EVALUATION OF SOFTWARE UNDERSTANDABILITY USING SOFTWARE MATRICES

Apeksha Nayak[*],Anurag Chandna[1],Ankur Goel[2]

[*]M.Tech Scholar, Department of Computer Science and Engineering, Roorkee College of Engineering Uttarakhand Technical University, Dehradun India

[1]Asst. Professor, Computer Science & Engineering, Roorkee College of Engineering, Roorkee, India,

[2]Asst. Professor, Computer Science & Engineering, Roorkee College of Engineering, Roorkee, India

ABSTRACT

Understandability is one of the important characteristics of software quality, because it may influence the maintainability of the software. Cost and reuse of the software is also affected by understandability. In order to maintain the software, the programmers need to understand the source code. The understandability of the source code depends upon the psychological complexity of the software, and it requires cognitive abilities to understand the source code. The understandability of source code is get effected by so many factors, here we have taken different factors in an integrated view. In this we have chosen rough set approach to calculate the understandability based on outlier detection. Generally the outlier is having an abnormal behavior, here we have taken that project has may be easily understandable or difficult to understand. Here we have taken few factors, which affect understandability, and brings forward an integrated view to determine understandability.

Keywords: *Understandability, Rough set, Outlier, Spatial Complexity.*

1-INTRODUCTION

Software products are expensive. Therefore, software project managers are always worried about the high cost of software development, and are desperately looking for way-outs to cut development cost. A possible way to reduce development cost is to reuse parts from previously developed software. In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality. When programmers try to reuse code which are written by other programmers, faults may occur due to misunderstanding of source code. The difficulty of understanding limits the reuse technique. On Software Development Life Cycle (SDLC) the maintenance phase tends to have a comparatively much longer duration than all the previous phases taken together, obviously resulting in much more effort. It has been reported that the amount of effort spent on maintenance phase is 65% to 75% [5]of total software development.

In Figure 1.1, the programmers of the original system were absent, then the other programmers need to reuse the components to enhance the functionalities and correcting faults[16]. Fig.1.1 shows the communication between programmers and software, in the evolution of software systems. Programmer 1 writes the current version of a software system, programmer 2 evolves next version of that software from the current version[14]. If it is difficult to understand, changes to it may cause serious faults, these changes may cost more time than remaking the software systems. However, it is not easy to measure software understandability because understanding is an internal process of humans.

Object-Oriented programming (OOP)languages like C++, Java, C#, Python, Visual Basic etc. supports the concept of reusability. Reuse of the something that already existed is always nice rather than to create the same all over again. Reusability feature may increase the reliability, decrease the cost and time.

Objectives

1- We want to evaluate software understandability of different projects, by using different metrics.

2- For the evaluation of software understandability, we have followed different approaches.

3- To propose new metric, which can be used in evaluation of software understandability.

2-BACKGROUND

Understandability of software also requires few metrics. Here few metrics of code understandability[3] are explained which are used by many organizations. Source code readability, quality of documentation, should be taken into account while measuring the software maintainability.

LOC: A common basis of estimate on a software project is the LOC(Lines of Code). LOC are used to create time and cost estimates.

Comment percent: RSM(Resource Standard Metrics) counts each comment line. The degree of commenting within the source code measures the care taken by the programmer to make the source code and algorithms understandable. Poorly commented code makes the maintenance phase of the software life cycle an extremely expensive.

$$\text{comment percent} = \frac{\text{comment line count}}{\text{logical line count}} \times 100$$

Logical Line Count = LOC + Comment Lines + Blank Lines

LEN* Length of names: If the names of procedures, variables, constants etc are long, then the more descriptive they probably are.

UNIQ measures the uniqueness of all names.

UNIQ = Number of unique names / total number of names

Function Metrics: In this we can measure the number of functions and the lines of code per function. Functions that have a larger number of lines of code per function are difficult to comprehend and maintain. They are a good indicator that the function could be broken into sub functions whereby supporting the design concept that a function should perform a singular discrete action.

Function Count Metric: The total number of functions within your source code determines the degree of modularity of the system. This metric is used to quantify the average number of LOC per function, maximum LOC per function and the minimum LOC per function. In addition to the function count, we may consider Average lines of code, maximum LOC per function, minimum LOC per function metrics are also used.

Macro Metrics [2]: Macro will make your less understandable and difficult to maintain. As macros are expanded prior to the compilation step, most debuggers will only see the macro name and have no context as to the contents of the macro, therefore if the macro is the source of a bug in the system, the debugger will never catch it. This condition can waste many hours of labor.

Class Metrics: In this we can measure the number of classes and the lines of code per class can be taken. The number of classes in a system indicates the degree of object orientation of the system. In addition to this, we determine the average lines of code per class, maximum LOC per class and minimum LOC per class.

Code and Data Spatial Complexity [7]: Spatial ability is a term that is used to refer to an individual cognitive abilities relating to orientation, the location of objects in space, and the processing of location related visual information.

Code spatial complexity: To compute the code-spatial complexity, the module is considered as the basic unit, as every module is defined at one place, but is called many times

code-spatial complexity of a module (MCSC) is defined as average of distances (in terms of lines of code) between the use and definition of the module.

$$\text{MCSC} = \sum_{i=1}^{n} \text{Distance}_i / n$$

where 'n 'represents count of calls/uses of that module.

Distance is equal to the absolute difference in number of lines between the module definition and the corresponding call/use.

Class method spatial complexity: Every class consists of many methods, A method basically means a function/subroutine in any language containing some processing steps.

Distance = (distance of definition from the top of the file containing definition)+(distance of declaration of the method from the top of the file containing declaration).

3- MATERIAL AND METHODS

The following factors affect the understandability of Source code [16]

Comment Ratio

Comment ratio (CR) is used to judge the readability of the source code. Comment Ratio can be calculated, by using the number of commenting lines to the total number of lines of code.

$$\text{Comment ratio(CR)} = \frac{\text{comment lines}}{\text{lines of code}}$$

Fog Index

The quality of documentation is judged using the Fog index. The Gunning's fog index is a metric that is used to measure the readability of the document. The Gunning's fog index of a document can be calculated as:

$$\text{Fogindex} = 0.4 \times \left[\left(\frac{\text{word}}{\text{sentence}}\right) + 100\left(\frac{\text{complex words}}{\text{words}}\right)\right.$$

The Number of Components

Generally, software consists of a large number of components, to perform different functionalities. In Literature[14], when a worker needs to reuse/reconstruct one software system, the number of attempts needed to reconstruct one software system is the comprehension of the worker. The number of components is correlated with the understandability of software.

Cognitive Functional Size

Lin. et al [16] developed the cognitive functional size on the basis of the cognitive weights. They had assigned some weights to the basic control structures, which represent the information flow of the program.

Halstead complexity

This complexity metric was introduced by Maurice Howard Halstead in 1977 [11]. Understandability of the code can be evaluated by using this metric.

This metric was proposed to measure the complexity of the source code as

$$H = n1 \times \log n1 + n2 \times \log n2$$

$N = N1 + N2,$

n = n1 + n2,

$V = N \times \log(n1 + n2)$,

$D = \left(\frac{n1}{2}\right) \times \left(\frac{N2}{n2}\right)$,

Code and Data Spatial Complexity

Chhabra et al. [8][7] presented two measures of spatial complexity, which are based on two important aspects of the program–code as well as data. They calculated the spatial complexity which was the lines of code between the definition and use of the code and data. From their experimental results, it is observed that the lower is the DMSC value better is the understandability.

Basic Concepts

In rough set terminology, a data table is also called an information system. Let IS = (U,A) be an information system, where U is a non-empty set of finite objects(the universe) and A is a non-empty finite set of attributes so that a:U → Va for every a ∈ A with Va being called value set of a. For any P ⊆ A, there exists an associated equivalence relation IND(P):

IND(P) = {(x,y)∈ U 2 |a∈ P, a(x)=a(y)}

The partition generated by IND(P) is denoted by U/IND(P) or abbreviated to U/P and is calculated as follows:

U/IND(P) = ⊗ { a ∈ P | U/IND(a)}.

Rough Entropy

Given an information system IS = (U,A,V,f), where U is a non-empty finite set of objects, A is a non empty finite set of attributes. For any B ⊆ A, let IND(B) be the equivalence relation as the form of U/IND(B) = {B1,B2,B3,...Bm}. The rough entophy E(B) of equivalence relation IND(B) is defined by

$$E(B) = -\sum_{i=1}^{m} \frac{|B_i|}{|U|} \log \frac{1}{|B_i|}$$

where $|B_i|/|U|$ denotes the probability of any element x ∈ U being in equivalence class Bi ;1 ≤i ≤ m. And | U |denotes the cardinality of set U.

Rough Entropy Outlier Factor [15]

In this method we are using the rough entropy outlier factor (REOF), which can indicate the degree of outlierness for every object of the universe in an information system. Let IS = (U,A) be an information system, the rough entropy outlier factor REOF(x) of object x in IS is defined as follows.

$$REOF(x) = \frac{(\sum_{j=1}^{k} RE_{aj}(x) \times W_{aj}(x))}{k}$$

Where REaj (x) is the relative rough entropy of object x, for every singleton subset aj ∈ A, 1≤ j ≤ k. For any a ∈ A, Wa: U→(0,1] is a weight function such that for any x∈U, Wa(x) = 1- | [x]a |/|U|

Let IS= (U,A) be an information system, and v be a given threshold value. For any object x∈U, if REOF (x) > v, then an object x is called a RE-based outlier in IS, where REOF(x) is the rough entropy outlier of x in IS.

4-IMPLEMENTATION & RESULT

Here in this implementation we have considered the metrics like Fog Index, CR, Components, CFS, Complexity, DMSC. The information in the table of 9 different projects each with 6 attributes. Here we transformed the data into required format which is used in rough set theory. The values for an attributes are normalized based on the mean and standard deviation of the attributes. Here

Original Data Table 1

p	Fog Index	CR	Components	CFC	Complexity	DMSC
1	**7**	**5**	**42**	**96**	**0.35**	**0.22**
2	**9**	**7.2**	**50**	**109**	**0.5**	**0.24**
3	**10**	**8**	**56**	**101**	**0.54**	**0.35**
4	**9**	**7**	**64**	**99**	**0.61**	**0.38**
5	**8**	**8**	**72**	**109**	**0.6**	**0.42**
6	**9**	**8**	**79**	**102**	**0.75**	**0.39**
7	**9**	**7**	**81**	**116**	**0.68**	**0.4**
8	**8**	**8**	**95**	**132**	**0.74**	**0.53**
9	**10**	**7**	**86**	**128**	**0.69**	**0.48**

After transformation of the data, we get Table 2:.

The partitions induced by all singleton subsets of A are as follows:

U/IND (Fog) = ({p_1, p_5, p_8}, {p_2, p_4, p_6, p_7 }, {p_3, p_9})

U/IND (CR) = ({p_1, p_2, p_4,p_7, p_9}, {p_3, p_5, p_6, p_8})

U/IND (Components) = ({p_1, p_2, p_3,p_4, p_9}, {p_5, p_6, p_7},{p_8})

U/IND(CFS)= ({p_1, p_3, p_4,p_6, p_7}, {p_2, p_5}, {p_8, p_9})

U/IND(Complexity) = ({p_1, p_2, p_3,p_5}, {p_4, p_7, p_9}, {p_6, p_8})

U/IND(DMSC) = ({p_1, p_2, p_3}, {p_4, p_5, p_6, p_7}, { p_8,p_9}

From the definition of rough entropy, we can obtain that

$$E(Fog) = -\left[\frac{3}{4}\log\left(\frac{1}{3}\right) + \frac{4}{9}\log\left(\frac{1}{4}\right) + \frac{2}{9}\log\left(\frac{1}{2}\right)\right].$$

=0.4935

E(CR)=0.655, E(Components)=0.547, E(CFS)=0.522, E(Complexity)=0.493, E(DMSC)=0.4

We can remove one object p from all the objects, and then calculate entropy values. After that calculate Relative entropy values of different objects and find weight of the attributes of the objects.

After that calculate REOF of different objects

Data After Transformation Table 2

p	Fog Index	CR	Components	CFC	Complexity	DMSC
p_1	0	0	0	0	0	0
p_2	1	0	0	1	0	0
p_3	2	1	0	0	0	0
p_4	1	0	0	0	1	1
p_5	0	1	1	1	0	1
p_6	1	1	1	0	2	1
p_7	1	0	1	0	1	1
p_8	0	1	2	2	2	2
p_9	2	0	0	2	1	2

$$REOF(p_1) = \frac{0.9148 \times \frac{6}{9} + 0.919 \times \frac{4}{9} + 0.8773 \times \frac{4}{9} + 0.8649 \times \frac{4}{9} + 0.8782 \times \frac{5}{9} + 0.9158 \times \frac{6}{9}}{6}$$

= 0.4815.

REOF (p_2) = 0.524,

REOF (p_3) = 0.5251,

REOF (p_4) = 0.4654,

REOF (p_5) = 0.5822,

REOF (p_6) = 0.5437,

REOF (p_7) = 0.500,

REOF (p_8) = 0.7346,

REOF (p_9) = 0.6321.

From the above REOF values, it may be seen that the object(project)p_8 is having abnormal value, i.e it is an outlier. Here the outlier is the one which is may be less understandable or highly understandable. By considering different attribute values of projects, we can say technically it is difficult to understand. Here, we have proved by using rough set approach.

5-CONCLUSION

Software understandability affects quality of overall software engineering. If software understandability is favorable, software development process can be mastered definitely. In this work, we considered so many different types of metrics. But, we want to focus few more metrics in our further research. Here in chapter 4, we used a rough set approach to detect the project which is having abnormal behavior. This type of behavior tells us that the particular project is either easily understandable or very much difficult to understand. The algorithm which is used by us is having less time complexity than fuzzy based approach. In our further work we want to include threshold values which have been calculated based on the standard values of different attributes, based on that threshold value we will give outlier ranking.

6--REFERENCES

[1] Rough set. Website. http://en.wikipedia.org/wiki/Rough set.
[2] Rsm metrics. Website. http://msquaredtechnologies.com /m2rsm/docs/rsm metrics.
[3] Understandability metrics. Website. http: www. Aivosto .com/project/help.
[4] Using uml part two- behavioral modelling diagrams. Website. http://www.sparxsystems.com.
[5] Krishan K. Aggarwal, Yogesh Singh, and Jitender Kumar Chhabra. An integrated measure of software maintainability. pages 235-240, GGS Indraprastha University, Delhi and Regional Engineering College, Kurukshetra, 2002. 2002 PROCEEDINGS Annual RELIABILITY and MAINTAINABILITY Symposium.
[6] Richard H. Carver, Steve Counsell, and Reuben V. Nithi. An evaluation of the mood set of object-oriented software metrics. IEEE Trans. Software Eng., 24(6):491-496, 1998.
[7] Jitender Kumar Chhabra, K. K. Aggarwal, and Yogesh Singh. Code and data spatial complexity: two important software understandability measures. Information & Software Technology, 45(8):539-546, 2003.
[8] Jitender Kumar Chhabra, K. K. Aggarwal, and Yogesh Singh. Measurement of object-oriented software spatial complexity. Information & Software Technology, 46(10):689-699, 2004.
[9] Jitender Kumar Chhabra and Varun Gupta. Evaluation of object-oriented spatial complexity measures. ACM SIGSOFT Software Engineering Notes, 34(3):1-5, 2009.
[10] Nicolas E. Gold, Andrew Mohan, and Paul J. Layzell. Spatial complexity metrics: An investigation of utility. IEEE Trans. Software Eng., 31(3):203-212, 2005.
[11] Maurice H. Halstead. Elements of Software Science. ISBN:0-444-00205-7. Amsterdam, 1977.
[12] Seyyed Mohsen Jamali. Object oriented metrics. Department of Computer Engineering Sharif University of Technology, 2006.
[13] Feng Jiang, Yuefei Sui, and Cungen Cao. A rough set approach to outlier detection. volume 37, pages 519-536. International Journal of General Systems, october 2008.
[14] K.Shima, Y.Takemura, and K.Matsumoto. An approach to experimental evaluation of software understandability. Proceedings of the 2002 International Symposium on Empirical Software Engineering(ISESE'02), 2002.
[15] Xiangjun LI and Fen RAO. An rough entropy based approach to outlier detection. Journal of Computational Information Systems, pages 10501-10508, 2012. Department of Computer science and Technology, Nanchang University,Nanchang 330031, China and College of Economy and Management, Nanchang University, Nanchang 330031, China.
[16] Jin-Cherng Lin and Kuo-Chiang Wu. A model for measuring software understandability. In CIT, page 192, 2006.
[17] Jin-Cherng Lin and Kuo-Chiang Wu. Evaluation of software understandability based on fuzzy matrix. In FUZZ-IEEE, pages 887-892, 2008.
[18] Rajib Mall. Fundamentals of Software Engineering. Prentice Hall, 3rd edition, 2009.
[19] Sanjay Misra and A. K. Misra. Evaluating cognitive complexity measure with weyuker properties. In IEEE ICCI, pages 103-108, 2004.
[20] Yuto Nakamura, Kazunori Sakamoto, Kiyohisa Inoue, Hironori Washizaki, and Yoshi-aki Fukazawa. Evaluation of understandability of uml class diagrams by using word similarity. In IWSM/Mensura, pages 178-187, 2011.
[21] R.Horrison, S.Counsell, and R.Nithi. An overview of object-oriented design metrics. Dept. of Electronics and Computer Science, Southampton, 1997. IEEE.
[22] S. W. A. Rizvi and R. A. Khan. Maintainability estimation model for object-oriented software in design phase (memood). CoRR, abs/1004.4447, 2010.
[23] Patricia L. Roden, Shamsnaz Virani, Letha H. Etzkorn, and Sherri L. Messimer. An empirical study of the relationship of stability metrics and the qmood quality models over software developed using highly iterative or agile software processes. In SCAM, pages 171-179, 2007.
[24] Yingxu Wang and Jingqiu Shao. Measurement of the cognitive functional complexity of software. In IEEE ICCI, pages 67-74, 2003.

[25] Tong Yi, Fangjun Wu, and Chengzhi Gan. A comparison of metrics for uml class diagrams. ACM SIGSOFT Software Engineering Notes, 29(5):1-6, 2004.
[26] Aida Atef Zakaria and Dr. Hoda Hosny. Metrics for aspect-oriented software design. pages 228-233. ACMpress,1975