# Structure Refinement for Vulnerability Estimation Models using Genetic Algorithm Based Model Generators

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

*In this paper, a method for model structure refinement is proposed and applied in estimation of cumulative number of vulnerabilities according to time. Security as a quality characteristic is presented and defined. Vulnerabilities are defined and their importance is assessed. Existing models used for number of vulnerabilities estimation are enumerated, inspecting their structure. The principles of genetic model generators are inspected. Model structure refinement is defined in comparison with model refinement and a method for model structure refinement is proposed. A case study shows how the method is applied and the obtained results.*

**Keywords**: *model structure refinement, model generators, gene expression programming, software vulnerabilities, performance criteria, software metrics.*

## 1 Quality and Security Assessment

A software quality system is made up of characteristics and attributes or subcharacteristics. Consider the system containing *m* characteristics $CH_1$, $CH_2$, …, $CH_m$. For each characteristic $C_j$ models are built to estimate its level. Such model takes the form:

$$M_{jh}\text{:}\ y_{jh} = f_{jh}(X_1, X_2, ...X_k)$$

where

$M_{jh}$ – the $h^{\text{th}}$ model for estimating the $CH_j$ characteristic

$y_{\text{jh}}$ - estimated level of $CH_j$ through $M_{jh}$

$X_1$, $X_2$, .., $X_k$ – independent variables associated to the identified influence factors.

In ISO 9126 standard [1], security is defined as an attribute of the functionality quality characteristic, related to the ability of a software product to prevent unauthorized access intentional or unintentional to programs and data.

For a proper software quality management, indicators must be built and used to measure different aspects of security. Also, models are built in order to estimate the indicators with respect to factors that influence the phenomenon.

There are software categories for which security is a very important aspect: operating systems, web servers, web browsers, application servers, office suites, applications that offer network services.

An important aspect in assessing security is represented by vulnerabilities. Vulnerabilities represent an important fraction of the software flaws that need to be repaired.

In [2] the following definition is used: *"A security vulnerability is a flaw in a product that makes it infeasible – even when using the product properly —to prevent an attacker from usurping privileges on the user's system, regulating its operation, compromising data on it, or assuming ungranted trust".*

The vulnerability discovery process is very important because any vulnerability permits an attacker to cause important losses in terms of data and money. The developer must patch any vulnerability found in order to keep its position on the market and gain the trust of the users.

When a vulnerability is found, it is made public and reported to public databases like [3], [4] and [5] for further observation. The fact the vulnerability is found and made public does not guarantee that it hasn't already been found by possible attackers and hasn't already been used to produce damages.

The vulnerability discovery process is useful also for managers. If the process follows a definite law, models are developed to help the decision making. It is useful to know that there are moments after the release date of the product when more resources must be allocated for treating vulnerabilities, and moments, when such resources can be directed to other purposes. 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.

Recording the number of vulnerabilities discovered for a certain software product helps estimation of future values and also for comparison between products from the same developer or different developers.

## 2. Software Vulnerabilities Estimation Models

When a phenomenon or a process is modeled, the nature of the variables and factor interaction are analyzed and a set of models is identified, models which accurately represent the structure, the dynamics and the behavior of the phenomenon or the process. A criterion for model ordering is defined, data is recorded, model coefficients are estimated and model quality assessed. From the ordered list of models, only a few are chosen and those are taken into account for validation. After validating models, only a model is left and this is subject for refinement.

In [6] are presented many models used for estimating software vulnerabilities. They are used for predicting the cumulative number of vulnerabilities, at a certain moment in time, denoted by Ω(t), starting with the moment of the product release date:

- Anderson Thermodynamic Model:

$$\Omega(t) = \frac{k}{\gamma}\ln(C \cdot t)$$

where $k$ is a constant and γ is a weighting factor;

- Alhazmi-Malaiya Logistic Model:

$$\Omega(t) = \frac{B}{B \cdot C \cdot e^{-ABt} + 1},$$

where A and $C$ are constants and $B$ is the total number of vulnerabilities estimated to be found;

- Rescorla Linear Model:

$$\Omega(t) = \frac{B \cdot t^2}{2} + K \cdot t,$$

where $B$ are $K$ parameters;

- Rescorla Exponential Model:

$$\Omega(t) = N(1 - e^{-\lambda t}),$$

where $N$ is the total number of vulnerabilities and λ is a rate;

- Logarithmic Poisson Model (LP):

$$\Omega(t) = \beta_0 \ln(1 + \beta_1 \cdot t),$$

where $\beta_0$ and $\beta_1$ are coefficients.

Also, in [7], the linear model is presented and tested for cumulative number of vulnerabilities estimation:

$$\Omega(t) = S \cdot t + k,$$

where $S$ is the slope and $k$ is a constant.

Those models have been tested and applied for operating systems, web servers and other vulnerability prone software, both open source and closed source. Those models have different structures, and give different results for different datasets. When analyzing a dataset, it is useful to develop specific models that work best with it, than using a model developed using another dataset having other characteristics.

## 3. Genetic Algorithm Based Model Generators

Model generators are software instruments for obtaining models from a certain model class given the list of variables, the model structure, existence restrictions and datasets.

Model classes group models with the same structure, e.g. linear models, linear models with lagged variables, nonlinear models. For each class a model generator is developed as a software module. Each dataset contains data series for the recorded variables. The endogenous variable is specified and the generator builds analytical expressions using influence factors, coefficients, simple operators and functions. For each model structure, coefficients are estimated and a performance indicator is computed. The resulting model list is ordered by the performance indicator. The analyst chooses between the best models an appropriate form that later will be used in estimating the studied characteristic.

In [8], linear model generators are presented. In [9], linear model generators with lagged variables are presented. Nonlinear model generators based on combinatorial algorithms are presented in [10]. Model generators based on genetic algorithms are presented in [11].

Those instruments are useful for obtaining models that estimate the evolution of a certain phenomenon influenced by a list of factors. As seen, vulnerabilities are estimated according to the time factor. Model generators are used to develop a model expression taking into account time, operators, coefficients and functions.

When using genetic type algorithms, which implement the model of population evolution, one important application is symbolic regression. Symbolic regression evolved itself with the introduction of genetic programming and later with the gene expression programming as presented in [12]. Starting from a dataset in which it is specified the dependent variable and the independent variables, an initial population of chromosomes is built and then it is subject to a replication process including specific rules.

These algorithms have a very specific way of representing analytical expressions of models. The chromosome is a linear structure of fixed length made up of genes. The role of the chromosome is to code an analytical expression. A gene is also a linear structure and corresponds to the syntax tree of the expression it represents and each entry in the gene corresponds to a tree node. The tree nodes are numbered starting from the root and then level by level and from left to right obtaining a linear structure. Each node contains an operator, a constant or a variable. Like the nonlinear generator does, the domain for the generated expressions depends on the accepted operators set and the set of operands built up from variables and coefficients. To create the final expression from the chromosome, the subexpressions derived from genes are aggregated using a simple aggregation function like summation of multiplication.

An initial population of chromosomes is randomly generated. The evolution is obtained through iteratively applying genetic operators:

- elitism implies extracting the best chromosome from the current population and its replication as is in the next generation;
- selection implies extracting a number of individuals from the population based on a measure for their fitting to the aimed objective
- mutation implies random changes of some positions in the chromosome; a certain position contains an operand or an operator and its change leads to a new analytical form, differing from the initial one, when the expression is rebuilt; this genetic operator is essential to introduce a degree of variability in the generation process
- transposition implies changing the position of sequences of elements from a gene inside a chromosome
- recombination implies pairing two parent chromosomes and obtaining a new chromosome inheriting contents from both parents

The genetic operators are applied for a number of generations leading to a best fitting model in the given hypothesis.

An issue in analytical expression generation using gene expression programming is the building of apparent high complexity expressions in contrast with the objective of model refinement. However, this is not the case as simple operating the constants reduces the model complexity to a simpler form.

## 4. Model Structure Refinement Using Genetic Algorithms

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 [13]. 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 [14].

In [13] model refinement with linear model generation was applied. The purpose of that case study was creating a refined model to estimate McCabe complexity of a module according to a list of influence factors. A model list was built containing the best generated models according to a statistical performance criterion. It was observed that certain variables had a higher frequency of apparition than others, and also the associated coefficients in the model kept the same sign and similar values. This fact conducted to the idea that when using model generators, the analyst should pay attention to the distribution of generated model structures in order to find patterns that indicate a model structure is more fit for the purpose of the research.

A method is proposed for model structure refinement. In order to define model structure refinement, 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. Refinement can be 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. Genetic algorithms in general, and gene expression programming used in the case study in particular have a pseudorandom behavior. Building the initial population is pseudorandom: expression elements are randomly chosen from a list of elements; variables are chosen from the dataset and constants making up coefficients are chosen from a random generated list of values. At each run, the constant list is different. Also, the genetic operators like selection, mutation, the exchange of genetic material are applied randomly. When running the algorithm for several times, using the same dataset, the best generated models are different regarding the apparent structure and the coefficients. However, after applying all operations between constants, 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.

The model structure obtained by applying a genetic algorithm for model generation is complex due to the way coefficients are estimated. A certain coefficient is estimated by evolution, not by classical optimization methods. The coefficient is estimated by a model expression part that contains operations between initial constants. Through evolution, the expression improves over the generations and finally builds up, or approximate, the coefficient. Suppose that the algorithm uses random generated constants in [-1; 1] range. This does not affect generality as expression evolution also produces the generation of other real constants. Consider that in the model, the constant 9.7 is needed. Presuming that constants 0.1 and 0.97 exist in the random generated constant list, 9.7 is constructed using the division operator through the expression 0.97/0.1.

The model given as output from a gene expression programming model generator, where *TIME* is a variable

(0.1253548*(0.1253548/((0.3993210/0.7591132)/TIME)))

has 5 operands and 4 operators and its apparent complexity is 19.6. Note the way the expression can be further reduced. Step by step, constants are operated:

(0.1253548*(0.1253548/((0.5260361)/TIME)))
0.1253548*0.2383007*TIME
0.0298721*TIME.

The real model structure lying beneath the initial expression is $a$**TIME*. It has 2 operands and 1 operator, its Halstead complexity is 2. The $a$ coefficient is approximated by operating the above constants.

When running the algorithm for a specified number of times certain model structures appear with a higher frequency than 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.

$$\sum_{i=1}^{n} f_i = 1 .$$

The performance of a certain model $M_i$ differs from the performance of the model obtained by estimating the coefficients of the corresponding structure $S_j$ by using an optimization method like least squares. There are models of type $S_{j1}$ that perform better than the model of another type $S_{j2}$, as there are models in $S_{j2}$ that estimate better than the models in $S_{j1}$. At this moment it cannot be decided which model class gives better results. It is assumed the performance of the models is comparable. The list of model structures is sorted in descending order according to the frequency of apparition, obtaining the list $S_{k1}$,$S_{k2}$, …, $S_{kn}$, 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 with respect to the relation

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

The models $M_{k1}$, $M_{k2}$, …, $M_{ks}$ are built, having the corresponding chosen structures $S_{k1}$, $S_{k2}$, …, $S_{ks}$ and their coefficients are estimated using a least squares algorithm. For each model complexity *C* is measured and performance or fitness *FIT* is computed.

Structurally, a model is made up of expressions containing operands and operators. The model *M* has its complexity is assessed by an indicator that emphases the operation volume:

$$C(M) = n_1 \log_2 n_1 + n_2 \log_2 n_2 ,$$

where

$n_1$ – the number of operands in the model
$n_2$ – the number of operators in the model.

The fitness function that assesses the statistical performance of a model M is

$$FIT(M) = \frac{\sum_{i=1}^{l} e_i^2}{l} ,$$

where

*l* – the length of the dataset
$e_i$ – the difference between the actual value of the dependent variable in the dataset and the estimated value using the models

In order to achieve refinement, the complexity of the model is taken into account, when evaluating model lists. A performance indicator that takes into account both statistical performance and complexity to assess a model *M* is given by

$$AP(M) = FIT(M)^p \cdot C(M)^q ,$$

where

*FIT(M)*– statistical performance of model *M*
*C(M)* – complexity of *M*
*p – importance coefficient for FIT*
*q - importance coefficient for C.*

The properties of such aggregated performance indicator are presented in [15].

A table is built containing information about structures, models and indicators shown in table 1.

**Table 1.** The refined structure list, the corresponding models and their characteristics

| Structure | Model | Performance | Complexity | Aggregated performance |
|---|---|---|---|---|
| $S_{k1}$ | $M_{k1}$ | $FIT(M_{k1})$ | $C(M_{k1})$ | $AP(M_{k1})$ |
| $S_{k2}$ | $M_{k2}$ | $FIT(M_{k2})$ | $C(M_{k2})$ | $AP(M_{k2})$ |
| ... | ... | … | … | … |
| $S_{ks}$ | $M_{ks}$ | $FIT(M_{ks})$ | $C(M_{ks})$ | $AP(M_{ks})$ |

In the context of model generation, refinement aims to choose a model with good statistical performance and a small complexity from a list of models having different values for those characteristics. The list in table 1 is sorted ascending by the *AP* indicator and the first model structure is further used. If the objective of the analysis is obtaining a very precise model, then when evaluating the list of models using AP, only the quality of the fitness is taken into account, then *q*=0. If the objective of the analysis is to obtain a good, simple model, that is easy to apply and interpret, the analyst chooses positive values for *q*. The first model structure in the ordered list is the refined model structure.

## 5. Experimental results

In order to test the proposed method, a case study shows the results obtained in model structure refinement.

Many open source projects have reached a degree of maturity where software development is under a well defined management.

The development of a widely used web server, the *Apache httpd project*, is under such strict management that assures consistent results and a sustainable development process. Public information about the project is found at [16]. For this project there is a public section where all the security vulnerabilities are presented [17]. It is intended to build a model to estimate the number of cumulative vulnerabilities at a certain moment in time.

The raw data collected from the published security reports [16] contains information about the released version of the software, the date of the release, the number of security issues classified as low - LOW, moderate - MOD, important -IMP and critical - CRI that were fixed with that version of software. The raw data is presented in table 2.

**Table 2.** Raw data collected from security reports

| VERSION | DATEFIXED | LOW | MOD | IMP | CRI | TOTAL |
|---|---|---|---|---|---|---|
| Apache httpd 2.0.35 | 06.04.2002 | | | | | 0 |
| Apache httpd 2.0.36 | 08.05.2002 | 1 | | | | 1 |
| Apache httpd 2.0.37 | 18.06.2002 | | | | 1 | 1 |
| Apache httpd 2.0.40 | 09.08.2002 | 1 | | 1 | | 2 |
| Apache httpd 2.0.42 | 24.09.2002 | | 1 | | | 1 |
| Apache httpd 2.0.43 | 03.10.2002 | 1 | 1 | | | 2 |
| Apache httpd 2.0.44 | 20.01.2003 | | | 1 | 1 | 2 |
| Apache httpd 2.0.45 | 02.04.2003 | | | 1 | | 1 |
| Apache httpd 2.0.46 | 28.05.2003 | 1 | | 2 | 1 | 4 |
| Apache httpd 2.0.47 | 09.07.2003 | 1 | 1 | 1 | | 3 |
| Apache httpd 2.0.48 | 27.10.2003 | 1 | 1 | | | 2 |
| Apache httpd 2.0.49 | 19.03.2004 | 1 | | 2 | | 3 |
| Apache httpd 2.0.50 | 01.07.2004 | 1 | | 1 | | 2 |
| Apache httpd 2.0.51 | 15.09.2004 | 3 | | 1 | 1 | 5 |
| Apache httpd 2.0.52 | 28.09.2004 | | | 1 | | 1 |
| Apache httpd 2.0.53 | 08.02.2005 | 1 | 1 | 1 | | 3 |
| Apache httpd 2.0.55 | 14.10.2005 | 3 | 2 | 1 | | 6 |
| Apache httpd 2.0.58 | 01.05.2006 | 1 | 1 | | | 2 |
| Apache httpd 2.0.59 | 27.07.2006 | | | 1 | | 1 |
| Apache httpd 2.0.61 | 07.09.2007 | | 4 | | | 4 |
| Apache httpd 2.0.63 | 19.01.2008 | 1 | 2 | | | 3 |

A cumulative series is built for the total number of vulnerabilities, denoted by CUMULATIVE, and also a cumulative series is built for de difference measured in days between two version release dates, denoted by TIME. Table 3 presents the dataset containing cumulative values.

**Table 3.** Cumulative series for the number of found vulnerabilities and the difference

| VERSION | TOTAL | CUMULATIVE | DATEFIXED | DAYDIF | TIME |
|---|---|---|---|---|---|
| Apache httpd 2.0.35 | 0 | 0 | 06.04.2002 | 0 | 0 |
| Apache httpd 2.0.36 | 1 | 1 | 08.05.2002 | 32 | 32 |
| Apache httpd 2.0.37 | 1 | 2 | 18.06.2002 | 41 | 73 |
| Apache httpd 2.0.40 | 2 | 4 | 09.08.2002 | 52 | 125 |
| Apache httpd 2.0.42 | 1 | 5 | 24.09.2002 | 46 | 171 |
| Apache httpd 2.0.43 | 2 | 7 | 03.10.2002 | 9 | 180 |
| Apache httpd 2.0.44 | 2 | 9 | 20.01.2003 | 109 | 289 |
| Apache httpd 2.0.45 | 1 | 10 | 02.04.2003 | 72 | 361 |
| Apache httpd 2.0.46 | 4 | 14 | 28.05.2003 | 56 | 417 |
| Apache httpd 2.0.47 | 3 | 17 | 09.07.2003 | 42 | 459 |
| Apache httpd 2.0.48 | 2 | 19 | 27.10.2003 | 110 | 569 |
| Apache httpd 2.0.49 | 3 | 22 | 19.03.2004 | 144 | 713 |
| Apache httpd 2.0.50 | 2 | 24 | 01.07.2004 | 104 | 817 |
| Apache httpd 2.0.51 | 5 | 29 | 15.09.2004 | 76 | 893 |

| | | | | | |
|---|---|---|---|---|---|
| Apache httpd 2.0.52 | 1 | 30 | 28.09.2004 | 13 | 906 |
| Apache httpd 2.0.53 | 3 | 33 | 08.02.2005 | 133 | 1039 |
| Apache httpd 2.0.55 | 6 | 39 | 14.10.2005 | 248 | 1287 |
| Apache httpd 2.0.58 | 2 | 41 | 01.05.2006 | 199 | 1486 |
| Apache httpd 2.0.59 | 1 | 42 | 27.07.2006 | 87 | 1573 |
| Apache httpd 2.0.61 | 4 | 46 | 07.09.2007 | 407 | 1980 |
| Apache httpd 2.0.63 | 3 | 49 | 19.01.2008 | 134 | 2114 |

Note that the first release 2.0.35 does not contain any security fixes. It is considered as a starting point in our analysis.

The evolution of the cumulative number of vulnerabilities with time is given in figure 1.

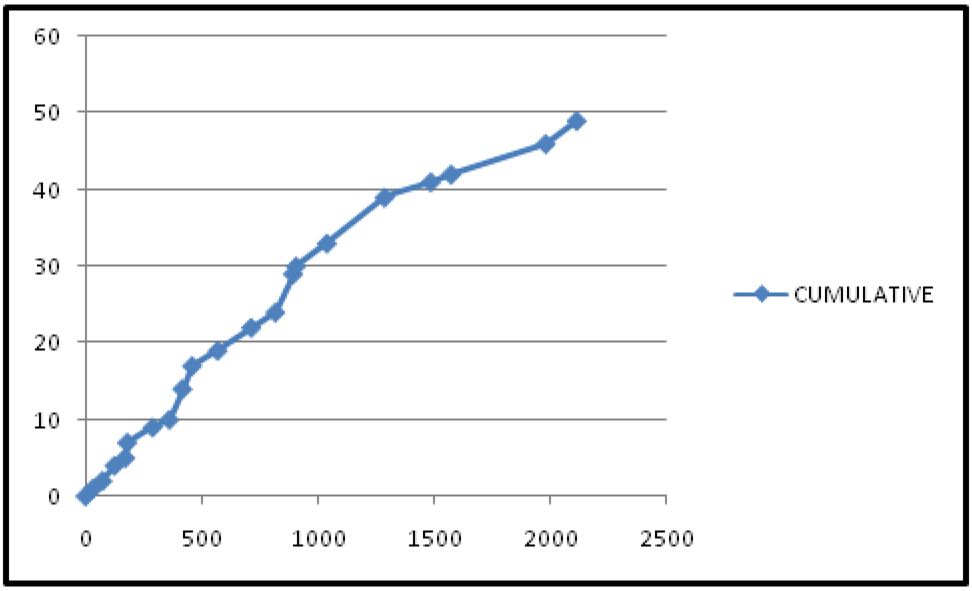

**Fig. 1.** The evolution number of fixed vulnerabilities in time

The needed model has to estimate the cumulative number of vulnerabilities covered – *CUMULATIVE* with respect to the number of days elapsed from the launch of the product. Using the gene expression programming generator a sample of 23 models is generated. Table 4 shows the list of generated models accompanied by their statistical performance indicator.

**Table 4.** Generated models output from the gene expression programming generator

| Model ID | EXPRESSION | FIT |
|---|---|---|
| $M_1$ | (((TIME*(0.558161191436537*0.558161191436537))^(0.768022689394663^0.72624326996796))^0.72624326996796) | 12.6063424743838 |
| $M_2$ | (0.0485667032415824*((TIME/(0.485911058953922+0.380868223673137))*0.485911058953922)) | 13.361586437786 |
| $M_3$ | (0.816733738322153*(((0.194515941755155/0.630825894247194)*TIME)^0.630825894247194)) | 9.12573472267031 |
| $M_4$ | ((TIME^0.517728796469853)+ln((0.233303776119511+(0.00702137127845612*0.00702137127845612)))) | 28.9973223280455 |
| $M_5$ | (TIME*(0.0128907486856406^0.837603060453014)) | 13.8676424057511 |
| $M_6$ | (0.52023783536639+(TIME^0.52023783536639)) | 56.5486463117635 |
| $M_7$ | ((0.163407823146976*(TIME+0.976262840896967))*0.163407823146976) | 13.2519176757507 |
| $M_8$ | (TIME^(((0.859372181752404*0.680791525487225)/0.680791525487225)*0.543734460856642)) | 45.6893294145408 |
| $M_9$ | (TIME*(((0.846643705315256/0.127865843068746)/0.127865843068746)^ln(0.403221100756536))) | 13.890390795984 |

| | | |
|---|---|---|
| $M_{10}$ | ((TIME*(0.877454309667207/(0.460519756404925/0.0257088388436049)))^0.877454309667207) | 10.6126614831786 |
| $M_{11}$ | ((TIME*((0.0632099733050959*0.773589949018131)*(0.773589949018131*0.773589949018131)))^0.998194032347851) | 17.4135343874989 |
| $M_{12}$ | ((0.702126900061093*TIME)^0.518656560461343) | 20.6603265718446 |
| $M_{13}$ | (0.560466382447847*(TIME^0.560466382447847)) | 22.8284625410467 |
| $M_{14}$ | (TIME^(0.39682478150205*((0.877461587487469/0.703766557250063)^0.877461587487469))) | 19.6096404744368 |
| $M_{15}$ | (TIME^(0.39682478150205*((0.877461587487469/0.703766557250063)^0.877461587487469))) | 29.9609686522458 |
| $M_{16}$ | (TIME^0.477832195105884) | 33.2239762050619 |
| $M_{17}$ | (TIME*0.0261055706190437) | 13.9066543592166 |
| $M_{18}$ | (TIME*(0.851695158449791^((0.726531111042263^0.298261328273574)/(0.0120070627946439/0.298261328273574)))) | 13.3513484193383 |
| $M_{19}$ | (TIME^0.508447564443782) | 30.8635969349109 |
| $M_{20}$ | (TIME^0.511974330764252) | 35.0301571388812 |
| $M_{21}$ | (ln((0.609602831122281+0.609602831122281))*((0.325523775688151^0.0005415859634716)+(TIME^0.609602831122281))) | 215.045427274938 |
| $M_{22}$ | (TIME^((0.260342915197994+(0.169856560029954^0.260342915197994))*0.544435880866105)) | 27.6292267234255 |
| $M_{23}$ | (((TIME*0.539203863842042)^0.539203863842042)+0.0319035942814795) | 18.0775110750353 |

The corresponding model structures for the models in table 4 are shown in table 5.

**Table 5.** Models and their corresponding structure

| Model ID | Structure |
|---|---|
| $M_1$ | $A*TIME^B$ |
| $M_2$ | $A*TIME$ |
| $M_3$ | $A*TIME^B$ |
| $M_4$ | $TIME^A+B$ |
| $M_5$ | $A*TIME$ |
| $M_6$ | $TIME^A+B$ |
| $M_7$ | $A*TIME+B$ |
| $M_8$ | $TIME^A$ |
| $M_9$ | $A*TIME$ |
| $M_{10}$ | $A*TIME^B$ |
| $M_{11}$ | $A*TIME^B$ |
| $M_{12}$ | $A*TIME^B$ |
| $M_{13}$ | $A*TIME^B$ |
| $M_{14}$ | $TIME^A$ |
| $M_{15}$ | $TIME^A$ |
| $M_{16}$ | $TIME^A$ |
| $M_{17}$ | $TIME^A$ |
| $M_{18}$ | $A*TIME$ |
| $M_{19}$ | $TIME^A$ |
| $M_{20}$ | $TIME^A$ |
| $M_{21}$ | $A*TIME^B+C$ |
| $M_{22}$ | $TIME^A$ |
| $M_{23}$ | $A*TIME^B+C$ |

The gene expression programming based model generator has many parameters to be set:

- size of the population
- length of a gene
- number of genes in a chromosome
- operator list to be used containing +,-,*/, pow, ln, sin and other functions
- constant list length
- dataset variables specifying dependent and independent variables

As presented above, a large diversity of operators and functions used for generating models creates the impression that the number of model structures that instantiate models is also large.

It is observed that certain model structures appear with higher frequency and the structure diversity is small. The identified model distribution is presented in table no 6.

**Table 6.** Model structure apparition frequency

| Model Structure | Absolute frequency | Relative frequency |
|---|---|---|
| $A*TIME^B$ | 6 | 0.26087 |
| $A*TIME$ | 4 | 0.173913 |
| $TIME^A+B$ | 2 | 0.086957 |
| $A*TIME+B$ | 1 | 0.043478 |
| $TIME^A$ | 8 | 0.347826 |
| $A*TIME^B+C$ | 2 | 0.086957 |
| *TOTAL* | 23 | 1 |

Note that expressions containing *TIME* variable as argument for ln function, or denominator of a fraction were not generated as expected. Also, it is an interesting fact, that models contain only positive constants. This remained true for a larger number of algorithm runs than the sample presented here.

Ordering in descending order the model structures according to their relative frequency, table 7 is obtained, that also shows the cumulative frequencies.

**Table 7.** Model structure list ordering

| Structure ID | Model Structure | Absolute frequency | Relative frequency | Cumulative frequency |
|---|---|---|---|---|
| $S_1$ | $TIME^A$ | 8 | 0.347826 | 0.347826 |
| $S_2$ | $A*TIME^B$ | 6 | 0.26087 | 0.608696 |
| $S_3$ | $A*TIME$ | 4 | 0.173913 | 0.782609 |
| $S_4$ | $TIME^A+B$ | 2 | 0.086957 | 0.869566 |
| $S_5$ | $A*TIME^B+C$ | 2 | 0.086957 | 0.956523 |
| $S_6$ | $A*TIME+B$ | 1 | 0.043478 | 1.000001 |

As a note, all the generated models take the form or particular forms of $A*TIME^B+C$. If threshold $h$=1 is chosen then all the structures are chosen for further processing. Based on the chosen structures, models are built and coefficients are estimated.

$$M_{S1}: CUMULATIVE = TIME^{0.495}\ FIT=24.37\ C=2$$
$$M_{S2}: CUMULATIVE = 0.14 * TIME^{0.772}\ FIT= 4.36\ C=6.75$$
$$M_{S3}: CUMULATIVE = 0.027*TIME\ FIT = 13.28\ C=2$$
$$M_{S4}: CUMULATIVE = TIME^{0.527}-7.796\ FIT=6.89\ C=6.75$$
$$M_{S5}: CUMULATIVE = 0.264*TIME^{0.695}-2.619\ FIT=3.59\ C=12.75$$
$$M_{S6}: CUMULATIVE = 0.025*TIME+2.972\ FIT=9.75\ C=6.75$$

To evaluate the aggregated performance indicator, different values for p and q are chosen. The data about the models is structured in table 8.

**Table 8.** Characteristics for models based on the structure list

| Model ID | FIT | C | AP(p=1;q=0) | AP(p=0;q=1) | AP(p=1;q=1) | AP(p=2;q=1) |
|---|---|---|---|---|---|---|
| $M_{S1}$ | 24.37 | 2 | 24.37 | 2 | 48.74 | 1187.794 |
| $M_{S2}$ | 4.36 | 6.75 | 4.36 | 6.75 | 29.43 | 128.3148 |
| $M_{S3}$ | 13.28 | 2 | 13.28 | 2 | 26.56 | 352.7168 |
| $M_{S4}$ | 6.89 | 6.75 | 6.89 | 6.75 | 46.5075 | 320.4367 |
| $M_{S5}$ | 3.59 | 12.75 | 3.59 | 12.75 | 45.7725 | 164.3233 |
| $M_{S6}$ | 9.75 | 6.75 | 9.75 | 6.75 | 65.8125 | 641.6719 |

Ordering the list by the $AP(p=1;q=0)$ is the same as ordering by *FIT*, obtaining: $M_{S5}$, $M_{S2}$, $M_{S4}$, $M_{S6}$, $M_{S3}$, $M_{S1}$. This ordering pays attention only to the statistical performance.

Ordering the list by the $AP(p=0;$ q=1) is the same as ordering by C obtaining: $M_{S3}$, $M_{S1}$, $M_{S2}$, $M_{S4}$, $M_{S6}$, $M_{S5}$. This ordering pays attention only to the complexity of the expression.

Ordering the list by the $AP(p=1; q=1)$ gives: $M_{S3}$, $M_{S2}$, $M_{S5}$, $M_{S4}$, $M_{S1}$, $M_{S6}$. This ordering pays attention to both statistical performance and complexity.

Ordering the list by the $AP(p=2; q=1)$ gives: $M_{S2}$, $M_{S5}$, $M_{S4}$, $M_{S3}$, $M_{S6}$, $M_{S1}$. This ordering pays more attention to the statistical performance than to complexity.

The best performance in terms of fitting quality is given by the $M_{S5}$ model. This also has the most complex expression. At the opposite, models $M_{S1}$ and $M_{S3}$ are the least complex, but also have the worst statistical performance. If performance has a greater importance than complexity, as when using $AP(p=2; q=1)$, but both are taken into account, then models $M_{S2}$ is chosen.

In the situation when $M_{S2}$ is chosen to estimate the number of vulnerabilities, there must be a validation step, in which to apply a statistical test.

The Chi Squared $X^2$ test is applied and the computed value is 8.1399. The critical value in this case, for a risk α=5%, the critical value is 27.58. As the computed value is less than the critical value, the model is accepted.

Also, model $M_{S2}$ has another advantage over $M_{S5}$. All the output values of $M_{S2}$ can be interpreted, such as the value at the initial moment, *TIME*=0, $M_{S2}(0)=0$, which also corresponds to the actual data. For $M_{S5}$, $M_{S5}(0)=-2.619$ which apparently doesn't have sense, as the number of vulnerabilities cannot be negative.

The graphic showing the actual recorded values versus the values given by $M_{S2}$ are presented in figure 2.

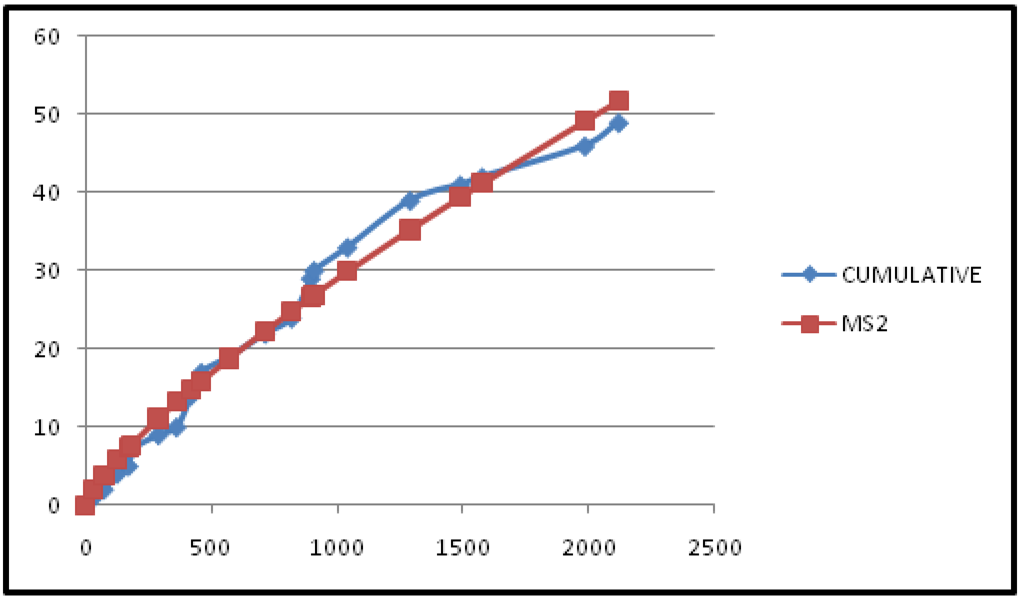

**Fig. 2.** $M_{S2}$ gives a good result in estimating the actual values

The statistical validation is always followed by the validation in practice, where over a period of time, model outputs are compared with the real encountered values.

## 6. Conclusions

Model generators are useful instruments, automating important aspects of phenomena modeling process. Models from different classes are generated and selected according to objective criteria.

Model refinement is necessary in order to obtain models with a good explanation of the studied phenomenon and, the same time, with a small degree of complexity, which makes them easy to interpret and with little data collection effort.

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. When using nonlinear model generators based on combinatorial algorithms, all model structures based on the initial parameters are built. When using genetic algorithms for model generation, the initial population of models evolves through generations. 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. The model built on the refined structure can be further subject for model refinement.

The experimental results presented in this paper, along with results obtained in [18] offer an opening to using modern refinement techniques and to further research.

## References

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

[2] Scott Culp. (2008). Definition of a Security Vulnerability. [Online] Available:

http://technet.microsoft.com/ro-ro/library/cc751383(en-us).aspx

[3] *Common Vulnerabilities and Exposures* [Online], Available: http://www.cve.mitre. org

[4] *Security Focus* [Online], Available: http://www.securityfocus.com/bid

[5] *The Open Source Vulnerability Database* [Online], Available: 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,

[8] I. Ivan and A. Vişoiu. Generator de structuri pentru modele economice şi sociale. *Revista Romana de Statistica. no. 4*, pp. 43 – 52

[9] A. Vişoiu and I. Ivan. Generator de modele liniare cu argument intarziat. *Revista de Comerţ. vol. 5, no.1*, pp. 47-50

[10] A. Vişoiu and G. Garais, “Nonlinear model structure generator for software metrics estimation,” in *Proc. 37th International Scientific Symposium of METRA*, Bucharest, Romania, 2006, published on CD

[11] A. Vişoiu, “Utilizarea algoritmilor genetici pentru rafinarea metricilor software”, *in Proc. Simpozionul Internaţional al Tinerilor Cercetători*, Chişinău, 2008

[12] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence, 2nd ed.*, Springer Publishing, 2006

[13] I. Ivan, A. Vişoiu, Rafinarea metricilor software, *Economistul*, *supliment Economie teoretică şi aplicativă. no.1947(2973)*, pp. III

[14] I. Ivan, A. Vişoiu, “A Comparative Analysis of Software Refinement Techniques,” *in Proc. Cybernetics and Information Technologies, Systems and Applications CITSA 2008*, Orlando, USA, 2008, pp. 235-239

[15] A. Vişoiu. (2007, March). Performance Criteria for Software Metrics Model Refinement. *Journal of Applied Quantitative Methods* [Online], *vol. 2, issue 1*, pp. 118-128. Available: http://www.jaqm.ro/ issues/volume-2,issue-1/ pdfs/visoiu.pdf

[16] *The Apache httpd project* [Online], Available: http://httpd.apache.org/

[17] *Apache httpd Security Report* [Online], Available: http://httpd.apache.org/security_report.html

[18] A. Vişoiu.(2008). Neural Network Based Model Refinement. *Informatica Economică Journal* [Online], *vol. 12, no. 1*. Available: http://revistaie.ase.ro/ content/45/2%20-%20Adrian%20Visoiu.pdf

**Adrian VIŞOIU** 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 a PhD student of the Doctoral School of Bucharest Academy of Economic Studies in the field of Economic Informatics. He is an assistant lecturer in the Economic Informatics Department of the Bucharest Academy of Economic Studies and PhD candidate. 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.