

Fuzzy Rule Base System for Software Classification

Adnan Shaout* and Juan C. Garcia+

The Electrical and Computer Engineering Department
The University of Michigan – Dearborn, Dearborn, Michigan

*shaout@umich.edu; +garcia.jc@umd.umich.edu

ABSTRACT

Given the central role that software development plays in the delivery and application of information technology, managers have been focusing on process improvement in the software development area. This improvement has increased the demand for software measures, or metrics to manage the process. This metrics provide a quantitative basis for the development and validation of models during the software development process. In this paper a fuzzy rule-based system will be developed to classify java applications using object oriented metrics. The system will contain the following features:

Automated method to extract the OO metrics from the source code,

Default/base set of rules that can be easily configured via XML file so companies, developers, team leaders, etc, can modify the set of rules according to their needs,

Implementation of a framework so new metrics, fuzzy sets and fuzzy rules can be added or removed depending on the needs of the end user,

General classification of the software application and fine-grained classification of the java classes based on OO metrics, and

Two interfaces are provided for the system: GUI and command.

KEYWORDS

Fuzzy Based Rule model, Object Oriented Principles, Object Oriented Metrics, Java Patterns, Transitive Closure Relation, Decomposition Trees, Software classification, Software reliability.

1. INTRODUCTION

With the development of Object-Oriented (OO) paradigm since the early 1990s the development and use of metrics has been growing. Several studies and research papers were dedicated to the study of OO metrics and research of tools using these metrics. In 1994 Chidamber [1] developed and implemented a set of six metrics for OO design: response for a class (RFC), weighted methods per class (WMC), coupling between objects (CBO), lack of cohesion (LCOM), number of children (NOC), and depth of inheritance tree (DIT). These metrics are described in section 2 of this paper.

Despite the number of investigations in several areas and the development of some tools to gather metrics, OO metrics haven't been widely adopted by the software development community. This seems to be due to the following factors:

- As pointed by Ampatzoglou and Chatzigeorgiou [4], Sarkar et al [8] some metrics are collected manually, or there is manual intervention during their collection or preprocessing.
- Chidamber and. Kemerer[1], Rosenberg [2], Ampatzoglou, Chatzigeorgiou [4], Sarkaret al [8] showed that the metrics seem to be independent from each other and managers/leaders/architects have to analyze metrics separately.
- In their research Pizzia, Pedrycz [7], K. Elish and M. Elish [6] showed there are usually complex methodologies that need to be applied after the metrics are extracted in order to obtain the analysis, results and prediction of the system.
- Thwin and Quah [3], Quah [5], K. Elish and M. Elish [6], and Pizzia and Pedrycz [7] demonstrated that the metrics computed with other factors help predict the reliability and quality of the system, but the metrics haven't been used to produce the classification of the system.
- Additionally as presented by Virtual Machinery [9] in their application the JHawk, the results of the metrics give a complexity analysis and statistical information, but do not produce any classification or suggestion how to reduce the value of the metrics.

Due to these factors and because of Object Oriented metrics present concepts like loosely or tightly coupled, high or lack of cohesion, etc. that provide unsharp boundaries and allow gradual transition closer to human interpretation we propose to develop a system to classify java applications based on OO metrics. The system will contain the following features:

- Automated method to extract the OO metrics from the source code
- Default/base set of rules that can be easily configured via XML file so companies, developers, team leaders, etc, can modify the set of rules according to their needs.
- Implementation of a framework so new metrics, fuzzy sets and fuzzy rules can be added or removed depending on the needs of the end user.
- General classification of the software application and fine-grained classification of the java classes based on OO metrics.
- Two interfaces are provided for the system: GUI and command.

The paper is organized as follows: Section 2 shows the definition of the traditional and object oriented metrics utilized. Section 3 shows the software application design including use cases, sequence and class diagrams. Section 4 shows the design of the fuzzy system including block diagram and the definition of membership functions and fuzzy rules. Section 5 shows the data of the applications being evaluated. Section 6 shows the experiments and results of the fuzzy system compared to those of a manual analysis. Finally section 7 concludes and presents future work for this paper.

1.1 Background

Many traditional and object oriented metrics extract information of the application regarding traditional principles and object oriented principles like complexity, inheritance, coupling, cohesion, polymorphism, etc. The following is the description of the metrics that will be used in this paper:

Lines of Code – LOC: This is a traditional metrics and counts all lines within the class including blank lines, command lines and comment lines. Size of a class is used to evaluate the ease of understanding of code during development and maintenance [2]. The rationale of this class is that

high number of lines of code increases complexity of the code, making it difficult to understand, maintain and test.

Weighted Methods per Class – WMC: This is object oriented metric was developed by Chidamber [1], and counts the number of methods implemented within a class or the sum of the complexities of the methods [1]. The rationale of this metric is that classes with many methods are likely to be more application specific, limiting the possibility of reuse [2].

Response for a Class – RFC: This object oriented metric counts the set of methods that can be invoked in response to a message to an object of the class or by some method in the class [1]. The rationale of this metric is that the larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class. Therefore testing and debugging becomes complicated since it requires a greater level of understanding from the tester [2].

Lack of Cohesion – LCOM2: It counts the percentage of methods that do not access a specific attribute averaged over all attributes in the class [1]. For this metric a low cohesion increases complexity, thereby increasing the likelihood of errors during the development process. Equation (1) shows the LCOM2 metric.

$$LCOM2 = 1 - \frac{\sum(mA)}{m * 1} \quad (1)$$

where mA is the number of methods that access a variable, m is the number of methods in a class and A is number of variables (attributes) in a class.

Coupling Between Object Classes – CBO: This metric counts the number of other classes to which a class is coupled [1]. Excessive coupling is detrimental to modular design and prevents reuse. Therefore the tighter the coupling the more sensitivity are the changes in other parts of the application [2].

Depth of Inheritance Tree – DIT: This metric measures the maximum inheritance path from the class to the root class [1]. The deeper a class within the hierarchy the greater the number methods it is likely to inherit making the code more complex to predict its behavior. Deeper trees constitute greater design complexity, since more methods and classes are involved [2].

Number of Children - NOC: This metric count the number of immediate subclasses subordinate to a class in the hierarchy. NOC and DIT are closely related because NOC measures the breadth of a class hierarchy, where maximum DIT measures the depth [1]. A high value of this metric increases the likelihood of improper abstraction and the probability of misusing sub classing [2].

Method Hiding Factor – MHF: This metric measures how variables and methods are encapsulated in a class in relation to all the classes in the application. The invisibility of a method is the percentage of the total classes from which this method is not visible. The ideal value is between 8% and 24%. A low value indicates insufficiently abstracted implementation and a high value indicates very little functionality. The larger the proportion of methods unprotected the higher the probability of errors [13]. Equation (2) shows the MHF metric.

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)} \quad (2)$$

where TC is the total number of classes, M_h is the number of methods hidden and M_d is the number of methods defined in a class.

Attribute Hiding Factor – AHF: this metric measure how variables are encapsulated in a class in relation to all the classes in the application. The invisibility of an attribute is the percentage of the total classes from which this method is not visible. Encapsulation indicates that the attributes should have no visibility to other classes therefore the ideal value for this metric is 100% [10]. Equation (3) shows the AHF metric.

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)} \quad (3)$$

where A_h is the number of attributes hidden and A_d is the number of attributes defined in a class.

Method Inheritance Factor – MIF: this metric measure the inherited methods in a class in relation to all the classes in the application. For this metric a very high value indicates superfluous inheritance, wide member scopes. Low value indicates lack of inheritance and heavy use of Overrides. The ideal value for this metric should be between 20%-80% [13]. Equation (4) shows the MIF metric.

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \quad (4)$$

where M_i is the number of methods inherited and M_a is the number of methods defined in a class.

Attribute Inheritance Factor – AIF: This metric measure the number of attributes inherited in a class in relation to all the classes in the application. The ideal value for this metrics is between 0% and 48%. A very high value indicates superfluous inheritance and wide member scopes. A low value indicates lack of inheritance and heavy use of Overrides [10]. Equation (5) shows the AIF metric.

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)} \quad (5)$$

where A_i is the number of attributes inherited and A_a is the number of attributes defined in a class.

Coupling Factor – COF: It measures the actual coupling among classes in relation to the maximum number of possible couplings. The ideal value for COF is between 0% and 12%. A very high value should be avoided because tightly coupled relations increase complexity, reduce encapsulation, reduce potential reuse, and limit understandability and maintainability [10]. Equation (6) shows the COF metric.

$$COF = \frac{\sum_{i=1}^{TC} [\sum_{j=1}^{TC} is_client(C_i, C_j)]}{TC^2 - TC} \quad (6)$$

where $is_client(C_i, C_j)$ is 1 if C_j is a client of C_i , otherwise is 0.

Polymorphism Factor – POF: This metric measures the degree of method overriding in the class inheritance tree. Polymorphism should be used to a reasonable extent to keep the code clear, but excessively polymorphic code is too complex to understand. Equation (7) shows the POF metric.

$$POF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) * DC(C)]} \quad (7)$$

where M_o is the number of methods overridden and M_n is the number of new methods defined in a class.

2. APPLICATION DESIGN

Use case Diagram: For the application, five main use case diagrams were designed: Run Diagnose, Load Configuration, Extract Metrics, Fuzzy Diagnose, and Generate Report. These use cases are shown in figure 1 below.

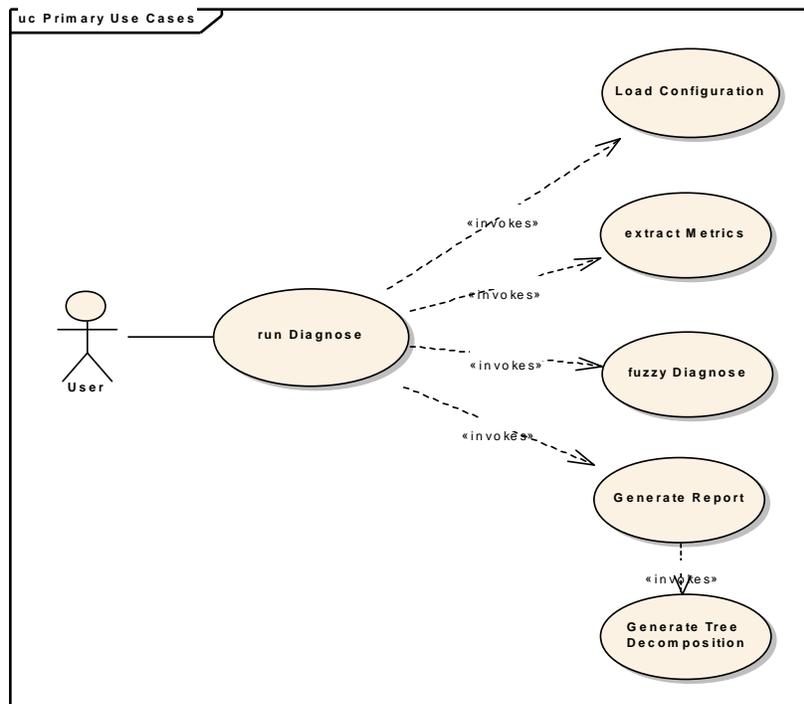


Figure 1. Use Case Diagram

Run Diagnose is the main use case diagram and orchestrates the execution of all the other use cases. Load Configuration loads the configuration of the fuzzy system; this configuration supports fuzzy sets, fuzzy rules and definition of OO Metrics. In addition, this use case loads the information of the classes, class variables methods, variables methods, etc., and it is utilized during the calculation of the OO metrics. Fuzzy Diagnose use case basically processes all the fuzzy rules. It computes two outcomes: fine-grained report for each of the classes within the application and a comprehensive report for the entire application. Finally Generate Report use case generates a classification report including a decomposition tree in two formats: XML and screen.

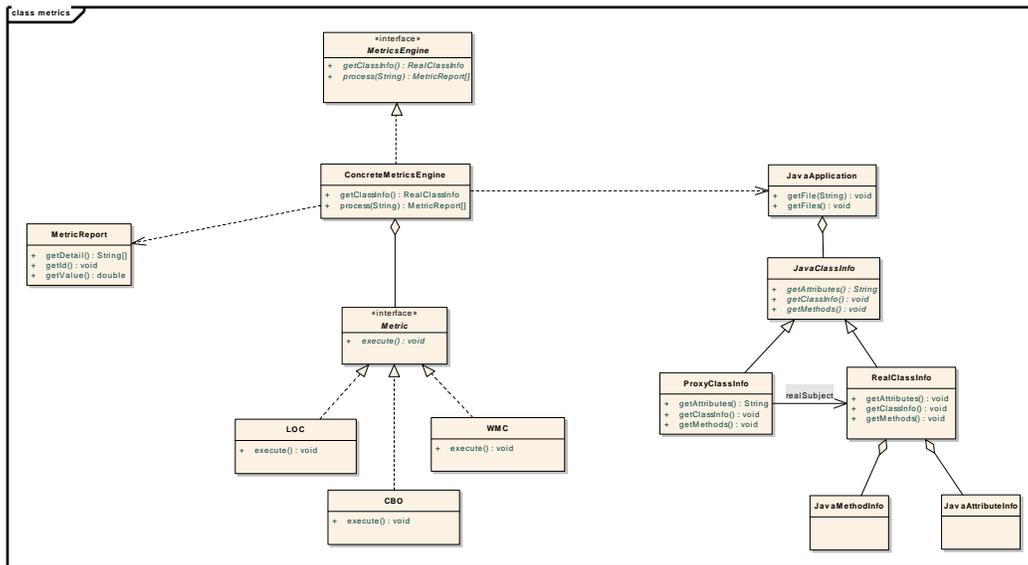


Figure 3. Class Diagram – OO Metrics Subsystem

3. FUZZY SYSTEM DESIGN

3.1 Design of the diagnosis system

The objective of the system is to classify the reliability and potential design flaws of java applications based on OO Metrics. Figure 4 shows the block diagram of the overall system.

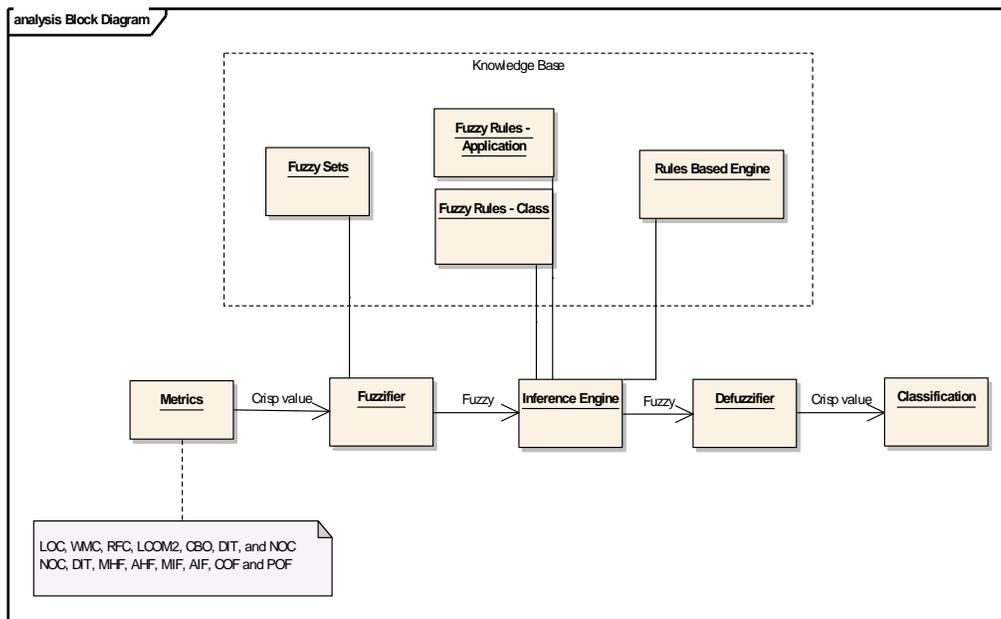


Figure 4. Block Diagram of the fuzzy system

The input of the system is the OO metrics extracted from java application. The Fuzzifier calculates the matching degree of the metrics that matches the condition of the fuzzy rules. The Inference Engine calculates the rule's conclusion based on its matching degree by using the clipping method, and combines the conclusion inferred by all fuzzy rules in a final conclusion. Finally, the Defuzzifier process converts the fuzzy conclusion into a crisp value by using the mean of max method. There are two types of results given by the system: A fine-grained classification for each of the classes and a general classification for the java application being evaluated [11]. For this reason two sets of rules are defined: one set of rules to classify single classes and another one to evaluate the application. Moreover a decomposition tree is generated for all the classes reported during the fined-grain classification. This tree will help the developer to analyze and address classes with similar values. Unfortunately due to performance constraints only systems that report less than 200 classes will generate this similarity tree.

3.2 Input variables

The input variables of the system are the following Object Oriented metrics:

LOC = Lines of code; WMC = Weighted Methods per Class

RFC = Response for a Class; LCOM2 = Lack of Cohesion

CBO= Coupling Between Object Classes; NOC= Number of Children; DIT = Depth of Inheritance Tree; MHF= Method Hiding Factor; AHF= Attribute Hiding Factor; MIF= Method Inheritance Factor; AIF= Attribute Inheritance Factor; COF= Coupling Factor;

POF= Polymorphism Factor

3.3 Output variables

The output variable of the system is defined as:

OODC= Software classification.

3.4 Definitions

The definition of the values used in the fuzzy sets and fuzzy rules are: C- Critical, H – High, M – Medium, N – Normal, L – Low, VL – Very Low

3.5 Membership function definition

The membership functions have been designed based on empirical results presented by Chidamber [1], Rosenberg [2] and Briand [13]. The design of the membership functions has followed the conditions proposed by Yen and Langary [11] where “each function overlaps only with the closest neighboring membership function and for any possible input data, its membership values in all relevant fuzzy sets should sum to 1 or nearly so”. The following is the definition for each of the metrics:

WMC: For this metric three membership functions were designed: normal (x: 10, 20), medium (x: 10, 20, 30) and high (x: 20, 30). The normal value was chosen based on two observations: Rosenberg's experiment [2] showed a histogram with values between 0 and 20 for most of the classes; on the other hand Chidamber [1] reported most of the cases with values between 0 and 10. As a result the values for the normal membership function are chosen with values between 0

and 20. The medium and high membership functions are derived based on Yen and Langary [11] using an overlap of 10. The membership functions are shown in the figure 5.

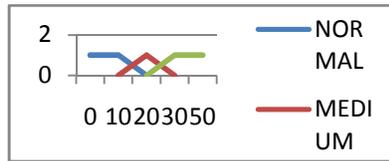


Figure 5. WMC membership functions

RFC: For this metric three membership functions were created: normal (x: 30, 40), medium (x: 30, 40, 50) and high (x: 40, 50). The empirical data from Rosenberg [2] showed that the majority of classes only invoke between 0 and 40 methods; on the other hand Chidamber [1] reported a median value between 6 and 29. Based on this information the normal membership function is defined with values between 0 and 40; and medium and high fuzzy sets are derived based on Yen and Langary [14]. The membership functions are shown in figure 6.

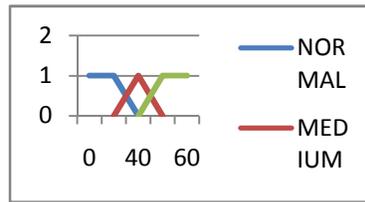


Figure 6. RFC membership functions

LCOM2: For this metric three membership functions were created: normal (x: 60, 70), medium (x: 60, 70, 80) and high (x: 70, 80). Rosenberg [2] did not present statistical data for this metric however she said that “smaller LCOM to its maximum value the better”. On the other hand Briand [13] obtained a median value 64% and min value 18%. The normal value is derived based on this median and the other two membership functions are derived based on Yen and Langary [11]. The membership functions are shown in figure 7.

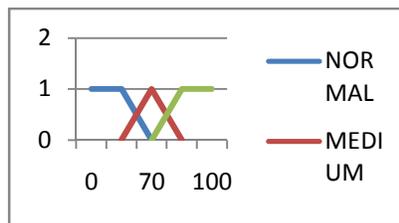


Figure 7. LCOM2 membership functions

CBO: For this metric three membership functions were defined: normal (x: 5, 10), medium (x: 5, 10, 15), high (x: 10, 15). Rosenberg [2] reported that more than one-third of the classes reported values between 0 and 10 and fewer classes between 11 and 13. On the other hand Chidamber [1] obtained a median value between 0 and 9. Based on this information the normal membership function is defined with values between 0 and 10; the other fuzzy sets are derived based on Yen and Langary [11] using overlapping of 5. The membership functions are shown in figure 8.

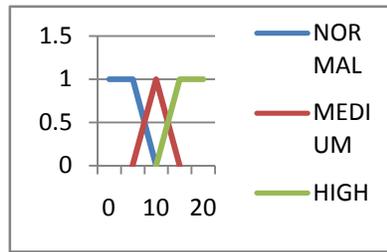


Figure 8. CBO membership functions

DIT: For this metric three membership functions were defined: normal (x: 3, 6), medium (x: 3, 6, 9) and high (x: 6, 9). Rosenberg [2] reported 60% classes with a DIT less than or equal to 1, 20% between 2 and 3; and only 5% greater than 5. Chidamber [1] on the other hand reported a maximum DIT of 10, and a median value between 1 and 3. Based on this information the normal membership function is chosen between 0 and 6 and the other membership functions are derived based on Yen and Langary [11]. The membership functions are shown in figure 9.

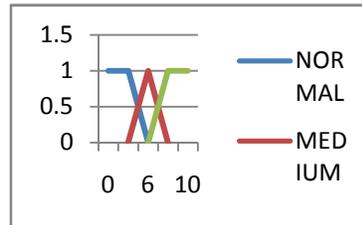


Figure 9. DIT membership functions

NOC: For this metric three membership functions were created: normal (x: 10, 20), medium (x: 10, 20, 30) and high (20, 30). For this metric Rosenberg [2] reported that most of the classes had between 0 and 10 children, and fewer classes between 10 and 20 children. On the other hand Chidamber [1] reported that most of the classes did not have children, and that the maximum value obtained in this metric was between 42 and 50. Giving an overlap of 10 the normal membership function is chosen with values between 0 and 20; and the other fuzzy sets are derived based on Yen and Langary [11]. The membership functions are shown in figure 10.

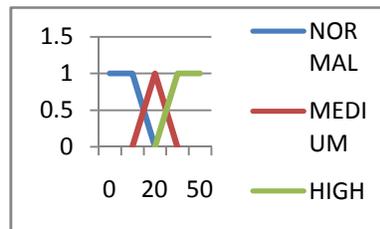


Figure 10. NOC membership functions

LOC: for this metric three membership functions were defined: normal (x: 750, 1000), medium (x: 750, 1000, 1250) and high (x:1000,1250). The fuzzy sets for this metric have been defined based on intuition rather than empirical or theoretical data. As per definition of this metric a class with a low number of lines will be less complex and easier to maintain and test. We considered a class with a normal value to be between 0 and 1000 lines of code. The other membership

functions are derived with an overlap of 250 based on Yen and Langary [11]. The membership functions are shown in figure 11.

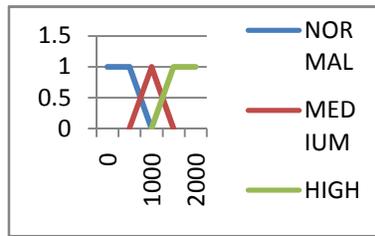


Figure 11. LOC membership functions

MHF: For this metric five membership functions were created: very low (x: 5, 10), low (x: 5, 10, 15), normal (x: 10, 15, 20, 25), medium (20, 25, 30) and high (x: 25, 30). The normal membership function was defined based on the statistical distribution reported by Brito et al [10]. In this case most of the cases and contained values between 8-25%. Therefore a normal fuzzy set is defined using trapezoidal form with values 10,15,20,25. The other membership functions are derived based on Yen and Langary [11] using an overlap of 5. The membership functions are shown in figure 12.

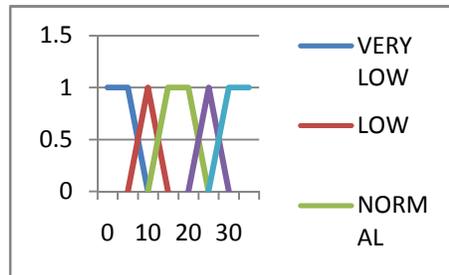


Figure 12. MHF membership functions

AHF: For this metric three membership functions were defined: normal (x: 80, 90), medium (70, 80, 90) and high (70, 80). Due to the encapsulation paradigm Brito et al [10] concluded that a good value for this metric should be 100%. However taking into account static variables shared across different classes then the normal membership function is defined with values between 80 and 100. The other membership functions are defined based on Yen and Langary [11] using an overlap of 10. The membership functions are shown in figure 13 below.

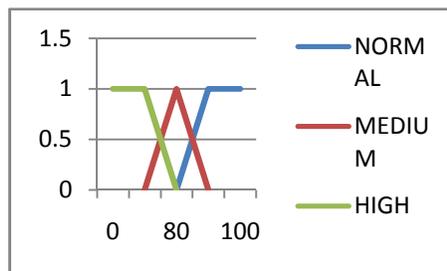


Figure 13. AHF membership functions

MIF: For this metric five membership functions were defined: very low (x: 15, 20), low (15, 20, 25), normal (x: 20, 25, 75, 80), medium (x: 75, 80, 85) and high (80, 85). Brito et al [10] reported an average of 85% among 5 applications and suggested a metric value between 20% and 80%. Therefore the normal membership function is defined in a trapezoidal form with values 20, 25, 75 and 80. The other fuzzy sets are derived based on Yen and Langary [11] using an overlap of 5. The membership functions are shown in figure 14 below.

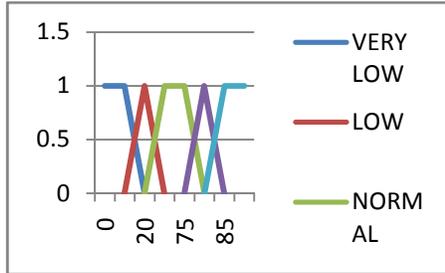


Figure 14. MIF membership functions

AIF: For this metric three membership functions were defined: normal (x: 40, 50), medium (x: 40, 50, 55, 65) and high (x: 55, 65). Brito et al [10] reported a statistical distribution with most classes between 50 and 65%, maximum value 80% and minimum value of 40%. They also suggested a metric value between 0 and 48%. For this reason a normal membership function is defined using a trapezoidal form with values 40, 50, 55, 65 and the other membership functions are derived based on Yen and Langary [11] with an overlap of 10. The membership functions are shown in figure 15.

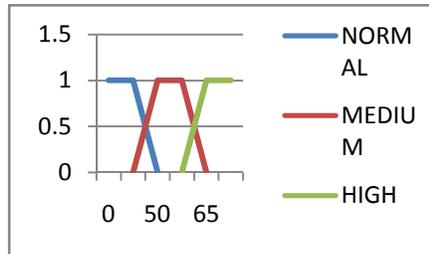


Figure 15. AIF membership functions

COF: For this metric three membership functions were created: normal (x: 10, 20), medium (x: 10, 20, 30) and high (x: 20, 30). Brito et al [10]. The reported statistical distributions with most values are between 5% and 15%, the maximum value is 30% and the minimum value is 3%. They also suggested an ideal value of less than 12%. Therefore a normal membership function is defined with values between 0 and 20 and the other membership functions are derived based on Yen and Langary [11] using an overlap of 10. The membership functions are shown in figure 16.

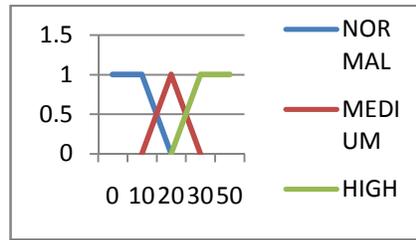


Figure 16. COF membership functions

POF: For this metric three membership functions were created: normal ($x: 0, 10$), medium ($x: 0, 10, 20$) and high ($x: 10, 20$). Brito et al [10] reported two tendencies for this metric: one of them states that POF should be greater than 10% due to polymorphism increases the flexibility of the application, and the other one states that POF should be less than 10% because complexity increases testability, maintainability and decreases understandability. In my opinion having very low polymorphism defeats an important principle of object oriented programming, therefore a normal membership function is chosen with values between 10 and 100. The other membership functions are derived based on Yen and Langary [11] using an overlap of 10. The membership functions are shown in figure 17.

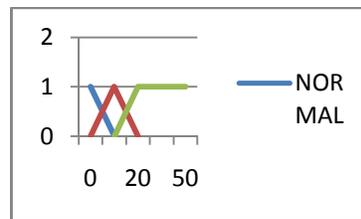


Figure 17. POF membership functions

OODC: The output membership function has been defined with three membership functions: critical ($x: 80, 90, 100$), high ($x: 70, 80, 90$) and medium ($x: 60, 70, 80$). If the matching degree does not fall within these fuzzy sets then it is considered normal and not reported. The membership functions are shown in figure 18.

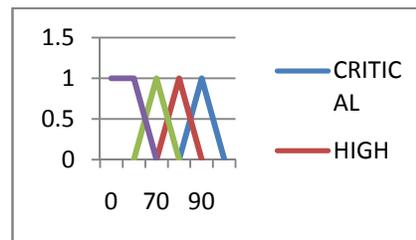


Figure 18. OODC – Output membership functions

3.6 Fuzzy Rules Definition

The fuzzy rules are divided in two groups: fuzzy rules to classify single java classes and fuzzy rules to classify the entire java application. The metrics to classify single java classes are: LOC, WMC, RFC, LCOM2, CBO, DIT, and NOC. These metrics gather information about complexity,

cohesion, coupling and hierarchical tree of single classes. The metrics to classify the entire application are: NOC, DIT, MHF, AHF, MIF, AIF, COF and POF; and they gather information about encapsulation, inheritance, coupling and hierarchical tree structure of the entire application. Both groups of metrics are complementary and their results provide detail and general information about the application. It is worth to mention that despite of DIT and NOC do not gather information at application level, their maximum value is used during the classification of the application due to these metrics have a direct impact in the hierarchical structure of the application.

The metrics are also divided in groups based on their objectives. Metrics that share the same objective are grouped together and metrics that do not share same objective are left alone. Table 1 shows the results of this clustering.

Table 1. Metrics classification

Group #	Metrics	Objective
1	LOC, WMC, RFC	Complexity
2	DIT, NOC	Hierarchical tree.
3	LCOM2	Cohesion
4	CBO, CFO	Coupling
5	MIF, AIF	Inheritance
6	MHF, AHF	Encapsulation
7	POF	Polymorphism

These groups are used during the definition of the fuzzy rules. A single condition is defined by one cluster and the entire fuzzy rule is defined with several conditions. The evaluation of the metrics within the cluster is performed by using the OR command, and the evaluation of the clusters within the fuzzy rule is performed by using the AND command. For example rule R1 defined as:

R1 → IF (LOC IS HIGH OR WMC IS HIGH OR RFC IS HIGH) AND (DIT IS HIGH AND NOC IS HIGH)... THEN

is evaluated within the fuzzy context using equation (8):

$$R_1 = \max \left[\begin{array}{l} \min\{\mu_A(x_1), \mu_B(x_2), \mu_C(x_3)\}, \\ \min\{\mu_D(y_1), \mu_E(y_2)\} \dots \end{array} \right] \quad (8)$$

The next two sections explain the definition of the fuzzy rules for single java classes and for the entire application.

Fuzzy Rules for Classification of Single Java Classes: These rules use the membership functions LOC, WMC, RFC, LCOM2, CBO, DIT, and NOC; and classify the class as critical, high or medium. Table 2 explains the conditions for each of the classifications.

Table 2. Conditions for class classification

Fuzzy Consequent	Fuzzy Conditions
Critical	At least three of the clusters being evaluated have high value;
High	At least two of the clusters being evaluated have high value;
Medium	At least one of the clusters being evaluated have high value;

As shown in the table a rule with critical consequent evaluates if at least three of the clusters within the fuzzy rule have a high value. A class under this classification has a very poor object oriented design and does not follow at least three OO metrics. This class can impact other classes within the application and a considerable amount of work is expected to address all the metrics. For this reason a critical classification should trigger immediate attention of the software developer and technical leader for review and modification.

A rule with high classification evaluates if at least two of the clusters within the fuzzy rule have a high value. A class under this classification does not conform to at least two of the metrics, therefore testing and maintaining the class can become a challenge. A class under this classification will most probably move to critical instead of moving to medium or normal stage, as a result this classification should trigger the attention of the developer, architect and technical leader for review.

Finally a rule with medium consequent evaluates if at least one of the clusters has a high value. A class under this classification should be reviewed and verified to make sure that there are no potential design issues. Java classes under this classification will most probably move to high or critical instead of moving back to normal. This classification should trigger the attention of the developer for verification. The rules derived are shown in table 3.

Table 3. Fuzzy rules for class classification

Rule#	COMPLEXITY			INHERITANCE TREE		COHESION	COUPLING	Results
	LOC	WMC	RFC	DIT	NOC	LCOM2	CBO	
1	H			H M N		H M N	H M N	medium
2		H M N		H		H M N	H M N	medium
3		H M N		H M N		H	H M N	medium
4		H M N		H M N		H M N	H	medium
5	H			H		H M N	H M N	high
6	H			H M N		H	H M N	high
7	H			H M N		H M N	H	high
8		H M N		H		H	H M N	high
9		H M N		H		H M N	H	high
10		H M N		H M N		H	H	high
11	H			H		H	H M N	critical
12	H			H		H M N	H	critical
13	H			H M N		H	H	critical
14		H M N		H		H	H	critical

Fuzzy Rules for Application Classification: Similar to the fuzzy rules at the class level, the rules of the application have critical, high or medium classification. These rules use the membership functions: NOC, DIT, MHF, AHF, MIF, AIF, COF and POF. Table 4 shows the conditions for each of the classifications mentioned.

Table 4. Conditions for application classification

Fuzzy Consequent	Fuzzy Conditions
Critical	At least three clusters have a high value;
High	At least two clusters have a high value;
Medium	At least one cluster has a high value.

As shown in the table a rule with critical fuzzy set consequent evaluates if at least three clusters in the fuzzy rule have high value. Most probably the application has a very poor object oriented design and a considerable amount is needed to redesign the application. This classification should trigger immediate attention of the designer, architect, leader or project manager.

A rule with high fuzzy set consequent evaluates that at least two of the clusters in the fuzzy rule have high value. If the result of the application falls under this classification then a review must be performed and high values addressed. Medium values on the other hand should be reviewed for potential redesign and modification. Usually an application under this classification will most probably move to critical stage instead of medium or normal. This classification should trigger the attention of the designer, architect, project leader or project manager for review of the design and correction of the metrics.

Finally a rule with medium fuzzy set consequent evaluates if at least one of the clusters within the fuzzy rule has high value. An application under this classification should be reviewed for evaluation and verification. Without revision the application most probably will move to high or critical instead of moving back to normal. This classification should trigger the attention of the designer, leader or project manager for review. The rules derived are shown in table 5:

Table 5. Fuzzy rules for application classification

Rule#	Inher. Tree		Inher. Class		Encapsulation		Coupling		Polymorphism		Results
	NOC	DIT	MIF	AIF	MIF	AIF	COF	POF	COF	POF	
1	H		H MN		H MN		H MN		H MN		medium
2	H MN		H		H MN		H MN		H MN		medium
3	H MN		H MN		H		H MN		H MN		medium
4	H MN		H MN		H MN		H		H MN		medium
5	H MN		H MN		H MN		H MN		H		medium
6	H		H		H MN		H MN		H MN		high
7	H		H MN		H		H MN		H MN		high
8	H		H MN		H MN		H		H MN		high
9	H		I MN		I MN		I MN		I MN		high
10	H MN		H		H		H MN		H MN		high
11	H MN		H		H MN		H		H MN		high
12	H MN		H		H MN		H MN		H		high
13	H MN		H MN		H		H		H MN		high
14	I MN		I MN		I MN		I MN		I MN		high
15	H MN		H MN		H MN		H		H		high
16	H		H		H		H MN		H MN		critical
17	H		H		H MN		H		H MN		critical
18	H		H		H MN		H MN		H		critical
19	I MN		I MN		I MN		I MN		I MN		critical
20	H		H MN		H		H MN		H		critical
21	H		H MN		H MN		H		H		critical
22	H MN		H		H		H		H MN		critical
23	I MN		I MN		I MN		I MN		I MN		critical
24	H MN		H MN		H		H		H		critical

4. DATA ANALYSIS AND DATA PREPARATION

Three applications were diagnose during the experiments: two of them were provided by CGI, one of the largest IT companies in Canada, and the other one is the java application developed for this project. The experiments were executed using most of the java classes; however JUnit test classes and Exception classes were excluded from the experiments. Because of its nature these classes do not follow object oriented design principles, therefore can affect the results of the metrics. Table 6 shows detail information of the applications.

Table 6. Java applications used during the experiment

Application	# Packages	# Classes	Lines of Code
OODiagnose	6	90	6088
BIE Portal	43	842	77395
ETLF	4	45	2497

Manual analysis of the application was performed using histograms of the metrics, class diagrams and java code. For this analysis concepts like rigidity, fragility, immobility and viscosity were used to classify the application. Rigidity states that a simple change causes a cascade change in the dependent modules. Fragility is defined as the tendency of a program to break in many places when a single change is made. Immobility is the unsuccessful software reuse of the same design. And viscosity is when usability and employability of the existing methods is very poor making viscosity of the design very high [14]. These results were compared to those of the fuzzy application for validation.

5. EXPEREIMENTS AND RESULTS

5.1 OO Diagnose application

Results of the Fuzzy System

During the diagnosis of the application the fuzzy system classified the application as normal. Only MHF and NOC metrics were categorized as medium and the other metrics as normal. Table 7 shows the details of the metrics reported under this classification. These results showed a high polymorphism, low coupling, good encapsulation, normal inheritance class and normal inheritance tree reinforcing good object oriented principles.

Table 7. Classification of the Diagnose Application

Metric	Classification	Value
AHF	Normal	96.80842
AIF	Normal	35.80247
COF	Normal	3.655041
DIT	Normal	2
MHF	Medium	24.18428
MIF	Normal	69.55381
NOC	Medium	15
POF	Normal	90.625

Regarding the diagnosis of the classes, the system reported 21 classes out of 90 classes being evaluated: 3 of the classes with high and 18 medium classification. Figure 19 shows the Classes vs. Classification per Metric. LCOM2 had the highest number of classes reported with 20 out of 21 classes with high and medium classification. On the other hand, DIT, LOC and NOC reported normal classification for all the classes. As a result, the classes did not have a high complexity, the inheritance was kept under control, and the coupling was low. The only concern was the high of classes reported with multiple responsibilities.

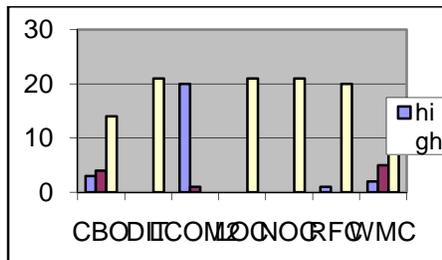


Figure 19. Classes Reported vs Classification per Metric

The system also generated a decomposition tree for the classes reported. As shown in figure 20, twenty one levels were generated, and the classes were grouped depending on the similarity of the metrics. Two sample groups were verified:

1. Similarity group 2.76190476190476 reported JavaClassInfo and FuzzyRulesEngine within the same group. They had normal CBO, DIT, LOC, RFC and WMC. One of the classes reported medium and the other one high LCOM2.
2. Similarity group 1.78061224489796 reported DecompositionTreeAlgorithm and RFC within the same group. They had high LCOM2, and normal DIT, LOC, RFC and WMC. Once class had medium and the other normal CBO.

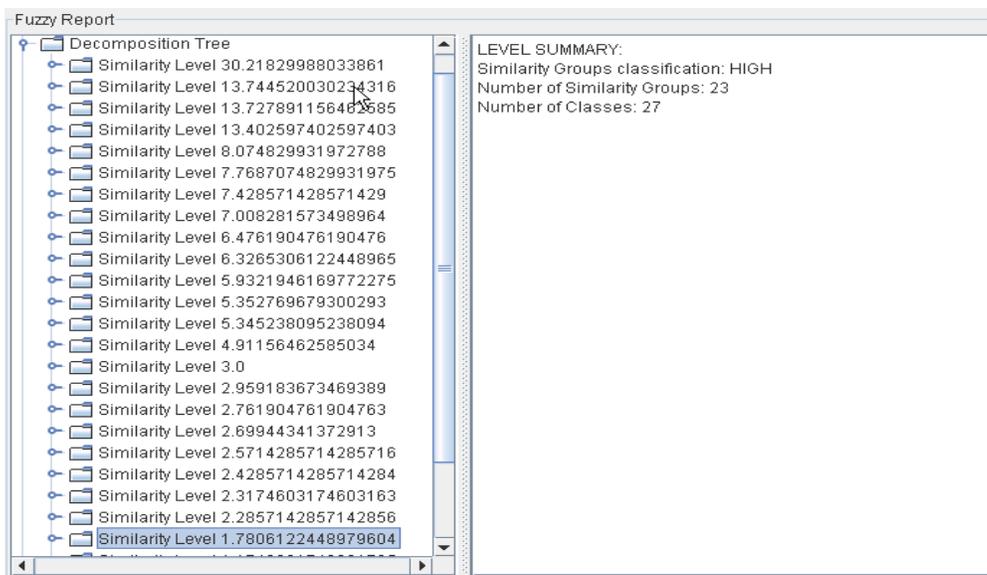


Figure 20. Similarity Groups- OO Design Application

Results of the Manual Analysis

During the manual analysis the rigidity, fragility, immobility and viscosity of the application showed low values based on the results of the metrics shown in figure 21. The inheritance of the attributes (AIF) with 40% seems to be high because all the attributes should be encapsulated.

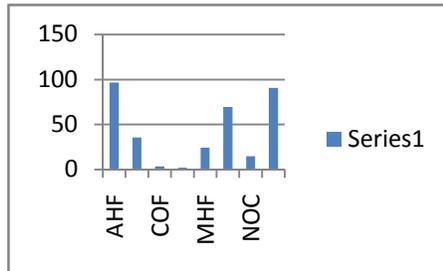


Figure 21. Metrics Results used during classification of the application.

The application seems to have “top heavy” architecture because DIT and NOC have lower values keeping the inheritance under control. Application seems to be very flexible because POF and MIF have high values. Due to these values it is suggested that the application has high inheritance and high polymorphism. Revision of the source code demonstrated that this is due to the usage of bridge and strategy pattern [15].

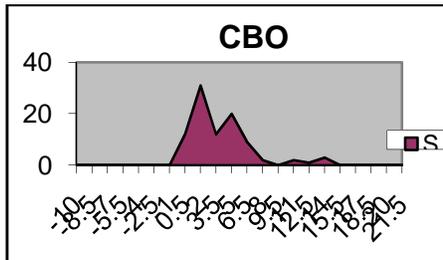


Figure 22. Histogram – CBO metric

Regarding the classification of the java classes the following observations draw the attention during the verification: In general the classes seem to be well written however coupling (CBO) seems to have a couple of outlier classes that need to be reviewed. This is shown in figure 22.

Comparison and Discussion

The results of the manual analysis and the fuzzy system are comparable. In general both results reported a relatively good object oriented design. Regarding the fine-grained details both results reported high lack of cohesion. There seems to be a problem with LCOM2 metric because false-positives are being reported. During manual verification of the source code *java bean* objects are being reported with medium and high values despite of the fact that they do have single responsibility, low coupling and high encapsulation. *Java Beans* are reusable objects utilized in java to represent objects and follow conventions about method naming, construction and behavior; therefore these classes should be valid objects with normal cohesion [12].

The only difference between the diagnosis procedures is that low values in deep of inheritance tree (DIT) and number of children (NOC) were detected during the manual analysis of the histograms. The fuzzy rules seem to be overseeing low values for these metrics; however this appears to be a subjective assessment. As pointed by Rosenberg [2] higher values indicate higher complexity which affects the maintainability and testability of the classes and therefore the application [2]. The metrics provide a trade-off and their values should be assigned depending on human experts, company policies, etc.

5. CONCLUSION AND FUTURE WORK

In this paper we developed a software system to diagnose the reliability of java applications using object oriented metrics and a fuzzy rule-based system. The fuzzy membership functions and fuzzy rules have been defined using statistical data from previous studies that have defined and analyzed the different object oriented metrics. Three applications for different business purposes and sizes have been analyzed, and results of the fuzzy system have been compared to those of a manual analysis. The following can be inferred from the experiment:

- The fuzzy system has an appropriate default set of fuzzy sets and fuzzy rules to classify object oriented java applications.
- The decomposition tree is a very useful analysis tool for developers who need to address issues with classes with similar metric values.
- The results help the developers, designers and team leaders to enforce the use of object oriented principles in the design and development of java applications.
- Unfortunately the fuzzy system does not prevent metrics to report false-positives therefore manual analysis is needed in cases were abnormal results are suspected.
- Overall the current fuzzy sets and fuzzy rules provide accurate results however the system does not report low values in NOC and DIT at the application level, therefore modifications of these rules and fuzzy sets are expected if these values need to be considered by the final user.
- The fuzzy rules did not entirely utilize medium and normal membership functions, but these fuzzy sets are provided so the user can modify the current fuzzy rules if a more accurate result is needed.
- The fuzzy system provides objective results because they contain information from statistical sources and several human experts in contrast to manual analysis that is bias and can vary depending on the knowledge and experience of the expert.

The following suggestions are provided for future work:

- Integration of the fuzzy system with the popular java compiler *ant*, to obtain instant results at compilation time.
- Include a neural network prediction system to forecast the reliability of the applications using statistical and historical information of the fuzzy reports.
- Integrate the fuzzy system with a continuous monitoring system (Hudson dashboard, etc) so historic and current reports are available to developers, project leaders, architects, managers and clients in order to increase productivity, reliability, usability, testability of the application.

REFERENCES

- [1] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. Soft. Eng., vol. 20, no. 6, pp. 476-493, Jun. 1994
- [2] L. Rosenberg, "Applying and Interpreting Object Oriented Metrics," in Soft. Tech. Conf. Utah, 1998.
- [3] M. M. Thwin and T. S. Quah, "Application of neural networks for software quality prediction using object-oriented metrics," J. Syst. Soft., vol. 76, pp. 147-156, Jun. 2004.
- [4] A. Ampatzoglou, and A. Chatzigeorgiou, "Evaluation of object-oriented design patterns in game development," Inform. Soft. Tech. vol. 49, pp. 445-454, Aug. 2006.
- [5] T. S. Quah, "Estimating software readiness using predictive models," J. Inf. Sci., vol. 179, pp. 430-445, Oct. 2008.
- [6] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," J. Syst. Soft., vol. 81, pp. 649-660, Oct. 2007.
- [7] N. J. Pizzia and W. Pedrycz, "Effective classification using feature selection and fuzzy integration," Fuz. Sets and Syst., vol. 159, pp. 2859-2872, Mar. 2008.
- [8] S. Sarkar, "Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software," IEEE Trans. Soft. Eng., vol. 34, no. 5, pp. 700-720, Sep. 2008.
- [9] Virtual Machinery. (2010). JHawk 5 Product Overview [Online]. Available: <http://www.virtualmachinery.com/jhawkprod.htm>
- [10] F. Brito, "The Design of Eiffel Programs: Quantitative Evaluation using the MOOD Metrics," INESC, Lisboa, Portugal, Proc. Tools'96 USA Rep., Jul. 1996.
- [11] J Yen and R. Langary, "Basic Concepts of Fuzzy Logic," in *Fuzzy Logic: intelligence, control, and information*, Upper Saddle River, NJ: Prentice-Hall, 1998, pp. 21-53.
- [12] G. Voss. (1996, Nov.). Java Beans: Introducing Java Beans [Online]. Available: <http://java.sun.com/developer/onlineTraining/Beans/Beans1/index.html>
- [13] L.C. Briand, "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems," Proc. 5th Int. Symp. Soft. Metr. 1998, pp. 20-21.
- [14] M. Sarker, "An overview of Object Oriented Design Metrics," M.S. thesis, Dept. Comp. Scn., Umeå Univ., Umeå, Sweden, 2005.
- [15] E Gamma, "Design Pattern Catalog," in *Design Patterns – elements of Reusable Object-Oriented Software*. Indianapolis, IN: Addison-Wesley, 1994, pp. 151-315.

Authors

Dr. Adnan Shaout is a full professor in the Electrical and Computer Engineering Department at the University of Michigan – Dearborn. At present, he teaches courses in fuzzy logic and engineering applications and computer engineering (hardware and software). His current research is in applications of software engineering methods, computer architecture, embedded systems, fuzzy systems, real time systems and artificial intelligence. Dr. Shaout has more than 29 years of experience in teaching and conducting research in the electrical and computer engineering fields at Syracuse University and the University of Michigan - Dearborn. Dr. Shaout has published over 140 papers in topics related to electrical and computer engineering fields. Dr. Shaout has obtained his B.S.c, M.S. and Ph.D. in Computer Engineering from Syracuse University, Syracuse, NY, in 1982, 1983, 1987, respectively.



Juan C. Garcia is a graduate student in the Electrical and Computer Engineering Department at the University of Michigan – Dearborn