International Journal of Computer Applications (0975 – 8887)
Volume 22– No.6, May 2011

Finding Proneness of S/W using Class Hierarchy Method

Prof. Malan V. Gaikwad
Bharati Vidyapeeth Deemed University COE, Pune.

Prof. Akhil Khare
Associate Professor
Bharati Vidyapeeth Deemed University COE, Pune.

Prof. Aparna S. Nakil
Asst. Professor
Sinhgad College of Engineering Pune.

ABSTRACT

When it is requiring, releasing a new version of software it is helpful to take reference of old version. If developers know in advance which classes or methods of software may be change or will cause faults then developers can focus more on these classes.

It is require identifying classes at earlier stages of development which may cause changes or faults in other classes, so that those classes can be given special attention. The technique presented here improves the quality and reliability of the software. To achieve corrective and adaptive maintenance we require making changes during the software evolution. It is important to analyze the frequency of changes in individual classes and also to identify and show related changes in multiple classes as these changes are key components of software.

Prediction of change-prone and fault prone classes of a software is an active topic under research in the area of software engineering. Such prediction can be used to predict changes and faults in different classes of a system from one release to the next release of software. Finding the change-prone and fault prone classes from software in advance can help the developers to focus more attention on these classes.

The proposed model presenting a technique for finding dependency of software, change-prone classes and fault-prone classes of Object Oriented Software.

Keywords

Fault-Proneness, Change-Proneness

1. INTRODUCTION

In a recent research work, they used the probabilistic approach to predict the probability of changes in any object oriented software that might produce nearly accurate results for change proneness, but the process is seems to be bit lengthen. The proposed work presents a class hierarchy method which is easier and correctable as compared to other relevant methods.

The classes have a tendency to change and cause a fault in other classes are termed as change-prone and fault-prone classes. The detection of fault prone and change prone classes in early stages of development can enable the developers and experts to spend their valuable time and resources on these areas of software.

The proposed model is used for predicting the change prone classes and finding a probability of change-proneness by using class hierarchy method. In previous work of probabilistic approach they are taking dependencies of classes directly from UML diagrams and then change-prone classes are identified. In proposed work dependencies between the classes are identified by using class hierarchy method by traversing all classes, sub classes, and inherited classes. These dependencies are used to construct adjacency matrix and adjacency list which can be then used to find change-prone classes and probability of change-proneness.

In this historical approach the model reading all class names and their contents and keeping both in double dimensional array where each class name is binding with its contents in the respective matrix cell.

The model finds the fault-prone classes and fault-proneness by using OO metrics. The metrics like Weighted Methods per Class, Response for class, coupling between Objects are used to find the fault-proneness of classes. As the RFC is high then the fault-proneness is high. In the proposed model a class hierarchy method is used to find the fault-proneness. To find metrics of classes the source packages are taken as input and by reading all classes of package the metrics are calculated. After finding all metrics the model constructs the adjacency matrix of s/w metrics and classes. Then the model detects whether the classes are high fault-prone or very high fault-prone or medium or low fault-prone by using adjacency matrix and centroid formula.

2. LITERATURE SURVEY

Several researchers have proposed several modules which are used to find the change-proneness and fault-proneness of OO software. Let’s see some works out of these.

Arnold and Bohner [1] propose a model for change propagation. Several tools and techniques based on code dependencies and algorithms like slicing and transitive closure to assist in code propagation are demonstrated. The proposed methodology represents a system as a set of data dependency graphs. Graves et al. [2] represents model that finds the change-history of the system to be a better predictor than code metrics. In that model, Graves et al. assign weights to the perceived changes, with the most recent receiving the most. These weighted values provide a trend that is used to predict the number of faults in an upcoming period. Girba et al.[3] proposes an approach that consists of identifying the classes that were changed the most in the recent history and at the same time checking whether the same classes are among the most changed ones in the successive versions. However, in this only the addition or removal of methods is considered as changes. Mockus and Weiss [4] proposed a model to predict faults in a software system based on information extracted from changes to the system (e.g., lines of code modified, the changed components, etc.). Arisholm et al.[5] makes the use of dynamic coupling measures as indicators of change proneness. Their approach is based on correlating the

In other research work, they used c

number of changes to each class with dynamic coupling measures and other class-level size and static coupling measures. C. Catal, U. Sevim, and B. Diri [6] use clustering methods to find fault-proneness. X-means and fuzzy C-means algorithms are used to find fault-proneness. The centroid method is used to find the clusters center that is used for prediction of fault.Toshihiro Kamiya, Shinji Kusumoto and Katsuro Inoue [7] makes the use of OO s/w to find the fault. Completeness, correctness methods are used.

Sunint K. Khalsa [8] explains use of fuzzy algorithm to find fault-proneness. Fuzzy system model uses CK-metrics and defect density metrics to find fault-proneness. In fuzzy system metrics are given as input parameters which are converted into fuzzy values then defuzzyfication module maps these to crisp (original) values. Analysis of fault-proneness is done as whether it is low, high, higher, or vlow depending onto the ck-metrics. Parvinder S. Sandhu, Satish Kumar Dhiman, Anmol Goyal [10] applies the genetic algorithm to find fault – prone classes in different generations of s/w development.

3. PROPOSED METHOD

3.1 Change-Proneness

The proposed model of change-proneness by using hierarchical method is explained in detail in paper [12]. Here we will see only important aspect of change-proneness.

In this approach we are concentrating not only the classes but also the class components.

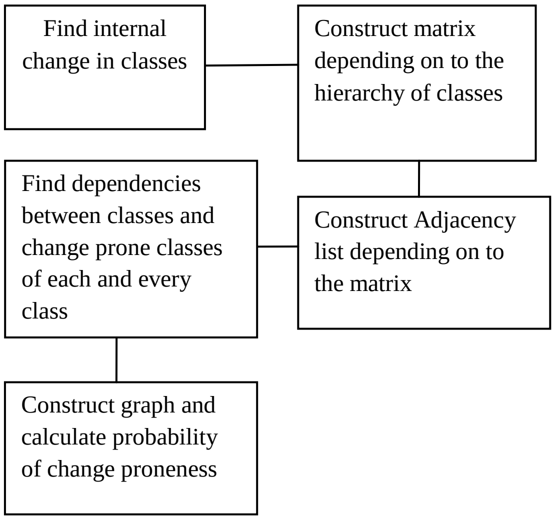

Find internal change in classes
Construct matrix depending on to the hierarchy of classes
Find dependencies between classes and change prone classes of each and every class
Construct Adjacency list depending on to the matrix
Construct graph and calculate probability of change proneness

Figure 1 Proposed model of change-Proneness

Proposed method involves following Analysis methods-

1. Find whether the internal changes in class
2. Find the dependencies between classes
3. Construct the adjacency matrix
4. Create the adjacency list depending onto the adjacency matrix
5. Construct the graph of dependency list
6. Find the change prone classes
7. Find the change proneness of a class(using probability)

3.1.1 Internal changes

The programmer or designer or a person involved in that project may make changes in project. Changes made in class are internal changes of that class; change is local to that class only it will not affect on any other class which is depending or inheriting from that class. When the change made in class is affecting to other class then it is external change to that class.

3.1.2 Adjacency matrix

In previous work different technologies and algorithms are used to find change-prone classes and change-proneness, which are bit complex. The proposed model uses easier and faster class hierarchy method to find the change prone classes. Module identifies the dependencies between classes and constructs a matrix.

In this approach we are considering no of different classes as a nodes of graphs and then depending on to these nodes we are constructing graphs.

```
Adjacency Matrix Algorithm
    Start
        Take array for matrix creation
        Read classes
        Find class name as a substring in all classes
        If substring found then
        Place 1 in array as matrix component
        Otherwise place 0 in matrix
        Display matrix
        End
```

Figure 2 Adjacency Matrix algorithm

Suppose there are three classes A, B, C and class B is depending on to the class B, class C is depending on to the class A. then we can show the dependency graph just like this.

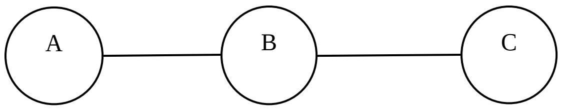


Figure 3 Dependency graph for class A, B, C

Adjacency matrix is a two dimensional array which stores the values 0 and 1 for graph. for the matrix each cell a[i][j]=1 if there is an edge i ➔ j and a[i][j] = 0 if there is not edge in between I and j. so for above graph the dependency using matrix we can show as following. A➔B=1 and B➔C=1 as there is edge between them.

$$\begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 1 & 0 \\ B & 1 & 0 & 1 \\ C & 0 & 1 & 0 \end{array}$$

Figure 4 Adjacency matrix for Figure 3

Adjacency matrix provides good solution for dense graphs, which implies having constant number of vertices.

3.1.3 Adjacency list

The adjacency list stores a list of vertices for every vertex in the graph which are adjacent to current vertices.

```
Adjacency List Algorithm
Start
Read classes from matrix
Find substring in matrix
If substring found then
Place class no array as list component
Display list
End
```

Figure 5 Adjacency List algorithm

A graph is stored in more compact form in adjacency list than adjacency matrix. Adjacency list is a good solution for sparse graphs.

0	1
1	0 2
2	1

Figure 6 Adjacency List for Matrix in Figure 4

Although there are disadvantages of both matrix and list, but here in my proposed model I used both the adjacency matrix and the adjacency list to get the desired output.

And finally the dependency and classes which are change prone to the particular class are identified. The probability of change proneness is calculated by using probability method.

P (D) = P (D) +P (D|C)*P(C) - P (D)*(D|C)*P(C) [9]

The P(D) is internal change in D is 0.5,P(C) is internal change in C is 0.5,P(D|C) is change propagating from C to D is 0.25.

Suppose there are four classes 0,1,2,3. Dependencies found by module for classes are 0→1, 1→2, 3→0, 3→1, 3→2.

The result of change proneness generated by module is-

Table 1 Results generated by Change-proneness Module

Class No.	Change-Proneness
0	0.5703125
1	0.5625
2	0.5
3	0.9960667

3.2 Fault Proneness

When new releases of software are needed to generate it is times consuming task to the testers to test the new generations of software. if testers will know in advance which classes of methods are fault-prone and its fault proneness then they can concentrate on to these classes. It results saving of time.

In the proposed model of finding the fault proneness, the fault-prone classes are identified by using the OO metrics like weighted methods per class (WMC), Coupling between objects (CBO), Response for class (RFC). The metrics are calculated and matrix is created depending on to the calculated metrics.

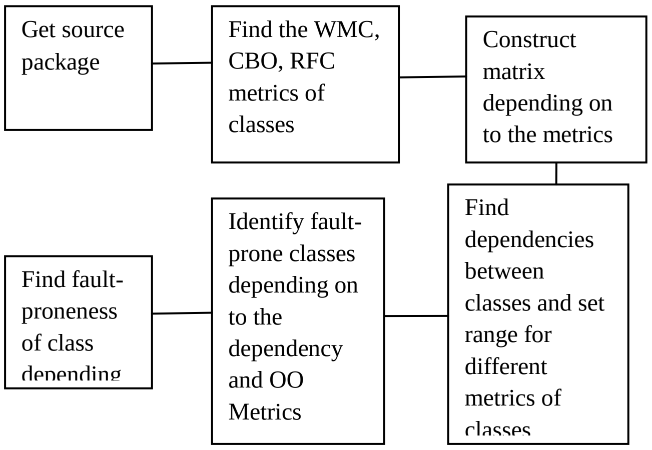

Get source package
Find the WMC, CBO, RFC metrics of classes
Construct matrix depending on to the metrics
Find dependencies between classes and set range for different metrics of classes
Identify fault-prone classes depending on to the dependency and OO Metrics
Find fault-proneness of class depending

Figure 7 Proposed model to find Fault-proneness of class

WMC-the number of classes in each class is the weighted methods of that class.

RFC-is the number functions that are called by another classes and member functions.

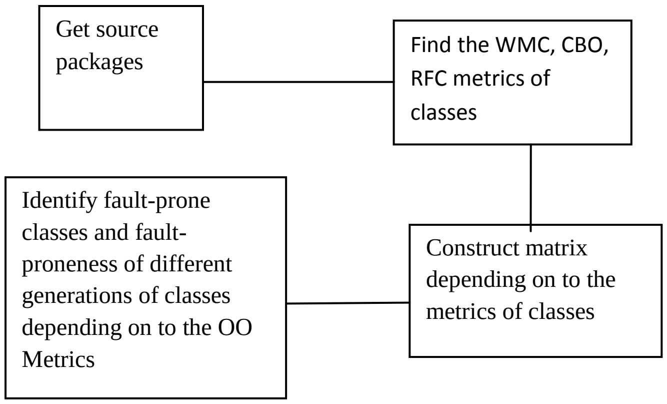

Get source packages
Find the WMC, CBO, RFC metrics of classes
Identify fault-prone classes and fault-proneness of different generations of classes depending on to the OO Metrics
Construct matrix depending on to the metrics of classes

Figure 8 proposed model for fault-proneness of classes of different releases.

```
Fault proneness Algorithm
Start
        Read source package
        Get all class names and contents
        Get all class dependencies
        Find the Weighted Methods per Class
        Find response for class
        Find coupling between the objects
        Construct adjacency matrix for metrics
        Get range of matrix to find fault-proneness
        If range is between some range
        Then fault-proneness is low, high, medium or very high
        Print whether FP is high, low, very high, and medium
        By using metrics values Find fault-proneness
End
```

Figure 9 Fault-Proneness algorithm

CBO-is the number of classes coupled to the given class. The coupling may be afferent coupling or efferent coupling. Efferent coupling is the number of classes that the measured class is depended upon and afferent coupling is the number of classes that depends upon the measured class.

After Calculation of WMC, RFC and CBO I am creating their matrix, where each row represents the class name and column represents their respective value of WMC, RFC and CBO. Then by observing the above matrix we need to set the parameter range for metrics, like Very low, Low, Medium, High and very high. To find fault proneness of class we need to set a range and to find fault-proneness of different releases no need to set a range of metrics.

Then by using knowledge base we will calculate the very low and very high parameters of the object oriented matrix and then by using centroid method equation we calculate fault proneness.

Centroid method equation used in our fault prone module is.

$$\frac{((wmclow*wmchigh) + (rfclow*rfchigh) + (cbolow*cbohigh))}{(Wmchigh+rfchigh+cbohigh)}$$

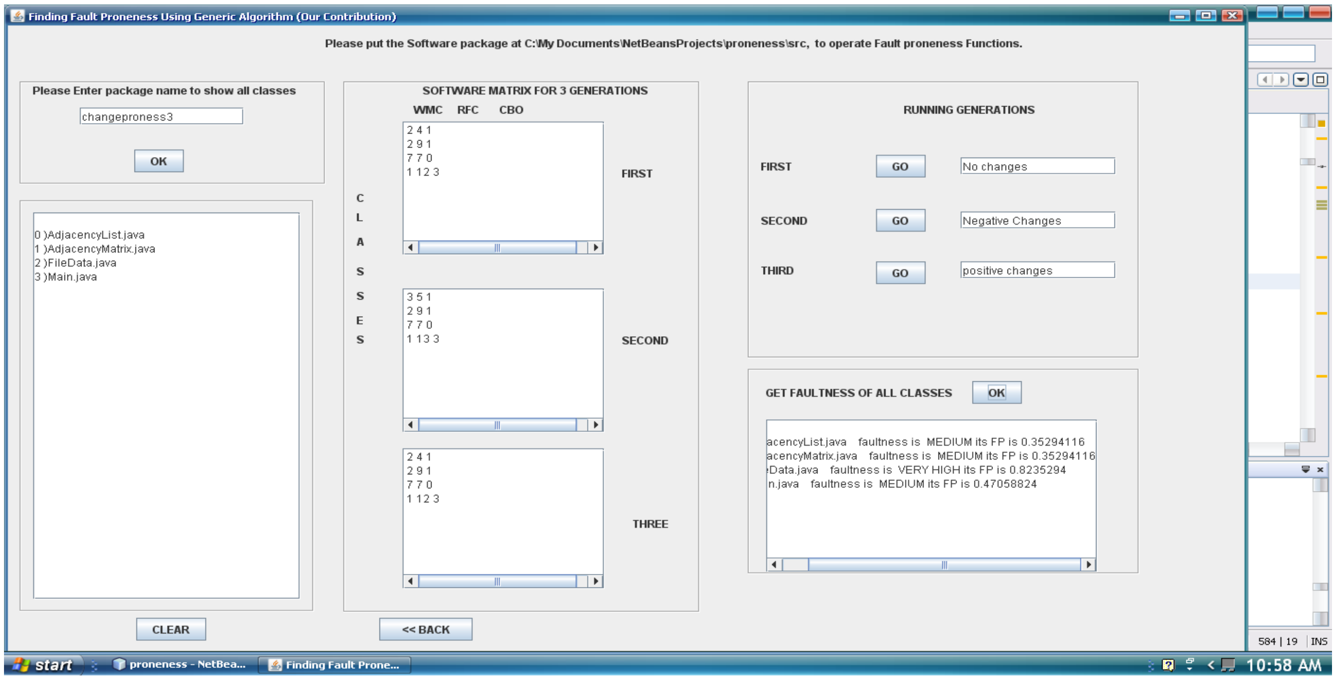
Consider the following figure where we are considering the last release of the software, in our case it is third generation.

- From the third generation matrix first we will get the sum of all WMC and CBO of the matrix that is equal to 17.00
- Then gets the average by dividing sum from number of classes that is 17/ 4= 4.25.
- And then we consider average * 2 (that is 8.5=da)as the maximum value for any class.(here value refers to the sum of WMC+CBO of any class).
- then for the first class value=3 then we did like (3/8.5) then we get its faultness that is 0.3529
- This will be continuing for all classes.
- And we consider a class as low faulty which gives a value b/w 0 to 25% of da that is 0 to 2.125(8.5*0.25=2.125)
- And we consider a class as Medium faulty which gives a value b/w 26 to 50% of da that is > 2.125 to <= 4.25(8.5*0.5=4.25)
- And we consider a class as High faulty which gives a value b/w 51 to 75% of da that is >4.25 to <= 6.375(8.5*0.75=6.375)

And we consider a class as Very High faulty which gives a value b/w 76 to100% of da that is >6.375 to <= 8.5(8.5*1.0=8.5)

Table 2 Fault-Prone Result

Class no	WMC	RFC	CBO
0	2	4	1
1	2	9	1
2	7	7	0
3	1	12	3



Please put the Software package at C:\My Documents\NetBeansProjects\proneness\src, to operate Fault proneness Functions.

Please Enter package name to show all classes: changeproness3

```
0 )AdjacencyList.java
1 )AdjacencyMatrix.java
2 )FileData.java
3 )Main.java
```

SOFTWARE MATRIX FOR 3 GENERATIONS

CLASSES

```
WMC  RFC  CBO
2 4 1
2 9 1
7 7 0
1 12 3       FIRST

3 5 1
2 9 1
7 7 0
1 13 3       SECOND

2 4 1
2 9 1
7 7 0
1 12 3       THREE
```

RUNNING GENERATIONS

FIRST: No changes

SECOND: Negative Changes

THIRD: positive changes

GET FAULTNESS OF ALL CLASSES

```
acencyList.java    faultness is  MEDIUM its FP is 0.35294116
acencyMatrix.java    faultness is  MEDIUM its FP is 0.35294116
Data.java    faultness is  VERY HIGH its FP is 0.8235294
n.java    faultness is  MEDIUM its FP is 0.47058824
```

Figure 10 Result of Fault-Proneness model

4. CONCLUSION

In previous work all data which required is taken manually and from UML diagrams. The methods used by various peoples are difficult to understand and quite complex to implement.

The proposed method of using class hierarchy method is too simple to understand and implementation. And one important thing is that I have not taken any data manually, it is calculated by this model only. The proposed model finds change-prone classes and change-proneness of classes and fault-prone classes and fault-proneness of classes. The previous works takes the OO metrics directly from the tools available for finding the s/w metrics but the proposed model finds the S/w metrics by using class hierarchy method.

5. REFERENCES

[1] R. Arnold and S. Bohner, "Impact Analysis – Toward a Framework for Comparison," in Proceedings of the IEEE International Conference on Software Maintenance (ICSM), 1993, pp. 292–301.

[2] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy,"Predicting fault incidence using software change history,"IEEE Trans. on Soft. Eng., vol. 26, no. 7, pp. 653–661,2000

[3] T. Girba, S. Ducasse, and M. Lanza, "Yesterdays Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes," in Proceedings of the IEEE International Conference on Software Maintenance (ICSM), 2004, pp. 284–293

[4] A. Mockus and D. M. Weiss, "Predicting risk of software changes," Bell Labs Technical Journal, vol. 5, no. 2, pp. 169 – 180, April 2000.

[5] E. Arisholm, L. Briand, and A. Foyen

[6] , "Dynamic Coupling Measurement for Object-Oriented Software," IEEE Trans.on Soft. Eng., vol. 30, no. 8, pp. 491–506, 2004.

[7] Software Fault Prediction of Unlabeled Program Modules by C. Catal, U. Sevim, and B. Diri, Member, IAENGin the Proceedings of the World Congress on Engineering 2009 Vol I

[8] Software Fault Prediction of Unlabeled Program Modules by C. Catal, U. Sevim, and B. Diri, Member, IAENG in the Proceedings of the World Congress on Engineering 2009 Vol I

[9] A Fuzzified Approach for the Prediction of Fault Proneness and Defect Density in the Proceedings of the World Congress on Engineering 2009 Vol. I WCE 2009, July 1 - 3, 2009, London, U.K. by Sunint K. Khalsa

[10] Change Prediction in Object-Oriented Software Systems: A Probabilistic Approach by Ali R. Sharafat and Ladan Tahvildari in Journal of software vol. 3, no. 5, may 2008.

[11] A Genetic Algorithm Based Classification Approach for Finding Fault Prone Classes in the World Academy of Science, Engineering and Technology 60 2009 by Parvinder S. Sandhu, Satish Kumar Dhiman, Anmol Goyal

[12] Predicting the Probability of Change in Object-Oriented Systems Nikolaos Tsantalis, Alexander Chatzigeorgiou, in, in IEEE Transactions on software engineering, vol.31, no.7, July 2005.

[13] "Class hierarchy method to find change proneness" by Malan V.Gaikwad, prof. Akhil khare, A.S.Nakil in the International journal of Computer Science and Engg.Vol.3 issue 1.