Reducing Quantization Error and Contextual Bias Problems in Software Development Processes by Applying Fuzzy Logic

Francesco Marcelloni
Dipartimento di Ingegneria della Informazione
University of Pisa
Via Diotisalvi, 2 -56156 Pisa, ITALY
france@iet.unipi.it

Mehmet Aksit
Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
aksit@cs.utwente.nl

Abstract

Object-oriented methods define a considerable number of rules, which are generally expressed using two-valued logic. For example, an entity in a requirement specification is either accepted or rejected as a class. There are two major problems how rules are defined and applied in current methods. Firstly, two-valued logic cannot effectively express the approximate and inexact nature of a typical software development process. Secondly, the influence of contextual factors on rules is generally not modeled explicitly. This paper terms these problems as quantization error and contextual bias problems, respectively. To reduce these problems, we adopt fuzzy logic-based methodological rules. This approach is method independent and is useful for evaluating and enhancing current methods. In addition, the use of fuzzy-logic increases the adaptability and reusability of design models.

1. Introduction

During the last decade, a considerable number of object-oriented methods [4] have been introduced. Methods aim to create software artifacts by exploiting object-oriented concepts[1] through the application of a large number of rules. For example, OMT [4] introduces rules for identifying and discarding classes, associations, part-of and inheritance relations, state-transition and data-flow diagrams. Basically, these rules are based on two-valued logic. For example, a candidate class is generally identified by applying the rule: *If an entity in a requirement specification is relevant then select it as a candidate class.*

We consider two major problems, termed as *quantization error* and *contextual bias* problems, in the way how rules are defined and applied in current object-oriented methods. Firstly, two-valued logic cannot really express the approximate and inexact nature of a typical software development process. 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. The software engineer can perceive that the entity partially fulfills the relevance criterion and may conclude that the entity is, for instance, substantially relevant. However, two-valued logic-based methodological rules 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 modeled and therefore cannot be considered explicitly in the subsequent phases of the development process. Secondly, 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.

To reduce these problems, we adopt fuzzy logic-based methodological rules. Fuzzy logic can express uncertainty and imprecision. Further, fuzzy logic provides a sound framework to define a language, to associate a meaning with each expression of the language and to provide a means to compute these expressions. A software engineer can, therefore, describe his/her perception using his/her natural language and this perception can be modeled and maintained along all the development process. He/she can identify, for instance, the relevance of an entity as weakly, slightly, fairly, substantially or strongly relevant. Capturing as much as possible the perception of the software engineer reduces the loss of information and, consequently, improves the quality of the software development process. Also, the influence of contextual factors on the validity of methodological rules can be controlled by adapting the meaning associated with each expression. The adaptation process may be defined by means of heuristic rules based in their turn on fuzzy logic as well. Finally, fuzzy logic allows managing a number of design alternatives and associating a measure with each alternative. Measures prove to be particularly useful in selecting the best alternative in a set of possible conflicting

[1] The term *concept* refers to the types of software artifacts of object-oriented development process. Typical examples of object-oriented concepts are Class, Object, Association, Part-of relation, Inheritance relation, Attribute, Operation, State-transition diagram.

design alternatives.

2. The Quantization Error Problem

Assume that the following rule is used to identify candidate classes:

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 in a requirement specification* and *a candidate class* are the two object-oriented artifacts to be reasoned. *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, we will refer to similar rules, which are commonly adopted by 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*. If all the rules, which are applicable to an entity in a requirement specification, reject that entity, then the entity is practically discarded. Classifying not correctly artifacts, especially in the first phases of the development process, may irremediably deteriorate the quality of whole 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 the concept of quantization error clear we can refer to the area of digital signal processing. Here, quantization process consists of assigning 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. Quantization error is defined as the difference between an analog and the corresponding quantized signal sample. Less the number of quantization levels, higher the quantization error.

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 fulfils 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. A formulation of the quantization error in the case of two-valued logic-based rules has been presented in [1].

One of the dramatic effects of the quantization error on the development process is early elimination of the 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 a 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 [4] 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 because it is considered non-relevant. The discarded entity, however, could have been included as a candidate class, if the software engineer had gathered more information about its structure and operations. 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 an iteration remains as a difficult task.

3. The Contextual Bias Problem

Contextual factors may influence validity of the result of a methodological rule in two ways. Firstly, the input of a rule can be largely context dependent. In the rule *Redundant Class Elimination*, for instance, the elimination of a class is based on the perception of the software engineer whether he or she finds a candidate class more descriptive than an equivalent class.

Secondly, validity of a rule may depend on contextual factors such as application domain, changes in user's interest and technological advances. Let us consider the following rule *Inheritance Modification* extracted by [3].

IN THE CLASS HIERARCHY, IF THE NUMBER OF IMMEDIATE SUBCLASSES SUBORDINATED TO A CLASS IS LARGER THAN 5, THEN THE INHERITANCE HIERARCHY IS COMPLEX.

If this rule concludes that the inheritance hierarchy is complex, then the hierarchy may be modified. The success of this rule heavily depends on the type of application. For example, in graphics applications, it appears natural that many classes inherit directly from class Point. This is because class Point represents a very basic abstraction in a graphic processing system. Using metrics based rules may not eliminate the effects of context either. As some authors indicate [2], metrics must be associated with some interpretation to determine the threshold of a design rule. But this interpretation must be given in a context. Only when the variables, which can influence the measure, are fixed, the interpretation of the metrics becomes univocal. Otherwise, the result is either an improper interpretation or a large amount of possible interpretations. We term the effects of context to the development process as the *contextual bias problem.*

4. Fuzzy logic-based method

To reduce the quantization error and contextual bias problems in methodological rules, a new expressive form rather than two-valued logic has to be investigated. Such a form has to be able to capture as much as possible the software engineer's perception so as to increase the number of quantization levels. To this aim, two requirements are strongly demanded: i) similarity to the natural language typically used by the software engineer and ii) capability to reason on the linguistic expressions to deduce conclusions and conduct the development process. Further, the new expressive form has to model the influence of contextual factors. Fuzzy logic looks to be the ideal solution.

As L. Zadeh claims in [6], one of the main contribution of fuzzy logic is computing with words. Fuzzy logic provides a sound framework to define a language, to associate a meaning with each expression of the language and to compute these expressions. Basic in fuzzy logic is the concept of linguistic variable: A linguistic variable is a variable whose values, called *linguistic values*, have the form of phrases or sentences in a natural language [5]. Each linguistic value is associated with a fuzzy set that represents its meaning. Relations between linguistic variables are defined by means of fuzzy rules which are typically expressed as: IF X IS A THEN Y IS B, where *X* and *Y* are linguistic variables and *A* and *B* are linguistic values. Given a fact and a rule, one of the most known fuzzy inference tools, the *generalized modus ponens*, allows deducing a fuzzy conclusion. If a crisp value is required, the corresponding fuzzy set is *defuzzified* by using a *defuzzification* operation [5].

4.1. Reducing the quantization error

Denote each object language concept as *[C, (P_1, D_1), (P_2, D_2),...,(P_n, D_n)]* where *C* is the concept name, P_i is a property of *C* and D_i is the definition domain of P_i. An example of a concept is *[Entity, (Relevance, {True, False}), (Autonomy, {True, False})]*. Here, *True* and *False* are the only two values that *Relevance* and *Autonomy* can assume in current methodological rules. A software artifact is an instantiation of its concept and can be expressed as *[C, id, $(P_1: V_1)$, $(P_2: V_2)$,...,$(P_n: V_n)$]*, where *C* is the concept of the artifact, *id* is the unique identifier of the artifact, and V_i is a value defined in domain D_i of property P_i. Artifacts can be also named. In the following example, *Name* is the name of the artifact:

*Name← [**Entity**, id, (**Relevance**: True), (**Autonomy**: True)]*

From our experience, two values are not enough to codify the software engineer's perception about properties of an artifact. Therefore, we considered each property as a linguistic variable and investigated which values and meanings can be significant to a software engineer. For instance, we verified that the property *Relevance* of the artifact *Entity* can be expressed as *weakly*, *slightly*, *fairly*, *substantially* and *strongly relevant* and the property *Autonomy* as *dependently*, *partially autonomy* and *fully autonomy*. The meaning of the linguistic values of *Relevance* and *Autonomy* are shown in Figures 1 and 2. Here, the X and Y axes indicate the universe of discourse and the membership values, respectively. The universes are supposed to vary from 0 to 1. In these figures, each linguistic value is shown as a different line type. Consequently, the concept *Entity* can be expressed as:

*[**Entity**, (**Relevance**, {Weakly, Slightly, Fairly, Substantially, Strongly}), (**Autonomy**, {Dependently, Partially Dependently, Fully Autonomously})]*

Fitting the software engineer's perception increases the number of quantization levels and therefore decreases the quantization error [1]. Methodological rules are also expressed using fuzzy logic. Consider, for example, the modified rule *Candidate Class Identification*:

IF AN ENTITY IN A REQUIREMENT SPECIFICATION IS <u>RELEVANCE VALUE</u> RELEVANT AND CAN EXIST <u>AUTONOMY VALUE</u> AUTONOMOUS IN THE APPLICATION DOMAIN, THEN SELECT IT AS A <u>RELEVANCE VALUE</u> RELEVANT CANDIDATE CLASS.

Here, an entity and a candidate class are the concepts to be reasoned, *Relevance* and *Autonomy* are the properties, and <u>relevance value</u> and <u>autonomy value</u> indicate the domains of these properties. The rule *Candidate Class Identification* can be represented in the following way:

*P ← [**Entity**, id_1, (**Relevance**: $V_1 \in$ {Weakly, Slightly, Fairly, Substantially, Strongly}), (**Autonomy**: $V_2 \in$ {Dependently, Partially Dependently, Fully Autonomously})]*

$\Rightarrow$

*P ← [**CandidateClass**, id_2, (**Relevance**: $V_3 \in$ {Weakly, Slightly, Fairly, Substantially, Strongly})]*

Here, *P* and symbol ⇒ indicate a generic artifact name and the fuzzy implication operator, respectively. Each combination of relevance and autonomy values of an entity has to be mapped into one of the five candidate class relevance values. 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 are respectively *Strongly* and *Fully Autonomously*, then the candidate class relevance value is *Strongly*. We selected these output values based on our intuition and knowledge on object-oriented methods.

	P ← Entity, Autonomy:		
P ← CandidateClass, Relevance:	Dependently	Partially Dependently	Fully Autonomously
Weakly	*Weakly*	*Weakly*	*Weakly*
Slightly	*Weakly*	*Slightly*	*Slightly*
Fairly	*Weakly*	*Slighlty*	*Fairly*
Substantially	*Weakly*	*Fairly*	*Substantially*
Strongly	*Slightly*	*Fairly*	*Strongly*
P ← Entity, Relevance:			

Table 1. Sub-rules of the rule *Candidate Class Identification.*

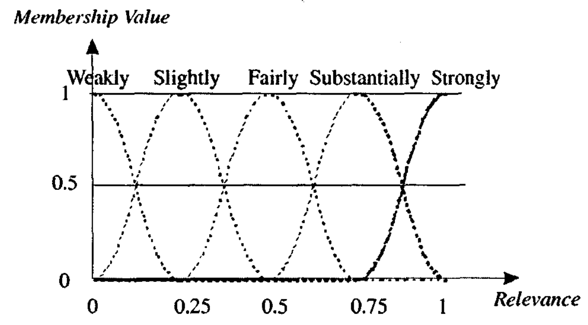

Figure 1. Linguistic variable Relevance.

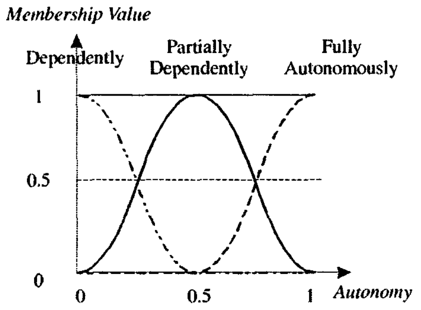

Figure 2. Linguistic variable Autonomy.

If fuzzy logic-based methodological rules are applied, none of the concepts are theoretically eliminated: artifacts can be 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. 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 a very adaptable and reusable design models.

Further, fuzzy logic allows managing a number of design alternatives and associating a measure with each alternative. Consider, for instance, that during an object-oriented development process, the software engineer judges an entity as a *substantial* candidate class and a *slight* attribute. The concepts of class and attribute are considered as conflicting in object-oriented paradigm. During the whole 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 delivered, 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.

4.2. Reducing the Contextual Bias Problem

Contextual factors can affect inputs of the rules and compromise the validity of the rules themselves. In our fuzzy-logic based approach, the first effect is reduced by increasing the number of quantization levels. Consider the rule *Candidate Class Identification.* Selection of an entity as a candidate class is based on the software engineer's perception of relevance. This perception can be different from software engineer to software engineer. In two-valued logic based methods, a little difference in perception can cause contradictory results. For example, assume that the same entity in a requirement specification is considered differently by two software engineers, one as *slightly* and the other as *substantially relevant.* In case of a two level quantization process, it is likely that the first software engineer would reject and the second one would accept the entity as a candidate class. By increasing the number of quantization levels, the difference between the input values caused by contextual factors is not amplified.

The effect of contextual factors on the validity of a rule can be reduced by modeling the influence of the context explicitly. The validity of a rule is determined by the validity of its conditions. For instance, let us consider the rule *Inheritance Modification* as defined in section 3. The condition of this rule may not be valid for certain kinds of applications. Our solution to this problem is to adapt the meaning of linguistic values based on the contextual factors. Consider the fuzzy logic rule *Inheritance Modification*:

*P_1 ← [**Class**, id_1, (**ImmediateSubclasses**: V_1 ∈ {Low, Medium, High})] ⇒*
*P_2 ← [**Inheritance**, id_2, (**Complexity**: V_2 ∈ {Low, Medium, High})]*

The validity of this rule depends on the meanings associated with linguistic values *Low*, *Medium* and *High*. Different contexts may associate different meanings with a linguistic value. For instance, in case of a graphics application, the membership functions associated with the linguistic values should be adapted so that higher values of number of immediate subclasses could be acceptable.

A membership function can be adapted by translating, compressing and dilating. Translation operation is used to shift the membership function along the Y axis. Figure 3 shows a linear dilation function with factor 2. The compression or dilation function has to be related to the contextual factors. In general, it is difficult to formalize this relation 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. For instance, let us consider to use a linear dilation to adapt the meaning of linguistic values *Low*, *Medium* and *High* for property *Number of Immediate Subclasses*. The relation between the type of application and the degree of dilation may be expressed by the following contextual rule.

*P1 ← [**GraphicProcessingApplication**, id1, (**Certainty**: V1 ∈ {Doubtfully, Approximately, Certain})] ⇒*
*P2 ← [**Dilation**, id2, (**Degree**: V2 ∈ {Low, Medium, High})]*

Depending on the type of application, the contextual rules determine a value for linguistic variable *Dilation*. By defuzzifying this value, the dilation factor can be obtained.

5. Conclusions

This paper identifies the concept of quantization error and contextual bias problems that one may experience during software development process. To minimize these problems, fuzzy logic-based methodological rules have been proposed. It has been shown that 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. This allows reducing quantization error and is useful in adapting design rules with respect to changing contexts. In addition, the application of fuzzy-logic based reasoning opens new perspectives to software development, such as accumulative software life-cycle and integrated design documentation. Indeed, a fuzzy logic based method implements an accumulative learning process; after each process, a new aspect of the software being developed can be learned. This naturally results in a very adaptable and reusable design model. Further, each concept in the fuzzy logic based method has a set of property-value pairs, which can be modified through the application of new rules implemented as fuzzy logic operations. These operations can be stored as a history information. Fuzzy logic-based object models, therefore, naturally document the complete software development history. 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. A small fuzzy-logic based method has been implemented using our experimental CASE environment and tested on an example problem.

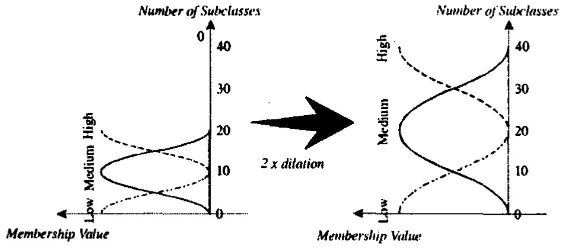

Figure 3. Adapting to context through dilation.

References

[1] M. Aksit and F. Marcelloni. Reducing Quantization Error and Contextual Bias Problems in Object-Oriented Methods by Applying Fuzzy Logic Techniques, University of Twente, Report, 1997.

[2] V.C. Basili and H.D. Rombach. The TAME Project: Towards Improvements-Oriented Software Environments. *IEEE Transactions on Software Engineering*, Vol. 14, No. 6, pp. 758-772, June 1988.

[3] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476-492, June 1994.

[4] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[5] L.A. Zadeh. Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-3, No.1, pp. 28-44, January, 1973.

[6] L.A. Zadeh. Fuzzy Logic = Computing with Words. *IEEE Transactions on Fuzzy Systems*, Vol. 4, No. 2, pp. 103-111, May, 1996.