Advance Software Engineering (SE-406)

LAB A1-G3

Laboratory Manual

Department of Software Engineering

DELHI TECHNOLOGICAL UNIVERSITY(DTU)

Shahbad Daulatpur, Bawana Road, Delhi-110042

Submitted to: -

Ms. Parul Sharma

Submitted by:-

Name: ASHISH KUMAR

Roll number: 2K18/SE/041

EXPERIMENT 3

- ASHISH KUMAR
- 2K18/SE/041

Aim:- To Compute Chidamber and Kemerer Metrics for a given source code.

Introduction:-

The Chidamber & Kemerer metrics suite originally consists of 6 metrics calculated for each class which are:

1. Weighted Methods Per Class (WMC)
2. Depth of Inheritance Tree (DIT)
3. Number of Children (NOC)
4. Coupling between Object Classes (CBO)
5. RFC and RFC´ Response for a Class
6. Lack of Cohesion of Methods (LCOM1)

1. Weighted Methods Per Class (WMC)

Despite its long name, WMC is simply the method count for a class.

WMC = number of methods defined in class

A high WMC has been found to lead to more faults. Classes with many methods are likely to be more application specific, limiting the possibility of reuse. WMC is a predictor of how much time and effort is required to develop and maintain the class. A large number of methods also means a greater potential impact on derived classes, since the derived classes inherit (some of) the methods of the base class.

2. Depth of Inheritance Tree (DIT)

A high DIT has been found to increase faults. The deeper a class is in the hierarchy, the more methods and variables it is likely to inherit, making it more complex. Deep trees as such indicate greater design complexity.

Setting limit for DIT: An increase in DIT increases the density of bugs and decreases quality. A recommended DIT is 5 or less. The Visual Studio .NET documentation recommends that DIT <= 5 because excessively deep class hierarchies are complex to develop. Some sources allow up to 8.

3. Number of Children (NOC)

NOC measures the breadth of a class hierarchy, whereas maximum DIT measures the depth.

NOC = number of immediate child classes derived from a base class.

Depth is generally better than breadth, since it promotes reuse of methods through inheritance. NOC and DIT are closely related. Inheritance levels can be added to increase the depth and reduce the breadth.

A high NOC is an indicator of:

- ❖ High reuse of base class (Inheritance)
- ❖ Base class may require more testing.
- ❖ Improper abstraction of the parent class.
- ❖ Misuse of sub-classing.

Setting limit for NOC:

1. A redesign may be required for a class with a high NOC and a high WMC indicates complexity at the top of the class hierarchy. This type of class is potentially influencing a large number of descendant classes.
2. All classes need not have the same number of subclasses. Classes higher up in the hierarchy should have more subclasses than those lower down.

4. Coupling between Object Classes (CBO)

High CBO is undesirable. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

Setting limit for CBO: In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A high coupling has been found to indicate fault-proneness.

5. RFC and RFC´ Response for a Class

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. RFC is simply the number of methods in the set.

RFC = M + R (First-step measure)

RFC' = M + R' (Full measure)

Where,

M = number of methods in the class

R = number of remote methods directly called by methods of the class

R’ = number of remote methods called, recursively through the entire call tree

Since RFC specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.

RFC is the original definition of the measure. It counts only the first level of calls outside of the class. RFC’ measures the full response set, including methods called by the callers, recursively, until no new remote methods can be found. If the called method is polymorphic, all the possible remote methods executed are included in R and R’.

Setting limit for RFC:

1. A large RFC has been found to indicate more faults. Classes with a high RFC are more complex and harder to understand. Testing and debugging is complicated. A worst case value for possible responses will assist in appropriate allocation of testing time.
2. An increase in RFC increases the density of bugs and decreases quality, though the optimum level has not been suggested by any researcher.

6. Lack of Cohesion of Methods (LCOM1)

In low cohesion only one class is responsible to execute lots of jobs that are not in common which reduces the chance of reusability and maintenance. But in high cohesion, there is a separate class for all the jobs to execute a specific job, which results in better usability and maintenance.

- High cohesion is when you have a class that does a well-defined job. Low cohesion is when a class does a lot of jobs that don’t have much in common.
- High cohesion gives us better-maintaining facility and Low cohesion results in monolithic classes that are difficult to maintain, understand and reduce re-usability

<u>Source Code(in C++):</u>

```
#include<bits/stdc++.h>
using namespace std;
class employee{
public:
        int empid;
        string emailid;
        vector<string> getmeetinglist(){

        }
        void getinfo(){

        }
};
class SDE: public employee{
public:
        SDE(int eid,string email){
        empid=eid;
        emailid=email;
        }
        string managername;
        string businessunitid;
        bool ishandlingsecurity(){

        }
        int getdepid(){
                string info = getinfo();
        }
};

int main(){
SDE ashish(1,"abc@gmail.com");
ashish.businessunitid=2;
ashish.getdepid();
ashish.getmeetinglist();
ashish.getinfo();
return 0;
}
```

Result:-

a) WMC

There are two class name “SDE” and “employee”. In employee class, there are 2 methods & 2 attributes whereas in SDE (i.e. derived class) there are 2 attributes & 2 methods.
So, Total WMC = Total methods in class = 2 + 2 = 4

b) DIT

Since SDE is the derived class, so its depth would be 1 and for employee which is the base class, its depth would be 0.
So, Total DIT = 0 + 1 = 1

c) NOC

Since NOC is the opposite of DIT. For SDE class, NOC is 0 and for employee, NOC is 1.
So, Total NOC = 0 + 1 = 1

d) CBO

CBO is defined as "Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class". In SDE class, I have used one method which is defined in base class i.e. employee.

So, **CBO would be 1 and these two classes are coupled.**

e) RFC

Since RFC is simply the number of methods in the class, so in employee class, there are 2 methods whereas in SDE (i.e. derived class) there are 2 methods.

f) LCOM

For LCOM calculation:

M1= {0}   //since all the 4 methods are not using any attributes

M2= {0}

M3= {0}

M4= {0}

LCOM= no. of ones – no. of zeroes = 0 – 4 = 0

<u>Learning from experiment:</u>- We have successfully learnt about Chidamber and Kemerer Metrics and we compute the 6 metrics of a given source code.