Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Science and Engineering

Elixir Comp. Sci. & Engg. 91 (2016) 38605-38609

ISSN: 2229-712X

AOP and its impact on software quality

Kotrappa Sirbi[1] and Prakash J Kulkarni[2]

[1]Department of Computer Science & Engineering, K L E's Dr.M.S.Sheshgiri College of Engineering & Technology, Belgaum, India.
[2]Department of Computer Science & Engineering, Walchand College of Engineering, Sangli, India.

ARTICLE INFO

Article history:
Received: 11 May 2012;
Received in revised form:
23 February 2016;
Accepted: 27 February 2016;

Keywords
Aspect Oriented Programming (AOP),
Metrics,
Design patterns (dps),
Separation of concern,
Aspects.

ABSTRACT
This has been asserted that AOP allows attaining better design quality systems than those built with OOP, namely by reducing scattering and tangling from the research literature. There is limited quantitative and qualitative research supporting those arguments which already were published. Software metrics are very important validating procedure of quality software and software properties of aspect oriented programming (AOP). Object oriented programming (OOP) implementations may suffer certain drawbacks that will affect the design quality of the system and thus its system maintainability, comprehensibility, understandability and testability. Aspect oriented programming (AOP) facilitates with powerful quantification constructs to handle design quality. In this paper, we compare two versions (Java and AspectJ) of observer patterns to further investigate within the context of design quality. We evaluated the system based on the value of these design quality metrics RFC, CBM, LCO and WMC.We claim that the AOP has significance effect on design quality than OOP

Introduction

To validate the impact of aspect oriented on software quality the quantitative and qualitative studies are essential [1, 10, 7, 15]. The research may consist of empirical assessment of verification of AOP and its impact on software characteristics such as evolvability, maintainability, understandability, and quality [7]. The critical question is how we can quantify when using AOP is advantageous. Metrics is an important method in quantifying software and software development characteristics [16]. However, metrics have to be used proficiently and cautiously. Academic and empirical support of metrics and of their relation to software attributes are an unwieldy and lengthy process. It is of principal importance that we authenticate the usefulness of metrics we use in order to enable others to use them as well [2]. Till now, the metrics used and projected for AOP are seldom validated. It is not enough to confirm their definitions right but also their utility to explain software characteristics has to be validated [2]. In many instances, this can only be accomplished through forbidden experiments or through analyzing huge volumes of data. In both situations, statistical assessment is a key method to check hypotheses [16]. Our main objectives were to provide the grounds to answer the following research questions.

1. Do Design Patterns have effect on quality when we change system from OOP to AOP?
2. Will there be any a significant change on the overall design quality metrics?

The roadmap of the paper is as follows, Section 2 provides motivation on design quality metrics evaluations. Section 3 give an overview of Aspect Oriented (AO) design quality metrics for Aspect-Orientated Programming (AOP), Section 4 discuss experimental study for AOP design quality metrics, results and analysis of experimental study, Section 6 gives threats to the validity of software quality metrics and Section 7 gives about related work in AOP software quality metrics. Lastly Section 8 includes the conclusion of the paper.

Tele:
E-mail addresses: kotrappa06@gmail.com

Motivation

Evolution of OOP in three decades of its existence has proved the superiority over the procedure, function, logical and objects oriented paradigms. AOP in late 1990's has become known as new buzzword for modularization of crosscutting concerns which otherwise spread across the system. There are many research studies have recommended about AOP quality and modularization of crosscutting concerns concepts [5, 6, 7, 9, 10]. With this limited number of empirical studies, these statements about separation of concern by AOP or coupling measure by AOP can't be generalize beyond university examples [2]. We further investigate about AOP and its effect on software quality in evaluating genuine existence of case studies.

AOP Quality Metrics

AOP Quality Model

Our quality model defines a terminology and clarifies the relationships between the reusability, maintainability and the metrics suite. It is a useful tool for guiding software engineers in data interpretation. It was defined based on a set of assumptions. The definition of our quality model is based on: (i) an extensive review of a set of existing quality models [6], (ii) classical definitions of quality attributes and traditional design theories, such as Parnas' theory [2], which is commonly accepted among researchers, and practitioners and (iii) the software attributes impacted by the aspect-oriented abstractions. The quality model has been built and refined using Basilis GQM methodology [10, 12] (see Figure 1). The metrics for AOP software quality shown in Table 2 and model is shown in Table 1.

For comparison of software quality of Java and AspectJ techniques, we adopted the G-Q-M (Goal-Question-Metric) method .G-Q-M Describes a measurement software system on three stages (Figure 1) begin with goal [13].

The goal is to treat in questions that split into proven statements. Every questions are connected with metrics that, when measured, will offer information to reply the question. Our main objective is to only compare of Java and AspectJ system

TABLE I. AOP QUALITY MODEL

Quality Type	Product Perspective	Characteristics	Sub-characteristics
Quality Software Product		Functionality	Suitability
			Accuracy
			Interoperability
			Security
			Reusability
		Reliability	Maturity
			Fault Tolerance
			Recoverability
		Usability	Understand-dability
			Learn-ability
			Operability
			Attractiveness
			Complexity
		Efficiency	Time behavior
			Resource behavior
			Code-reducibility
		Maintainability	Analyzability
			Changeability
			Stability
			Testability
			Modularity
		Portability	Adaptability
			Replace-ability
			Install-ability
			Co-Existence
		Evolvability	Extensibility
			Sustainability
			Design Stability
			Configurability

TABLE II. METRICS DEFINITIONS

Metrics	Definition
WOM/ WMC	Number of operations in a given module and its equivalent to the WMC metrics from CK metrics suite.
DIT	Length of the longest path from a given module to the class/aspect hierarchy root.
CIM	Number of modules or interfaces explicitly named in the pointcuts of a given module
CFA	Number of interfaces or modules declaring fields that are necessary by a given module.
CBM/ CBO/ CFA/ CMC	Number of modules or interfaces declaring methods or fields that can be called or accessed by a given module. It's equivalent to CBO metric from CK metric suite and combination of CFA and CMC.
LCO/ LCOM	Number of pairs of operations working on different class fields minus pairs of operations working on common fields (zero if negative). Its similar to LCOM of OO metric.
RFM/ RFC	Number of methods and advices potentially executed in response to a message received by a given module. Its is similar to RFC from CK metrics suite.

Measurement System

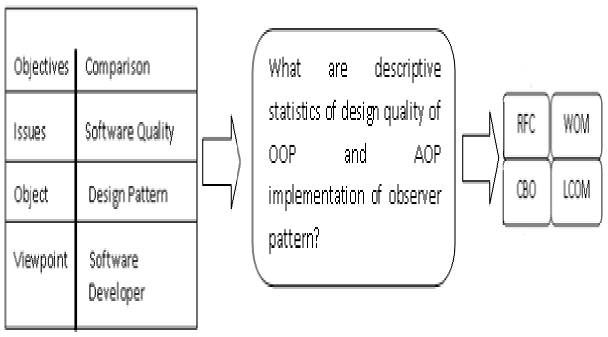

Figure 1.G-Q-M Model

with regards to design quality from software developer viewpoint. Some experimental studies [1, 2] are conflict design quality with the help of modularity metrics. With these set of software attributes was firstly suggested by Yourdon and Constantine [2] for measuring of modularity. Later this structured design methodology was adapted to OO methodology by Coad & Yourdon, Grady Booch and Meyer. There are many pragmatic studies [2] confirm that impact of coupling and cohesion on modularity. In spite of concept of cohesion and coupling in software design for almost more than 50 years, still we do not have generally accepted metrics for them[2]. In the empirical study [2], the author maintained CBO (Coupling Between Object classes) and LCOM (Lack of Cohesion in methods), adapted from C&K metrics suite [2]. CBO is a measure of the number of other modules to which a module is coupled .These two modules are coupled when methods declared in one module use methods or instance variables of the other module [7, 2].LCOM is the degree to which methods within a module are related to one another. It is a count as the number of pairs of methods working on different attributes minus the pairs of methods working on at least one shared attributes(zero if negative)[7].Experimental study

This study uses implementations of the GoF design patterns made freely available by Hannemann & Kiczales [4]. For each pattern there is a small example that makes use of the pattern, and implemented the example in both Java and AspectJ. The AspectJ implementations are considered as one of the nearest things to examples of good AOP style and design [2]. The Java implementations correspond to the sample C++ implementations in the GoF book [3]. In the measurement process, the data was gathered by the AOPmetric's tool [11]. This tool implements the metric suite proposed by Ceccato & Tonella [14]. For experimentation purpose, we have taken popular observer pattern for GoF design patterns and many researchers used this example, but ours is different because it's based Aspect Oriented Quality model to evaluate the improved AOP quality.

Case Study

Observer pattern, known as Model-View is indented to "define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically". Object oriented implementations of the Observer pattern; usually add a field to all potential Subjects that stores a list of Observers interested in that particular Subject. When a Subject wants to report a state change to its Observers, it calls its own *notify* method, which in turn calls an *update* method on all Observers in the list. Figure 2 shows a concrete example of the Observer pattern in the context of a simple figure package. In such a system the Observer pattern is used to cause mutating operations to figure elements to update the screen. The code for implementing this pattern is spread across the classes.

The underlined methods contain code necessary to implement this instance of such a pattern. All participants (i.e. *Point* and *Line*) have to know about their role in the pattern and consequently have pattern code in them. Adding or removing a role from a class requires changes in that class. Changing the notification mechanism requires changes in all participating classes.In the AspectJ version [14] all code pertaining to the relationship between Observers and Subjects is moved into an aspect, which changes the dependencies between the modules, as shown in Figure 3. Subject and Observer roles crosscutclasses, and the changes of interest (the *subjectChange* pointcut) crosscuts methods in various classes. In this paper, we have decided to assess the implementation of the observer design patterns in both Java and AspectJ. First, we applied the metrics in Hannemann and Kiczales original code [4]. Afterwards, we changed their implementation to add new participant classes to play pattern roles. These changes were introduced because Hannemann and Kiczales' implementation encompasses few classes per role (in most cases only one) [4]. Hence we have decided to add more participant classes in order to investigate the pattern crosscutting structure. Finally, we have applied the chosen metrics to the changed code. We analyzed the results after the changes, comparing with the results gathered from the original code (i.e. Before the changes) [4, 14].

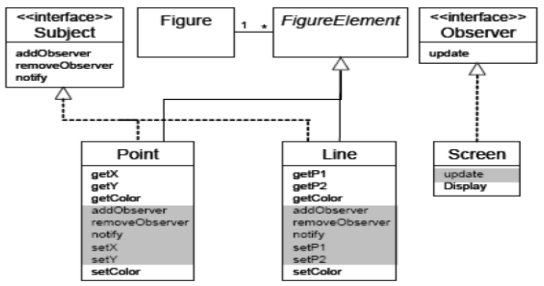

Figure 2. Observer pattern in Java

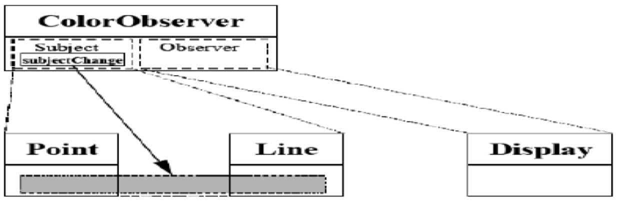

Figure 3. Observer pattern in AspectJ

Results

Table 2 gives total, mean, and maximum and standard deviation values of OOP and AOP C & K metrics for observer patterns. Also, it compares values of total, mean, maximum, standard deviation and 95 % confidence interval for significance level 0.05 (100*(1- α)%) for both OO and AO observer pattern. Here, we present the measurement results for the observer design patterns; we focus on the presentation of results related to RFC, CBM, LCO and WMC from the C&K metric suite and their effects on software quality (see table 2). The relatively high value of standard deviation for CBM and LCO indicates a high variation among the values of these metrics. The results show smaller average for a number of operations per class(WOM(WMC)), response for class(RFM(RFC)), coupling between objects(CBO(CBM)) and lack of cohesionO(LCOM)) values for AOP observer patterns. The rest of the metrics shows almost same trends. Additionally, low standard deviation for almost all of the AOP metrics make these averages more meaningful and consistent. At times, the mean value of an entity may be misleading particularly when there is a very large variation among the values.

Analysis

TABLE III. OBSERVER PATTERN QUALITY METRICS – DESCRIPTIVE STATISTICS

Metrics	**RFC/ RFM**		**WOM/ WMC**		**CBO/ CBM**		**LCOM /LCO**	
	OO	AO	OO	AO	OO	AO	OO	AO
Total	21	28	21	18	5	8	28	19
Mean	4.2	3.11	4.2	2	1	0.8	5.6	2.1
Maximum	10	8	10	7	2	5	24	10
Std.Dev	4.26	2.6	3.83	2.3	1	1.6	10.43	4.2
95% *Confidence Interval*	3.73	1.5	3.36	1.0	0.8	1.7	9.14	2.7

The general observation on the overall design quality of OO and AO metrics values is shown in Table 3 which indicates smaller variations of standard deviation for all of AO metrics as compared to their OO version. This is because of the reason that almost all metrics values fall in line a small range with very small outliers.

RFM/RFC

These metrics indicates the coupling measures and as per table 2 this metrics shows reduced coupling for AO observer pattern as compared to OO version. We can conclude that there is an improvement in design quality with AOP over OOP. A small value of RFC/RFM is treated as a good design which is assumed to increase the understandability and testability.

WOM/WMC

These metrics will indicates complexity.A small value of WOM/WMC is treated as a good design which is assumed to reduce complexity and maintainability. There is a clear indication that by a small value of this metric seen in the improved performance measurements.

CBO/CBM

CBM of AOP is near concomitant to the CBO of OOP version of the observer pattern. It is possibly that the mainly important measure characteristics of the couplings which are the important inspiration for the normal shift from OOP to AOP. This metric indicates the improvement of maintainability and reusability from OOP to AOP of observer pattern.

LCOM/LCO

There must be high cohesion between methods/operations in OO or AO design. High variations were observed for OOP than AOP.A prominent popular of metric AOP measures are indicating a very large cohesion which signify an enhanced design practice.

Based on the results, we have observed that the measure relative to cohesion (LCOO, RFC, LCOM, CBO) and complexity of operations (WOC (WMC), RFC, DIT). In general, the AO solutions were superior in terms of RFC, WMC and LCOM measures, since the use of AspectJ reduces the overuse of inheritance mechanisms. However, as illustrated in Table 2, most measures indicated that AspectJ implementations resulted in higher coupling (CBO (CBC)).However, a careful analysis of the implementation show that these higher CBC and LOC values for AO solutions in general are related to the presence of generic aspects in several AspectJ pattern implementations, which have the intension of making the solution more reusable. In this paper, we assessed that the impact of the observer pattern on AOP design quality attributes such as maintainability, understandability, reusability, efficiency and testability. We argue that based on measures of metrics and their combined effect on software quality attributes AOP solutions are superior to OOP solutions , but this statement cannot be generalized still AOP adoption needs a lot more empirical studies to accepted it as a better quality than OOP. The comparative analysis of improved AOP software quality measures are shown in Table 4.The complete description of the data and a more detailed discussion of the results of this empirical study are beyond the scope of this paper.

TABLE IV. IMPROVED AOP SOFTWARE QUALITY

Metric	**AOP Software Quality**		
	Characteristics	***Winner***	***Loser***
RFC/ RFM	Understandability	AO	OO
	Testability		
WOM/ WMC	Performance	AO	OO
	Maintainability		
	Complexity		
CBO/ CBM	Maintainability	OO	AO
	Reusability		
LCOM/ LCO	Reusability	AO	OO
	Efficiency		

Threats to validity

We have seen a number of limitations of this research that are significance mentioning. Applying metric gives numeric value for some property as a result, but that alone is not useful in general. Every metric model must define how it should be measured and what are good/bad measured mean values. Also, the metrics is used to improve software quality based on their relation with quality factors and what are methods and techniques required to improve the software to improve results. Metrics gives us objective information about properties of software to evaluate software quality. As depth of inheritance (DIT) grows, it is likely that classes on the lower level inherits lots of methods and overrides few. DIT measures reuse via inheritance and larger DIT means greater design complexity. Response for a class (RFC) as it grows complexity of software increases and understandability decreases. Number of children (NOC) with very high value may be a candidate for refactoring to create much maintainable hierarchy. NOC is a measure of reuse and also indicator of required testing time. As RFC grows testability become harder and with high RFC, there would be a better class division. High coupling between objects (CBO) would decreases understandability, increases complexity and makes maintenance difficult. With high LCOM classes can be more fault-prone and also good class encapsulation. Weighted methods per class (WMC) is an indicator of the amount of efforts required to implement and test a class. As the number of methods for a class grows, they become more specific to application and thus limiting possibilities for reusability. High coupling between object classes (CBO) is undesirable because more coupling between object classes is harmful to modular design and prevents reusability.

Related work

There is a few related research works focusing on quantitative and qualitative assessment of AOP design patterns quality. Some different reimplementations of DPs in existing systems from real world have been performed to improve the Dips quality [1, 7]. A set of existing metrics has been used to evaluate the quality of different AOP implementations [7, 14]. In [4] Hannemann and Kiczales give an implementation of the original solution to the observer pattern. Although it is known that AOP Aspect J provides developer separating crosscutting concerns, the impact of AOP on software modularity is not yet well investigated [2]. Acceptable value for metrics evaluation with different views of quality , and it is hard to find a numerical value for quality which could be acceptable by all the people. Also, having different views affects software categorization in certain classification by considering the numerical value as the only parameter on software evaluation [6]. A very large collection of C&K's metrics values for OOP is already provided by an online measurement repository called OOMJ[12]. Recently the experiment was conducted by Adam P [2]. He also compared the AO and OO implementations of the Gang-of-Four patterns. Threshold of software metrics can be used as indicators to identify possible anomalies in software [14]. There are 10 AO software metrics proposed by Ceccato and Tonella, which revise the well known C& K's metrics suite [8, 14].

Conclusion/future work

In this paper, we investigate OO DPs and AOP effect on design quality. Our evaluation is based on measurements on AOP software metrics such as *RFC, CBM, LCO and WMC* from the C&K metric suite, which was adapted to AOP. An approach to reimplement observer patterns by AOP & Aspect J is presented in this paper and analyzed for its quality factors. Based on quality metrics, experimental results indicate that AO improves reusability, maintainability, understandability and testability. We hope that this research work stimulates some argument about the effect of AOP in software development. We argue that AOP has significance effect on improving AOP software quality, but this statement can't be generalized, it needs further investigation.

Future work will consider experimentation consisting of 23 GoF DPs and non-GoF DPs from different domains and also from the definition of new design solutions to improve the quality of AOP system. As for as we know, this is the first presentation of experimental study to this effect on OO DPs and AOP design quality.

References

[1] Khan .R.A and Mustafa, K et, al, "An Empirical Validation of Object -Oriented Design Quality Metrics", J King Saud Univesity., Vol, 19, Comp. & Info, Sci., pp 1-16, Riyadh (1427H), 2007.

[2] Adam, P, "An Empirical Assessment of the Impact of Aspect –Oriented Programming on Software Modularity", ENASE'2010, Athens – Greece, 22 - 24 July 2010.

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[4] Hannemann, J and Kiczales, G., "Design Patterns Implementation in Java and AspectJ", Proc. of Object Oriented Programming Systems Languages and Applications, OOPSLA '02, 161-173, Nov 2002.

[5] Hachani, O and Bardou D., "On Aspect-Oriented Technology and Object-Oriented Design Patterns" , Proc. of European Conference on Object Oriented Programming ,ECOOP 2003.

[6] Khashayar, K and Yann-Ga., "On Issues with Software Quality Models", workshop on quantitative approaches in Object-Oriented Software Engineering, 2005.

[7] Bernardi.M.L and Di Lucca, G.A, "Improving Design Pattern Quality Using Aspect Orientation", 13th IEEE Workshop on Software Technology and Engineering Practice, 2005.

[8] Chidamber and Kemerer, "A Metrics Suite for Object Oriented Design". IEEE Trans. Softw. Eng.20 (6), 476–493, 1994.

[9] Madeyski and Szała, "Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study". IET Software Journal 1(5), 180–187, 2007.

[10] Kumar and Grover, "A quantitative evaluation of aspect-oriented software quality model", AOSQUAMO, ACM SIGSOFT Software Engineering Notes Volume 34, Issue 5, September 2009.

[11] Stochmiałek , AOPmetrics, Web site http://aopmetrics.tigris.org

[12] Konstantina,G., Ayaz F, et al., Web site http://donner.cs.unimagdeburg.de:8080/oomj

[13] Basili, Caldiera and Rombach, "The Goal Question Metric Approach", Encyclopedia of Soft. Eng., vol. 2, September 1994, pp. 528-532, John Wiley & Sons, Inc, 1994.

[14] Komsan S and Pornsiri M, "Determining Threshold of Aspect-Oriented Software Metrics", the Third International Joint Conference on Computer Science and Software Engineering (JCSSE2006), Bangkok, Thailand, June 29-30, 2006.

[15] Kotrappa S and Prakash J Kulkarni, "Design Patterns Vs Aspect Oriented Programming –A Qualitative and a Quantitative Assessment Systems", International Journal of Computer Science & Communication (IJCSC), Volume 1, No.2,

pp: 233-237, July-December 2010.
[16] Foutse K and Yann-Gael, G, "Do Design Patterns Impact Software Quality Positively?" CSMR, 12th European Conference on software Maintenance and reengineering, 2008.

Kotrappa Sirbi received the Bachelor of Engineering from Mysore University, Mysore in 1985, M S (Software System) from BITS, Pilani in 1994 and M Tech (CSE) from VTU, Belgaum. In 2005 he became a PhD candidate in the Department of Computer Science & Engineering, Walchand College of Engineering, Sangli, India. He has published more than 10 research articles both in international and national conferences and journals. He is presently working at KLE's BCA, Belgaum, on deputation from K L E's College of Engineering & Technology, Belgaum. His current research interests include software engineering, object oriented software development, and aspect oriented software development and quality metrics for OOP and Aspect-Oriented Programming.