A Comparative Study of Cost Estimation Metrics using Class Diagram in Software Project Management

Santosh Kumar Sahoo¹
Department of Computer Science & Engineering
Gurukul Institute of Technology, Bhubaneswar
Biju Patnayak University of Technology, Odisha, India
citsantosh@gmail.com

Balaram Singh²
Joint Venture College Computer Education
B.B.Mahavidyalaya, Chandikhole
Utkal University, Odisha, India
balaramsingh.cse@rediffmail.com

Abstract—Software cost estimation is one of the important activities of software project management. It is preferably required in the early phase of development of software. Cost estimation includes many factors related to development in addition to the estimation of size, complexity, effort of development and duration of development of software. Many metrics are supported by practitioners for this estimation process. Estimating the size and level of complexity of a system in an object oriented development environment using class diagram can help not only the effort required for the entire system but also for the effort required for the data part development of the system. Many published papers have contributed a number of metrics for measuring size and complexity of class diagram. A detailed study on the existing available metrics and a comparison among them has been done by this paper.

Keywords: Metrics; Cost Estimation; Class Diagram; Software Engineering; Software Project Management (SPM)

1. INTRODUCTION

Accurate software cost estimations are critical to both developers and customers. They can be used for generating request for proposals, contract negotiations, scheduling, monitoring and control of software development and maintenance. Now days, most of the software are failed because of improper estimation. Underestimation leads to delay of software development and failure at the end. Overestimation may affect developers to get new projects in future. As software is not tangible, so the estimation processes itself is a difficult and more challenging task for project managers. A proper balance among overestimation and underestimation is necessary for project managers.

Accurate cost estimation is important because:

a. It can help to classify and prioritize development projects with respect to an overall business plan.

b. It can be used to determine what resources to commit to the project and how well these resources will be used.
c. It can be used to assess the impact of changes and support re-planning.
d. Projects can be easier to manage and control when resources are better matched to real needs.
e. Customers expect actual development costs to be in line with estimated costs.
f. Software cost estimation involves the determination of one or more of the following estimates [5]:
   - effort (usually in person-months)
   - project duration (in calendar time)
   - cost (in dollars or rupees)

Many effort estimation techniques are based on the input of size of the system compared to the level of complexity the system has. Once the effort is known then the duration of software development can be estimated. So, a study on different metrics used for measuring the size of a system is felt to be necessary before the estimation process. As the object oriented paradigm is very popular now days by the practitioners for software development, it is also required divert the study of metrics for object oriented system.

2. LITERATURE REVIEW

2.1 CK Metric-Suite

2.1.1 WMC (Weighted Method per Class)

The WMC is the sum of complexity of methods which are defined in the class. The complexity was originally the cyclomatic complexity [1].
The Weighted Method per Class is defined as follows [2]:

$$WMC = \sum_{i=1}^{n} C_i$$

(1)

Where $C_1, C_2, \ldots, C_n$ be the complexity of methods of a class with methods $M_1, \ldots, M_n$.

WMC measures size as well as logical structure of the software. The number of methods and the complexities of the involved methods are predictors of how much time and effort required to develop and maintain the class. Larger the number of methods in a class, greater the potential impact on inheriting class. Consequently more effort and time are needed for maintenance and testing. Classes with large numbers of method are likely to be more specific, limiting the possibility of reuse. WMC can also be used to estimate the usability and reusability of class. If all methods complexity is considered to be unity, the WMC equals to number of methods [3].

2.1.2 DIT (Depth Of Inheritance)

The depth of inheritance of a class is the DIT metric for class. In case of multiple inheritances, the DIT will be maximum length from the node to the root of the tree [2]. The depth of class within the inheritance hierarchy is maximum number of steps from the class node to root of the tree and is measured by number of ancestor class. The deeper class is in hierarchy the greater the number of methods it is likely to inherit, making it more complex to predict its behaviour. Deeper trees constitute greater design complexity, since more methods and classes are involved. The deeper a particular class is in hierarchy, the greater potential reuse of inherited methods [1].

2.1.3 NOC (Number Of Children)

The NOC metric equals to number of immediate subclasses subordinated to a class in the class hierarchy. Greater the number of children, greater the reuse, since inheritance is a form of reuse. Greater the number of children, greater the improper abstraction of the parent class, it may be a case of misuse of sub classing. The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class [3].

2.1.4 CBO (Coupling Between Object)

Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. Low coupling is preferred in software development. Higher the coupling restricts the reuse of the class and complex the system. More independent class is suitable for reuse [1].

2.1.5 RFC (Response for Class)

RFC count the methods that can be invoked in response to a message to an object of a class. It includes local method as well as methods in other class. If more number of methods are invoked through a message then complexity will increase accordingly. It also measures the amount of communication with other classes [1].

2.1.6 LCOM (Lack of Cohesion)

The Lack of Cohesion is one of the first metric to measure cohesion in object oriented development. The original definition of this metric considers all pairs of a class's methods. In some of these pairs both methods access at least one common field of the class, while in other pairs the two methods to not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that don't share a field access the number of method pairs that do. Note that subsequent definitions of this metric used as a measurement basis the number of disjoint graph components of the class's methods [4].

2.2 LI'S Metrics

2.2.1 NAC (Number of Ancestor Class)

Li proposed the NAC metric to count the total number ancestor classes from which a class inherits in the class inheritance hierarchy. The theoretical basis and viewpoint both are same as DIT metric. The unit for NAC metric is class. This unit is defined with reference to a standard which is class inheritance relation in object oriented programming [4].

2.2.2 NDC (Number of Descendent Class)

NDC metric is defined as the total number of descendent classes or sub classes of a class. The theoretical basis and viewpoint both are same as NOC metric. Li claimed that the NDC metric captures the classes attribute better than NOC [4].

2.2.3 NLM (Number of Local Methods)

NLM metric is defined as the number of local methods defined in the class which are accessible outside the class. The theoretical basis and viewpoints are different from WMC metric. Li stated three viewpoints for NLM metric as following:

The NLM metric is directly linked to a programmer’s comprehension effort when a class is reused in an object oriented design. More effort is required to comprehend the class behaviour for more local method of a class.
More effort is needed to design, implement, test, and maintain the class for larger number of local interface of a class [4].

As larger the local interface of a class, the more influence the class has on its descendent classes.

2.2.4 CMC (Class Method Complexity)

The CMC metric is the summation of the internal structural complexity of all local methods regardless whether they are visible outside the class or not. The theoretical basis and viewpoints are different from WMC metric. NLM and CMC are different from each other as they capture two independent attributes of a class. There are also some commonality in the view points of the two metrics, they affect the effort required to design, implement, test and maintain a class [4].

2.2.5 CTA (Coupling Through Abstract Data Type)

The CTA metric is defined as the total numbers of classes that are used as abstract data types in the data-attribute declaration in the data attribute declaration of a class. Li stated three viewpoints for CTA metric as following:

- It takes more time in understanding the interfaces of the used classes in order to create the design for a high CTA class than a low one.
- More effort is required to design test cases and perform testing for high CTA class than a low one because that the behaviours of the used classes also need to be tested.
- For a maintenance engineer, it also takes more time to understand a high CTA class than a low one because a high CTA class uses more class whose behaviours may comply the class [4].

2.2.6 CTM (Coupling Through Message Passing)

The CTM metric is defined as the total number of different message sent out from a class to other classes excluding the message sent to the objects created as local object in the local methods of the class. Li stated three viewpoints for CTM metric as following:

- It takes more time in understanding the services provided by other classes in a high CTM class than in a low CTM class because the outgoing message are directly related to the services as other classes provide.
- More effort is required to design test cases and perform testing for high CTM class than a low CTM class because a high CTM value means more other classes’ methods are involved in the logical paths of the class.
- For a maintenance engineer, the higher the CTM metric value, the more specific methods in other classes the engineer needs to understand in order to diagnose and fix problems [4].

2.3 LI & HENRY’S Metrics

2.3.1 MPC (Method Invocation to Other Classes)

MPC is defined as the number of send statements defined in class C. This includes only counting methods invocations to other classes, and these classes must not have an inheritance relationship. The number of send statements sent out from a class may indicate how dependent the implementation of the local methods is on the methods in other classes. MPC does not consider polymorphism as only send statements are accounted for, not method definitions which could be polymorphically invoked [1].

2.3.2 DAC (Number Of Attributes Of a Class)

DAC is defined as the number of attributes in a class that have another class as their type. DAC’ is defined as the number of different classes that are used as types of attributes in a class. Briand et al. have found that DAC and DAC’ do not fulfil all the properties for coupling measure proposed by Briand et al. This means that neither DAC nor DAC’ metrics can be classified according to Briand et al.’s framework, which defines the set of properties that length, size, coupling, complexity and cohesion metrics must fulfil. Li and Henry have applied these metrics (and others) to two real systems developed using Classic-ADA. They found that the maintenance effort (measured by the number of lines changed per class in its maintenance history) could be predicted from the values of these metrics (and others like DIT, NOC, etc.) [2].

2.3.3 NOM (Number of Local Methods in a Class)

NOM is defined as the number of local methods in a class. NOM represents the number of methods defined locally in a class, counting public as well as private methods. Overridden methods are taken account too. NOM is a simple metric showing the complexity of a class in terms of responsibilities. However, it does not make the difference between simple and complex methods. WMC is better suited for that. NOM can be used to build ratio based on methods [2].

2.4 MOOD’S Metrics

2.4.1 MHF (Method Hiding Factor)

Method Hiding Factor measure how methods are encapsulated in a class. Visibility is counted in respect to other classes. MHF represent the average among of hiding among all classes in the system. A private method is fully hidden. The number of visible methods is a measure of the class functionality. Increasing the overall functionality will reduce MHF. The implementation will use hidden methods, thus getting information hiding benefits and favouring a MHF increase. A low MHF indicates insufficiently abstracted implementation. A large proportion of methods are unprotected and the probability of errors is high. A high MHF indicates very little functionality. It may also indicate that the design includes a high proportion of specialized methods that are not available for reuse [2].

2.4.2 MIF (Method Inheritance Factor)
MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes [2].

A class that inherits lots of methods from its ancestor classes contributes to a high MIF. A child class that redefines its ancestors' methods and adds new ones contributes to a lower MIF. An independent class that does not inherit and has no children contributes to a lower MIF.

2.5 Lorenz and Kidd’s Metrics

2.5.1 PIM (Public Instance Method)

This metric counts the total number of public instance methods in a class. Public methods are those that are available as services to other classes [2].

2.5.2 NIM (Number of Instance Method)

This metric count all the public, protected, and private methods defined for class’ instances. NIM is a simple measure showing the amount of behaviour that a given class can reuse. It counts the number of methods.

2.5.3 NCM (Number of Class Methods)

This metric counts the total number of class methods in a class. A class method is a method that is global to its instances.

2.5.4 NMO (Number of Method Overridden)

The metric counts the total number of methods overridden by a subclass. A subclass is allowed to define a method of the same name as a method in one of its superclasses. This is called overriding the method [2].

2.5.5 NMI (Number of Method Inherited)

The Number of Methods Inherited metric is the total number of method inherited by a subclass.

2.5.6 NMA (Number of Methods in a Sub Class)

This metric counts the total number of methods defined in a subclass.

3. COMPARISION

3.1 WMC vs CMC & NLM vs MHF, PIM, NIM

As stated by Chidamber and Kemerer, WMC is sum of complexity of methods which are defined in the class. Li issued that two independent attributes cannot be measured at same time i.e. 1) Count of local methods and 2) sum of internal complexity of all local methods. So he proposed two new metric CMC and NLM metrics.

CMC metrics measure the internal structural complexity of a of a class via capturing the complexity of information hiding in the local methods. NLM measures the local method defined in a class, which is accessible to other classes. The private method in a class does not include in this metric, although it shows the properties of object and indicate the size of a class.

Li and Henry proposed the NOM metrics to measure the local methods. To measure the visibility of method MOOD proposed MHF metric. Lorenz and Kidd’s clears the idea about visibility by proposing PIM, NIM, and NCM metric.

3.2 DIT vs NAC vs MIF vs NMI

DIT metric is the maximum length from class node to the root of the tree. Ambiguity comes when it is applied on multiple inheritance or multiple roots, classes that do not inherit from any other classes, are present at the same time. Li claimed that number of ancestor classes of a class could be the best metric to capture the class inheritance relation and proposed NAC metric. Lorenz and kiddy’s proposed NMI metric which simplify the inheritance relationship. The NMI is the total number of method inherited by a subclass.

Above metric only measures the class inheritance relationship. To measure the method inheritance relationship MOOD proposed MIF metric. MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes.

3.3 NOC vs NDC vs NMA

CK proposed that NOC metric is the number of children or number of immediate subclasses subordinated to a class in the class hierarchy. Li claimed that immediate as well as non-immediate subclasses should be considered as a class has influence over all its subclasses. So, he proposed NDC metric considering both immediate and non-immediate subclasses. Lorenz and Kidd proposed NMA metric to measure the method in a subclass.

3.4 CBO vs CTA and CTM vs MPC vs DAC

CBO measures the number of non-inheritance related couples with other classes. Li proposed two new metrics based on different form of class coupling unit definition model: abstract data type and message passing.

CTA defines as the total numbers of classes that are used as abstract data types in the data attribute declaration of a class. Li and Henry proposed two metrics i.e. DAC and MPC. DAC is defined as the number of attributes in a class that have another class as their type. MPC is defined as the number of send statements defined in the class. This includes only counting methods invocations to other classes, and these classes must not have an inheritance relationship. CTM measures the number of different messages sent out from a class to other classes excluding the messages sent to the objects created as local objects in the local methods of the class.

3.5 LCOM1 Vs LCOM2

LCOM1 is defined as the number of pairs of method in the class using no attribute in common where as the LCOM2 is defined as the number of pairs of methods in the class using no attribute is common minus the number of pairs of methods that do. If this difference is negative, however, LCOM2 is set to zero [6].
CONCLUSION AND FUTURE WORK

In this paper we have discussed the advantages as well as shortcomings of several metrics that are associated with cost estimation purposes using class diagram. Some metrics have the edge over the others so; we have to choose the right metric for the better use. Similarly still some metrics are to be explored using ER diagram as well as Use-Case diagram that can be used for cost estimation purposes.

REFERENCES


[6] Lionel C. Briand, John Daly , Victor Porter , Jürgen Wüst , Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems.