Using software metrics and evolutionary decision trees for software quality control

Peter Kokol, Vili Podgorelec, Maurizio Pighin

Abstract

Reliability is one of the most important aspects of software systems of any kind (embedded systems, information systems, intelligent systems, etc.) The size and complexity of software is growing dramatically during last decades and especially during last few years. Various methods can be used to achieve the software reliability i.e. software reliability engineering, fault tolerance, testing strategies, fault tree techniques, simulation, machine learning and software metrics. We have used several software complexity metrics together with a fractal software metric as the attributes for learning evolutionary decision trees. In this way one should be able to estimate whether a software module is dangerous regarding the probability of containing software faults. The aim of this paper is to present the α metric together with decision trees as a fault predictive approach used to foresee dangerous software modules, which identification can largely enhance the reliability of software.

1. Introduction

Reliability is one of the most important aspects of software systems of any kind (embedded systems, information systems, intelligent systems, etc.) The size and complexity of software is growing dramatically during last decades and especially during last few years. The systems consisting of hundred of millions lines of code are not a rarity any more, and the demand for highly complex hardware/software systems is increasing more rapidly than the ability to design, implement and maintain them. When the requirements for and dependencies of computers increase, the possibility of crises from failures also increases. The impact of these failures ranges from inconvenience to economic damages to loss of lives - therefore it is clear that software reliability is becoming a major concern not only for software engineers and computer scientists, but also for the “society” as a whole.

Various methods can be used to achieve the software reliability i.e. software reliability engineering (SRE), fault tolerance, testing strategies, fault tree techniques, simulation, machine learning [19] and software metrics.

1.1. Software metrics and reliability

Software development is a complex and complicated process in which software faults are inserted into the code by mistakes during the development process or maintenance. It has been shown that the pattern of the faults insertion phenomena is related to measurable attributes of the software, especially with the software metrics. For example, a large software system consists of various modules and each of these modules can be characterized in terms of attribute measures – it would be quite useful to be able to construct “dangerous module” prediction models based on these measurable attributes.

This project was partially supported by the grants: Ministry of Science of Slovenia – Project INCOMPETENT J2-0514-0796-98; Italian-Slovenian bilateral project "New Software Metrics for Real World Applications"; USA-Slovenian bilateral project "Physical Based Software Complexity Metrics".

1.2. Weaknesses of traditional software metrics

Although the software metrics are becoming more and more recognized in software engineering, many weaknesses remain. Some of them concerning software design are:

- The software is written in different programming languages ranging from "old fashioned third generation languages" FORTRAN, PASCAL or COBOL, trough fourth generation languages, spreadsheets and data base systems to fifth generation languages like PROLOG, LISP, etc. Still worse for some systems the source files have been lost, only the machine/executable code did remain. But the conventional software metrics are language dependent. We have to use (or even design) a different metric or at least a different tool for each programming language, again another metric for a program written in an assembler language and worse using traditional software metrics we are not able to asses the complexity of object or executable code.
- Software can be represented in many forms: requirements, specification, documentation, help files, requirements, user interfaces, etc. and all that representations can be manifested in very different appearances: written text, graphical, symbolic, formal languages, etc. To be able to assess the complexity of such a system in a holistic manner it is not enough to measure just the source code of the software, all other forms and artifacts have to be assessed – unfortunately no such metric does exist.
- To be able to track the progress of the software development process we have to measure the software during various stages like requirements engineering, architectural design, implementation and maintenance using a single metric – again no such metrics does exist.
- Normally software designers aren't metric's experts and while the output of a traditional complexity metric is a number, usually without any "physical" meaning and unit it is hard for them to interpret the assessed results. Is the number large or small? What is the unit? What does the measured number imply about complexity, reliability, effort already spent, effort needed for maintenance, quality, etc.?

1.3. Software metrics and complexity

The majority of experts in the computing field agree that complexity is one of the most relevant characteristics of computer software. For example Brooks states that computer software is the most complex entity among human made artifacts [20]. There are three possible, not completely distinct, viewpoints about software complexity:

- the classical computational complexity [22, 23],
- traditional software metrics, and
- recent "science of complexity" [21, 2, 7, 9].

Recently two attempts have been made to use the science of complexity in the measurement area. In her keynote speech at FESMA conference 1998 Kitchenham [1] argued that software measurement or estimates and predictions or assumptions based on them are infeasible. This is because software development, like in management science and economics, is a complex, non-linear adaptive system and inaccuracy of predictions is inherently emergent in such systems. As a consequence it is important to understand uncertainty and risk associated with that. Kokol et al [5-10] in their papers represent the similar view but their conclusion is that one not only has to understand the principles and findings of science of complexity but can use them to assess and measure the software more successfully, for example with the employment of so called α metric. This metric is based on the measurement of information content and entropy and it is known that the entropy is related to reliability – and thereafter α metric is a very viable candidate for software reliability assessment and software fault prediction.

1.4. The Aim and Scope of the Paper

The aim of this paper is to present the α metric and the employment of α together with decision trees as a *fault predictive metric* used to foresee dangerous software modules. Identified modules can be re-designed or tested and maintained more cautiously and any other special care can be devoted to these modules. In such manner the reliability can be largely enhanced.

In the second chapter some weaknesses of traditional software complexity metrics are enumerated. Next a physical background of α metric and some results of its use are presented. A study based on the analysis of 217 modules of a software system is presented in the last part of the paper together with concluding remarks. It is showed how α metric can be used together with other software complexity measures to induct a decision tree for the identification of potentially dangerous software modules. Decision trees are built with evolutionary algorithms, what increases their prediction power; the brief description of evolutionary approach in inducting decision trees is also provided.

2. Physical background

Both computer programs and natural language texts (human writings in the following text) can be in general classified as means for communicating. Communication can be defined as [14] *the interaction between systems or parts of a system using a prearranged **code,*** and the human communication as [15] *the organized behavior among a human group, according to a shared cultural and semantic **code***. It is clear from both of the above definitions that the **code** is crucial for effective communication, however it is not always so obvious that the code is in general complex, offering physical, perceptive, syntactic, semantic, pragmatic and cultural aspects in accordance with the characteristics of the environment shared by the communication systems. In general we can argue that the code is embodied in the communication language (or simply the language in the text that follows). There are many more or less related definitions of language [15], but an interesting view has been pointed out by a Slavic linguist Jakobson, building on the theories of de Saussure [13]. He argued that all languages share certain structures and rules that allow some combinations and forbid others. Without such structures and rules the words have no inherent meaning (i.e. “oo” occurs in “soothe” and “cool”).

Another aspect necessarily associated with human writings and communication is the concept of meaning [16]. Like for the definition of language there are many different definitions of meaning, but we are only interested in the relation meaning – information in this paper. In that context [15] **meaning** is mainly subservient to implicit knowledge i.e. formerly acquired and organized **information.**

Many different quantities [3] have been proposed as measures of complexity to capture all our intuitive ideas about what is meant by complexity. Some of the quantities are computational complexity, information content, algorithmic information content, the length of a concise description of a set of the entity’s regularities, logical depth, etc., (in contemplating various phenomena we frequently have to distinguish between effective complexity and logical depth - for example some very complex behavior patterns can be generated from very simple formula like Mandelbrot’s fractal set, energy levels of atomic nuclei, the unified quantum theory, etc.- that means that they have little effective complexity and great logical depth). Li [17] relates the complexity with difficulty concerning the system in question, for example the difficulty of constructing a system, difficulty of describing the system, etc. It is also well known that complexity is related to entropy. Several authors speculate that the relation is one to one (i.e. algorithmic complexity is equivalent to entropy as a measure of randomness) [12] but [17] shows that the relation is one to many or many to one depending on the definition of the complexity and the choice of the system being studied.

Using the assumption that meaning and information content in text is founded on the correlation between language symbols one of the meaningful measures of complexity of human writings is entropy as established by Shannon [11]. Yet, when a text is very long it is almost impossible to calculate the Shannon information entropy so Grassberger [12] proposed an approximate method to estimate entropy. But entropy does not reveal directly the correlation properties of texts so another more general measure is needed. One possibility is to use Fourier power spectrum, however a method yielding much more quality scaling data was introduced recently. This method, called ***Long-range correlation*** [4] is based on the generalization of entropy and is very appropriate for measuring complexity of human writings.

3. Long-range correlations

Various quantities for the calculation of long range correlation in linear symbolic sequences were introduced in the literature and are discussed by Ebeling [18]. The most popular methods are dynamic entropy, scaling exponent $1/f^{\delta}$, higher order cumulates, mutual information, correlation functions, mean square deviations, and mapping of the sequence into random walk. It is agreed by many authors [18, 4] that the mapping into random walk is the most effective and successful approach in the analysis of human writings.

Long-range power law correlation (LRC) has been discovered in a wide variety of systems. As a consequence the LRC is very important for understanding the system's behavior, since we can quantify it with a critical exponent. Quantification of this kind of scaling behavior for apparently unrelated systems allows us to recognize similarities between different systems, leading to underlying unification. For example LRC has been identified in DNA sequences and natural language texts [4, 18] - the consequence is that DNA and human writings can be analyzed using very similar techniques.

3.1. Calculation of the long-range power law correlation

In order to analyze the long-range correlation of a string of symbols the best way is to first map the string into a Brownian walk model [4]. Namely, the Brownian walk model is well researched and publicized and the derived theories and methodologies are widely agreed and in addition easy to implement with the use of computer. There are various possibilities to implement the above mapping [6]. In this paper we will use the so-called CHAR method described by Schenkel [4] and Kokol [6]. A character is taken to be the basic symbol of a human writing (Figure 3). Each character is then transformed into a six bit long binary representation according to a fixed **code table**. It has been shown by Schenkel that the selection of the code table does not influence the results as long as all possible codes (i.e. we have 64 different codes for the six bit representation – in our case we assigned 56 codes for the letters and the remaining codes for special symbols like period, comma, mathematical operators, etc) are used. The obtained binary string is then transformed into a two dimensional Brownian walk model (Brownian walk in the text which follows) using each bit as a one move - the 0 as a step down and the 1 as a step up.

An important statistical quantity characterizing any walk is the root of mean square fluctuation F about the average of the displacement. In a two-dimensional Brownian walk model the F is defined as:

$$F^2(l) \equiv \overline{[\Delta y(l, l_0)]^2} - \left[\overline{\Delta y(l, l_0)}\right]^2$$

where

$$\Delta y(l, l_0) \equiv y(l_0 + l) - y(l_0)$$

l is the distance between two points of the walk on the X axis
l_0 is the initial position (beginning point) on the X axis where the calculation of $F(l)$ for one pass starts
y is the position of the walk – the distance between the initial position and the current position on Y axis

and the bars indicate the average over all positions l_0.

The $F(l)$ can distinguish two possible types of behavior:

- if the string sequence is uncorrelated (normal random walk) or there are local correlations extending up to a characteristic range i.e Markov chains or symbolic sequences generated by regular grammars [17], then

$$F(l) \approx l^{0.5}$$

- if there is no characteristic length and the correlations are "infinite" then the scaling property of $F(l)$ is described by a power law

$$F(l) \approx l^{\alpha} \text{and } \alpha \neq 0.5.$$

The power law is most easily recognized if we plot $F(l)$ and l on a double logarithmic scale (Figure 3). If a power law describes the scaling property then the resulting curve is linear and the slope of the curve represents α. In the case that there are long range correlation in the strings analyzed, α should not be equal to 0.5.

The main difference between random sequences and human writings is purpose. Namely, the writing or programming is done consciously and with purpose that is not the case with random processes, thereafter we anticipate that α should differ from 0.5. The difference in α between different writings can be attributed to various factors like personal preferences, used standards, language, type of the text or the problem being solved, type of the organization in which the writer (or programmer) works, different syntactic, semantic, pragmatic rules etc.

4. Evolutionary decision trees

Inductive inference is the process of moving from concrete examples to general models, where the goal is to learn how to classify objects by analyzing a set of instances (already solved cases) whose classes are known. Instances are typically represented as attribute-value vectors. Learning input consists of a set of such vectors, each belonging to a known class, and the output consists of a mapping from attribute values to classes. This mapping should accurately classify both the given instances and other unseen instances.

A decision tree [26] is a formalism for expressing such mappings and consists of tests or attribute nodes linked to two or more sub-trees and leafs or decision nodes labeled with a class which means the decision (figure 1). A test node computes some outcome based on the attribute values of an instance, where each possible outcome is associated with one of the sub-trees. An instance is classified by starting at the root node of the tree. If this node is a test, the outcome for the instance is determined and the process continues using the appropriate sub-tree. When a leaf is eventually encountered, its label gives the predicted class of the instance.

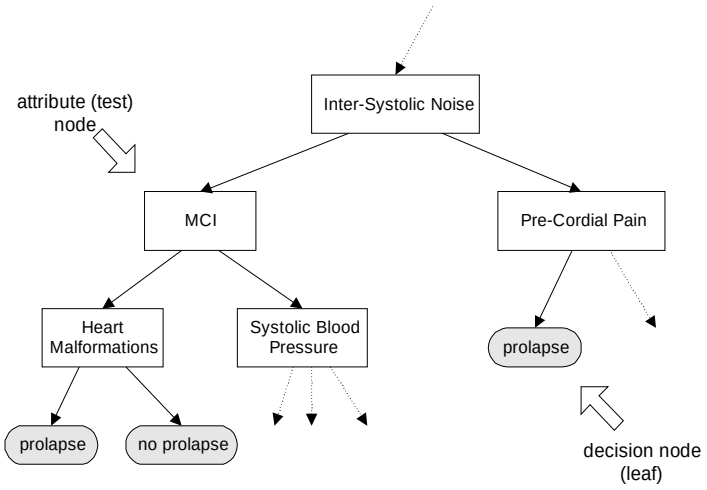

Figure 1. An example of a decision tree.

Evolutionary algorithms are adaptive heuristic search methods which may be used to solve all kinds of complex search and optimization problems. They are based on the evolutionary ideas of natural selection and genetic processes of biological organisms. As the natural populations evolve according to the principles of natural selection and "survival of the fittest", first laid down by Charles Darwin, so by simulating this process, evolutionary algorithms are able to evolve solutions to real-world problems, if they have been suitably encoded. They are often capable of finding optimal solutions even in the most complex of search spaces or at least they offer significant benefits over other search and optimization techniques.

As the traditional decision trees' induction methods contain several disadvantages we decided to use the power of evolutionary algorithms to induct the decision trees. In this manner we developed the evolutionary decision support model that evolves decision trees in a multi-population genetic algorithm SAEDT: self-adapting evolutionary decision trees [25]. Many experiments have shown the advantages of such approach over the traditional heuristic approach for building decision trees, which include better generalization, higher accuracy, possibility of more than one solution, efficient approach to missing and noisy data, etc.

4.1. SAEDT algorithm

When defining the internal representation of individuals within the population, together with the appropriate genetic operators that will work upon the population, it is important to assure the feasibility of all solutions during the whole evolution process. Therefore we decided to present individuals directly as decision trees. This approach has some important features: all intermediate solutions are feasible, no information is lost because of conversion between internal representation and the decision tree, the fitness function can be straightforward, etc. The problem with direct coding of solution may bring some problems in defining of genetic operators. As decision trees may be seen as a kind of simple computer programs (with attribute nodes being conditional clauses and decision nodes being assignments) we decided to define genetic operators similar to those used in genetic programming where individuals are computer program trees [27].

For the selection purposes a slightly modified linear ranking selection was used. The ranking of an individual decision tree within a population is based on the local fitness function:

$$LFF = \sum_{i=1}^{K} w_i \cdot (1 - acc_i) + \sum_{i=1}^{N} c_i \cdot attr(t_i) + w_u \cdot nu$$

where K is the number of decision classes, N in the number of attribute nodes in a tree, acc_i is the accuracy of classification of objects of a specific decision class d_i, w_i is the importance weight for classifying the objects of a decision class d_i, $attr(t_i)$ is the attribute used for a test in a node t_i, c_i is the cost of using the attribute of $attr(t_i)$, nu is number of unused decision (leaf) nodes, i.e. where no object from the training set fall into, and w_u is the weight of the presence of unused decision nodes in a tree.

According to this local fitness function the best trees (the most fit ones) have the lowest function values – the aim of the evolutionary process is to minimise the value of LFF for the best tree. A near optimal decision tree would classified all training objects with accuracy in accordance with importance weights w_i (some decision classes may be more important than the others), would have very little unused decision nodes (there is no evaluation possible for this kind of decision nodes regarding the training set) and would consist of low-cost attribute nodes (in this manner the desirable/undesirable attributes can be prioritised).

Crossover works on two selected individuals as an exchange of two randomly selected sub-trees. In this manner a randomly selected training object is used to determine paths (by finding a decision through the tree) in both selected trees. Then an attribute node is randomly selected on a path in the first tree and an attribute is randomly selected on a path in the second tree. Finally, the sub-tree from a selected attribute node in the first tree is replaced with the sub-tree from a selected attribute node in the second tree and in this manner an offspring is created which is put into a new population.

Mutation consists of several parts: 1) one randomly selected attribute node is replaced with an attribute, randomly chosen from the set of all attributes; 2) a test in a randomly selected attribute node is changed, i.e. the split constant is mutated; 3) a randomly selected decision (leaf) node is replaced by an attribute node; 4) a randomly selected attribute node is replaced by a decision node.

With the combination of presented crossover, which works as a constructive operator towards local optimums, and mutation, which works as a destructive operator in order to keep the needed genetic diversity, the searching for the solution tends to be directed toward the global optimal solution, which is the most appropriate decision tree regarding our specific needs (expressed in the form of LFF). As the evolution repeats, more qualitative solutions are obtained regarding the chosen fitness function. The evolution stops when an optimal or at least an acceptable solution is found or if the fitness score of the best individual does not change for a predefined number of generations.

4.1.1. Self-adaptation by information spreading in multi-population model

The multi-population model consists of three independent populations, one main population and two competing ones. First population is a single decision tree, initially induced in traditional way with the C4.5 algorithm [26], and the decision trees in the other two populations evolve in accordance with the evolutionary algorithm, described in the previous section. The best individual from the main population, which serves as the general solution, is competing with the best individuals from other populations in solving a given problem upon a training set. When it becomes dominant over the others (in the sense of the accuracy, sensitivity and specificity of the classification of training objects) it spreads its "knowledge" to them. In this way the other solutions become equally successful and the main population has to improve further to reclaim its dominance. This can be done by adjusting the local fitness function – for this purpose the weights used for the LFF is modified:

$$\forall i, i \in [1..K]: w_i = w_i + (1 - acc_i) * rand(av)$$

where av is the adaptation value and $rand(av)$ is random value from [0,av].

One global fitness function (different from the local ones – those used in the evolutionary process to evaluate the fitness of individual decision trees) is used to determine the dominance of one population over another. It is predetermined as a weighted sum of accuracy, sensitivity and specificity (look at the results in the next section) of the given solution, together with the cost of chosen attributes:

$$GFF = w_{acc} \cdot (1 - acc) + w_{sens} \cdot (1 - sens) + w_{spec} \cdot (1 - spec) + \sum_{i=1}^{N} c_i \cdot type(t_i) + w_u \cdot nu$$

where the last two parts are the same as in the local fitness function and *acc* is the accuracy of classified training objects, *sens* is the sensitivity of classified training objects, *spec* is the specificity of classified training objects, w_{acc} is the importance weight of accuracy, w_{sens} is the importance weight of sensitivity, and w_{spec} is the importance weight of specificity.

5. Fault prediction using decision trees and α metrics

To test the usability of decision trees and fractal metrics α in predicting potentially dangerous modules and in this manner designing reliable software we performed a study in a real world environment. We tested a software system consisting of 217 modules representing more than 200000 lines of code.

First the modules have been identified either as OK or DANGEROUS by applying the model developed by Pighin [24]. The purpose of this model is to use the software complexity metrics to define complexity risk thresholds. The modules are divided into two classes – bellow and above some fault threshold value (5 faults in our case).

A set of 168 attributes, containing various software complexity measures, have been determined for each software module and the α coefficient has been calculated for each software module. From all 217 modules 167 have been randomly selected for the learning set, and the remaining 50 modules has been selected for the testing set. Several decision trees have been induced for predicting dangerous modules. In the figure 2 an induced decision tree is presented as an example. The results of classification using induced decision tree is presented in tables I-III.

Table I. Classification results on learning set (number of software modules).

	Classified as OK	**Classified as DANGEROUS**
OK	108	11
DANGEROUS	4	44

Table II. Classification results on testing set (number of software modules).

	Classified as OK	**Classified as DANGEROUS**
OK	30	6
DANGEROUS	4	10

Table III. Results of induced decision tree for predicting dangerous software modules.

	learning set	**testing set**
accuracy	91.02	80.00
sensitivity	91.67	71.43
specificity	90.76	83.33

Table IV. Results using only α metric as a fault predictive method.

	testing set
accuracy	70.0
sensitivity	66.7
specificity	72.7

```
alpha
  |--[<0.63233] OK
  |--[>=0.63233] signif_di_commento
     |--[<1480.04000] StrCtrl_linee
     |  |--[<44.55000] OK
     |  |--[>=44.55000] DANGEROUS
     |--[>=1480.04000] istr_di_selezione
        |--[<3.79900] DANGEROUS
        |--[>=3.79900] funz_param_formali
           |--[<19.16200] DANGEROUS
           |--[>=19.16200] signif_di_commento
              |--[<3026.50800] OK
              |--[>=3026.50800] DANGEROUS
```

Figure 2. Evolutionary induced decision tree for predicting potentially dangerous software modules.

Above facts showed the usability of decision trees together with various software complexity measures and α metric in software fault prediction. The Table IV shows that the accuracy, when only α metric is used, is about 70% - that means that 70% of modules are correctly classified. When α value is used together with other complexity measures to build the decision tree, the accuracy is increased to 80%, which represents a big improvement, especially regarding the fact that both the sensitivity and the specificity increase.

One of the most interesting characteristic of the induced decision tree (figure 2) is that alpha has been chosen as the most important attribute (root node). This fact shows that α metric is actually useful in predicting dangerous software modules. The combination of α metric with some other software complexity measures as the attributes for decision trees proved to be very successful.

Another important fact is the size of the induced decision tree. As it is very small, regarding the number of possible attributes, it can be concluded that the induced decision tree is highly generalized and is not biased to over-fitting.

6. Conclusion

In the paper we present the use of evolutionary decision trees as a fault predictive approach. We show that the α metric as an attribute together with other software complexity measures can be successfully used to induce decision trees for predicting dangerous modules (modules having a lot of undetected faults). Redesigning such modules or devoting more testing or maintenance effort to them can largely enhance the quality and reliability, making the software much safer to use. In our future studies we would like to use more advanced text mining methods to even better assure the reliability of software.

References

[1] Kitchenham, B., “The certainty of uncertainty”, Proceedings of FESMA (Eds: Combes H et al), Technologish Institut, 1998, pp. 17-25.

[2] Morowitz, H., “The Emergence of Complexity”, Complexity 1(1), 1995, pp. 4.

[3] Gell-Mann, M., “What is complexity”, Complexity 1(1), 1995, pp. 16-19.

[4] Schenkel, A., Zhang, J., Zhang, Y., “Long range correlations in human writings”, Fractals 1(1), 1993, pp. 47-55.
[5] Kokol, P., Kokol, T., “Linguistic laws and computer programs”, Journal of the American Society for Information Science 47(10), 1996, pp. 781-785.
[6] Kokol, P., Brest, J., Žumer, V., “Long-range correlations in computer programs”, Cybernetics and systems 28(1), 1997, pp. 43-57.
[7] Kokol, P., Podgorelec, V., Brest, J., “A wishful complexity metric”, Proceedings of FESMA (Eds: Combes H et al), Technologish Institut, 1998, pp. 235 - 246.
[8] Kokol, P., “Analysing formal specification with alpha metrics”, ACM Software Engineering Notes, 24(1), 1999a, pp. 80-81.
[9] Kokol, P., Podgorelec, V., Zorman, M., Pighin, M., "Alpha - a generic software complexity metric", Project control for software quality : proceedings of ESCOM-SCOPE 99, Shaker Publishing BV, 1999b, pp. 397-405.
[10] Kokol, P., Podgorelec, V., Zorman, M., "Universality - a need for a new software metric", International workshop on software measurement, 1999c, pp. 51-54.
[11] Shannon, C.E., “Prediction and entropy of printed English”, Bell System Technical Journal, 1951, pp 30, 50.
[12] Grassberger, P., “Estimating the information content of symbol sequences and efficient codes”, IEEE Transactions on Information Technology, 1989, pp. 35, 669.
[13] Gibbs,W.W., “The Profile of Claude Levi-Strauss”, Scientific American 278(1), 1998, pp. 24-25.
[14] Young, J.Z., “Programs of the Brain”. Oxford U.P., Oxford, 1978.
[15] Francois, C. (ed.), “International encyclopaedia of systems and cybernetics”, K.G. Saur, Munchen, 1997.
[16] Parkinson, G.H.R. (ed.), “The Theory of Meaning”, Oxford U.P., Oxford, 1982.
[17] Li, W., “On the Relationship Between Complexity and Entropy for Markov Chains and Regular Languages”, Complex Systems 5(4), 1991, pp. 381 – 399.
[18] Ebeling, W., Neiman A., Poschel T., “Dynamic Entropies, Long-Range Correlations and Fluctations in Complex Linear Structures”, Proceedings of Coherent Approach to Fluctations, World Scientific, 1995.
[19] Lyu, M.R. (ed.), “Software Reliability Engineering”, Computer Society Press, 1995.
[20] Brooks, P.F., "No silver bullet: essence and accidents of software engineering", *IEEE Computer* 1987, 20(4)10-19, 1987.
[21] Pines, D. (Ed.), "Emerging syntheses in science", Addison Wesley, 1988.
[22] Cohen, B., Harwood, W.T., Jackson, M.I. "The specification of complex systems", Addison Wesley, 1986.
[23] Wegner, P., Israel, M. (Eds.), "Symposium on Computational Complexity and the Nature of Computer Science", Computing Surveys 27(1)5 - 62, 1995.
[24] Pighin, M., Kokol, P., "RPSM: A risk-predictive structural experimental metric", Proceedings of FESMA'99, Technologish Institut, 1999, pp. 459–464.
[25] Podgorelec, V., Kokol, P., "Self-adapting evolutionary decision support model", Proceedings of the 1999 IEEE International Symposium on Industrial Electronics ISIE'99, Bled, Slovenia, IEEE Press, 1999, pp. 1484-1489.
[26] Quinlan, J.R., "C4.5: Programs for Machine Learning", Morgan Kaufmann, 1993.
[27] Koza, J.R., "Genetic Programming: On the Programming of Computers by Natural Selection", MIT Press, 1992.