Assessment of Programming Language Reliability Utilizing Soft-Computing

[1]Obi J.C. and [2]Akwukuma V.V.N.
[1,2]Department of Computer Science. University of Benin, P.M.B. 1154. Benin City. Nigeria.
[2]e-mail: vakwukwuma@yahoo.com; +234(0)8033440003;
***Corresponding author**:*
[1]*e-mail: tripplejo2k2@yahoo.com; +234(0)8093088218*

Abstract

Reliability is a critical program evaluation criterion so much so that program writing (expressing one's computational ideas) will be of no use if the end result is not consistent. Programmers cognizant pertaining to a particular programming language selected for solving a particular computational problem will yield the desired result. This can only be achieved if the program written in a particular programming language either structured (C, C++) or Objected-Oriented (Java, C#, Ruby) is gauge with the criteria's for evaluating program reliability. Adopting a soft-computing based approach will help model solutions to program reliability utilizing linguistic variables. Simulating a soft-computing system comprising of genetic algorithm and fuzzy logic for determining program language reliability was the rationale behind this research paper. The reliability of programming language could be determined in terms of "Reliable", "Moderate Reliable" and "Not Reliable" based on the possessed reliability criteria of the examined programming language.

Keywords

Fuzzy logic, Fuzzy set, Reliability, Soft-computing.

1.0 Introduction

Reliability is an essential program attribute. Reliability is closely linked with the quality of measurement which is determined by the "consistency" or "repeatability" of program measure (Jiantao, 1999). Program reliability is the probability of failure-free program operation for a specified period of time under a specified set of conditions or defined as producing expected results for all correct input or the ability of a program to perform to specification under all conditions (Robert, 2006). Program reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection (Jiantao, 1999).

The stability or consistency of scores over time or across raters determines reliability. Reliability pertains to *scores* not people. Thus, it is highly inadequate to associate reliability with persons.
Program (application) unreliability is usually tied in non-compliance with coding best practice. This non-compliance can be detected by measuring the static quality attributes of an application

program reliability criteria's. Assessing the static attributes underlying an application's (program) reliability provides an estimate of the level of program risk and the likelihood of potential application failures and defeats the application will experience when placed in operation (CISA, 2010).

Determining programming language reliability is a pivoted point of this research utilizing a simulated soft-computing system which integral genetic algorithm and fuzzy logic.

2.0 Review of Related Literature

The reviewed of certain literatures provided fundamental basic for our research work. This research work includes:

Sleiman et al., (2010) reviewed ten programming language: C++, C#, Java, Groovy, JavaScript, PHP, Schalar, Scheme, Haskell and AspectJ with the aim of determining which programming language is more suitable for a particular language domain; Web applications development, OO based abstraction. At the end, we will give our conclusion that which languages are suitable and which are not for using in some language domain. The work also provides evidence and analysis on why some languages are better than other or have advantages over the other on some criterion.

Fajuyigbe et al**.,** (2009) proposed an appropriate language for introducing object oriented programming in tertiary institutions. The aim of their works was equipping students with this skill using the most appropriate language. Adopting the enhanced Alexander's scheme, they identified and defined a set of criteria critical to the selection of the most appropriate programming language for introducing students to Object-Oriented concepts and programming. Furthermore, based on these criteria, they produced a rating showing the appropriateness of selected programming languages to the teaching and learning of OOP. The rating result showed the suitability of these languages to the teaching and learning of OOP. The programming languages selected were the programming languages rated among the IT skills required in a tough job market: Java, C#, C++, C and Visual BASIC.

Observing carefully this research works provided a basic for exploring the reliability of programming language utilizing a simulated model.

3.0 Materials

Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth - truth values between "completely true" and "completely false" (Kasabov, 1998; Robert, 2000; Christos and Dimitros, 2008). The theory of fuzzy logic provides a mathematical strength to capture the uncertainties associated with human cognitive processes, such as thinking and reasoning. A fuzzy set A is called trapezoidal fuzzy number (Figure 1) with tolerance interval [a, b], left width α and right width β if its membership function has the following form

$$A(t) = \begin{cases} 1-(a-t)/\alpha & \text{if } a-\alpha \le t \le a \\ 1 & \text{if } a \le t \le b \\ 1-(t-b)/\beta & \text{if } a \le t \le b+\beta \\ 0 & \text{otherwise} \end{cases}$$

and we use the notation A = (a, b, α, β). It can easily be shown that $[A]^{\gamma}$ = [a − (1 − γ) α, b + (1 − γ) β], ∀ γ ε [0, 1].

The support of A is (a − α, b + β).

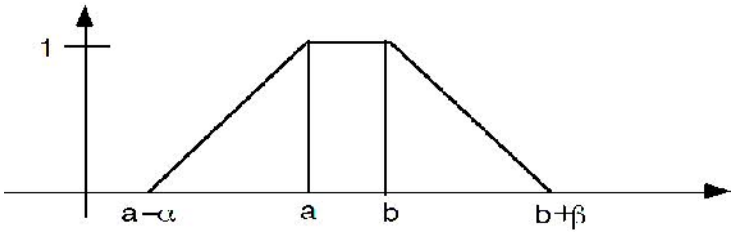

Figure 1: Trapezoidal fuzzy number (Leondes, 2010)

Fuzzy systems often learn their rules from experts. When no expert gives the rules, adaptive fuzzy systems learns by observing how people regulate real systems (Leondes, 2010). The difference between classical and fuzzy logic is something called "the law of excluded middle" (Bart and Satoru, 1993 and Ahmad, 2011). In standard set theory, an object does or does not belong to a set. There is no middle ground. In such bivalent systems, an object cannot belong to both its set and its compliment set or to neither of them. This principle preserves the structure of the logic and avoids the contradiction of object that both is and is not a thing at the same time (Zadeh, 1965). However, fuzzy logic is highly abstract and employs heuristic (experiment) requiring human experts to discover rules about data relationship (Angel and Rocio, 2011).

Fuzzy classification assumes the boundary between two neighboring classes as a continuous, overlapping area within which an object has partial membership in each class (Kuang; Ting-Hua and Ting-Cheng, 2011). It not only reflects the reality of many applications in which categories have fuzzy boundaries, but also provides a simple representation of the potentially complex partition of the feature space. (Sun and Jang**,** 1993 and Ahmad, 2011) propose an adaptive-network-based fuzzy classifier to solve fuzzy classification problems. Conventional approaches of pattern classification involve clustering training samples and associating clusters to given categories. The complexity and limitations of previous mechanisms are largely due to the lacking of an effective way of defining the boundaries among clusters. This problem becomes more intractable when the number of features used for classification increases (Robert, 2000; Kasabov, 1998; Rudolf; 2008, Christos and Dimitros, 2008). A fuzzy set has been derived as an extension of the classical notion of a set. According to classical set theory, an element either belongs or does not belong to the set. In contrast, fuzzy set theory permits the gradual assessment of the membership of elements in a set. A membership function can be described as a graph that defines how each point in the input space is mapped to the membership value between 0 and 1. Fuzzy Inference Systems are designed based on the concept of Fuzzy Logic. Fuzzy inference can be defined as the process of formulating the mapping from a given input to an output. A Fuzzy Logic System or a Fuzzy Inference System maps the crisp inputs into crisp outputs. It contains five

components: Rule-base, Data-base, Fuzzifier (transformation of crisp values into fuzzy set), Inference and Defuzzifier (transformation of fuzzy set into crisp value).

Professor John Holland in 1975 proposed an attractive class of computational models, called Genetic Algorithms (GA) that mimic the biological evolution process for solving problems in a wide domain. Genetic algorithm is made-up of five main components namely (Amit, 1999 and Zhang and Nguyen 1993).

a. **Fitness Function:** also called evaluation function, is the genetic algorithm component that rate a potential by calculating how good they are relative to the current problem domain.
b. **Selection:** is a way for the genetic algorithm to move toward promising regions in the search space. The individual with the highest fitness are selected and they will have a higher probability of survival to the next generation
c. **Mutation:** is a genetic operator that changes one or more gene values in a chromosome. The mutation process helps to overcome trapping at local maxima. The offspring's produced by the genetic manipulation process are the next population to be evaluated.
d. **Crossover:** Exchanging Chromosomes portions of genetic materials.
e. **Reproduction:** Birth of a new generation

Proposed Pseudo-code for Genetic Algorithm (Obi and Imianvan, 2013)

Input:

Size of population (k)
Rate of mutation (v)
Rate of crossover (c)
Number of iteration (n)
Number of crossover (nc)
Number of mutation (nm)

// Initialization
1. Randomly generate K possible solution;
2. Save solution in population Kok;

// Loop till terminal point
3. For m = 1 to n do;

// Crossover
4. Number of crossover nc = (k –nm)/2;
5. For u = 1 to n do;
6. Select two solutions randomly EA and FG for K;
7. generate GV and HN by two-point cross to EA and FG;
8. Save GV and HN to K2;
9. end for;

//Mutation
10. for u = 1 to n do;
11. Selection a solution Yh from K2;
12. Mutate each bit of Yh under nm and generate a new solution Yhi
13. If Yhi is impossibe
14. renovate Yhi with possible solution by modifying Yhi
15. end if
16. renovate Yh with Yhi in K2

17. end for

// renovate

18. renovate K = K2;
19. // return the best solution
20. Return the best solution Y in K

3.1 Factors for Assessing Computational Programming language Reliability

Several parameters have been utilized in determining programming language reliability but the under listed seven are the most appropriate (Robert, 2006).

a. Type Checking is simply testing for type errors in a given program, either by the compiler or during program execution. Type checking is an important factor in language reliability. Because run-time type checking is expensive, compile -time checking is more desirable. Furthermore, the earlier errors in program are detected, the less expensive it is to make the required repairs (Robert, 2006). It is also the process of ensuring that operands of an operator are of compatible types. A compatible type is one that either is legal for the operator or is allowed under languages rules to be implicitly converted by compile-generated codes (or the interpreter) to legal types. This automatic conversion is called coercion. Programming language reliability is close tied to type checking.

b. Exception Handling is the ability of a program to interrupt run-time error, as well as unusual condition detected by the program takes corrective measure before proceeding with program execution. . As long as exception handling exists in a programming language utilized for solving particular computational problem reliability is assured.

c. Restricting Aliasing: Aliasing is the process whereby multiple variables are assigned the same memory location or address. Aliasing is a threat to program reliability because it allows the value assigned to variables to be changed by been assigned to another variable. For example if both variables *sum* and *total* are aliases, any change to the value of total, changes the value of sum and vice versa. Two pointers are aliased when they point to the same memory location (Robert, 2006). Restricting aliasing will result in reliable programming language and reliable developed program.

d. Restricting Program-orthogonality: Program orthogonality simple means the freedom residing in the hand of the programmer which allows the programmer to combine language constructs in other to achieve desirable results. While it fosters program writability, it is a dangerous end for program reliability. Therefore preventing or restricting programming language orthogonlity will foster program reliability.

e. Program Uniformity: Program uniformity is tied to the consistency in the forms of language constructs. Uniformity has to do with format. Program uniformity will go a long way in determining program reliability.

f. Program Abstraction is the ability to define and use complicated structures or operations in a manner which permits many of its details to be ignored. Abstraction is a key concept in contemporary programming language design. The range of program

abstraction allowed by a programming language and the naturalness of its language expression enhances writ ability and reliability. Reliability is enhanced if the abstraction is moderately allowed.

g. **Program Expressivity** simply defines consistency of language features with natural way of thinking for solving a computational problem. Without program expressivity, reliability cannot be achieved.

4.0 Methodology Applied

In other to achieve our underlining objectives, the under listed criteria were applied in designing the proposed soft-computing model.

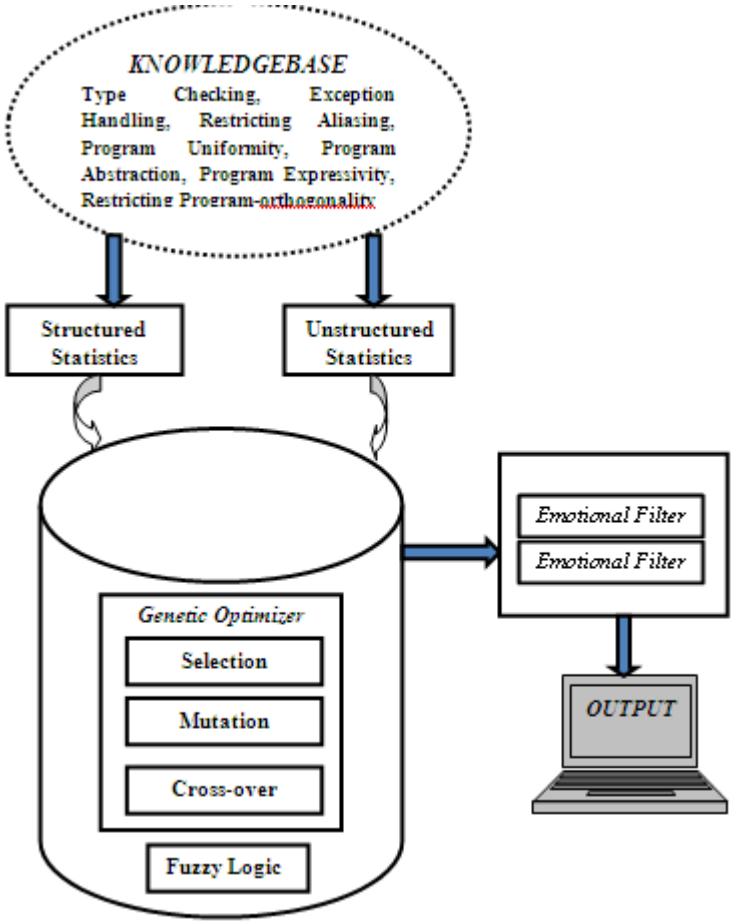

Fig.2: Soft-computing Model for the Assessment of programming Language Reliability

The model pictorially illustrated in fig.2, comprises of certain features, which includes

a. **Knowledgebase:** A varid form of database that hold special information. The criteria (parameters or fuzzyset) for determinimg programming language reliability is held in the knowdebase.

b. The structured and unstructures statistic simply refer to database information (usually well arranged) and dispered information residing in the knowledge-base respectively.
c. The **Genetic Optimizer** handles membership optimization following the procedure specified within the architecture.
d. **Fuzzy Logic:** the impreciseness (vagueness) associated with the parameters for assessing programming language reliability where handled utilizing fuzzy-if-then rules.
e. **Decision support:** predicate output result based on emotional filter and cognitive filter. The cognitive filter of the decision support engine takes as input the output report of knowledge base and applies objective rules to rank the programming languages criteria's. The emotional filter takes as input the output report of the cognitive filter and applies the subjective rules in the domain of studies in order to rank the programming language criteria's.
f. **The stationery system:** produces the output based on the paramters received.

5.0 Result and Discussions

The fuzzy partition for each input feature consists of the parameters for assessing programming language reliability. However, it can occur that if the fuzzy partition for assessing programming language reliability is not set up correctly, or if the number of linguistic terms for the input features is not large enough, then some patterns will be misclassified. The rules that can be generated from the initial fuzzy partitions of the classification for assessing programming language reliability are thus:

a. Not Reliable (Class: C_1)
b. Moderately Reliable (Class: C_2)
c. Reliable (Class: C_3)

If the assessed Programming Language (LP) possesses less than or equal to two ($LP \leq 2$) of the parameters for assessing program reliability *THEN* (C_1), If the assessed Programming Language (LP) possesses three ($LP = 3$) of the parameters for assessing program reliability *THEN* (C_2) and if the assessed Programming Language (LP) possesses four ($LP \geq 4$) or more parameters for assessing program reliability (*THEN* (C_3).

The Fuzzy IF-THEN Rules (R_i) for assessing program reliability:

R1: IF programming Language possesses *Type checking ability THEN* it is in class C_1.
R2: IF programming Language possesses *Type checking ability and Exception handling THEN* it is in class C_1.
R3: IF programming Language possesses *Type checking ability, Exception handling and Restricting Aliasing THEN* it is in class C_2.
R4: IF programming Language possesses *Type checking ability, Exception handling, Restricting Aliasing and Program uniformity THEN* it is in class C_3.
R5: IF programming Language possesses *Type checking ability, Exception handling, Restricting Aliasing, program uniformity and program Abstraction THEN* it is in class C_3.
R6: IF programming Language possesses *Type checking ability, Exception handling, Restricting Aliasing, Program uniformity, Program abstraction and Program expressivity THEN* it is in class C_3

R7: IF programming Language possesses *Type checking ability, Exception handling, Restricting Aliasing, Program uniformity, Program abstraction, Program expressivity and Restricting program orthogonality THEN* it is in class $C_{3.}$

A typical data set that contains the seven parameters is presented in Table 1. This shows the degree of intensity (membership) for assessing programming language reliability.

Table 1: Data Set showing the Degree of membership for Programming Language reliability

Parameters or Fuzzy sets For Programming Language Reliability	**Codes**	**Membership Function for Programming Language Reliability**		
		Cluster 1 (C_1)	**Cluster 2 (C_2)**	**Cluster 3 (C_3)**
Type Checking	R01	0.50	0.15	0.35
Exception Handling	R02	0.20	0.20	0.60
Restricting Aliasing	R03	0.10	0.80	0.10
Program Uniformity	R04	0.20	0.10	0.70
Program Abstraction	R05	0.30	0.60	0.10
Program Expressivity	R06	0.05	0.05	0.90
Restricting Program Othogonality	R07	0.00	0.50	0.50

Utilizing the algorithm proposed by obi and Imianvan (2013), the optimization of our membership was achieved..

Genetic Algorithm Inference:

R1: IF R01 THEN C1 = 0.50
R2: IF R01 AND R02 THEN C2 = 0.18
R3: IF R01, R02 AND R03 THEN C2 = 0.38
R4: IF R01, R02, R03 AND R04 THEN C3 = 0.44
R5: IF R01, R02, R03, R04 AND R05 THEN C3 = 0.37
R6: IF R01, R02, R03, R04, R05 AND R06 THEN C3 = 0.46
R7: IF R01, R02, R03, R04, R05, R06 AND RO7 THEN C3 = 0.46

We then convert these resolved values into whole numbers and imply them to be the fitness function (*f*) of the initial generation (Parents) as shown in Table 2.

R1: 50, **R2:** 18, **R3:** 38 **R4:** 44 **R5:** 37 **R6:** 46 **R7:** 46

Table 2: 1st and 2nd Generation Table

S/N	Selection	Chromosomes (Binary; 0 or 1)			Fitness function
		Parent (1st Gen)	Crossover	Parent (2nd Gen)	
1	50	110010	1 & 6	**110**101	**53**
2	46	101110	2 & 4	**101**100	**44**
3	46	101110	Mutation	1011**0**0	**44**
4	44	101100	2 & 4	**101**110	**46**
5	38	100110	5 & 7	**100**010	**34**
6	37	100101	1 & 6	**100**010	**34**
7	18	010010	5 & 7	**010**110	**22**

To create our 2nd and 3rd generation from the parents (1st generation) we chose the third bit from the left to be our crossover point. In the 4th generation each bold bit signifies the cross-over bits, a single bold bit signifies mutation of that bit and an italicized chromosomes signifies elitism. See Tables 3 and 4.

Table 3: 2nd and 3rdGeneration Table

S/N	Selection	Chromosomes (Binary; 0 or 1)			Fitness function
		Parent (2nd Gen)	Crossover	Parent (3rd Gen)	
1	53	**110**101	1 & 3	110**100**	**52**
2	46	**101**110	2 & 6	101**010**	**42**
3	44	**101**100	1 & 3	101**101**	**45**
4	44	**101**100	4 & 5	101**010**	**42**
5	34	**100**010	4 & 5	100**100**	**36**
6	34	**100**010	2 & 6	100**110**	**38**
7	22	**010**110	Mutation	0101**0**0	**20**

Table 4: 3rdand 4th Generation Table

S/N	Selection	Chromosomes (Binary; 0 or 1)			Fitness function
		Parent (3rd Gen)	Crossover	Parent (4th Gen)	
1	52	110100	Mutation	11011**0**	**54**
2	45	101101	2 & 3	1011**10**	**46**
3	42	101010	2 & 3	1010**01**	**41**
4	42	101010	6 & 4	1010**00**	**40**
5	38	100110	5 & 7	1001**00**	**40**
6	36	100100	6 & 4	1001**10**	**38**
7	20	010100	5 & 7	0101**10**	**22**

The best fourth generation (stopping criterion) is that with the best fitness function, 54. This implies that the clusters of the various parameters has been searched and optimized to 0.54, serving as the boundary in determining high and low degree of intensity of the served membership function. Therefore any parameter(s) with membership function (MF > 0.50) implies

high degree of membership and any parameters(s) with membership function (MF < 0.50) implies low degree of membership.

Based on the optimized membership function, the result in table 5 is obtained.

Table 5: Data Set showing the Degree of membership for Programming Language reliability

Parameters or Fuzzy sets For Programming Language Reliability	**Codes**	**Membership Function for Programming Language Reliability**		
		Cluster 1 (C_1)	**Cluster 2 (C_2)**	**Cluster 3 (C_3)**
Type Checking	R01	0.50	0.15	0.35
Exception Handling	R02	0.20	0.20	0.60
Restricting Aliasing	R03	0.10	0.80	0.10
Program Uniformity	R04	0.20	0.10	0.70
Program Abstraction	R05	0.30	0.60	0.10
Program Expressivity	R06	0.05	0.05	0.90
Restricting Program Othogonality	R07	0.00	0.50	0.50
Result		**Not Reliable**	**Moderately Reliable**	**Reliable**

6.0 Conclusion

The ability in determining if certain programming language constructs fosters programming reliability is essential for the survival, utilization and continuous propagation of such programming language. The needs to design a system that would assist application programmers' in general ascertain programming language reliability parameters cannot be over emphasized. The practicality of genetic algorithm and fuzzy logic (Genetic-fuzzy system) has been demonstrated by our proposed model. This model which uses a set of fuzzified data set incorporated into genetic algorithm system is more precise than the traditional system for recognizing reliable program language. The system designed is an interactive system that explores programming language limitation and gains. A system of this nature should be introduced in the IT Sector to ease the job of IT professionals, programmers and application developer in recognizing and filtering obsolete programming language.

Reference

[1] Ahmad H. (2011), "Fuzzy approach to Likert Spectrum in Classified levels in surveying researches" retrieved http://www.tjmcs.com.

[2] Angel C. and Rocio R. (2011), "Documentation management with Ant colony Optimization Metaheuristic: A Fuzzy Text Clustering Approach Using Pheromone trails" retrieved from soft computing in Industrial applications, Advances in intelligent and soft Computing, vol. 96, 2011, 261-70, DOI: 10.1007/978-3-642-20505-1_23.

[3] Bart K. and Satoru I. (1993), "Fuzzy Logic", retrieved from http//:Fortunecity .com/emachines/e11/86/fuzzylog.html.

[4] Christos S. And Dimitros S. (2008) "Neural Network", retrieved from http://www.docstoc.com/docs/15050/neural-networks.
[5] CISA: Certified Information System Auditor (2011), "Cisa review manual 2010", Chapter 5, 305.
[6] Fajuyigbe O., Onibere E., Ekuobase G. (2011), "Appropriate language for introducing object oriented programming in tertiary institutions" retrieved online from www.ajol.info/index.php/ijonas/article/view/49925
[7] Jiantao P. (1999), "Software Reliability", retrieved online from http://ece.cmu.edu/koopman/Des_a 99/sw-reliability/
[8] Kasabov N. K. (1998), "Foundations of neural networks, fuzzy systems, and knowledge engineering", A Bradford Book, The MIT Press Cambridge, Massachusetts London, England, ISBN 0-262-11212-.
[9] Kuang Y. H., Ting-H. C. and Ting-Cheng Chang (2011), "Determination of the threshold value β of variable precision rough set by fuzzy algorithms" retrieved from http://www.sciencedirect.com/science/article/pii/S0888613X11000831.
[10] Lamb J.D. (2006), "Insertion Heuristics for Central Cycle Problems" retrieved from aura.abdn.ac.uk/bitstream/2164/96/1/ISSN%200143-06-07.pdf
[11] Leondes C (2010), "The Technology of Fuzzy Logic Algorithm retrieved from Suite101.com/examples-of-expert-System-application-in-artificial Intelligence.
[12] Robert Full´er,(2000) Introduction to Neuro-Fuzzy Systems, Advances in Soft Computing Series, Springer-Verlag, Berlin/Heildelberg, 289 pages. (ISBN3-7908-1256-0)(MR1760972.
[13] Robert W. S. (2006), "Concept of Programming Language" 7ed, University of Colorado at Colorado spring, USA.
[14] Rudolf K. (2008), Neuro-fuzzy systems, retrieved from http://ask.com
[15] Sleiman R., Jiang Li., Mingzhi L., Yuanwei L. (2010), "Comparative Studies of 10 Programming Languages within 10 Diverse Criteria" retrieved from" http://arxiv.org/abs/1009.0305
[16] Sun C.T. and Jang J.S. (1993) "A neuro-fuzzy classifier and its applications", in: Proc. IEEE Int. Conference on Neural Networks, San Francisco, pp.94–98.
[17] Zadeh L.A. (1965), "Fuzzy sets. Information and Control", Vol.8, pp.338-353.