# METRICS TO QUANTIFY SOLID SOFTWARE DESIGN PRINCIPLES

Dr. Sunil Sikka
Associate Professor
Amity School of Engineering and Technology, Amity University Haryana, Gurugram, India

***Abstract:*** It has been validated by many studies that software design has major impact on the overall quality of software. Various software design guidelines are available to guide the user to produce best possible design. One of such guidelines is SOLID design principles. To measure the effectiveness of these principles quantitative assessment using metrics is required. Lot of software design metrics are available in literature. But metrics to assess specifically SOLID design principles are rare. This paper is aimed to identify the metrics that can be used for quantitative assessment of SOLID design principles.

***IndexTerms* – SOLID principles, Software Design, Software Metric**

## I. INTRODUCTION

Object-Oriented methodology aids to develop better quality and highly maintainable software. But merely using Object-oriented methodology for developing software can't be guarantee better quality and maintainability. Object-oriented methodology has ability to provide better quality and maintainable software subject to object-oriented design principles are properly obeyed. Many such principles are available in the literature. One of such set of principles is known by a name SOLID. SOLID is a set of five design principles which together are able to provide more understandable, flexible and maintainable software. It includes Single Responsibility Principle (SRP), Open/Closed Principle (OCP), Liskov Substitution Principle (LSP), Interface Segregation Principle (ISP) and Dependency Inversion Principle (DIP).The description of these principles and the theirs benefits are discusses discussed by various authors [1][2][3] are as follows.

SRP states that "a class should have only one reason to change". A Software may provide many functionalities which are encapsulated in various classes in object-oriented software. According to this principle each class should encapsulate only a single part of the functionality so that any single change should lead to change in only one class.

OCP states that "the software entity should be open for extension, but closed for modification". The objective is to introduce new functionalities without modifying the existing classes. Because changes in one class may introduce some other changes in dependent entities. One of the aims of good design is to minimize such changes. Object-oriented languages implement this principle using method overriding.

LSP state that "object in a program should be replaceable with instances of their sub types without altering the correctness of the program". Reference to the Base class should be replaceable to derived class object without altering the functionality of the program i.e. derived class must be substitutable to its base class.

ISP states that "Many client specific interfaces are better than one general purpose interface". The objective is that clients should not be forced to depend upon interfaces that they do not use. It increases the portability of the class. In case of multiple clients of a class, implementing requirements of all clients in a single class is not a good idea as per good design. The better option is to break general purpose interface into set of client specific interface and realize the interfaces by classes.

DIP states that "Depend upon Abstractions. Do not depend upon concretions." This could be achieved by if every dependency in the design should target an interface, or an abstract class not a concrete class. The rationale behind this principle is the fact that changes to interface are very less or rare as compare to the implementation. So idea is to increase the dependencies to the entities which are less change-prone.

This paper is aimed to identify the metrics that can be used to quantify the SOLID principles. Rest of the paper is divided into two sections. Section 2 provides the metrics identified from literature to quantify the SOLID principles. Finally section 3 presents the conclusion of the paper.

## II. METRICS TO QUANTIFY THE SOLID PRINCIPLES

The most commonly used Chidamber & Kemerer (CK) metrics are validated by various researchers as a fault proneness, maintenance efforts, changeability etc. predictor[4][5][6][7][8][9][10]. But few studies have been found in literature which validates CK or other metrics in context of SOLID principles. The literature review in context to metrics for SOLID principles is discussed in this session.

Singh and Hassan[1] empirically assess the SOLID principles using two design versions of the same problem. One version violating the SOLID design principles and other version following the SOLID design principles. Seven metrics which includes six CK metrics for Java programs (CKJM) and Number of Public Methods(NPM) are used to assess both versions of the design. Results shows that the design version which follows the SOLID principles provides better quality due to reduction of coupling and increase of cohesion and reusability.

In a presentation Jason Gorman[3] presented some simple design violating SOLID principles and then refactored the design following SOLD principle. Along with each principle one metric is presented to quantify the principle.

For SRP author defined Lack of Cohesion of Methods metric. Lack of Cohesion of Methods (LCOM) = $(((\sum R(A_i))/A) - M)/1-M$

Where M = total no. of methods in class, A= Total no. of attributes in the class, R(A,) = No. of methods use the attribute $A_i$ and $\sum R(A_i))/A$ = Average number of methods that access each attribute.

For OCP author defined a metric which is equal to the classes "extended and not modified / classes extended and/or modified". More number of classes extended without any modification supports this principle. Modification in classes may introduce new bugs which furthers increases the testing efforts. SRP is also related to this principle because less the responsibilities of the class lower is reasons to change it. For LSP author emphasis that it can be achieved if derived class follow all those rules that are applied on super type which includes pre-conditions for calling methods, post-conditions of methods called, and invariants that always apply between method calls. This principle is quantified by counting number of Unit test pass by a class which is designed for all of its super classes. To fully support this principle the class should pass all the tests. ISP for a type T is quantified by n/N. Where N is the total number of methods exposed by T and n is the total number of methods used a client of T. Further average of n/N for all clients of T can be computed. If all client use all the method exposed by the T then the average value of n/N comes to the 1 which is the best result. If some of the methods of the type are not used by any client then it indicates the type is not specific to the client. DIP is quantified by the computing "dependencies on abstractions / total dependencies". The ideal value of the metric is 1 which is possible only if all dependencies are only on abstractions, which is the objective of this principle.

## III. CONCLUSION

SOLID principles are the set of principles to ensure the better object-oriented designs and finally better quality software. However for a large design it becomes cumbersome to make sure that all principles are obeyed. Therefor quantification and automation of the principles is important. The literature survey yield very less metrics available to measure the SOLID principles. Also existing metrics are not validated for SOLID principles. So there is a scope of further research to define new metrics and to validate the existing/new metrics in context of SOLID principles.

## REFERENCES

[1] Harmeet Singh, Syed Imtiyaz Hassan "Effect of SOLID Design Principles on Quality of Software: An Empirical Assessment "International Journal of Scientific & Engineering Research, Volume 6, Issue 4, April-2015

[2] Robert C. Martin "Design Principles and Design Patterns", 2000.
http://www.cvc.uab.es/shared/teach/a21291/temes/object_oriented_design/materials_adicionals/principles_and_patterns.pdf

[3] Jason Gorman "Object oriented Design Principles", 2006.
http://pagesperso.lina.univ-nantes.fr/~molli-p/pmwiki/uploads/Main/oometrics.pdf

[4] S.R. Chidamber, C.F. Kemerer "A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering", June 1994 Vol.20, No.6, 1994.

[5] Victor R.Basili, Lionel Briand and Walcelio L. Melo." A Validation of Object-Oriented Design Metrics as Quality Indicators. Technical Report, Univ. of Maryland, Dept. 0f Computer Science, College Park, MD, 20742 USA, 1995.

[6] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, Francois Lustman, and Guy St-Denis, " Design Properties and Object-Oriented Software Changeability". In Proceedings of the 4th European Conference on Software Maintenance and Reengineering,2000, pp. 45-54.

[7] Yuming Zhou and Hareton Leung, Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. IEEE Transactions on Software Engineering, Vol. 32, No. 10, 2006

[8] Yogesh Singh, Arvinder Kaur and Ruchika Malhotra, "Software Fault Proneness Prediction Using Support Vector Machines" . Proceedings of the World Congress on Engineering (WCE), Volume I, 2009.

[9] JieXu, Danny Ho and Luiz Fernando Capretz, "An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction", Journal of Computer Science Vol. 04 Issue7, pp. 571-577, 2008.

[10] Unger B. and Prechelt L., "The Impact of Inheritance Depth on Maintenance Tasks – Detailed Description and Evaluation of Two Experimental Replications" Technical Report, Karlsruhe University: Karlsruhe, Germany, 1998.