INVESTIGATION OF AUTOMATIC PREDICTION OF SOFTWARE QUALITY

Joshua Osbeck
Wartburg College
Waverly, Iowa 50677
joshua.osbeck@wartburg.edu

Shamsnaz Virani
University of Texas at El Paso
El Paso, Texas 79968
ssvirani@utep.edu

Olac Fuentes
University of Texas at El Paso
El Paso, Texas 79968
ofuentes@utep.edu

Patricia Roden
University of North Alabama
Florence, Alabama 35632
plroden@una.edu

***Abstract*—The subjective nature of software code quality makes it a complex topic. Most software managers and companies rely on the subjective evaluations of experts to determine software code quality. Software companies can save time and money by utilizing a model that could accurately predict different code quality factors during and after the production of software. Previous research builds a model predicting the difference between bad and excellent software. This paper expands this to a larger range of bad, poor, fair, good, and excellent, and builds a model predicting these classes. This research investigates decision trees and ensemble learning from the machine learning tool Weka as primary classifier models predicting reusability, flexibility, understandability, functionality, extendibility, effectiveness, and total quality of software code.**

I. INTRODUCTION

Designing and building quality software is an important part of every computer programmers' career. In their education, computer science students learn to incorporate design techniques and object-oriented programming into developing high quality software. Though in practice, some of these concepts might be abandoned for convenience at the time or out of pure carelessness. Either way, this leads to creation and sale of low quality software. Such software creates unsatisfied customers and potential large economic problems. The National Institute of Standards and Technology's 2002 study on software reports the cost of inadequate software as $59.5 billion dollars a year [1]. Just three years later, it was reported that organizations spend about $1 trillion on software and services annually, and 5 to 15 percent of those projects are flawed, defect prone, or with defects [2]. The software industry has grown tremendously since 2005, and proportionally so does the costs of software defects. Since the software market is big, many groups are currently trying to estimate the amount of bad software and its economic impact today. The Standish Group's CHAOS Summary 2009 [3] estimate that only 32% of all software projects complete on time and within budget. Of the other 68%, 44% of them come out late, over budget, or below standards, and 24% of software projects are scrapped altogether [3].

To prevent these problems, models such as Capability Maturity Model for Integration (CMMI) [4] and Business Driven Development (BDD) [5] have been developed. These models help assess the software development process, but do not try to quantify what experts look for when determining the software product quality. These models look at process metrics, like faults per line of code, not at product metrics like polymorphism and inheritance. Product metrics take measurements of code in terms of concepts that programmers know make excellent software. Therefore, quantifying expert opinion of software quality based on product metrics should lead to determine the quality of software.

The major problem is that software product quality assessment is not easy or consistent to measure. Most companies rely on expert views in conducting these assessments. However, there are no uniform standards to base software experts' evaluations. Therefore, there are no consistencies regarding software quality among companies or even among experts within the same company.

With a model that can predict quality at any point during the software development process, software managers reduce their dependence on expert opinion. Virani, Messimer, Roden, and Etzkorn [6] built a model for predicting software quality factors as bad and excellent. This paper expands this model for the entire range of bad, poor, fair, good, and excellent. Most software code is not bad or excellent, but rather somewhere in between. By having three other classes, software can be identified similar to the way a human would classify it. Previous research shows high accuracy in predicting between bad versus excellent and poor verses good, but accuracy reduces when trying to make predictions on a larger range. The previous research models [6] were built using the C4.5 decision tree. C4.5 is a standard machine learning decision tree. Machine learning is an important aspect in predicting software product quality because the more a classifier can learn, the better decisions it will make in building a predictive model [7]. In this paper, we explore other classifiers and ensemble methods in order to expand the range and improve the accuracy of the C4.5 model.

II. RESEARCH METHODOLOGY

The goal of this research is to expand the range of prediction from two to five classes. In order to do so machine learning algorithms are applied. The primary tool used for generating models and testing classification was the machine learning tool Weka [8]. Weka consists of classification algorithms and filtering techniques allowing for easy preprocessing, implementation, and comparison of a variety algorithms. The algorithms investigated in this research are J48, Part, and Random Forest, and the ensemble learning techniques examined were boosting, bagging, and stacking [8].

A diagram of the stacking classifier used is shown in Figure 1. In this case, Part and Random Forest (with default values) were used as base classifiers and J48graft was used as the meta classifier. They were chosen because of previous experience testing these classifiers.

978-1-61284-968-3/11/$26.00 ©2011 IEEE

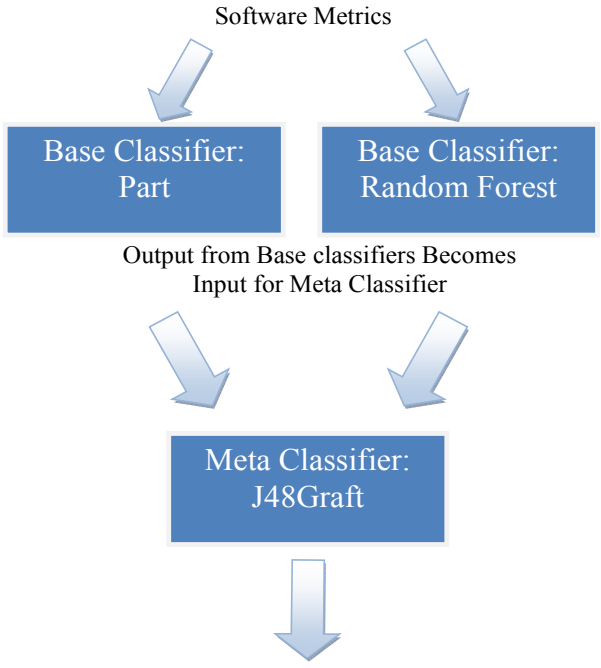

Fig. 1 Model of Stacking Technique Used

J48 is a Java based version of the C4.5 decision tree built in Weka, and will be the base model to make comparisons with the other algorithms. Part was chosen as the first classifier to investigate because it is a direct extension of J48 [9]. From partial decision trees of J48, Part directly builds rules to be used in classification. The second classifier, Random Forest, was chosen because it is a decision tree that uses random feature selection and bagging in creating rather accurate models [10]. Since random forest is actually an ensemble learner that only uses random trees [11], the next step was to look into applying techniques, such as bagging, used in random forest to the J48 classifier. Bagging is an example of ensemble learning in which classification is improved by voting where all models have the same weight. Two similar techniques to bagging, boosting and stacking, were tried to improve the results of J48. In boosting, classification models are built and given different weights for the voting process. In stacking, there are two levels of classifiers. The first level, the base level, consists of several different classifiers that each makes predictions. In the second level there is one classifier, the meta classifier, that learns from the output of the base classifiers and learns which classifier to trust when making its own predictions [12][11].

The metrics used are Quality Model for Object-Oriented Design (QMOOD) metrics defined by Bansiya and Davis [13] and used by Virani et al [6]. The metrics and the quality factors used in this research are restricted to Bansiya and Davis' model. Table 1 gives Bansiya and Davis' definitions for each of the quality factors and metrics used. The data contains 31 software packages with the metrics as the

Table 1. Bansiya and Davis' Model [13]

Quality Factor Definition	Metric Definition (QMOOD metric)
Reusability Reflects the presence of object oriented design characteristics that allow a design to be reapplied to a new problem without significant effort.	**Design Size (DSC)** A measure of number of classes used in the design. **Hierarchies(NOH)** Hierarchies are used to represent different generalization-specialization aspects of the design. **Abstraction(ANA)** A measure of generalization- specialization aspect of design. **Encapsulation(DAM)** Defined as the enclosing of data and behavior within a single construct. **Coupling(DCC):** Defines the inter dependency of an object on other objects in a design. **Cohesion(CAM)** Accesses the relatedness of methods and attributes in a class. **Composition(MOA)** Measures the "part-of," has", "consists –of", or "part-whole" relationships, which are aggregation relationships in object oriented design. **Inheritance (MFA)** A measure of the "is-a" relationship between classes. **Polymorphism(NOP)** It is a measure of services that are dynamically determined at run-time in an object. **Messaging (CIS)** A count of number of public methods those are available as services to other classes. **Complexity(NOM)** A measure of the degree of difficulty in understanding and comprehending the internal and external structure of classes and their relationships.
Flexibility Characteristics that allow the incorporation of changes in a design. The ability of a design to be adapted to provide functionality related capabilities.	
Understandability The properties of designs that enable it to be easily learned and comprehended. This directly relates to the complexity of design structure.	
Functionality The responsibility assigned to the classes of a design, which are made available by classes through their public interfaces.	
Extendibility Refers to their presence and usage of properties in an existing design that allow for the incorporation of new requirements in the design.	
Effectiveness This refers to the designs ability to achieve the desired functionality and behavior using object oriented design concepts and techniques.	

Table 2. Reusability Confusion Matrix Using J48

	Predicted Reusability						
Real Reusability		Bad	Poor	Fair	Good	Excellent	Completeness
	Bad	0	38	151	25	0	0.00%
	Poor	0	216	292	18	0	41.06%
	Fair	3	170	753	98	0	73.54%
	Good	2	85	310	125	0	23.95%
	Excellent	0	19	21	4	0	0.00%
	Correctness	0.00%	40.91%	49.31%	46.30%		
	Accuracy	46.95%					

independent variables and the rating as the dependent variable. The ratings options are bad, poor, good, fair, and excellent. Each package was rated individually by a group of experts for each of Bansiya and Davis' factors and for total quality. The software packages examined were both open source and student written in C++. The tool used to extract the metrics was Cantata++. The complete data set is made up of 2330 instances for each quality factor and is the same data set that was used by Virani et al [6] in their prediction model for bad or excellent.

The results were obtained by importing the data into Weka, running the corresponding classifier five times, and using the classification output to calculate accuracy, correctness, and completeness in each case.

III. RESULTS

For each classifier, a ten-fold cross validation technique is used to train and test the model. The results were placed in a confusion matrix in each case in order to calculate values for correctness, completeness, and accuracy. Completeness is calculated for each class (bad, poor, fair, good, and excellent) as the number of instances classified correctly divided by the number instances that belong to that class and can be calculated as

$$\frac{n_{ii}}{\sum_{j=1..5} n_{1j}}$$

where *i* goes from 1 to 5. With 1=bad and 5=excellent, such that n_{11} is the number of instances that were classified correctly as bad, and n_{12} would correspond to the number of instances that were predicted as poor, but were actually bad. Correctness is determined for each class by the number of instances correctly classified divided by the total number predicted for that class. The equation for correctness is

$$\frac{n_{jj}}{\sum_{i=1..5} n_{i1}}$$

To determine accuracy, we take the total number of instances classified correctly divided by the total number of classes. This can be expressed with the equation

$$\frac{\sum_{i=1..5} n_{ii}}{\sum_{i,j=1..5} n_{ij}}$$

Using the J48 decision tree classifier, the accuracy of each quality factor ranged from 46.95% to 56.78%, with reusability being the least accurate and functionality being the most accurate. Table 2 shows the confusion matrix of the J48 decision tree for reusability. The rest of this paper will present comparisons to the matrix for reusability, for there were no cases in which one classifier performed much better on one quality factor then all the others. The results given from testing reusability are representative of all of the quality factors in terms of improvement. This shows that the Bansiya and Davis' collection of product metrics can efficiently represent all of the quality factors that software managers look for. However, reusability has the lowest percentage of accuracy with the J48 decision tree and has the lowest value for accuracy unless noted otherwise.

A. Five Classes

The first tests were run using the full five class data set of bad, poor, fair, good, and excellent.

1) Classifiers

The first classifier investigated was Part because of its connections to the J48 decision tree algorithm. No major improvements were made using Part, with reduction in overall accuracy for reusability. With each quality factor, the difference in accuracy varied up to 1.2% from that of J48.

The difference in accuracy for Random Forest was only negative. Overall accuracy dropped between 7.38% and

Table 3. Reusability Confusion Matrix Using Stacking with 200 trees in Random Forest

	Predicted Reusability						
Real Reusability		Bad	Poor	Fair	Good	Excellent	Completeness
	Bad	86	45	63	17	3	40.19%
	Poor	28	289	132	67	10	54.94%
	Fair	39	107	799	72	7	78.03%
	Good	23	80	151	261	7	50.00%
	Excellent	4	9	8	4	19	43.18%
	Correctness	47.78%	54.53%	69.30%	62.00%	41.30%	
	Accuracy	62.40%					

12.45% for all quality factors. The literature [10] shows that random forest are fairly accurate prediction algorithms, however in this research they were not successful. Random forest are known to sometimes have trouble with imbalanced data sets and this could be one reason for these results [14]. In terms of reusability, only 9.81% and 1.91% of the instances in the data set belong to the classes bad and excellent respectively.

In order to resolve this problem, the default number of features and the default number of trees were varied [15]. The default number of features is $\lfloor \log_2 M + 1 \rfloor$ where M is the number of features and the default number of trees is 10 [15]. Values for number of features used were tested from 2 to 5, and the number of trees used were 50, 100, 150, and 200. There was little or no improvement with changing the number of features which is in agreement with [15]. However, changing the numbers of trees also resulted in little or no improvement, which differs from the results of [15]. In the case of 200 trees, the overall improvement in accuracy for reusability was only .04% over the default random forest. Even though, random forests themselves did not produce ideal results, the concept of comparing trees could be directly applied to the J48 classifier.

2) Ensemble Learning

A bagging algorithm, using components of random forest, was run with the J48 classifier. For each quality factor, the overall accuracy of the classification went down, with the largest drop of 2.02%. Instead of giving each model an equal weight, we investigated AdaBoostM1 with J48 to favor the better models. In each case, there was no improvement in accuracy, and in two cases, boosting was not possible. Reusability was one of these two cases, so the confusion matrix for AdaBoostM1 was the same as Table 2. Boosting was most likely not possible because of inconsistencies of expert opinion within the data that did not allow the creation of other trees. Inconsistencies can be handled better by stacking because different types of classifiers can be trained, tested, and compared.

Stacking was the only method that showed good improvements in overall accuracy for all quality factors. The average improvements ranged from 5.48% and 9.87% over J48, with reusability having the greatest improvement. From these positive results, it was decided to change the number of trees used in the random forest to 200 because it has shown to help in some cases [15].

For the data set used in this research, the overall accuracy increased in all cases with an average range of 9.36% to 15.38% over J48. Table 3 shows the confusion matrix for this model, and Table 4 shows the overall accuracy percentage of the two stacking models compared to J48 for all quality factors. In the first stacking example, reusability had the lowest accuracy with 56.59% and in the second stacking example, understandability had the lowest accuracy with 62.06%.

The percentage values for completeness and correctness also showed vast improvement with stacking. As seen in Table 3, each class was over 40% in terms of both completeness and correctness, and the middle three classes had over 50% in each case. This shows great progress over J48, which had 0% for both bad and excellent in terms of completeness and correctness. However, because more than one classifier was used, the size of the model also increased.

The models generated by stacking are fairly large, with an average of 270 nodes and 136 leaves. These trees can easily be rewritten in rule form to make explanation of prediction easier. Table 5 gives the first three rules generated for determining reusability for each class. The rest of the rules are omitted because of the size of the model and is beyond the scope of this paper. The rules are based on probabilities of a classifier predicting an instance of a certain class. Random Forest and Part are abbreviated RF and PT respectively, and the classes are represented by their first letter. For example, the first node of the first rule for bad checks if the probability of Part classifying an instance as fair is greater than 0.653846. In this particular model, there are another 9 prediction rules for bad, 23 for poor, 49 for fair, and 28 for good. Stacking was then applied to a four class range for comparison.

B. Four Classes

All of these algorithms were tested again by filtering out all instances that were determined fair by any expert. In reusability, 1024 out of 2330 instances were considered fair and this led to major imbalances in the data. By removing the instances that were classified as fair, we hoped to improve any errors in training that were caused by this large imbalance.

As expected, the algorithms with the most improvement used stacking. Table 6 shows overall accuracy comparison of J48 and the two stacking algorithms. The accuracies ranged from 69.91% for reusability to 78.54% for functionality. These values are fairly good and represent a good prediction model for determining these quality factors based solely on QMOOD metrics.

Table 4. Five Class Accuracy Comparisons

Quality Factor	J48	Stacking: Meta is J48Graft, Base is PART and RF	Stacking: Meta is J48Graft, Base is PART and RF with 200 trees
Effectiveness	51.38%	59.03%	63.49%
Extendibility	48.46%	58.33%	63.84%
Flexibility	51.40%	58.05%	62.12%
Functionality	56.78%	62.71%	66.14%
Reusability	46.95%	56.59%	62.08%
Understandability	48.54%	57.37%	62.06%
Total Quality	52.88%	58.36%	62.93%

Table 5. A Set of Stacking Prediction Rules for Reusability

Predictions	Rules
Bad	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F > 0.183278 and RF:G <= 0.621063 and RF:P > 0.001667 and PT:E <= 0 and RF:F <= 0.827918 and PT:G <= 0.517007 and PT:P > 0.008772 and PT:G > 0.1079 and PT:F <= 0.549708
	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F > 0.183278 and RF:G > 0.621063 and RF:F <= 0.623569
	If PT:F <= 0.653846 and PT:P > 0.086957 and PT:E <= 0 and PT:B <= 0.027027 and RF:G <= 0.286128 and PT:G <= 0.222222 and RF:G <= 0.002857 and PT:G <= 0
Poor	If PT:F <= 0.653846 and PT:P > 0.086957 and PT:E <= 0 and PT:B <= 0.027027 and RF:G <= 0.286128 and PT:G <= 0.222222 and RF:G <= 0.002857 and PT:G > 0 and RF:G > 0.000714
	If PT:F <= 0.653846 and PT:P > 0.086957 and PT:E <= 0 and PT:B <= 0.027027 and RF:G <= 0.286128 and PT:G <= 0.222222 and RF:G > 0.002857 and RF:P <= 0.294975 and RF:F > 0.75148 and PT:F > 0.557471
	If PT:F <= 0.653846 and PT:P > 0.086957 and PT:E <= 0 and PT:B <= 0.027027 and RF:G <= 0.286128 and PT:G <= 0.222222 and RF:G > 0.002857 and RF:P > 0.294975 and RF:F <= 0.453381 and RF:G <= 0.209347 and PT:F <= 0.348837 and RF:P <= 0.555395
Fair	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F <= 0.183278
	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F > 0.183278 and RF:G <= 0.621063 and RF:P <= 0.001667 and PT:F > 0.372549 and RF:F <= 0.557135
	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F > 0.183278 and RF:G <= 0.621063 and RF:P > 0.001667 and PT:E <= 0 and RF:F <= 0.827918 and PT:G <= 0.517007 and PT:P > 0.008772 and PT:G <= 0.1079
Good	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F > 0.183278 and RF:G <= 0.621063 and RF:P <= 0.001667 and PT:F <= 0.372549
	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F > 0.183278 and RF:G <= 0.621063 and RF:P <= 0.001667 and PT:F > 0.372549 and RF:F > 0.557135
	If PT:F <= 0.653846 and PT:P <= 0.086957 and RF:F > 0.183278 and RF:G <= 0.621063 and RF:P > 0.001667 and PT:E <= 0 and RF:F <= 0.827918 and PT:G <= 0.517007 and PT:P <= 0.008772
Excellent	If PT:F <= 0.653846 and PT:P > 0.086957 and PT:E > 0 and RF:F > 0.184677 and PT:G <= 0.075 and RF:F > 0.294202 and RF:P > 0.449136 and RF:E <= 0.002
	If PT:F <= 0.653846 and PT:P > 0.086957 and PT:E > 0 and RF:F > 0.184677 and PT:G > 0.075 and RF:G > 0.117362 and PT:E <= 0.16 and RF:E <= 0.079853 and PT:E > 0.108108 and RF:F > 0.344333
	If PT:F <= 0.653846 and PT:P > 0.086957 and PT:E > 0 and RF:F > 0.184677 and PT:G > 0.075 and RF:G > 0.117362 and PT:E > 0.16 and PT:P <= 0.138889 and RF:P <= 0.114728 and RF:P > 0.114728

Table 6. Four Class Accuracy Comparisons

Quality Factor	J48	Stacking: Meta is J48Graft, Base is PART and RF	Stacking: Meta is J48Graft, Base is PART and RF with 200 trees
Effectiveness	67.96%	72.98%	74.40%
Extendibility	51.99%	65.12%	70.37%
Flexibility	63.22%	73.36%	75.39%
Functionality	75.31%	78.01%	78.54%
Reusability	58.19%	66.48%	69.91%
Understandability	60.71%	70.59%	74.29%
Total Quality	58.44%	66.86%	72.00%

IV. Conclusion

This paper looks at different classifiers and learning methods in order to build models for six software quality factors and overall quality. The ensemble learning method of stacking was the most effective way to accurately create predictions models. The five class models can predict a class with over 60% accuracy. The four case models can predict a class with over 70% accuracy. Although these values are not ideal, accuracy at this level can give insight into software quality without spending lots of time studying software code. Without these types of models, software quality would not be easy to determine until the project is finished. These models can help a software manager determine software quality during the development progress without relying exclusively on the inconsistent evaluations of experts.

Future research will look into making smaller models and improving overall accuracy. This can be done by experimenting with other classifiers and configurations of classifiers inside the stacking model. Other research will look into building similar fuzzy inference systems that can also use rules to predict software quality.

The models here are robust and not ideal to explain how quality is determined. At this moment, the model would not be convenient for a software developer to use because of its size and complexity. Hopefully these issues can be resolved in future work. With a smaller model, software packages can be automated and tested periodically to catch defects and potential product problems early on in development. This insight into the development process will reduce the amount of faulty software that gets released, and therefore lower the cost that software companies spend to fix problems.

Acknowledgment

This work was supported by the NSF under grant IIS0852066.

References

[1] Research Triangle Institute, "The economic impacts of inadequate infrastructure for software testing", National Institute of Standards and Technology, Gaithersburg, MD, 2002.

[2] R. N. Charette, "Why software fails-the reasons that software projects go awry are well known. Yet failures, near failures, and just plain old bad software will continue to plague us", *IEEE Spectr.*, vol. 42, no. 9, Sept. 2005.

[3] Standish Group, "New Standish Group report shows more project failing and less successful projects", The Standish Group International, Inc., 2009.

[4] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley Professional, 1995.

[5] B. Amaba, "Business driven development", *IIE Annu. Conf. and Expo.*, Orlando, FL, 2006.

[6] S. S. Virani, S. Messimer, P. Roden, and L. Etzkorn, "Software quality management tool for engineering managers", *Proc. of the 2008 Industrial Engineering Research Conf.*, Vancouver, Canada, 2008, pp. 1401-1406.

[7] T. M. Mitchell, Machine Learning, 1st ed. New York: McGraw-Hill, 1997.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, "The WEKA Data Mining Software: An Update", SIGKDD Explorations, vol. 11, no. 1, pp. 10-18, 2009.

[9] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham. "Weka: Practical Machine Learning Tools and Techniques with Java Implementations", *Proc. of ANNES'99 Int. Workshop on Emerging Engineering and Connectionnist-based Information System*, Dunedin, New Zealand, 1999, pp. 192-196.

[10] L. Breiman, "Random forests", *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[11] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Boston, MA: Morgan Kaufmann, 2005.

[12] D. H. Wolpert, "Stacked generalization", *Neural Networks*, vol. 5, no. 2, pp. 241-259, 1992.

[13] J. Bansiya and C.G. Davis, "A hierarchical model for object-oriented design quality assessment", *IEEE Trans. Softw. Eng.,* vol. 28, no. 1, pp.4-17, 2002.

[14] C. Chen, A. Liaw, and L. Breiman. "Using random forest to learn imbalanced data", University of California, Berkeley, 2004.

[15] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest", *19th IEEE Conf. on Tools with Artificial Intelligence*, Patras, Greece, 2007, pp. 311-317.