ISSN 2047-3338

A New Hybrid Model for Predicting Change Prone Class in Object Oriented Software

Deepa Godara[1] and R.K. Singh[2]
[1]Computer Science Engineering, Uttarakhand Technical University, Dehradun, India
[2]Electronic and Communication Engineering, Uttarakhand Technical University, Dehradun, India
[1]deepa.fet@mriu.edu.in, [2]rkgec@gmail.com

***Abstract*— In the field of software engineering, during the development and maintenance of software, the information on the classes which are more prone to be changed is very useful. Developers can make more flexible software by modifying the part of classes which are more sensitive to changes.The incessant changes effected in software every day have assumed such an alarming proportion causing untold and unimagined paradoxes that it is highly essential to initiate instant and immediate steps to balance this blitz. It does not mean that no endeavor has been made in the bye-gone era to tackle the issue. In fact, several methods to solve this dilemma were introduced in the past by predicting the changes in the software. To handle this problem with an eye on good prediction accuracy, a new hybrid model is introduced in our paper. Our proposed model combines features such as behavioral dependency generated from UML diagrams, execution time and trace events generated from source code to predict change prone class. The rationale behind this approach is that in a well designed software system feature enhancement or corrective maintenance should affect a limited amount of existing code.**

***Index Terms*– UMl Diagrams, Change Prone Class and Behavioral Dependency**

I. INTRODUCTION

CHANGE-PRONE classes in software require more attention because they require increased effort, development and maintenance costs. Identifying such classes can enable developers to focus preventive actions such as, peer-reviews, testing, inspections, and restructuring efforts on the classes that are sensitive and change prone. As a result, developers can deliver higher quality products in a timely manner by efficiently utilizing the resources.

Software systems are incessantly subjected to modification. This is all the more essential for addition of novel traits, to become accustomed to contemporary environment, to put right bugs or to re-factor the source code [4]. Modification may be on account of diverse factors like improvement, modification, perfect upkeep or do away with drawbacks. Several elements of the software may be susceptible to modifications than their counterparts. A proper understanding of the classes which are change-prone is highly advantageous as the change-proneness may be some sign of particular fundamental quality issues [3]. Managing change is one of the pivotal factors in the realm of software engineering. Evolutionary growth has been suggested as a competent method to tackle risks like modern technology and vague or varying needs [15]. If a preservation method has the competence to ascertain the components of the software which are change-prone then explicit corrective steps can be initiated. Therefore, a sound knowledge of the domain which had maximum modification over a certain interval will go a long way in spotting the crucial change-prone classes and interactions, then it is easy for development procedure to concentrate its attention on them [3].

Change-prone classes in software call for meticulous attention as they need endeavor and augment growth and preservation overheads. Recognizing and typifying them enables developers to take remedial measures like peer-reviews, testing, inspections, and streamlining endeavors on the classes with the parallel traits in the coming years. Consequently, developers are capable of exploiting their wherewithal more professionally and dispense superior quality products in an appropriate way [9]. If defective classes are recognized in the initial stages of the growth project's life phase, extenuating remedies can be considered including alert inspections. Forecast patterns by means of plan metrics can be employed to recognize defective classes in advanced stages [13]. The accuracy of the forecast effect decides the accuracy of cost evaluation and quality of project preparation [14].

UML is now extensively acknowledged in the software engineering circle as a general notational benchmark. It extends a helping hand to object-oriented plans which on the other hand promote module reprocess. It is competent to furnish numerous outlooks of the system under blueprint [6]. The UML based plan enables us to execute prescribed authentication and corroboration method [5]. The unified modeling language (UML) is a graphical language for visualizing, denoting, building, and recording software-intensive techniques. UML offers a typical method of scripting system plans, covering abstract things, classes written in a definite programming language, database schemes and reusable software components [2]. UML has appeared assuming the role of the software industry’s leading language and is, by now, an Object Management Group (OMG) benchmark. It symbolizes a set of finest engineering exercises that have been established as triumphant in the modeling of mega and intricate techniques. OMG is now recommending the UML pattern for global homogeny for information technology [8]. As the application of object-oriented plan and programming grows up in the industry, we see that legacy and polymorphism are being utilized more often to perk up internal reprocess in a system and to assist conservation [12].

The rest of this paper is organized as follows. Section II presents related works. Section III presents our new model for predicting change prone classes. Section IV presents results, and finally, Section V concludes the paper.

II. RELATED WORK

Change-proneness prediction is associated with change impact analysis. The former predicts which classes are likely to change in the future (i.e., change over successive versions), whereas the latter predicts which classes may be impacted by a given change. A brief review of some of the recent researches is given below.

In the course of the growth and preservation of object-oriented (OO) software, the data on the classes which are more prone to change is highly advantageous. Developers and maintainers are able to create further adaptable software by changing the segment of classes which are susceptible to modifications. Conventionally, nearly all change-proneness forecast has been investigated according to source codes. Nevertheless, change-proneness forecast in the initial stage of software growth can offer an easier method for evolving durable software by changing the existing plan or selecting substitute plans prior to execution. To tackle this requirement, Ah-Rim Han *et al.* [16] have offered an innovative and a systematic method for estimating the behavioral dependency measure (BDM) which enables proper forecast of change-proneness in UML 2.0 brand. The anticipated measure has estimated on a multi-version medium size open-source project namely JFreeChart. The outcomes clearly exhibited the fact that the BDM is a functional pointer and is competent to be harmonizing to current OO metrics for change-proneness forecast.

The estimation of change-proneness of components of a software mechanism is an energetic theme in the arena of software engineering. Such evaluation may be profitably used to forecast modification to diverse classes of a system from one version to the next. Ali R. Sharafat and Ladan Tahvildari [17] have come out with a novel method to forecast modifications in an object-oriented software mechanism.

The key dilemma in software growth procedure is to evolve inaccuracy recognition to initial stages of the software life span. With this end in view, the Verification and Validation (V&V) of UML diagrams undertake a very significant function in identifying defects at the plan stage itself. It has a discrete relevance for software safety, where it is highly essential to spot safety faults before they can be subjugated. V. Lima *et al.* [18] have played a vital role in this regard by offering a formal V&V method for one of the most admired UML diagrams viz. sequence diagrams.

Mehdi Amoui *et al.* [19] have established that an awareness of probable time of occurrence of modifications will motivate managers and developers to design their preservation functions with superior proficiency. The dilemma is tackled by the invention of the innovative Neural Network-based Temporal Change Prediction (NNTCP) structure. This novel structure has successfully recognized the probable location of occurrence of such alterations called hot spots, and thereafter included the time dimension to forecast the probable time of occurrence of the changes concerned. Premature detection of error prone and alteration prone classes enables the developers and experts to utilize their precious time and resources on these zones of software. Malan V. Gaikwad *et al.* [20] have the credit of introducing a novel a method of employing class hierarchy technique which is easily comprehend-able and executable. Another significant fact was that they have not taken any data physically; it was assessed by that version only. The new model has successfully spotted change-prone classes and change-proneness of classes and fault-prone classes and fault-proneness of classes. Recognizing the change-prone and inaccuracy prone classes earlier can help concentrating interest on these classes. Malan V. Gaikwad *et al.* [21] have intelligent focused on locating reliance of software that may be obtained by assessing the proneness of Object Oriented Software. Two major kinds of proneness were linked with OO software namely Fault Proneness and Change Proneness. They have investigated the two methods, matrix and list method, employed for estimating the proneness and reliance of classes. In the earlier methods, all needed data was taken manually and from UML diagrams.

Recognizing change-prone classes enables developers to devote further interest to classes with parallel traits in the future and thus investigation resources and time can be utilized more efficiently. Xiaoyan Zhu *et al.* [22] have gathered a group of static metrics and modification data at class level from an open-source software product, Datacrow. Using this data, they have initially authenticate Pareto’s Law and observed that about 80% of the lines transformed are situated in only 20% of the classes. Moreover, Emanuel Giger *et al.* [24] have presented a paper for capturing the fine-grained Source Code Changes (SCC) and their semantics and also Ali R. Sharafat and Ladan Tahvildari [25] have proposed a novel method for the prediction of changes in object oriented software system, in which the quality aspects were qualified by the probability of change in each class.

III. NEW HYBRID MODEL FOR PREDICTING CHANGE PRONE CLASS

In our proposed work, the main intention is to develop an hybrid model for predicting change prone classes. The input of our proposed work is an application. In our work, we have to predict the change proneness of any given application. In order to predict, at first, the features of every application need to be collected from the application. After the identification of these features, the change proneness in each class can be predicted by using ID3 Decision tree algorithm. If a class is classified as predicted class for change proneness, then the value of the change proneness prediction is also calculated.

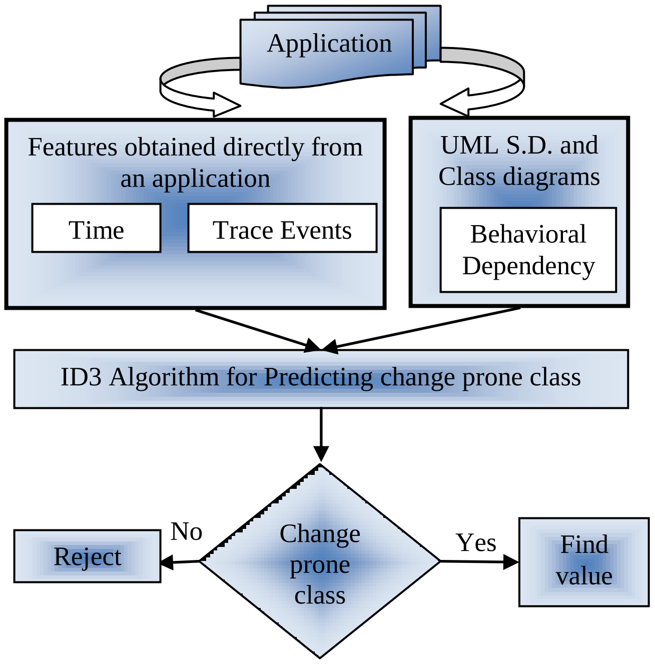
Application
Features obtained directly from an application
Time
Trace Events
UML S.D. and Class diagrams
Behavioral Dependency
ID3 Algorithm for Predicting change prone class
Change prone class
No
Yes
Reject
Find value

Fig. 1: Proposed Model for Predicting Change Prone class

Features used to predict change prone classes in object oriented software are:

i) Features obtained directly from an application
ii) Feature obtained from UML sequence and class diagrams

A) *Features obtained directly from an application*

Execution Time and Trace events can be directly obtained from any application.

Time: From the input source code, we can find this feature value, time. In source code, many classes and methods are presented. For a class, there are *n* numbers of methods included in it. Each will be called by other classes also. To get the time feature, we have to calculate the execution time of each methods of each class. This execution time of every function facilitates the time feature value, which is helpful in predicting change prone class.

Trace Events: Some of the methods in the source code are not executed in the run time. During runtime of the source code, we have to identify the following things:

(i) What are the methods executed during the runtime?
(ii) How many times, these methods are executed during the runtime?

Using these details, the feature trace event is computed directly from a given application

B) *Feature obtained from UML sequence and class diagrams*

UML Diagrams are one of the diagrams supporting our work, from which we can compute the features of the application. UML (Unified Modeling Language) is a general-purpose modeling language, which helps to represent the structure of complex software in a visual form, and utilize in software engineering. In general, 8 kinds of UML diagrams are generated. But for our purpose we need only 2 kinds of UML diagrams – (i) Sequence Diagram (ii) Class diagram

UML Sequence Diagram: - Sequence diagram is a kind of interaction diagram, which represents a sequence of interaction between the objects.

UML Class Diagram: They help to create graphical logical models of a system, further used to create the source code for the classes represented on the diagram. Class diagrams represent the relationship between classes and interfaces.

Behavioral Dependency: Behavioral Dependency is one of the important features to predict the change proneness. This feature is obtained from UML sequence and class diagrams. In a source code, if one class gets changes; it also affects the other class. It can be found only through the dependencies between the classes or objects.

The relationship between the sender object and receiver object is an example of the behavioral dependency, while sending a message between two objects. The changes in the class of the receiver object also affect the class of the sender object. The inheritance and polymorphism are also taken into account, during the measurement of behavioral dependency. The higher changes in behavioral dependency indicate the possibility of more changes to happen.

Two kind of behavioral dependencies are:

Direct Behavioral Dependency: Consider two objects O_1 and O_2. If O_1 wants services to be get from O_2, then a synchronous message is sent to O_2 and O_1 waits for a reply that received from O_2. This kind of dependency is called as direct behavioral dependency. This is denoted as, $O_1 \rightarrow O_2$.

Indirect Behavioral Dependency: Consider *n* objects $O_1, O_2, O_3 ..., O_n$. Indirect dependency between the objects O_1 and O_n is denoted as, $O_1 \Rightarrow O_n$, except $n = 2$ that represents direct behavioral dependency $O_1 \rightarrow O_2$. Because the indirect behavioral dependency is represented as $(O_1 \rightarrow O_2) \mapsto (O_2 \rightarrow O_3) \mapsto .... \mapsto (O_{n-1} \rightarrow O_n)$. Here, "$\mapsto$" indicates the External service request relation. For example, if an object O_1 needs a service from the object O_3 through O_2, then it is indicated as $(O_1 \rightarrow O_2) \mapsto (O_2 \rightarrow O_3)$.

C) *ID3 Algorithm for classifying the classes as change prone*

ID3 (Interactive Dichotomizer version 3) is a simple decision tree learning algorithm that is used for classification purpose. It builds decision trees based on greedy search in an up-down manner, which contains the set of nodes and edges that connect these nodes. The leaf nodes of the decision tree have class names and the non-leaf nodes are the decision nodes which take decision whether a particular class belongs to a class or not. Every non-leaf node corresponds to an input attribute and every edge to a possible value of that attribute. The decision nodes or the non-leaf nodes made an attribute test with every attribute at every tree node being a possible value of the attribute, through the given input sets. If the attribute classifies the given input sets perfectly, then the ID3 stops its process. Otherwise, it recursively operates on all the attributes in the decision tree to get the "best" attribute. The decision node to which the attribute goes from the non-leaf node is decided by the information gain metric of ID3. The resultant tree provides the classification of the samples given as the input to this tree.

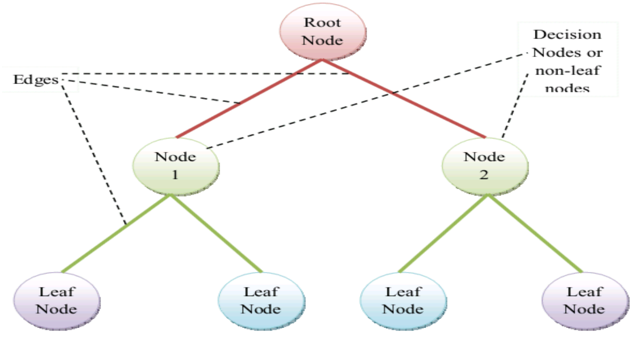

Fig. 2: General structure of decision tree diagram

The probability of a class belongs to PR is $\frac{a}{a+b}$ and the probability of a class belongs to NR is $\frac{b}{a+b}$. The intermediate information required to make the decision tree as a source of the result PR or NR is given as,

$$I(a,b)=\begin{cases} -\frac{a}{a+b}\log_2\frac{a}{a+b} - \frac{b}{a+b}\log_2\frac{b}{a+b}, & when \quad a,b \neq 0 \\ 0 & , \quad otherwise \end{cases}$$

The expected information EI needed for the decision tree with X as the root ($EI(X)$), is found as a weighted average as given below:

$$EI(X) = \sum_{i=1}^{N} \frac{a_i + b_i}{a+b} I(a_i, b_i)$$

IV. EXPERIMENTS AND RESULTS

Our proposed work for predicting change proneness is implemented using Java. Hospital Management application is given as the input for our proposed work. In this application, a large number of classes and methods are presented. These are taken for the process of our proposed work. The time and trace events are the two features that are directly obtained from the input Hospital Management application. The sample output for the time and trace events features are given in Table I.

Table I: Output for time and trace events

Classes	Feature - Time	Feature – Trace Events
a11	0.008	3
a12	0.001	3
a13	0.001	3
a20	0.0	2
Eight	0.001	1
Fifth	0.01	32

The next feature behavioral dependency is generated by converting the input into UML sequence and class diagram and then the feature is identified by constructing a model of Object behavioral dependency model and System object behavioral dependency model UML diagrams.

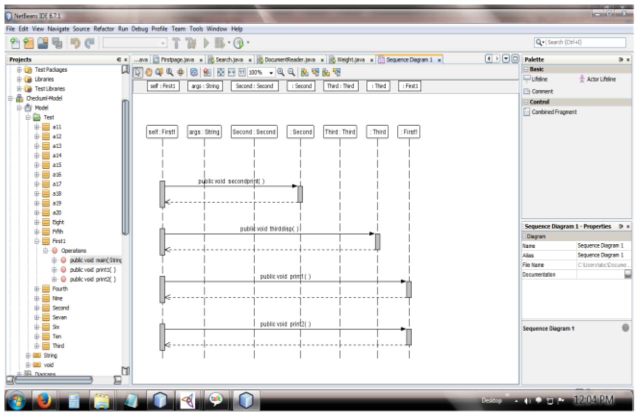

Fig. 3: UML sequence diagram

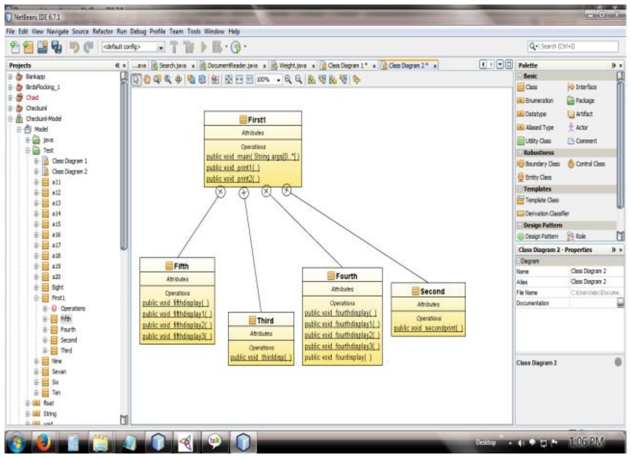
First1
Attributes
Operations
public void main(String args[0..*])
public void print1( )
public void print2( )
Fifth
public void fifthdisplay( )
public void fifthdisplay1( )
public void fifthdisplay2( )
public void fifthdisplay3( )
Third
public void thirddisp( )
Fourth
public void fourthdisplay( )
public void fourthdisplay1( )
public void fourthdisplay2( )
public void fourthdisplay3( )
public void fourdisplay( )
Second
public void secondprint( )
Class Diagram 2

Fig. 4: UML class diagram

After generating these UML SD and class diagrams, the process of identifying the behavioral dependency feature is carried out. Some of the behavioral dependency feature values for some classes are given in the following Table II.

Table II: Output for behavioral dependency feature values

Classes	Feature – Behavioral Dependency
a11	2.690476
a12	0.896825
a13	2.690476
a20	0.309523
Eight	0.246031
Fifth	0.198412

The output from the ID3 provides two classes based on a particular feature value. The output is to assess whether a particular class predicts the change-prone or not. The result of output of ID3 is given in the following Table III.

Table III: Output of ID3 algorithm for identifying the prediction of classes

```
if( obdm == "0.1984126984126984") {
        class = "no";
} else if( obdm == "2.6904761904761902") {
        class = "yes";
} else if( obdm == "0.8968253968253967") {
        class = "yes";
} else if( obdm == "0.3095238095238095") {
        class = "no";
} else if( obdm == "0.24603174603174602") {
        class = "no";
} else 0 Seconds
```

V. CONCLUSION AND FUTURE SCOPE

In this paper, an effective change proneness prediction system was effectively constructed. For our proposed work, we have taken a hospital management application as an input application. At first, the features from this application were taken. The features taken were time, trace events, behavioral dependency. The features such as time, trace events were directly computed from the input application and the feature behavioral dependency was recognized from the UML diagrams of the input application.. Then the classes utilized in the application are categorized into two sets which specify whether the classes forecast the change-proneness or not by means of a decision tree algorithm, ID3. The tests were implemented on Java platform for authenticating any of the application regarding the fact whether it forecasts the change prone or otherwise. In future, we can add more features such as frequency and popularity to predict change prone classes.

REFERENCES

[1]. Aida Omerovic, Anette Andresen, Havard Grindheim, Per Myrseth, Atle Refsdal, Ketil Stolen, and Jon Olnes, "Idea: a feasibility study in model based prediction of impact of changes on system quality", In Proceedings of the Second international conference on Engineering Secure Software and Systems, pp. 231-240, 2010.

[2]. Mario Kušek, Saša Desic, and Darko Gvozdanović, "UML Based Object-oriented Development: Experience with Inexperienced Developers", In Proceedings of 6th International Conference on Telecommunications, pp. 55-60, June 2001.

[3]. James M. Bieman, Anneliese A. Andrews, and Helen J. Yang, "Understanding Change-proneness in OO Software through Visualization", In Proceedings of the International Workshop on Program Comprehension, 2003.

[4]. Daniele Romano, and Martin Pinzger, "Using Source Code Metrics to Predict Change-Prone Java Interfaces", In Proceedings of 27th IEEE International Conference on Software Maintenance, pp. 303-312, 2011.

[5]. András Pataricza, István Majzik, Gábor Huszerl and György Várnai, "UML-based Design and Formal Analysis of a Safety-Critical Railway Control Software Module", In Proceedings of the Conference on Formal Method for Railway Operations and Control Systems, 2003.

[6]. Kathy Dang Nguyen, P.S. Thiagarajan, and Weng-Fai Wong, "A UML-Based Design Framework for Time-Triggered Applications ", In Proceedings of 28th IEEE International Symposium on Real-Time Systems, pp. 39 - 48 , 2007.

[7]. Vahid Garousi, Lionel C. Briand and Yvan Labiche, "Analysis and visualization of behavioral dependencies among distributed objects based on UML models", In Proceedings of the 9th international conference on Model Driven Engineering Languages and Systems, pp. 365-379, 2006.

[8]. Kleanthis C. Thramboulidis , "Using UML for the Development of Distributed Industrial Process Measurement and Control Systems", In Proceedings of IEEE Conference on Control Applications, pp. 1129-1134, September, 2001.

[9]. A. Güneş Koru, and Hongfang Liu, "Identifying and characterizing change-prone classes in two large-scale open-source products", Journal of Systems and Software, Vol. 80, No. 1, pp. 63-73, January, 2007.

[10]. Nikolaos Tsantalis, Alexander Chatzigeorgiou, and George Stephanides, "Predicting the Probability of Change in Object-Oriented Systems", IEEE Transactions on Software Engineering, Vol. 31, No. 7, pp. 601-614, July 2005.

[11]. M.K. Abdi, H. Lounis, H. Sahraoui, "A probabilistic Approach for Change Impact Prediction in Object-Oriented Systems", In proceedings of 2nd Artificial Intelligence Methods in Software Engineering Workshop, 2009.

[12]. Erik Arisholm, Lionel C. Briand, and Audun Føyen, "Dynamic Coupling Measurement for Object-Oriented Software", IEEE Transactions on Software Engineering, Vol. 30, No. 8, pp. 491-506, August 2004.

[13]. Daniela Glasberg, Khaled El Emam, Walcelio Melo, and Nazim Madhavji, "Validating Object-Oriented Design Metrics on a Commercial Java Application", National Research Council, September 2000.

[14]. Mikael Lindvall, "Measurement of Change: Stable and Change-Prone Constructs in a Commercial C++ System", In Proceedings of IEEE 6th International Software Metrics Symposium, pp. 40-49, 1999.

[15]. Erik Arisholm, Dag I.K. Sjøberg, "Towards a framework for empirical assessment of changeability decay", The Journal of Systems and Software, Vol. 53, No.1, pp. 3-14, 2000.

[16]. Ah-Rim Han, Sang-Uk Jeon, Doo-Hwan Bae, and Jang-Eui Hong, "Behavioral Dependency Measurement for Change-Proneness Prediction in UML 2.0 Design Models", In Proceedings of 32nd Annual IEEE International Conference on Computer Software and Applications, pp. 76-83, 2008.

[17]. Ali R. Sharafat and Ladan Tahvildari, "Change Prediction in Object-Oriented Software Systems: A Probabilistic

Approach", Journal of Software, Vol. 3, No. 5, pp. 26-40, May 2008.

[18]. V.Lima, C. Talhi, D. Mouheb, M. Debbabi, L. Wang, and Makan Pourzandi, "Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages", ELSEVIER Electronic notes in Theoretical Computer Science, Vol. 254, pp. 143-160, 2009.

[19]. Mehdi Amoui, Mazeiar Salehie, and Ladan Tahvildari, "Temporal Software Change Prediction Using Neural Networks", International Journal of Software Engineering and Knowledge Engineering, Vol. 19, No. 7, pp. 995–1014, 2009.

[20]. Malan V. Gaikwad, Akhil Khare, and Aparna S. Nakil , "Finding Proneness of S/W using Class Hierarchy Method", International Journal of Computer Applications, Vol. 22, No. 6, pp. 34-38, May 2011.

[21]. Malan V.Gaikwad, Aparna S.Nakil, and Akhil Khare, "Class hierarchy method to find Change-Proneness ", International Journal on Computer Science and Engineering, Vol. 3 No. 1, pp. 21-27, Jan 2011.

[22]. Xiaoyan Zhu, Qinbao Song, and Zhongbin Sun, "Automated Identification of Change-Prone Classes in Open Source Software Projects", Journal of Software, Vol. 8, No. 2, pp. 361-366, February 2013.

[23]. Nachiappan Nagappan, Andreas Zeller ,Thomas Zimmermann, Kim Herzig and Brendan Murphy, "Change Bursts as Defect Predictors", In proceedings of IEEE 21st International Symposium on Software Reliability Engineering, pp. 309-318, November 2010.

[24]. Emanuel Giger, Martin Pinzger and Harald C. Gall, "Can We Predict Types of Code Changes? An Empirical Analysis", In Proceedings of 9th IEEE Working Conference on Mining Software Repositories, pp. 217-226, 2012.

[25]. Ali R. Sharafat and Ladan Tahvildari, "A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems", In Proceedings of IEEE 11th European Conference on Software Maintenance and Reengineering, pp. 27-38, 2007.