VOL. 10, NO. 17, SEPTEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2015 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

MACHINE LEARNING BASED METHODOLOGY FOR TESTING OBJECT ORIENTED APPLICATIONS

N. Kannadhasan and B. Uma Maheswari
Department of Master of Computer Applications St. Joseph's College of Engineering, Chennai, India
E-Mail: kannadhasan2010@gmail.com

ABSTRACT

The objective of software testing is to find the maximum number of errors as early as possible in the software development phase. Software testing ensures the quality of the source code and testing authenticates the incorrectness of the program. The detection of flaws in the C++ source code which leads to logical errors are identified. The code is improved by automatically detecting the defects which are not detected by the compiler. The users are relieved from the liability of detecting these defects. These frameworks of defects are automatically extracted from the C++ source code by analyzing the object oriented applications. The piece of code contains defect are found and feature values are assigned for training the system. The feature values of defects and non-defect programs are converted into attribute relation format file which constitutes the data set. Machine learning algorithm was used to classify the fault and non-fault statements in the object oriented applications.

Keywords: object oriented applications, machine learning, ADTree, training, fault prediction, WEKA.

INTRODUCTION

Faults may occur in any circumstances of the program which causes malfunctioning and prevents the realization of expected or previously specified results. All the translators are not furnished to identify all kinds of faults. A C++ code snippet can be tested to improve the confidence by revealing probable flaws or deviations from user`s requirements. In this paper various rules were developed which automatically detects the faults C++ programs. Machine learning algorithms were used to detect the faults.

Fault identification in a program begins at the lowest level with unit testing, and progresses towards the integration testing where the units are tested together to ensure that they interacts correctly. The various tools for testing object oriented programs and various testing techniques are presented. The paper is organized as follows: The various existing survey is discussed in section 2, machine learning based methodology for testing the object oriented applications is explained in Section 3, the results discussions are illustrated in Section 4 and the Section 5 Concludes the proposed work.

EXISTING WORKS

Software testing techniques are classified in to black box and white box categories. Black box testing determines whether the program meets its functional and non-functional requirements without regard to its implementation. Functional test cases are constructed based on the specification of the unit. Black box testing is achieved by some ways like boundary value analysis and equivalence class partitioning and graph based testing

White box testing examines the procedural details of the unit. It tests the statements, branches, control flows and dataflow of the source code without any reference to the specification. Coverage criteria are applied to determine the code coverage of the unit. Basis path testing and control structure testing are some of the white box testing methods. Basis path testing is performed using the flow graph. McCabe's cyclomatic complexity is used to find the upper bound of the test to be conducted over the software. Condition is testing, data flow testing and loop testing fall under control structure testing. These traditional testing techniques are not adequate for testing the object oriented software.

Extensible Markup Language (XML) helps data exchange in business applications. Verifying the accuracy between these objects is done through object mapping [1]. The discussed works detect defects which lead to logical and execution errors in object oriented languages. The author [2] performed static testing for finding the inheritance related bugs and property errors in the product. Code analysis extracts the code for testing, and data flow testing was based on the bug analysis of the program.

Bayesian network model [3] relates object oriented software metrics to software fault content and proneness. They addressed the internal product and external quality metrics. Wrong usage of formal parameters lead to logical errors. The author [5] identified defects in hybrid inheritance, runtime polymorphism, member function call, conditional statement and function overriding. Also dangling reference problem was addressed and missing address and new operators were identified in C++ applications.

Data flow testing was performed by [6] using the state based testing. Various algorithms [7] were developed for the detection of defects in GUI applications. Eclat tool generates the test cases and classifies the test inputs in the JAVA program [8]. The author [9] represents the actions of the GUI applications as a tree and each sub-tree denotes one or more test cases. A group of strongly related components are macro components which have high level operations.

The user interface testing tool (UITT) was developed using GUI testing modeling language (GTML) based on Extended Backus Naur Form (EBNF) grammar, for testing JAVA swing based applications. They tested the swing, classes, visual macro model editor and word processor components.

The relationship between object oriented metrics, the prediction accuracy of the machine learning classifier and fault detection on open source software [10] has been performed. Algorithms were designed [11] to identify the defects in graphical user interface widgets. Machine learning is the application of algorithms which makes the machine to learn through experience. Machine learning algorithms [12-13] are used in complicated domains where human effort is insufficient.

The supervised and unsupervised learning algorithms are used for classifying the complex data. The unsupervised learning algorithms do not use class labels for training. The detailed survey [14] explains the various machine learning based testing. Supervised machine learning models are trained with class labels. The authors [15] used various machine learning algorithms to reveal the faults.

MACHINE LEARNING BASED OBJECT ORIENTED TESTING

To overcome the problems in the existing system machine learning based approach was used for finding the faults in the object oriented applications. Various object oriented programs were collected from the bench mark suite.

All the C and C++ programs were analyzed for faults which are not identified [5] by the compiler. The use case diagram for identifying the faults in object oriented applications is shown in Figure-1.

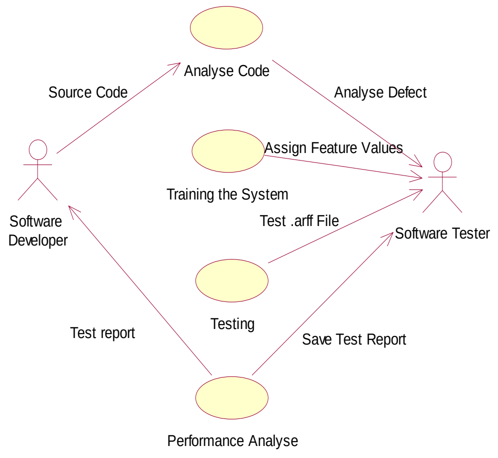


Figure-1. Use Case diagram for identifying the faults in object oriented applications.

Finite automata and regular expressions were used to analyze the source code of the object oriented applications. Code constructs are evaluated for finding the violations. Feature values are assigned to the code fragments which leads to logical errors undetected by the compiler. The construction of the feature values with the class label is used to train the machine learning based testing system. The machine learning tool kit WEKA, data mining algorithm is used to train the fault prediction system.

Training the Fault Prediction System

Machine learning algorithms such as data pre-processing, classification, regression, clustering, association rules, and visualization are used in data mining. To find the faults in the object oriented programs data pre-processing and classification algorithms are used in the proposed work. All the attributes are preprocessed in the training instances.

Attribute Table is a collection of data set for applying data mining tasks. The Table-1 describes features, feature values and the result of the prediction system for identifying the faults. If non-fault was identified from the source code, “correct” is the feature value for the “result” attribute, Otherwise if fault is identified then “Fault” is the feature value of the for the “result” attribute which is the target label of the machine learning system.

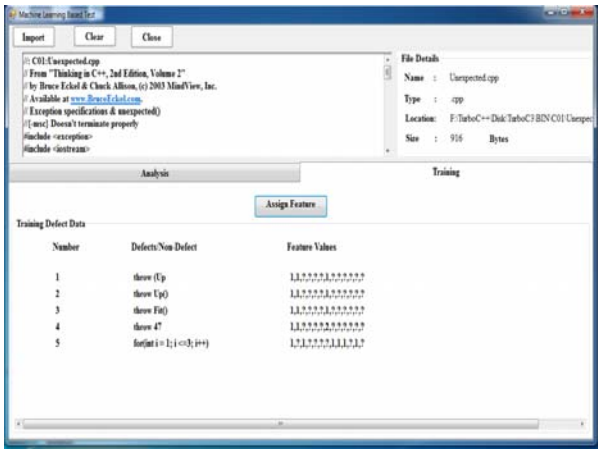
Table-1. Feature values and the class label for the training data set.

Program Constructs	Syntactic Features	Feature Value	Result
Array	Variable	1	Correct
	Open Bracket	1	Correct
	Positive Value	1	Correct
	Close Bracket	1	Correct
	Others	2	Fault
Char	Keyword	1	Correct
	Space	1	Correct
	Variable	1	Correct
	Equal	1	Correct
	String	1	Correct
	Others	2	Fault
Throw	Keyword	1	Correct
	Space	1	Correct
	String	1	Correct
	Others	2	Fault
If, while, for	Keyword	1	Correct
	Open Bracket	1	Correct
	Arg1	1	Correct
	Condition	1	Correct
	Arg2	1	Correct
	Close Bracket	1	Correct
	Others	2	Fault

Bench mark suite was taken to build the training data set for fault prediction system. The various program constructs such as Array, Character, Throw, If, While and for are analyzed for faults. The syntactic features for the various program constructs and their corresponding feature values are listed. The result attribute, which is the target label for the machine learning algorithm is listed in Table-1. These program constructs are analyzed for faults in their syntactic features such as keyword, space, open bracket, positive array values, variable, equal operator, string, arguments, conditional operators, close bracket and increment operators. The Table-2 is the sample of valid and invalid training instances to build the fault prediction system. The construction of the training data set is shown in Figure-2 which was developed using C#. The C# analyse system build the large training data set for finding the faults in the object oriented system. The algorithm Alternate Decision Tree (ADTree) is used to train the system.

Table-2. Training data set for the object oriented systems.

Program Constructs	Keyword	Space	Open bracket	Positive value	Variable	Equal	String	Arg1	Condition	Arg2	Close Bracket	Increment	Result
Array	?	?	1	1	1	?	?	?	?	?	?	1	Correct
Array	?	?	1	2	1	?	?	?	?	?	?	1	Fault
Char	1	1	?	?	1	1	1	?	?	?	?	?	Correct
Char	1	1	?	?	1	1	2	?	?	?	?	?	Fault
Throw	1	1	?	?	?	?	1	?	?	?	?	?	Correct
throw	1	1	?	?	?	?	2	?	?	?	?	?	Fault
If	1	?	1	?	?	?	?	1	1	1	?	1	Correct
If	1	?	1	?	?	?	?	1	2	1	?	1	Fault
while	1	?	1	?	?	?	?	1	1	1	?	1	Correct
while	1	?	1	?	?	?	?	1	2	1	?	1	Fault
For	1	?	1	2	1	?	?	1	1	1	1	1	Correct
For	1	?	1	1	2	?	?	1	1	1	1	1	Fault



```
//: C01:Unexpected.cpp
// From "Thinking in C++, 2nd Edition, Volume 2"
// by Bruce Eckel & Chuck Allison, (c) 2003 MindView, Inc.
// Available at www.BruceEckel.com.
// Exception specifications & unexpected()
//{-msc} Doesn't terminate properly
#include <exception>
#include <iostream>
```

File Details

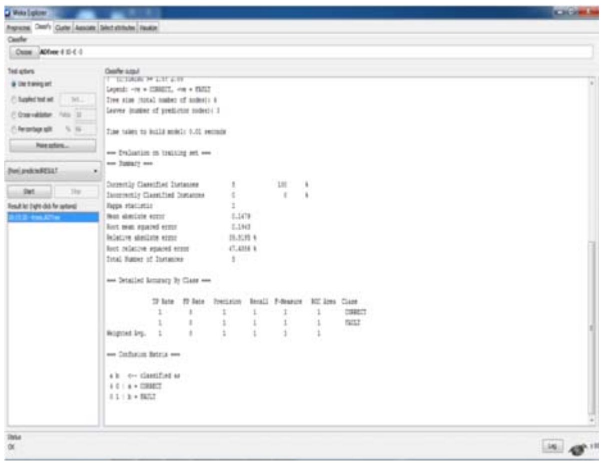
Name : Unexpected.cpp
Type : .cpp
Location: F:\TurboC++\Disk\TurboC3\BIN\C01\Unexpec
Size : 916 Bytes

Training Defect Data

Number	Defects/Non-Defect	Feature Values
1	throw (Up	1,1,?,?,?,?,1,?,?,?,?,?,?
2	throw Up()	1,1,?,?,?,?,1,?,?,?,?,?,?
3	throw Fit()	1,1,?,?,?,?,1,?,?,?,?,?,?
4	throw 47	1,1,?,?,?,?,2,?,?,?,?,?,?
5	for(int i = 1; i <=3; i++)	1,?,1,?,?,?,?,1,1,1,?,1,?

Figure-2. C # Fault analysis system.

The result model of the machine learning classifier is shown in Figure-3.



ADTree -B 10 -E -3

```
Legend: -ve = CORRECT, +ve = FAULT
Tree size (total number of nodes): 4
Leaves (number of predictor nodes): 3

Time taken to build model: 0.01 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances        5          100    %
Incorrectly Classified Instances      0            0    %
Kappa statistic                       1
Mean absolute error                   0.1479
Root mean squared error               0.1963
Relative absolute error              35.3135 %
Root relative squared error          47.4356 %
Total Number of Instances             5

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0         1          1        1          1        CORRECT
                 1         0         1          1        1          1        FAULT
Weighted Avg.    1         0         1          1        1          1

=== Confusion Matrix ===

 a b   <-- classified as
 4 0 | a = CORRECT
 0 1 | b = FAULT
```

Figure-3. ADTree machine learning classifier.

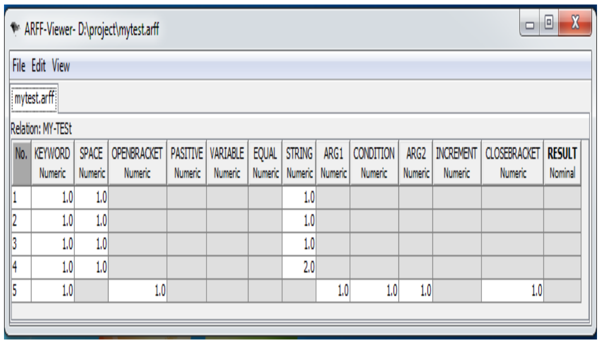
Experimental Results

The untrained Program under Test (PUT) is applied to the fault prediction system. The attribute file format is constructed for the prediction system. The class label is set as "?" for the testing instances. This testing instance is applied to the training data set. The attribute file format of the program under test shown in Figure-4.

```
@relation train-test
@attribute keyword numeric
@attribute space numeric
@attribute open bracket numeric
@attribute positive numeric
@attribute variable numeric
@attribute equal numeric
@attribute string numeric
@attribute arg1 numeric
@attribute condition numeric
@attribute arg2 numeric
@attribute increment numeric
@attribute close bracket numeric
@attribute result{correct, fault}
@data
1,1,?,?,?,?,1,?,?,?,?,?,?
1,1,?,?,?,?,1,?,?,?,?,?,?
1,1,?,?,?,?,1,?,?,?,?,?,?
1,1,?,?,?,?,2,?,?,?,?,?,?
1,?,1,?,?,?,?,1,1,1,?,1,?
```

Figure-4. Attribute file format of testing data sample.

The ARFF view of the test data of program under test is shown in Figure-5. The attributes which are not available and the class label are marked as "?", hence the corresponding slots are empty in the ARFF view.

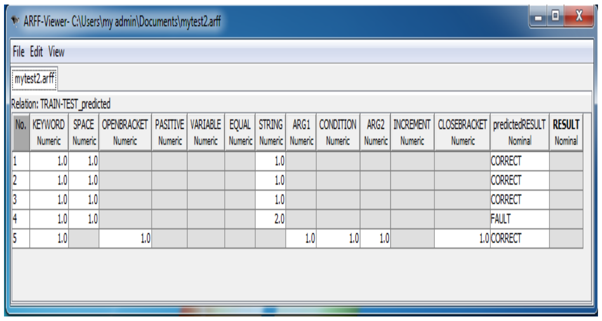

ARFF-Viewer- D:\project\mytest.arff

Relation: MY-TESt

No.	KEYWORD Numeric	SPACE Numeric	OPENBRACKET Numeric	PASITIVE Numeric	VARIABLE Numeric	EQUAL Numeric	STRING Numeric	ARG1 Numeric	CONDITION Numeric	ARG2 Numeric	INCREMENT Numeric	CLOSEBRACKET Numeric	**RESULT** Nominal
1	1.0	1.0					1.0						
2	1.0	1.0					1.0						
3	1.0	1.0					1.0						
4	1.0	1.0					2.0						
5	1.0		1.0					1.0	1.0	1.0		1.0	

Figure-5. ARFF view of test data of program under test.

The designed fault prediction system evaluate and analyze these testing instances and detect the faults and correct program statements using the training data set. The ARFF View of predicted faults is shown in the Figure-6. The "predictedResult" attribute shows the classification of program constructs in the object oriented programs.


ARFF-Viewer- C:\Users\my admin\Documents\mytest2.arff

Relation: TRAIN-TEST_predicted

No.	KEYWORD Numeric	SPACE Numeric	OPENBRACKET Numeric	PASITIVE Numeric	VARIABLE Numeric	EQUAL Numeric	STRING Numeric	ARG1 Numeric	CONDITION Numeric	ARG2 Numeric	INCREMENT Numeric	CLOSEBRACKET Numeric	predictedRESULT Nominal	**RESULT** Nominal
1	1.0	1.0					1.0						CORRECT	
2	1.0	1.0					1.0						CORRECT	
3	1.0	1.0					1.0						CORRECT	
4	1.0	1.0					2.0						FAULT	
5	1.0		1.0					1.0	1.0	1.0		1.0	CORRECT	

Figure-6. Predicted result of program under test.

The tree view of the machine learning classifier and the Region of Characteristics (ROC) curve is shown in Figures [7-8].

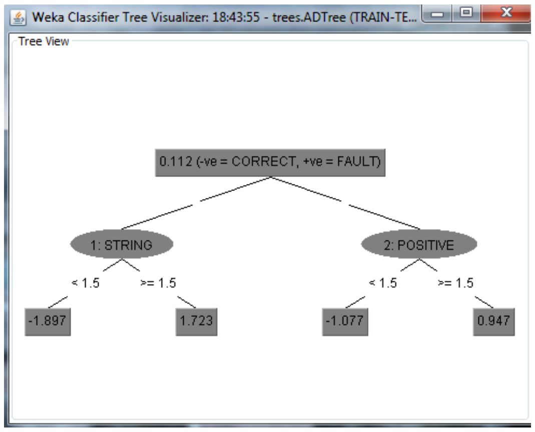

Weka Classifier Tree Visualizer: 18:43:55 - trees.ADTree (TRAIN-TE...
Tree View
0.112 (-ve = CORRECT, +ve = FAULT)
1: STRING
2: POSITIVE
< 1.5
>= 1.5
< 1.5
>= 1.5
-1.897
1.723
-1.077
0.947

Figure-7. Tree of the program under test.

The analysis the performance of the data set classification by the ADTree machine learning classifier is shown in Figure-8. That visualization of the attributes and based on the data values is displayed as Threshold Curve.

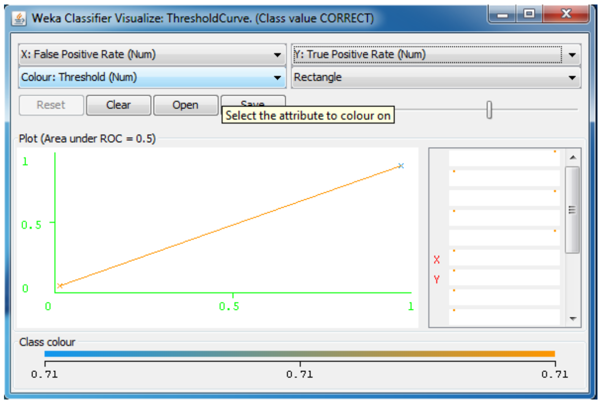


Figure-8. ROC of the data set.

CONCLUSIONS

The objective has been achieved by detecting defects in C++ source code which leads to logical error. The proposed fault prediction system classifies the program constructs in to correct and fault statement which is not detected by the compiler. Traditional object oriented testing methods takes more time to detect the faults if the Lines of Code (LOC) is very high. This issue is resolved, while applying the machine learning algorithms with the existing system. An attempt has made to reduce the defect detection rate.

REFERENCES

[1] Raha, D and Jadhav, MK 2008, ‘Automation method for testing XML/DB/XML layers’, Proceedings of IEEE Conference on Software Testing, Verification and Validation. pp. 458-464.

[2] Geetha, BG, Palanisamy V, Duraiswamy K and Singaravel G. 2008, ‘A tool for testing of inheritance related bugs in object oriented software’, Journal of Computer Science. vol. 4, no. 1, pp. 59-65.

[3] Pai, GJ and Bechta Dugan, J 2007, ‘Empirical analysis of software fault content and fault proneness using Bayesian methods’, IEEE Transactions on Software Engineering. vol. 33, no. 10, pp. 675-686.

[4] Harrold M.J and Rothermal G. 1994. Performing Data Flow Testing on Classes’, ACM Software engineering Notes. Vol 19, No. 5, pp. 154-163.

[5] Sarala, S and Valli, S 2004, ‘A tool to automatically detect defects in C++ programs’, Proceedings of International Conference on Information Technology, pp. 302-314.

[6] Tsai, B. Y., Stobart, S., Parrington, N., and Mitchell, I. 1999, ‘A state-based testing approach providing data flow coverage in object-oriented class testing’, International Conference on Advantage Science and Technology. pp. 1-18.

[7] Uma Maheswari, B and Valli, S 2011, ‘Algorithms for the Detection of Defects in GUI Applications’, Journal of Computer Science. vol. 7, no. 9, pp. 1343-1352.

[8] Pacheco, C, and Ernst, MD 2005, ‘Eclat: Automatic generation and classification of test inputs’, Proceedings of nineteenth European Conference on Object Oriented Programming. pp. 504-527.

[9] Chen, WK, Shen, ZW and Chang CM, 2008, ‘GUI test script organization with component abstraction’, Proceedings of IEEE conference on secure system integration and reliability improvement. pp. 128-134.

[10] Malhotra, R and Singh, Y 2011, ‘On the applicability of machine learning techniques for object oriented software fault prediction’, Software Engineering: An International Journal, vol. 1, no. 1, pp. 24-37.

[11] Uma Maheswari, B and Valli, S 2013, ‘Algorithms for Detecting Defects in User Interface Widgets’, European Journal of Scientific Research. vol. 94, no. 2, pp. 261-272.

[12] Mitchell T. 1997, ‘Machine learning’, First Edition, McGraw Hill, New York.

[13] Nilsson NJ 1997. Introduction to machine learning’, Available from <http://ai.standford.edu/~nilsson/mlbook.html>.

[14] Uma Maheswari, B and Valli, S. 2014, ‘Survey on Graphical User Interface and Machine Learning based Testing Techniques’, Journal of Artificial Intelligence. vol. 7, no. 3, pp. 94-112.

[15] Di Stefano, JS, and Menzies, T. 2002. ‘Machine learning for software engineering: Case studies in software reuse’, Proceedings of IEEE International conference on Tools with Artificial Intelligence. pp. 246-251.