Quantitative Assessment of Inheritance Hierarchies for Aspect Oriented Software Development using a proposed Aspect Inheritance Reusability Model

Senthil Velan S
Department of Computer Science and Engineering
Amity University
Dubai, UAE
svsugana@gmail.com

***Abstract*—Aspect Oriented Software Development (*AOSD*) is a promising methodology for efficiently capturing the crosscutting functionalities (*concerns*) into modular units called *aspects*, thereby reducing maintainability, and increasing reusability and ease of software evolution. Since inheritance of classes and aspects plays a vital role in defining the units of encapsulation, it is essential that the impact of introducing inheritance in *AOSD* need to be quantitatively captured using core design level metrics and mapped onto higher level quality attributes. In this research work, a new set of metrics have been proposed to quantify the impact of using multi-level inheritance hierarchies in an Aspect Oriented software. A model that captures the reusability using inheritance of aspects has been used for defining and applying the proposed metrics and later relates them onto higher level quality attributes. Further, the proposed metrics and model have been applied to *AspectJ* versions of an *SOA* based case study application. Based on the measurement, it was inferred that aspectization has improved the higher level quality attributes of reusability, modularity and maintainability of the case study over its versions.**

Keywords—AOSD, AOP, Multi-Level Inheritance, AOP Metrics

I. INTRODUCTION

Aspect Oriented Software Development (*AOSD*) [1] methodology lets designers and eventually programmers to neatly encapsulate cross-cutting concerns. This method of encapsulation was initially proposed by Gregor Kiczales [2] and quite a number of research work have been attempted to measure its quality. *AOSD* allows the software developer to isolate the units of functionality that cuts across the basic functionalities of the system under development. These isolated functionalities are encapsulated into well-defined units called *Aspects*. Aspects are the fundamental units in this novel software development methodology.

Inheritance is a very common and basic notion which can be used to extend or redefine the functionalities encapsulated in any object oriented software. Almost all software developed using Object Oriented Software Development (*OOSD*) methodology uses inheritance in various forms for the efficient utilization of encapsulated entities. There are also several issues related to inheritance [3] that have been well debated in the literature. For example, multiple inheritance is considered to create a lot of complexities compared to its utility which is accepted in large by the research community. This is evident from the statement "Multiple Inheritance is good, but there is no good way to do it" written by Steven Cook in OOPSLA '87 [4]. Hence, the usage and impact of inheritance in any methodology need be thoroughly studied.

The concept of inheritance can also be used in development of software using *AOSD* methodology. It introduces additional set of elements like abstract and concrete aspects. These elements are very similar to abstract, base and extended classes available in object-oriented software development. Extension of different kinds of aspects and classes have their own limitations.

The consequence of applying the concept of inheritance to *AOSD* methodology need to be further explored. This will enable the designer to understand its impact towards the higher level quality attributes. Hence, a set of metrics is required to measure the effect of inheritance in *AOSD* methodology. This research work proposes a set of related metrics and a third one to measure the impact of inheritance in classes and aspects and correlates them with the higher level quality attributes.

The rest of the paper is organized as follows: Section II expands the existing work in the literature about the measurement of inheritance in Aspect Oriented Programming (*AOP*). The reusability model in *AOP* is explained in Section III. The new set of metrics proposed in this research work is introduced in Section IV. Section V lists the core objectives of the research explained in this article. The metrics are applied on to a case study application which has been explained in Section VI. The metrics values obtained by applying them on to the case study application is explained in Section VII. Section VIII discusses the inferences on applying the proposed metrics on the case study application. Finally, Section IX concludes and provides pointers for the future scope of this research work.

II. EXISTING WORK

It is found in the literature that very little has been done in addressing the complexities introduced by inheritance in *AOSD*. Most of the authors have only proposed different types of coupling that are introduced while weaving aspects to the classes. Some authors have even measured it and explained about the effect of aspectization.

Zhao [5] has defined a new metric called Aspect-Inheritance Dependence Measure that counts the dependencies between an aspect and all the ancestor classes of the aspect. The metric is specified and categorized according to the mathematical properties proposed by Briand et al., [6] for coupling measures of object-oriented systems. It is also stated in the conclusion that the influence of aspect and class inheritances need to be explored.

Maximilian et. al., [7] has presented methods for programmers to automatically check the impact of

introductions and hierarchy modifications on existing programs. The author also points out the impact of dynamic or binding interference emerging out by using the specified *AspectJ* features. A prototype has been developed and reasonable results were obtained, but the implementation of the impact analysis and extension of the set of analyzable programs was not done.

A composition model has been proposed by Havinga et. al., [8] which can be used to express object and aspect compositions including various forms of inheritance and delegation. The author proposed a novel composition model and has implemented single, multiple and Beta-style inheritance and delegation of calls to the right object during composition.

Vinobha et. al., [10] have defined a new set of metrics capturing the occurrence of different forms inheritance in both *OO* and *AO* software. A proposed evaluation model has been applied on a multi-version case study application. The results found using a tool for measurement have shown encouraging results of introducing inheritable entities in *AO* versions of the software. Similarly, a weighted set of metrics extended from the *CK Metrics Suite* [9] will throw additional insight on the effect of introducing inheritance in *AOP*. Senthil Velan [11] has extended this work by calculating the complexity of computational intelligence for a multi-version *AspectJ* application.

III. The Proposed AIRM Model

In order to understand the impact of using inheritance in AOP, a simple model is required to capture the interactions between the design elements of the system. The elements of the system in focus are the abstract and the concrete aspects as well as the abstract and concrete classes. To capture the interactions that happen between these different elements an inheritance relationship model is proposed for the specification of metrics. Consider the inheritance model specified in Fig. 1.

IV. Proposed Metrics

In order to measure the effect of applying inheritance in *AOSD*, it must be possible to state the specific quantitative units of measurement. Metrics that precisely captures the inheritance effect in *AOSD* is required to be defined and used.

Consider the following units before specifying the metrics:

Let N_1 be the number of join points of all pointcuts only in class *C*,
Let N_2 be the number of join points of all pointcuts in the ancestor classes of class *C*, and
Let N_3 be the number of join points of all pointcuts in the concrete and ancestor aspects woven on class *C*.

The following are the first set of metrics proposed for measuring the impact of inheritance in *AOSD*:

Effect of Inheritance of classes in class *C*,

$$E_o I_o(C) = \frac{N_1}{N_1 + N_2} \tag{1}$$

Effect of Inheritance of aspects in aspect *A*,

$$E_o I_o(A) = \frac{N_1}{N_1 + N_3} \tag{2}$$

The third metric is an extension of Depth of Inheritance (*DIT*) of the *CK Metric Suite* [9]. This metric is modified in order to include the effect of inheritance in *AOSD*.

Consider the following units of encapsulation before specifying the metric:

Let η_1 be the number of ancestor classes that can affect class C, and
Let η_2 be the number of ancestor aspects that can affect an aspect *A* and directly used by class *C*.

The following is the third metric proposed for measuring the impact of inheritance in *AOSD*:

Class Aspect Ratio for Depth of Inheritance,

$$CAR_o DIT = \frac{\eta_1}{\eta_2} \tag{3}$$

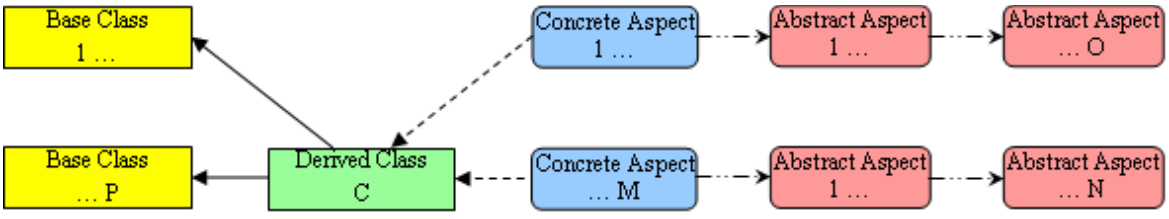

Fig. 1. Aspect Inheritance Reusability Model (*AIRM*)

V. Research Question

This objective of this research work has been to explore the multiple facets of the notion of inheritance and composition in the context of *AOSD*, quantitatively capture them with appropriate new design-level metrics and map them to higher level quality attributes such as reusability, maintainability and extensibility.

Based on the *AIRM Model* shown in Fig. 1, the following set of interactions can be identified:

1) The effect of having multiple concrete and abstract aspects whose pointcuts advice the join points present only in the derived class, class *C*.
2) The effect of having multiple concrete aspects whose pointcuts advice the join points present only in the base classes of the derived class *C*.

3) The effect of having multiple abstract aspects whose pointcuts advice the join points present only in the base classes of the derived class *C*.

The Depth of Inheritance Tree (*DIT*) proposed in *CK Metrics Suite* [9] can also be extended to *AOSD* by measuring and comparing the following interactions:

1) The effect of class inheritance on the derived class *C*.
2) The effect of aspect inheritance on the derived class *C*.

VI. Case Study Application

Quantitative assessment of the proposed metrics requires a testbed for its application. In order to do so, a multi-version SOA based case study application is considered, which was developed and used in our previous work. The architecture of the case study application is shown in Fig. 2.

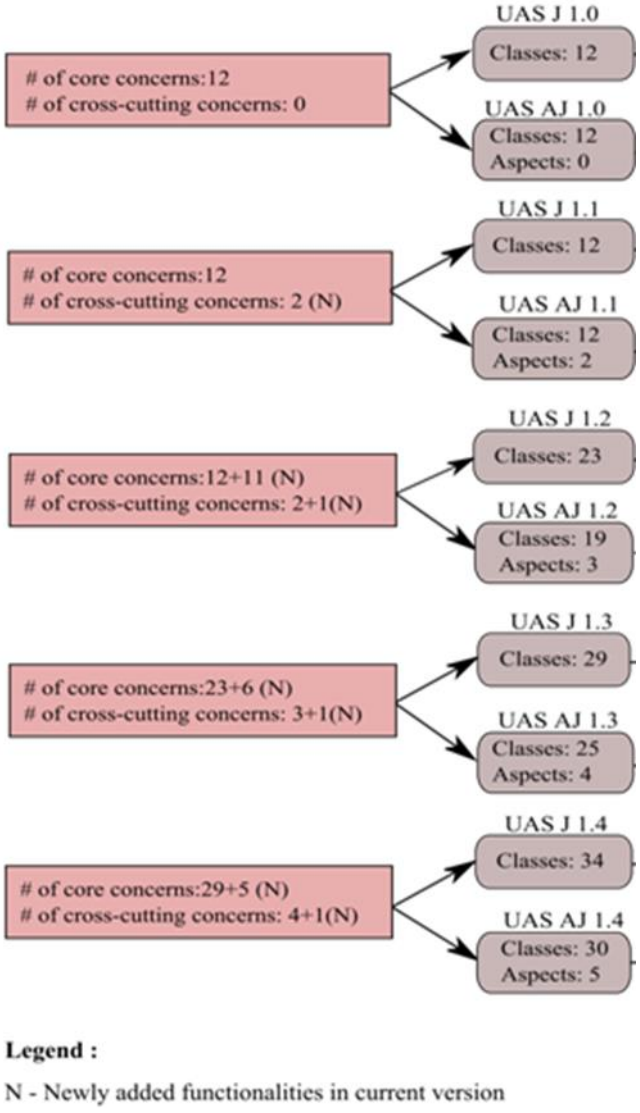

Fig. 2. Concerns in the versions of University Automation System (UAS)

Five incremental versions of *SOA* based University Automation System (*UAS*) case study have been developed using *Java* and *AspectJ* programming languages. Initially, new concerns have been added in the *Java* versions and later the cross-cutting concerns in the *Java* versions have been refactored and modelled as aspects in the AspectJ versions. The first version *UAS J 1.0* and its equivalent *UAS AJ 1.0* do not have any cross-cutting concerns modelled in their design and implementation. Second version of the case study contains 12 core concerns and 2 cross-cutting concerns modelled in the respective classes in Java version and aspects in the *AspectJ* version. The fifth version of the case study is modelled with 34 core concerns and 5 cross-cutting concerns.

Inheritance of classes and aspects are modelled using classes and aspects in the respective versions of the case study application. The cross-cutting concerns of *logging* and *exception handling* mechanisms have been modelled using abstract and concrete classes in the *Java* version and abstract and concrete aspects in the *AspectJ* versions of *UAS*. The abstract classes and aspects are extended using multi-level inheritance hierarchies in the respective versions of the case study application.

VII. Measured Values Of Proposed Metrics

TABLE I. Measured Metric Values For AspectJ (*UAS AJ*) Versions of UAS Application

Version	$E_oI_o(C)$	$E_oI_o(A)$	CAR_oDIT
UAS AJ 1.0	0	0	0
UAS AJ 1.1	0.5	1	1
UAS AJ 1.2	0.5	0.5	1
UAS AJ 1.3	0.36	0.57	0.66
UAS AJ 1.4	0.35	0.55	0.55

Three new metrics have been proposed to understand the effect of using multi-level Inheritance in *AOP* using *AspectJ* programming. In order to understand its impact the proposed metrics have been applied on the *AspectJ* versions of the *UAS* case study application. The measured values of the proposed metrics are tabulated in Table I. Measured values for all the three metrics across versions of the *UAS AJ* cases study application is available in the following Table.

Measurements have been made only for the *AspectJ* (*UAS AJ*) versions of the case study application. The reason behind the identifiable measurement is that all the proposed metrics has primarily considered only the constructs defined by *AOP* in *AspectJ* programming language. The constructs considered in the measurement are join points and pointcuts well defined in the hierarchy during multi-level inheritance.

VIII. Discussion on Measurements

In order to understand the effect of using multi-level inheritance in AOP, a clear understanding of the quantitative assessment is required by the software designers and modelers. To enable such an understanding, the measured values of the proposed metrics have been pictorially represented as a bar graph in Fig. 3

The following inferences are derived based on the quantitative assessment of the case study using the proposed metrics:

- Values of the *EoIo(C)* metric decreases over versions of *UAS AJ* application. This has happened since the inheritance hierarchy based cross-cutting functionalities

have been refactored and modelled as inheritance hierarchies. Since *AOP* was introduced to improve software modularity, these aspects are now able to improve the overall modularity of the *UAS AJ* over its versions. This naturally improves the maintainability and reusability of modular elements in the case study application.

- Notably, the values of *EoIo(A)* metric increases over versions of *UAS AJ* case study application. The reason behind this is the increase in the number of inheritance hierarchies of aspects. The inheritance hierarchies have been modelled as tangled and scattered functionalities in *Java* (*UAS J*) versions of the case study application. Aspectization has helped in modularizing the cross-cutting inheritance hierarchies from *Java* to *AspectJ* versions. The effect of aspectization has increased the modularity of the case study application
- The third metric, *CARoDIT*, decreases over AspectJ versions case study application. The reason behind this is the initial versions are modeled with lesser number of inheritance hierarchies which is spread equally in classes and aspects. The last two versions of the case study are modelled with *logging* and *exception handling* cross-cutting functionalities. When these concerns are modelled in *AspectJ*, the number of inheritance hierarchies modelled as aspects increases and thereby decreasing the value of the metric. Modelling the functionalities in this manner improves the modularity and reusability of the software.

In an overall point of view, the aspectized versions of the *UAS* case study has shown improvement in the higher level quality attributes, such as, reusability, maintainability and modularity.

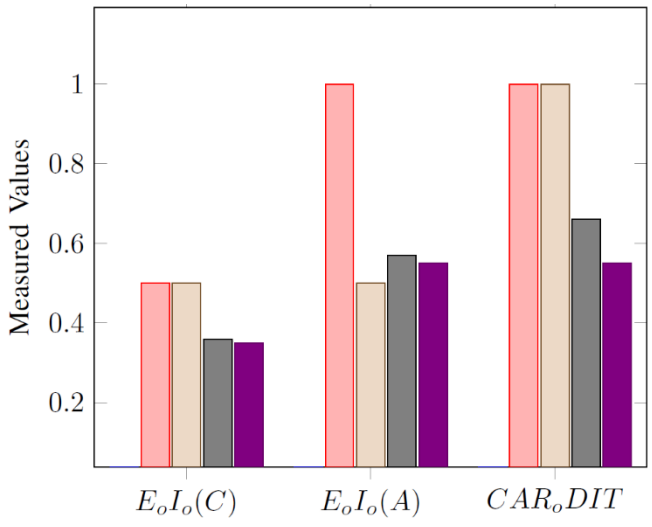

Fig. 3. Comparison of proposed metrics over *UAS AJ* versions

IX. Conclusion and Future Work

Object Oriented Software Development (*OOSD*) inherently provides capabilities and constructs to design the modular entities for reusability. *AOSD* provides mechanisms for the clear separation of cross-cutting concerns and the tangled and scattered inheritance hierarchies in *OO* can be aspectized to improve the modularity of the software. Hence, it has become necessary to study the impact of using inheritance hierarchies for software modelled using *AOSD*.

Focused on the need stated in the last paragraph, it is necessary to quantitatively evaluate the effect of modelling inheritance hierarchies in versions of software developed using *AOP*. In this research, a new set of specific metrics focusing on quantifying inheritance hierarchies have been defined and applied on five *AspectJ* versions of the case study. The measurements have been used to understand the effect of aspectizing *OO* software over its versions onto higher level quality attributes.

Based on the quantitative measurement it was inferred that using inheritance in *AOP* has improved the modularity, maintainability and reusability of the case study application over its versions. Since the application was modelled using multi-level inheritance a natural extension of this work is to understand the effect of using different types of inheritance hierarchies modelled in the case study. Further, the proposed metrics will also be applied in more case study application to understand more about domain specific inferences.

References

[1] Henrique Rebêlo and Gary T. Leavens, "Aspect-Oriented Programming Reloaded," In *Proceedings of the 21st Brazilian Symposium on Programming Languages (SBLP 2017)*, ACM, New York, NY, USA.

[2] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C. V., Loingtier J. M., and Irwin J., "Aspect-Oriented Programming," In *European Conference on Object Oriented Programming*, pp. 220-242, 1997.

[3] Taivalsaari, Antero, "On the notion of inheritance," *ACM Computing Surveys (CSUR)*, Vol. 28, No. 3, pp. 438-479, 1996.

[4] Jos Warmer, John Hogg, Steve Cook and Bran Selic, "Experience with Formal Specification of CMM and UML," Object-Oriented Technology, ECOOP'97 Workshop Reader, ECOOP'97 Workshops, pp. 216-220, 1997.

[5] Jianjun Zhao, "Measuring Coupling in Aspect-Oriented Systems," In *10th International Software Metrics Symposium (METRICS'2004)*, (Late Breaking Paper), Chicago, USA, September 14-16, 2004.

[6] Briand L. C., J. Daly and J. Wuest., "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, Vol.25, No.1, pp. 91-121, January/February 1999.

[7] Maximilian Störzer, Jens Krinke, "Interference Analysis of AspectJ Programs," *3rd German Workshop on Aspect-Oriented Software Development*, Essen, Germany, March 2003.

[8] Wilke Havinga, Lodewijk Bergmans Mehmet Aksit, "A model for composable composition operators: Expressing object and aspect compositions with first-class operators," In Proceedings of *9th International Conference on Aspect-Oriented Software Development, AOSD '10*, ACM Press, 145-156, Rennes and Saint-Malo, France, 2010.

[9] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, Issue 6, June, 1994, pp. 476-493, IEEE Press, Piscataway, NJ, USA.

[10] Vinobha A., Senthil Velan S. and Babu C., "Evaluation of reusability in aspect oriented software using inheritance metrics," *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, Ramanathapuram, 2014, pp. 1715-1722.

[11] Senthil Velan S., "Investigating the Complexity of Computational Intelligence using the Levels of Inheritance in an AOP based Software, " *2019 Advances in Science and Engineering Technology International Conferences (ASET 2019)*, Dubai, 2019.