Using Fuzzy Threshold Values for Predicting Class Libraries Interface Evolution

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

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

Hakim Lounis
CRIM, Canada
hlounis@crim.ca

Abstract

This work presents a technique to circumvent one of the major problems associated with building and applying techniques to build software quality estimation models, namely the use of precise metric thresholds values; we used a fuzzy binary decision tree to investigate the stability of a reusable class library interface, using structural metrics as stability indicators. To evaluate this new approach, we conducted a study on different versions of a commercial C++ class library. The obtained results are very promising when compared to those of two classical machine learning approaches, Top Down Induction of Decision Trees and Bayesian classifiers.

Keywords

Software reusability, quality assessment, machine learning, fuzzy logic

1. Introduction

Object-oriented software products are becoming more complex and time consuming, so is also the writing of newer versions. Pressman estimated at 60% the part devoted to maintenance in the total effort of the software development industry [19], from which 80% is devoted directly or indirectly to software evolution (adaptive and perfective maintenance) [18]. In spite of the benefits of object-oriented technology, OO class libraries, like the majority of software systems, are not exempt from this rule. Moreover, their evolution must take into account an additional constraint: to preserve, as much as possible, the compatibility among versions.

In this respect, it has become important to develop tools that allow the prediction of class evolvability through the symptomatic detection of potential instabilities during the design phase of such libraries. This may help avoid later problems.

In our context, we define evolvability as 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 evolution of a system that uses a library is directly influenced by the evolvability of the library. For instance, a system that uses version i of a library can easily be upgraded with version i+1 of the same library if the new version preserves the interface of the older one.

In this paper, we study the hypothesis that the inheritance aspects of an OO class library may be good indicators of its capacity to evolve. To this end, we propose a fuzzy logic-based approach for building and assessing evolvability estimation models. In a first step, we used two well-known techniques of machine learning, Top Down Induction of Decision Trees (TDIDT) using C4.5 [21] and Bayesian classification using RoC [12]. Then, we tried a new approach using fuzzy logic in an attempt to improve the obtained prediction accuracy rates by solving the problem of thresholds values.

2. Predicting interface evolution using inheritance aspects

An important part of the quality characteristics of software products is not directly measurable a priori. As a result, empirical investigations of measurable internal attributes and their relationship to external quality characteristics are a crucial issue for improving the assessment of a software product quality [10]. In this context, a large number of object-oriented (OO) measures have been proposed in the literature (see for example [4], [7], [14] and [3]).

Basili & al. show in [1] that most of the metrics proposed by Chidamber and Kemerer in [7] are useful for predicting the fault-proneness of classes during the design phase of OO systems. In the same context, Li and Henry showed that the maintenance effort could be predicted with combinations of metrics collected from the source code of OO components [14].

In the case of reusable components, Demeyer and Ducasse show in [9] that, for the particular domain of OO frameworks, size and inheritance metrics are not reliable to detect problems, but are good indicators for the stability of a framework. Basili & al. [2] conducted a study to model and understand the cost of rework for a library of reusable software components. A predictive model of the impact of error source on rework effort was built. In the same vein, Price and Demurjian [20] presented a technique to analyze and measure the reusability of OO designs; a set of eight metrics were derived from the combination of two classifications: general vs. specific and related to other classes vs. unrelated. These metrics would help evaluate OO systems from a reuse standpoint. For example, a dependency from a General class to another General class in related hierarchies is good for reuse, while a dependency from a General class to a Specific class in related hierarchies is bad for reuse.

In the past, our team explored both statistical and machine learning techniques as modeling approaches for software product quality. For instance, we have proposed a set of models to measure reusability [15] and class fault-proneness [5]. In [8], we conducted an empirical study of different ML algorithms to determine their capability of generating accurate correctability models. The study was accomplished on a suite of very-well known, public-domain ML algorithms belonging to three different families of ML techniques. The algorithms were compared in terms of their capability to assess the difficulty of correct Ada faulty components.

2.1 Working hypothesis

Because, we cannot easily measure the ability of a software product to evolve in a direct way, starting from its initial version, an indirect approach is to perform the assessment

using the relationships that may exist between evolvability and measurable characteristics such as size, cohesion, coupling or inheritance.

In this work, we focused our attention on how inheritance aspects can be good indicators of the interface evolution of an OO class library. More specifically, we investigated whether there is a causal relationship between some inheritance metrics, defined below, and the stability of OO library interfaces.

2.2 Identifying changes in library interfaces

Changes in object-oriented software were widely studied from the perspective of their impact. Kung et al. [11] propose a set of types of changes in an OO class library. Changes can concern data, a method, a class or the class library. 25 types changes are identified. In the same way, Li and Offutt define in [13] another set of change types for OO software. Changes are classified in two categories: change on a method (7 types of changes) and change on a data member (6 types of changes). More recently, Chaumum et al. [6] adapt this classification by adding changes on classes; 13 types of changes are identified.

In these three projects, the authors were interested in an exhaustive classification of changes to study their impact on 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 are upgraded to the next version.

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 $i+1$. This happens in four cases:
    1. C is removed
    2. $C_{i+1} = C_i$ – some public members
    3. $C_{i+1} = C_i$ – some protected members
    4. $C_{i+1} = C_i$ – some private members

B. The interface C_i is still valid in version $i+1$. This happen in two cases
    5. $C_{i+1} = C_i$
    6. $C_i \subset C_{i+1}$

Types of change (1 to 6) are ranged from worst to best (5 and 6 being equal) according to the degree of impact of each type. For example, the deletion of a class has a more serious impact than the deletion of a subset of its protected methods. Also, the types are exclusive: The change of class is classified into type k only if it cannot be classified into the $k-1$ previous types. For example, if, for a class C, some public methods are deleted and some other public methods are added, C belongs to type 2 and not to type 6. Finally, if a class is renamed, this is considered as a deletion of the class (type 1) and the creation of a new class. In the same way, a change in a method signature is considered as a method deletion. A scope change that narrows the visibility of a method (from public to protected or private and from protected to private) is considered also as a method deletion.

2.3 Defining the inheritance metrics

Three aspects of inheritance that may influence the evolution of a class interface are: (1) the location of the class in the inheritance tree; (2) the ancestors and descendants of the class; (3) the addition, inheritance and overwriting of methods. Each of these aspects will be studied in the case of simple inheritance.

Since the location of a class may be defined with respect to either the root of the inheritance tree or a leaf, we used two metrics, DIT and CLD (see Table 1), to specify its value. On the other hand, it may be more interesting to measure the location of the class relative to the longest path containing the class. Indeed, the information that a class is in the third level of inheritance out of a path of 8 levels is more meaningful than just saying the class is in the third level. This led us to define an additional metric, PLP, to provide this information.

The ancestors and descendents of the class were measured using three standard metrics[1], NOC, NOP and NOD (see Table 2 for definitions).

Finally, counting the new and the inherited methods was accomplished with the following metrics: NMA, NMI, NMO and NOM (see Table 3 for definitions). In the same way as for the location parameter, the percentages of added, inherited and overridden methods, PMA, PMI and PMO, were introduced to possibly provide more useful information that than just counting absolute numbers.

Symbol	Name	Comments
DIT	Depth of Inheritance Tree	Measures the size of the longest path from a class to a root class within the same inheritance tree.
CLD	Class to leaf Depth	Measures the size of the longest path from a class to a leaf class within the same inheritance tree.
PLP	Position in the longest path	DIT/(CLD+DIT)

Table 1. Class location metrics

Symbol	Name	Comments
NMA	Number of methods added	New methods in a class
NMI	Number of methods inherited	Methods inherited and not overridden
NMO	Number of methods overridden	Methods overridden
NOM	Number of methods	NMA + NMI +NMO
PMA	Percentage of methods added	NMA/NOM
PMI	Percentage of methods inherited	NMI/NOM
PMO	Percentage of methods overridden	NMO/NOM

Table 2. Class methods-related metrics

[1] We do not consider the number of ancestors (NOA) since it is equal to DIT in the case of simple inheritance

Symbol	Name	Comments
NOC	Number of children	
NOD	Number of descendants	
NOP	Number of parents	NOP $\in \{0, 1\}$ in the case of simple inheritance

Table 3. Class ancestors/descendents metrics

3. Machine learning techniques

In previous work, we have privileged the use of ML algorithms in order to build software quality predictive models. Our raison was that real-life software engineering data are incomplete, inexact, and often imprecise; in this context, ML could provide good solutions. Another reason was that, somehow, ML produces predictive models with superior quality than models based on statistical analysis. ML is also fairly easy to understand and use. But, perhaps the biggest advantage of a ML algorithm –as a modeling technique- over statistical analysis lies in the fact that the interpretation of production rules is more straightforward and intelligible to human beings than principal components and patterns with numbers that represent their meaning.

In most ML techniques, the classification process depends on threshold values that are derived from a learning set. This dependency creates a problem in the light of the representativity of the training samples, which, often, do not reflect the variety of real-life systems. In this respect, what is need is not the determination of specific thresholds but the identification of trends.

Another problem concerns the classification problem itself. 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.

To address these concerns, we propose the fuzzy logic approach of the next section

3.1 A fuzzy logic-based approach

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. While this approach works well for disjoint classes where different categories can be separated with clearly defined boundaries, it 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;

– Simultaneously fire several rules.

These possibilities are not available from algorithms such as C4.5, RoC, and most algorithms that rely on statistics or classical information theory to build the decision tree. In each case, the obtained tree leads to a set of rules of which only one is validated at a time, and where the antecedent of each rule is evaluated to be either true of false, leading to a consequent that is also either true or false. Because each antecedent consists of threshold comparisons to determine whether a given input falls within a decision region, the end result is that only one of the leaves in the tree will be reached at any given time, all the other leaves being ignored.

On the other hand, the use of a fuzzy decision process allows for two or more rules to be simultaneously validated with gradual certainty and the end result will be the outcome of combining several partial results, each contributing its weight to the decision process.

The creation of a fuzzy decision tree follows the same steps as that of a classical decision tree: a training set of examples is used in conjunction with a set of attributes to define the tree based on some metric. Then partitions of the attributes are defined and a chain of if-then rules is applied to subsequent inputs in *modus-ponens* fashion to identify a given class. The differences between the two approaches stems from the metrics used, the way partitions are created and the way the obtained tree is interpreted.

3.1.1 Creation of a fuzzy decision tree

As we obtained better results using the C4.5 algorithm over RoC, we decided to study a fuzzy version of C4.5. The TDIDT approach is easily transplantable 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 [22]. For a class C with a set of values $\{c_k\}$, 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 [23]:

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

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 (figure Figure 1).

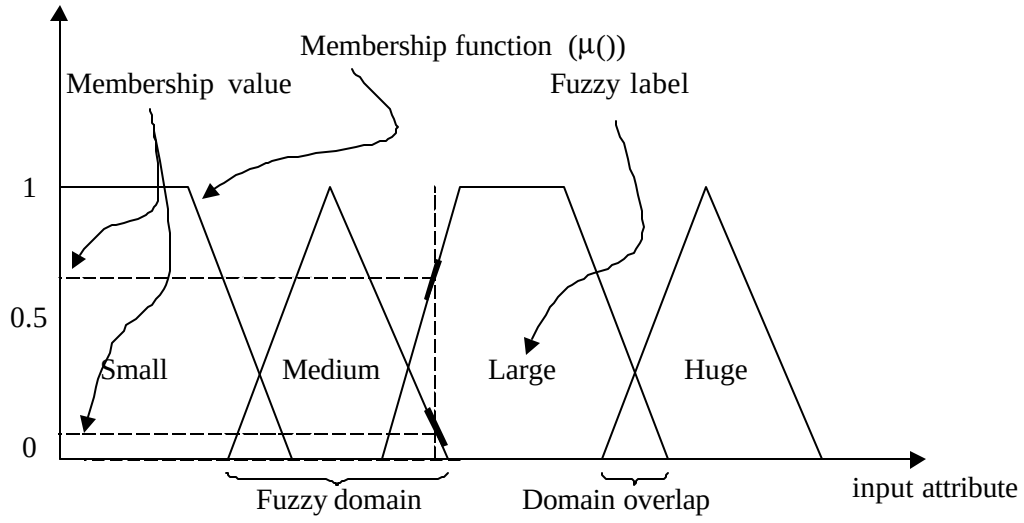


Figure 1. Basic concepts of fuzzy logic

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 [16]. 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.

3.1.2 Fuzzy tree inference for object classification

Another difference between a classical and a fuzzy decision tree is the decision process that they use. Figure 2 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[2].

[2] The results where obtained by using the minmax algorithm : minimum truth value along each tree path, maximum truth value for each end leaf.

Decision example for (DIT=3,CLD=2,NOM=4)

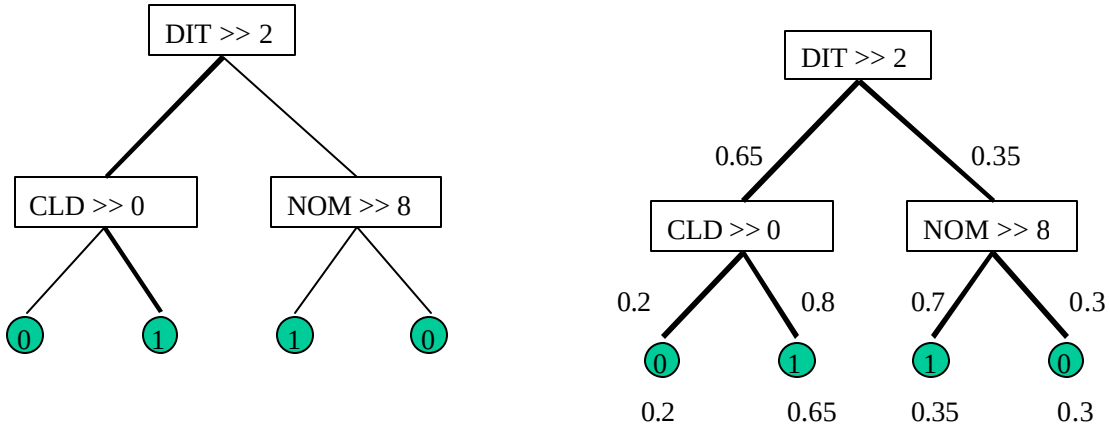


Figure 2. Classification using binary inference (left) and fuzzy inference (right)

The obtained fuzzy results may be defuzzified by computing the center-of-gravity of classes 0 and 1 considered as singletons (i.e. by computing the average of classes 0 and 1, weighted by their truth values) and then by choosing the class that is closest in value to the obtained COG. Alternately, we may simply select the class with the maximum truth-value. This is the approach used in this work as both methods of defuzzification yield the same result in the case of a decision tree with two classes.

4. Experiment

As stated in the first section, our hypothesis is that inheritance aspects may serve as indicators of library interface stability or, more precisely, that there is a relation between some inheritance metrics and a measure of OO library interface stability.

4.1 Data collection

To build the estimation model, we used three versions of a C++ class library called OSE [17]. 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.

Looking at the distribution of the cases (classes) by change types given in table Table 4, we notice that change types 0, 2 and 4 are not sufficiently represented. This observation led us to also consider, in our experiment, the change categories (A and B), as additional factors (see table Table 5).

Change type	Learning data	Test data	Total
0	2	2	4
1	35	13	48

2	2	1	3
3	46	17	63
4	6	3	9
5	89	30	119
Total	180	66	246

Table 4. Distribution of classes by change types

Categories	Learning data	Test data	Total
0 or A	88	30	118
1 or B	95	33	128
Total	183	63	246

Table 5. Distribution of classes by change categories

4.2 Building interface evolution models

We conducted experiments using both absolute metrics (DIT, CLD, NOC, NOP, NOD, NMA, NMI, NMO and NOM) and relative metrics (PLP, PMA, PMI, PMO) substituted for the corresponding absolute metrics. In addition we looked at their effect on both change types and categories. This led us to build four prediction models with each of the three ML techniques that were described above. The models were as follows:

1) Model A2 based on the 2 categories of changes and the absolute metrics.
2) Model A6 based on the 6 types of changes and the absolute metrics.
3) Model R2 based on the 2 change categories and the set of metrics obtained by combining absolute and relative metrics.
4) Model R6 based on the 6 types of changes and using the same metrics as in model R2.

4.3 Results and discussion

Figure 3 presents one of four decision trees obtained by using the fuzzy logic approach, corresponding to model R2. Each node contains a condition of classification relating to a metric and an interval which defines the values for which there is an uncertainty on the truth of the condition (see section 3.1.1).The remaining 15 models are not shown for lack of space but follow similar patterns.

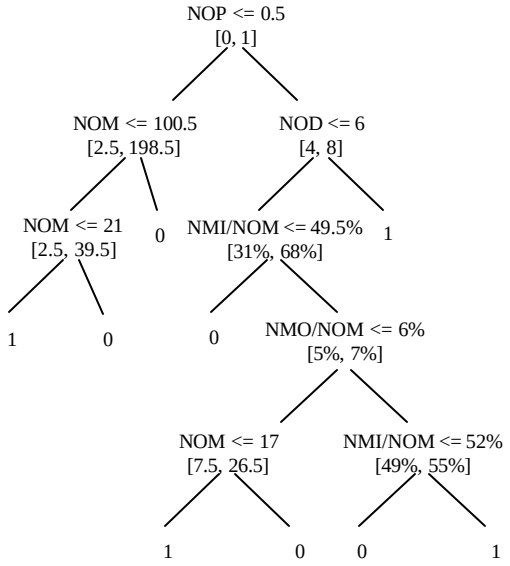
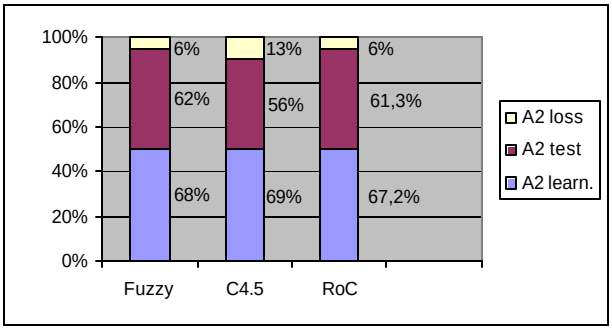
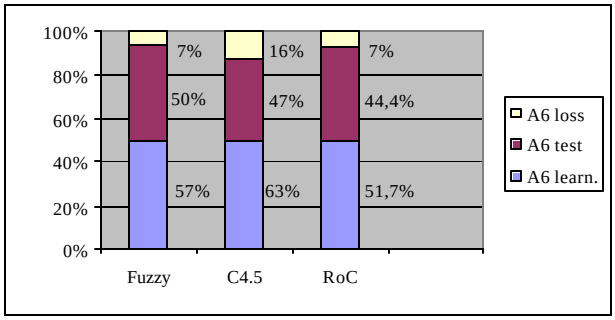


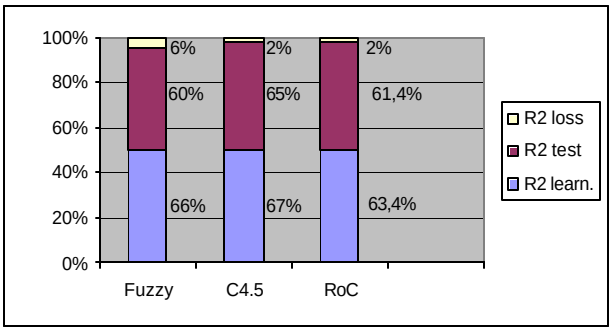
Figure 3. An example of fuzzy decision tree (evolvabiliy estimation model)

One striking observation that can be made from this figure is the absence of class location metrics (DIT, CLD and PLP). This was true for all the obtained models, leading to the conclusion, within the limitations of our training set representativeness, that these metrics do not appear to be good indicators of class interface stability between consecutive library versions. All the others metrics appeared at least in one of the four models associated with each technique and, therefore, were retained as potential indicators.

As all approaches yielded the same results regarding the relevance of the proposed metrics as indicators, the use of a fuzzy decision tree did not appear to bring improvements over existing ML techniques.







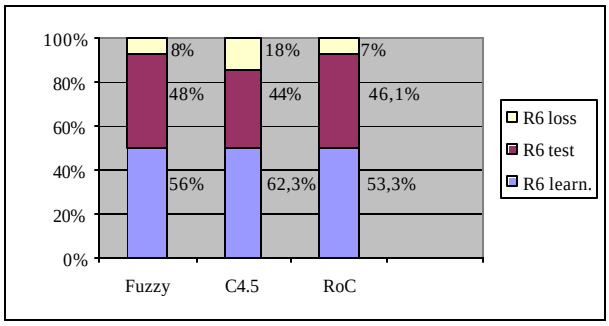


Figure 4. Estimation accuracy rates of class interface stability

We also compared the three techniques form the standpoint of estimation accuracy rates. The comparison was made using the computed estimation accuracy rates obtained with both the training data and the test data. In addition, we compared the loss of accuracy when moving form the learning to the test data.

Our results show that C4.5 presents the higher estimation accuracy rates for all four models while the fuzzy approach has comparable rates in most cases as shown in Figure 4; 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 4).

Consequently, the results shows that the fuzzy technique improves the estimation accuracy rates either from the perspective of stability as we move from the training to the test data (in comparison to C4.5), or from that of numerical value (in comparison with RoC). This can be explained by the facts that the fuzzy approach modifies C4.5 by keeping its inherent strength at identifying relevant indicators and removing the inconvenience of using absolute threshold values.

5. Conclusion and future directions

This paper presented a fuzzy-based approach for building interface stability estimation models for OO class libraries. Through an empirical study conducted on various versions of a commercial OO class library, we tried to answer the two following questions:

1. Can inheritance aspects be used as indicators for class library interface stability?

2. 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, like the used models, comprises uncertainties: Yes, if we consider that the obtained models show that aspects such as types of methods, and the ancestors/descendents have a relationship with the categories of changes. No, if we consider that our sample may not be representative enough to generalize our results and if we consider that the obtained estimation rates are still not as high as desired (about 60%).

The response to the second question is definitely yes. First, the threshold values in C4.5 and the other “classical” techniques are too specific to the learning sample to be easily

generalized. This explains the difference between the learning and the test rates. By changing the threshold values to intervals, we capture trends rather than specific values, thereby increasing the estimation accuracy rates. Also, this work uses a simple fuzzy algorithm for building our estimation models. The use of more comprehensive algorithms that use fuzzy logic to its full potential (B trees rather than binary trees, better fuzzification of the inputs, etc.), would probably yield more significant results.

References

[1] V. R. Basili, L. Briand & W. Melo, "How Reuse Influences Productivity in Object-Oriented Systems". *Communications of the ACM*, Vol. 30, N. 10, pp104-114, 1996.

[2] V. R. Basili, S. E. Condon, K El Emanm, R. B. Hendrick, and W. Melo, "Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components". In Proc. of 19th International Conference on Software Engineering, 1997.

[3] J.M. Bieman, B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System", in Proc. ACM Symp. Software Reusability (SSR'94), 1995.

[4] L. Briand, P. Devanbu, W. Melo, "An Investigation into Coupling Measures for C++", In Proc. of 19th International Conference on Software Engineering, 1997.

[5] L. Briand, J. Wüst, S. Ikonomovski & H. Lounis, "Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study". In proceedings of the 21st IEEE International Conference on Software Engineering, 1999.

[6] M. A. Chaumun, H. Kabaili, R. K. Keller, F. Lustman, and G. St-Denis, "Design Properties and Object-Oriented Software Changeability". In Proceedings of the Fourth Euromicro Working Conference on Software Maintenance and Reengineering, 2000.

[7] S.R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, 20 (6), 476-493, 1994.

[8] M.A. De Almeida, H. Lounis & W. Melo. "An Investigation on the Use of Machine Learned Models for Estimating Software Correctability". In *the International Journal of Software Engineering and Knowledge Engineering*, p. 565–593, vol. 9, number 5, October 1999.

[9] S. Demeyer, S. Ducasse, "Metrics, Do they really help ?", In Proc. of LMO, 1999.

[10] N. E. Fenton N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System", To appear in *IEEE Transactions on Software engineering*, 2000.

[11] 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.

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

[13] 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.

[14] W. Li, S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, 23 (2), 111-122, 1993.

[15] 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.

[16] 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.

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

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

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

[20] M. W. Price and S. A. Demurjian, "Analyzing and Measuring Reusability in Object-Oriented Design". In Proc. of OOPSLA'97, 1997.

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

[22] 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.

[23] 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.