Predicting Defects in Software Using Grammar-Guided Genetic Programming

Athanasios Tsakonas and Georgios Dounias

University of the Aegean, Department of Finance and Management Engineering,
Fostini 31 str., Chios, Greece
tsakonas@stt.aegean.gr, g.dounias@aegean.gr

Abstract. The knowledge of the software quality can allow an organization to allocate the needed resources for the code maintenance. Maintaining the software is considered as a high cost factor for most organizations. Consequently, there is need to assess software modules in respect of defects that will arise. Addressing the prediction of software defects by means of computational intelligence has only recently become evident. In this paper, we investigate the capability of the genetic programming approach for producing solution composed of decision rules. We applied the model into four software engineering databases of NASA. The overall performance of this system denotes its competitiveness as compared with past methodologies, and is shown capable of producing simple, highly accurate, tangible rules.

Keywords: Software engineering, defect prediction, genetic programming.

1 Introduction

Addressing software quality can ensure cost reduction and efficient resource allocation. A major factor for the assessment of software code is whether the code module is prone to defects in the future. To estimate the software quality several metrics have been developed in the past. Static code metrics [9],[6] are inexpensive, easy to calculate, and they are widely used. However, these measurements have been criticized [4][5][16] on their effectiveness and efficiency, as standalone instruments. Later work [11], has shown that applying data mining techniques can dramatically increase the power of the aforementioned metrics. The main target of such a data-mining task is to effectively predict whether modules will present code defects in the future, so as the management could efficiently allocate resources for monitoring them. Genetic programming (GP) [7] is a computational intelligence methodology which carries expedient attributes such as variable length solution representation, and functional solution nodes. It has been applied in numerous problems, and its domains of applications are constantly increasing. This work inherits recent advances on genetic operators' adaptive rates [18]. The data mining task in this work is two-fold. Firstly, we aim to produce simple and comprehensible rules that can be used without the assistance of software. Secondly, we seek for high classification rates, if possible better to those found in literature. The paper is organized as follows. Next section

J. Darzentas et al. (Eds.): SETN 2008, LNAI 5138, pp. 413–418, 2008.

describes the background, presenting the software defects prediction domain and the grammar guided genetic programming. Following this section, we deal with the design and the implementation of the system. Next, the results and a following discussion are presented. The paper ends with our conclusion and a description of future work.

2 Background

2.1 Software Defects Prediction

Among the principal tasks during software project management is the assessment of the software cost. Additionally, extensive assessment is required for high assurance software. This software cost is affected directly by the software quality. To address this need, there have been developed various techniques of software code assessment, such as the static code metrics. The available metrics for the code derive from the work of [9] and [6].

2.2 Grammar-Guided Genetic Programming

Genetic programming [7] is an extension to the genetic algorithms concept. The main advance is the ability to maintain a population consisted of variable-length, tree-structured individuals, in which each node can have functional ability. By applying grammars, a genotype - a point in the search space- corresponds always to a phenotype - a point in the solution space, an approach known as legal search space handling method [9]. We apply legal search in this work using a context-free grammar [2][3][8][13][17][18][19].

3 Design and Implementation

3.1 Data Pre-processing

We have tested the methodology in four software engineering data sets: CM1, KC1, KC2 and PC1. These datasets have been addressed in [11] and [12]. All software modules come from NASA and their metrics have become recently available by the PROMISE repository of public domain software engineering data sets. Table 1 summarizes the features of this data. Further details for each feature can be found in [9] and [6].

3.2 Genetic Programming Setup

To improve the search process and control the solution size, an adaptive scheme for the operation rates was followed. These parameters were adapted from past work of the authors [18], and they do not necessarily represent the best values for these datasets.

Table 1. NASA software metrics data examined

Name	Data set	Total instances	Defects	No defects	Language
CM1	Spacecraft instrument	498	49	449	C
KC1	Storage management for ground data	2109	326	1783	C++
KC2	Science data processing	552	105	415	C++
PC1	Flight software for earth orbiting satellite	1109	1032	77	C

During the run, the actual training data set is used to evaluate candidate solutions. However, in order to promote a candidate as the solution of the run, in our approach it is required that at least one of the following conditions applies:

- this candidate achieves higher fitness score in the validation set too,
- the absolute difference between validation fitness and training fitness score is smaller.

The first rule is the common approach used in all validation models; the second rule is introduced in this work, and it was experimentally observed to produce solutions that carried significantly higher generalization ability in the problems encountered. In other words, this approach promotes solutions that demonstrate no overfitting to one of the sets (either the actual training set or the validation set), but it rather requires the fitness improvement in one set to be in step with the other [14].

3.3 Fitness Function

In order to validate this software engineering data, various measures have been proposed in literature. In [1], the following measures have been used, in a genetic programming model for a number of generic problems encountered:

$$\text{Recall} = pd = \frac{tp}{tp + fn} \tag{1}$$

$$TNRate = \frac{tn}{tn + fp} \tag{2}$$

$$fitness = \text{support} = \text{Recall} \cdot TNRate \tag{3}$$

In [10] the fitness measure that involves the *accuracy*, is proposed based on results that show that this metric presented the smaller deviation in classification success between the training and the test set, for a number of experiments. On the other hand, in [1], when using the *Recall* and the *TNRate*, there is an equivalent treatment for both classes as far as the classification reward is concerned, irrespectively of the relative size for each class. Hence we adopted the latter measure for our fitness function. Additionally, another metric is calculated in our experiments, *precision*, to allow future comparisons:

$$prec = \frac{tp}{fp + tp} \tag{4}$$

This *precision* measure is analogous to the *support* measure we have used in our system, as it can be seen in the equation (5). Hence, using the *support* measure as a fitness measure is also in concordance to literature that requires a system scoring also high *precision* values (i.e. aiming for high *support* values can assist in qualifying high *precision* rates).

$$\frac{1}{\text{support}} = \left(\frac{1}{prec} - 1\right) \cdot \frac{fn}{tn} + \frac{fn}{tp} + \frac{fp}{tn} + 1 \tag{5}$$

Having discussed the system design, in the following session we describe the results of the application of our methodology in the four software engineering databases.

4 Results and Discussion

We performed 10-fold cross validation. Table 2 summarizes our best results found for each measure during the 10-fold run, in the test set, and includes the mean and the standard deviation of these results. The solution for the CM1 problem is as follows:

```
If count_of_lines_of _comments > -0.94 then true else false
```

The promoted solution for the KC1 data is:

```
If  essential_complexity < 0.76 then
        (if total_operands < -0.95 then false else true)
else false
```

For the KC2 problem, the system derived the following rule:

```
If design_complexity = 0.46 then
        (if line_count_of_code < 0.95 then false else true )else

        (if unique_operands > -0.90 then true else false)
```

Finally, the following rule was found for the PC1 data:

```
If program_length < -0.53 then
        (if difficulty > -0.21 then
        (if total_operators > -0.68 then true else false)
        else
(if software_size_lines_of_code < -0.93 then true
 else false))
 else false
```

In all problems, our model succeeded in producing small, easily comprehensible results that need not any further software to be applied in practice. In Table 3, we compare the best-promoted solutions of our system during the 10-fold validation, to the best models found in literature.

5 Conclusion and Further Research

This paper described an effort to address the software quality domain, by using computational intelligence for effective decision-making. Our approach makes use of the genetic programming paradigm, in its grammar-guided advance, in order to produce decision rules. Further tuning is enforced to the genetic operators, and special use of the validation set fitness is applied. The model is applied on four databases that are consisted of software metrics of NASA's developed software. In two of the databases, our model is proved superior to the existing literature in both comparison variables, and in the rest two databases, the system is shown better in one of the two variables.

Table 2. Grammar-guided GP, 10-Fold Cross Validation Results

		CM1			*KC1*	
	Best	Mean	Std.Dev	Best	Mean	Std.Dev
Support	0.7085	0.5982	0.0538	0.5731	0.5579	0.0107
PD	1.0000	0.5967	0.2344	0.8750	0.7544	0.0935
PF	0.2000	0.2719	0.0724	0.2569	0.3135	0.0399
PREC	0.3077	0.1905	0.0586	0.3553	0.3062	0.0312
Accuracy	0.8750	0.7295	0.0768	0.7393	0.6967	0.0309
Generation		38	60		55	77
Size		191	224		259	196
		KC2			*PC1*	
	Best	Mean	Std.Dev	Best	Mean	Std.Dev
Support	0.7127	0.6697	0.0304	0.7508	0.6442	0.0548
PD	0.8182	0.7482	0.0594	0.8750	0.7441	0.0615
PF	0.1428	0.1929	0.0400	0.0000	0.2911	0.2301
PREC	0.5714	0.5039	0.0488	1.0000	0.9728	0.0207
Accuracy	0.8302	0.7830	0.0336	0.8559	0.7414	0.0547
Generation		30	36		82	62
Size		260	203		296	201

Table 3. Results comparison

Model	PD	PF		PD	PF
	CM1			*PC1*	
Menzies et al. [11]	0.350	*0.100*		0.240	0.240
Menzies et al. [12]	0.710	0.270		0.480	0.170
This paper (#8)	*1.000*	0.311	(#3)	*0.757*	*0.125*
	KC1			*KC2*	
Menzies et al. [11]	0.500	*0.150*		0.450	0.150
This paper (#6)	*0.818*	0.275	(#7)	*0.800*	*0.142*

Moreover, the system managed to produce small and comprehensible solutions that do not require a computing environment to apply. The application of our system to such data is a straightforward process, and adds little complexity to the classification task of the modules. Hence we believe that software engineers can easily adapt such a data mining system, which can then be used in an inexpensive way, combined with the static metrics calculation. Further investigation involves the application of our methodology into more software quality problems, involving other databases, in an attempt to provide a transparent view on its effectiveness for this class of problems.

References

1. Berlanga, F.J., del Jesus, M.J., Herrera, F.: Learning compact fuzzy rule-based classification systems with genetic programming. In: 4th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2005), Barcelona, pp. 1027–1032 (2005)
2. Blickle, T., Theile, L.: A mathematical analysis of tournament selection. In: Eshelman, L.J. (ed.) Proc. of the 6thInternational.Conference on Genetic Algorithms, pp. 9–16. Lawrence Erlbaum Associates, Hillsdale (1995)
3. Eads, D., Hill, D., Davis, S., Perkins, S., Ma, J., Porter, R., Theiler, J.: Genetic Algorithms and Support Vector Machines for Time Series Classification. In: Proc. SPIE, vol. 4787, pp. 74–85 (2002)
4. Fenton, N., Ohlsson, N.: Quantitative Analysis of Faults and Failures in a Complex Software System. IEEE Trans. Software Eng., 797–814 (2000)
5. Fenton, N.E., Pfleeger, S.: Software Metrics: A Rigorous and Practical Approach. Int'l Thompson Press (1997)
6. Halstead, M.: Elements of Software Science. Elsevier, Amsterdam (1977)
7. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
8. Koza, J., Bennett, F., Andre, D., Keane, M.: Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis. Morgan Kaufmann, San Francisco (2003)
9. McCabe, T.: A Complexity Measure. IEEE Trans. Software Eng. 4, 308–320 (1976)
10. Menzies, T., Dekhtyar, A., Distefano, J., Greenwald, J.: Problems with Precision: A Response to Comments on; Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Trans. on Soft. Eng. 33(9), 637–640 (2007)
11. Menzies, T., DiStefano, J., Orrego, A., Chapman, R.: Assessing Predictors of Software Defects. In: Proc. Workshop Predictive Software Models (2004)
12. Menzies, T., Greenwald, J., Frank, A.: Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Trans. on Soft. Eng. 32(11) (January 2007)
13. Montana, D.J.: Strongly Typed Genetic Programming. Evolutionary Computation 3(2) (1995)
14. Quinlan, J.R.: Bagging, boosting, and C4.5. In: Proc. 13th Nat. Conf. Art. Intell., pp. 725–730 (1996)
15. Singleton, A.: Genetic Programming with C++. BYTE Magazine (February 1994)
16. Shepperd, M., Ince, D.: A Critique of Three Metrics. J. Systems and Software 26(3), 197–210 (1994)
17. Tsakonas, A., Dounias, G.: Hierarchical Classification Trees Using Type-Constrained Genetic Programming. In: Proc. of 1st Intl. IEEE Symposium in Intelligent Systems, Varna, Bulgaria (2002)
18. Tsakonas, A., Dounias, G.: Evolving Neural-Symbolic Systems Guided by Adaptive Training Schemes: Applications in Finance. App. Art. Intell. 21(7), 681–706 (2007)
19. Yu, T., Bentley, P.: Methods to Evolve Legal Phenotypes. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 280–291. Springer, Heidelberg (1998)