A Fuzzy Logic Framework to Improve the Performance and Interpretation of Rule-Based Quality Prediction Models for OO Software

Houari A. Sahraoui[1], Mounir Boukadoum[2],
Hassan M. Chawiche[1], Gang Mai[1], Mohamed Serhani[1]
[1]DIRO, Université de Montréal; [2]Université du Québec À Montréal
{sahraouh, chawichh, maig, serhanim}@iro.umontreal.ca; mounir.boukadoum@uqam.ca

Abstract

Current object-oriented (OO) software systems must satisfy new requirements that include quality aspects. These, contrary to functional requirements, are difficult to determine during the test phase of a project. Predictive and estimation models offer an interesting solution to this problem. This paper describes an original approach to build rule-based predictive models that are based on fuzzy logic and that enhance the performance of classical decision trees. The approach also attempts to bridge the cognitive gap that may exist between the antecedent and the consequent of a rule by turning the latter into a chain of sub rules that account for domain knowledge. The whole framework is evaluated on a set of OO applications.

1. Introduction

The use of software is now part of our everyday life and the process of developing it has gone through several technological changes. In the early days of computing, developing software applications was simply the process of writing procedures to perform given tasks. Since then, the rapid changes in hardware and user requirements, and the nature of the data to be processed have led to several efforts to create robust and maintainable software. Of the various paradigms that have been proposed, OO technology is the one that has gathered the most support, and the use of objects has considerably reduced the gap between user requirements and their implementation in software. Originally, OO was simply meant to build systems that closely match functional requirements, relegating specific implementation aspects to a secondary level. This has led to the development of more easily reusable and maintainable software. Since then, the success of OO has encouraged its use in newer, constantly evolving applications that require frequent changes at all levels of consideration. This is the case for telecommunications, E-commerce and Internet software.

One consequence of this evolution is that it is no longer enough that a system match functional specifications; it must be adaptable to future changes. Therefore, today's software must also satisfy non functional requirements such as those of maintainability, evolvability, etc. Unfortunately, the classical verification/validation techniques are not well adapted to these newer requirements, which usually cannot be directly assessed at the end of development. For instance, several years of operation are typically needed to determine if a given software is maintainable. What we need are predictive models that allow us to evaluate quality aspects starting from available structural aspects, and that help us intervene during the software development stage to perform preventive maintenance.

This paper describes a framework for building and using such quality prediction models. In section 2, we describe the main problems of current solutions; in section 3, we describe the issues associated with assessing software stability. Then, we describe how, starting from a classical rule-based system, we can circumvent its weaknesses, namely the use of numerical threshold values in the derived rules and the limited usefulness of the latter for decision support, by using fuzzy logic and by adding domain heuristics. We also provide an empirical validation of the framework by using it to predict the stability among versions of three different OO systems. Finally, we conclude with a discussion section.

2. Software quality prediction models issues

Most of the work done so far to build efficient and usable software quality estimation models falls in two families. The first one relies on historical data to achieve its goal (see for example [4], [1] and [8]). The quality of these models depends heavily on the quality of the used samples, which is usually poor in software engineering. Indeed, contrary to other domains, the small size and the heterogeneity of the samples make it difficult to derive widely applicable models. As a result, the models may capture trends, but do so by using sample-dependent threshold values [12]. Also, as stated by Fenton and Neil

[6], the majority of the produced models are naïve; they cannot serve for decision support during the software development process. This is because, often, the predictive variables and the quality characteristic to predict show no obvious causal link between them. The models behave as simple black boxes that take the predictive variables as input and the predicted variable as output.

The second family of work to build software quality estimation models uses knowledge extracted from domain-specific heuristics. The obtained predictive models use judgments from experts to establish an intuitively-acceptable causal relationship between internal software attributes and a quality characteristic (see for example [7]). Although they are adapted to the sought decision-making process, these models are hard to generalize because of a lack of widely acceptable common knowledge in the field of software quality.

Based on the preceding, it appears that a hybrid approach that combines the use of historical measurement data and of domain knowledge would circumvent the disadvantages of both.

3. Building rule-based prediction models

Thus, our need is for an inductive model that is not subject to the threshold values problem, and that can use domain-knowledge enhancements (which are inherently declarative) to help explain the causal relationship between predictive and predicted variables. One approach to achieve this is first to fuzzify an existing rule-based predictive model, or to create a fuzzy predictive model directly from the software metrics. The obtained fuzzy predictive model may then be enhanced by the inclusion of domain knowledge. In the following, we describe a technique to build such a model and show our solution to the threshold and naïve model problems.

3.1. Classical decision tree approach

One way to build a rule-based predictive model is the Top Down Induction of Decision Trees (TDIDT) technique, of which the C4.5 algorithm [11] is a typical representative. C4.5 uses a binary decision tree to represent the induced knowledge from a set of examples. Each example consists of a number of attribute/value pairs of which one is the class of the example. Often, this attribute only takes the values {true, false}, or {yes, no}.

The key step of the C4.5 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 $V=\{a_i\}_{i=1,\ldots,n}$, the information conveyed by this attribute is given by Shannon's entropy [13]:

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

where $p(a_i)$ is the probability that $A=a_i$ within the set V. 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.

```
DIT <= 1 :
|   DAM <= 0.7 : 1
|   DAM > 0.7 :
|   |   OCAEC <= 0 :
|   |   |   NOC <= 0 : 1
|   |   |   NOC > 0 :
|   |   |   |   NOC <= 2 : 1
|   |   |   |   NOC > 2 : 2
|   |   OCAEC > 0 :
|   |   |   NOC <= 1 : 1
|   |   |   NOC > 1 :
|   |   |   |   NOC <= 8 : 2
|   |   |   |   NOC > 8 : 1
DIT > 1 :
|   NOC <= 0 : 1
|   NOC > 0 :
|   |   OCAEC > 1 : 1
|   |   OCAEC <= 1 :
|   |   |   OCAEC > 0 : 1
|   |   |   OCAEC <= 0 :
|   |   |   |   DIT > 4 : 1
|   |   |   |   DIT <= 4 :
|   |   |   |   |   DAM <= 0.83 : 1
|   |   |   |   |   DAM > 0.83 : 2
```

Figure 1. The decision tree for stability prediction

Figure 1 provides an example of C4.5's output, where the stability of a class interface is predicted (1=instable, 2=stable) based on the values of various structural metrics (defined in Table 1). In Figure 1, each line represents a rule that either leads to another rule (absence of a number after the colon) or to a leaf. From the obtained tree, and after a pruning stage, a set of rules may be extracted from the different paths.

Table 1. Metrics definition

DIT	Depth of the Inheritance Tree
DAM	Data Accessibility Metric: the percentage of private and protected attributes in a class
NOC	Number of classes
OCAEC	Other Class Attribute Export Coupling: The use of a class to define other classes.
NAA	Number of Available Attributes
NAM	Number of Available Methods
NOP	Number of Polymorphic Methods

Rule 11: DIT > 1 $\Rightarrow$ class 1

Rule 1: DAM <= 0.7 $\Rightarrow$ class 1

Rule 6: OCAEC > 0 $\Rightarrow$ class 1

Figure 2. Pruned rules derived from the tree of Figure 1

3.2. Handling the threshold value problem with a fuzzy binary decision tree

As shown in Figure 2, the typical tree-based prediction model includes a set of rules consisting, each, of one or more conditions and a conclusion. A condition typically compares the numerical value of one of the structural metrics to a threshold value. If all the conditions are true, a value is assigned to the quality characteristic (the conclusion). For example, the rule

$$\text{NOM(c)} > 20 \Rightarrow \text{LM(c)} = 3$$

is read: *If the number of methods in class c is greater than 20, then c has a level of maintainability of 3.* The threshold value of 20 is normally derived form the training data that served to define the rule, and the value of 3 represents a value from a user-defined scale.

Since the value of 20 depends on the training data, it is difficult to generalize the previous rule to all classes. Still, most experts would agree with the following rule: *If the number of methods in a class c is big, then c is difficult to maintain.* Using the previous notation, this more general rule is:

$$\text{NOM(c)} = \text{BIG} \Rightarrow \text{LM(c)} = \text{DIFFICULT}$$

Thus, a solution to the generalization problem is simply to have the variables accept symbolic values instead of the regular numerical values, and to perform linguistic comparisons. This can be achieved by using fuzzy logic to turn the variables and the threshold values of the naïve model into fuzzy ones. Then, the rules of the model will no longer reference specific numerical values.

In such a fuzzy decision tree, the processing of input attribute values starts with the fuzzification of each attribute so that it becomes a linguistic variable that takes values from a discrete set of labels (e.g. “bigger”, “smaller”, etc.) Each label has an associated membership function that sets the degree of membership of a given numerical 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. In essence, the fuzzification process takes the different attribute values, which normally obey an order relationship (e.g. each one is either bigger, equal or smaller than a threshold), and replaces them with a set of tuples that represent the degree of membership of each value to different fuzzy labels. As a result, when performing a threshold comparison, the outcome is no longer a choice between “bigger”, “equal” or “smaller”, but is “all of them”. The impact of this on a binary decision tree is that the different paths are no longer mutually exclusive; they are simultaneously valid choices and it is only at the leaves that a decision has to be made on the conclusion to draw from the different outcomes, based on their respective truth values

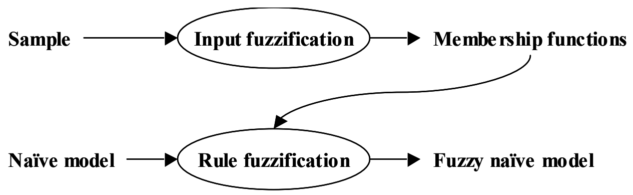

Figure 3. Fuzzification of a naïve model

3.2.1. Fuzzification of classical rules

Figure 3 illustrates the process: In a first step, the predictive and predicted variables are converted to fuzzy ones. Then, the naïve model’s rules are modified to use the new fuzzy variables. As a result, the naïve model’s antecedents and consequents become fuzzy statements.

3.2.1.1 Input fuzzification

It consists of assigning the input data to overlapping clusters of values that represent fuzzy sets. Many techniques exist in the literature to fuzzify quantitative attributes. Some of them are simple to use, others are more sophisticated [3]. In our approach, we start with the distribution of the measurement data. We begin by calculating, for each input (metric) value, its relative frequency. Then, we derive contiguous, overlapping, clusters from the resulting curve. The obtained clusters represent the labels (or linguistic values) of the input metric viewed as a fuzzy variable. In some cases, we must first preprocess the input data before deriving the frequency curve and the ensuing clusters. Thus, the algorithm for input fuzzification is as follows:

1. *Preprocess the input data to improve cluster identification if needed*
2. *Draw a frequency histogram of the input data*
3. *Identify the clusters and assign them to fuzzy sets, each identified by a label.*
4. *Define the cluster boundaries and associated membership functions with a suitable fuzzy clustering algorithm*

Table 2. Sample metric values for a set of classes

	DIT	NOC	NPA	NAA	…	OCAEC
Class 1	1	1	3	5		1
Class 2	2	3	4	10		0
…						
Class n	1	2	0	8		1

To illustrate this technique, consider the contents of Table 2, which shows various metric values associated with a set of classes. When looking at the frequency histogram, of metric OCAEC, it is difficult to see more than one cluster. Since most classes do not serve to define the attributes of other classes, the value of OCAEC is mostly nil. As a result, the first value in the histogram makes it difficult to interpret the other values. One solution to this problem is to apply a logarithmic transformation to the frequency values to boost the smaller ones and flatten the larger ones. The result, in this case, clearly reveals the existence of three clusters.

Based on the obtained clusters, we can define three labels for the OEAEC metric, *small*, *medium* and *large*. The next step in the fuzzification of OCAEC is to turn these clusters into fuzzy ones by defining overlapping membership functions for each of them. This involves the definition of the functions' boundaries and their shapes with a suitable fuzzy clustering algorithm. The c-means algorithm was used in this work [2].

The obtained membership functions can be simplified, for processing purposes, by approximating them with standard shapes (trapeze or triangle). The approximation is accomplished by drawing intersecting lines segments, tangent to the curves so as to define the membership functions by the obtained sequences of segments. Figure 4 shows the final memberships function shapes for the fuzzy sets that were defined for the values of OEAEC.

Thus, fuzzification of the OEAEC metric results in the creation of a fuzzy variable with the same name, and with three labels and associated membership functions. Each value of OEAEC may then be mapped to three membership values, one for each label.

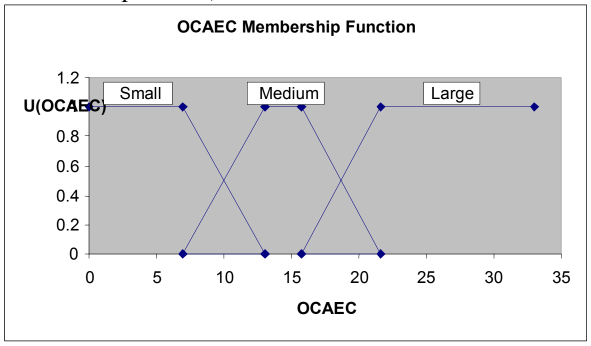

Figure 4. Membership functions for the OCAEC metric

3.2.1.2 Rule fuzzification

The fuzzification of a rule concerns both the antecedent and the consequent parts of the rule.

The antecedent is dealt with by looking at the fuzzified input data and determining which of the labels or combination thereof best matches the threshold value used in the condition of the rule. For example, if the condition is "*if number of methods greater than 20*", and the value of 20 is part of the input values that define the fuzzy label *large*, then an equivalent fuzzy condition is "*if number of methods is large*".

Fuzzification of the consequent is more straightforward. Since the conclusion of a rule is a classification, the desired mapping is one-to-one. For example, if the possible values for the level of maintainability of a class are 1, 2, 3 or 4, we can associate the labels "*very easy*", "*easy*", "*difficult*" and "*very difficult*" to them.

3.2.2. Fuzzy top-down induction of decision tree

The previous fuzzification process starts from a decision tree that is already defined. An alternative and potentially better approach is to use fuzzy means from the start in the definition of the decision tree. This calls for a review of the TDIT algorithm where fuzzy entropy replaces classical entropy and where a fuzzy clustering algorithm replaces the regular classification algorithm. We called the obtained algorithm FLAQ (for more details about FLAQ, see [12]).

Fuzzy entropy (also called star entropy) is an extension of Shannon's entropy where classical probabilities are replaced by fuzzy ones [13]. For an attribute A with fuzzy 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 $P^*()$ stands for the fuzzy probability defined by Zadeh [14]:

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

In this equation, $\mu_{a_i}(j)$ is the degree of membership of input value j to a_i, and $P_{a_i}(j)$ is the frequency of occurrence of j within a_i.

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. 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. They can also be performed using clustering algorithms such the c-means [2], the fuzzy k-means with extragrades [5], and mathematical morphology [9]. In the application that will be presented, the creation of fuzzy partitions was accomplished using the latter. The mathematical morphology algorithm works by applying a sequence of antagonistic, but asymmetrical filtering operations to the input data, until fuzzy kernels are obtained that include, each, mostly representatives of one class.

3.2.3. Difference between classical and fuzzy decision trees

A major difference between a classical and a fuzzy decision tree is the decision process that they use. Figure 5 illustrates two binary trees of the same height, where one uses sharp thresholds and the other fuzzy thresholds to process the input data. After 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 0.4 and class 1 with truth-value 0.6[1]. The final class membership may be decided by defuzzifying these results.

One possible way to do so is to compute 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 our application as both methods of defuzzification yield the same result in the case of a decision tree with two classes.

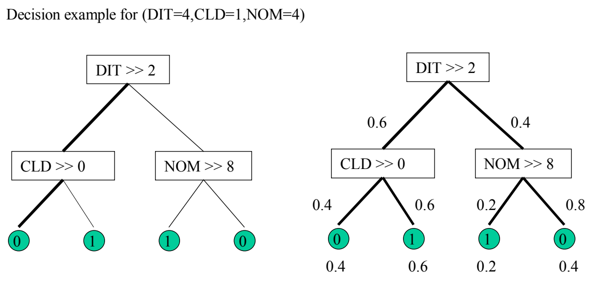

Figure 5. Classification using binary inference (left) and fuzzy inference (right)

3.3. Handling the naïve model problem with chains of fuzzy rules

As mentioned in Introduction, although the rules of the naive models may show reasonable relationships between the predicted and the predictive variables, it is difficult to use them for decision support during software development. An essential task to achieve this is to add parts that would enable us to better understand the causality relations between the internal attributes of the software under scrutiny and the quality characteristic that is predicted. We may then be in a better position to decide which action to take to improve the latter. To accomplish this, we break up each rule by including intermediate clauses that are intuitively interpretable to an expert of the field.

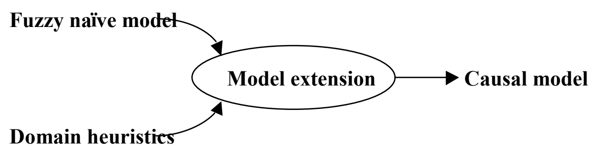

Figure 6. Model extension using domain heuristics

[1] To compute the truth values at the leaves, we use the min-max algorithm: minimum truth value along each tree path, maximum truth value for each end leaf.

To illustrate this process, consider the rule "*If DAM(c) ≤ 0.7 then c may be instable (as the system evolves)*". Suppose that condition "*If DAM(c) ≤ 0,7*" becomes, after fuzzification, "*If DAM is small or medium*" (because it corresponds to values covered by the *small* and *medium* labels). The obtained fuzzy rule may be visualized as in Figure 7, where the dashed arrows mean that the relation between the input and output variables is not obvious and needs further scrutiny. This is achieved by searching the domain knowledge for heuristics that can connect the assertion *DAM is medium* or *DAM is small* to intermediate conclusions which, in turn, may lead to the final conclusion (*class instable*).

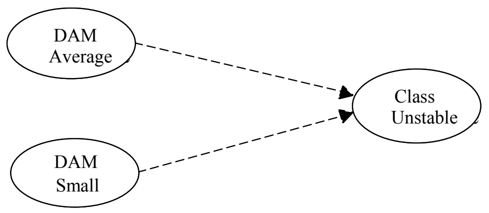

Figure 7. Naive rule with fuzzy thresholds values

As illustrated in Figure 8, several intermediate relations may be added. For example, If DAM is small or medium, we know that then the majority of the attributes is visible to other components in the system.

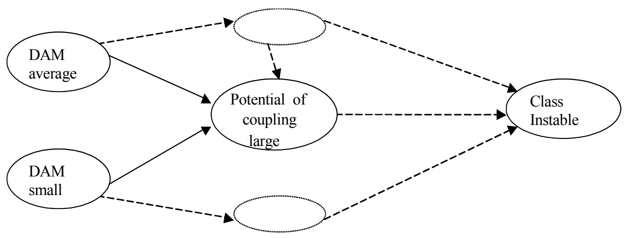

Figure 8. Introduction of intermediate rules

As a result, the potential for coupling between the studied class and the remainder of the system is high. We represent this relation in Figure 8 with a solid arrow, meaning that it is intuitively sound.

The previous step can be repeated to refine all of the relations represented by dashed arrows. In our example, the relation between *potential coupling large* and *class instable* can be broken up by using a domain heuristic stating that a large potential coupling leads to the risk of seeing unforeseen and unwanted secondary effects from a class change. Finally, since the relation between the risk of secondary effects from changes and class instability is intuitive, it is represented by a final solid arrow ().

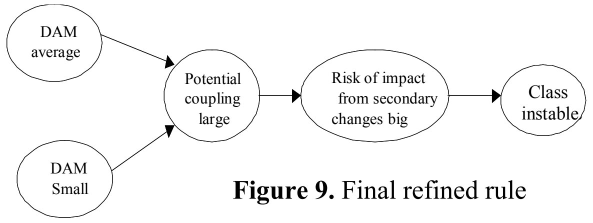

Figure 9. Final refined rule

As another example of rule refinement by adding domain knowledge, consider the naive rule "*If OCAEC > 0 then class instable*". In this case, we consider all possible non zero values of OCAEC, leading to the consideration of fuzzy labels *small, medium* and *large* for OCAEC.

Figure 10 shows the obtained results. Notice that an additional metric, NAA, is introduced in order to derive the refined rule from small and medium values of OCAEC. We need a low value of NAA in order to reach the intermediate conclusion "*effect of modification of an attribute large*". As in the previous example, we continue to include domain heuristics until we have relations that are all intuitively acceptable to experts in the field.

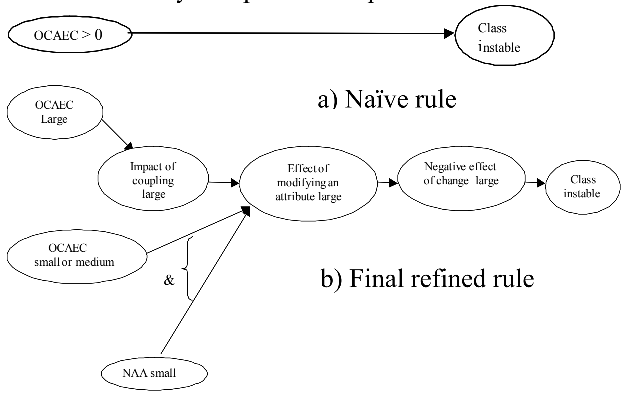

Figure 10. Second example of rule refinement

Several additional structural metrics may be needed before we end up with the final refined model. The following example shows that to refine the rule *"If DIT > 1 then class instable"* we need to input two additional metrics, NAM and NOP in order to include domain heuristics. Figure 11 shows the final result.

In summary, the process of refinement starts from the antecedent of each rule in the naïve model and continues until all the relations are intuitively verifiable. The end result is a model that is more understandable, more explicit, and more suitable for decision support than the naïve model it originates from.

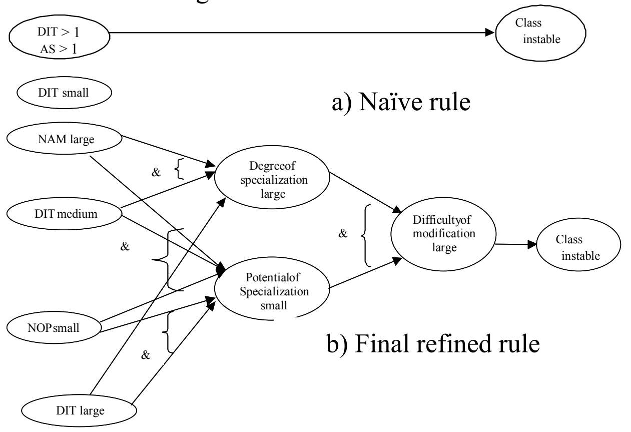

Figure 11. Third example of rule refinement

3.3.1. Using the extended rules for decision support

Consider the following example, taken from one of the systems that were studied in this work. Table 3 gives the metric values and the classification result by C4.5, of a class called ServletDirectory.

Table 3. Example of metric values and classification

Class name	NOP	DIT	NAM	Classification
ServletDirectory	0	5	160	1 (instable)

C4.5 based its conclusion on a naïve rule generated after analysis of the library that contains ServletDirectory. The rule was:

DIT > 1 : 1 (1= "class instable")

Since ServletDirectory verified the rule (DIT=5), it was declared to be instable. However, from a developer's standpoint, this conclusion is of limited use as it only suggests reducing the value of DIT in order to achieve stability, a not so obvious proposition.

On the other hand, if we look at the expanded rule of Figure 11b, we see that several other factors may lead to the instability conclusion. For instance, with the variable truth values shown in Figure 12, we have first that the instability of the ServletDirectory class is a direct consequence of the difficulty to bring modifications to it. The difficulty, in our case, may be due to either a low potential of specialization or a high degree of specialization of ServletDirectory (truth value of 0.45 for both options). At this point, we know that improving the stability of the class requires increasing its potential of specialization and/or reducing its actual specialization. Backtracking further through Figure 14 reveals that, in order to achieve this goal, an alternative to reducing DIT is to increase NOP. Thus, the class could be refactored by adding more abstract methods to increase the number of polymorphic ones (larger NOP). As for reducing DIT, we noticed that we could not do it in this particular example.

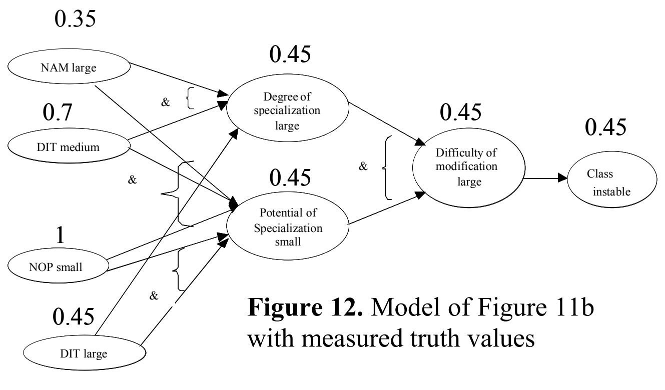
0.35
NAM large
0.45
&
Degree of specialization large
0.7
DIT medium
0.45
0.45
Difficulty of modification large
Class instable
&
&
0.45
Potential of Specialization small
1
NOP small
&
0.45
DIT large

Figure 12. Model of Figure 11b with measured truth values

3.3.2. Rules extension algorithm

The previous examples show that, in order to derive a causal fuzzy model from a naïve one, we start by breaking the link between the antecedent and consequent of each naïve rule. Then, we insert a sequence of intermediate rules derived from domain-specific heuristics, with the conclusion of each rule serving as the condition of the next.

4. Application to predict OO software stability

To evaluate the predictive abilities of the proposed framework, we used it to build prediction models for the stability of the class interfaces between versions of a software package. Our decision to use stability was motivated by the fact that, for this characteristic, we don't need hard to collect external data, such as defect data, maintenance effort, etc. Stability can be determined by simply comparing the evolution of a class interface among the major versions of the software under study. We selected 5 java systems that have at least two major versions. The sizes, in number of classes, of the initial versions of these systems are given in Table . Three of them, called[2] A, B and C were used to build the stability predictive models, the two others, named D and E, served for the evaluation.

Table 4. Selected systems

System	Type	Size
A	Programmer's text editor	82
B	Programmer's text editor	48
C	Web server platform	356
D	Tool for Building Interactive Web Sites	392
E	Internet voting system	50

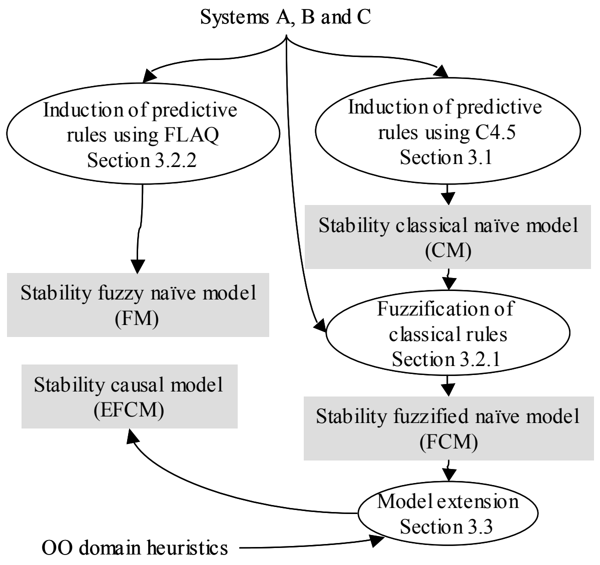
Systems A, B and C
Induction of predictive rules using FLAQ Section 3.2.2
Induction of predictive rules using C4.5 Section 3.1
Stability classical naïve model (CM)
Stability fuzzy naïve model (FM)
Fuzzification of classical rules Section 3.2.1
Stability causal model (EFCM)
Stability fuzzified naïve model (FCM)
Model extension Section 3.3
OO domain heuristics

Figure 13. Derivation of the stability models

As shown in Figure 13, we used systems A, B and C to derive the stability prediction models. Four models were obtained corresponding to the different techniques explained in sections 3.1 to 3.3. They were as follows: (CM) classical naïve model obtained using C4.5, (FCM) fuzzified version of model CM, (EFCM) extended model of FCM using domain heuristics, and (FM) fuzzy naïve model obtained by applying FLAQ.

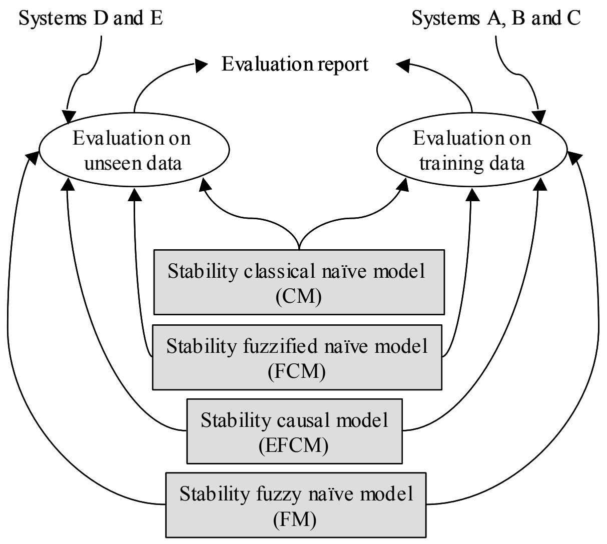
Systems D and E
Systems A, B and C
Evaluation report
Evaluation on unseen data
Evaluation on training data
Stability classical naïve model (CM)
Stability fuzzified naïve model (FCM)
Stability causal model (EFCM)
Stability fuzzy naïve model (FM)

Figure 14. Evaluation of the stability models

We conducted two evaluations (see Figure 14). The first one concerned the original training data. This evaluation had two goals: (G1) verify the hypothesis that the fuzzification and the extension of the C4.5 model preserve its performance and possibly improve them; (G2) verify the hypothesis that on the same learning data, fuzzy-based learning (FLAQ) can generate better models than classical learning (C4.5) in the particular case of quality prediction. The second evaluation concerned three other systems (D and E) that were not used in the training. The goal of this evaluation (G3) was to determine how each model performed when dealing with new data.

The evaluations were accomplished using another custom-built tool called OO1-predict. This tool allows performing the prediction using classical or fuzzy rule-based models. Table 2 provides the results that we obtained for the different models.

Table 2. Accuracy of predicting instable classes

Data	CM	FCM	EFCM	FM
Training sets (A, B, C)	69%	69%	69%	80%
Test set D	75%	81%	81%	88%
Test set E	42%	57%	61%	78%

[2] Since the IP owners didn't give us permission to mention the actual system names, we replaced them by letters.

As one can see, several conclusions can be drawn from them regarding the hypotheses that were tested:

- The fuzzification of C4.5 with extension preserves the performance of the original C4.5 model (G1).
- FLAQ offers better performance than C4.5 on the training data. (G2)
- Either of our solutions (Fuzzyfied c4.5 or FLAQ) to the threshold value problem significantly improves the performance of the prediction (G3).
- Model extension, beside the fact that it makes a model more useful for decision, can actually improve the prediction results (set E). This could be due to the new metrics that are introduced (G3).
- FLAQ systematically offers better performance than the other models.

5. Summary and conclusion

We presented a framework to develop and use rule-based software quality predictive models. By using fuzzy logic-based tools, we circumvented the major problem of threshold values in classical models. In addition, by extending our models with domain heuristics, we provided a way to bridge the semantic gap between predictive and predicted software attributes, making the models more useful as decision support tools. Finally, the extended model opened the door to perform preventive maintenance on software at the early stages of its life cycle.

Although the obtained results are encouraging, there remain issues open for improvement. These include the automation of the rule extension phase by creating a repository of domain heuristics and by defining efficient algorithms to exploit its contents.

6. 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] J. C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, New York, 1981.

[3] B. Bouchon-Meunier, C. Marsala, "Learning Fuzzy Decision Rules". In: J. Bezdek, D. Dubois and H. Prade (Ed.), Fuzzy Sets in Approximate Reasoning and Information Systems, Kluwer Academic Pub., Handbooks on Fuzzy Sets Series, chap. 4, pp.279-304, 1999.

[4] L. Briand, J. Wüst, H. Lounis, "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs", *J. Empirical Software Engineering,* Vol 6, No 1, 11-58, 2001.

[5] J.J. DeGruijter and A.B. McBratney, "A modified fuzzy k means for predictive classification". In: Bock, H.H.(ed), Classification and Related Methods of Data Analysis, Elsevier Science, Amsterdam, pp. 97-104. 1988.

[6] N. E Fenton, M. Neil, "Software Metrics : Roadmap" proc. *ICSE 2000 - Future of SE Track*, pp. 357-370

[7] N. E. Fenton, N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System", *IEEE Transactions on Software engineering*, 26(8), 797-814, 2000.

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

[9] C. Marsala, and B. Bouchon-Meunier. "Fuzzy partioning using mathematical morphology in a learning scheme". Proc. *5th Conference on Fuzzy Systems*, New Orleans, Sept. 1996.

[10] F. W. Opdyke, "Refactoring Object-Oriented Frameworks*",* Ph.D thesis, University of Illinois, 1992.

[11] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Pub., 1993.

[12] H. A. Sahraoui, M. Boukadoum, H. Lounis, F. Ethève, "Predicting Class Libraries Interface Evolution: an investigation into machine learning approaches", Proc. *7th IEEE Asia-Pacific Software Engineering Conference*, Singapore, Dec. 2000.

[13] C.E. Shannon, "A mathematical theory of communication", Bell Sys. Tech. J., pp. 379-423, 623-656, 1948.

[14] 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.