Applying Fuzzy Logic Techniques in Object-Oriented Software Development

Francesco Marcelloni[1] and Mehmet Aksit[2]

[1] Department of Information Engineering, University of Pisa,
Via Diotisalvi, 2-56126, Pisa, Italy.
email: france@iet.unipi.it
[2] TRESE Project, Department of Computer Science University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands.
Email: aksit@cs.utwente.nl, www server: http://wwwtrese.cs.utwente.nl

1 Introduction

In the last several years, a considerable number of object-oriented methods have been introduced to create robust, reusable and adaptable software systems [1], [2], [3], [4]. Object-oriented methods define a considerable number of rules which are generally expressed by using two-valued logic. For instance, an entity in a requirement specification is either accepted or rejected as a class. We consider two major problems in the way how rules are defined and applied in current object-oriented methods. The first problem, termed quantization problem, is a natural result of the incapacity of two-valued logic to express the approximate and inexact nature of a typical software development process. The second problem, termed contextual bias problem, arises because most of methods are not able to model the effects of the context on the validity of the method. To reduce these problems, we propose a new fuzzy logic-based object-oriented software development technique. This technique is not specific to a particular object-oriented method, but can be used to evaluate and enhance current methods. In addition, the application of fuzzy logic-based reasoning opens new perspectives to software development, such as fuzzy artifacts and accumulative software life-cycle.

2 Problem Statement

2.1 The Quantization Problem

Object-oriented methods exploit object-oriented concepts through the application of a large number of rules [1], [2], [3], [4]. 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. However, two-valued logic does not provide an effective means for capturing the approximate and inexact nature of a typical software development process. Let us consider, for instance, the following rule, termed *Candidate Class Identification*, which is typically applied to identify a candidate class:

IF AN ENTITY IN A REQUIREMENT SPECIFICATION IS RELEVANT
AND CAN EXIST AUTONOMOUSLY IN THE APPLICATION DOMAIN
THEN SELECT IT AS A CANDIDATE CLASS

The software engineer has to determine whether the entity being considered is relevant or not for the application domain. The software engineer may conclude that the entity partially fulfils the relevance criterion, and may prefer to use expressions like the entity is 70% percent relevant or substantially relevant. However, two-valued logic forces the software engineer to take crisp decisions, such as accepting or rejecting the entity as a class and, then, to quantize the available information into fixed levels. This quantization process results in a high quantization error and consequently in loss of information because the partial relevance of the entity is not modeled and cannot be considered explicitly in the subsequent phases. Referring to the area of digital signal processing where quantization errors have been extensively studied, the root mean square value of the quantization error can be also formulated for the software development process. For instance, we evaluated that the information contained in the conclusion inferred from rule *Candidate Class Identification* is only 71.7 percent of the available information present at the input of the rule. Further, we verified that coupling of rules, which is typically adopted in software development methods, increases dramatically the loss of information.

2.2 The Contextual Bias Problem

Contextual factors may influence validity of the result of a rule in two ways. Firstly, the input of a rule can be largely context-dependent. In the rule *Candidate Class Identification*, the relevance and autonomy of an entity depend on the perception of the software engineer. Secondly, the validity of a rule may largely depend on factors such as the application domain, changes in user's interest, and technological advances. Unless the contextual factors that influence a given rule are defined explicitly, the applicability of that rule cannot be determined and controlled effectively.

3 Our Approach

In the previous sections, we discussed the quantization and the contextual bias problems. An obvious solution to reduce the quantization error is to increase the number of quantization levels. In the *Candidate Class Identification* rule, the relevance of an entity can be expressed only as relevant or not relevant. To increase the number of quantization levels, we have to split the range of relevance into more levels such as *weakly*, *slightly*, *fairly*, *substantially* and *strongly* relevant. This transformation is not forced as a software engineer naturally expresses these gradual perceptions. Although each level can correspond to a crisp set, we observed that, for most of the inputs, the transition between membership and non-membership appears gradual rather than abrupt. On the other hand, in two-valued logic defining this crisp boundary is necessary because a proposition

can be either true or false. It is therefore worth to investigate other forms of logic than two-valued logic, such as fuzzy logic.

In fuzzy logic, the concept of vagueness is introduced by the definition of fuzzy set. A fuzzy set S of a universe of discourse U is characterized by a membership function which associates with each element y of U a number in the interval $[0, 1]$ which represents the grade of membership of y in S [5]. Based on the definition of fuzzy sets, the concept of linguistic variables is introduced to represent a language typically adopted by a human expert. A *linguistic variable* is a variable whose values, called *linguistic values*, have the form of phrases or sentences in a natural or artificial language [5]. For instance, the relevance of an entity in a requirement specification can be modeled as a linguistic variable which might assume linguistic values *weakly*, *slightly*, *fairly*, *substantially*, and *strongly* relevant.

Each linguistic value is associated with a fuzzy set representing its meaning. To represent multiple levels conveniently, also rules have to be modified and expressed in terms of fuzzy logic. For instance, the *Candidate Class Identification* fuzzy rule is:

IF AN ENTITY IN A REQUIREMENT SPECIFICATION IS *Relevance_Value* RELEVANT
AND CAN EXIST *Autonomy_Value* IN THE APPLICATION DOMAIN
THEN SELECT IT AS A *Relevance_Value* RELEVANT CANDIDATE CLASS

Here, an entity and a candidate class are the concepts to be reasoned, and *Relevance_Value* and *Autonomy_Value* indicate the domains of properties *Relevance* and *Autonomy* of these concepts. Assume that *Relevance_Value* and *Autonomy_Value* represent the sets of values {*Weakly, Slightly, Fairly, Substantially, Strongly*} and {*Dependently, Partially Dependently, Fully Autonomously*}, respectively. Each combination of relevance and autonomy values of an entity has to be mapped into one of the five candidate class relevance values. This requires 15 rules (*subrules*) in total. An example of subrule is the following:

IF AN ENTITY IN A REQUIREMENT SPECIFICATION IS STRONGLY RELEVANT
AND CAN EXIST FULLY AUTONOMOUSLY IN THE APPLICATION DOMAIN
THEN SELECT IT AS A STRONGLY RELEVANT CANDIDATE CLASS

We would like to point out that the shift from two-valued to fuzzy logic rules in software development is quite natural. This is because most object-oriented design rules are applied to uncertain and vague domains.

To reduce the contextual bias problem, the influence of the context on the results inferred from the design rules has to be formulated. By increasing the number of quantization levels, contextual factors such as, for instance, software engineer's perception, affect less the inputs of the rules. Further, the effect of contextual factors on the validity of a rule can be reduced by modeling the influence of the context explicitly. This can be obtained by adapting the meaning of linguistic values based on contextual factors, that is, by translating, compressing or dilating the corresponding membership functions. The degree of translation, compression or dilation has to be related to the contextual factors. In general, it is not possible to formalize analytically this relation and therefore heuristics

have to be adopted. Since rules defining the effect of contextual factors are typically expressed in terms of linguistic expressions, fuzzy logic is again used to implement these rules. The linguistic values inferred from these rules determine the type and the degree of transformation of the membership functions.

4 Evaluation of the Method

As well as reducing the quantization error, the increase of the number of quantization levels coupled with the use of fuzzy logic allows a software engineer to deal with design rules expressed in more familiar way. Further, the conclusion inferred from each design rule is not an crisp decision on maintaining or eliminating an artifact. Two-valued logic, not being able to express intermediate levels, causes the elimination of artifacts even if all the relevant information is not collected. In the fuzzy logic-based method each artifact (called *fuzzy artifact*) collects fuzzy information and survives during the whole software process. From this viewpoint, 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. Only when all the relevant information is collected, a fuzzy artifact can be transformed into an artifact and then maintained or eliminated.

It follows that during the development process an entity can be considered, for instance, both as an attribute and as a class. It is the learning process which tips the scales in favour of one or the other concept, deciding for the elimination of one of the two fuzzy artifacts. To avoid early elimination of artifacts means, however, to increase the cost of the development process. A possible CASE environment has to maintain all the fuzzy artifacts in each phase of the development. Anyway, most concepts will be likely to have the lowest grade of property values. This makes it possible to minimize storage requirements by registering only the artifacts that have a grade of property values over a fixed threshold. Similar to designing digital signal processing systems, the fuzzy logic-based method provides a unique opportunity to tune the quality of the development process with respect to memory and processing costs.

References

1. Booch, G.: Object-Oriented Design with Applications. The Benjamin/Cummings Publishing Company, Inc. (1991).
2. Coleman, D. et al.: Object-Oriented Development, The Fusion Method. Prentice Hall (1994).
3. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-Oriented Software Engineering – A Use Case Driven Approach. Addison-Wesley/ACM Press, (1992).
4. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice-Hall, Inc., (1991).
5. Zadeh, L.A.: Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. IEEE Transactions on Systems, Man, and Cybernetics **SMC-3** 1 (1973) 28-44.