Fault Proneness of Classes in Object-Oriented Systems

Naeib S Gill and Sunil Sikka
1 Head, Department of Computer Science & Applications M.D. University, Rohtak (Haryana) E-mail: nsgill_2k4@yahoo.com
2 Research Scholar, Department of Computer Science & Applications M.D. University, Rohtak (Haryana) E-mail: sunil_sikka@rediffmail.com

ABSTRACT: Predicting fault proneness of classes in object oriented systems (OOS) is one of the fault management activities and may be considered as one of the cost effective ways to deal with faults. Prediction of faults needs identification and measurement of various factors that affects fault proneness of classes in OOS. In this paper, several design oriented factors have been identified that help in codetermining the fault proneness of a class in object oriented systems. Fault proneness is finally aimed at indicating the risk levels associated with the classes so as to focus on these classes in design of object-oriented systems accordingly. Further, existing relevant object-oriented metrics have also been suggested for measuring the fault proneness.

Keywords: Fault Proneness, Class Complexity and Object Oriented Metrics.

1. INTRODUCTION

Fault free software development using extensive fault detection and correction methods such as inspection, review, testing, etc. have shown positive impact on software quality but a heavy cost has to be paid for it. For optimum utilization of resources and reducing the cost of software, fault detection and elimination process must be properly planned and for this type of planning prediction of fault prone area is gaining importance among researchers.

Developing fault free software in object oriented environment needs detection and elimination of faults from classes. In Object oriented software fault proneness of classes strongly influences the
fault detection and removal strategy. Fault proneness of a class can be defined as the likelihood of finding faults from it. On the basis fault proneness, testing strategy can be planned. The classes having more fault proneness must be tested more rigorously. Many researches over the past several years have devoted to the analysis of object oriented features and metrics. It has been identified by many researches that object oriented metrics play very important role in predicting fault proneness of software.

First section of the paper discusses the components of a class. Second section identifies factors that help in codetermining the fault proneness of class. Third section is aimed at suggesting the relevant object oriented metrics that may be used in measuring fault proneness factors. Last section presents the conclusions of the present study and suggests the future scope of the present work and its applicability.

2. EXPLORING OBJECT-ORIENTED SYSTEM AND CLASSES
Object oriented system is a collection of classes and relationships among them. A class is used to create set of objects that share common attributes and methods. Each class belongs to a package or namespace. To identify the various factors that affect the fault proneness of a class it is necessary to understand its various components. Following are components of a class:

Method
A method is a member functions defined in the class. Name of method need not be unique within a class. But combination of name and signature must be unique. The visibility of a method is controlled by access modifier. It can be a constructor, destructor, event handler or property.

Attribute
An attribute is a data member of class that is available to every object of class. It is manipulated by methods of class. Access modifier controls its visibility.
Relations
Relationships are mean of communication among classes. A class is related to another class by two means: coupling and inheritance.

3. Factor Affecting Fault Proneness of Class
There are two types of class complexities:

- Member Complexity ($M_c$) and
- Relational Complexity ($R_c$).

A class consists of data members, methods and relationships with other classes. Complexity due to data members and methods is called member complexity and complexity of class due to relationships is called relational complexity. Therefore, complexity ($C$) of a class can be denoted as

$$C = M_c + R_c$$

The complexity ($M_c$) due to member function of class depends upon the logic of the method whereas relational complexity ($R_c$) of class depends upon coupling and inheritance. Higher complexity causes higher fault proneness. Possible factors affecting fault proneness of a class are discussed below.

Coupling
It is the degree of interrelationships between two classes. A class is coupled with another class if it uses the method or instance variable of another class. A strong coupling between classes in an Object Oriented design can also increase the complexity of the system by introducing multiple kinds of interdependencies among the classes [1]. Various earlier empirical researches show that higher coupling causes higher faults.

Inheritance
Inheritance is a method of reusability. Besides the benefit of reusability, inheritance introduces another kind of relational complexity in class. Class situated at deep level in inheritance hierarchy is more suspected to faults.
Cohesion

It is the degree to which methods within a class is related to each other. It is the measure of conceptual consistency within objects [1]. Cohesion of a class is its tightness. Low cohesion increases the method complexity, therefore, increases the fault proneness.

Polymorphism

Polymorphism allows the implementation of a given operation to be dependent on the object that contains the information. It introduces dynamic complexity in the software. It can be used to keep the code clear but excessive polymorphic code may be too complex to understand and fault prone. Various researches have been conducted to find out the desirable value of polymorphism. But we do not have any standard desired value/range for polymorphism.

Information Hiding

It is the hiding of details of one object from another object. In object oriented environment it is achieved by controlling the access of members of class by using visibility modifiers. Earlier research shows that increased hiding decrease fault proneness.

From above factors it is clear that fault proneness of a class is function of method complexity, coupling, cohesion, inheritance, polymorphism and information hiding. In functional form, we may write

Fault Proneness of class = \( c_0 + c_1 \times \text{method complexity} + c_2 \times \text{coupling} + c_3 \times \text{cohesion} + c_4 \times \text{inheritance} + c_5 \times \text{polymorphism} + c_6 \times \text{information hiding} \)

where \( c_0 \) is constant term. \( c_1, c_2, \ldots, c_6 \) are weighting factors for the respective proneness factor.
4. METRICS FOR MEASURING FAULT PRONENESS FACTORS

Eight metrics are identified from literature [1],[2],[3],[4], that may be used to measure fault proneness factors presented in Section 2.

Weighted Method Per Class (WMC)

WMC is the sum of complexity of all the methods defined in a class. The complexity of a method can be measured using traditional cyclomatic complexity metric. If complexity of each method is equal to one then WMC is equal to number of methods in the class. It is the measure of method complexity of a class.

Response for a Class (RFC)

It is the number of methods defined in class and directly called by methods of the class that can be executed in response to a message received by an object of that class. Modified definition of RFC (RFC') counts total number of methods defined in class and called recursively through entire call tree that can be executed in response to a message received by an object of that class. RFC measures the complexity of the software by counting the number of methods in the class and also captures the information about the coupling of the class to other classes.

Coupling Between Object Classes (CBO)

CBO of a class is equals to number of other classes to which given class is coupled. It measures the interdependencies between the classes.

Depth of Inheritance (DIT)

DIT is inheritance metric. DIT of a class is the path length from root class to given class in the inheritance hierarchy.

Lack of Cohesion of Methods (LCOM)

It measures the dissimilarity of methods in a class by finding the difference between number of pair of methods that have no common
attribute and number of pair of methods that have common attribute. Greater than zero value of LCOM indicates lack of cohesion. Higher the value of LCOM lowers the cohesion.

**Number of Methods Overridden (NMO)**

It computes the polymorphism of class by measuring the number of methods in class that overrides the method of base class.

**Method/Attribute Hiding of Class (MHC/AHC)**

MHC/ AHC measures amount of method/attribute hiding of class. It count average amount of method/attribute hiding among other class in the software. It can be computed as

\[
MHC = \frac{\text{total number of classes where each method of a class is invisible}}{\text{total no of classes}-1} / \text{total no. of methods in the class}
\]

\[
AHC = \frac{\text{total number of classes where each attribute of class is invisible}}{\text{total no of classes}-1} / \text{total no. of attributes in the class}
\]

MHC and AHC are derived from Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF) metrics respectively given in literature. MHF and AHF compute the average amount of hiding in overall system whereas MHC and AHC computes average amount of hiding of a particular class with respect to other classes in the software.

5. **CONCLUSIONS**

In this paper various factors have been presented that affect the fault proneness of classes in object oriented software. Further, eight existing metrics have also been suggested that can be used to measure fault proneness factors. Fault proneness is also described as a function of several factors such as method complexity, coupling, cohesion, inheritance, polymorphism and information hiding. Further research is required to derive the value of \(c_0, c_1, \ldots, c_6\). Our study is restricted to only software design-oriented factors; however, this study can be extended for identification and measurement of non-design oriented factors.
REFERENCES


