Extending Software Quality Predictive Models Using Domain Knowledge

Houari Sahraoui, Mohamed Adel Serhani
University of Montreal
{sahraouh, serhanim}@iro.umontreal.ca

Mounir Boukadoum
University of Quebec at Montreal
boukadoum.mounir@uqam.ca

Abstract

Current methods to build software quality estimation models suffer from two problems: The use of precise threshold values and their interpretation in the absence of formal models, and the crudeness of the derived rules which can only serve to build naïve models. In this position paper, we describe a novel approach to alleviate both problems. We propose to build fuzzy decision processes that combine both software metrics and heuristic knowledge from the field. As a result, quality estimation models are obtained that are both efficient and that provide a more comprehensive explanation of the relationship that exists between the observed data and the predicted software quality characteristic.

1 Position

Building efficient and usable software quality estimation models is an important challenge for both research and industry communities. The work done so far can be classified into two families. The first one deals with the use of historical measurement data to build quality estimation models (see for example [6]). The quality of these models depends heavily on the quality of the used samples. Most of them capture the trends but fail in the definition of widely applicable threshold values for the rules that are generated [3]. Moreover, as stated by Fenton & Neil in [2], the majority of the produced models are naïve; they cannot provide a decision support during the software development process. The cognitive distance between the internal measures and the quality characteristic to predict is too far.

The second family of techniques to build software quality estimation models uses knowledge extracted from domain-specific heuristics. The obtained predictive models involve judgments from experts to establish a causal relationship between internal software attributes and quality characteristics (see for example [5]). 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.

This position paper claims that we can get the best from both worlds by using a hybrid approach for building predictive models. This approach would allows to:

- Circumvent the problem of specific threshold values in rules by using fuzzy ones
- Benefit from the historical data to establish statistically significant relationships
- Integrate the domain knowledge to make the models more useful

Section 2 presents our approach; it explains the rationale behind it and describes its main steps. A brief discussion then follows in section 3.

2 Deriving causal models from naïve ones

2.1 Rationale

As said in the previous section, naïve models, whether built using statistical or artificial intelligence techniques, efficiently capture the relationship (trends) between software internal attributes and quality characteristics. However, they present two major problems. Firstly, the threshold values that define boundaries for the obtained classes are too specific to the data samples used to derive them. As it is hard to perform a rigorous sampling due to the lack of reliable data, the obtained models are hard to generalize. In a previous work [3], we showed that the generalization problem comes more from the thresholds values than from the derived trends. The second problem is related to the fact that the models cannot be used to make comprehensive decisions during the software lifecycle. To illustrate these two problems, consider the following examples: The rule below is part of a maintainability predictive model obtained by applying a machine-learning algorithm to a set of classes (sample).

If the number of methods in a class c is greater than 20, then c is hard to maintain

This rules establishes a relationship between the number of methods and the maintainability of a class. However, the proposed threshold value (20) is obtained from empirical data. This brings the problem of its precision and interpretation. For instance, why 20 in this example and not 19 or 21? Why is a class with 20 methods easy to maintain and one with 21 not?

As a second example, consider the family of models *effort=f(size)* obtained using linear regression. A manager cannot use these models for risk assessment because they do not explain the relationship between the two variables [2]. We cannot answer questions such as "Why do large systems need more development effort?" or "What can I do if the number of available resources is less than the number predicted by the model?"

The approach we propose aims at solving these two problems. It consists in the transformation of a naïve model by:

- Mapping precise threshold values into fuzzy ones.
- Including domain knowledge to explain the relationship between independent and depend variables.

In a first stage, we are only applying our approach to machine learning-based models. The procedure includes two main steps: 1) fuzzification of the naïve model's rules, starting with the internal attributes metrics used as input for the prediction, and 2) combination of the obtained fuzzy rules with domain heuristics and creation of new rules. The inputs and outputs of these two steps are shown in
Figure 1.

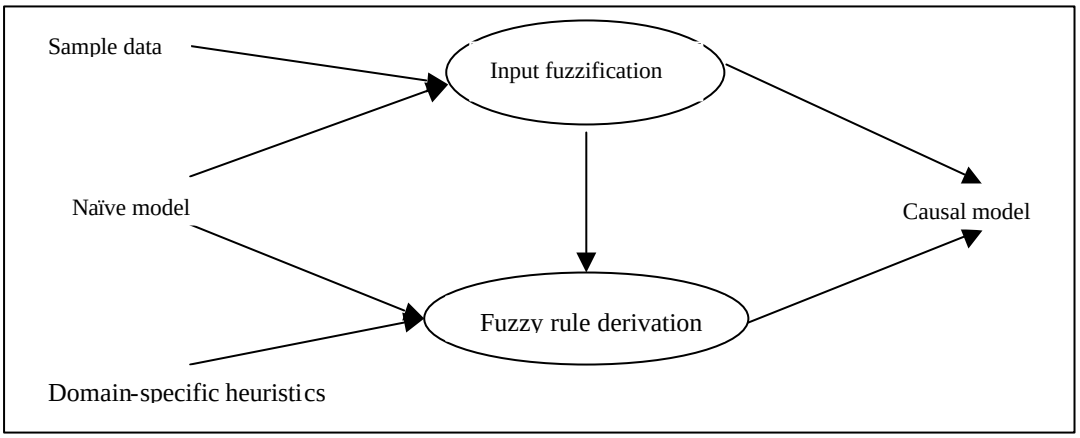

Figure 1. Naïve to causal model transformation

The following sections describe the two steps of the transformation.

2.2 Input fuzzification

Many fuzzification techniques have been proposed in the literature to fuzzify quantitative attributes. Some of them are simple to use, others are more sophisticated [4]. In our approach, we use a simple technique that is based on the distribution of the measurement data. We calculate for each input (metric) value its frequency and we derive homogeneous clusters from the resulting curve. The clusters represent the fuzzy labels of the input metric. In some cases, we must first apply an additional processing to the curve in order to reveal the clusters.

To illustrate this technique consider the following table containing the values of some metrics for a set of classes.

	DIT	NOC	**NPA**	NAA	……..	OCAIC	OCAEC
Classe 1	1	1	3	5		2	1
Classe 2	2	3	4	10		1	0
…..							
Classe n	1	2	3	8		1	1

Figure 2 shows the distribution of the values of NPA (Number of Public Attributes) metric.

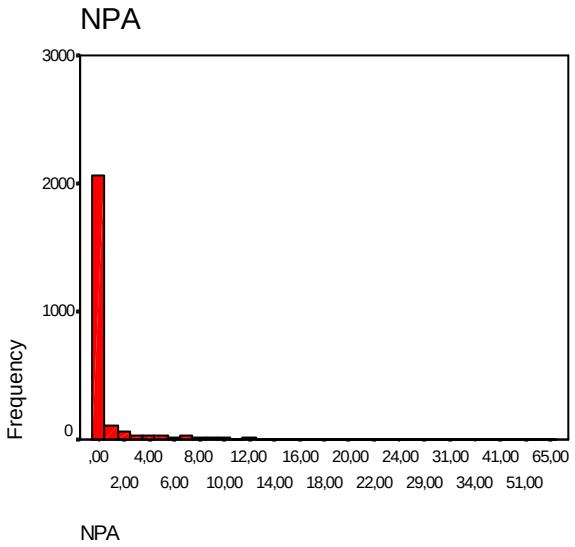

Figure 2. Table of metric value and their relative histogram

By looking at the obtained result, it is hard to derive homogenous clusters. As most of the classes have no public attributes; this makes the bars for the other values difficult to see. To solve this problem, we apply a logarithmic transformation to the frequency values to boost the smaller values and to flatten the larger ones. The result is shown in Figure 3.

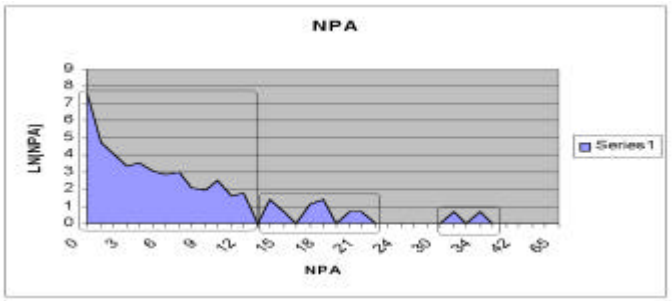

Figure 3. Frequency logarithm curve

In Figure 3, three homogeneous clusters of values can now be seen. Starting from these clusters, we can define three fuzzy labels for the NPA metric and associate a membership function with each of them. The next step is to define and assign a membership function to each one of them as shown in Figure 4. Standard membership function shapes (trapeze or triangle) may be used.

Any value of the metric NPA can then be mapped into the three labels with different membership values (between 0 and 1).

It is easy to associate the threshold values with the rules using the obtained membership function. The principle is to choose the labels that best match the condition in the rule. For the example given

in section 2.1, the condition *if number of methods greater than 20* becomes *if number of methods is large.* The obtained rule is then

if number of methods of a class c is large, then c is hard to maintain

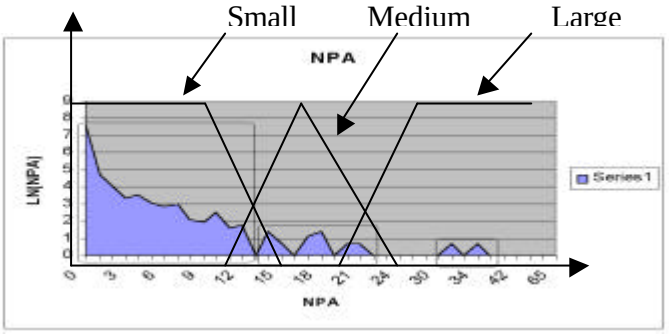

Figure 4. Membership functions

2.3 Rules derivation

Unfortunately, although the previous rule shows a reasonable relationship, it is hard to use as a decision support. It is not clear why large classes are hard to maintain. The key point at this point is to add the missing parts that would enable us to explain the causality between the internal software attributes and the predicted quality characteristic. To do this, we break a rule into a set of intuitive and easier to validate sub rules by including domain knowledge. To illustrate the domain knowledge integration, let's take the particular case of the following rule.

If DAM(c) <= 0,7, then c can become unstable during the evolution of the system

DAM(Data Access Metric) is the percentage of private and protected attributes in a class.

The condition *If DAM(c) <= 0,7* can be mapped to the fuzzy condition *if DAM is small or medium* because *<= 0,7* matches the membership functions of the labels *small* and *medium.* The new rule can be viewed as in Figure 5 where a dotted arrow means that the relation between the two variables is to be refined. This is accomplished by searching the domain knowledge for heuristics (relationships) that can connect the facts *DAM medium* and/or *DAM* small to an intermediate conclusion, which can possibly lead to the final conclusion (class unstable).

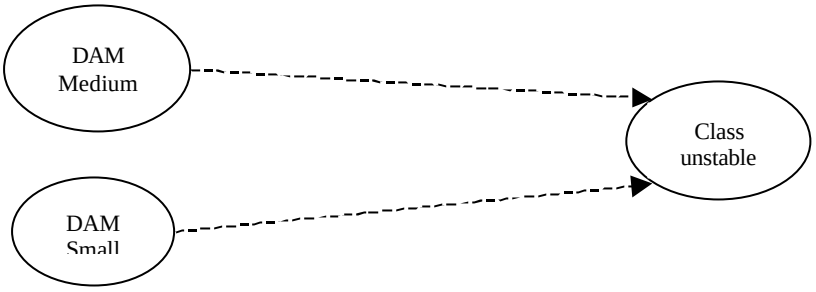

Figure 5. Naive rule with fuzzy threshold values

As illustrated in Figure 6, different intermediate relations can be found. One of them is that if DAM is medium or small, then most of the attributes are visible to the other component of the system. This would lead to the conclusion that the potential coupling between the studied class and the rest of the system is high and we represent this relation with a solid arrow to mean that it is intuitive and verifiable.

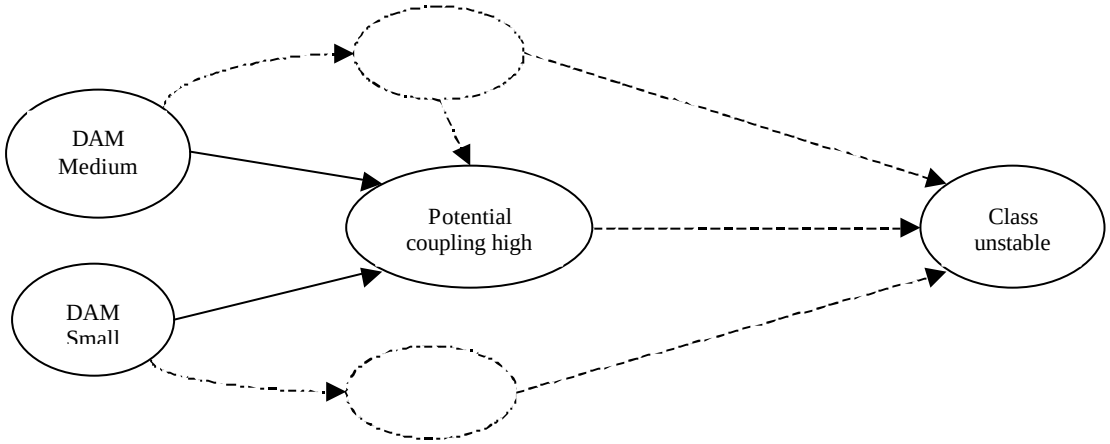

Figure 6. Introduction of a first level of intermediate conclusions

Following the same principle, we must refine all the relations represented by dotted arrows. In our example, the relation between *potential coupling high* and *class unstable* can be decomposed using the domain heuristic that state that a potential high coupling can lead to a high risk of change side effect (see Figure 7). Indeed, any change in the system can generate errors in the studied class. The relation between *High risk of change side effect* and *Class unstable* is considered as intuitive and is then represented by a solid arrow. The process of refinement continues until all the relations are intuitive and verifiable. The obtained partial causal model is better at showing the reasoning process and can then be used a decision support during the software development process.

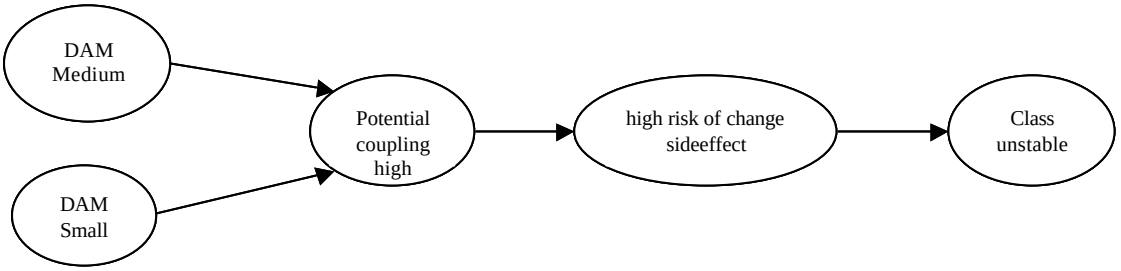

Figure 7. Final refinement of the rule of Figure 5

The fuzzy rule derivation step can be summarized in the following constraint programming algorithm.

Ri : Rule
NR : Set of naïves rule
CR : Set of causal rules
ICR : Set of initial rules
FCR : Set of final rules

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 condition of each rule in CRi-ICRi are conclusion of rules in CRi
d - Each rule in CRi represents a verifiable causal relationship

END DO

3 Discussion and conclusion

In this position paper, we propose a hybrid approach to bridge the gap between two families of work for building and using software quality predictive models. The two families offer complementary advantages. Historical measurement data-based approaches derive statistically-correct relationships between software internal attributes metrics and quality characteristics. In the same time, domain knowledge-based approaches are easy to use for their explanatory capabilities. Rather building the latter from scratch, we propose to extend the former using domain heuristics. In addition to this extension, we transform the precise threshold values using fuzzy logic.

We are currently conducting experiments to compare our models with measurement data-based ones. In these experiment, we extend existing predictive models by adding domain knowledge. This comparison will be made on two aspects: (1) the accuracy of the models on a set of unseen cases, and (2) the usability of the models and their explanatory capabilities.

The next stage of this work is to automate the two main steps. The automation of the fuzzification process can be easily done except for defining the label boundaries. Different fuzzification techniques can be used and most of them can be automated (see, for example, the technique proposed by Marsala and Meunier in [1] which uses mathematical morphology concepts - dilatation and erosion). The second step is hard to automate. One solution can be the creation of a repository containing a large set of intuitive relationships between the different attributes of the software, as an alternative to the dynamic involvement of the expert during the creation of the model. The refinement can be then automated by searching in this repository the appropriate relationships.

References

[1] "Fuzzy Partitioning Using Mathematical Morphology in a Learning Scheme" Christophe Marsala, Bernadette Bouchon-Meunier.

[2] Norman E Fenton, Martin Neil, *Software Metrics : Roadmap* ICSE - Future of SE Track 2000: 357-370

[3] Houari A. Sahraoui, Mounir Boukadoum, Hakim Lounis, Frédéric Ethève, Predicting Class Libraries Interface Evolution: an investigation into machine learning approaches, In Proc. of 7th Asia-Pacific Software Engineering Conference, 2000.

[4] Bart Kosko, Fuzzy Engineering, Prentice Hall, 1996.

[5] 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.

[6] 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.