2014 IEEE 38th Annual International Computers, Software and Applications Conference

A Scenario-Based Approach to Predicting Software Defects Using Compressed C4.5 Model

Biwen Li, Beijun Shen, Jun Wang, Yuting Chen
School of Software
Shanghai Jiao Tong University
Shanghai, China
{amis.lbw, bjshen}@sjtu.edu.cn

Tao Zhang, Jinshuang Wang
Institute of Command Information System
PLA University of Science and Technology
Nanjing, China

***Abstract*—Defect prediction approaches use software metrics and fault data to learn which software properties are associated with what kinds of software faults in programs. One trend of existing techniques is to predict the software defects in a program construct (file, class, method, and so on) rather than in a specific function scenario, while the latter is important for assessing software quality and tracking the defects in software functionalities. However, it still remains a challenge in that how a functional scenario is derived and how a defect prediction technique should be applied to a scenario. In this paper, we propose a scenario-based approach to defect prediction using compressed C4.5 model. The essential idea of this approach is to use a *k-medoids* algorithm to cluster functions followed by deriving functional scenarios, and then to use the C4.5 model to predict the fault in the scenarios. We have also conducted an experiment to evaluate the scenario-based approach and compared it with a file-based prediction approach. The experimental results show that the scenario-based approach provides with high performance by reducing the size of the decision tree by 52.65% on average and also slightly increasing the accuracy.**

Keywords-Defect Prediction; Scenario; Software Clustering; C4.5 Model

I. Introduction

Defect prediction has been widely used in industry to predict the amount and the kinds of software defects in a system [1-5]. It can be used to identify the modules that are most likely to contain defects prior to testing. A number of defect prediction techniques and prediction models have been proposed to predict defects in large-scale software systems, which can be divided into two respects: metric-based and historical data-based. The former refers to an idea of predicting of defects on the basis of the software and its structure, size, coding style, etc. [6-8], and the latter refers to the idea of predicting of defects by mining the code repositories, analyzing the source code changes, and learning from the previous software defects [9-14].

However, many existing techniques tend to predict the defects in some program constructs (file, class, method, and so on) rather than in some specific function scenarios, while the latter is important and necessary for assessing the software quality and tracking the defects in software functionalities [7, 15, 16]. It needs to perform defect prediction at the scenario level, which is motivated by two examples:

- When a program is suspected to contain some software defects, an engineer usually reveals and recovers the defects in the program by diagnosing the software functions or scenarios and locating the defects inside.
- A modification of a program may several methods be changed. When a defect is hidden in a change, a programmer has to go through the software functions and scenarios instead of inspecting the modified code line by line.

It still remains a challenge in that how a defect is located and fixed in some software function or scenario, although some previous researches have shown that a fine-grained defect prediction can result in an overall improved expressiveness (i.e., granularity and context given to a developer) and effectiveness (i.e., accuracy of the prediction) [17]. One main difficulty for this is to derive a functional scenario and apply a defect prediction technique to a scenario. For example, when a functional scenario is defined by chaining all the functions called, it is not easy to predict the defects on the basis of the state of the function call stack, and it is also not easy to locate the actual defect when a failure is caused by the top function in the stack.

In this paper, we propose a scenario-based approach to defect prediction using compressed C4.5 model. In the study, we use a *k-medoids* algorithm to cluster functions followed by deriving functional scenarios, and then apply the C4.5 model to predict the fault in the scenarios. A two-dimension distance matrix is used in construction of the call graph, which reduces the computational complexity to $O(|V|^3)$. Compressed C4.5 model is used to improve the defect prediction accuracy during the model learning phase. C4.5 algorithm is an algorithm developed by Quinlan, which builds a decision tree from a set of training data by using the concept of information entropy [27]. The Spearman's rank correlation coefficient is introduced into the new model so that the choosing of attribute as the root node of the decision tree becomes linear.

The main contributions of this paper are summarized as follows:

- Approach. We propose a scenario-based approach to deriving functional scenarios from programs, and to defect prediction based on compressed C4.5 model.
- Experiment. We have conducted an experiment to evaluate the scenario-based approach and compared it with a file-based prediction approach. The experimental results show that the scenario-based approach provides with high performance by reducing

the size of the decision tree by 52.65% on average and also slightly increasing the accuracy.

The outline of this paper is organized as follows: Section II presents the related work. Section III introduces a scenario-based approach, which is used to extract the functional scenarios by using a clustering algorithm. Section IV introduces the compressed C4.5 models. Section V presents an experiment for assessing the effectiveness of the defect prediction approach, and then makes a discussion. Section VI draws the conclusions and points out the future work.

II. Related Work

This section presents the background knowledge about defect prediction and introduces the related work about graph mining and program clustering in source code and the researches in C4.5 models.

A. *Defect Prediction*

Software defect prediction is an effective way to optimize the allocation of testing resources and improve software quality. It can be used to identify modules that are most likely to contain defects prior to the testing phase. In the past decades, various methodologies have been proposed and validated to be effective in predicting software defects. Many researchers have designed new defect prediction algorithms and/or new metrics for predicting software defects, most of which are used to predict defects through machine learning approaches. Here we focus on data prediction at various levels, such as at the components, files, and methods levels.

Many researches have been conducted to predict defects on coarse-grained level [7, 13, 41]. Zimmermann and Nagappan [41] have predicted defects based on code dependencies. They build dependency graphs and extract metrics using network measures. Further work has been done to leverage the architectural dependencies and churn measures [13]. New metrics have been defined and evaluated on Windows Server 2003.

Schröter et al. [7] have used relationships between components. Their aim was to help designers explore and assess design alternatives in terms of predicted quality easily. The study was conducted on 52 Eclipse plug-ins, and the results indicated that the software design, as well as the past failure history, can be successfully used in defect prediction. The models require relationships between components and this information is typically defined at the design phase, helping identify the failure-prone components as early as possible.

Defect prediction at the coarse-grained level can offer satisfactory prediction performance, while fine-grained prediction approaches can be more helpful in finding bugs. Hata et al. [42] have developed a fine-grained version control system for quality assurance. The experimental results showed defect prediction model at the method level is more effective than those at the package or file levels. Their work indicates that prediction accuracy is sacrificed for bug finding and should be solved in future work, which motivates our study.

In our work, we focus on functional scenarios. Nagappan et al. [4] have verified that function call related metrics can be used to predict the likelihood of defects accurately. In our study we propose a methodology about how the functional scenario is derived and clustered in order to increase accuracy at fine-grained level.

B. *Graph Mining and Program Clustering in Source Code*

Graph mining and program clustering has been widely used in analyzing software source code. Mitchell and Mancoridis [22] have presented and analyzed a clustering system, named Bunch. To produce a decomposition of a system in subsystems, Bunch uses searching techniques to partition a graphical representation of the program which represents software entities and their relations. Doval et al. [23] have proposed a structural approach to grouping software entities into clusters on the basis of the genetic algorithms. Similar to [22], the quality of clustering depends on the definition of fitness functions and searching algorithms.

Clustering algorithms based on structural information in source code have already been successfully used in the analysis of the software architecture evolution [24, 25]. For example, Wu et al. [24] have presented a comparative study of a number of clustering algorithms (e.g., an agglomerative clustering algorithm based on the Jaccard coefficient and the complete linkage update rule which uses 0.75 and 0.90 as the cutting points). To partition a software system into some meaningful subsystems, all algorithms need to be manually configured (e.g., the specification of cutting points and fitness functions).

Similarly, Bittencourt et al. [25] have presented an empirical study to evaluate four widely known clustering algorithms on a number of software systems implemented in Java and C/C++. The algorithms include edge betweenness clustering, k-means clustering, modularization quality clustering, and design structure matrix clustering.

One application for clustering algorithm is to debug defect prediction [38, 39]. Fry et al. [38] have adopted a clustering algorithm to leverage both syntactic and structural information available in static bug reports to accurately cluster the related reports, thus expediting the maintenance process. Scanniello et al. [39] have proposed fault prediction at the class level by using the BorderFlow clustering algorithm.

The scenario extraction approach we adopt in this study is based on call graph mining and k-medoids clustering algorithm. The function call graph is taken as one of the graph mining algorithm and has been adopted for defect prediction [30]. However, the scenario-based approach is different from the previous researches, since it combines function call with the clustering algorithm and is used to predict defects at a fine-grained level.

C. *C4.5 Decision Tree*

Decision tree learners have been applied to defect prediction but been proved not precise enough. Knab et al. [26] has adopted decision tree learners to predict defects on the basis of source code metrics, modification report metrics and defect report metrics, and they have achieved 62 percentages on prediction accuracy but wrongly classified nearly 40 percent of instances.

C4.5 is a decision tree algorithm that constructs decision trees in a top–down recursive divide-and-conquer manner [27]. C4.5 algorithm has been effectively applied to defect prediction [31-33].

Existing techniques have improved C4.5 in different ways [34-37, 40]. Quinlan [34] has shown the weakness of C4.5 with continuous attributes and applied an MDL-inspired penalty to decrease the tree size and increase the predictive accuracy. Ruggieri [35] has proposed an efficient version of the algorithm, called EC4.5. It improves C4.5 by adopting the best among three strategies for computing a information gain of continuous attributes. Zhou et al. [36] have combined neural network with C4.5 algorithm. The algorithm trains a neural network ensemble, and then employs the trained ensemble to generate a new training set through replacing the desired class labels of the original training examples with those outputs from the trained ensemble. Baglioni et al. [37] have improved C4.5 by means of prior knowledge. The adaption considers other knowledge in real application (e.g. owned by experts of the field) that can be used in conjunction with the one hidden inside the examples. Thakuret et al. [40] have re-optimized ID3 and C4.5 decision tree algorithm by providing a simple modification to the attribute selection methods. The optimization modifies the SplitInfo calculation in C4.5 and gets a decision tree with high classification accuracy.

In this study, the Spearman's rank correlation coefficient is introduced into the new models so that the choosing of attribute as the root node of the decision tree becomes much more in line.

III. A SCENARIO-BASED APPROACH

Next provides the details about the scenario-based approach. The approach consists of two steps:

1) Call Graph Distance Matrix Algorithm. Functional scenarios are extracted statically from the source code. The codes will be represented as a call graph, where the nodes are the functions and the edges are the call relationships among the functions. A 2-dimension matrix is adopted to represent the distances among the nodes in the call graph.

2) Scenario Clustering. Based on the graph representation, scenarios are grouped into clusters using k-medoids algorithm.

A. Call Graph Distance Matrix Algorithm

A scenario is defined on the basis of function calls. The first step of scenario deriving is to build a call graph. A call graph is a directed graph that represents the function calling relationship among subroutines in a program [18].

The distance between two nodes in a call graph is defined as the shortest distance between the two nodes. One classical algorithm for shortest distance is the Dijkstra Algorithm[1].

Let a graph contain $|V|$ nodes and $|E|$ edges. The time complexity of Dijkstra algorithm is $O(|V|^2)$. If a min-priority queue (i.e. the Fibonacci Heap) is used, the time complexity can be reduced to $O(|E| + |V|log|V|)$ [19].

A 2-dimension matrix can also be used to represent the distances among the nodes in the call graph. When the 2-dimension matrix is used, the shortest distance will be computed for $\frac{1}{2}(|V|^2)$ times, and the overall time complexity will reach up to $O(|V|^4)$.

Three properties are held by a call graph:

- The distance between 2 nodes is of a length 1.
- The graph diameter is small with respect to the number of nodes. For example, a call graph with over 10000 nodes has a diameter less than 100.
- A graph contains a limited number of edges. In other words, a function call graph is a sparse graph.

With these properties, a call graph distance matrix is relatively sparse and most distance values in the matrix are small positive integers. Different from the existing algorithms for construction of call graphs, we build the distance matrix on the basis of the connection relationships among nodes.

Algorithm 1 shows the algorithm for construction of call graph distance matrix. The input includes an initial graph and an empty 2-dimension distance matrix. The distances among nodes increase from 1 to the diameter of the graph.

Algorithm 1 Call Graph Distance Matrix

```
1:  function Distance (Graph, matrix) :
2:      for each row i column j in matrix do
3:          matrix[ i ][ j ] = infinity
4:      end for
5:      distance = 1
6:      for each edge e( i , j ) in Graph do
7:          matrix[ i ][ j ] = distance
8:          matrix[ j ][ i ] = distance
9:      end for
10:     updated = true
11:     while updated do
12:         distance ++
13:         updated = false
14:         for each row i column j in matrix do
15:             if matrix[ i ][ j ] == distance - 1 then
16:                 for each vertex k in Graph do
17:                     if k != i && matrix[ j ][ k ] == 1&& matrix[ i ][ k ] > distance then
18:                         matrix[ i ][ k ] = distance
19:                         matrix[ k ][ i ] = distance
20:                         updated = true
21:                     end if
22:                 end for
23:             end if
24:         end for
25:     end while
26: end function
```

The workflow is defined below:

1. Set the initial values in the distance matrix as positive infinity.
2. For each *edge(i,j)* in *Graph*, set the *distance* value (*matrix[i][j]* and *matrix[j][i]*) as weight 1.
3. Let *distance*++. For each value in the matrix, if the value *matrix[i][j]* is (*distance-1*), there exists a call relationship between *i* and *j*. Set the edge value as *distance*.
4. Repeat step 3 until there is no change in the distance value.

When one iteration completes, each distance value is greater than or equal to its previous value. Since each $matrix[i][j]$ $(i! = j)$ will be checked at most once, when

[1] http://en.wikipedia.org/wiki/Dijkstra's_algorithm

every node is visited, the time complexity in a worst case is $O(|V|^3)$, which is more efficiency than the Dijkstra algorithm whose time complexity is $O(|V|^4)$.

B. Scenario Clustering

Clustering is used to reduce scenario duplication. Suppose defects are predicted directly after the call graph building process, scenarios may share some common call paths. The main idea of clustering is to group instances with similar attributes by taking static classification of call paths. As a result, the instances in each group have relatively high similarity.

A commonly used clustering algorithm is k-means [28], while it is not suitable for scenario clustering. If we use functions of the call graph as the clustering objects, and define the frequency of inter function calling as the object distance, the k-means algorithm can satisfy the requirements for scenario extraction. However, the centroids of the k-means can barely be any of the input objects (i.e., functions). Thus it becomes impossible to figure out the distance between objects and the centroid.

In our study, we use a mutation of k-means, which is called k-medoids [29], to solve the problem. In contrast to the k-means algorithm, k-medoids additionally takes a medoid as an object. Medoids are representative objects of a data set or a cluster with a data set whose average dissimilarity to all the objects in the cluster is minimal. Medoids are similar in its concept to centroids, but they are always the members of the data set. Commonly, the sum of distances between medoids and the other objects in the cluster is the shortest. The most common realization of k-medoid clustering is the Partitioning Around Medoids (PAM) algorithm. The workflow is as follows:

```
1. Set initial value of medoids, and randomly select $k$ of
   the $n$ data points as the medoids.
2. Associate each data point to the closest medoid.
   ("closest" is defined using Euclidean distance)
3. For each medoid $\mu_i$
       For each non-medoid data point $x_j$
           Swap $\mu_i$ and $x_j$ and compute the total cost of
           the configuration $V_{ij}$
4. Select the configuration with the lowest cost.
5. Repeat steps 2 to 4 until there is no change in the
   medoid.
```

The most time-consuming step of k-medoids is the third step. The complexity of computing V_{ij} is $O(n)$. The complexity of swapping step for each medoid μ_i and each non-medoid data point x_j is $O(kn^2)$. Suppose the upper bound of the iteration is t, the time complexity of k-medoids is $O(ktn^2)$. To increase the computation efficiency, the third step is taken in parallel in our study.

IV. Compressed C4.5 Models

This section provides an overview of C4.5 models and Spearman's rank correlation coefficient, and then presents the compressed C4.5 models in detail.

A. Preliminaries

C4.5 algorithm uses the concept of information entropy to build a decision tree from a set of training data. The Spearman's rank correlation coefficient is introduced into the new models so that the choosing of attribute as the root node of the decision tree becomes much more in line.

1) The C4.5 Algorithm: Let S be a set with n data samples, S can be divided into c different classes $C_i (i = 1, 2, \cdots, c)$, and every class C_i have n_i samples. The entropy of dividing S into c classes is defined,

$$E(S) \equiv -\sum_{i=1}^{c} p_i log_2(p_i) \tag{1}$$

where $p_i = \frac{n_i}{n}$ is the probability of a sample in S that belongs to class C_i. Entropy characterizes the purity of a sample set.

Let the set of all the different values of attribute A be X_A, and S_v be the subset of samples with value v on attribute A, that is $S_v = \{s \in S | A(s) = v\}$. After an attribute A is chosen to be the root of a sub-tree, the entropy of classifying S_v is defined,

$$E(S, A) \equiv \sum_{v \in X_A} \frac{|S_v|}{|S|} E(S_v) \tag{2}$$

where $E(S_v)$ is the entropy of dividing samples in set S_v into c classes. The information gain of attribute A to the sample set S is,

$$Gain(S, A) \equiv E(S) - E(S, A) \tag{3}$$

C4.5 uses the gain ratio as the basis of choosing attributes as the root of a sub-tree when the decision tree is constructed. The *gain ratio* is,

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInfo(S, A)} \tag{4}$$

Here *split information* is

$$SplitInfo(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|} \tag{5}$$

where S_i is c sample subsets by dividing S using c values of attribute A. Split information is the entropy of S on all values of attribute A.

2) Spearman's Rank Correlation Coefficient: The Spearman's rank correlation coefficient is used to study the relationships among variables and to quantify the degree of correlation of two columns of Pearson correlation coefficients among the ranked variables. It is calculated during the construction of the decision tree by choosing an attribute with values from all instances as X_i, letting defects number of every instance be Y_i, and then converting the n raw scores X_i, Y_i to ranks x_i , y_i .The Spearman correlation coefficient ρ is computed:

$$\rho \equiv \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \tag{6}$$

Tied values are assigned a rank equal to the average of their positions in an ascending order. For example, the third and fourth values are equal, and the corresponding rank is $\frac{3+4}{2} = 3.5$

B. C4.5 Model Improvement

In our previous study we have compared Bayes Network, CART and C4.5 by analyzing their confusion matrix (when instances are classified, a confusion matrix is be generated by WEKA for every model), and found that C4.5 is the best.

However, the existing prediction models may not be precise enough and an optimization of the decision tree algorithm is anticipated. To achieve better accuracy, we attempt to improve C4.5 in three directions [20].

1) Compressed C4.5 model I: The first model multiplies the Spearman's correlation coefficient and the gain ratio, and then uses the product to replace the original gain ratio for selecting test attributes. The Spearman'srank correlation coefficient can be positive or negative.The gain ratio is defined as

$$GRMod1\ (S, A) \equiv GainRatio(S, A) \times \rho \tag{7}$$

We adopt multiplication instead of using addition, according to the situation that both the gain ratio and the Spearman's rank correlation coefficient can represent the relationships between the metrics and the defects, while the weights of them are not clear.

2) Compressed C4.5 model II: In the second compressed C4.5 model, the Spearman's rank correlation coefficients are sorted in an ascending order. Let the ranks of the coefficients be $Rank(\rho)$. Compressed C4.5 model II uses $Rank(\rho)$ instead of ρ as the multiplier. This model ignores the value of the Spearman's rank correlation coefficient, but takes the importance of every attribute into account.

The gain ratio for compressed C4.5 model II is defined as

$$GRMod2\ (S, A) \equiv GainRatio(S, A) \times Rank\ (\rho) \tag{8}$$

3) Compressed C4.5 model III: The third model introduces the Spearman's rank correlation coefficient into the process of calculating the gain ratio in order to balance the fluctuation of gain ratio in different metrics. The main idea of the C4.5 algorithm is to choose the attribute with the biggest information gain (in C4.5, we choose the root node on the basis of the gain ratio which is calculated by dividing information gain by using the split information, while the split information is introduced only to solve the multi-valued bias problem in ID3 [21], the ancestor of C4.5.) as the root node of a sub-tree in which the information gain is the compression of the entropy expectation caused by assigning the value of attribute A. Along with the information gain, we adopt Spearman's rank correlation coefficient as the basis. The first step of using the compressed C4.5 model III is to reduce the importance of information gain. Therefore, we re-define *E(S,A)*, and the entropy of classifying S_v by attribute A is,

$$E'(S, A) \equiv \sum_{v \in X_A} \left(\frac{S_v}{S} + \rho\right) E(S_v) \tag{9}$$

Since the split information is used to reduce the influence of the multi-valued bias problem, we keep it in the second step, but let it be significant when calculating gain ratio. Split information in formula (5) is re-defined,

$$SplitInfo'(S, A) \equiv -\sum_{i=1}^{c} \left(\frac{|S_i|}{|S|} + \rho\right) log_2 \frac{|S_i|}{|S|} \tag{10}$$

The modified gain ratio is calculated by using the formula

$$GRMod3(S, A) \equiv \frac{Gain'(S, A)}{SplitInfo'(S, A)} = \frac{E(S) - E'(S, A)}{SplitInfo'(S, A)} \tag{11}$$

V. Experimental Study

We have performed an experiment to evaluate the effectiveness of scenario-based defect prediction approach on improved C4.5 models. Next describes the detailed setup, including the defect prediction process, the subject systems, the metrics (features) and evaluation measures.

A. Defect Prediction Process

To evaluate our approach, we have compared our approach with the file-based prediction approach.

The predict process is divided into three steps: mining metrics from a software repository including source code metrics, change metrics and defect history metrics; inputting the formatted metrics into the improved C4.5 models and outputting prediction accuracy, verifying the effectiveness of the scenario-based approach by comparing it with the file-based approach.

Fig.1. describes the workflow of experiment, where *DeriveScenario* tool is implemented by the authors to extract functional scenarios; *Understand* tool[2] is used to extract the source code metrics from source codes and extracted scenarios; *FilterMetrics* tool, *LinkBugs* tool and *GenerateWekaData* tool are developed by the authors to filter metrics from *Understand* and to mine change metrics and defect metrics and format metrics according to *ARFF*[3] format respectively.

Along with the scenario-based approach through *DeriveScenario* tool, we have extracted file-level metrics to build file-based defect prediction approach. For each approach, the output defect density is examined and the classification accuracy of training data set and testing data set is recorded.

[2] http://www.scitools.com/index.php

[3] An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. http://www.cs.waikato.ac.nz/~ml /weka/arff.html

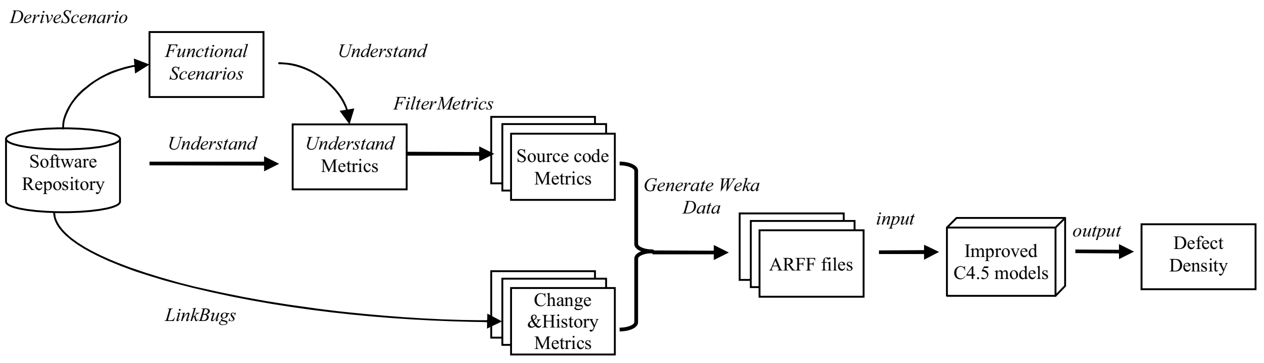

Fig. 1. Defect Prediction Process

We have further compared the sizes of the decision trees. The inputs of the improved C4.5 models include source code metrics along with the change metric and defect history metric. The output is a classification of the source code files into ten levels by their defects number. In the experiment, WEKA[4] was used to predict defects.

B. Data

We perform our experiments on the Eclipse platform[5]. Eclipse is a popular open source system that has been extensively studied before. In the experiment, we focused on the metrics on the *org.eclipse.jdt.core* component.

Table I shows an overview of the *org.eclipse.jdt.core* versions used in this study. We downloaded the repository from CVS[6] and the bug reports from Bugzilla[7]. The versions lasted from June 27, 2002 to September 9, 2011.

C. Metrics

1) Source Code Metrics: We have used two different levels of source-code metrics. Each version downloaded from CVS was preprocessed and only files with defect history in Bugzilla were left and tracked. After that, *Understand* tool was used to extract source code metrics.

Table II lists the metrics used for file-based defect predictor. Table III lists the function-level metrics that can be used to extract scenario-based source code metrics. We re-organize them according to scenarios before data generation.

2) Change & History Metrics: We only use one change metric, i.e., Number of Revisions (NR). The NR metric represents the number of revisions of a given Java class during the development of the investigated releases of a software system.

D. Evaluation Measures

The performance measures used for the scenario-based defect predictor and file-based defect predictor are summarized below.

TABLE I. VERSIONS OF ECLIPSE JDT MODULE

No	Version	Release Date	No	Version	Released Date
1	2.0.0	Jun 27, 2002	16	3.2.2	Feb 12, 2007
2	2.0.1	Aug 29, 2002	17	3.3.0	Jun 21, 2007
3	2.0.2	Nov 7, 2002	18	3.3.1	Sep 21, 2007
4	2.1.0	Mar 27, 2003	19	3.3.2	Feb 21, 2008
5	2.1.1	Jun 27, 2003	20	3.4.0	Jun 13, 2008
6	2.1.2	Nov 3, 2003	21	3.4.1	Sep 11, 2008
7	2.1.3	Mar 10, 2004	22	3.4.2	Feb 11, 2009
8	3.0.0	Jun 25, 2004	23	3.5.0	May 27, 2009
9	3.0.1	Sep 26, 2004	24	3.5.1	Sep 17, 2009
10	3.0.2	Mar 11, 2005	25	3.5.2	Feb 11, 2010
11	3.1.0	Jun 27, 2005	26	3.6.0	Jun 3, 2010
12	3.1.2	Sep 29, 2005	27	3.6.1	Sep 9, 2010
13	3.1.2	Jan 18, 2006	28	3.6.2	Feb 10, 2011
14	3.2.0	Jun 6, 2006	29	3.7.0	June 13, 2011
15	3.2.1	Sep 21, 2006	30	3.7.1	Sep 9, 2011

- Accuracy. Accuracy measures the percentage of correctly classified instances of both the defective (true) and non-defective (false) classes.
- Decision tree size. The size of a generated decision tree represented by a leaf node number and non-leaf node number is the key factor that determines the running time of classifying defects and therefore reflects the efficiency of the models.

E. Result and Analysis

Next presents the detailed experimental results for comparisons of file-based and scenario-based with and without defect-free functions.

1) With Defect-free Functions: In the experiments, 10 runs of 10-fold cross validation were performed when we trained the classification models on the fit data set.

[4] http://www.cs.waikato.ac.nz/ml/weka/
[5] http://www.eclipse.org
[6] http://archive.eclipse.org/arch/
[7] https://bugs.eclipse.org/bugs/

TABLE II. FILE-LEVEL METRICS OF UNDERSTAND

Understand Metrics	Source Code Metrics
SumCyclomatic	WMC
MaxInheritanceTree	DIT
PercentLackOfCohesion	LCOM
CountOuput	FanOut
CountLineCode	LOC
CountInput	FanIn
CountDeclMethodAll	RFC
CountClassDerived	NOC
CountClassCoupled	CBO
CountDeclClassMethod	NOM
CountDeclClassVariable	NOA

TABLE III. FUNCTION-LEVEL METRICS OF UNDERSTAND

Understand Metrics	Source Code Metrics
Cyclomatic	WMC
Essential	ESS
MaxNesting	MN
CountOuput	FanOut
CountLineCode	LOC
CountInput	FanIn
CountPath	CP
RatioCommentToCode	RCC

TABLE IV. COMPARISON OF FILE-BASED AND SCENARIO-BASED APPROACH ON ACCURACY AND DECISION-TREE SIZE (WITH DEFECT-FREE FUNCTIONS)

		File-based		Scenario-based	
Training set	**true**	61076	93.0923%	61462	93.6806%
	false	4532	6.9077%	4146	6.3194%
Testing set	**true**	60925	92.8622%	60787	92.6518%
	false	4683	7.1378%	4821	7.3482%
non-leaves		103		629	
leaves		52		315	

Table IV compares the accuracy and decision-tree size between the file- and scenario-based approaches. As shown in table IV, the accuracy indicates that there exists no apparent difference. Even more, file-based approach (92.8622%) was better than scenario-based approach (92.6518%) on testing set. The result is not satisfactory with the decision-tree size of scenario-based approach 6 times larger than the file-based approach.

TABLE V. COMPARISON OF FILE-BASED AND SCENARIO-BASED APPROACH ON ACCURACY AND DECISION-TREE SIZE (WITHOUT DEFECT-FREE FUNCTIONS)

		File-based		Scenario-based	
Training set	**true**	2999	64.5919%	3392	73.0877%
	false	1644	35.4081%	1249	26.9123%
Testing set	**true**	1596	34.3743%	1710	36.8455%
	false	3047	65.6257%	2931	63.1545%
non-leaves		1329		629	
leaves		665		315	

To find out the reason, we have taken a deep investigation about the confusion matrix built in *WEKA* (when instances were classified, a confusion matrix was generated by *WEKA* for every model). Table VI shows the confusion matrix of file-based and scenario-based approach (with defect-free functions). In the matrix, instances were classified into ten levels by their defect number and most instances are grouped into the first level *a*. Take the first row of the training set in file-based approach as an example. 60939 of 60965 instances are of level *a*. In fact, most instances were defect-free (92.88% in file-based approach while 92.85% in scenario-approach). This draws a conclusion that *"an overly centralized data distribution will affect the eventual learning result and lead to bad defect prediction performance"*.

2) Without Defect-free Functions: In order to reduce the impact of the centralized data distribution, we have removed all the instances in level *a* and build the defect prediction model again. We have pruned the defect-free instances and only worked on the instances with one or more defects. The results are listed in table V and tableVII.

During the removing phase, data in confusion matrix is evenly-distributed in table VII. In Table V we compare the results of the scenario-based approach with the ones of file-based approach for accuracy and decision-tree size. The predictive accuracy of scenario-based approach is 73.0877% in training set and 36.8455% in testing set. Although the accuracy decreases for both approaches since the defect-free instances are pruned, our scenario-based shows better performance.

In summary, the accuracy of scenario-based defect prediction is higher than that of the file-based defect prediction. The scenario-based approach increases the accuracy by 8.5% in the training set and 2.5% in the testing set. For running time, the scenario-based approach offers better result, which reduces the decision tree size by 52.65% on average with leave nodes and non-leave nodes of size 315 and 629.

F. Summary and Threats to Validity

Our evaluation shows that scenario-based defect prediction on improved C4.5 model offers high performance. If the defect-free instances are removed, the scenario-based approach outperforms file-based approach in its accuracy and running time.

The threats to the validity are as follows. The threat to internal validity lies in the implementation of the experimental study. To reduce this threat, the authors of this paper reviewed and tested the code.

TABLE VI. CONFUSION MATRIX OF FILE-BASED AND SCENARIO-BASED APPROACH (WITH DEFECT-FREE FUNCTIONS)

			Level									
			a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**
file	**Training set**	**a**	60939	3	3	0	0	0	0	0	0	20
		b	1820	20	4	0	0	0	0	1	0	9
		c	844	3	15	0	0	0	0	1	0	12
		d	443	4	0	1	0	1	0	0	0	6
		e	380	2	1	0	3	1	0	0	0	6
		f	160	2	0	0	0	4	0	0	0	5
		g	170	1	1	1	0	0	0	0	0	6
		h	106	1	2	0	0	0	0	4	0	4
		i	86	0	0	0	0	0	0	1	5	4
		j	409	1	1	0	1	2	0	1	3	85
	Testing set	**a**	60824	60	30	7	6	2	1	2	3	30
		b	1793	20	10	6	2	1	1	1	0	20
		c	834	15	5	3	1	2	0	2	0	13
		d	424	7	6	2	0	3	1	1	1	10
		e	372	9	1	2	1	2	0	0	1	5
		f	150	5	3	0	2	0	0	0	2	9
		g	167	2	0	1	0	0	0	2	0	7
		h	99	6	3	0	1	0	1	0	1	6
		i	79	4	1	0	0	2	0	0	1	9
		j	386	17	7	4	4	8	1	1	3	72
scenario	**Training set**	**a**	60918	12	8	10	2	1	0	0	0	16
		b	1800	161	4	4	1	1	1	0	3	12
		c	780	8	75	3	1	0	1	0	1	10
		d	393	10	3	67	1	0	0	0	0	6
		e	249	15	9	4	22	0	0	0	1	6
		f	120	6	3	3	2	9	1	0	3	5
		g	113	8	4	6	0	0	15	0	0	7
		h	70	4	6	2	1	2	0	6	0	9
		i	63	3	6	0	3	1	1	1	13	2
		j	269	17	9	9	5	1	1	2	2	176
	Testing set	**a**	60645	112	61	41	13	9	9	7	9	61
		b	1837	48	24	13	10	6	3	2	4	40
		c	783	26	18	6	9	2	4	3	2	26
		d	414	17	11	5	5	3	0	2	1	22
		e	254	19	8	7	1	0	3	1	0	13
		f	122	13	4	6	0	0	1	1	1	4
		g	120	10	8	5	0	1	1	0	2	6
		h	76	5	3	2	1	0	0	1	1	11
		i	66	5	6	2	1	1	1	1	0	10
		j	317	43	16	14	9	4	5	7	8	68

The threat to external validity is from the objective programs used in the experiment. To reduce the threat, we have studied 30 versions to avoid bias. Moreover, the module we choose has been widely used in software testing and defect prediction approach.

The second external validity lies in the influence of defect-free functions. It is obvious that more experimentation is required, to generalize out results and help us take a further investigation of the modules with defect-free instances that have no obvious improvement when the scenario-based approach is adopted.

The third external threat is induced by our choice to use compressed C4.5 models. To reduce the threat, our previous work has concluded that C4.5 is best. Besides, our experimental results are measured by both accuracy and the decision tree size, which has been widely used in the evaluation of the prediction approaches.

The threat to conclusion validity concerns issues that affect the ability of drawing a correct conclusion from the analysis of the gathered data. We chose a set of statistical tests to ensure that our observations do not occur by chance.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a scenario-based approach to defect prediction using compressed C4.5 model. The approach applies a 2-dimension distance matrix and derives scenarios with a time complexity of $O(|V|^3)$, and adopts the *k-medoids* clustering algorithm to group scenarios. After building models through our improved C4.5 models, we have performed a case study on eclipse JDT components. In the experiment, we conclude that:

- With defect-free functions, the prediction accuracies of the scenario-based approach and the file-based approach are close to each other. The scenario-based approach is 0.5883% higher than the file-based approach in the training set and 0.2104% lower than the file-based approach in the testing set.
- Without defect-free functions, scenario-based defect prediction approach offers high performance. The scenario-based approach improves the prediction accuracy by 8.5% on the training set and 2.5% on the testing set.
- Without defect-free functions, the scenario-based approach reduces the size of the decision tree by 52.65% on average. The number of the leave nodes and non-leave nodes are 315 and 629.

One disadvantage of our experiment is the choosing of modules. The module studied in this paper has few defects and the distribution of defects tends to be one or two defects in one file. This generates a little interference to the comparison of the scenario-based defect prediction approach and the file-based approach. In the future we will apply the scenario-based approach to more open source software to overcome the problem of few defects in a file. Apart from that, the study result shows that the scenario-based approach performs well only when the defect-free function instances have been removed. We will also focus on analyzing and reducing the impact of defect-free functions by using other projects.

Although C4.5 is concluded to be best, further work should be conducted to compare our work with the scenario-based approach using other models.

TABLE VII. CONFUSION MATRIX OF FILE-BASED AND SCENARIO-BASED APPROACH (WITHOUT DEFECT-FREE FUNCTIONS)

			Level									
			a	b	c	d	e	f	g	h	i	j
file	Training set	a	0	0	0	0	0	0	0	0	0	0
		b	0	1651	84	27	40	5	5	10	2	30
		c	0	320	474	26	8	6	3	5	3	30
		d	0	166	57	202	7	4	2	1	2	14
		e	0	137	28	28	174	5	1	1	1	18
		f	0	53	20	12	9	60	0	0	2	15
		g	0	61	25	15	7	7	49	3	1	11
		h	0	34	12	6	7	1	2	42	3	10
		i	0	25	4	8	3	3	3	2	34	14
		j	0	105	24	21	14	9	5	8	4	313
	Testing set	a	0	0	0	0	0	0	0	0	0	0
		b	0	1143	298	120	96	32	25	28	14	98
		c	0	463	180	71	32	19	8	16	12	74
		d	0	238	93	33	30	7	7	10	1	36
		e	0	207	49	28	59	7	7	1	5	30
		f	0	83	29	12	9	7	4	3	1	23
		g	0	80	25	16	14	11	13	2	3	15
		h	0	42	14	9	6	1	3	16	4	22
		i	0	34	22	7	2	1	3	3	3	21
		j	0	164	78	37	26	23	11	12	10	142
scenario	Training set	a	0	0	0	0	0	0	0	0	0	0
		b	0	1839	57	27	14	6	10	3	6	25
		c	0	251	553	16	12	2	8	1	2	34
		d	0	105	40	302	6	1	4	4	2	16
		e	0	72	32	20	163	4	2	1	1	11
		f	0	47	15	16	5	50	3	2	3	11
		g	0	42	17	8	4	4	66	2	0	10
		h	0	20	12	10	9	4	1	30	0	14
		i	0	20	12	6	8	1	2	2	34	8
		j	0	49	24	23	12	9	7	6	6	355
	Testing set	a	0	0	0	0	0	0	0	0	0	0
		b	0	1259	317	120	89	38	34	9	21	100
		c	0	433	189	92	47	13	19	11	6	69
		d	0	202	88	87	23	14	14	10	8	34
		e	0	157	54	28	17	2	8	4	8	28
		f	0	62	32	13	7	4	6	2	5	21
		g	0	65	28	14	9	3	0	5	4	25
		h	0	33	15	9	9	6	5	5	0	18
		i	0	34	16	11	8	2	1	1	2	18
		j	0	142	70	37	38	17	15	15	10	147

ACKNOWLEDGMENT

Beijun Shen is the corresponding author. This research is supported by Science Innovation Project of Shanghai Municipal Commission of Science and Technology (No. 12dz1506900), National Science and Technology Major Project of Science and Technology of China (No. 2012zx03006-002) and National Natural Science Foundation of China (Grant No. 91118004, 61100051).

REFERENCES

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen, “A systematic and comprehensive investigation of methods to build and evaluate fault prediction models,” J. Syst. Softw., vol. 83, no. 1, 2010, pp.2–17.

[2] T. Jiang, T. Lin, and K. Sunghun, “Personalized defect prediction,” 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE). IEEE, 2013, pp.279-289.

[3] B. Caglayan, A. T. Misirli, G. Calikli, A. Bener, T. Aytac and B. Turhan, “Dione: an integrated measurement and defect prediction solution,” Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 2012, pp.20-23.

[4] N. Nagappan, T. Ball, and A. Zeller, “Mining Metrics to Predict Component Failures,” Proc. of 2006 Int'l Conference on Software Engineering (ICSE 2006), Shanghai, China, 2006, pp.452-461.

[5] M. Li, H. Zhang, R. Wu, and Z.H. Zhou, “Sample-based software defect prediction with active and semi-supervised learning,” 2012 IEEE/ACM 27th International Conference on Automated Software Engineering (ASE). IEEE, 2012, pp.201-230.

[6] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” IEEE Trans. Softw. Eng., vol. 20, no. 6, 1994, pp. 476–493.

[7] A. Schr¨oter, T. Zimmermann, and A. Zeller, “Predicting component failures at design time,” in Proceedings of the International Symposium on Empirical Software Engineering. ACM, 2006, pp. 18–27.

[8] N. Nagappan, B. Murphy, and V. Basili, “The Influence of Organizational Structure on Software Quality: An Empirical Case Study,” Proc. of 2008 Int'l Conference on Software Engineering (ICSE 2008), Leipzig, Germany, 2008, pp. 521-530.

[9] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, “Predicting Fault Incidence Using Software Change History,” IEEE Transactions on Software Engineering, vol. 26, 2000, pp. 653-661.

[10] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Where the Bugs Are,” in International Symposium on Software Testing and Analysis (ISSTA), 2004, pp. 86-96.

[11] R. Moser,W. Pedrycz, and G. Succi, “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,” Proc. of 2008 Int'l Conference on Software Engineering (ICSE 2008), Leipzig, Germany, 2008, pp.181-190.

[12] S. Kim, T. Zimmermann, E. Whitehead Jr, and A. Zeller, “Predicting Faults from Cached History,” Proc. of 2007 Int'l Conference on Software Engineering (ICSE 2007), 2007, pp.489-498.

[13] N. Nagappan,and B. Thomas, “Using software dependencies and churn metrics to predict field failures: An empirical case study,” Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. IEEE, 2007, pp.364-373.

[14] K. Herzig, S. Just, A. Rau and A. Zeller, “Predicting defects using change genealogies,” 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2013, pp.118-127.

[15] S. Shivaji, E. J. W. Jr., R. Akella, and S. Kim, “Reducing features to improve bug prediction,” In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), 2009, pp. 600-604.

[16] A. E. Hassan, “Predicting faults using the complexity of code changes,” Proc. of 2009 Int'l Conference on Software Engineering (ICSE 2009), Vancouver, Canada, 2009, pp. 78–88.

[17] H. Kagdi, “Improving change prediction with fine-grained source code mining,” In Proceedings of the twenty-second IEEE/ACM international

conference on Automated software engineering. ACM, 2007, pp.559-562.

[18] B. G. Ryder, “Constructing the call graph of a program,” IEEE Transactions on Software Engineering, vol. 3, 1979, pp. 216-226.

[19] M. Chen, R. A. Chowdhury, V. Ramachandran, “Priority queues and dijkstra's algorithm,” Technical report, UTCS Technical Report TR-07-54, 2007.

[20] J. Wang, S. Beijun, and C. Yuting, “Compressed C4. 5 Models for Software Defect Prediction,” 2012 12th International Conference on Quality Software (QSIC), IEEE, 2012, pp. 13-16.

[21] J. R. Quinlan, “Induction of decision trees,” Machine learning, Vol. 1 ,1986, pp. 81-106.

[22] B. S. Mitchell and S. Mancoridis, “On the automatic modularization of software systems using the bunch tool,” IEEE Trans. on Softw. Eng., vol. 32, 2006, pp. 193–208.

[23] D. Doval, S. Mancoridis, and B. S. Mitchell, “Automatic clustering of software systems using a genetic algorithm,” in Proceedings of the Software Technology and Engineering Practice. IEEE Computer Society, 1999, pp. 73–82.

[24] A. E. Wu, J. Hassan and R. C. Holt, “Comparison of clustering algotithms in the context of software evolution,” in Proceedings of the International Conference on Software Maintenance. IEEE Computer Society, 2005, pp. 525–535.

[25] R. A. Bittencourt and D. D. S. Guerrero, “Comparison of graph clustering algorithms for recovering software architecture module views,” in Proceedings of the European Conference on Software Maintenance and Reengineering. IEEE Computer Society, 2009, pp. 251–254.

[26] P. Knab, M. Pinzger, and A. Bernstein, “Predicting defect densities insource code files with decision tree learners,” in Proceedings of the 2006 international workshop on Mining software repositories (MSR), New York, NY, USA: ACM, 2006, pp. 119 – 125. [Online]. Available: http://doi.acm.org/10.1145/1137983.1138012.

[27] J. R. Quinlan, “C4. 5: programs for machine learning,” Vol. 1. Morgan kaufmann, 1993.

[28] J. A. Hartigan, and A. W. Manchek, “Algorithm AS 136: A k-means clustering algorithm,” Journal of the Royal Statistical Society. Series C (Applied Statistics) 28.1 (1979), pp. 100-108.

[29] L. Kaufman, and P. J. Rousseeuw, “Clustering by means of medoids,” (1987). In Dodge Y, editor. Statistical data analysis based on the L1 norm and related methods. Amsterdam: North Holland/Elsevier. pp. 405-416.

[30] M. Yin, B. Li, and C. Tao, “Using cognitive easiness metric for program comprehension,” Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on. IEEE, 2010, pp. 134-139.

[31] Y. Peng, G. Kou, G. Wang, W. Wu, Y. Shi, “Ensemble of software defect predictors: an AHP-based evaluation method,” International Journal of Information Technology & Decision Making, 2011, pp. 187-206.

[32] S. Wang and X. Yao, “Using Class Imbalance Learning for Software Defect Prediction,” IEEE Transactions on Reliability, 2012 (DOI:10.1109/TR.2013.2259203).

[33] Y. Peng, G. Wang, and H. Wang, “User preferences based software defect detection algorithms selection using MCDM,” Information Sciences 191, 2012, pp. 3-13.

[34] J. R. Quinlan, “Improved use of continuous attributes in C4.5,” arXiv preprint cs/9603103 (1996).

[35] S. Ruggieri, “Efficient C4.5 [classification algorithm],” IEEE Transactions on Knowledge and Data Engineering,Vol. 14, no. 2, 2002, pp. 438-444.

[36] Z. H. Zhou and Y. Jiang, “NeC4.5: neural ensemble based C4.5,” IEEE Transactions on Knowledge and Data Engineering, Vol. 16, 2004, pp. 770-773.

[37] M. Baglioni, Miriam, B. Furletti, and F. Turini, “DrC4.5: Improving C4.5 by means of prior knowledge,” Proceedings of the 2005 ACM symposium on Applied computing. ACM, 2005, pp. 474-481.

[38] Z. P. Fry,and W. Westley, “Clustering static analysis defect reports to reduce maintenance costs,” Reverse Engineering (WCRE), 2013 20th Working Conference on. IEEE, 2013, pp. 282-291.

[39] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies, “Class level fault prediction using software clustering,” 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), IEEE, 2013, pp. 640-645.

[40] D. Thakur, N. Markandaiah, D.S. Raj, “Re-optimization of ID3 and C4.5 decision tree,” Computer and Communication Technology (ICCCT), 2010, pp. 448- 450.

[41] T. Zimmermann, and N. Nagappan, “Predicting subsystem failures using dependency graph complexities,” 18th IEEE International Symposium on Software Reliability Engineering, 2007, p. 227-236.

[42] H. Hata, O. Mizuno, and T. Kikuno, “Bug prediction based on fine-grained module histories,” Proc. of Int'l Conf. on Software Engineering (ICSE2012), Zurich, Switzerland, 2012, pp. 200-210.