Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp174-179)

Cognitive Information Complexity Measure of Object-Oriented Software – A Practitioner's Approach

DHARMENDER SINGH KUSHWAHA and A.K.MISRA
Department of Computer Science and Engineering
Moti Lal Nehru National Institute Of Technology
Allahabad,
INDIA.

Abstract: - Cognitive informatics is an interdisciplinary research area that paves way for finding solutions to problems in the related field such as problem understanding, artificial intelligence, cognitive sciences etc. It studies the information processing mechanism and its relation to the behavior thereupon, may it be system or an individual. Cognition defines the ease of understanding or the property of comprehension. Comprehension is the key feature that distinguishes any entity as being complex or simple. Comprehensibility of a problem helps in efficient design solution and improvement of software product quality. Thus property of comprehensibility can be used in all the different phases of software engineering. Hence, from a practitioner's point of view, we need a object-oriented metric based on cognition that will act as a yardstick in designing efficient software systems.

Keywords: - Cognitive informatics, Problem analysis and design metrics, Functional overlap of classes, Cognitive complexity of inheritance, Cognitive information complexity measure of classes.

1 Introduction

Metrics are a useful means for monitoring progress, attaining more accurate estimation of milestones and developing a software system that contains minimal faults thus improving the quality. Measures are necessary to identify weaknesses of the development process [1]. They also prompt the necessary corrective activities and enable us to monitor the results. Hence they act as feedback mechanism that plays a vital role in the improvement of the software development process. There is an urgent need of software metrics to monitor the software development process for improving the overall quality of the software [2]. Since complexity metrics are a significant and determinant factor of a systems success or failure, there is always a higher risk involved when the complexity measurement is ignored. Software metrics have been used for over three decades to assess or predict properties of software systems, but success has been limited by factors such as lack of sound models, the difficulty of conducting controlled repeatable experiments in educational or commercial context, and the inability to compare data obtained by different researchers.

Object technology is not a magic solution by itself [1]. There is a need to establish some basic standards and guidelines that practitioner should follow. In designing modular and complex software systems, object oriented analysis and design (OOAD) techniques provide many benefits. Despite all its benefits, the OOAD software development lifecycle is by means less difficult than the typical procedural approach. Therefore it is necessary to derive dependable guidelines.

Most of the object-oriented metrics have been defined using only the syntactic aspects of the object-oriented software, producing a single numerical measure, which hardly provides any useful information about the ways of improving the good design and related quality factors [7]. The focus on process improvement has increased the demand for software measures [9]. Given the central role that the software development plays in the delivery and application technology, practitioners are increasingly focusing on process improvement in the software development area.

2 Why Cognitive Complexity Approach?

Wang [11] describes a model showing relationship between NI-OS and NI-APP as one of the foundations of cognitive informatics. Wang [2] defines NI-Sys as a real time natural intelligence [NI] processing system NI-Sys, which is configured by a predetermined operating system NI-OS and a set of acquired life application Ni-App. This model defines cognitive functions each inheriting some properties of the function above it in the hierarchy and interacts with NI-App and NI-Os.

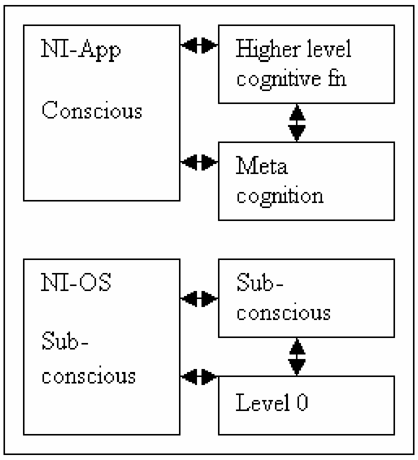

Fig. 1: Relationship Between NI-OS and NI-App.

This model can be interpreted as under:

1. Entities at level 0 communicate and acquire certain features from the entities at level 1 and so on. This corresponds to the abstraction mechanism.
2. It establishes that the various entities communicate via message passing.
3. It shows that these entities (class or object) also access or call global variables, attributes and friend functions.
4. NI-Sys can be mapped into individuals understanding (brain) that includes cognition and semantics relevant to that application. This would prompt the practitioner to construct sensible class definitions.
5. NI-App can be mapped to reusable class library.

We have made an attempt to demonstrate that the object-oriented paradigm is closely related to above model proposed by Wang. Cognitive informatics is still exploring the solutions to a vital question "How does the human natural intelligence process information?" The natural intelligence is derived among many things by self-learning. The self-learning is based on the interaction with the real world. In real world, things around us are objects and the essence of the object-oriented design is based on decomposing the problem / system into objects. Hence there is a close linkage between the cognitive informatics and the object-oriented paradigm and we have made an attempt to establish the linkage as illustrated in fig.2. Hence the abstraction from the above model of cognitive informatics is closely related to the object-orientation concepts and hence, it can be used to draw some useful metrics that will help the object-oriented software developer at various stages of software development.
To establish the linkage and derive the complexity of any object-oriented software, our work is based on the following proposed development model as illustrated in table 1.

Cognitive complexity of the Software Product
Cognitive Complexity of Main Function
Cognitive complexity in Inheritance
Cognitive complexity of Class
Problem Analysis and design metric

Table 1: Proposed Development Model for Cognitive Complexity Measure

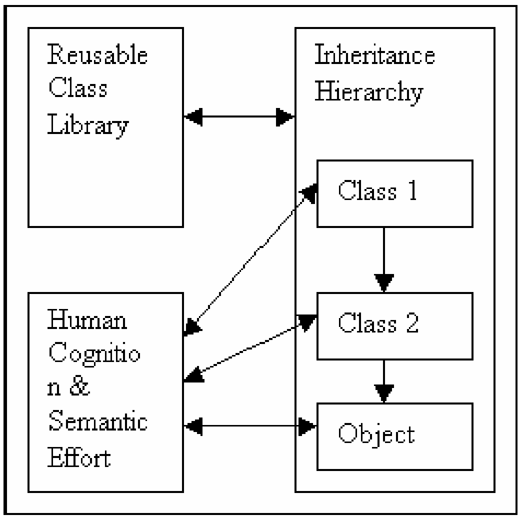

Fig. 2: Cognitive model of Object-oriented Software Development

The term object is sometimes used in this paper to emphasize the close relation and mapping of the real world objects to the object-oriented paradigm. But the terms objects and classes are considered equivalent here and will be used interchangeably throughout this paper.

3 Problem Analysis and Design Metric (PADM)

There are two important techniques to ensure software reliability and quality [2]. One that has been researched for long is by the use of metrics/measures for complexity, but still important and relatively unexplored is to utilize the software metrics to monitor and improve the quality of analysis and design phase of the software development by appropriate metrics.
Our proposed metric for problem analysis and design is based on the following factors that influence OOAD.

1. Number of methods per class (MPC).
2. Reference to other object (RTOO).
   Number of methods that need to access the method or instance variable of another object should be discouraged.
3. Number of independent functions performed by methods in class (IFP) should be encouraged.
4. Number of lines of code per method (LOCPM) should be restricted to ten as suggested in various literature before.
5. Probability of use of instance variable (PUIV) should be more.
6. Amount of functional overlap of classes (FOC) should be reduced to maximum possible extent.

The basis of the above factors is as under.

MPC

1. As the number of methods in a class increase, there shall be a tendency by the practitioner to loose his understanding of the class. For efficient cognition of methods in a class, we propose, based on the comprehensibility and researches, that it should not be more than ten [10].
2. Too many methods per class tend to reduce the probability of class reuse, which is a major factor in O-O development.
3. According to Chidamber [10], most classes tend to have small number of methods (0 to 10).
4. This will also aid in software maintenance and future re-engineering needs.
5. This will reduce the complexity of class as mentioned in section 5.

PUIV

Let a class contain 'i' number of methods. Let there be 'j' number of instance variables per class. Let M_i be the method using N_j number of instance variables. Then probability of method M_i using Nj number of instance variables = Nj/j.

Probability of use of instance variable for all the methods = $\sum_{k=1}^{i} (Nj/j)_k$.

If PUIV > 1, it is a cohesive class and we should design such that high cohesion is achieved. This also indicates how closely the local methods are related to the local instance variables.

FOC

The amount of functional overlap of one class with other class is a important design issue for the practitioners. If there is high overlap, it indicates that the design of functionality breakdown structure is poor. Let a class A perform functions f1, f2, f3. Let a class B perform functions f2, f4, f6. The functional overlap is given by $A \cap B$ = f2. A good design should aim to eliminate these functional overlaps.

Having examined the factors that determine and affect our design decisions, we now introduce the factors and their impact on design decisions using their impact weight when the above factors assume true or false value.

S.No	Factor	True	False
1	MPC ≤ 10	0	1
2	RTOO ≤ 0	0	1
3	LOCPM ≤ 10	0	1
4	PUIV > 1	0	1
5	FOC ≤ 0	0	1

Measuring the weights of the factors as listed in the table gives us a fair indication of our OOAD process. Therefore if MPC+RTOO+LOCPM+PUIV+FOC =0, our design strategy is simple and efficient.

4 Cognitive Complexity of Inheritance (CCI)

Inheritance is the most important feature for software reuse and it supports the class hierarchy design and captures the is-a relationship between a class and its sub-class [2]. Results show that multiple inheritances are rarely used [1]. The average level of inheritance one to two of the newly developed classes indicate that object-oriented developers involved in the development activities utilize reusable class library. The inheritance hierarchy is a directed acyclic graph that can be described as a tree structure with classes as nodes, leaves and root [9]. The design decision is based on the ratio between depth and breadth of the inheritance tree. By measuring the depth of the inheritance tree, one will be able to ascertain the level of class abstraction. This will help the practitioner in deciding upon the proper level of specialization and dependence amongst the similar classes in a system. The idea is that if a class has a large number of immediate children, there shall be increase in the message transfer among them. This will have a larger influence on system design and will make testing more complex [5]. It is also to be noted that if a class has a large number of children, it also indicates improper abstraction, but a greater amount of reuse in system. There are also instances when a class cannot respond to a message (i.e. it lacks a corresponding method of its own), then it will pass the message on to its parents and this fact should be captured by metrics concerned with the object-oriented development.

Wang [11] proposed one of the properties in informatics laws of software based on information entropy. The entropy theory has been applied to measure the complexity of the software. Entropy has been used as a measure of uncertainty. According to Shanon's definition, the higher the uncertainty associated with the signal, the greater is the amount of information conveyed by the signal. The entropy increases with the increase in the messages [5]. Value of complexity measure for the program consisting of two objects is higher than that of the complexity measure for the program consisting of one object.

Let 'x' be the number of messages sent or received by an object O_x. Let 'n' be the number of total messages exchanged within the inheritance tree. The reference probability of object $O_x = x/n$. Entropy $H = -[x/n*\log(x/n)]$.

Therefore cognitive complexity of inheritance (CCI) for all the objects/classes in the inheritance tree is:

$$CCI = [\sum_{o=1}^{k} -\{(x/n)\log(x/n)\}_o]$$

Where o = number of objects $O_1, O_2 \ldots\ldots\ldots O_x$
x = number of messages related to object O_x
n = total number of messages.

Hence we can assert that the inter-object complexity measure depends on the number of objects and the number of messages exchanged between the objects. Therefore entropy measure can be used by the practitioners to resolve the conflict between the improper abstraction and reuse, by designing such inheritance hierarchies so as to limit the entropy and hence the complexity. This will reduce the testing efforts required and increase the probability of class reusability.

Entropy Reduction

Entropy can be reduced by one of the following two methods:

1. Reducing the number of messages and
2. Reducing the length of messages i.e. the number of characters / bits in the messages.

5 Cognitive Information Complexity Measure of Classes (CICMC)

Since the code inside a method is not distinguished as being procedural or object-oriented, we calculate the complexity of a class by calculating the cognitive complexity of each method in a class by cognitive information complexity measure (CICM) [3]. This measure is computationally simple and a robust one [4], since it adheres to all the nine Weyuker properties.

The CICM defines complexity as WICS * SBCS.
WICS is defined as

$$WICS = \sum_{k=1}^{LOCS} WICL_k$$

SBCS is defined as

$$SBCS = \sum_{i=1}^{n} (W_i)$$

Where $W_1, W_2 \ldots W_n$ be the cognitive weights of the basic control structures [3].
WICL is defined as ICS_k / [LOCS – k],

Where
$$ICS = \sum_{k=1}^{LOCS} I_k$$
Where I_k = Information contained in k_{th} line of code,
LOCS = Total lines of code in the software.

Since a class consists of number of methods, the complexity of a class is calculated by calculating the complexity of each method contained in a class.

Let $M_1, M_2, M_3 \ldots M_n$ be the methods in a class.
Let $CICM_1$, $CICM_2 \ldots CICM_n$ be the CICM of each method.

Then CICM of the class is defined as $\sum_{i=1}^{n} (CICM)_i$

The definitions of our metrics are based on the concept of object-orientations and hence are independent of the object-oriented programming language used. Thus CICMC will guide the developer in designing classes such that the class complexity is reduced and hence improving its reusability and maintainability.

6 Cognitive Information Complexity of Main Function (CICMF)

Since the main function of any object-oriented software resembles a typical procedural program containing program body which is composed of various basic control structures, its complexity is calculated using the cognitive information complexity measure (CICM) as described in section 5.

7 Total Complexity of Object – Oriented Software Product

Since object-oriented software are composed of all the aspects that have been derived above, the product complexity is calculated as PUIV + CCI + CICMC + CICMF.

8 Conclusion

Optimization of object-oriented software and improvement of its quality cannot be done after the development process. In order to produce high quality software, there is a need of metrics being used at all the stages of software development process. Our measure though computationally simple, will help the practitioner at all the stages of software development process right from analysis to the optimization of the finished product. This will reduce the rework and backtracking saving effort and time.

References:

[1] B.Stiglic, M.Hericko, and I. Rozman "How to Evaluate Object-Oriented Software Development", *ACM SIGPLAN Notices*, Vol. 30, No. 5, May 1999.

[2] C.Chung, and M.Lee "Inheritance based Object-Oriented Software Metrics", *IEEE Region 10 Conference,* Nov 1992, Melbourne, Australia.

[3] Kushwaha D.S.and.Misra A.K "A Modified Cognitive Information Complexity Measure of Software", *ACM SIGSOFT Software Engineering Notes, Vol.* 31, No. 1, January 2006,

[4] Kushwaha D.S.and.Misra A.K "Evaluating Cognitive Information Complexity Measure ", *13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)* To Appear, 2006.

[5] K.Kim, Y.Shin and C.Wu "Complexity Measures for Object-Oriented Programs Based on the Entropy", *Proceedings of the Second Asia Pacific Software Engineering Conference, IEEE Computer Society*, 1995.

[6] K.S.Mathias, J.H.Cross II, T.D.Hendrix and L.A.Barowski "The Role of Software Measures and Metrics in Studies of Program Comprehension", *ACM Southeast Regional Conference,* 1999.

[7] L.Etzkorn and H.Delugach "Towards a Semantic Metric Suite for Object-Oriented Design", *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00) IEEE Computer Society,*2000.

[8] S.Purao and V.Vaishnavi "Product Metrics for Object-Oriented Systems", *ACM Computing Surveys,* Vol. 35, No. 2, June 2003, pp 191-221.

[9] S.R.Chidamber and C.F.Kemerer "Towards a Semantic Metric Suite for Object Oriented Design", *IEEE Transactions on Software Engineering,* Vol. 20, No. 6, June 1994.

[10] S.Xu, V.Rajlich and A.marcus "An Empirical Study of Programmer Learning during Incremental Software Development".

[11] Y.Wang "On Cognitive Informatics", *IEEE International Conference on Cognitive Informatics (ICCI'02)",* 2002.