Fuzzy logic is also useful in adapting design rules with respect to changing contexts. Furthermore, application of fuzzy logic-based reasoning opens new perspectives to software development, such as accumulative software life cycle, integrated design documentation and active methods.

Acknowledgements

We highly appreciate the corrections made by Pim van den Broek on the formulation of the quantization error.

References

[1] M. Aksit, F. Marcelloni, Deferring elimination of design alternatives in object-oriented methods, Concurrency Comput.—Practice Experience 13 (14) (2001) 1247–1279.
[2] V.C. Basili, H.D. Rombach, The TAME project: towards improvements-oriented software environments, IEEE Trans. Software Engrg. 14 (6) (1988) 758–772.
[3] L. Benedicenti, G. Succi, T. Vernazza, A. Valerio, Object oriented process modeling with fuzzy logic, in: Proc. 1998 ACM Symp. on Applied Computing, ACM, Atlanta, Georgia, United States, 1998, pp. 267–271.
[4] G. Booch, Object-oriented design with applications, The Benjamin/Cummings Publishing Company Inc., Menlo Park, CA, 1991.
[5] D. Budgen, Software Design, Addison-Wesley, Reading, MA, 1994.
[6] S.R. Chidamber, C.F. Kemerer, A metrics suite for object-oriented design, IEEE Trans. Software Engrg. 20 (6) (1994) 476–492.
[7] S. Cîmpan, F. Oquendo, Fuzzy indicators for monitoring software processes, in: Proc. 6th European Workshop on Software Process Technology, Springer, Berlin, Germany, 1998, pp. 43–59.

[8] S. Cîmpan, F. Oquendo, On the application of fuzzy set theory to the monitoring of software-intensive processes, in: Proc. 8th Internat. Fuzzy Systems Association World Congress, National Central University, Chungli, Taiwan, 1999, pp. 1071–1076.
[9] S. Cîmpan, F. Oquendo, Dealing with software process deviations using fuzzy logic-based monitoring, ACM Appl. Comput. Rev. 8 (2) (2000) 3–13.
[10] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes, Object-oriented Development: The Fusion Method, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[11] I. Graham, Object-oriented Methods, 2nd ed., Addison-Wesley, Reading, MA, 1994.
[12] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard, Object-oriented Software Engineering—A Use Case Driven Approach, Addison-Wesley/ACM Press, Reading, MA/New York, 1992.
[13] R. Johnson, B. Foote, Designing reusable classes, J. Object-Oriented Programming 1 (2) (1988) 23–35.
[14] G.W. Jones, Software Engineering, Wiley, New York, 1990.
[15] G.J. Klir, B. Yuan, Fuzzy Sets and Fuzzy Logic—Theory and Applications, Prentice-Hall, Upper Saddle River, NJ 07458, 1995.
[16] B. Lazzerini, F. Marcelloni, Some considerations on input and output partitions to produce meaningful conclusions in fuzzy inference, Fuzzy Sets and Systems 113 (2) (2000) 221–235.
[17] X.F. Liu, J. Yen, An analytic framework for specifying and analyzing imprecise requirements, in: Proc. 18th Internat. Conf. on Software Egineering, Berlin, Germany, 25–30 February 1996, pp. 60–69.
[18] F. Marcelloni, M. Aksit, Reducing quantization error and contextual bias problems in software development processes by applying fuzzy logic, in: Proc. NAFIPS'99, IEEE Press, New York, 1999, pp. 268–272.
[19] F. Marcelloni, M. Aksit, Improving object-oriented methods by using fuzzy logic, ACM Appl. Comput. Rev. 8 (2) (2000) 14–23.
[20] F. Marcelloni, M. Aksit, Leaving inconsistency using fuzzy logic, Inform. Software Tech. 43 (2001) 725–741.
[21] A. Papoulis, Probability, Random Variables, and Stochastic Processes, McGraw-Hill, New York, 1991.
[22] A. Riel, Object Oriented Design Heuristics, Addison-Wesley, Reading, MA, 1996.
[23] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object-oriented Modeling and Design, Prentice-Hall, Englewood Cliffs, NJ, 1991.
[24] M. Schwartz, Information Transmission, Modulation, and Noise, 4th ed., McGraw-Hill, New York, 1990.
[25] B. Tekinerdogan, Design and experimentation of a fuzzy logic controller for evaluating domain knowledge, in: Proc. 2001 SCASE, University of Twente, 8–9 February 2001.
[26] A. Yazici, R. George, B.P. Buckles, F.E. Petry, A survey of conceptual and logical data models for uncertainty, in: L.A. Zadeh, J. Kacprzyk (Eds.), Fuzzy Logic for the Management of Uncertainty, Wiley, New York, 1992, pp. 281–295.
[27] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, IEEE Trans. Systems Man Cybernet. SMC-3 (1) (1973) 28–44.
[28] L.A. Zadeh, Fuzzy logic = computing with words, IEEE Trans. Fuzzy Systems 4 (2) (1996) 103–111.