Applying Object Oriented Metrics to C#(C Sharp) programs

Arti Chhikara
Maharaja Agrasen College, Delhi, India.

R.S.Chhillar
Deptt. Of Computer Sc. And Applications, Rohtak, India.

Sujata Khatri
Deen Dyal Upadhyaya College, Delhi, India.

Abstract: Object-oriented technology is becoming increasingly popular in industrial software development environments. This technology helps in the development of a software product of higher quality and lower maintenance costs. Since the traditional software metrics aims at the procedure-oriented software development so it cannot fulfill the requirement of the object-oriented software, as a result a set of new object oriented software metrics came into existence. Object Oriented Metrics are the measurement tools adapted to the Object Oriented paradigm to help manage and foster quality in software development.

In this research paper we investigate several object oriented metrics proposed by various researchers. These object oriented metrics are than applied to several C sharp programs. A critical analysis of the results is presented listing the crucial points about the language so that software managers and developers can use this results for building object oriented system in C#.

Keywords: Object Oriented Software Development, Software Metric, Software Product, C#.

1. Introduction:
The design and development of software using object oriented paradigm is gaining popularity day by day. Object Oriented Analysis and Design of software provide many benefits to both the program designer and the user. Object Orientation contributes to the solution of many problems associated with the development and quality of software product. This technology promises greater programmer productivity, better quality of software and lesser maintenance cost [1].

Object oriented software development requires a different approach from more traditional functional decomposition and data flow development methods. While the functional and data flow approaches commence by considering the systems behavior and/or data separately, object oriented analysis approaches the problem by looking for system entities that combine them. Object oriented analysis and design focuses on objects as the primary agents involved in a computation; each class of data and related operations are collected into a single system entity [2, 3].

There are several object oriented programming languages that supports object oriented paradigm. Most commonly used are Java, C++, C sharp, Vb.net. C sharp is Microsoft’s new programming language for .net platform. It combines some of the best features of modern programming language such as java, c++ or visual basic[2]. In this research paper different C sharp programs are studied and object oriented software metrics are applied to them and a study is made based on the results obtained by applying object oriented metrics to C# source code.
The rest of the paper is organized as follows. Section 2 give a brief overview of object oriented metrics. Section 3 presents an example of C# source code. Section 4 presents results obtained by applying object oriented metrics to C# source code. Section 5 presents conclusion.

2. Literature Research

2.1 Object Oriented Metrics

The Object Oriented technology forced the growth of object oriented metrics. Although many metrics have been proposed, few have been based on the sound measurement theory or, further have been empirically validated. One of the first attempts to do this was by Chidamber and Kemerer (C&K). They have proposed six new OO metrics based on theoretical concepts [4]. These metrics are:

- Weighted Methods per Class (WMC),
- Depth of Inheritance Tree (DIT),
- Number of Children (NOC),
- Coupling Between Object classes (CBO),
- Response For a Class (RFC) and
- Lack of Cohesion between Methods (LCOM)

Several studies have been conducted to validate CK’s metrics. Their metrics have been criticized, specially the LCOM metric, for being too ambiguous for practical applications and for not being language independent [Churcher 95]. Basili et al. [5] presented the results of an empirical validation of CK’s metrics. Tang et al. [6] validated CK’s metric suit using real time systems.

Li, et al. have also empirically evaluated C&K's metrics as being predictors of maintenance effort [Li 95]. In addition, Li, et al. [7] proposed new metrics that were used in their study including:

- Message passing coupling,
- Data abstraction coupling, and
- Number of local subunits.

where message passing coupling and data abstraction coupling refine C&K's coupling between objects metric. They found a significant correlation between many of the metrics and the number of lines changed per class during maintenance. The metrics proposed by C&K and Li, et al. seem to offer the greatest potential as being valid metrics for object-oriented design. The metrics have been properly derived and are well on their way to being empirically validated. Most of their metrics are also applicable to C sharp software developed using an object-oriented methodology.

To summarize, the relevant metrics are:
3. Metric Definition

To better define and understand how these metrics are calculated using C#, source code example is used.

3.1: C# source code[1,2]

```csharp
using System;
public class employee                        //employee class
{
    private:
        String name;                         //employee name
        int number;                          //employee number
    public:
        void getdata()
        {
            Console.WriteLine("enter name:");
            name= Console.ReadLine();
            Console.WriteLine("enter number:");
            number=int.parse(Console.ReadLine());
        }
        void putdata()
        {
            Console.WriteLine("The name is:" +name);
            Console.WriteLine("Number= " +number);
        }
}
public class manager : employee             //management class
{
    private:
        String title ;                      //“vice-president “ etc.
```
double dues ;     // golf club dues
public:
void getdata()
{
    base.getdata();
    Console.WriteLine("enter title :");
    title=Console.ReadLine();
    Console.WriteLine("enter golf club dues:");
    dues=double.parse(Console.ReadLine());
}
void putdata()
{
    base.putdata();
    Console.WriteLine("title:" +title);
    Console.ReadLine("dues:"+dues);
}
public class scientist : employee     // scientist class
{
private:
    int pubs ;
public:
    void getdata()
    {
        base.getdata();
        Console.WriteLine("enter number of pubs:");
        pubs=int.parse(Console.ReadLine());
    }
    void putdata()
    {
        base.putdata();
        Console.WriteLine("number of pubs:" +pubs);
    }
}
public class laborer : employee     // laborer class
{
    private:
        int a;
    public:
        int hours;
        void getdata()
        {
            base.getdata();
            Console.WriteLine("Enter number of hours:");
            hours=int.parse(Console.ReadLine());
        }
}
void cal()
{
    int total=0;
    total = LEN*40;
}

void putdata()
{
    base.putdata();
    Console.WriteLine("number of hours :" +hours);
    Console.WriteLine("Total:" +total);
}

public class hourlyemployee: laborer  //hourlyemployee class
{
    private:
        double sal;
    public:
        void getdata()
        {
            base.getdata();
            Console.WriteLine("enter number of hours:");
            hours=int.parse(Console.ReadLine());
        }
        void salary()
        {
            sal=hours*250;
        }
        void putdata()
        {
            base.putdata();
            Console.WriteLine("The salary is: " +sal);
        }
        void main()  //main method
        {
            manager m1 = new manager();
            manager m2 = new manager();
            scientist s1 = new scientist();
            laborer L1 = new laborer();
            hourlyemployee h1 = new hourlyemployee();
            Console.WriteLine("Enter data for manager 1"); //get data for several employees
            m1.getdata();
            Console.WriteLine("Enter data for manager 2");
            m2.getdata();
            Console.WriteLine("Enter data for scientist 1");
            s1.getdata();
            Console.WriteLine("Enter data for laborer 1");
L1.getdata();
Console.WriteLine("Enter data for hourlyemployee 1");
h1.getdata();
Console.WriteLine("Data on manager 1");
m1.putdata();
Console.WriteLine("Data on manager 2");
m2.putdata();
Console.WriteLine("Data on scientist 1");
s1.putdata();
Console.WriteLine("Data on Laborer 1");
L1.putdata();
Console.WriteLine("Data on hourly employee");
h1.putdata();
}

3.3 Object Oriented Software Metrics Applied on Example 1:

1. WMC (Weighted Method per Class): WMC is calculated by counting the number of methods in each class. [4] therefore:
   WMC for Employee= 2
   WMC for Manager = 2
   WMC for Scientist = 2
   WMC for Laborer = 3
   WMC for Hourlyemployee = 3

2. RFC (Response for a Class): The RFC is the number of functions or procedures that can be potentially be executed in a class. Specifically, this is the number of operations directly invoked by member operations in a class plus the number operations themselves [4].
   RFC for Employee= 2
   RFC for Manager = 4
   RFC for Scientist = 4
   RFC for Laborer = 4
   RFC for Hourlyemployee = 7

3. DIT (Depth of Inheritance tree): The depth of inheritance is defined to be the level of the class in the inheritance hierarchy, with the root class being Zero [4].
   DIT for Employee= 0
   DIT for Manager = 1
   DIT for Scientist = 1
   DIT for Laborer = 1
   DIT for Hourlyemployee = 2

4. NOC (Number of Children): The number of children is the number of direct descendents for a class [4].
   NOC for Employee= 3
   NOC for Manager = 0
   NOC for Scientist = 0
   NOC for Laborer = 1
   NOC for Hourlyemployee = 0
5. MPC (Massage Passing Coupling): Message Passing coupling is the count of total number of function and procedure calls made to external units [7].
   MPC for Employee = 0
   MPC for Manager = 2
   MPC for Scientist = 2
   MPC for Laborer = 2
   MPC for Hourlyemployee = 4

6. DAC (Data Abstraction Coupling): Data Abstraction coupling is the count of total number of instances of other classes within a given class [7].
   DAC for Employee = 0
   DAC for Manager = 0
   DAC for Scientist = 0
   DAC for Laborer = 1
   DAC for Hourlyemployee = 0

7. NUS (Number of Subunits): The number of subunit is the total number of functions and procedures defined for the class [7].
   NUS for Employee = 2
   NUS for Manager = 2
   NUS for Scientist = 2
   NUS for Laborer = 3
   NUS for Hourlyemployee = 3

4. Study of the object oriented software metrics programs

These metrics were calculated and tested on several C sharp programs and following points are observed

1 The WMC metric is a predictor of how much time and effort is required to develop and maintain the class. The larger the number of methods in a class, the greater the potential impact on children; children inherit all of the methods defined in the parent class. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. It was observed that an increase in the average WMC increases the density of bugs and decreases quality.

2 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. 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. A study of C sharp programs suggests that an increase in RFC increases the density of bugs and decreases quality.

3 The deeper a class is in the hierarchy, the more methods it is likely to inherit, making it more complex. Deep trees as such indicate greater design complexity. As a positive factor, deep trees promote reuse because of method inheritance. C sharp programs have intermediate value for DIT metric.
4 High NOC indicates high reuse, since inheritance is a form of reuse. A large number of children (high NOC) may also mean improper abstraction of the parent class. If a class has too many children, it may indicate misuse of sub-classing. A class with many children may also require more testing. High NOC has been found to indicate fewer faults. This may be due to high reuse, which is desired. In C# the value of this metric depends on program to program. All classes do not have the same number of sub-classes. However, it is observed that for better results, classes higher up in the hierarchy should have more sub-classes then those lower down.

5 Message passing coupling is the count of the total number of functions and procedures calls made to external units. The assumption behind this metric is that classes interacting with many other classes are harder to understand and maintain. When we applied object oriented metrics on several C sharp programs, we observed that the value of Message Passing Coupling (MPC) metric is low for C# programs.

6 Data abstraction coupling is a count of total number of instances of other classes within a given class. It is the count of total number of external classes the given classes uses. Since C sharp is an object oriented language so there is a data security. Data is not allowed to move freely around the system. As a result the value of Data Abstraction Coupling metric is low for C#.

7 The number of local subunits is the total number of functions and procedures defined for a class. Classes with large number of operations are harder to maintain and are more fault prone. If the complexity for each operations the NUS metric is 1 then the NUS metric is same as the WSC metric. The value of Number of local Subunits metric is found to be high for C# programs.

5 Conclusion and Future Work

C# is a modern and powerful language which is fully object oriented language. This research paper presented various object oriented metrics that can be successfully applied to C#. The results of object oriented metrics implemented on C# in above research paper are comparatively good.

However, the metrics presented in this research paper are by no means a complete set of object oriented metrics for C#. But this analysis can be used as a reference by software developers and managers for building a fault free, reliable and easy to maintain software product in C#. There are many distinguished features in C# that make it different from other object oriented languages. So future work will be to refine the current metrics and define additional metrics.

References:


