Genetic Algorithm based Refinement Methods for Security Metrics

Adrian VIŞOIU
Academy of Economic Studies, Bucharest, Romania
Economic Informatics Department
adrian.visoiu@csie.ase.ro

This paper presents two genetic algorithm based model refinement methods used for vulnerability estimation models. A method presents how model structure refinement is applied to obtain models that estimate the cumulative number of vulnerabilities for a certain product. In this case, empirical observation of similarities between consecutive versions of the product is taken into account. Model structure refinement is presented as procedure. The experimental results show how the method is applied and the results are discussed. The second method uses an aggregated performance indicator as selection criterion in the genetic algorithm. It is shown that simpler models are produced, keeping the quality of estimation comparable with more complex ones. Experimental results confirm the hypotheses.

Keywords*: model structure refinement, vulnerability estimation, genetic algorithms, validation, security metrics, aggregated performance*

1 Security Metrics

The ISO 9126 standard [1] defines security as an attribute of the functionality, related to the ability of a software product to prevent unauthorized access intentional or unintentional to programs and data.
An important aspect in assessing security is represented by assessing vulnerabilities as defined in [2]. Vulnerabilities represent an important fraction of the software flaws that need to be repaired. The study of vulnerability discovery and fixing process is necessary for a proper management where models are developed to support the decision making. The number of discovered vulnerabilities may be small in early states, may increase in maturity stages of the product and finally may decrease, as the market looses interest in using the product. Also, the number of discovered vulnerabilities may increase exponentially, or take any other evolution. For software users, the number of vulnerabilities is important because it decides the number of patches they have to apply to the product to keep it safe from attackers and this is a time consuming activity.
From the moment a software product is launched, it enters the maintenance phase. Efforts are made to implement adaptive and corrective maintenance including people and resources. Recording the number of vulnerabilities discovered for a certain software product helps estimation of future values and for comparison between products from the same developer or different developers. Public databases like [3], [4] and [5] keep records of identified vulnerabilities. Data sources may be extracted in order to analyze the evolution for a certain software product.

2 Refinement of Estimation Models

There are many factors that influence the number of found vulnerabilities in a software product, of which some important are: the size of the software, the number of implemented functions, the market share of the product, the time elapsed from the release of the product.
A model for cumulative number of vulnerabilities estimation must take into account those factors and other ones as well. The effort for data collection for such model is high. In order to simplify, another factor is considered that aggregates the influence of all other ones. This factor is time.
The number of vulnerabilities existing in a software product is estimated using models developed by specialists. Most of the models are adaptations from reliability estimation models. They are used for predicting the

cumulative number of vulnerabilities, at a certain moment in time t, denoted by $\Omega(t)$ and representing the primitive function of the vulnerability distribution function over time, starting with the moment of the product release date. Common model structures are described in [6] and [7]:
$\Omega(t) = a*\ln(b*t)$ (thermodynamic model)
$\Omega(t) = a/(b*e^{-ct}+1)$ (logistic model)
$\Omega(t) = a*t^2+b*t$ (linear model)
$\Omega(t) = a(1-e^{-bt})$ (exponential model).
For each analyzed software product, coefficients are estimated and computed values are compared with the real ones. Each model gives best results for certain software products.
The fitness function *FIT* used to assess the statistical performance of a model is the mean squared error.

In the context of model generation, refinement is a procedure that takes a model *M* of complexity *C* and transforms it to a model *M'* of complexity *C'*, such way that $C>C'$ and the fitness of *M'* doesn't differ substantially from the fitness of *M* as presented in [8]. The complexity indicator takes into account the number of operands and operators and the fitness is chosen among the existing statistical indicators that assess the quality of a model. A comparison between software metrics refinement techniques is included in [9].

3 Genetic Algorithm based Model Structure Refinement

When using model generators described in [10], the analyst must pay attention to the distribution of generated model structures in order to find patterns that indicate a model structure is more fit than others for the purpose of the research. Consider a list of model structures $S_1, S_2, \ldots, S_L$, used to estimate the levels of a certain dependent variable, according to a list of factors. Structure refinement is defined as a procedure that takes the initial list of structures $S_1, S_2, \ldots, S_L$ and retains a subset $S_{i1}, S_{i2}, \ldots, S_{iL'}$ where $L'<L$, and the subset $S_{i1}, S_{i2}, \ldots, S_{iL'}$ contains the best structures according to a performance criterion.

There are peculiarities that make the evolutionary algorithms fit for this approach as presented in [11]. Gene expression programming, introduced in [12], has a pseudorandom behavior when building the initial population; the genetic operators like selection, mutation, the exchange of genetic material are also applied randomly. When running the algorithm for several times, using the same dataset, it is observed that the number of generated structure types is small, the algorithm having a stronger preference for generating models from certain structures than from others.
Consider the models $M_1, M_2, \ldots, M_r$ obtained after r generation algorithm runs for a certain dataset. A number of n model structures $S_1, S_2, \ldots, S_n$ is obtained, having the relative apparition frequency $f_1, f_2, \ldots, f_n$, respectively, the number of runs being greater than the number of structures. Each structure corresponds to a model which was the best after evolving a certain population.
The list of model structures is sorted in descending order according to the frequency of apparition, obtaining the list $S_{k1}, S_{k2}, \ldots, S_{kn}$, where S_{k1} is the most frequent apparition and S_{kn} is the structure with the least frequent occurrence. A threshold h is defined and the first s structures are chosen with respect to the relation:

$$\sum_{i=1}^{s} f_{ki} < h.$$

The models $M_{k1}, M_{k2}, \ldots, M_{ks}$ are built, having the corresponding chosen structures $S_{k1}, S_{k2}, \ldots, S_{ks}$ and their coefficients are estimated using a least squares algorithm. For each model performance or fitness *FIT* is computed and the best models are chosen and they are subject to further validation of use.
The experimental results aim to show how model structure refinement helps in building or choosing a model that is fit to estimate the cumulative number of vulnerabilities in a software product, denoted by CUMULATIVE, when previous versions of the product are available.
Three versions of Apache HTTPD Server, 1.3, 2.0 and 2.2 are considered. For this open source project there is a public section where

all the security vulnerabilities are presented [11]. Data is collected and datasets are built.
For Apache 1.3 and 2.0 the evolution of the cumulative number of vulnerabilities is given in Figure 1. Data is available for over 3700 days since the initial release for Apache 1.3 and over 2000 days for Apache 2.0.

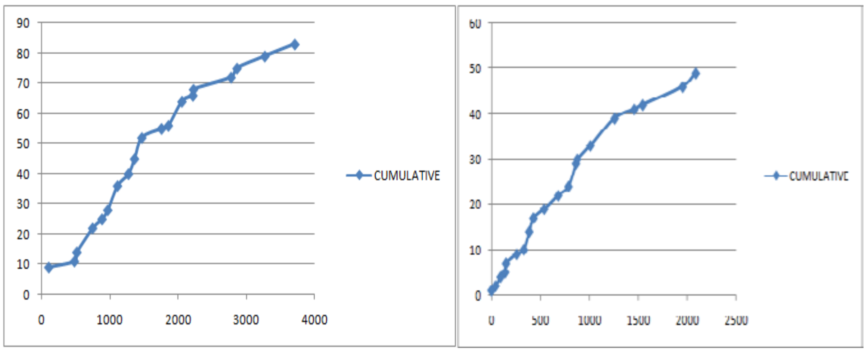


Fig. 1. Cumulative number of vulnerabilities discovered and fixed for Apache 1.3 and 2.0

For apache 2.2 the evolution of the cumulative number of vulnerabilities is given in Figure 2. The dataset is shorter, containing records for over 800 days.

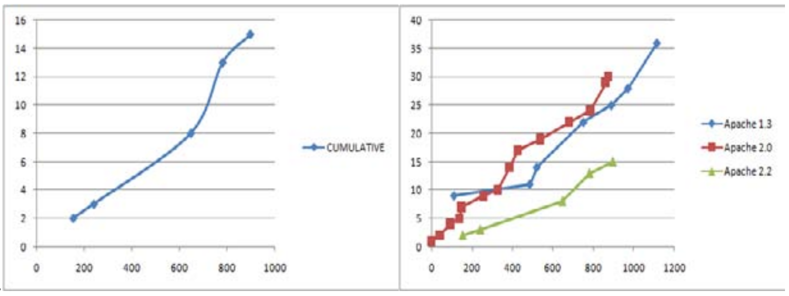


Fig. 2. The evolution of vulnerabilities for Apache 2.2 and the comparison between the three products for a similar period

The comparison between the evolutions of the process for each version for the first 1000 days is also presented in Figure 2.
The evolution of vulnerability discovery and fixing process shows similar evolution in a similar time interval. The correlation coefficient between interpolated values recorded for a certain version and all the others is greater than 0.9, which conducts to an empirical conclusion that the vulnerability discovery and fixing process follows a similar evolution for each version of the product.
The gene expression programming is used to build models to fit the recorded data. The generator is run a number of times for each dataset, such way the analyst observes a particular distribution of the generated model structures. The model generation process conducted to distributions of models structures for Apache 1.3 and Apache 2.0, as presented in Table 1.

Table 1. Model structure distributions

Apache 1.3	**Apache 2.0**
S1: $TIME^{a}$ 50%	*S1'*: $TIME^{A}$ 34.78%
S2: $A*TIME$ 4 33.33%	*S2'*: $A*TIME^{B}$ 26.08%
S3: $A*\ln(TIME)$ 16.66%	S3': $A*TIME$ 17.39%
	S4': $TIME^{A}+B$ 8.69%
	S5': $A*TIME^{B}+C$ 8.69%
	S6':$A*TIME+B$ 4.34%

After the parameter estimation, for Apache 1.3 and Apache 2.0, the models presented in Table 2 are obtained.

Table 2. Estimated models

Apache 1.3	Apache 2.0
M_{S1}: $CUMULATIVE(TIME)=TIME^{0.534}$ FIT=60.31	M_{S1}': $CUMULATIVE = TIME^{0.495}$ FIT=24.37
M_{S2}: $CUMULATIVE(TIME)=0.027*TIME$ FIT=51.93	M_{S2}': $CUMULATIVE = 0.14 * TIME^{0.772}$ FIT= 4.36
M_{S3}:$CUMULATIVE(TIME)=6.852*\ln(TIME)$ FIT=345.	M_{S3}': $CUMULATIVE = 0.027*TIME$ FIT = 13.28
	M_{S4}': $CUMULATIVE = TIME^{0.527}$-7.796 FIT=6.89
	M_{S5}': $CUMULATIVE = 0.264*TIME^{0.695}$-2.619 FIT=3.59
	M_{S6}': $CUMULATIVE = 0.025*TIME$+2.972 FIT=9.75

As seen, parameter values are very similar both for $TIME^A$ model structure type and $A*TIME$. In the case of the linear model, the slope is identical. Taking into account the similarity between the particular evolutions of the vulnerability fixing and discovery for all three products it is considered that S_1 and S_2 are fit structures to build models to make estimations also for Apache 2.2. The estimation of coefficients for Apache 2.2 gives:

M_1: $CUMULATIVE(TIME) = TIME^{0.359}$ FIT=11.06

M_2 $CUMULATIVE(TIME)=0.016*TIME$ FIT= 1.40

In [13] the most frequently generated models are statistically validated for their corresponding datasets. For Apache 2.2 the models are validated using Chi Squared X^2 test. For a risk of 5%, for the Apache 2.2 dataset, the threshold value of X^2 is 9.48. X^2 computed for M_1 is 15.77 which fall over the threshold, which means it is not statistically significant. The X^2 value computed for M_2 model is 1.05 which is a significantly smaller than the threshold and the model is validated.

The statistical validation shows that for the first period in the life of Apache 2.2 software product, the M_2 linear model is fit to estimate the number of vulnerabilities. The cumulative number of discovered and fixed vulnerabilities is increasing in rate with time. When looking at the graphs of the other versions it is observed that there is also a linear trend after the launch of the product. M_1 model must not be abandoned. It's performance must be assessed in practice, as there is a strong resemblance between the three products. M_1 model might be more fit for estimation in the second part of the product life, when the evolution of cumulative number of vulnerabilities starts to reduce its slope.

New model structure refinement must be performed each time new data is available, to reestimate model coefficients or to observe if there is a change in the model structure when the phenomenon changes its evolution.

4 Genetic Algorithm based Model Structure Refinement by Adapting Selection Criterion

Applying the genetic operators iteratively leads to selecting the best individual of a generation based on a performance criterion which takes into account the quality of the estimate on the data set considered. Also, the selection of individuals that give rise to new generations is all due to the same performance indicator; the individuals have a probability of selection proportional to the estimation quality of the corresponding model demonstrates.

A problem with the construction of analytical expressions using gene expression programming is obtaining expressions with an apparent high complexity, which is in contradiction with the objectives of refining. However, the gross model provided by the algorithm loses complexity, if, as expected, it is in a form they are made all operations between the original constants.

The criterion used in selecting models and individuals population of chromosomes is the mean squared error, MSE, which is a criterion of minimized. By their nature, genetic algorithms generate analytical expressions of high complexity.

A solution to reduce complexity of generated models is a convenient choice of algorithm specific parameters, if analyst recourses to one or more of the following options:

- restricting the list of variables
- restricting the list of operators, recommending the use of operators with small number of arguments
- limiting the size of a chromosome using chromosome with a single gene or small number of genes
- limiting maximum size of a gene by choosing a small value for the gene head parameter.

There are cases where small values chosen for the above parameters lead to adverse results relating to the identification of links, failure of operations between initial constant to restore true values of coefficients.

To obtain refining, the use of the aggregate indicator is proposed, which takes into account both aspects identified in the model, its performance statistics and complexity of the expression. A proposed structure for the aggregated performance AP indicator is calculated using the formula used in [14]:

$$AP(M) = MSE(M)^p \cdot C(M)^q \text{, where}$$

MSE(M) –*MSE* indicator for model *M*;
C(M) – complexity of *M*.
p – importance coefficient for statistical performance indicator, p>=0
q - importance coefficient for complexity, q>=0.

Aggregate indicator of performance, AP retains the minimum criteria that needs to be fulfilled as complexity must be minimized, deviations should be minimized, and the composition by multiplication in the ranges of values for the operands lead to an increasing function for both values. By using AP indicator the algorithm is forced to promote individuals which have developed both characteristics of good estimation of the phenomenon and small complexity, serving the objectives of refining models. Depending on the values chosen for parameters *p* and *q*, the importance given to each criterion shall be considered.

The complexity of the model obtained using aggregate indicator as selection criterion in the genetic algorithm drops and the capacity to estimate the phenomenon studied remains comparable to that of complex models obtained taking into account only the statistical performance indicators.

Another indicator of aggregate performance structure has the form presented in [10]:

$AP'(M) = MSE^p + C^q$,

where elements appear with the same meaning.

The criterion used by the algorithm for this type of aggregated performance indicator is also to be minimized. Generated model distribution is studied after running for a number of times the algorithm for a data set, results are presented in Table 3.

Models are correlated with the corresponding model structure as shown in Table 4.

Model structures show different frequencies of apparition as shown in Table 5.

The average complexity of the generated model structures in this sample is 2.991324. The average complexity of the generated model structures in the previous method is 4.795826. There is a decrease in the average complexity of models generated, highlighting the simple models with a better quality of estimates, having their appearance with an increased frequency compared to refining method of structure based on only a statistical indicator of performance. Structure ranking is shown in Table 6.

Table 3. Generated models using as fitness function AP' with p=1 q=2

ID	Model expression	Complexity	AP' (p=1, q=2)
M_1	(TIME/(-0.7917/-0.0318))	6.75	224.08
M_2	((TIME^0.3053)/0.3053)	6.75	118.00
M_3	(TIME^0.4994)	2	29.01
M_4	(TIME^0.4927)	2	28.57
M_5	(TIME^exp((-0.3632+-0.3632)))	9.50	118.87
M_6	((TIME^exp(0.2926))^0.3492)	9.50	135.06
M_7	(TIME*0.0257)	2	18.66
M_8	(TIME^0.3737)	2	248.80
M_9	(TIME^exp(-0.7347))	4	47.57
M_{10}	((TIME^0.4151)+0.4151)	6.75	188.14
M_{11}	((TIME^0.4894)+-0.1585)	6.75	70.62
M_{12}	(TIME^0.4637)	2	54.73
M_{13}	(0.0827*(TIME*0.1698))	6.75	213.66
M_{14}	(TIME^(exp(0.3643)*0.3643))	9.50	150.33
M_{15}	(TIME^0.3895)	2	212.20
M_{16}	(TIME^(0.3712^0.6982))	6.75	71.03
M_{17}	(((-0.6582+0.3100)*TIME)*-0.0977)	12.75	223.07
M_{18}	((TIME^0.0814)/0.0814)	6.75	225.18
M_{19}	((0.6948+TIME)^(0.6948* 0.6948))	12.75	191.84
M_{20}	(((-0.9328+0.4791)+ 0.4791)* TIME)	12.75	178.25
M_{21}	(TIME^0.4637)	2	54.73
M_{22}	(0.6893+(TIME^exp(-0.8005)))	9.50	161.28
M_{23}	(TIME^0.4115)	2	160.73
M_{24}	(TIME^(exp(exp(0.5545))^-0.4340))	12.75	205.19
M_{25}	(-0.0318/(-0.7917/TIME))	6.75	224.08
M_{26}	((TIME*-0.1369)*-0.2264)	6.75	74.56

Table 4. Correspondence between generated models and identified structures

Model	Structure	Model	Structure
M1	$A*TIME$	M14	$TIME^A$
M2	$A*TIME^B$	M15	$TIME^A$
M3	$TIME^A$	M16	$TIME^A$
M4	$TIME^A$	M17	$A*TIME$
M5	$TIME^A$	M18	$A*TIME^B$
M6	$TIME^A$	M19	$(A+TIME)^B$
M7	$A*TIME$	M20	$A*TIME$
M8	$TIME^A$	M21	$TIME^A$
M9	$TIME^A$	M22	$TIME^A+B$
M10	$TIME^A+B$	M23	$TIME^A$
M11	$TIME^A+B$	M24	$TIME^A$
M12	$TIME^A$	M25	$A*TIME$
M13	$A*TIME$	M26	$A*TIME$

Table 5. Apparition frequencies of model structures

Model Structure	Complexity	Absolute frequency	Relative Frequency
$A*TIME$	2	7	0.2692307
$A*TIME^B$	6.754888	2	0.076923
$TIME^A$	2	13	0.5
$TIME^A+B$	6.754888	3	0.1153846
$(A+TIME)^B$	4	1	0.0384615
Total	-	26	1

Table 6 Structure ranking

Structure ID	Model Structure	Absolute frequency	Relative Frequency	Cumulated frequency
S_1	*$TIME^A$*	13	0.5	0.5
S_2	*$A*TIME$*	7	0.2692	0.769231
S_3	*$TIME^A+B$*	3	0.1153	0.884615
S_4	*$A*TIME^B$*	2	0.0769	0.961538
S_5	*$(A+TIME)^B$*	1	0.0384	1

Table 7. Generated models for $TIME^A$ structure

Generated model	Evolved coefficient
(TIME^exp(-0.72498476445907))	0.484331948
(TIME^0.451378393662804)	0.451378393662804
((TIME^0.519262224211945)^0.957872770707064)	0.497387145
(TIME^exp(-0.616214980658244))	0.539984426
(TIME^0.530303061255395)	0.530303061255395
((TIME^0.72121787337643)^0.72121787337643)	0.520155221
(TIME^exp(-0.821984323590055))	0.439558562
(TIME^0.472517949749026)	0.472517949749026
(TIME^0.514797061921468)	0.514797061921468
((TIME^exp(0.349290777626117))^0.349290777626117)	0.495315794
(TIME^0.470532909254792)	0.470532909254792
(TIME^(-0.321436142698599/-0.733539882923262))	0.43819859
(TIME^exp(-0.644560026770718))	0.52489343
(TIME^0.469861407983052)	0.469861407983052
(TIME^0.462390707089748)	0.462390707089748
(TIME^0.476668837702213)	0.476668837702213
(TIME^exp(-0.644560026770718))	0.52489343
(TIME^(0.681852963604896*0.681852963604896))	0.464923464
(TIME^0.489165376633948)	0.489165376633948
(TIME^0.500250477111084)	0.500250477111084
(TIME^exp(-0.662999510608148))	0.515303356
(TIME^0.471989210914815)	0.471989210914815

The first two structures that the algorithm proposes are also presented using the structure refinement method. Using the aggregate performance indicator as a criterion for ordering those simpler structures represent about 77% of all models generated.

The M_{S1} model that corresponds to S_1 structure has the form:

M_{S1}: *$CUMULATIVE = TIME^{0.495}$*.

The M_{S2} model is obtained from S_2 structure after estimating the coefficients and has the form:

M_{S2}: *CUMULATIVE* = 0.027**TIME*.
The characteristics of those models have been presented previously.
To study the influence of using the aggregated performance indicator over the distribution of evolved constants and the placement of classically estimated coefficients, the generated models corresponding to the $TIME^A$ structure are presented in Table 7.
Descriptive statistics is presented in Table 8.

Table 8. Descriptive statistics for model coefficients

Indicator	Value
Mean	0.488855
Median	0.486749
Mode	0.524893
Standard deviation	0.000887
Range	0.101786
Minimum	0.438199
Maximum	0.539984
Sum	10.7548
Number	22
Trusted range(95.0%)	0.013201

With a probability of 95%, the mean of the evolved coefficient population is range [0.488855-0.013201; 0.488855+0.013201], containing the value of the classically estimated coefficient. It is shown that using the aggregated performance indicator as selection criterion does not influence the capacity of the algorithm to converge to valid coefficients.

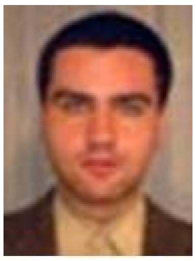
5 Conclusion

Model structure refinement helps analysts by automating model generation process, using verified generation algorithms and by automating model selection process, using objective criteria for selection.
Model structure refinement is necessary in order to reduce the almost infinite solution space containing models that estimate the value of a dependent variable, to a small set.
Using genetic algorithms for model generation, the analyst has control over the parameters of the algorithm, over the operand types, over the selection criteria, having the instruments to create less complex, but fit and easy to interpret models. The best model is recorded for a large number of algorithm runs while observing if there are model structures that have a higher frequency of apparition. The set of selected structures with higher occurrence is used to build models.
Validation is necessary, when choosing a model, but also, during effective usage, the results obtained through the model must be compared with the real recorded values. If the differences are large, a new estimation of coefficients or a new model structure refinement must be done to adapt to the new evolution of the process.
The aggregated performance indicator shows its usefulness for model selection, in order to respect the principles of model refinement. The obtained results indicate that it does not affect the ability of the algorithm to give a good solution for the given dataset.
Both proposed methods highlight the fact that the quality of the solutions increases as the algorithm is run for a sufficient number of times. This way, the negative influence of random elements inside the algorithm is removed.

References

[1] ISO/IEC 9126-1:2001, *Software engineering - Product quality - Part 1: Quality model*, International

Organization for Standardization, 2001.
[2] S. Culp, *Definition of a Security Vulnerability*, 2008, Available at: http://technet.microsoft.com/ro-ro/library/cc751383(en-us).aspx
[3] *Common Vulnerabilities and Exposures* [Online], Available at: http://www.cve.mitre.org/
[4] *Security Focus* [Online], Available at: http://www.securityfocus.com/bid
[5] *The Open Source Vulnerability Database* [Online], Available at: http://osvdb.org/
[6] O. H. Alhazmi and Y. K. Malaiya, "Modeling the Vulnerability Discovery Process," in *Proc. 16th International Symposium on Software Reliability Engineering*, Chicago, USA, 2005, pp. 129-138
[7] O. H. Alhazmi, Y. K. Malaiya and I. Ray, "Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems," *Computers and Security Journal*, vol. 26, Issue 3, pp. 219-228, May 2007.
[8] I. Ivan and A. Visoiu, "Rafinarea metricilor software," *Economistul*, supliment Economie teoretică si aplicativă, No.1947-2973, 2005.
[9] I. Ivan and A. Visoiu, "A Comparative Analysis of Software Refinement Techniques", *in Proc. Cybernetics and Information Technologies, Systems and Applications CITSA 2008*, Orlando, Florida, USA, 2008, pp. 235-239.
[10] A. Visoiu, "Rafinarea metricilor software", PhD Thesis, Academy of Economic Studies, Bucharest, 2009.
[11] *Apache httpd Security Report* [Online], Available at: http://httpd.apache.org/security_report.html
[12] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence 2nd Edition*, Springer Publishing, 2006.
[13] A. Visoiu, "Structure Refinement for Vulnerability Estimation Models using Genetic Algorithm Based Model Generators", *Informatica Economică Journal* [Online], Vol. 13, No. 1 , 2009, Available at: http://revistaie.ase.ro/content/49/007%20-%20Visoiu.pdf
[14] A. Visoiu, "Performance Criteria for Software Metrics Model Refinement", *Journal of Applied Quantitative Methods* [Online], Volume 2, Issue 1, pp. 118-128, 2007, Available at: http://www.jaqm.ro/issues/volume-2,issue-1/pdfs/visoiu.pdf



Adrian VISOIU graduated the Bucharest Academy of Economic Studies, the Faculty of Cybernetics, Statistics and Economic Informatics. He has a master degree in Project Management. He is an assistant lecturer in the Economic Informatics Department of the Bucharest Academy of Economic Studies. He published 16 articles alone or in collaboration and he is coauthor of three books. His interests include: object oriented programming, data structures, multimedia programming, software quality management, software metrics refinement.