A Heuristics on Interface Complexity

K. Maheswaran[1], A. Aloysius[2]
Research scholar[1], Assistant Professor[2]
Department of Computer Science, St. Joseph's College (Autonomous), Tiruchirappalli, Tamil Nadu, India. 620002
mahes161@gmail.com[1], aloysius1972@gmail.com[2]

Abstract—: **Measurement is an essential part of software engineering. Classes and Interfaces are the basic concepts of object oriented paradigm. In Software engineering interface has been used for more than twenty-five years but there is no proper study the evaluate the effect of an interface while implementing into the class, so far only little information's are available how interfaces are actually complex. This paper briefly discussed the cognitive complexity of class with interface and without interface and proposed the heuristics. The empirical validation has been done and their results were discussed.**

Index Terms— **Keyword: class, interface, object oriented, complexity.**

I. Introduction

Software engineering metrics are important measurements for project planning and project measurements. The increasing importance of software measurement and metrics led to the development of new software measures and metrics. Many researches have progressed in software engineering during the last three decades. Object oriented (OO) software development has received higher acceptability among the programming community since it is built on the notion of real world entities. The concept of an interface in object oriented programming is quite old. Software engineering has been using interfaces for more than twenty-five years. Software measurement activities were not addressed to most of their requirements for providing information and to support for managerial decision making [1]. Several metrics are available to measure the complexity of class, method, cohesion, coupling, inheritance, and polymorphism.

Abstraction is one of the major pillars of OO programming. It is achieved by using interface and abstract class in Java. In order to use interface, we need to extend and implement an abstract method with concrete behavior. One example of Abstraction is creating interface to denote common method without specifying any details about how that behavior works. The idea in this paper is to find the cognitive complexity between class with interface and without interface. Interfaces allow only method definitions and constant attributes. Methods defined in the interfaces cannot have behavior of the method. Classes can implement these interfaces by providing bodies for the methods defined in the interface. Any intended change on the methods defined in the interface will impact the classes. The following are possible changes.

1) Changing the name of a method
2) Changing the signature of a method and
3) Changing the return type of a method.

If any new method is included to an interface, this will also impact the classes that currently use or implement the interface. Heuristics plays a significant role in software development and is widely used to provide a link between design principles and software measurement [7]. It offers insightful information based upon experience that is known to work in practice. Heuristics are not meant to be exact; in fact, they derive their benefits from this imprecision by providing an informal guide to good and bad practices. They provide a means by which knowledge and experience can be delivered from the expert to the novice [6]. The aim of this paper is to identify the complexity to understand class with interface (CWI) and class without interface (CWOI) program. The entire paper has been organized into five major sections. The following section contains literature review. The empirical study of an interface was presented in section III. The experimental results and discussions were described in section IV. Finally, we make the conclusion in section V.

II. Review of Literature

Abstraction in java is achieved by using the concept of interface and abstract class. An interface or abstract class is not concrete, something which is incomplete. If we need to use interface or abstract class, then extend and implement an abstract method with concrete behavior. Java does not support multiple inheritance, we can achieve this by using interfaces, as a class can implement more than one interfaces. The concept of interfaces has been measured in java programming by Fried Stiemann and Co [2]. He represented that the usage of interfaces compared to classes are 4:1 and the added effort of interfaces, created complexity to the programmer for their maintenance. Philip mayer created a metrics suite namely Number of Different Access Sets (NODAS), Actual Context Distance (ACD), Best Context Distance (BCD), Interface Minimization Indicator (IMI) and tool for interfaces to support the developer [3]. Krishnapriya compared the interface along with inheritance [4]. It is evaluated with the help of a set of seven known structural complexity measures and compared for their usage in object oriented programming. The results show that the interface concepts are better performance compared to inheritance concept, so that software reliability will increase. The author not concentrates the complexity of an interface; only compare the performance of interface with inheritance.

.

Rakesh et al. discussed variety of OO metrics and suggested some heuristics based on them to help designers with the task of understanding, evaluating and improving their products [8]. M Yadav et al. compared inheritance and interface with examples and concluded that cohesion will increase in interface so interface is more reusable as compare to inheritance [9]. In literatures, relatively very little information has been published about an interface and its measure in object oriented programming (OOP). There is a need to study and analyze the complexity of an interface by including cognitive perspective. No researchers were analyzed the cognitive complexity of an interface. This research work briefly identified whether the cognitive complexity of an interface is significantly increased when interface is implemented into the class.

III. EMPIRICAL STUDY OF INTERFACE

An interface in java has static constants and abstract methods. The interface is a mechanism used to achieve abstraction. It contains only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritances in java. The novel idea of this paper is to measure the cognitive complexity of class with/without interface. Classes can implement the interface by providing bodies for the methods defined in the interface. An interface is a contract between a client class and a server class [5].

This paper aims to measure the complexity of a class by comparing average comprehension time (ACT) due to the implementation of interface. The group 80 out of 110 students who scored 65% and above is selected for an experiment. These students had sufficient exposure in analyzing the OO programs; they are participated our comprehension test. Seven different programs were taken for the test and it were named has Program I, II, III, IV, V, VI, VII which is listed in following Table 1.

Prog. No.	**Title of the program**
I	To find sum of two numbers
II	To check prime or Armstrong number
III	To read and Merge two arrays
IV	To read, calculate and display Student details
V	To find simple Arithmetic calculations.
VI	To Read the one-dimensional array with ten elements and calculate biggest, smallest of ten no's, ascending, second big, search element in the array.
VII	To read two matrix A and B and calculate addition, subtraction, multiplication of two matrices then find the transpose of matrix A and row sum of matrix B.

Table 1: Program Description

All the seven programs were written in two aspects namely class without interface (CWOI) and class with interface (CWI). It named as follows *Program I class* means the first program written using only class without interface and *Program I_inter* means the first program written using class with interface. There are totally fourteen programs.

The programs were classified into two sets. Each set contains seven programs. It has given to each student's for testing (empirical study) and expected to write the outputs after understanding the program. The two sets of programs were classified as shown in the Table 2.

Set No.	**Class with Interface (Program No.)**	**Class without Interface (Program No.)**
Set-1	I,II,III, VII	IV, V, VI
Set-2	IV, V, VI	I,II,III, VII

Table 2: Program Classification

The UML representation of the programs I, III, V out of seven programs has shown in Fig. 1, 2, 3 and 4 respectively as a sample. By the concept of an interface, A class is implemented an interface the class must use / define all the methods declare in the interface. Otherwise, we will get compiler error. Many times, some of the methods declare in the interface not used by the class.

In Table 3, Number of methods in class with interface 2 (1) indicates the class contains two methods out of which one method is interface method. i.e., one method is declared in the interface which is implemented in the class.

Program No.	**Number of Methods**	
	Class with Interface	**Class without Interface**
I	2 (1)	2
II	3 (2)	3
III	5 (3) *	4
IV	6 (4) *	5
V	7 (5) *	6
VI	8 (6) *	7
VII	9 (7) *	8

Table 3: Method Classification in Programs

Heuristic 1: *From the view of a programmer, defining all the methods of an interface with in a class (even though some of the methods are not used) increases the complexity of a design or program.*

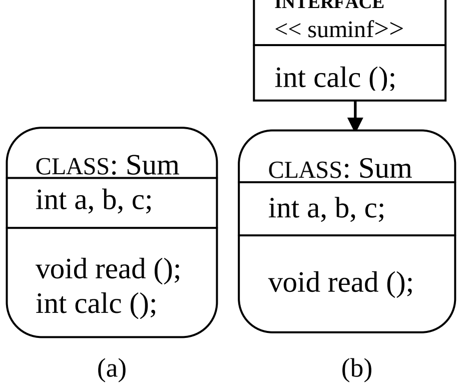

Fig. 1 UML Representation for Program I
(a) Class With Out Interface (Program I_class)
(b) Class With Interface (Program I_inter)

In Fig. 2(b), 3 (b), 4(b) - *denotes the method declared in the interface not used (defined) in the class. These methods must be defined, so that it is simply defined as an empty method in the class.

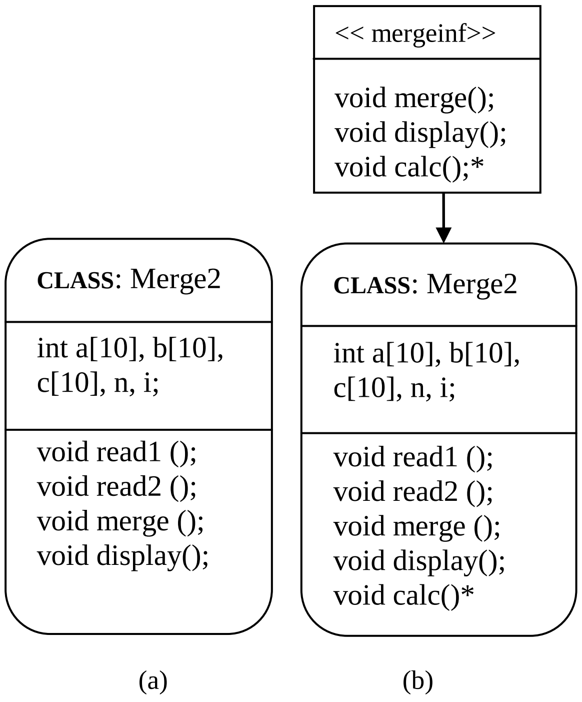

Fig. 2 UML Representation for Program III
(a) Class With Out Interface (Program III_class)
(b) Class With Interface (Program III_inter)

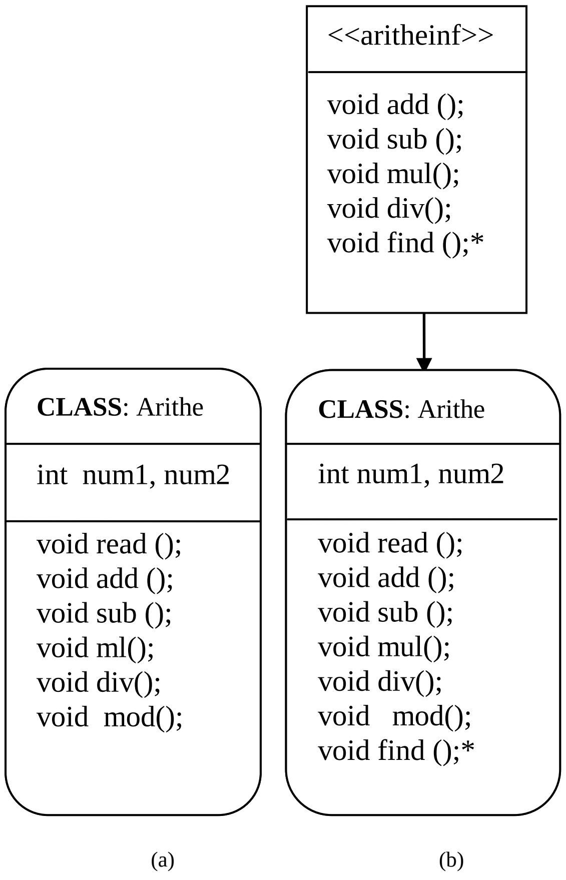

Fig. 3 UML Representation for Program V
(a) Class With Out Interface (Program V_class)
(b) Class With Interface (Program V_inter)

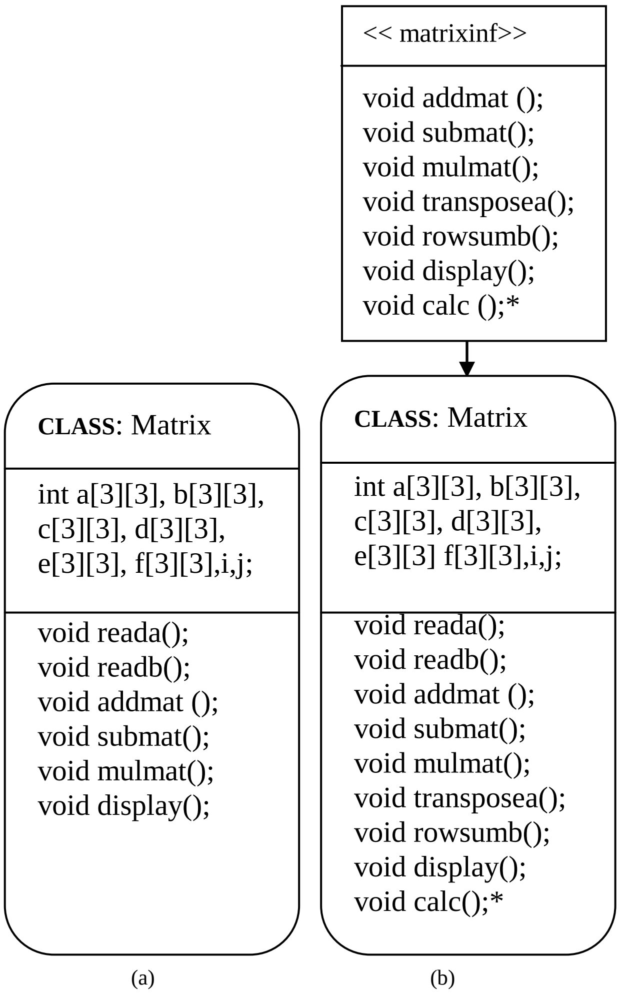

Fig. 4 UML Representation for Program VII
(a) Class With Out Interface (Program VII_class)
(b) Class With Interface (Program VII_inter)

Heuristic 2: *If the number of methods and its parameters increases in the interface then the complexity of a class is increased when implementing an interface.*

IV. EXPERIMENTAL RESULTS AND DISCUSSION

The average comprehension time of each pair of programs (Refer Table 1) both CWOI and CWI is calculated and displayed in Table 4.

Prog. No.	**Average Comprehensive Time**	
	Class With Out Interface (CWOI)	**Class With Interface (CWI)**
I	105.60	115.80
II	276.60	307.20
III	391.20	435.60
IV	444.00	514.20
V	590.40	679.80
VI	643.20	764.40
VII	882.00	1035.60

Table 4: Experimental results of CWOI Vs CWI

Table 4 shows that there is a complexity difference between the understandability of the programs of CWOI and CWI since the average comprehension time of CWI are greater than CWOI. In program I, the ACT difference between the CWI and CWOI is almost same because the interface has minimum number of method. If the number of methods in an interface is increases then the ACT to understand the class is also increase. From the second programs, onwards the ACT of CWI and CWOI was increased significantly since the numbers of methods in an interface increases. Especially, the number of methods in an interface is more than seven then time difference to understands CWOI and CWI is increases immensely. Thus there is complexity between CWI and CWOI. Hence there is a need to measure the complexity of an interface.

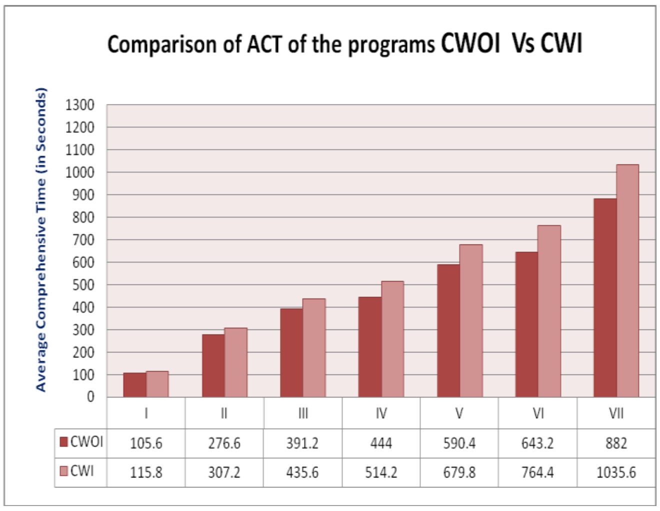

Fig. 4 ACT Comparison of CWOI Vs CWI

Heuristic 3: *The complexity to understand the programs of CWI and CWOI are differs significantly since the ACT of CWI is more than CWOI. Hence, if an interface is implemented in a class then somehow the complexity of a class is increased.*

Heuristic 4: *From Heuristic 3, it is understood that the complexity of a class is affected or increased when an interface is implemented. Hence there is a need to measure interface complexity of a class.*

V. CONCLUSION

In software development, the concepts interfaces are heavily used in almost all the object oriented programming languages. This paper elaborately addresses the complexity difference between the class with and without interface. The empirical study was conducted and the results were displayed and also four heuristics were discussed. In software industry the logic of interfaces are advisable to be used in large type of applications [10]. The class complexity affects due to an implementation of an interface. In future, there is a need to propose for complexity metrics to measure the interface.

REFERENCES

[1] Norman E Fenton, Shari Lawrence Pfleeger, “Software Metrics – A Rigorous & Practical Approach”, Third Edition, CRC Press, November 2014.
[2] F Steimann, W Siberski, T Kuhne, “Towards the systematic use of interfaces in JAVA programming”, Proceedings of 2nd International Conference on the Principles and Practice of Programming in Java (PPP), pp. 13-17, ACM, 2003.
[3] Philip Mayer, “Analyzing the Use of Interfaces in Large OO Projects”, Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 382-383, 2003.
[4] Krishnapriya V and Ramar K, “Comparison of Class Inheritance and Interface Usage in Object Oriented Programming through Complexity Measures”, International Journal of Computer Science & Information Technology (IJCSIT), Vol. 2, No. 6, pp. 28-36, 2010.
[5] http://en.wikipedia.org/wiki/Object-oriented-programming.
[6] Deepali Gupta, Rakesh Kumar, “An Heuristic Approach To Object Oriented Paradigm“ International Journal of Computer Technology and Application (IJCTA), Vol. 2, No. 4, pp. 823-826, 2011.
[7] Churcher N, “Supporting OO Design Heuristics”, Proceedings of the 2007 Australian Software Engineering Conference, IEEE Computer Society, pp. 101-110, 2007.
[8] Rakesh Kumar, Deepali Gupta, “Heuristics Based on Object Oriented (OO) Metrics”, International Journal of Emerging Technology and Advanced Engineering Volume 2, Issue 5, pp. 393-395, 2012.
[9] Maya Yadav, Jasvinder Pal Singh, Pradeep Baniya, “Complexity Identification of Inheritance and Interface based on Cohesion and Coupling Metrics to Increase Reusability”, International Journal of Computer Applications Vol. 64, No. 8, 2013.
[10] Varsha Mishra, Shweta Yadav, “Better Object Oriented Paradigm Inheritance and Interface through Cohesion Metrics”, International Journal of Computer Applications, Vol. 66, No. 21, pp. 13-17, 2013

ABOUT THE AUTHORS

Mr. K. Maheswaran is working as Assistant Professor and Scholar in Department of Computer Science, St. Joseph’s College (Autonomous), Tiruchirappalli, Tamil Nadu, India. He has 7 years of experience in teaching and research. He has published research articles in the National / International conferences and journals. He was qualified with State Eligibility Test (SET) in 2012 and currently pursuing Doctor of philosophy in Computer science. His current area of research is Cognitive Aspects in Programming

Dr. A. Aloysius is working as Assistant Professor in Computer Science, St. Joseph's College (Autonomous), Tiruchirappalli, Tamil Nadu, India. He has 16 years of experience in teaching and research. He was qualified with State Eligibility Test (SET) in 2012. He has published many research articles in the National / International conferences and journals. He has also presented 2 research articles in the International Conferences on Computational Intelligence and Cognitive Informatics in Indonesia. He has acted as a chair person for many national and international conferences. His research interests are: Big Data, Cloud Computing, Software Measurement, Cognitive Aspects in Programming, Cloud Computing, Big Data and Predictive Analysis.