Volume 5, Issue 1, January 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Project Spatial Complexity: A New Measure for Understandability of Class Inheritance

Sheikh Kashif Ahmed
Student of MIT,
RGPV University, Bhopal, India

Sriram Yadav
Asst. Professor & Head (CSE) MIT,
RGPV University, Bhopal, India

Abstract--With the advancement in hardware and memory devices, computer software's are increasing in size and becoming more complex. This increase in complexity adversely effects software understandability and maintainability. As understandability is a human internal process, cognitive abilities are required to understand the source code. Spatial abilities are also required to correlate the orientation and location of various entities with their processing. In an object oriented software, where it is important to understand the use of attributes and methods, spatial complexity plays a very important role.

Inheritance is a key feature of object oriented software. It helps in reducing the source code but increases software complexity. As todays software are all object oriented and are using inheritance, it is required to evaluate spatial complexity of object oriented software comprising inheritance. Here a metric named Project Spatial Complexity (PSC) is introduced that helps in understandability of source code and shall help in development of maintainable software.

Keywords--Cognitive ability, Maintainability, Metrics, Spatial complexity, Understandability

I. INTRODUCTION

Software reuse is promoted by object oriented technology or component-ware technology [1]. Object Oriented Programming (OOP) languages like C++, Java, C#,Python, Visual Basic etc. supports the concept of reusability. It helps in reducing the efforts and time required to develop the software. However when developers try to reuse a software system developed by other developers, the difficulty of understanding the system limits reuse [2]. For enhancing functionality, fault correction, adapting to new environment or increasing performance, it is required to modify the existing software. The developer needs to understand the source code before making any change. The understandability of the source code depends upon the psychological complexity of the software, and it requires cognitive abilities to understand the source code [3]. If the changes that need to be done are not understood correctly, then it will cause errors in software. Such changes takes time and more efforts are required which adds to the cost for the company. Furthermore in order to perform maintenance for software, it is necessary to understand the software domain. Maintenance requires reading the source code and other software artifacts which are generated by someone else. The more understandable the source code is, the more quickly and accurately a programmer can obtain critical information about a program by reading the source code.

Boehm defined software understandability as a characteristic of software quality which means ease of understanding software system [4]. In his model, understandability is placed as a factor of software maintenance. For developing quality software it is highly desirable that the software developer follows standard software engineering practices. Understandability of software also effects software testing. Studies shows that maintainers spend half of their time in understanding what others are doing and what needs to be done. From the time they actually spend on maintenance, 60% of the time is spent in searching the source code where changes need to be done.

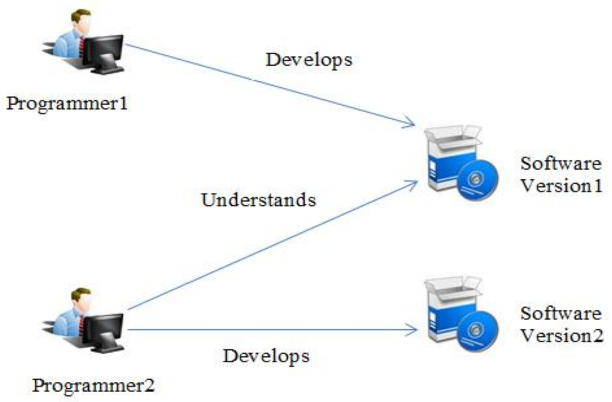

Figure 1: Programmers developing different versions

Referring to Figure 1, let say programmer1 has developed software version1. After some months or years programmer1 left the organization and programmer2 was assigned the task to maintain that software. Programmer2 needs to understand the source code written by programmer1 for generating the new version i.e. version2. Now if programmer1 has followed standard software engineering practice then it will be easy for programmer2 to understand the source code and the required maintenance will be done easily. The difficulty in understanding the source code will lead to delay in delivery of software. The rate of hardware obsolescence, the immortality of a software product, and the demand of the user community to see the existing software products run on newer platforms, run in newer environments, and/or with enhanced features has made maintenance the most significant phase of the software life cycle.

II. RELATED WORK

Several studies have been conducted for understandability of software.Uchida et. al. propose a method called 'software overhaul' and based on this method propose a probability model for software understandability evaluation [5]. Sheldon et. al. proposes two metrics for maintenance of object oriented software comprising of inheritance hierarchies [6]. Chhabra et. al. proposes two software understandability measures which were based on data and code [7]. Malik et. al. provides a technique which helps in determining how well the requirements are understood by development team by reusing the inputs of software cost estimation process [8]. Shivprakasamet. al. presents measures for spatial complexity of object oriented software which were based on definition and usage of classes [9]. Gupta et. al. proposes cognitive-spatial complexity measures for computation of software complexity. For software complexity measurement both spatial and architectural aspect of software wereconsidered[10]. Chhabra proposes a metric named Code Cognitive Complexity(CCC) for measuring complexity by combining spatial distances, impact of control statement, and the effect of input and output parameters [11]. Rajnish proposes a metric named class complexity metric(CCM) for predicting the understandability of classes [12].

In object oriented programming, class design is very important as it deals with functional requirements of the system. Object oriented approach has a key feature called inheritance. Use of inheritance claims to reduce amount of software maintenance necessary and ease the burden of testing [13]. Most of the research done in understandability of object oriented software considered spatial and cognitive complexity of objects and classes. Not much work is done in evaluating understandability of a program comprising inheritance. As inheritance is a key concept and is used by programmers to get the benefit of reusability, a new measure is needed which evaluates the understandability of program involving inheritance.

Furthermore previous researches were focused on evaluating understandability of individual classes only. A measure is needed which evaluates the understandability of complete project

III. ATTRIBUTE SPATIAL COMPLEXITY

In object oriented programming attributes and methods of a class are usually accessed by objects of that class. While generating attribute spatial complexity, if a method is declared and defined inside a class itself, usage of attributes inside those methods is not considered as difference in line of codes between the attribute declaration and usage will be less and its complexity can be ignored. Similarly usage of attributes inside a method defined outside the class is also ignored as methods declared inside the class are usually defined immediately after the class definition. So only attributes which are accessed using object are considered.

Attribute spatial complexity (CASC) is given as

$$ASC=\sum_{i=1}^{p} Distancei/p$$

where p represent count of attribute and Distancei is difference in LOC between current use of attribute through an object from its just previous definition/use.

As there are many attributes in class so class attribute spatial complexity(CASC) will be

$$CASC= \sum_{i=1}^{q} ASCi/q$$

where q is the count of attributes in a class.

IV. METHOD SPATIAL COMPLEXITY

In object oriented programming attributes and methods of a class are usually accessed by objects of that class. It is a standard practice to usually declare methods inside the class and define it outside the class, if the methods are large in size and are containing control statements. While generating method spatial complexity, if a method is declared and defined inside or outside a class, its complexity can be ignored as difference in line of codes between the declaration and definition is less and hence can be ignored.

Method spatial complexity (MSC) is given as

$$MSC=\sum^{p} Distancei/p$$

$i=1$

where p represent count of method and Distancei is difference in LOC between current use of method through an object from its just previous definition/use.
As there are many methods in a class, so class method spatial complexity (CMSC) will be

$$CMSC= \sum_{i=1}^{q} CASCi/q$$

where q is the count of methods in a class

V. CLASS SPATIAL COMPLEXITY

Class spatial complexity is sum of class attribute spatial complexity and class method spatial complexity.

$$CSC=CASC+CMSC$$

where CSC is class spatial complexity, CASC is class attribute spatial complexity and CMSC is the class method spatial complexity.

1. Derived Class Attribute Spatial Complexity

Derived class attribute spatial complexity is generated by summing its own attribute spatial complexity and attribute spatial complexity of all its base classes i.e.

$$DCASC= \sum_{i=1}^{n} CASCi/n + \sum_{i=1}^{m} CIASCi$$

Where DCASC is the derived class attribute spatial complexity, 'n' is the number of attributes of its own class, 'm' is the number of base class.CASC is the Class Attribute Spatial Complexity of the class under consideration and CIASC is the Class Attribute Spatial Complexity of all inherited base class/classes

2. Derived Class Method Spatial Complexity

Derived class method spatial complexity is generated by summing its own method spatial complexity and method spatial complexity of all its base classes i.e.

$$DCMSC= \sum_{i=1}^{n} CMSCi/n + \sum_{i=1}^{m} CIMSCi$$

Where DCMSC is the derived class method spatial complexity, 'n' is the number of methods of its own class, 'm' is the number of base class. CMSC is the Class Method Spatial Complexity of the class under consideration and CIMSC is the Class Method Spatial Complexity of all inherited base class/classes.

3. Derived Class Spatial Complexity

Derived class spatial complexity is the sum derived class attribute spatial complexity and derived class method spatial complexity.

$$DCSC=DCASC+DCMSC$$

where DCSC is derived class spatial complexity, DCASC is derived class attribute spatial complexity and DCMSC is the derived class method spatial complexity.

VI. PROJECT SPATIAL COMPLEXITY

Previous research for calculation of spatial complexity focuses on individual classes. Spatial complexity of a class is calculated as the sum of spatial complexity of attributes and methods. Object oriented software has collection of classes with each class designed and used for specific purpose. Each class in object oriented software has its own spatial complexity which varies with respect to the number of attribute and methods declared and used.Object oriented software supports many other concepts apart from the concept of class. For example inheritance, polymorphism, constructors, etc. are all used in object oriented software. Previous research concentrates on class as it is the base of any object oriented programming language. Spatial complexity of class is calculated with no focus on other object oriented concepts.Inheritance affects the spatial complexity of a class as attribute and method are inherited from other class/classes. Inheritance leads to increase in spatial complexity as more attribute and method are added in a class. So it is necessary to measure effect of inheritance in calculating the spatial complexity of a class and for the complete project.

For the definition of metrics for spatial complexity of complete project, terms from graph theory is used. In the directed graph, such as Figure 2, whose vertices represent the classes and edges represent the preceding relationship (inheritance), vertex i is a predecessor of vertex j under the following conditions. If there exists a path from vertex i to vertex j, and vertex j is a successor of vertex i**[14],** we define function PRED and SUCC as follows:

• PRED(i): the total number of predecessors of node i,
• SUCC(i): the total number of successors of node i,

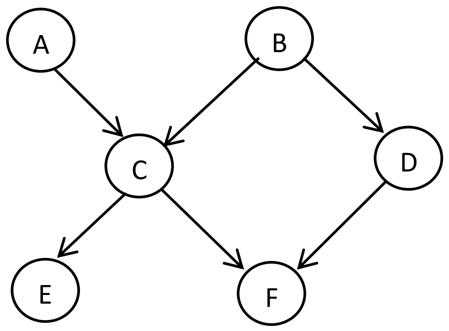

Figure 2: A Directed Acyclic Graph depicting Class Inheritance

Further graph theory had terminology such as degree, sink vertex, source vertex which will be used here to calculate the spatial complexity of project. With respect to our concept these terminologies can be explained as follows:

•DEGR(i): total number of edges connected to node i

•SOURCE_VERTEX: node i which does not have any incoming edge.

•SINK_VERTEX: node i which does not have any outgoing edge.

For example in Figure 1,

DEGR (A) = DEGR (B)=2, DEGR(E)=1

SOURCE_VERTEX: A, B

SINK_VERTEX: E, F

In object oriented programming some classes act as abstract class while some classes are used for processing by creating there object. These processing classes are usually SINK_VERTEX whose object are created. So, spatial complexity of project can be calculated by calculating the summation of spatial complexity of all those classes which does not have direct relationship (inheritance) with each other that is SINK_VERTEX.

The project spatial complexity comprising of multiple classes with many different types of inheritance can be defined as follows:

$$TSCP = \sum_{i=1}^{n} CSC_{SINK\_VERTEXi}$$

where n is the total number of SINK_VERTEX in inheritance hierarchy and CSCSINK_VERTEXi is the class spatial complexity of SINK_VERTEXi.

VII. RESULT AND DISCUSSION

For evaluation of proposed metric, a tool was developed in java. Five projects of different size and functionalities were selected and there attribute spatial complexity method spatial complexity and project spatial complexity was calculated using tool. The computed values are given in table 1

Table1: Spatial Complexity of Project

P.NO.	LOC	ASC	MSC	PSC
1	33	14	12	26
2	37	37	17	54
3	38	31	43	74
4	44	51	24	75
5	52	86	104	190

1. The value of PSC gives a hint about the efforts needed to understand the source code. Higher value of PSC means more efforts are needed to understand source code. In order to verify our intuition, perfective maintenance is applied on all projects. Ten diploma students (all of almost same knowledge and skills) were selected and five teams were formed, each team comprising of two programmers. Each team is allocated one project and the target to improve time efficiency by 5% was decided. Each team was asked to identify those components in project where improvement is possible. This activity requires thorough understanding of working of source code. The result is show in table 2

Table 2: Maintenance Time and PSC of Project

P.NO.	Line Of Code (LOC)	Maintenance Time(Person-Hrs)	PSC
1	33	0.25	26

2	37	0.47	54
3	38	1.22	74
4	44	1.36	75
5	52	1.50	190

From table 2 it can be observed that project5 has the highest PSC and maintenance time required is also highest among five project. The least value for PSC is for project1 and as per our intuition less maintenance time is required for project1.

2. The value for PSC gives a hit of size of project. As the size of project increases/decreases, PSC also increases/decreases. It is difficult to understand long project as more lines of code increases distance between the call and definition of attributes and methods and hence increases PSC. From figure 3, it can be observed that project5 with 52 lines of code has the highest PSC of 190 and the project with 33 lines of code has PSC value of 26, which proves our intuition.

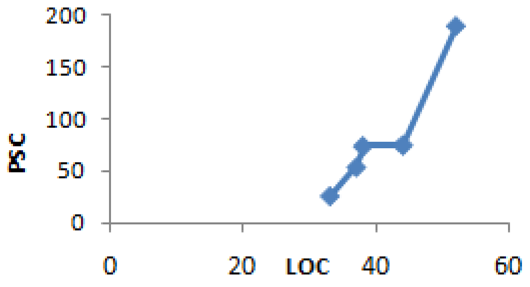

Figure 2: Project Spatial Complexity Versus LOC

From the graph it is clear that PSC increases with increases with increases in LOC.

3. There is a relation between PSC and number of classes in the project. With the increase in number of classes PSC increases accordingly. From the graph given in figure 4, it can be observed that project with only one class has PSC value of 26 whereas project with total of three classes has PSC of 75 and 190 respectively. For project with three classes PSC values have a huge difference because PSC depends on total of attributes, methods and their spatial distance along with total classes in a project.

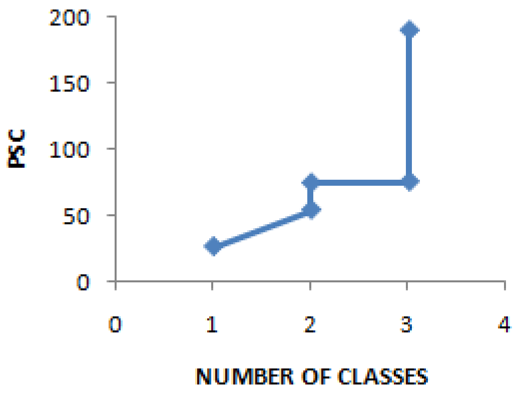

Figure 4:Project Spatial Complexity Versus LOC

4. There is a strong correlation between class member usage and class spatial complexity (CASC). As the class member are declared in class and are used in main function by creating object, increase in class number usage, increases class spatial complexity. Some programmers declare many attributes in project but do not use it. While calculating class spatial complexity only those attributes and methods are considered in complexity calculation which are declared in class and are used in main function. Attributes and methods which are declared in class but are not used in main function, are ignored while calculating complexity.

5. Usage of inheritance in project is directly proportional to project spatial complexity (PSC). Deeper the inheritance hierarchy more is the value for PSC. More the number of classes inherited by a class, more is the value of class spatial complexity for that class.

VIII. CONCLUSION

Spatial complexity measures for understandability of object-oriented software comprising of inheritance hierarchies can be calculated by calculating the summation of spatial complexity of all SINK_VERTEX i.e. the summation of all the classes in inheritance hierarchies which are not the base class for any other class.

Summation of spatial complexity of all SINK_VERTEX is here referred as project spatial complexity (PSC). The project with large value of PSC will be more difficult to understand. Thus, PSC can be a very useful indicator of understandability of project. PSC can be helpful to measure the understandability of interaction among classes.

Lower values of PSC denote better understandability of object oriented project.
Certain guidelines can be derived about the acceptable range of PSC .This range can be used by software managers to judge the understandability of the software. Concept of PSC can play a very important role in developing testable and maintainable software, which is desirable in the software industry.

IX. FUTURE WORK

The future work of this measure requires a detailed evaluation over large programs to find its suitability. As measure is applied on typically small program there relevance for the big project need to be tested.
Spatial complexity for the program with inheritance hierarchies is considered. Effect of templates, preprocessor directive, this pointer, macros, structure, etc. on spatial complexity is another direction to work upon.

REFERENCES

[1] M. Aoyama, “Component-based software engineering: can it change the way of software development?” Proc. of the 20th International Conferenceon Software Engineering, vol. 2, pp. 24-27, 1998

[2] G. Caldiera, and V. R. Basili, “The qualification of reusablesoftware components,” pp. 117-119

[3] A. Baddeley, Human Memory: Theory and Practice, Revised Edition, Hove Psychology Press, 1997

[4] B. W. Boehm, et. al, “Characteristics of Software Quality,” North-Holland, 1978.

[5] S. Uchida, K. Shima, An Experiment of Evaluating Software Understandability using a Probabilistic Model.

[6] F. Sheldon, K. Jerath, H. Chung, “Metrics for maintainability of class inheritance hierarchies”, Journal of Software Maintenance and Evolution: Research and Practice, Feb 7, 2002.

[7] J. K. Chhabra, K. K. Aggarwal, Y. Singh, “Code and Data Spatial Complexity: two important software understandability measure”, Information and Software Technology, pp. 539-546, 2003.

[8] A. A. Malik, B. W. Boehm, A. W. Brown, “Predicting Understandability of a Software Project Using COCOMOII Drivers”

[9] P.Sivaprakasam, V. Sangeetha, C. P. Rangarajan, “Measurement of Object-Oriented Software Spatial Complexity” Journal of Computer Applications, Vol. 1, Issue 3, July-Sept 2008.

[10] V. Gupta, J. K. Chhabra, “Object- Oriented Cognitive –Spatial Complexity Measures”, World Academy of science, Engineering and Technology, Vol. 3, 2009.

[11] J. K. Chhabra, “Code Cognitive Complexity: A New Measure”, Proceeding of the World Congresson Engineering.

[12] K. Rajnish, “Class Complexity Metric to predict Understandability”, I. J. Information Enginerring and Electronics Business, pp. 69-76, 2014.

[13] Chidamber SR, Kemerer CF. A metrics suite for object-oriented design. IEEE Trans. on Software Engineering 1994;20(6):476–493.

[14] Horowitz E, Sahni S. Fundamentals of Data Structures in C. Computer Science Press: New York, 1993; 303–304.