International Journal of Scientific Research in Computer Science and Engineering

Research Paper
Volume-1, Issue-1, Jan- Feb-2013
Available online at www.isroset.org

ISSN: 2320 - 7639

Defect Prediction in Software Entities Classified in Terms of Level Dependencies

Narendra Kumar Rao. B#1, Rama Mohan Reddy. A#2, Bhaskar Kumar Rao. B#3

#1Dept. of CSE, JNTU, India, narendrakumarrao@yahoo.com
#2Dept. of CSE, SVU, India, ramamohansvu@yahoo.com
#3Dept. of CSE, SRM, India, bhaskarkumarraob@gmail.com

Abstract- **Unit testing is the core fundamental to ensure code is in accordance with the design specifications. The coding and unit testing standard reflects the stability of project (not to mention the testing effort).Code stability is greatly influenced by the efforts of unit testing, which can be automated to reduce the human efforts. In spite of several tools identified for unit testing, tools need to be able to identify the level dependencies or depth of program entity usage in software fragments. This factor greatly influences unit testing complexity. Higher the level of dependency, the greater the complexity of unit testing the code. Here based on level dependencies we predict defects in any expression. A predicting defect-prone software component is an economically important activity and so has received a good deal of attention. However, making sense of the many, and sometimes seemingly inconsistent, a result is difficult. The main objectives of this paper are unbiased and comprehensive comparison between competing prediction systems. This paper mainly focuses on two learning algorithms OneR, Naive Bayes. By using those two algorithms we calculate the error rate. We can predict defects based on those error rates.**

Keywords- **Unit Testing, Level Dependency, Defect Prediction.**

1. INTRODUCTION

Unit testing is the first and the most important level of testing. As soon as the programmer develops a unit of code, the unit is tested for various scenarios. As the application is being built it is much more economical to find and eliminate the bugs early on. Hence Unit Testing is the most important of all the testing levels [1]. As the software project progresses ahead it becomes more and more costly to find and fix the bugs [2].

Steps in Unit Testing:

Step 1: Create a Test Plan.

Step 2: Create Test Cases and Test Data.

Step 3: If applicable create scripts to run test cases.

Step 4: Once the code is ready execute the test cases.

Step 5: Fix the bugs if any and re test the code.

Step 6: Repeat the test cycle until the “unit” is free of all bugs.

Extensive research effort is being invested into software unit testing automation for several years and the emergence of commercial applications implementing some of the resulting ideas are evidence of the attraction of automated testing solutions. One approach to fully automated testing is random testing.[3] In the Unit software testing literature, the random strategy is often considered to be one of the less preferred approaches.

2. DEFECT PREDICTION MECHANISM

Defect prediction it’s a new research area for software quality surety. A project team always designs to produce a quality product with zero or few defects. Quality of a product is correlated with the number of defects as well as money and time. So, defect prediction mechanism is very important in the field of software quality.

Software Defect: A software defect is an error, flaw, mistake, failure or fault in a computer program or system that produces an incorrect or unexpected result.

Defect Identification: Identifying and locating defects in software projects is a difficult task. Further, estimating the density of defects are more difficult. So, the software project team is fully focused on finding and fixing all the defects.

Defect Prediction: Defect prediction is defined as predicting defects in software components. A learning algorithm is selected and used to build a dataset and predict software defect[8].

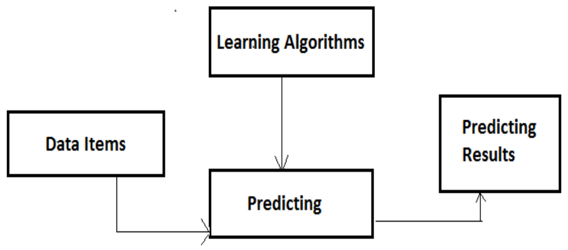

Figure1 Defect Prediction Mechanism

Data Items: A data item describes an atomic state of a particular object concerning a specific property at a certain time point. A collection of data items for the same object at the same time forms an object instance (or table row). In this paper, data items are identifiers.

Learning Algorithms: It focuses on the prediction, based on known properties learned from the training data.

In this paper, two algorithms are used for defect prediction.

The two algorithms are:

1. OneR algorithm
2. Naive Bayes algorithm

2.1 OneR Approach:

OneR is a simple and a very effective classification mostly used in machine learning applications. OneR is difficult to be improved further, due to its simplicity it can be enhanced by providing better methods for handling some of the exceptions[9].
Def: OneR short for "one rule", it's a simple classification algorithm that gives a one level decision tree.
It is simple and accurate. It is also able to predict missing values and numeric attributes.
A rule is simply a set of attribute values bound to their majority class; one such binding for each attribute value of the attribute the rule is based on.
Pseudo-code for the OneR algorithm:

```
INPUT: training data T, attributes H
For each attribute A,
    For each value VA of the attribute, make a rule as follows:
        count how often each class appears
        find the most frequent class Cf
        create a rule when A=VA; class attribute value = Cf
        calculate NCWA
     End For-Each
     Calculate the error rate of all rules
End For-Each
If more then one rule has the smallest error rate
        Choose the rule with the Highest NCW among the equal error rate rules
Else
        Chose the rule with the smallest error rate
End If
OUTPUT:
The output of above algorithm is it returns the attribute with the lowest error rate
```

If two attributes have the same error rate,then it chooses randomly. So in sometimes an error may be occur. So,we have to calculate Net Class Weight for each artifact.

Net Class Weight:
It is defined as the probability of giving a correct value with the available values. As oneR is based on a single artifact, we calculate the NCw for each artifact.

$$NCW_A = \sum_{\text{for each class } c} \frac{\varepsilon_C^A}{\varepsilon_C}$$

Where:
$€_C$= Total nmber of class C in the data set.
$€_C^A$= Total number of class C correctly classified by attribute A.

EXAMPLE: Following example illustrates the outcome of the above algorithm when applied on the weather data given in the following table.

OUTLOOK	TEMP	HUMIDITY	WINDY	PLAY
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Table 2.1: shows the dataset for playing tennis
From the above table:
Total number of "yes" classes in the data set=9
Total number of "no" classes in the data set=5
For example let us consider outlook attribute for calculating the error rate and NCW:

		Play	
		Yes	No
Outlook	Sunny	2	3
	Overcast	4	0
	Rainy	3	2

Table 2.2: Total no of chances for playing tennis
There are three cases are possible for outlook. They are
if outlook=sunny then play =no
if outlook=overcast then play =yes
if outlook=rainy then play =yes

	Errors	Total errors	NCW
Sunny—no	2/5	4/14	7/9+3/5= 1.377
Overcast--yes	0/4		
Rainy—yes	2/5		

Table 2.3: calculation of error rate and NCW

So that it calculates the error rate and NCW for each and every attribute. Then it finally selects "Humidity" as the rule as it has the highest NCW among the two rules the smallest error rate.
Advantages:

- It is simple and easy to understand.
- It calculates net class weight for each artifact due to these errors may be reduced.

Disadvantages:

- Randomly selecting an artifact when error rates are equal.
- Over fitting of nominal artifacts with near values

2.2 Naive Bayes Approach:
The Naive Bayes Classifier assigns an instance s_k with attribute values(A_1=V_1, A_2=V_2,........,A_n=V_n) to class C_i with maximum probability(C_i/(V_1,V_2,……,V_n)) for all i.
It uses the Bayes rule and assumes independene of attributes. It is mainly based on Bayesian theorem. It needs discrete values to work properly[10]. For each column, a domain has to be associated. It uses the following formula in order to calculate the probability of each attribute:

$$\boxed{P(H/E)=P(H)/P(E)\prod_I P(E_i/H)}$$

Where:

E_i = fragments of evidence E_i

P(H)	=	Prior Probability
P(H/E)	=	Posterior Probability

Pseudo code for Naive Bayes algorithm :

```
INPUT: training set T, attributes H, initial number of attributes k.
Initialize M with one attribute.
k ← k0
repeat
Add k new attributes to M, initialized using k
random examples from T.
Remove the k initialization examples from T.
repeat
E-step: Fractionally assign examples in T to attributes, using M.
M-step: Compute maximum likelihood parameters for M,
using the filled-in data.
If log P(H|M) is best so far, save M in Mbest.For every loop,
prune low-error rate attributes of M.
until log P(H|M) fails to improve by ratio δEM.
M ← Mbest
Return Mbest.
OUTPUT: selects an attribute with highest probability and returns to main.
```

EXAMPLE: Following example illustrates the outcome of the above algorithm when applied on the weather data given in the following table.

If we apply the naive bayes algorithm for the table 2.1 the following procedure has to be followed:

From the table 2.1:

Total number of “yes” classes in the data set=9

Total number of “no” classes in the data set=5

It calculates the probability for each and every attribute whether he/she can play or not in particular condition.

For example, let us consider outlook attribute. Here three conditions are there. They are sunny, overcast, rainy. So it calculates the probability for each and every case:

Outlook	PLAY=yes	PLAY=no
Sunny	2/9	3/5
Overcast	4/9	0/5
Rainy	3/9	2/5

Likewise, it calculates for temperature, humidity, wind.

Temperature	Play=yes	Play=no
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=Yes	Play=No
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=Yes	Play=No
Strong	3/9	3/5
Weak	6/9	2/5

- Naive Bayes algorithm mainly concentrates on play=no classes .So that we calculate the probability for each and every attribute. From the above all cases it is observed that the humidity when play=no attribute has the highest probability.so it selects that particular attribute and return.
- It also checks the probability of playing aor not when we given a condition.

Advantages:

- The Naive Bayes algorithm affords fast to train and fast to evaluate.
- It scales linearly with the number of predictors and rows.
- Surprisingly good for real-world problems

Disadvantages:

- Not capable of solving more complex problems.

3. Proposed System

A technique is proposed in which test inputs generated by random generator so as to make the output more useful to a test engineer. The technique aims to help the engineer isolate the root cause of a failing test input resulting in a failure. The result of the technique makes it easier to isolate the cause of failure in the form of back tracing process. Current paper aims at performing the level-based testing of given program in the form of back tracing for defect identification and correction. Random test cases are selected from test suit, if test cases passes it generates the expected results, else any test case failed, the tool starts identification of defect by the identifying the dependent nature of programming entities and its dependencies over other sub entities. Any Programming entity used in the program is dependent on other entities which may result in appropriate or in-appropriate results of its dependent entities. For example, Consider the task of testing a procedure that to find the roots of quadratic equation. The quadratic equation in the form of ax2+bx+c=0 the roots of the quadratic equation is given by formula $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$ given quadratic equation, the roots are either real and equal,(or) the roots are either real and distinct, (or) roots are imaginary. Figure-1 shows the program of quadratic equation:

```
    void main()
        {
1:          int a,b,c,d,e,f,g;
2:          printf(“Enter the three values”);
3:          scanf(“%d%d%d”,&a,&b,&c);
4:           d=((b*b)-(4*a*c));
5:           if(d==0)
            {
6:                  printf(“Roots are real and equal”);
7:                  f=-b/(2*a);
8:                   printf(“x1=%d\nx2=%d”,f,f”);
            }
9:          else if(d>0)
            {
10:                  printf(“Roots are real and distinct”);
11:                  e=sqrt(d);
12:                 f=(-b+e)/(2*a);
13:                 g=(-b-e)/(2*a);
14:                 printf("x1=%d\nx2=%d",f,g);
          }
15:        else
          {
16:                printf("Roots are imaginary");
17:                d=-d;
18:                 e=sqrt(d);
19:                f=-b/(2*a);
20:                g=e/(2*a);
21:        printf("x1=%d+%d\nx2%d-%d",f,g,f,g);
          }
          getch();
        }
```

Above program evaluates the roots of the given quadratic equation:

The above program is input to the Unit testing tool(CUnit), test cases are automatically generated by the tool. These test cases together are called as test suite.[6] Tool selects a test case randomly from the test suite and executes the test case. If it gives the expected results then the test case is pass, otherwise it is failed. We have to identify the root cause of a failing test input which results in a failure. In this current approach, a tool is built which automatically builds corresponding tree for a given program and easily notify the root cause of the defect, using a procedure discussed below.

4. PROCEDURE TO BUILD ENTITY TREE:

For example, the above program shows the quadratic equation program which is input to a Unit testing tool, then it randomly generates a test case and run. Suppose it is failed at statement12 then the expression is given below. f = (-b+e)/(2*a);//test case is failed at this statement. The statements relevant to the failed statements are given below:

d=((b*b)-(4*a*c));

e=sqrt(d);

f=(-b+e)/(2*a);

The tool automatically verifies which level of testing should be done. If a level1 testing is selected this tool performs one step tracing and verifies the statements in the code. Here above expression contains three variables i.e. b, e and a. The actual 'f' value is affected by these variables (sub divided into integrative sub-components), so at this point we have to verify these three variables. This tool generates the tree for the above expression as shown in Figure-2

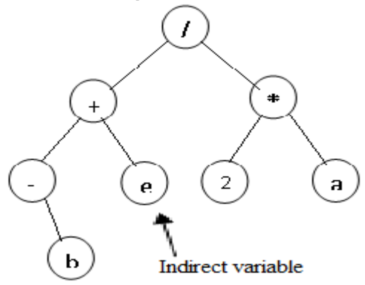

Figure-2 shows level-1 testing of expression (-b+e)/(2*a).

The above tree contains three variables, in which 'b','a' are direct variables and 'e' is indirect variable. so we have to verify the direct and indirect variables. If these variables return the correct values then fault in L-value otherwise fault in R-value.

The faults in R-value can be of three types which are as follows:

1. Faults in direct variables: Faults in direct variables are only caused by giving the wrong inputs by the user or by assignment of constants.
2. Faults in indirect variables: The second type faults occur due to previous wrong assignment values to the indirect variables or expressions.
3. Faults in functions: The third type faults occur due to the functions which are either standard functions or user-define functions.
   - Functions return wrong.
   - Parameters are either direct variables or indirect variables (reference variables).

In the above experiment for these wrong assignment values, a level-2 testing is required. This tool generates tree for level-2 testing of expression is shown in Figure 3.

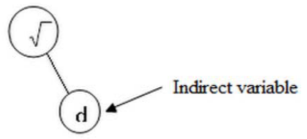

Figure-3 shows level-2 testing of expression sqrt(d).

To find wrong assignment values the tool generates tree for level-3 testing of expression is shown in figure-4.

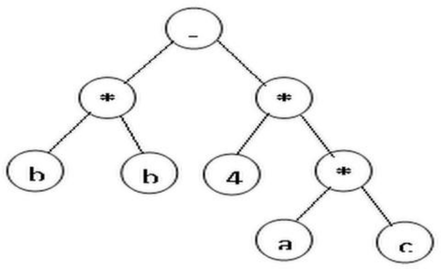

Figure-4 shows level-3 testing of expression (b*b)-(4*a*c)

Procedure Main:

Step 1: Generate Test cases for the code to be tested, by creating a test suit, along with expected results for the corresponding steps.

Step 2: Select a Random Generation tool (CUnit) for selecting a test case to be tested.

Step 3: Execute the test case ,record the result in form of logs, and perform the following

a. If the test case executes correctly, the program terminates successfully, resulting in expected result.

b. If the test case fails at a particular condition, the asserted condition fails, and logs are recorded so far and procedure *Level-Defect-Verification* is invoked.

Step 4: Terminate instantaneously.

Procedure Level-Defect-Verification [7]:

Step 1: Split the aborted statement of the program to integrative sub-components.

Step 2: Identify the individual programming entities from subcomponents which can be one of the following

- Direct variable
- Indirect variables
- Functions

Step 3: Perform the following steps until End-of-File is encountered for the log file (assertion logs).
Step 4: Generate the Tracing tree for Integrative Subcomponents of the statement encountered in Step1.
Step 5: For the sub tree identify the corresponding sub entities, because the sub tree may again comprise of entities which may be an one of the following of the form in Step 2.
Step 6: If absurd values are identified at the given step, then verification has identified an error and it needs to be corrected, else proceed and repeat the generation of tracing tree from Step 4.
Step 7 : Return to main function.

Procedure level-defect-prediction:
Step 1: Takes the tested identifiers as input from the procedure Level-Defect-Verification.
Step 2: Builds table which consists of tested identifiers and their properties.
Step 3: The table is manually or automatically filled according to their conditions.
Step 4: Next , it applies any one of the above algorithm to calculate error rate of each and every attribute.
Step 5: After calculating error rate, it predict which one is occurring most frequently and returns to main.

Defect Prediction: Defect prediction means predicting the defects in a simple program or module or a project or a software component [8]. After constructing a table for datasets as described above, a learning algorithm is selected. Then this module selects an algorithm, builds a prediction model and predict software defect. It is very useful for improving the generalization ability of the predictor. After the predictor is built, it can be used to predict the defect proneness of new software components.

For defect prediction, we are using two algorithms as described above. First, it generates the table as shown below.

Items	Range	Dtype	Result
e	Within	Equal	true
e	Within	Ntequal	False
e	Within	Ntequal	True
e	Out of	Equal	True
e	Out of	Ntequal	False
b	Within	Equal	true
b	Within	Ntequal	False
b	Within	Ntequal	True
b	Out of	Equal	True
b	Out of	Ntequal	False
a	Within	Equal	true
a	Within	Ntequal	False
a	Within	Ntequal	True
a	Out of	Equal	True
a	Out of	Ntequal	False
c	Within	Equal	true
c	Within	Ntequal	False
c	Within	Ntequal	True
c	Out of	Equal	True
c	Out of	Ntequal	False
d	Within	Equal	true
d	Within	Ntequal	False
d	Within	Ntequal	True
d	Out of	Equal	True
d	Out of	Ntequal	False

If we apply the OneR algorithm as discussed in the section 2.1 for above table, the following steps are generated.
Step 1: It calculates total number of true and false cases for every attribute as shown below:

Attributes	True	False
E	3	2
B	3	2
A	3	2
C	3	2
D	3	2

Attributes	True	False
Within	10	5
Out of	5	5

Attributes	True	False
Equal	10	0
Ntequal	5	10

Step 2: Then, it calculates the error rate and net calss weight for every attribute.
Attribute = items total error rate = 0.40 NCW = 1.50
Attribute = range total error rate = 0.40 NCW = 1.1667
Attribute = dtype total error rate = 0.20 NCW = 1.1667
Step 3: Finally, it selects the attribute with the least error rate. OneR algorithm selects dtype as OneRule [9].
By using this approach we can predict defects that occur more frequently in a program.

If we apply the Naive Bayes algorithm as discussed in the section 1.1 for above table, the following steps are generated.
Step 1: First it calculates total number of true and false cases for every attribute.

For data items:

Attribute	True	False
e	3	2
b	3	2
a	3	2
c	3	2
d	3	2

For range items:

Attributes	True	False
Within	6	3
Out of	3	3

For dtype items:

Attributes	True	False
Equal	6	0
Ntequal	3	6

Step 2: Then, it calculates the total probability for each and every attribute in true and false cases.

Total probability for data items in case of true and false:

Items	True	False
e	0.20000	0.20000
b	0.20000	0.20000
a	0.20000	0.20000
c	0.20000	0.20000
d	0.20000	0.20000

Total probability for range items in case of true and false:

Range	True	False
Within	0.66667	0.50000
Out of	0.33333	0.50000

Total probability for dtype items in case of true and false:

Dtype	True	False
Equal	0.66667	0.00000
Notequal	0.33333	0.00000

Naive Bayes algorithm [10] considers only false cases as described above in section 2.2. From the above all tables we can conclude that it selects not equal as error as it contains least probability.

Naive Bayes selects not equal as error.
By using this approach we can predict defects and also the reason for the defect that occurs frequently in a program.

5. Results

The results show that we should choose different learning schemes for different data sets (i.e., no scheme dominates),and last, that our proposed framework is more effective and less prone to bias than previous approaches. In Naive Bayes approach, we can estimate or predict the defects based on artifacts i.e., more than one artifact. In oneR algorithm, it tells only what type of defect frequently occurs whereas in Naive Bayes approach, it tells the reason for the defect that occurs frequently. Both the algorithms are used for defect prediction only.

6. Conclusion and future work

The current work is in its primitive stages, still it needs to be endorsed for various syntactic constructs, needs to be verified with various programming paradigms for its
applicability, but one thing this approach does provide is clarity for developer for unit testing and its depth based on its entities which may be used in maintenance projects ,where a part of code added needs to be verified for its exactness. Also defect prediction mechanism has extended to modules, projects and software components.

References

[1]. Kent beck, test-driven development by example, addison wesley, **2002**.

[2]. Roy osherove, the art of unit testing with examples in .net, manning publications, **2009**.

[3]. C.nebut, "automatic test generation: a use case driven approach" , ieee transactions on software engineering, vol. 32, no. 3,pp**140-156**, march **2006**.

[4]. S.vegas, "maturing software engineering knowledge through classifications: a case study on unit testing techniques" , ieee transactions on software engineering, vol. 35, no 4,pp-**551-556**, july/aug-**2009**.

[5]. Ciupa, ilinca, et al. "experimental assessment of random testing for object-oriented software." **2007**.

[6]. Oriat, catherine. "jartege: a tool for random generation of unit tests for java classes." quality of software architectures and software quality **(2005)**: **242-256**.

[7]. Narendra kumar rao, b., a. Rama mohan reddy, and k. Ravi. "level dependencies of individual entities in random unit testing of structured code."electronics computer technology (icect), 2011 3rd international conference on. Vol. 6. Ieee, **2011**.

[8]. Qinbao song, zihan jia, martin shepperd, shi ying, and jin liu, "a general software defect-proneness prediction framework" ieee transactions on software engineering, vol. 37, no. 3, may/june **2011**.

[9]. Buddhinath, gaya, and damien derry. "a simple enhancement to one rule classification." department of computer science & software engineering. University of melbourne, australia (**2006**).

[10]. Radlinski, lukasz. "a survey of bayesian net models for software development effort prediction." international journal of software Engineering and Computing2.2 (**2010**): **95-109**.

AUTHORS PROFILE

Mr. B. Narendra Kumar Rao, obtained Bachelor Degree in Computer Science and Engineering from University of Madras, M.Tech in Computer Science from JNTU, Hyderabad and at present pursuing Ph.D. He has more than 10 years of experience in Area of Computer Science and Engineering which includes four years of Industrial Experience and six years of Teaching Experience. Research interests include Software Engineering and Embedded Systems. Currently he is working as Associate Professor in Department of Computer Science and Engineering at Sree Vidanikethan Engineering College.

Dr. A. Rama Mohan Reddy, obtained his Bachelor Degree in Mechanical Engineering and Master's degree in Computer Science Engineering from NIT Warangal and Ph.D degree from Sri Venkateswara University and at present working as an Professor in Department of Computer Science and Engineering, Sri Venkateswara University College of Engineering. His areas of interest include Software Architecture and data mining. He has more than 27 years of experience in teaching and research.

Mr.B.Bhaskar Kumar Rao,obtained his bacheolar in B.Tech Computer Science and Engineering from JNTU,Anantapur.He is currently pursuing his Post Graduation from SRM University,Chennai.His research areas include Software Engineering and Cloud Computing.