Software Readability Metric

Anamika Maurya
Department of MCA
PSIT, Kanpur, India

ABSTRACT

Software products are costly as compare to hardware. And developing software products at nominal cost are always a big issue for project managers. Therefore they desperately look for way-outs to cut development cost. While developing Software, its quality has become essential from the client point of view. So, Software understandability is vital and one of the most significant components of the software quality. The lack of understandability aspect often leads to false interpretation that may in turn lead to ambiguities, misunderstanding and hence to faulty development results. It plays an important role as far as the issue of delivering quality software is concerned. Therefore, Understandability is obviously relevant and significant in software maintenance. Software metrics can be derived using Class Inheritance Directed Acyclic Graph(CIDAG) approach to measure the understandability. In our approach as DIT is combined with predecessor and successor of class, the values of understandability metrics are higher in comparisons to existing approach. Our approach proposes a metrics for understandability measurement based on class inheritance, in an efficient way.

General Terms

Software Engineering, Algorithms et. al.

Keywords

Understandability, DIT, NOC, CIDAG

1. INTRODUCTION

Software engineering is much more distinct with other established branches of engineering, because of shortage of measuring units, lack of well accepted measures or metrics for software development. With the lack of such measuring units, software development and it's maintenance would have been stagnant in craft type models. To overcome these drawback, great experience, skill is required for study, adoption and for further improvement. Software can be quantitatively described with the help of metrics and the use of tool on the projects, productivity and quality can be evaluated.

Measurement is fundamental to any engineering discipline, so in software engineering. Software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. Typically metrics are essential to software engineering for measuring software complexity and quality, estimating cost and project effort. The traditional metrics like function point, software science and cyclomatic complexity have been used in procedural paradigm.

Understandability, is an essential activity of software maintenance and software quality. The increase in size and complexity of software drastically affects several quality attributes, especially Understandability and maintainability. Changes to software systems are called software evolution in the research field software maintenance. Changes to reuse software systems can be considered as evolution of reused software system. Therefore, Software Understandability can be placed as a factor of software evolution in reuse or maintenance. For understandability sub characteristics to comprehend software , the factors that influences are an internal process of human and an internal software quality itself. Software developers and maintainers needs to read and understand source programs and other documents of software. Understandability of software is thus important as ' the better we know what the things is supposed to do, the better we can test for it [19].' It is not easy to measure software understandability because understandability is an internal process of human.

Despite the fact that understandability is vital and highly significant to software development process, it is poorly managed [1]. The fundamental reality of measurement ' we cannot control what we cannot measure' highlights the importance and significance of good measure of software understandability [2]. This paper has been organized into the following sections. section 2 describes the related work done for software metrics measurements. section 3 describes the proposed approach. Section 4 presents implementation details and comparison results. Finally, Section 5 includes the conclusion and future directions of the paper.

2. RELATED WORK

Researchers have been discussing for decade, whether a separate set of OO software metrics is needed and what this set should include [3]. Initial proposal focused more towards extension of existing software metrics for procedure-object-oriented programming [4,5]. However almost all the recent proposals have been focused on OO programming [6-10]. Since the proposal of the six OO metrics by Chidamber and Kemerer (CK) [8] in 1991. Other researchers have made efforts to validate the metrics both theoretically and empirically. CK's revised paper [9] proposed a suit of OO metrics which have a set of six simple measures:

1. WMC: Weighted methods per class, which counts number of methods in a class
2. DIT: Depth of inheritance tree, which is the number of ancestor classes that can affect a class
3. NOC: Number of children, which is the number of subclasses that inherit the methods of a parent class
4. CBO: Coupling between classes which is a count of the number of other classes to which it is coupled
5. RFC: Response for a class, which is a set of methods that can be executed in response to a message received by an object of that class
6. LCOM: Lack of cohesion in methods, which is a count of the inter-relatedness between portions of a program

These metrics were evaluated analytically against

Weyuker's measurement theory principles [10] and an empirical sample of these metrics was provided from two commercial systems. Several studies have been conducted to validate CK's metrics. Basili et al. [9] presented the result of an empirical validation of CK's metrics. Their results suggest that five of six of CK's metrics are useful quality indicator for predicting fault-prone classes. Tang et al. [11] validated CK's metrics using real-time systems and the results suggest that WMC can be good indicator for faulty classes and RFC is a good indicator for faults. Li [12] theoretically validated CK's metrics using a metric evaluation framework proposed by Kitchenham et al. [17]. He discovered some of the deficiencies of CK's metrics in the evaluation process and proposed a new suit of OO metrics that overcome these deficiencies.

CK's DIT and NOC definition

DIT metrics : The depth of inheritance of a class is the DIT metric for the class. In case involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.

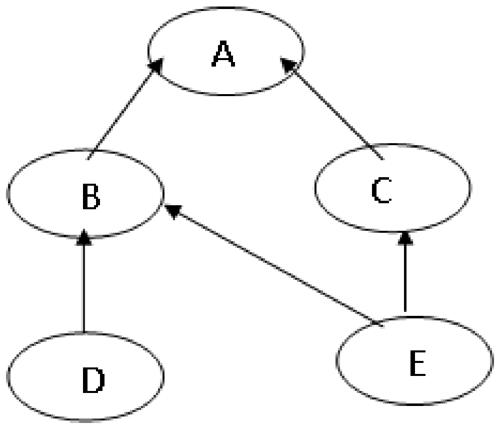

Figure. 1. A class Inheritance Tree

Theoretical basis : The DIT metric is a measure of how many ancestor classes can potentially affect this class.

(1) The deeper a class is in the hierarchy, the higher the degree of methods inheritance, making it more complex to predict its behavior.

(2) Deeper trees constitute greater design complexity , since more methods and classes are involved.

(3) The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

Example. Consider the class inheritance tree in Figure 1.

DIT(A) = 0 because A is the root class

DIT(B)=DIT(C)=1 because the length from class B andC to the root A is one each.

DIT(D)=DIT(E)= 2 because the maximum length from class D and E to the root A are two each.

NOC Metric: NOC is the number of immediate subclasses , subordinate to a class hierarchy.

Theoretical basis: NOC is a measure of how many subclasses are going to inherit the methods of parent class.

(1) The greater the number of children, greater the potential for reuse, since inheritance is a form of reuse.

(2) The greater the number of children, the greater the likelihood of improper abstraction of the parent class.

(3) The number of children gives an idea of the potential influence a class has on the design.

DIT indicates the extent to which the class is influenced by the properties of it's ancestors and NOC indicates the potential impact on the descendants. CK argue that depth is preferred to breadth in the hierarchy.

DAG Approach

The term class inheritance tree is not valid because if it has multiple inheritance it is not a tree but graph. Frederick T. et al. [13], used The most suitable mathematical model for describing an object taxonomy with inheritances is directed acyclic graph (DAG) with no loops [16]. Therefore, the notation of class inheritance DAG as in Figure 2.

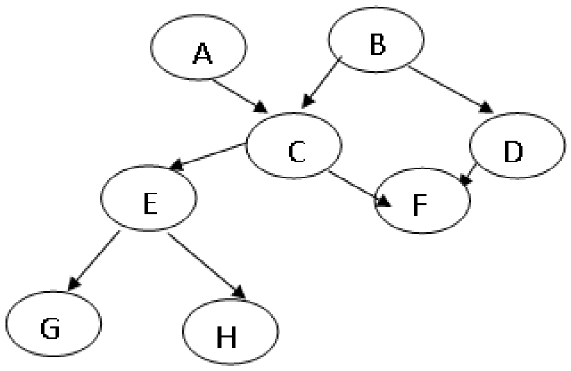

Figure. 2. A Class Inheritance DAG

For the definition of metrics for maintainability, they used the terms from graph theory. In the directed graph, where vertices represent the activities and edges represent the preceding relationship, vertex i is a predecessor of j under the following conditions. Function PRED (Predecessor) and SUCC (Successors) can be defined as follows, if there exists a path from vertex i to j, and vertex j [16]

- PRED(j) : the total number of predecessor of node i
- SUCC(i) : the total number of successors of node i

DAG approach has been used to propose to measure metrics of understandability.

3. PROPOSED WORK

In context of class inheritance, DAG approach has been used to design class inheritance graph. Understandability, DIT has been calculated with slight modification in existing formula.

Consequently, the Degree of Understandability (U) of a class is defined as follows :

$$U\ of\ class_i = PRED(C_i) + 1 \qquad (1)$$

Where Ci is ith class. The Total Degree of Understandability

(TU) of a Class Inheritance DAG (CIDAG) is defined as follows:

$$TU\ of\ CIDAG = \sum_{i=1}^{t} PRED(C_i) + 1 \qquad (2)$$

Where t is the total numbers of classes in the CIDAG

The DIT of a class is calculated as

$$DIT = \frac{\sum Depth\, of\ each\, class}{Number\ of\ class} \qquad (3)$$

With slight modification in above formula new formula is defined as and combining it with DIT

$$TU = \sum_{i=1}^{t} DIT * (PRED(C_i) + SUCC(C_i)) + 1)$$

(4) Where Ci is ith class, and Where t is the total numbers of classes in the CIDAG.

In above approach predecessor and successor both has been considered along with Depth Inheritance Class (DIT).

Coupling between the classes can be derived using following formula:

$$C = (1 - 1/\, d_i + 2 * C_i + d_o + 2 * c_o + g_d + 2 * g_c + w + r)$$

(5) Where d_i = number of input data parameters

C_i = number of input control parameters

d_o = number of output data parameters

c_o = number of output control parameters

g_d = number of global variables used as data

g_c = number of global variables used as control

w = numbers of modules called (fan-out)

r = numbers of modules calling the module under consideration (fan-in)

The proposed approach in this paper includes the following steps.

1. Draw the DAG for class inheritance for project.
2. Calculating TU and DIT using equation 3.2, 3.3 and 3.4.
3. Calculating coupling for each projects using equation 3.5.

The case study of Queue Technique for Rail Road Project is explained here. The CIDAG is shown in Figure 3.

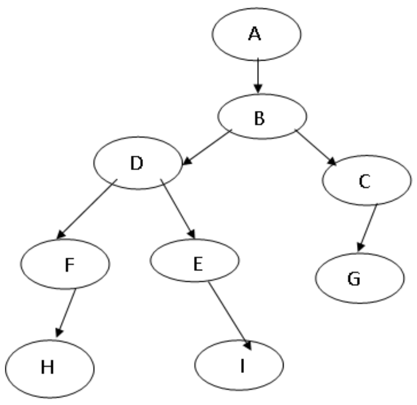

Figure 3. CIDAG for Queue Technique for Rail Road Project

There are total 9 classes in CIDAG and understandability of each class is calculated using equation 1 as follows:

U(A) = 1, U(B) = 2, U(C) = 3, U(D) = 3, U(E) = 4, U(F) = 4, U(G), = 4 , U(H) = 6, U(I) = 6

Hence, Total Understandability from equation 2 is as follows:

= (1 + 2 + 3 + 3 + 4 + 4 + 4 + 6 + 6)/9

= 3.66

DIT for each class is calculated using equation 3 as follows:

DIT(A) = 0, DIT (B) = 1, DIT (C) = 2, DIT (D) = 2, DIT (E) = 3, DIT (F) = 3, DIT (G), = 3, DIT (H) = 4, DIT (I) = 4

Hence Total DIT = (0 + 1 + 2 + 2 + 3 + 3 + 3 + 4 + 4)/9

= 2.44

Understandability for each class is calculated using equation 4 as follows:

U(A) = 1, U(B) = 4, U(C) = 5, U(D) = 7, U(E) = 7, U(F) = 7, U(G), = 4 , U(H) = 5, U(I) = 5

Hence, Total Understandability

= (1 + 4 + 5 + 7 + 7 + 7 + 4 + 5 + 5)/9

= 5

4. CASE STUDY

Proposed approach is illustrated by ten open source C++ based and self made projects. Understandability is measured using modified approach as well as through existing approach.

The proposed approach has implemented in following phases.

In first phase, DAG based design for class inheritance is designed for 15 C++ language projects shown in Table 1

Table 1. Results of Existing and Propose Approach

Projects	Understandability (Frederick T. et al)	DIT	Understandability (Proposed approach)	Coupling
P1	3.66	2.44	5	0.9588
P2	3.64	1.86	5.22	0.952
P3	2.63	1.75	4.25	0.9307
P4	2.79	1.78	3.5	0.9569
P5	3.67	2.66	5.66	0.9576
P6	2.83	1.83	3.7	0.9756
P7	5.35	4.41	8.88	0.9772
P8	3.06	2.06	4.06	0.9712
P9	4.2	3.15	6.5	0.975
P10	3.12	2.11	3.22	0.9384

In second phase metrics for Understandability and DIT measured for all these project using the standard definition given by Frederick T. Sheldon, Kshamta Jearth and Hong Chung [1]. With proposed modification in standard formula like only considering predecessor of class , both predecessor and successor were taken into consideration and metrics for Understandability and DIT were measured.

In third phase, coupling for all projects were calculated. Correlation Coefficients between coupling for all projects and Understandability metrics for above approach and new approach were measured.

Results of all projects are shown in Table 4.1. Graph for all projects between Understandability for existing and proposed approach is also shown in Figure 4.3.

Table 4.1 shows the detailed results of implementation. Here P1 to P15 are the 15 C++ projects on which approach have been applied. Understandability with Existing approach by Frederick T. et al , along with DIT, and Understandability with proposed approach is being shown. The value of Understandability ranges between lowest 2.67 to highest 5.35 for existing approach whereas Understandability ranges between lowest 3.5 to highest 8.88 for proposed approach in a scale of 0 to 10.

5. RESULT AND COMPARISON

Proposed approach was compared with Frederick T. Sheldon, Kshamta Jearth and Hong Chung [1] approach. As in the proposed approach DIT is combined, and also Predecessor and successor of the class has been considered, its value is higher with comparison to Frederick T. et al approach.

Correlation Coefficients between coupling for all projects and Understandability were also compared in table 2. From this comparison we can say that, understandability can be measured with metrics using proposed approach and our approach is more efficient in terms of correlation between understandabilty and coupling.

Table 2. Correlation between understandability and Coupling for Existing and proposed Approach

Approach	Correlation Coefficient
Frederick T.et al.	0.40
Proposed Approach	0.50

Figure 4 shows the graph between Understandability for existing approach and proposed approach for all 15 projects. Here U represents understandability for existing approach and NU represents understandability for proposed approach.

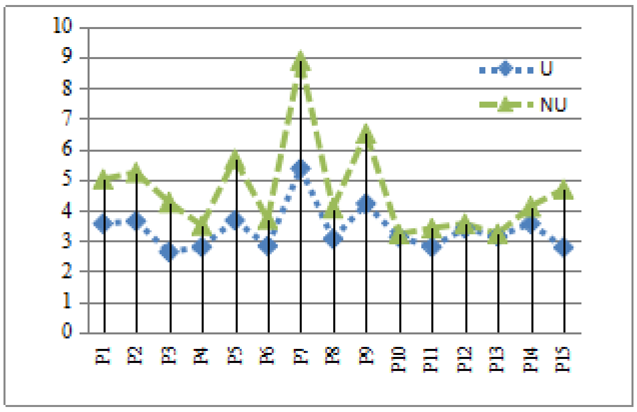

Figure 4. Graph between Understandability for existing and proposed approach

It is clear from the graph that there are similarities between both the approaches. The value of understandability is higher for proposed approach for almost all the projects, because DIT is combined in proposed approach with predecessor and successor both.

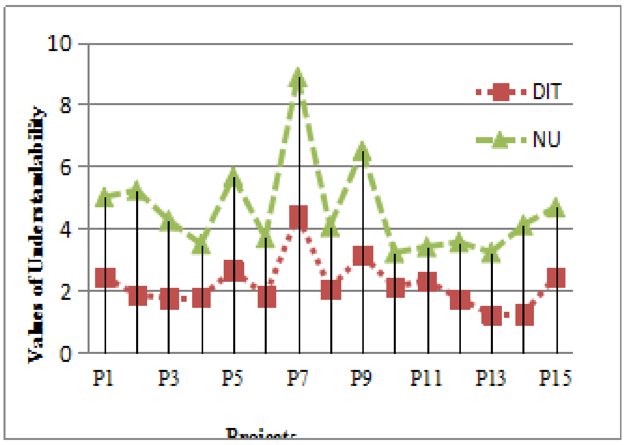

Figure 5. Graph between Understandability and DIT for proposed approach

Figure 5 shows the graph between Understandability for proposed approach, represented by NU and DIT for all 15 projects. It is clear from the graph that the values of understandability are varying in a uniform manner with the values of DIT. If the value of DIT is low the value of understandability is also low and if value of DIT is higher the value of understandability is also higher for all the projects. Hence we can say that DIT has an important role in measuring the understandability metrics.

6. CONCLUSION

Software understandability is vital and one of the most significant components of the software maintenance. The lack of understandability aspect often leads to false interpretation that may in turn lead to ambiguities, misunderstanding and hence to faulty development results. It plays an important role as far as the issue of delivering quality software is concerned. Therefore, Understandability is obviously relevant and significant in software maintenance.

From this discussion, it is clear that, measuring Understandability is important factor in software maintenance. Software metrics can be derived using Class Inheritance Directed Acyclic Graph(CIDAG) approach to measure the understandability. In our approach as DIT is combined with Predecessor and successor of class, the values of understandability metrics are higher in comparisons to existing approach. Our approach proposes a metrics for understandability measurement based on class inheritance, in an efficient way.

7. REFERENCES

[1] K.K Aggarwal, Y. Singh and J.K Chhabra, 2003. A Fuzzy Model for measurement of software Understandability “, International Symposium on Performance Evaluation of Computer & Telecommunication Systems, Montreal, Canada.

[2] T. DeMarco, 1982. Controlling Software Projects, Englewood Cliffs, NJ, Yourdon Press.

[3] Tahvildari L, 2000. Singh A. Categorization of Object-Oriented Software Metrics. IEEE Computer Society.

[4] Tegraden DP, Sheetz SD Monarchi DE. 1995. A software complexity model of object-oriented systems. Decision Support Systems.

[5] McCabe TJ, Dreyer LA, Dunn AJ, Watson AH. 1994. Testing an Object-Oriented Application. Quality Insurance Institute.

[6] Abreu F . 1995. The MOOD metrics set. Procedings of the 9th European Conference on Object-Oriented Programming. Workshop on Metrics. Springer: Berlin.

[7] Briand LC, Morasoa S. 1999. Defining and validating measures for object-based high design.IEEE Transation on Software Engineering.

[8] Chidamber SR, Kemerer CF. 1991. Towards a metric suite for object-oriented design. Proceeding of the Conference on Object-Oriented Programming systems, Languages, and Applications. ACM Press: New York NY.

[9] Chidamber SR, Kemerer CF. 1994. A metrics suite for object-oriented design. IEEE Transe. On Software Engineering.

[10] Weyuker EJ. 1988. Evaluating software complexity measures. IEEE Trans. On software Engineering

[11] Tang MH, Kao MH, Chen MH. 1999. An empirical study on object oriented metrics. Proceedings 23rd Annual International Computer Software and Application Conference. IEEE Computer Society; 242-249.

[12] Li W. Another metric suite for object-oriented programming. The Journal of Systems and Software 1998.

[13] Frederick T. Sheldon, Kshamta Jerath and Hong Chung. 2001. Metrics for maintainability of class inheritance hierarchies. Journal of software maintenance and evolution : Research and Practice.

[14] Rajib Mall. 2009. Fundamentals of Software Engineering. Prentice Hall, 3rd edition.

[15] Wang CC, Shih TK, Pai WC. 1997. An automatic approach to object-oriented software testing and metrics for C++ inheritance hierarchies. Proceedings International Conference on Automated Software (ASE'97). IEEE Computer Society Press

[16] Kitchenham B, Pfleeger SL, Fenton NE. 1995. Towards a framework for software measurement validation. IEEE Trans. On Software Engineering.

[17] I. Sommerville, 1996. Software Engineering, 5th Edition, Addition Wesely.

[18] B.Jacob, L. Niklas P. Waldermarsson, 2001. Relative Indicators for Sucess in Software development Department of Communication Systems, Lund University.

[19] K K Aggarwal & Yogesh Singh, 2007. Software Engineering (3rd ed.) New Age International Publishers.