Toward the Automatic Assessment of Evolvability for Reusable Class Libraries

Houari A. Sahraoui
DIRO, Université de Montréal, Canada
sahraouh@iro.umontreal.ca

Hakim Lounis
CRIM, Canada
hlounis@crim.ca

Mounir A. Boukadoum
Dept. d'informatique
Université du Québec à Montréal, Canada
Mounir.Boukadoum@uqam.ca

Frédéric Ethève
ENST, France
etheve@email.enst.fr

Abstract

Many sources agree that managing the evolution of an OO system constitutes a complex and resource-consuming task. This is particularly true for reusable class libraries, as the user interface must be preserved to allow for version compatibility. Thus, the symptomatic detection of potential instabilities during the design phase of such libraries may serve to avoid later problems. This paper presents a fuzzy logic-based approach for evaluating the interface stability of a reusable class library, by using structural metrics as stability indicators.

1. Introduction

It is widely recognized that evolving software is a hard and time-consuming activity. Pressman estimated at 60% the part devoted to maintenance in the total effort of the software development industry [8], from which 80% is devoted directly or indirectly to software evolution (adaptive and perfective maintenance) [7]. In spite of the benefits of object-oriented technology, OO class libraries, like the majority of the systems, do not derogate from this rule. Moreover, their evolution must take into account an additional constraint, to preserve, as much as possible, the compatibility (interfaces) among versions.

In our context, we define the evolvability in general by the ease with which a software system or a component can evolve while preserving its design as much as possible. In the case of OO class libraries, we restrict the preservation of the design to the preservation of the library interface. This is important when we consider that the evolvability of a system that uses a library is directly influenced by the evolvability of the library. Preserving the library interface must be a continuous goal starting from the initial design. A good design must allow the improvement of existing functionalities and the addition of new ones while preserving the library interface. This leads to the problems of assessing the goodness of a design from the perspective of evolvability, and of identifying the internal attributes that could be used as evolvability indicators.

In this paper, we study the hypothesis that inheritance aspects can be good indicators of the evolvability of an OO class library. To this end, we propose a fuzzy logic-based approach for building and assessing evolvability estimation models.

2. Fuzzy logic-based learning

There exist several techniques to derive quality estimation models. One category that gives interesting results is that of machine learning (ML) techniques. In this section, we first present briefly two popular algorithms, C4.5 and RoC and then show their limitations when dealing with software quality estimation models. In the rest of the section, we propose an approach for building estimation models based on fuzzy-logic.

2.1. Classical learning principle

Most of the work done in ML has focused on supervised machine learning algorithms. Starting from the description of classified examples, these algorithms produce definitions for each class. In general, they use an attribute-value representation language that allows the exploitation of the learning set statistical properties, leading to efficient software quality models. C4.5 is representative of the Top Down Induction of Decision Trees (TDIDT) approach [9]. We used it in many past works to generate estimation models in software engineering. This was the case, for instance, in [4] where the goal was to assess an empirical value of reusability starting from coupling, inheritance, and complexity metrics on OO systems.

C4.5 belongs to the divide and conquer algorithms family. In this family, the induced knowledge is generally represented by a decision tree. It works with a set of examples where each example has the same structure, consisting of a number of attribute/value pairs. One of these attributes represents the class of the example.

Closer to probabilistic approaches, another approach is illustrated by RoC, a Bayesian classifier (see [2]). It is trained by estimating the conditional probability distributions of each attribute, given the class label. The classification of a case, represented by a set of values for each attribute, is accomplished by computing the post probability of each class label, given the attributes values, by using Bayes' theorem. The case is then assigned to the class with the highest posterior probability.

RoC extends the capabilities of the Bayesian classifier to situations in which the database reports some entries as unknown. It can then train a Bayesian classifier from an incomplete database. More information about this process is given in ([10]).

One of the great advantage of C4.5, when compared to RoC, is that it produces a set of rules that is readily understandable by software managers and engineers. However, our past experience with this ML algorithm, when applied to software products data, as well as with "classical" ML approaches in general, reveals weaknesses in the learning/classification process. One of the main points concerns the fact that the generated estimation models are too specific or precise. This is first due to the algorithms themselves, but also to the non-availability of data sets. The consequence of this situation is that we obtain specific models that are not general enough to be efficiently applicable by software managers.

Another problem concerns the classification problem itself. It is related to the fact that, during the process of classifying a new case, an algorithm such as C4.5 exploits the first valid path/rule while we would expect it to consider all the valid paths/rules and, then, deduce a more consensual result.

In this context, we propose to use a fuzzy logic approach to manage the pre-cited problems. This is the content of the next section.

2.2. Fuzzy-based learning

The main cause for the problems outlined above is that, for most decision algorithms based on classical ML approaches, only one rule is fired at a time while traversing the decision tree. As a result, only one branch is followed from any given node, leading to one single leave as a conclusion, and exclusive of all other possible paths. This is not representative of most real-life problems where the input information is vague and imprecise, when not fragmentary. For such problems, the idea of setting thresholds at the nodes, and, then, of following decision paths based on whether given input attribute values are above of below the thresholds, may lead to opposite conclusions for any two values that are close to a threshold from opposite directions. In such situations, one would like to be able to partially fire a rule and simultaneously fire several rules.

The key step of C4.5 algorithm is the selection of the "best" attribute to obtain compact trees with high predictive accuracy. Information theory-based heuristics have provided effective guidance for this division process. A measure of entropy is used to measure how informative a node is. In general, if we are given a probability distribution $P=(p_1, p_2, \ldots, p_n)$, where n represents the number of values for an attribute, and where p_i is the probability of the i^{th} value, then, the information conveyed by this distribution is given by Shannon's entropy:

$$I(P) = -(p_1*\log(p_1) + p_2*\log(p_2) + .. + p_n*\log(p_n))$$

This notion is exploited to rank attributes and to build decision trees where at each node, we use the attribute with the greatest discrimination power.

We apply the same principle to the creation of fuzzy decision trees. However, in this case, fuzzy entropy is used to measure the information provided by a node. Fuzzy entropy (also called star entropy) is an extension of Shannon's entropy where classical probabilities are replaced by fuzzy ones [11]. For an attribute C with a set of fuzzy labels $\{c_k\}$, it is defined as:

$$H_S^*(C) = -\sum_k P^*(c_k)\log\left(P^*(c_k)\right)$$

where $P^*()$ usually stands for the fuzzy probability defined by Zadeh [12]:

$$P^*(c_k) = -\sum_i m(e_i)P(e_i)$$

Fuzzy probabilities differ from normal probabilities in that they represent the weighted average of a set of values provided by a membership function m. These values represent the degrees of membership of a fuzzy event, related to class c_k, to the different elements e_i of a fuzzy set.

In a fuzzy decision tree, the processing of input attribute values starts with the fuzzification of each attribute so that it takes values from a discrete set of labels. Each label has an associated membership function that sets the degree of membership of a given input value to that label. Because the membership functions of adjacent labels overlap, this results in the weighted and simultaneous membership to multiple labels of each input value, the degree of membership being equal to the value of the membership function.

Contrary to classical methods of converting numerical intervals into discrete partitions, the obtained fuzzy partitions are not disjoint but consist of overlapping domains, each of which consists of an independent fuzzy kernel and a shared transition region. Thus, the partitioning of a learning set into fuzzy attribute partitions involves both the identification of the partition domains, and the identification of the overlap boundaries. These tasks are often done heuristically, using an expert's experience. In this work, they were automated using an algorithm based on mathematical morphology [5]. The algorithm works by applying a sequence of antagonistic, but asymmetrical filtering operations to the input data, until fuzzy kernels are obtained where only representatives of one class exist for each.

Another difference between a classical and a fuzzy decision tree is the decision process that they use. Figure 1 illustrates two binary trees of the same height, where one uses sharp thresholds and the other fuzzy thresholds to process the input data. For the given input, applying the rules of binary inference for the first tree and of fuzzy

inference for the second, the conclusion reached by the first tree is that the input data corresponds to class 1 (with no possible assignment to class 0). On the other hand, the fuzzy decision tree leads to the conclusion that the input corresponds to class 0 with truth-value 0.65 and class 1 with truth-value 0.3.

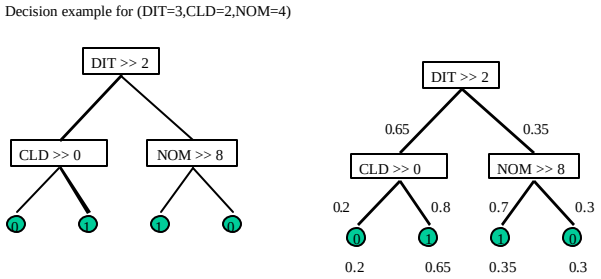

Figure 1. Classification using binary inference (left) and fuzzy inference (right)

3. Application to the prediction of evolvability

As stated in the first section, our hypothesis is that inheritance aspects may serve as indicators of library evolvability (interface stability). More specifically, we think that there is a relation between some inheritance metrics and a measure of OO library interface stability. As we are concerned with class libraries, we studied this hypothesis at the class level.

3.1. Categorizing changes in library interfaces

Changes in object-oriented software were widely studied from the perspective of their impact (see for example [1] and [3]). In these three projects, the authors were interested in an exhaustive classification of changes to study their impact on the components of the software. In the present work, we were specifically interested in the impact that changes among the versions of a class library have on systems that use a given version of the library and that will upgrade to newer versions.

In this respect, we identified two categories of changes at the class level, each one organized into types. Let's C_i be the interface of a class C in version i of the library and C_{i+1} be the interface of C in the version $i+1$. The two categories of changes for C are: (A) The interface C_i is no longer valid in version $i+1$ and (B) C_i is still valid in version $i+1$. *A* happens when: C is removed (1), some public (2), protected (3), private (3) members of C are removed. *B* happens when the interface remain unchanged (5) or augmented with new methods (6).

The types of changes (1 to 6) are ranged from worst to best (5 and 6 being equal) according to the degree of impact of each type.

3.2. Inheritance metrics

To best capture the different facets of the inheritance process, we selected 11 existing counting metrics (Table 1).

Symbol	Name
AID	Average inheritance depth
CLD	Class to leaf Depth
DIT	Depth of Inheritance Tree
NMA	Number of methods added
NMI	Number of methods inherited
NMO	Number of methods overridden
NOM	Number of methods
NOA	Number of ancestors
NOC	Number of children
NOD	Number of descendants
NOP	Number of parents

Table 1. Counting inheritance metrics

As some of the counting metrics presented in Table 1 did not sufficiently reflect the inheritance aspects that we were concerned with, we replaced some of them by the calculated metrics shown in Table 2.

Symb	Derivation rule	Name
PLP	DIT/(CLD+DIT)	Position in the longest path
PMA	NMA/NOM	Percentage of methods added
PMI	NMI/NOM	Percentage of methods inherited
PMO	NMO/NOM	Percentage of methods overridden

Table 2. Calculated inheritance metrics

3.3. Data collection

To build the estimation model, we used three versions of a C++ class library called OSE [6]. Version 4.3 of the library contains 120 classes while version 5.2 contains 126. For each of the 246 classes (120 +126), we extracted the change type and the values for the inheritance metrics. Then, we randomly selected 75% of the classes to serve in the learning process and 25% for testing the generated evolvability model. We consider, in our experiment, the change categories (A and B) in addition to the change types.

Using the extracted data, we derived 4 models by fuzzy-based learning: A2 for the 2 categories of changes and the counting metrics of Table 1, A6 for the 6 types of changes and the counting metrics of Table 1, R2 for the 2 change categories and the set of metrics obtained by replacing the metrics AID, DIT, CLD, NMA, NMO and NMI by the metrics of Table 2, and R6 for the 6 types of changes and using the same metrics as model R2.

When we examined the obtained models, we were surprised by the absence of class position metrics (DIT, AID, CLD and PLP). In other words, and within the limits of generalizing our results, these metrics (inheritance aspect) do not appear to be good indicators of the class/library evolvability. All the others metrics appear at least in one of the four models and, thus, they were retained as potential indicators.

Additionally, in order to compare our fuzzy approach to more traditional ML techniques, we also built the four models above using C4.5 and RoC (see section 2.1). All three approaches yielded the same results regarding the relevance of the proposed metrics as indicators. In this respect, the use of a fuzzy decision tree did not appear to bring improvements over existing ML techniques.

Another comparison criterion is the estimation accuracy rate when using the learning data and the test data. From our experience, the accuracy rate falls dramatically when we go from the learning data to the test data.

When using learning data, C4.5 presents the higher estimation accuracy rates for all four models while the fuzzy approach has comparable rates in most of the cases as shown in Figure 2. RoC provides the lowest rates.

When using the test data, the fuzzy-based algorithm has the best rates in the majority of cases while the rates of C4.5 drop by about 12% in three out of the four models. RoC maintains its rates (see Figure 3).

The good results of the fuzzy-based approach can be explained by the fact that uncertainty in considered in the decision process.

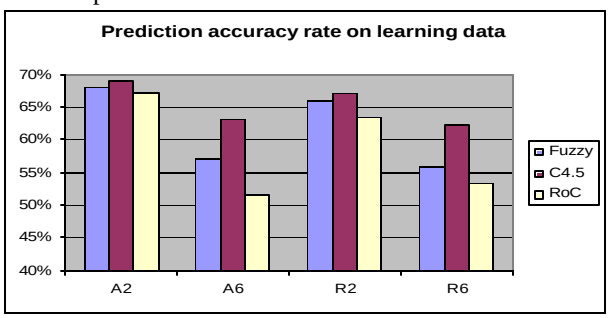

Figure 2. Estimation accuracy rate on learning data

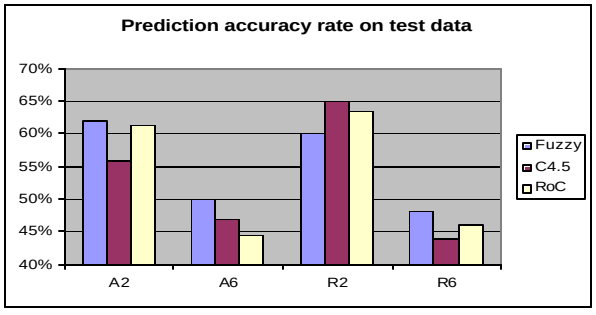

Figure 3. Estimation accuracy rate on test data

4. Conclusion

This paper presents a fuzzy-based approach for building evolvability estimation models for OO class libraries. Through an empirical study conducted on various versions of a commercial OO class library, we tried to answer two questions: Can inheritance aspects be considered as good indicators for class library interface stability? And Does fuzzy-based learning improves the quality of the estimation models?

If we analyze the results, we can say that the answer to question 1 comprises uncertainties. Yes, if we consider that the obtained models show that some aspects like types of methods, and the ancestors/descendents have a relationship with the categories of changes. No, if we consider that our sample is not representative enough to generalize our results and if we consider that the estimation rates are not as high as suitable (around 60%).

The response to question 2 is definitely yes and from two perspectives. First, the threshold values in C4.5 and the other "classical" techniques are too specific to the learning sample to easily generalized to the other libraries. This explains the difference between the learning and the test rates. By changing the threshold values to intervals, we increase the chance to have better estimation and to capture trends rather than specific values. The second perspective is that we use a simplified algorithm for building estimation models. We are convinced that by exploring the potential of fuzzy logic (B trees rather than binary trees, better fuzzification of the inputs, etc.), the results will be more significant.

References

[1] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen, "Change impact identification in object oriented software maintenance", Proc. of IEEE International Conference on Software Maintenance, 1994.

[2] P. Langley, W. Iba, and K. Thompson, "An analysis of Bayesian Classifiers". In Proc. of the National Conference on Artificial Intelligence, 1992.

[3] L. Li, A. J. Offutt, "Algorithmic Analysis of the Impact of Change to Object-Oriented Software". Proc. of IEEE International Conference on Software Maintenance, 1996.

[4] Y. Mao, H. A. Sahraoui and H. Lounis, "Reusability Hypothesis Verification Using Machine Learning Techniques: A Case Study", Proc. of IEEE Automated Software Engineering Conference, 1998.

[5] C. Marsala, and B. Bouchon-Meunier. "Fuzzy partioning using mathematical morphology in a learning scheme". In Proceedings of the 5th Conference on Fuzzy Systems, 1996.

[6] OSE, OSE Online Documentation, Dumpleton Software Consulting Pty Limited, 1999. Available in http://www.dscpl.com.au/ose-6.0/.

[7] T. M. Pigoski, "*Practical Software Maintenance*", Wiley Computer Publishing, 1997.

[8] R. S. Pressman, "*Software Engineering, A Practical Approach*", fourth edition, McGraw-Hill, 1997.

[9] J.R. Quinlan, "*C4.5: Programs for Machine Learning*", Morgan Kaufmann Publishers, 1993.

[10] M. Ramoni, and P. Sebastiani, "Parameter estimation in Bayesian networks from incomplete databases". *Intelligent Data Analysis Journal*, 2. 1998.

[11] H. Tanaka, T. Okuda, and K. Asai, "Fuzzy information and decision in statistical model". In *Advances in Fuzzy Set Theory and Applications*, pages 303-320. North-Holland, 1979.

[12] L. A. Zadeh, "Probability measures of fuzzy events". *Journal Math. Anal. Applic.*, 23. reprinted in *Fuzzy Sets and Applications: selected papers by L. A.Zadeh*, pp. 45-51, 1968.