Building Quality Estimation models with Fuzzy Threshold Values

Houari A. Sahraoui* — Mounir Boukadoum — Hakim Lounis****

**Département d'Informatique et recherche opérationnelle opérationnelle*
Université de Montréal
CP 6128 succ Centre-Ville, Montréal QC H3C 3J7, Canada
sahraouh@iro.umontreal.ca

*** Département d'informatique*
Université du Québec à Montréal
Case postale 8888, succursale Centre-ville, Montréal QC H3C 3P, Canada
{mounir.boukadoum, hakim.lounis}@uqam.ca

ABSTRACT: *This work presents an approach to circumvent one of the major problems with techniques to build and apply software quality estimation models, namely the use of precise metric thresholds values. We used a fuzzy logic based approach 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 three versions of a commercial C++ class library. The obtained results are very promising when compared to those of two classical machine learning (ML) approaches, Top Down Induction of Decision Trees and Bayesian classifiers.*

RÉSUMÉ : *Ce travail présente une approche pour résoudre un des principaux problèmes reliés aux techniques de construction et d'application des modèles d'estimation de la qualité de logiciel, à savoir l'utilisation des valeurs- seuils précises pour les métriques. Nous avons utilisé une approche basée sur la logique floue pour étudier la stabilité de l'interface des bibliothèques de classes. Nous nous sommes basés sur les métriques structurelles comme indicateurs de stabilité. Pour évaluer cette nouvelle approche, nous avons entrepris une étude sur trois versions d'une bibliothèque de classes commerciale écrite en C++. Les résultats obtenus sont très prometteurs comparativement à ceux de deux approches classiques d'apprentissage,TDIDT et classificateurs bayésiens.*

KEY WORDS: *Quality assessment, machine learning, fuzzy logic, software reusability*
MOTS-CLÉS: *Évaluation de la qualité, apprentissage, logique floue, réutilisation*

1. Introduction

Object oriented (OO) design and programming have reached the maturity stage, OO software products are becoming more complex and time consuming. Pressman estimated at 60% the part devoted to maintenance in the total effort of the software development industry [PRE 97], from which 80% is devoted directly or indirectly to software evolution (adaptive and perfective maintenance) [PIG 97]. Quality requirements are increasingly becoming determining factors in the choice of design alternatives during software development. Indeed, the adopted solutions often depend on whether the designer (programmer) privileges reliability, maintainability or reusability. In this context, it is crucial to automate the detection of symptomatic situations (e.g. problematic constructs in the code and/or design) according to some quality characteristic. It is also crucial to find ways to propose alternatives that allow the reach of given quality requirements.

Several works have shown that metrics can successfully be used to measure the quality of a system (see, for example, [HEN 96] and [LOR 94]). Software measures have been extensively used to help software managers, customers, and users to assess the quality of a software product. However, most of the quality characteristics are not directly measurable a priori. For example, the maintainability of a software product (which can be measured by the maintenance effort) can only be measured after a certain time of use. On the other hand, even if structural metrics cannot measure these characteristics directly, they can be good indicators of them. In this case, we speak of estimation models. Many large software companies have intensively adopted estimation models to better understand the relationships between software quality and software product internal attributes, in order to improve their software development processes. For instance, software product measures have successfully been used to assess software maintainability and error-proneness. Large software organizations, such as NASA and HP, have been able to estimate costs and delivery time via software product measures.

Estimation models can take different forms depending on the building technique that is used. For example, they can be mathematical models (case of statistical techniques like linear and logistic regression). They can also be rule sets or decision trees (case of machine learning algorithms). In all cases, they allow affecting a value to a quality characteristic based on the values of a set of software measures, and they allow the detection of design and implementation anomalies early in the software life cycle. They also allow organizations that purchase software to better evaluate and compare the offers they receive.

In most techniques, the estimation process depends on threshold values that are derived from a sample base. This dependency raises the problem of representativity for the samples, as these often do not reflect the variety of real-life systems. In this respect, what is needed is not the determination of specific thresholds but the identification of trends. This work circumvents the problem of using precise metric thresholds values associated with the estimation models by replacing them with fuzzy thresholds.

The paper is organized as follow. Section 2 gives an idea of existing work in the field. In section 3, we briefly present two examples of techniques to build quality estimation models and we discuss their limits. Our fuzzy logic based approach is described in section 4. To better illustrate our contribution, we discuss in section 5 its application to the particular case of assessing class library interface evolution. In section 6, we introduce a new approach that combines fuzzy logic with domain specific knowledge to improve the estimation accuracy. Finally, section 7 provides our conclusions.

2. Related work

Since an important part of the quality characteristics of software products is not directly measurable a priori, empirical investigations of measurable internal attributes and their relationship to external quality characteristics are crucial for improving the assessment of a software product quality [FEO 00]. In this context, a large number of object-oriented (OO) measures have been proposed in the literature (see, for example, [BRI 97], [CHI 94], [LIH 93] and [BIE 95]).

Basili & al. show in [BAS 96] that most of the metrics proposed by Chidamber and Kemerer in [CHI 94] are useful for predicting the fault-proneness of classes during the design phase of OO systems. In the same context, Li and Henry have shown that the maintenance effort may be predicted with combinations of metrics collected from the source code of OO components [LIH 93].

In the case of reusable components, Demeyer and Ducasse show in [DEM 99] that, for the particular domain of OO frameworks, size and inheritance metrics are good indicators for the stability of a framework but are not reliable to detect problems. Basili & al. [BAS 97] 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 [PRI 97] 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.

Close to the technique that we have followed, Genero et al. [GEN 00] have proposed a fuzzy regression tree-based approach to empirically assess the quality of entity relationship diagrams, the dominant conceptual modeling method in the database community.

In the past, our team has explored both statistical and machine learning techniques as modeling approaches for software product quality. For instance, we have proposed models to measure reusability [MAO 98] and class fault-proneness [BRI 99]. In [DAL 99], we conducted an empirical study of different ML algorithms

to determine their capability to generate 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.

3. Machine learning based approaches

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 this section, we present two popular algorithms, C4.5 and RoC, which represent the Top Down Induction of Decision Trees (TDIDT) and Bayesian approaches to ML. We then outline their limitations when dealing with software quality estimation models.

Most of the work done in ML has focused on supervised ML 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 TDIDT approach [QUI 93]. We used it in many past works to generate estimation models in software engineering. This was the case in [MAO 98] 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, a decision tree generally represents the induced knowledge. C4.5 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. Usually the class attribute takes only the values {true, false}, or {success, failure}.

The key step of the algorithm is the selection of the "best" attribute to obtain compact trees with high predictive accuracy. A measure of entropy is used to measure how informative a node is. Given an attribute *A* that takes on values from a set $\{a_i\}_{i=1,\ldots,n}$ and a probability distribution $P=\{p(a_i)\}_{i=1,\ldots,n}$, where $p(a_i)$ is the probability that $A=a_i$, the information conveyed by this attribute is given by Shannon's entropy:

$$H(A) = -\sum_{i=1}^{n} P(a_i)\log_2\left(P(a_i)\right)$$

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.

Closer to probabilistic approaches, RoC is a Bayesian classifier (see [LAN 92]). 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 posterior 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.

The following formula corresponds to the probability of the class value C=cj given the set of attribute values ek={A1=a1k, …, Am=amk}:

$$p(c_j / e_k) = \frac{\prod_{k=1}^{m} p(a_{ik} / c_j)\, p(c_j)}{\sum_{h=1}^{c}\prod_{k=1}^{m} p(a_{ik} / c_h)\, p(c_h)}$$

The simplifying assumptions underpinning the Bayesian classifier are that the classes are mutually exclusive and exhaustive and that the attributes are conditionally independent once the class is known. Recent empirical evaluations have found the Bayesian classifier to be accurate (see [KOH96]) and very efficient at handling large databases (e.g., data mining tasks). 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 [RAM 98].

One of the great advantages 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, as well as with "classical" ML approaches in general, when applied to production software data, 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.

In both C4.5 and RoC, 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, identifying trends is the learning set may be more useful than the determination of specific thresholds. For instance, the previous ML algorithms could induce rules as the following: "if number of methods of a class C is greater than 20, then class C is hard to maintain". Is it then justified to discuss

this value of 20; what does it mean? Can a class that has 19 methods be considered, in this context, as similar to one with 20 methods? Are they simply two large classes or do we have to distinguish between them?

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.

4. A fuzzy logic-based approach (FLAQ)

The main cause for the problems outlined above is that, in 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 a comparison to determine whether a given input is above or below a threshold, 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 the simultaneous validation of all the rules; each input value is considered to be both above and below the corresponding threshold, but with gradual and typically different certainties. The end result will be the outcome of combining all of the 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. As a result, 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 stem from the metrics used, the way partitions are created and the way the obtained tree is interpreted.

4.1. *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 can easily be ported to the creation of fuzzy decision trees by using fuzzy entropy 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 [TAN 79]. For an attribute A with values $\{a_i\}_{i=1,\dots,n}$, fuzzy entropy is defined as:

$$H^*(A) = -\sum_{i=1}^{n} P^*(a_i)\log_2\left(P^*(a_i)\right)$$

where $P^*()$ usually stands for the fuzzy probability defined by Zadeh [ZAD 68]:

$$P^*(a_i) = -\sum_{j=1}^{m} \mu(e_j)P(e_j)$$

Fuzzy probabilities differ from normal probabilities in that they represent the weighted average of a set of values provided by a membership function μ. These values represent the degrees of membership of a value a_i to the different elements (labels), e_i, of a fuzzy set.

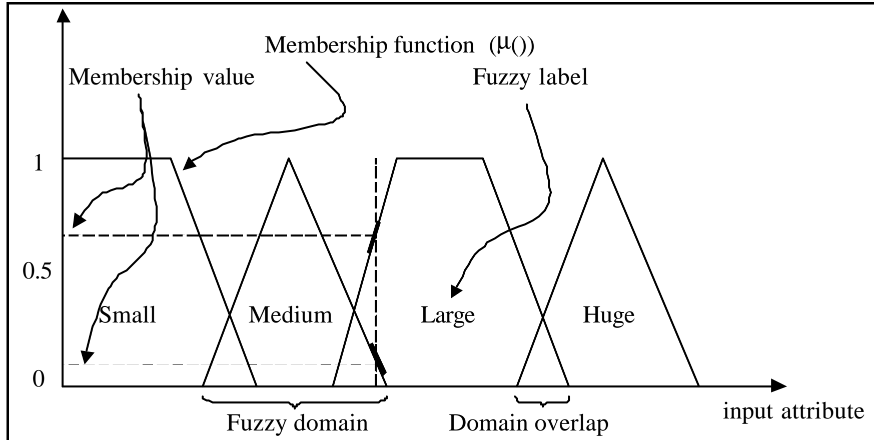

Figure 1. *Basic concepts of fuzzy logic*

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 1).

Contrary to classical methods of converting numerical intervals into discrete partitions, fuzzy partitions are not disjoint and consist, each, of an independent fuzzy kernel and a shared transition region. As a result, 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 a clustering algorithm based on mathematical morphology [MAR 96]. The algorithm works by applying a sequence of antagonistic, but asymmetrical filtering operations to the input data, until fuzzy kernels are obtained that mostly include representatives of one class each.

4.2. *Fuzzy tree inference for 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[1].

The obtained fuzzy results may be defuzzified by computing the center-of-gravity (COG) 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.

---

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

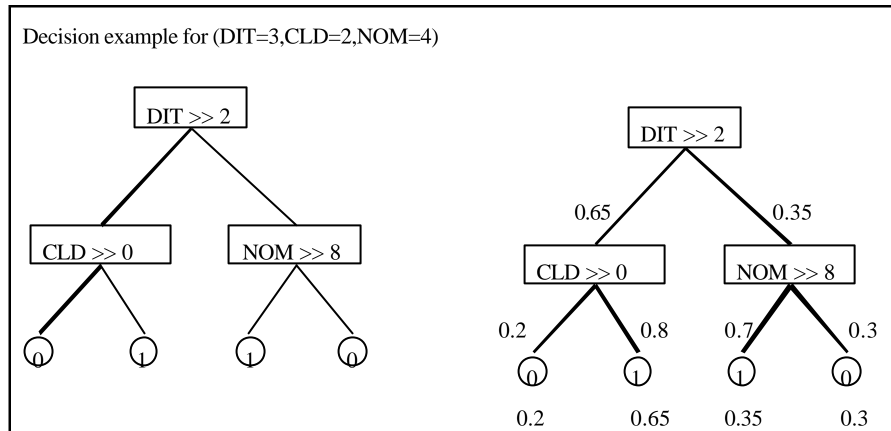

Figure 2. *Classification using binary inference (left) and fuzzy inference (right)*

5. Case study: predicting interface evolution using inheritance aspects

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.

5.1. *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.

5.2. *Identifying changes in library interfaces*

There have been extensively studies of the impact of changes on object-oriented software. Kung et al. [KUN 94] identify 25 types of changes that may occur in an OO class library. The changes may concern data, a method, a class or the class library. In the same way, in [LIO 96], Li and Offutt define another set of change types for OO software. Changes are classified in two categories: change of a method (7 types of changes) and change of a data member (6 types of changes). More recently, Chaumum et al. [CHA 00] have expanded this categorization to include changes on classes; 13 types of changes are identified.

In the previous projects, the authors were interested in an exhaustive classification of changes to study their impact on software in general. In our work, we were specifically interested in the impact that version changes in a class library may have on systems that use a given version of the library and that are upgraded to the next version. To this end, we identified two categories of changes at the class level, each one organized into types as follows: Let 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 version $i+1$. Then, 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. $C_i \quad C_{i+1}$

These types of change were ranked from worst to best 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. They were subsequently attributed numerical values in ascending order (1 to 6, 5 and 6 being equal), In addition, the types are conservative and exclusive: A 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 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 also considered as a method deletion.

5.3. *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.

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*

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.

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*

The ancestors and descendents of the class were measured using three standard metrics[2], 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
NOC	Number of children	
NOD	Number of descendants	
NOP	Number of parents	NOP {0, 1} in the case of simple inheritance

Table 3. *Class ancestors/descendents metrics*

5.4. *Application of FLAQ*

As stated in the section 5.1, 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.

5.4.1. *Data collection*

To build the estimation model, we studied three versions of a C++ class library called OSE [OSE 99], 4.3, 5.2 and 6.0, and we focused our attention on changes from version 4.3 to version 5.2, and from version 5.2 to version 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.

Looking at the distribution of the cases (classes) by change types given in table 4, we notice that change types 0, 2 and 4 are not sufficiently represented. This

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

observation led us to also consider, in our experiment, the change categories (A and B), as additional factors (see 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*

5.4.2. *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 using the FLAQ approach. 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.

5.4.3. *Results*

Figure 3 shows one of the four decision trees that were obtained by using FLAQ, the one 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 4.1). The remaining 3 models are not shown for lack of space but follow similar patterns.

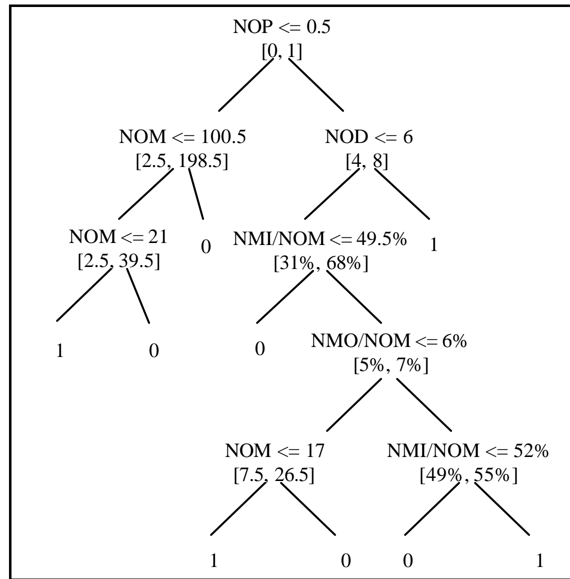

Figure 3. *An example of fuzzy decision tree (evolvabiliy estimation model)*

One observation that can be drawn from this figure is the absence of class location metrics (DIT, CLD and PLP). This was true for all of the obtained models and leads to the conclusion, within the limited size of our training set, that these metrics may not be good indicators of class interface stability between consecutive library versions. All the others metrics appeared at least in one of the four models and were, therefore, retained as potential indicators.

5.4.4. *Comparison with "classical" ML techniques*

To compare the performance of the models obtained with FLAQ with models obtained with C4.5 and RoC, we used the same data and built 4 models using each of these two techniques.

All three techniques 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.

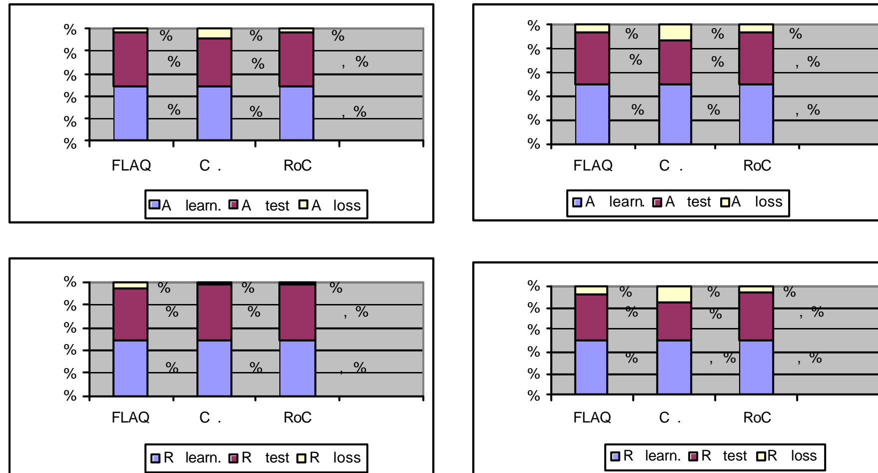

Figure 4. *Estimation accuracy rates of class interface stability*

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 for the learning data, C4.5 presents the higher estimation accuracy rates for all four models while FLAQ has comparable rates in most cases as shown in figure 4; RoC provides the lowest rates.

When using the test data, FLAQ 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.

6. Towards a domain-Knowledge based approach

As stated by Fenton and Neil in [FEN 00], most of the techniques used to build prediction models produce naïve models with a single level of decision. This is also true for our approach. For instance, the model of figure 3 shows a certain correlation

between the various inheritance metrics and interface stability, but it is weak at providing evidence for a causal relationship.

This limitation of naïve models is especially true when the variables under consideration are cognitively distant. As a result, these models are hard to use in making intelligent decisions for software refactoring or redesign.

To increase the efficiency of our fuzzy logic based approach, we propose to transform the single level decision models using a causal model approach. As a large part of the relevant data is missing from the sample used for learning, we propose to us domain-specific heuristics for this transformation. An analogy could be made with work in the ML community, where the opposition between weak domain, theory-based learning and strong domain, theory-based learning is stated. Obviously, a learning process based on an available theory domain produces a knowledge that fits more with the perception we generally have about knowledge.

6.1. *Overview*

The idea behind the new approach is to palliate the weaknesses of a naïve model at supplying causal relationships by enhancing its operation with heuristic rules. As shown in figure 5, rather than simply using a naïve prediction model, we add domain-specific heuristics to derive the fuzzy rules, in an attempt to create a causal prediction model.

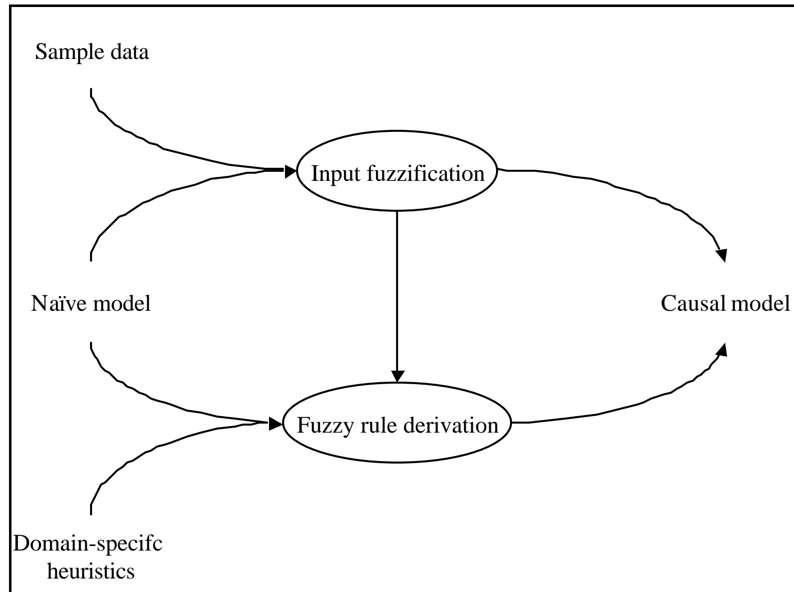

Figure 5. *Naïve to causal model transformation*

6.2. *Fuzzification and rule derivation*

Metric fuzzyfication in the naïve model is done the standard way, according to the algorithm below:

/ Fuzzification of the metrics in the naïve model */*

FOR EACH input metric Mi in the naïve model DO

FOR EACH condition "Mi comparator valuej" with a partition pj=[valuej1, valuej2] DO

1. Create a fuzzy label labelj

2. Derive the membership function of labelj from pj and the sample data

END DO

END DO

On the other hand, rules derivation is done both using the set of naïve rules and additional constraint based on heuristics. The algorithm is as follows:

/ Derivation of the set of naïve rules NR */*

FOR EACH path of the naïve model DO

1. *Derive a rule by replacing all the decision nodes by the corresponding fuzzy condition* "M1 labelj" with conjunctions.
2. *Add the derived rule to NR*

END DO

/ Derivation of the causal rules CR */*

FOR EACH rule ri in NR DO

Derive a set of rules CRi using domain specific heuristics such that:

a. *There exists a subset of rules ICRi where the conditions are fuzzy conditions of type "M1 labelj"*
b. *There exists a subset of rules FCRi where the conclusions are the estimation of the quality characteristic*
c. *The conditions of each rule in CRi –ICRi are conclusion of rules in CRi*
d. *Each rule in CRi represents a verifiable causal relationship*

END DO

Once these new rules are obtained, causal, multi-layered fuzzy decision trees can be built. We are presently working on such a model.

7. Conclusion and future directions

This paper presented a fuzzy-based approach to build interface stability estimation models for OO class libraries. Through an empirical study conducted on three 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. The answer is 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. It is 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.

Finally, this work used a simple fuzzy algorithm for building our estimation models. The use of more comprehensive algorithms that use fuzzy logic to its full potential (e.g. B-trees rather than binary trees, more comprehensive membership functions, etc.), and that make use of domain-based heuristics, would probably yield more significant results.

References

[BRA 95] BRAUN T., DIOT C., HOGLANDER A., ROCA V., An experimental user level of implementation of TCP, rapport de recherche no. 265. séptembre 1995, INRIA.

[BAS 96] 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.

[BAS 97] V. R. BASILI, S. E. CONDON, K EL EMANM, R. B. HENDRICK, & 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.

[BIE 95] J.M. BIEMAN, B.-K. KANG, "Cohesion and Reuse in an Object-Oriented System", in Proc. ACM Symp. Software Reusability (SSR'94), 1995.

[BRI 97] L. BRIAND, P. DEVANBU, W. MELO, "An Investigation into Coupling Measures for C++", In Proc. of 19th International Conference on Software Engineering, 1997.

[BRI 99] 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.

[CHA 00] 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.

[CHI 94] S.R. CHIDAMBER, C.F. KEMERER, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, 20 (6), 476-493, 1994.

[DAL 99] 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.

[DEM 99] S. DEMEYER, S. DUCASSE, "Metrics, Do they really help ?", In Proc. of LMO, 1999.

[FEN 00] N. E. FENTON, M. NEIL, "Software Metrics: Roadmap", In Proc. of the 22nd International Conference on Software Engineering, 2000.

[FEO 00] N. E. FENTON N. OHLSSON, "Quantitative Analysis of Faults and Failures in a Complex Software System", In *IEEE Transactions on Software engineering*, 26(8), 797-814, 2000.

[GEN 00] M. GENERO, L. JIMENEZ, M. PIATTINI, "Measuring the quality of entity relationship diagrams", In Proc. of 19th International Conference on Conceptual Modeling, 2000.

[HEN 96] B. HENDERSON-SELLERS, *"Object-Oriented Metrics: Mesures of Complexity",* Prentice-Hall, 1996.

[KOH 96] R. KOHAVI, "Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid". In Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining, 1996.

[KUN 94] D. KUNG, J. GAO, P. HSIA, F. WEN, Y. TOYOSHIMA, & C. CHEN, "Change impact identification in object oriented software maintenance", Proc. of IEEE International Conference on Software Maintenance, 1994.

[LAN 92] P. LANGLEY, W. IBA, & K. THOMPSON, "An analysis of Bayesian Classifiers". In Proc. of the National Conference on Artificial Intelligence, 1992.

[LIO 96] 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.

[LIH 93] W. LI, S. HENRY, “Object-Oriented Metrics that Predict Maintainability”, *Journal of Systems and Software*, 23 (2), 111-122, 1993.

[LOR 94] M. LORENZ, J. KIDD, *“Object-Oriented Software Metrics: A Practical Approach”*, Prentice-Hall, 1994.

[MAO 98] Y. MAO, H. A. SAHRAOUI & H. LOUNIS, “Reusability Hypothesis Verification Using Machine Learning Techniques: A Case Study”, Proc. of IEEE Automated Software Engineering Conference, 1998.

[MAR 96] C. MARSALA, B. BOUCHON-MEUNIER. “Fuzzy partioning using mathematical morphology in a learning scheme”. In Proceedings of the 5th Conference on Fuzzy Systems, 1996.

[OSE 99] OSE, OSE Online Documentation, Dumpleton Software Consulting Pty Limited, 1999. Available in http://www.dscpl.com.au/ose-6.0/.

[PIG 97] T. M. PIGOSKI, “*Practical Software Maintenance*”, Wiley Computer Publishing, 1997.

[PRE 97] R. S. PRESSMAN, “*Software Engineering, A Practical Approach*”, fourth edition, McGraw-Hill, 1997.

[PRI 97] M. W. PRICE AND S. A. DEMURJIAN, “Analyzing and Measuring Reusability in Object-Oriented Design”. In Proc. of OOPSLA’97, 1997.

[QUI 93] J.R. QUINLAN, “*C4.5: Programs for Machine Learning*”, Morgan Kaufmann Publishers, 1993.

[RAM 98] M. RAMONI, AND P. SEBASTIANI, “Parameter estimation in Bayesian networks from incomplete databases”. *Intelligent Data Analysis Journal*, 2. 1998.

[TAN 79] H. TANAKA, T. OKUDA, K. ASAI, “Fuzzy information and decision in statistical model”. In *Advances in Fuzzy Set Theory and Applications*, pages 303-320. North-Holland, 1979.

[ZAD 68] 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.