Effective Estimation of Modules' Metrics in Software Defect Prediction

S.M. Fakhrahmad, A.Sami

***Abstract*—The prediction of software defects has recently attracted the attention of software quality researchers. Many predictive classification systems have already been proposed, which aim at early discovery of software modules that are fault-prone and versa. The proposed methods are usually assessed using datasets available from NASA Metrics Data repository. These datasets include a combination of design-level and code-level metrics for different modules. To apply a defect predictor, all metrics have to be measured for any of the modules (to be used as the classifier inputs). The measurement of some of these metrics is easy and can be done straight forward. However, there are a number of metrics which are more difficult or time-consuming to quantify. Moreover, many of them do not have an exact value; so, they may get different values when using different formulas or tools. In this paper, we first discuss this hypothesis that some strong dependencies exist among various features of these datasets. Based on this hypothesis, we search for short combinations of features from the first category (easy-to-measure features), which can describe any of the features from the second category (hard-to-measure features) with a high accuracy. Then, we introduce a set of fuzzy modeling systems, each of which estimates the value of one of the second category features from its specified determinants. The evaluation of the estimation systems is carried out by computing the MSE values for all features. The experimental results are promising. The presented estimation system provides usability of the defect prediction system rather than its accuracy. Using this system, the user will not have to measure all the required mentioned metrics for any of the modules. All the features of the second category will automatically be estimated with a high accuracy.**

***Index Terms*— Software defect prediction, Fuzzy Modeling, Fuzzy Classification, Parameter Estimation, Approximate Dependencies**

I. Introduction

Quality of a software system is relative to the number of defects reported in the final product. Early discovery of software errors is very important and may cause significant cost savings, especially for large and complex systems. Software defect prediction is the task of classifying software modules into fault-prone (fp) and non-fault-prone (nfp) by means of metric-based classification [1, 2].

S. M. Fakhrahmad is Faculty member in the department of computer engineering, Islamic Azad University of Shiraz, and phD student in Shiraz University, Iran; e-mail: mfakhrahmad@ cse.shirazu.ac.ir.

A. Sami is Assistant Professor in the department of computer science and engineering, Shiraz University, Iran; e-mail: asami@ieee.org.

Software testing is the most expensive and time consuming issue in the process of software development. It usually requires about 50% of the whole project schedule. On the other hand, it has been proved by experience that the majority of a system's faults exist in a small fraction of modules. Thus, efficient prediction models are helpful for software testing. Accurate estimates of defective modules may help software developers in terms of allocating the limited resources and thus, decreasing testing times [3, 4].

During the past decade, several classification systems have been proposed, which perform predictive modeling efforts for detection of modules that are likely to contain faults. The evaluation of such systems has almost been carried out using a set of datasets available from NASA MDP repository [5]. Recently, via a set of experiments on NASA datasets, Lessman et al. [6] concluded and reported that there is not a high gap between predictive accuracies of different classification methods. In other words, even simple classifiers are able to classify software modules according to code attributes with a good accuracy. This is maybe due to the nature of majority of data sets which have been observed to be linearly separable.

Each of the NASA MDP datasets is related to one of the NASA projects and contains several modules. Each module is described by a set of code-level and design-level attributes. All discovered faults of the system are also registered in each dataset, together with the number of module containing the fault. Hence, there are two categories of modules; the modules with 1 or more errors (nfp modules) and those containing no fault (fp modules).

The number of module metrics in different NASA datasets varies from 21 to 43 metrics. To apply a defect prediction system to detect fp and nfp modules of a real software system, we have to measure all of these metrics for any of the modules (to be used as the classifier inputs).

The measurement of some of these metrics is easy and can be done straight forward. For instance, LOC metrics (such as *LOC-Comments* which represents the no. of code lines containing comments) and some other attributes can be easily measured. The measurement of such metrics can be handled manually or automatically.

On the other hand, some of the metrics are more difficult or time-consuming to measure. The main challenging metrics are the design metrics (e.g., complexity metrics) which require availability of design phase artifacts and design diagrams such

as DFDs, control flow graphs, Formal Description Language (FDL) graphs and UML diagrams, to be extracted from.

Moreover, some of the metrics can not be quantified easily and directly. Different formulas and tools have been proposed to estimate these features. Many of the formulas relate to some specific applications or processes. Some others have been defined to be used for some specific programming languages. The other challenge of this kind of metrics is that they require the program to be completed. *Effort* is an example of such features. This metric represents the mental effort needed to develop or maintain a software module. A high value for *Effort* means that the module is difficult to change. One of the estimations for this metric has been developed by Maurice Halstead [7], denoted by *Halstead_Effort*. *Halstead_Effort* is calculated for modules written in COBOL and PL1 using the following formula:

$$E = D * V \quad (1)$$

In this formula, D stands for *Difficulty* and is calculated as

$$D = (n1 / 2) * (N2 / n2), \quad (2)$$

Where n1 is the number of distinct operators
, n2 is the number of distinct operands and N2 is the whole number of operands.

The second parameter, V, stands for *Volume* and is calculated as

$$V = N * (LOG2\ n) \quad (3)$$

Where n is the module vocabulary size and is calculated as n = n1 + n2 and N is the module length calculated as N = N1 + N2.

During the past years, many researchers have attempted to evaluate different methods and several defect prediction systems have been proposed. The results of these systems are given in terms of classification accuracy, precision, performance, etc. However, these factors do not really show the goodness of the model.

In this paper, we will introduce an accurate software defect prediction system which aims to provide high usability beside good accuracy. In other words, the user will not have to measure all the above mentioned metrics for any of the modules.

The rest of the paper is organized as follows. In Section 2, we describe our approach and introduce different parts of the system in detail. The experimental results of any part of the system are also given in this section. Finally, Section 3 concludes the paper.

II. The Proposed System

As mentioned in Section 1, the main goal of this paper is to develop a defect prediction system which brings about a high degree of usability. If we can discover some probable hidden relations among different metrics, we will be able to develop an estimation system to estimate some metrics' values from a combination of others. So, the user will be required to provide just a few metric values. For this purpose, we first divide the set of metrics into two categories. The first category denoted as *Type-1-Faeatures* includes the set of metrics which are easily quantifiable. The second category denoted as *Type-1-Faeatures* represents design and Halstead metrics which have some problems to be measured, as discussed in the previous section. Table 1 indicates an example this categorization for the metrics belonging to KC1, which is one of the NASA MDP datasets. This example will be used in the next subsections to illustrate the experimental results.

Table I. Categorization of KC1 dataset metrics according to ease of measurement

Type-1-Features	Type-2-Features
Loc_Blank	Cyclomatic_complexity
Branch_Count	Design_Complexity
Loc_Code_And_Comment	Essential_Complexity
Loc_Comments	Halstead_Content
Loc_Executable	Halstead_difficulty
Num_Operands	Halstead_Effort
Num_Operators	Halstead_Error-Est
Num_Unique_Operands	Halstead_Length
Num_Unique_Operators	Halstead_Level
Loc_Total	Halstead_ProgTime
	Halstead_Volume

The proposed defect prediction system is composed of three major components, namely the approximate dependency miner, the estimation part and the fuzzy rule-based classifier. A high-level view of the system is shown in Fig. 1.

A. Approximate Dependency Miner

Functional dependencies (FDs) are defined as relationships between attributes of a relational scheme R, and are presented in expressions of the form $X \rightarrow A$. In this expression X (referred to as the Left-Hand Side (LHS) of the dependency) is a subset of attributes belonging to R and A (referred to as the Right-Hand Side (RHS) of the dependency) is an attribute of R. A functional dependency is said to be valid in a given relation r over R , if for all pairs of tuples t, u belonging to r, we have

$$(t[X_i] = u[X_i], \text{ for all } X_i \text{ in } X) \Rightarrow t[A] = u[A] \quad (4)$$

, where $t[x]$ is the value assigned to the attribute x of the tuple t.

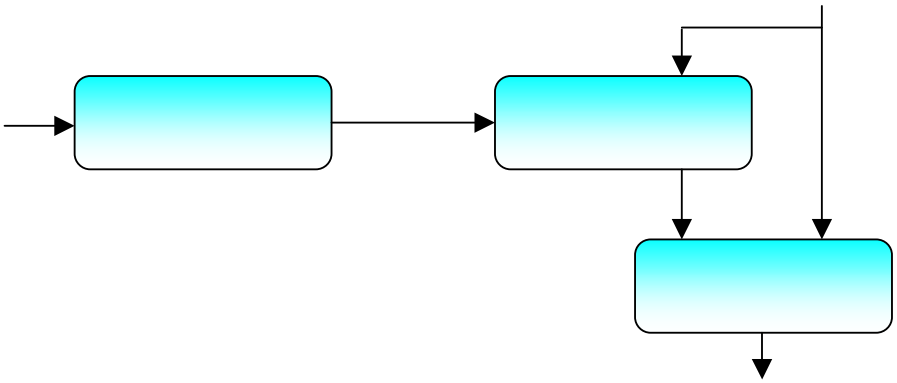


Fig. 1. The high-level architectural view of the proposed defect prediction system

Classical Functional dependencies are used in relational schema design in order to normalize relations to be free of redundancy and update anomalies. These dependencies do not allow for exceptions and are sensitive to noisy data. Approximate Dependencies (ADs) are dependencies which do not hold over a fraction of data and thus have a higher flexibility for exceptions and noisy data.

In dependency mining part of the system, we use AD-Miner which was proposed in [8] as an incremental mining algorithm. This algorithm uses logical operations on binary strings to find the set of minimal approximate dependencies (having an acceptable accuracy) between attributes. Most of other dependency mining approaches already proposed are not incremental and so have to re-scan all data and repeat the whole computations when a number of records are added to the database [9-13].

In this part of the system, we are interested in discovery of any existing dependency between the values of type-1 and type-2 features. In other words, we look for any short combination of Type-1-Features which can describe one or more features of Type-2. Since the algorithm requires discrete or nominal data, as a preprocessing step, we discretized all features into equi-size partitions. To decrease the dependence on the no. of partitions, the algorithm was run frequently, using 3 to 7 partitions for discritization. Finally, for any of Type-2-Features, the best combination of Type-1-Features (having the highest value of dependency in average) was selected as its determinant. In other words, the most accurate dependencies between a Type-2-Feature and a combination of Type-1-Features were extracted. The results of this phase over KC1 dataset features are shown in Table2. Determinant features found for Type-2-Features will then be used in the estimation part of the system.

Table II. The results of the Dependency Miner part: short-length dependencies between Type-1 and Type-2 Features

Approximate Dependency	Accuracy (%)
Branch_Count → Cylomatic_Complexity	98.9
Branch_Count , Num_Operands → Design_Complexity	98.2
Branch_Count , Loc_Blank → Essential_Complexity	95.6
Num_Unique_Operands , Loc_Executable → Halstead_content	88.7
Branch_Count , Loc_Blank → Halstead_difficulty	93.4
Branch_Count , Loc_Total → Halstead_Effort	96.5
Num_Operands , Loc_Executable → Halstead_Error-Est	96.2
Num_ Operands, Num_ Operators → Halstead_Length	93.6
Branch_Count , Loc_Blank → Halstead_Level	91.8
Num_Unique_Operands , Branch_Count → Halstead_ProgTime	93.3
Num_ Operands, Num_ Operators → Halstead_Volume	95.7

B. Fuzzy Estimation part

In this part of the system, we follow the Wang and Mendel's fuzzy rule learning method [14] and develop a set of fuzzy modeling systems with similar structures. Each of these fuzzy modeling systems is constructed to estimate the value of a Type-2-Feature using a combination of Type-1-features as its determinants (discovered in the previous section).

Unlike the Dependency Mining part, this part of the system will be used online, when the user will give a set of metric values for a module to the prediction system to be classified. At this time, the user will present just the values of Type-1-Features. As shown in Fig. 1, the values of Type-2-Features will be automatically estimated and forwarded to the classification system.

For each dataset, we used 90% of whole data to train the system and 10% for the test. For example, 1900 (out of 2107) modules contained in KC1 were used as train data and the remaining were left for test. In order to evaluate the

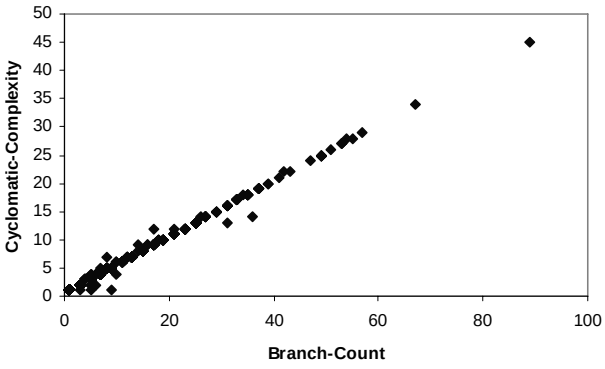
performance of the estimation part, the MSE values were computed on both train and test data for every Type-2-Feature. Table3 shows the evaluation results of this part of the system for KC1 dataset.

Fig. 2 visualizes the behavior of the estimation system in mapping the *Branch-Count* metric to *Cyclomatic-Complexity*. Fig. 2.(a) shows the existing relation (dependency) between these two metrics through 2107 modules included in KC1. Fig. 2.(b) indicates the approximated function generated by the estimation part which maps any value of *Branch-Count* to *Cyclomatic-Complexity*.

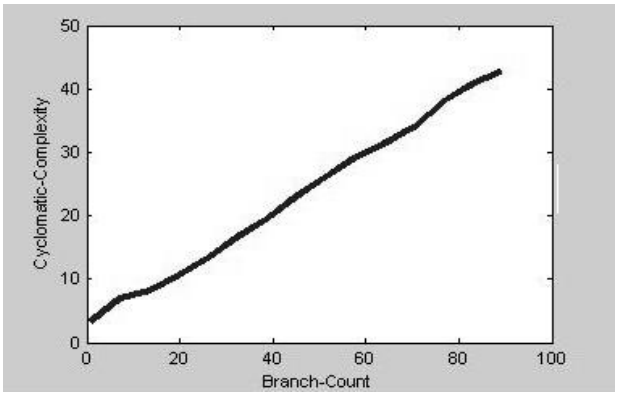
The approximated functions for other metrics (all having 2 determinants) are shown in Fig. 3.

Table III. Evaluation results of estimation system in terms of MSE and error significance

Estimation	MSE on Train Data	MSE on Test Data
Branch_Count → Cylomatic_Complexity	0.05	0.09
Branch_Count , Num_Operands → Design_Complexity	0.07	0.1
Branch_Count , Loc_Blank → Essential_Complexity	0.04	0.04
Num_Unique_Operands , Loc_Executable → Halstead_content	0.81	0.93
Branch_Count , Loc_Blank → Halstead_difficulty	0.35	0.38
Branch_Count , Loc_Total → Halstead_Effort	4.82	6.03
Num_Operands , Loc_Executable → Halstead_Error-Est	0.009	0.01
Num_ Operands, Num_ Operators → Halstead_Length	1.24	1.11
Branch_Count , Loc_Blank → Halstead_Level	0.02	0.08
Num_Unique_Operands , Branch_Count → Halstead_ProgTime	3.5	5.13
Num_ Operands, Num_ Operators → Halstead_Volume	2.88	3.43



(a)



(b)

Fig. 2. (a) *Branch-Count* Vs *Cyclomatic-Complexity* for KC1 dataset; (b) The approximated function which maps *Branch-Count* to *Cyclomatic-Complexity*

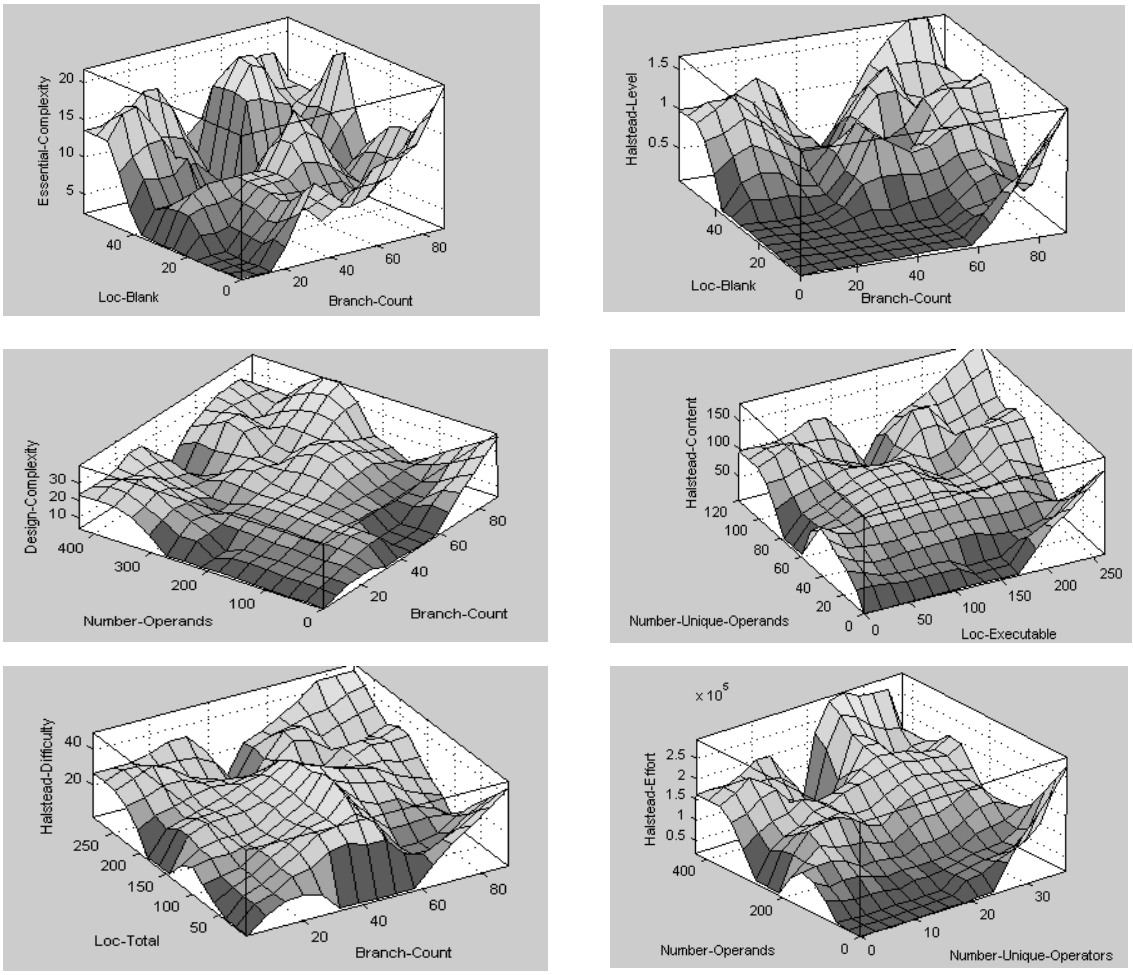


Fig. 3. The surface viewer of approximated functions mapping easy-to-measure metrics into hard-to-measure ones

III. Conclusion

In this paper, we introduced a highly usable software defect prediction system. The system was assessed using NASA which is a widely used benchmark dataset. In the mining part of the system, a set of dependencies among hard-to-measure features of the dataset and easy-to-measure ones were discovered. Then, we developed a set of fuzzy modeling systems, each of which estimates the value of one of the hard-to-obtain features from its specified determinants. In this part of the system, we followed the Wang and Mendel's fuzzy rule learning method. The evaluation of the estimation systems was accomplished by computing the MSE values for all features. The results showed the high ability of the system in terms of approximation. Using this system, the user will not have to measure all the required mentioned metrics for any of the modules. All of the hard-to-measure features will automatically be estimated with a high accuracy. As a future task, we are going to develop a fuzzy classification system using NASA datasets. The classifier accuracy will be verified in two ways. First, we use the actual values of all features from test data. The accuracy value obtained for this case will then be compared to the similar case, where just the first category features are fed into the classifier and the other features are automatically estimated (using the system proposed in this paper) and used. The very low difference between these two classification rates can be promising and will be another evidence for successfulness of the fuzzy estimation part proposed in this paper.

References

[1] L.C. Briand, W.L. Melo, and J. Wu¨ st, (2002). Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects," IEEE Trans. Software Eng., 28 (7), pp. 706-720.

[2] J. Dem_sar, (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. Machine Learning Research, 7, pp. 1-30.

[3] N. Nagappan, T. Ball, B. Murphy, Using Historical Data and Product Metrics for Early Estimation of Software Failures, In Proc. ISSRE 2006, Raleigh, NC, 2006.

[4] L. Briand, Measurement and Modeling in Software Engineering, presented at Foundations of Empirical Software Engineering Workshop—The Legacy of Victor Basili, International Conference Software Engineering, 2005.
[5] M. Chapman, P. Callis, and W. Jackson, "Metrics Data Program," NASA IV and V Facility, http://mdp.ivv.nasa.gov/, 2004.
[6] Stefan Lessmann, (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 34 (4), pp. 485-496.
[7] M.H. Halstead, (1977). Elements of Software Science. Elsevier.
[8] S.M. Fakhrahmad, M.H. sadreddini, M. Zolghadri jahromi, (2008). AD − Miner: A new incremental method for discovery of minimal approximate dependencies using logical operations, Intelligent Data Analysis 12, pp. 1–13.
[9] P. A. Flach and I. Savnik. (1999). Database dependency discovery: a machine learning approach, AI communications, 12 (3). pp. 139–160.
[10] Y. Huhtala, J. Kärkkäinen, P. Porkka and H. Toivonen. (1999) TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. The Computer Journal. 42(2). pp. 100–111.
[11] Y. Huhtala , J. Kärkkäinen , P. Porkka and H. Toivonen, Efficient Discovery of Functional and Approximate Dependencies Using Partitions, In: Proc. the Fourteenth International Conference on Data Engineering, (February 1998), pp. 392 - 401.
[12] S. Lopes , J.M. Petit , L. Lakhal, Efficient Discovery of Functional Dependencies and Armstrong Relations, in: Proc. ICDT 2000, the 7th International Conference on Extending Database Technology: Advances in Database Technology, vol 1777, pp. 350 – 364, 2000.
[13] S.L. Wang, J.S. Tsai, B.C. Chang, "Mining Approximate Dependencies using partitions on Similarity-Relation-based Fuzzy databases", in : Proc. IEEE SMC'99, Vol. 6, pp. 871–875, 1999.
[14] Wang L-X, Mendel JM (1992) Fuzzy basis functions, universal approximation, and orthogonal least squares learning. IEEE Trans. Neural Network 3, 807–814.