Building a Genetically Engineerable Evolvable Program (GEEP) Using Breadth-Based Explicit Knowledge for Predicting Software Defects[*]

K. Kaminsky
Department of Computer Science
University of Houston – Clear Lake
Houston, Texas 77058, USA
kchatfield@houston.rr.com

G. Boetticher
Department of Computer Science
University of Houston – Clear Lake
Houston, Texas 77058, USA
boetticher@cl.uh.edu

Abstract – **There has been extensive research in the area of data mining over the last decade, but relatively little research in algorithmic mining. Some researchers shun the idea of incorporating explicit knowledge with a Genetic Program environment. At best, very domain specific knowledge is hard wired into the GP modeling process. This work proposes a new approach called the Genetically Engineerable Evolvable Program (GEEP). In this approach, explicit knowledge is made available to the GP. It is considered breadth-based, in that all pieces of knowledge are independent of each other. Several experiments are performed on a NASA-based data set using established equations from other researchers in order to predict software defects. All results are statistically validated.**

I. Introduction

Genetic Programming (GP) supports automatic programming by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations. They have demonstrated some of their potential by evolving programs for medical signal filters [1], modeling complex chemical reactions [2], performing optical character recognition [3], target identification [4], and routing of analog electrical circuits [5].

Historically, incorporating explicit knowledge within a GP is not an accepted practice. Gero and Kasakov [6] claim that genetic programs are "knowledge lean machines; they make no use of knowledge in their execution" where this approach supposedly gives GP "robustness and breadth of applicability." Angeline [7] discourages the use of explicit knowledge, and argues that it should be used as "a last resort." He [7] claims, "including explicit knowledge for a particular problem removes the opportunity for contradictory knowledge to emerge." Thus, the historical approach forces all genetic programs to reinvent many existing algorithms.

Considering the decades worth of legacy data, the decades worth of software repositories, and the current rate of data propagation (2 exabytes per year), the GP modeling process would benefit immensely through the use of explicit knowledge in the form of algorithms. Lately, GP modeling has recognized the importance of explicit knowledge in GP modeling. The research literature contains examples of GP modeling using financial domain knowledge in GP modeling [8, 9, 10].

These successes demonstrate the importance of explicit domain knowledge with GP modeling. However, they tend to bundle the domain knowledge within the GP model. As a consequence, it is difficult to create dynamic GP models.

An alternative approach is to provide explicit (domain) knowledge, which is not bundled to the genetic programs, producing "knowledge-wise machines." This new technique is referred to as a **Genetically Engineerable Evolvable Program (GEEP).** Unbundling the explicit domain knowledge from the GP model enables the examination of the relationship between domain knowledge and GP model formation.

Predicting software defects is one of the most significant research areas in the Software Engineering community. It enables software practitioners to detect, track and resolve product anomalies that might affect human safety and lives. Additionally, defect prediction allows changes to be made earlier in the life-cycle process, thus lowering software costs and improving customer satisfaction.

Considering these benefits, many organizations tend to under-appropriate resources to their software quality group. Many theories could be postulated regarding the reasons for this situation.

This paper extends previous research in the area of defect prediction in two ways. It explores potential enhancements to previous defect models proposed by incorporated algorithms developed by various researchers in the Software Engineering community. Also, this paper explores potential synergy amongst these algorithms. Since the proposed algorithms are independent of each other it is viewed as breadth-based knowledge. Depth-based knowledge means that one algorithm is dependent upon a second algorithm.

This paper is organized as follows: section 2 provides an overview regarding the research in using machine learners to predict software defects; section 3 describes a NASA data set that will be used for performing the experiments; section 4 presents legacy defect prediction algorithms; section 5

[*] This work is partially supported by a Grant from the Institute for Space Systems Operations.

explains the GEEP approach. Based upon the groundwork set in sections 3 through 5, section 6 describes a series of experiments using explicit breadth-based knowledge along with the corresponding results; section 7 discusses these experiments. Finally, sections 8 and 9 offer implications and future directions.

II. Related Research

The literature abounds with defect prediction models. Despite the efforts of various researchers, there seems to be little consensus regarding what to measure when formulating defect prediction models [11].

One approach to address this problem is the application of expert systems and machine learners [11…16] in formulating a predictor. Such an approach is plausible since expert systems and machine learners handle noisy data and uncertainty rather well.

Fenton [11] develops a Bayesian Belief Network tool called AID, (Assess, Improve, Decide). He tests AID on 28 software projects from the Philips Software Centre. The results of this study are promising, however a great deal of data is required to train the network [17].

The application of Neural Networks to the problem of defect detection has received a great deal of attention. Hochmann extends the use of Neural Networks to software defects [14]. Thirty classification models are built with an equal distribution of fault-prone and non-fault-prone software modules. He compares an Evolutionary Neural Network, (ENN) to Discriminate Analysis. The experimental findings indicate that the error rates for the ENNs are statistically much lower than for the Discriminate Analysis [14].

Cohen and Devanbu [12] apply different Inductive Learning Programs (FOIL, FLIPPER, RIPPER) in predicting fault density for 122 C++ program examples. The authors use coupling metrics to formulate their models. They deploy various modeling strategies including suppression of negative clauses (monotonic information only) along with the implementation of aggregate operators. The experiments generate error rates ranging from 19.7 through 35.4 percent.

Evett et al. [13] apply Genetic Programs against several industrial-level repositories in predicting software faults. Their operator set includes *add*, *subtract*, *multiply*, *divide*, *sine*, *cosine*, *exponentiation*, and *logarithms*. The operands consist of various product metrics including *Halstead's vocabulary*, *McCabe's cyclomatic complexity*, and *Lines of Code (LOC)*. The author's assess their GP models by ranking data sets according to defect counts and comparing the top *n* percent of actual and predicted models where *n* ranges from 75 to 90 percent.

III. NASA's Project Data

The data for the machine learning experiments originate from a NASA project, which will be referred to as "KC2." KC2 is a collection of C++ program containing over 3000 "c" functions. The analysis focuses only on those functions created by NASA developers. This means COTS-based metrics are pruned from the data set. After eliminating redundant data, the final tally consists of metrics from 379 "c" functions.

The KC2 data set contains twenty-one software product metrics based on the product's size, complexity and vocabulary. The size metrics include *total lines of code*, *executable lines of code*, *lines of comments*, *blank lines*, *number of lines containing both code and comments*, and *branch count*. Another three metrics are based on the product's complexity. These include *cyclomatic complexity*, *essential complexity*, and *module design complexity*. The other twelve metrics are vocabulary metrics. The vocabulary metrics include *Halstead length*, *Halstead volume*, *Halstead level*, *Halstead difficulty*, *Halstead intelligent content*, *Halstead programming effort*, *Halstead error estimate*, *Halstead programming time*, *number of unique operators*, *number of unique operands*, *total operators*, and *total operands*.

The KC2 data set also contains the defect count for each module. The majority of the defect values range from 0 to 2. There are 272 instances of zero defects in the modules, 56 instances of one defect, and 25 instances of two defects. Of the remaining 24 module instances, nine have three defects, four have four defects, five have five defects, two have six defects, one has eight defects, one has ten defects, one has eleven defects, and one has thirteen defects. Thus, ninety percent of the defect data is concentrated in 27 percent of the defect value points. The defect distribution is illustrated in Fig. 1.

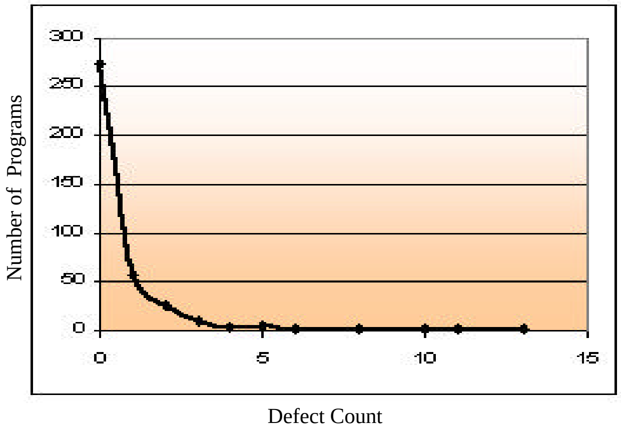

Fig. 1 Defect Frequency Distribution for KC2 Data Set

IV. Domain Knowledge: Defect Prediction Equations

The domain knowledge for the GP experiments consists of algorithms, in the form of equations, developed by researchers over the last several decades. This section describes several of these algorithms. For a critique of these algorithms, see [11].

Akiyama [18] provides one of the earliest studies in predicting software defects. In this study, two regression models (1) and (2) are formulated. Equation (1) uses lines of code, L, to predict defects.

$$D = 4.86 + 0.018 * L \tag{1}$$

Akiyama's second equation, as depicted in (2), utilizes a complexity metric, C, to predict defects.

$$D = 0.12 * C - 0.84 \tag{2}$$

Halstead [19] proposes a defect prediction model using a combination of vocabulary metrics. Equation (3) depicts his formula.

$$D = Volume / 3{,}000 \quad (3)$$

The *Volume* metric is based on the number of operators and operands within a program module. *Volume* corresponds to $N*log_2(n)$, where N is the *total number of operators* and *operands* in a module and n is the *unique number of operands* and *operands* in a module.

Lipow [20] formulates an equation (4) based on Halstead's theory.

$$D/L = A_0 + A_1*ln*L + A_2*ln^2*L \quad (4)$$

Each A_i depends on the average *number of operands* and *operators* per lines of code for a particular programming language. *L* represents *lines of code*.

Lipow provides a table with the values for A_i if the average number of operators and operands are known for the programming language. The average number of operators and operands for C++ is unknown. Therefore, for the purposes of this study, the number of operands and operators was calculated from the KC2 data set. KC2 also lists the number of lines of code. The number of operators and operands for this data set is 48,424. The total lines of code equal 18,556. Therefore, the average number of operators and operands for this data set is 2.61. The closest value Lipow provides is 2.5, so this is the index used to obtain the values for A_i. Therefore the equation used for the fourth experiment will be (5):

$$D = L * (0.000844 + (0.0007842)lnL + (0.00001546)ln^2L) \quad (5)$$

Gaffney [21] claims that the relationship between *D* and *L* are not language dependent. He proposes the following equation (6):

$$D = 4.2 + 0.0015*(L)^{4/3} \quad (6)$$

Based on the same premise, Compton [22] derives the following equation (7):

$$D = 0.069 + 0.00156*L + 0.00000047*L^2 \quad (7)$$

V. A GENETICALLY ENGINEERABLE EVOLVABLE PROGRAM (GEEP)

Genetic Programs represent solutions, called chromosomes, as tree structures of variable length. Normally these tree structures consist of a set of simple mathematical operators and corresponding operands.

Some of the more advanced GP models may include additional functions and/or procedures in the tree structure. However, these GP models are designed for a specific domain and the functions/procedures are "hard wired" into the GP.

The GEEP extends the traditional GP approach by supporting the use of functions and/or procedures which are not bundled with the GP. These functions/procedures may be domain specific or domain independent. Fig. 2 illustrates the usage of domain specific knowledge, in this case Trigonometry, in developing a GEEP-based solution.

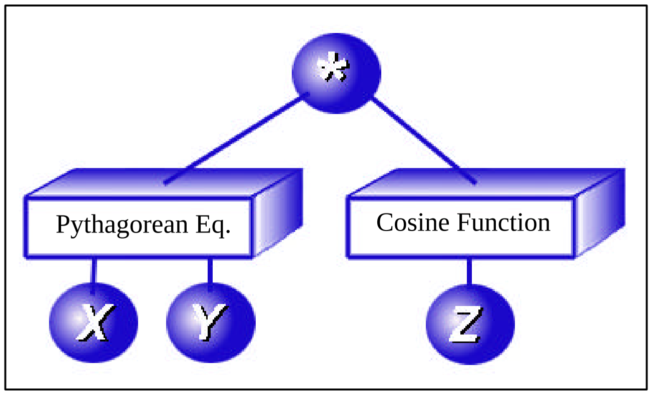

Fig. 2 Using Domain Specific Knowledge to Build Models

This proposed research holds the promise of producing a more sophisticated and accurate solution in less time by exploiting what is already known.

What distinguishes the GEEP approach from tradition GP modeling is the ability to seamlessly incorporate domain knowledge, in the form of procedures and functions, from various domains into the modeling process. This allows a GEEP to leverage existing domain knowledge and build more sophisticated models. As this research matures, it will spawn further research regarding domain analysis within GP. It may be possible to have a user direct which domain knowledge to include/exclude in the modeling process. Fig. 3 depicts a future architecture scenario for the GEEP model.

Thus there may be a vast repository of algorithmic knowledge, collected from multiple domains, available for the GEEP. The GEEP will be capable of building very sophisticated solutions. One potential benefit is the ability to reuse algorithms in domains in which they were not intended. Furthermore, as additional optimization techniques are added to the process, it will be possible to reduce solutions to their simplest form. This will promote better human-readable solutions.

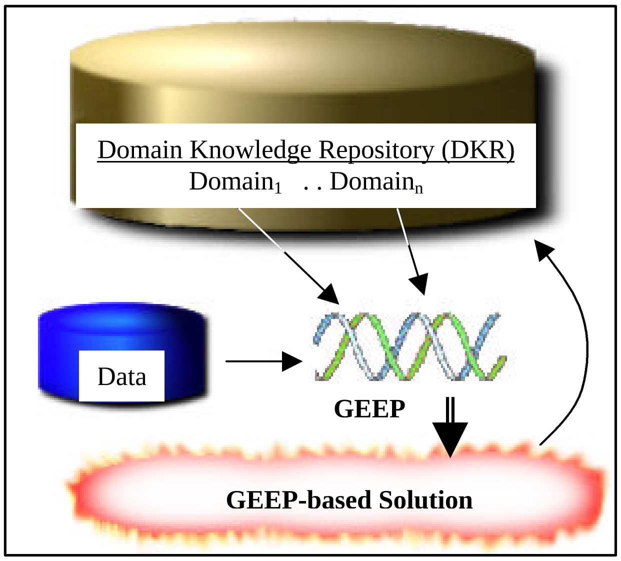

Fig. 3 GEEP Solution = DKR + GP + Data

VI. THE GEEP EXPERIMENTS

The goals of the GEEP experiments is to determine whether using explicit algorithmic knowledge improves upon the software defect prediction models produced and conversely, whether it is possible to improve upon one or more of the defined legacy algorithms.

A. *Equation Utility Against the NASA Data Set*

Using explicit knowledge in building a GP implies a certain "goodness" regarding the actual equations. In order to evaluate the usefulness of integrating knowledge into the Genetic Program, the utility of each of the equations must be applied to the data set. This is accomplished by determining the average error over the KC2 data set for each equation. Akiyama's first equation (1) generates an error rate of 474 percent. The second equation by Akiyama (2) produces an error rate of 53.33 percent. The error rate for Halstead's (3) equation is 24.83 percent. Lipow's equation (5) results in an error rate of 24.62 percent. Gaffney's (6) equation generates an error rate of 376 percent. Finally, Compton's (7) error rate is 25.79 percent. From these error rates, it may be deduced that Akiyama and Gaffney's equation may not add much value to the Genetic Program, while Halstead, Lipow and Compton's equations have the potential to improve performance.

B. *General Experimental Settings*

The GEEP experiments run for a maximum of 128 generations with an initial maximum tree height of six. The fitness function is equal to 1 minus the standard error. Each generation contains 512 chromosomes. Six experiments are performed consisting of thirty trials each. The first five experiments compare a vanilla-based Genetic Program, using simple mathematical operators against the KC2 data against a GEEP that is enhanced with equations from Akiyama, Halstead, Lipow, Gaffney, and Compton respectively. The last experiment extends the GEEP with all the algorithms simultaneously.

C. *Experiments*

The first experiment compares a GP using simple mathematical operators against a GEEP which includes Akiyama's equations (1) and (2). Table I shows an improvement in the GP enriched with the Akiyama equations. The one-tail t-Test result is under the alpha value of 0.05, indicating that the difference is statistically significant.

TABLE I
AKIYAMA T-TEST ON FITNESS VALUES

	Basic	**Akiyama**
Mean	0.7523	0.7583
Variance	0.0001	7.9410E-05
Observations	30	30
Pearson Correlation	-0.0179	
df	29	
t Stat	-2.2255	
P (T <= t) one-tail	0.0169	

The second experiment compares a GP with a GEEP enhanced with Halstead's equation. Table II shows the results of this experiment. The basic operators produce an average fitness value of 75.23 percent and the models that utilize the Halstead equation produce an average fitness value of 76.33 percent. A one-tailed t-Test, where alpha is 0.05, indicates that the results are statistically significant.

TABLE II
HALSTEAD T-TEST ON FITNESS VALUES

	Basic	**Halstead**
Mean	0.7523	0.7633
Variance	0.0001	0.0001
Observations	30	30
Pearson Correlation	-0.1079	
df	29	
t Stat	-3.5531	
P (T <= t) one-tail	0.0006	

The third experiment assesses the GEEP enhanced with Lipow's equation (5) against a basic GP model. Table III shows an improvement in the GP enriched with the Lipow equation versus the basic GP model. The basic operators produced an average fitness value of 75.23 percent and the trials using the Lipow equation produce an average fitness value of 76.45 percent. The one-tail t-Test result is under the alpha value of 0.05, indicating that the difference is statistically significant.

TABLE III
LIPOW T-TEST ON FITNESS VALUES

	Basic	**Lipow**
Mean	0.7523	0.7645
Variance	0.0001	1.3967E-05
Observations	30	30
Pearson Correlation	-0.1718	
df	29	
t Stat	-5.1647	
P (T <= t) one-tail	8.0230E-06	

The fourth experiment assesses the GEEP enhanced with Gaffney's equation (6). Table IV shows an improvement in the GEEP enriched with the Gaffney equation versus the basic GP. The basic operators produce an average fitness value of 75.23 percent while the Gaffney-based GEEP produce an average fitness value of 75.73 percent. The one-tail t-Test result is under the alpha value of 0.05, once again indicating that the difference is statistically significant.

TABLE IV
GAFFNEY T-TEST ON FITNESS VALUES

	Basic	Gaffney
Mean	0.7523	0.7573
Variance	0.0001	0.0001
Observations	30	30
Pearson Correlation	0.2641	
df	29	
t Stat	-2.0514	
P (T <= t) one-tail	0.0246	

The fifth experiment compares the basic GP against a GEEP that is enhanced with the Compton equation (7). The GP with the basic operators achieve an average fitness value of 75.23 percent and the GEEP enhanced with the Compton equation produces an average fitness value of 75.87 percent. The t-Test shows that the difference is statistically significant as shown in Table V.

TABLE V
COMPTON T-TEST ON FITNESS VALUES

	Basic	Compton
Mean	0.7523	0.7587
Variance	0.0001	0.0001
Observations	30	30
Pearson Correlation	0.8546	
Df	29	
t Stat	-5.7176	
P (T <= t) one-tail	1.733E-06	

The last experiment compares a GP with basic mathematical operators against a GEEP that includes equations (1) through (3) and (5) through (7). The GP with the basic operators produce an average fitness value of 75.23 percent while the GEEP enhanced all the explicit knowledge generates an average fitness value of 76.60 percent. The t-Test shows that the difference is statistically significant as shown in Table VI.

TABLE VI
ALL KNOWLEDGE T-TEST ON FITNESS VALUES

	Basic	All Knowledge
Mean	0.7523	0.7660
Variance	0.0001	5.4933E-05
Observations	30	30
Pearson Correlation	-0.0656	
df	29	
t Stat	-5.2278	
P (T <= t) one-tail	6.7335E-06	

VII. DISCUSSION

In all experiments the GEEP with the addition of explicit breadth-based knowledge produces statistically superior results than those models without access to any explicit knowledge.

It is interesting to note that the initial error rates generated for each equation against the KC2 data are inversely correlated with the average GEEP fitness values produced for each algorithm. As mentioned earlier, the fitness value equals 1 minus the standard error rate. Table VII confirms this finding. Perhaps calculating the initial error rate is a potential predictor of the utility of using a particular algorithm.

TABLE VII
RESULTS OF EXPERIMENTS IN DESCENDING ORDER BY ERROR RATE

Researcher's Algorithm	Equation Error Rate	Average GEEP Fitness Results
Lipow	24.62	0.7645
Halstead	24.83	0.7633
Compton	25.79	0.7587
Akiyama	53.33	0.7583
Gaffney	376.00	0.7573

The last experiment uses all the algorithms from the different researchers. These experiments produce the highest average fitness values of 0.7660. Applying a t-Test to the results from the model containing all the algorithms and the results from each individual algorithm, the "all algorithms" model was statistically superior to the Compton, Akiyama, and also Gaffney models, respectively. This certainly supports the argument of using explicit knowledge in the GP modeling process.

Equation 8 shows the model that generated the best result (*fitness = 0.7744*) using all the algorithms. Definitions of the operand terms are available from [23]. Although (8) is in human readable form, it is not necessarily human-understandable form.

((((Lipow((Compton((9 + 8) ^ (iv(g) - 9)) + d)) + (AkiyamaLoc((Compton(Gaffney(Lipow(9) ^ lOBlank)) ^ Lipow(b) ^ Lipow(((6 * AkiyamaLoc ((Gaffney((6 * AkiyamaLoc((Lipow(9)) + Lipow((Compton(AkiyamaComp(((Halstead((Compton((Halstead(Lipow(9) + AkiyamaLoc(AkiyamaLoc(l)) - Halstead(AkiyamaComp(iv(g)) * Halstead(Compton(AkiyamaComp(AkiyamaLoc(Compton(3)) ^ AkiyamaLoc(Halstead(AkiyamaComp(iv(g)) ^ Gaffney((lOComment ^ (Gaffney(Lipow(2)) ^ (Compton(6) ^ AkiyamaLoc((6 - 10)) + Compton(Gaffney(9) * Compton(Lipow(iv(g))) ^ Lipow(((6 * AkiyamaLoc((Gaffney(Lipow(((6 * AkiyamaLoc((Gaffney(Lipow((Compton(AkiyamaComp(AkiyamaLoc(Compton(AkiyamaLoc(l) ^ Gaffney(((Halstead((AkiyamaLoc(AkiyamaLoc((3 ^ lOBlank)) - 7)) * Halstead(AkiyamaComp(iv(g))^ (Gaffney(Lipow(2)) ^ (Compton(6) ^ AkiyamaLoc((6 - 10)) + Compton(3) ^ Gaffney(10))) + Compton(l) * Compton(l))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))) (8)

VIII. IMPLICATIONS

The aforementioned experiments demonstrate the feasibility of leveraging explicit knowledge in building a GP model. Considering the decade's worth of algorithms residing in repositories, it seems reasonable to place greater emphasis on algorithm mining within the data mining process.

This could lead to greater reuse of legacy algorithms in domains in which they were not intended; more accurate solutions; and the discovery of potential synergistic relationships among different domains.

IX. FUTURE DIRECTIONS

Future directions for this research include the following activities: apply the GEEP solutions against other data sets available from NASA. This would validate the models produced and give further credibility to this approach.

At times, the experimentation process would take several hours to complete 30 trials. A performance enhancement would be to build the GEEP using distributed processing environment.

Finally, equation (8) shows that potential solutions can be somewhat cumbersome. To make solutions more human-understandable, it is anticipated that an equation optimizer will be included in the modeling process.

REFERENCES

[1] Oakley, Howard, Two scientific applications of genetic programming: Stack filters and non-linear equation fitting to chaotic data. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 17, pages 369--389. MIT Press, 1994.

[2] Bettenhausen, K.D., S. Gehlen, P. Marenbach, and H. Tolle. BioX++ -- New results and conceptions concerning the intelligent control of biotechnological processes. In A. Munack and K. Schügerl, editors, *6th International Conference on Computer Applications in Biotechnology*, pages 324--327. Elsevier Science, 1995.

[3] Andre, David. Learning and upgrading rules for an OCR system using genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[4] Tackett, Walter Alden. Genetic programming for feature discovery and image discrimination. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303--309, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[5] Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 2000. Automatic design of analog electrical circuits using genetic programming In Cartwright, Hugh (editor). *Intelligent Data Analysis in Science*. Oxford: Oxford University Press. Chapter 8. Pages 172 - 202.

[6] Gero, JS and Kazakov, V, A genetic engineering extension to genetic algorithms, *Evolutionary Systems* 9(1): 71-92, 2001.

[7] Angeline, Peter John, Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4, pages 75--98. MIT Press, 1994.

[8] Dempster M A H and Jones CM. "*A real-time adaptive trading system using genetic programming*", Quantitative Finance Vol. 1, institute of Physics Publishing, (2001), 397- 413.

[9] Frick, et al., Genetic-Based Trading Rules - A New Tool to Beat the Market With -- First Empirical Results --, in Aktuarielle Ansätze für Finanz-Risiken, Proceedings of 6th International AFIR Colloquium, Nürnberg, 1.-3. October 1996, (Editor Pete Albrecht) Verlag Versicherungswirtschaft e.V. Karlsruhe, Volume I/II, pp. 997 - 1018 (with coauthors A. Frick, R. Herrmann, M. Kreidler and A. Narr).

[10] Mahfoud, Sam, and Ganesh Mani, Financial Forecasting using Genetic Algorithms, Applied Artificial Intelligence, 10:543-565, 1996.

[11] Fenton, Norman E. *A Critique of Software Defect Prediction Models.* IEEE Transactions on Software Engineering, Vol. 25, No 3, May/June 1999.

[12] Cohen, William W., Devanbu, Prem., *A Comparative Study of Inductive Logic Programming Methods for Software Fault Prediction*, Proc. 14th International Conference on Machine Learning., 1997.

[13] Evett, Matthew., Khoshgoftar, Taghi., *GP-based Software Quality Prediction.*, Genetic Programming 1998: Proceedings of the Third Annual Conference, 1998.

[14] Hochmann, J.P., Hudepohl, E.B., Allen, T.M., Khoshgoftaar, R. *Evolutionary Neural Networks: A Robust Approach to Software Reliability*, Eighth International Symposium on Software Reliability Engineering (ISSRE '97), pages 13-26, 1997.

[15] Neumann, Donald E., *An Enhanced Neural Network Technique for Software Risk Analysis*, IEEE Transactions on Software Engineering, pages 904-912, 2002.

[16] Zhang, Du. *Applying Machine Learning Algorithms in Software Development*, Proceedings of the 2000 Monterey Workshop on Modeling Software System Structures in a Fastly Moving Scenario, pages 275-291, 2000.

[17] Fenton, Norman E., Krause, Paul., Neil, Martin. *A Probabilistic Model for Software Defect Prediction*, for submission to IEEE Transactions in Software Engineering. 2001.

[18] Akiyama, F., "An Example of Software System Debugging," *Information Processing*, vol. 71, pp. 353-379, 1971.

[19] M.H. Halstead, *Elements of Software Science*. Elsevier North-Holland, 1975.

[20] M. Lipow, "Number of Faults per Line of Code," *IEEE Trans. Software Eng.*, vol. 8, no. 4, pp. 437-439, 1982.

[21] Gaffney, J.R., "Estimating the Number of Faults in Code," *IEEE Trans. Software Eng.*, vol. 10, no. 4, 1984.

[22] T. Compton, and C. Withrow, "Prediction and Control of Ada Software Defects," *J. Systems and Software*, vol. 12, pp. 199-207, 1990.

[23] Metrics Data Program, NASA IV&V facility. Information Data sets are available at: http://mdp.ivv.nasa.gov/about.html