



International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

Impact of Aspect Oriented Programming on Software Maintainability - A Descriptive Study

Sarita Rani¹, Puneet Jai Kaur²

University Institute of Engineering and Technology,
Panjab University, Sector 25, Chandigarh
INDIA

Abstract: Software maintenance is a term of major interest and it is a valuable part of software development cycle. One of the main aspects of software quality in software products is maintainability. Further, there are four sub categories of maintainability: analyzability, changeability, stability, and testability. Maintainability plays a vital role in improving software quality as software changes/updates are frequently required in this process. Object oriented programming has contributed in improving software maintainability, but crosscutting concerns affects the maintainability of object oriented software. In these days, Aspect oriented programming (AOP) is rising as a new methodology which provides more modularization of crosscutting concerns that help the programmers to reuse the code. This paper presents the various software maintainability metrics for an AOP and also discusses the various case studies which were conducted to assess the maintainability of software.

Keywords: Aspect Oriented Programming (AOP), Object Oriented Programming (OOP), AO system, OO system, Maintainability, Changeability.

I. Introduction

The major objective of software developer is to deliver highest software quality. ISO 9126 standard illustrated six main attributes of software quality: maintainability, functionality, efficiency, reliability, usability, and portability [1, 2]. Maintainability has further four forms: adoptive maintainability, corrective maintainability, preventive and perfective maintainability. Among these types a huge amount of effort in terms of cost is spent on the enhancements of composing elements of component-based software systems [3]. The sub-traits of maintainability are: testability, analyzability, changeability, and stability. Changeability is the most important attribute from organization's viewpoint, as most organizations operate other organization's software.

Software maintenance is a very important and costly activity that expends software development cost up to 50-70 percent [4]. Due to this reason, developers have designed methodologies for development that can lessen the effects of change, ease the interpretation of the program, and promote the initial detections of fault and can be preferred. OOP contributed in improving software maintainability by applying the approaches of object and encapsulation, but crosscutting concerns affect the maintainability of object oriented software. Crosscutting concerns can be described as the traits of software system whose operation is disseminated into many modules, which leads to code tangling and code scattering problem. This limitation is overcome by AOP. AOP is a new emerging methodology which provides more modularization of crosscutting concerns that helps the programmers to reuse the code. A concern is defined as a feature which includes all the functional and non functional requisites and the design constraints in the system, that a system should implement [5].

The remaining paper is formulated as follows. Section 2 represents a concise outline of aspect-orientation and AspectJ explaining what aspects are. Section 3 presents a brief introduction to maintainability and discusses the several surveys that were accompanied to figure out the resourcefulness of maintainability metrics. Section 3 also lists the various maintainability metrics. Section 4 provides conclusion of the study.

II. Aspect-Orientation & Software Metrics

AO allows a developer to specify, modularize and encapsulate the crosscutting concerns in modules instead of having them tangled within the system's components. For AO to achieve its objective of increasing software reuse and providing better software designs, software metrics are required. Software metrics are also needed for AO to determine the correct design practices when designing AO systems. AO System can also be poorly designed as one can write poor object-oriented software. Aspects introduction in the object-oriented software might increase the system complexity and reduce its understandability. Due to these reasons metrics are required for measuring how efficient, understandable, and reusable an aspect-oriented design is.

A. AspectJ

AspectJ [6] is a slightly modified form of Java, called JCore that yields, from the definition of new constructors, support for modular employment of crosscutting concerns. AspectJ is used for expressing the core functionality of a program. Various crosscutting concerns such as synchronization, consistency checking, protocol

management and others, has been successfully modularized by AspectJ. AspectJ supports the following definitions:

- *Aspect*: An aspect is the new modular unit design to implement a crosscutting concern and its definition contain when, where and how to invoke a concern.
- *Join points*: A joint point is well defined point in the code at which concern crosscut the application. There can be many join points. For example, method call, access to class members and the execution of exception handler blocks.
- *Pointcuts*: On the basis of specified criteria, a pointcut chooses a set of join points.
- *Advice*: An advice is used to put behavior before, after, or around the elected join points. The three types of advice is executed as follows:
Before Advice: before the join point
After Advice: after the join point
Around Advice: executes the join point zero or more times and surrounds its execution.
- *Introduction (Inter-type declaration)*: Introduction adds variables and new methods to a class, notifies that an interface is implemented by a class, and also permits aspects to customize the static structure of a program.

III. AO Maintainability Metrics

A. Background & Literature Review

In software development methodology, a crucial role is carried out by maintainability. Software maintainability is an essential attribute of software quality. Software maintainability is different from software maintenance. Software maintainability allows software system or component to be altered, fix the defects, enhances performance, or other attributes, whereas software maintenance reorganizes a software system or component after delivery to fix the defects for better performance [7]. Changeability is a sub attribute of software maintainability and it defines the amount of effort to change a system. Various factors that affect the maintainability of AO are size, coupling, cohesion and separation of concerns.

The case studies and the metrics that are exercised to estimate the maintainability of AOP software are summarized below in Table I.

Table I Summary of Maintainability Metrics

Author	Parameters	Metrics Tested	Result
Mguni et al. [8]	Changeability	San't Anna metrics and all Ceccato and Tonella metrics	• Concern level metrics (Sant'Anna metrics) are good indicators of changeability. • Structural complexity metrics (Ceccato and Tonella metrics) are poor indicators of changeability.
Shen et al. [10]	Changeability (coupling and maintenance tasks)	Ceccato and Tonella metrics for coupling	• Correlation model within system maintainability and coupling metrics
Kumar et al. [11]	Changeability	WOM	• Evaluated the correlation between change impact factor and WOM. • WOM cannot be adopted as a stable pointer of changeability.
Kulesza et al. [12]	Coupling, cohesion, separation of concerns	Sant'Anna metrics	• AOP attained an upgraded separation of concerns, exhibited components with weaker coupling and internal complexity as it required fewer LOC. • LCOO metric unsatisfactory for measuring maintainability.

Mguni et al. [8] computed the maintainability of COTS-based system developed using AOP. By using the programming language AspectJ, an identical implementation in AOP is generated by refactoring the OpenBravoPos and Jaspereports. The metrics were classified into two categories: concern level metrics and structural complexity metrics. An AOP based implementation is well maintainable than OO system and the concern level metrics are more acceptable indicators of changeability as compared to structural complexity metrics.

Mohanta [9] presented prototyping techniques and the modeling work for perfective maintainability that focuses on the relevance of project analysis. A set of exercises were proposed by using criterion such as time, quality, and efficiency which are mandatory for process quality analysis of perfective maintainability. This approach aids in accomplishing better performance of perfective maintainability with the concepts of AOP in case of any modification.

Shen et al. [10] undertook a study to measure software variations during system development. Their study represents a correlation model between coupling metrics and system maintainability and also a fine-grained coupling metrics suite for AO. AJMetrics, a coupling metrics tool, was implemented and an experimental study was carried out on eight aspect benchmarks. The correlation model is used for assessing the maintainability of AO systems and offers suitable information for this purpose.

Kumar et al. [11] studied change impact in AOP systems. The refactored AOP systems, from their OO versions, were used and 149 OO modules were refactored into 129 modules of AOP systems. Average change impact in OO is greater than in AO systems, which indicates that OO systems are less maintainable than AO systems. The change impact will be higher for these modules, if crosscutting concerns are not mined to aspects and others do.

Kulzeza et al. [12] estimated the amount to which each solution produces maintainable software decomposition by comparing the OO and AO versions of typical web-based information system with reference to essential maintainability characteristics. At the system and component levels, AO systems are somewhat preferable over OO systems. The inter-related behaviors were not accumulated by some aspects and it leads to lower cohesion in AO system.

Ceccato et al. [13] explored the trade-off between merits and demerits attained by AOP approach through a measuring routine. The routine is based on a metrics suite that expands the conventional metrics applicable with the OO paradigm. The valid properties, such as the ratio of the system influenced by an aspect and the amount of familiarity an aspect has of the modules it crosscuts, are acquired by the proposed metrics.

Sant'Anna et al. [14] presented a schema, to facilitate the computation of aspect-oriented software with regards to reusability and maintainability depends on a quality model and a suite of metrics. The proposed metrics fulfils essential pre-requisites in order to accomplish successful measurements in the AOSD context.

B. Metrics

Different researchers have proposed different metric suite. Sant'Anna et al. [14] proposed following metrics:

1. SoC Metrics:

It is the capability to analyze, encapsulate, and modify that part of software which is related to a specific concern. In following we describe the SoC metrics:

- *Concern Diffusion over Components (CDC):*
CDC computes the number of aspects and classes responsible for the implementation of a concern. The design metric also computes the number of other classes as well as the aspects that access them.
- *Concern Diffusion over Operations (CDO):*
CDO computes the number of primary operations such as methods and advices responsible for the implementation of a concern.
- *Concern Diffusion over LOC (CDLOC):* CDLOC for each concern computes the number of transition points with the help of code lines. Transition points are the points in the code where shadowed area converts into non- shadowed area and vice-versa.

2. Coupling Metrics

Coupling is defined as a degree of dependence among components. In following we describe the coupling metrics:

- *Coupling Between Components (CBC):*
CBC counts the number of other aspects and classes to which an aspect or class is coupled.
- *Depth of Inheritance Tree (DIT):*
DIT measures the maximum distance from a given module to the aspect/class hierarchy.

3. Cohesion Metric

The cohesion of a component refers to the degree to which its internal components are related. High cohesion generally correlates with low coupling. In following we describe the cohesion metric:

- *Lack of Cohesion in Operations (LCOO):*
LCOO computes the lack of cohesion of an aspect or a class, or it measures the relationship between the methods in a given module.

4. Size Metrics

The extent of a software system's design and code is software size. In following we describe the size metrics:

- *Vocabulary Size (VS)*:
VS computes the number of aspects and classes into the system.
- *Lines of Code (LOC)* :
LOC computes the number of lines of code.
- *Number of Attributes (NOA)*:
NOA computes the number of attributes of each aspect or class.
- *Weighted Operations per Components (WOC)*:
WOC computes the number of operations such as advices or methods of each aspect or class.

Ceccato et al. [13] proposed following metrics:

- *Weighted Operations per Module (WOM)*:
WOM counts the number of advices or methods in a module.
- *Number of Children (NOC)*:
NOC counts the number of immediate sub- classes or sub- aspects of a given module.
- *Coupling on Field Access (CFA)*:
CFA is the number of modules or interfaces defining fields which are accessed by a given module and it also measures the dependences of a given module on other modules.
- *Coupling on Method Call (CMC)*:
CMC is the number of interfaces or modules defining operations which are likely called by a given module.
- *Coupling Between Modules (CBM)*:
CBM is the number of fields and operations which can be represented by the number of outward arrows from a given module and are accessed by a given class.
- *Crosscutting Degree of an Aspect (CDA)*:
CDA is the number of modules in a given aspect influenced by pointcuts and introductions.
- *Coupling on Advice Execution (CAE)*:
CAE is the number of aspects consists of advices invoked in a given module by the execution of operations.
- *Response for Module (RFM)*:
RFM is the number of advices and methods possibly accomplished by a given module in response to a received message.
- *Lack of Cohesion of Operations (LCO)*:
LCO computes the interconnection in between the methods of a given module.

IV. Conclusions

This paper presented the various relevant metrics which are used to assess the maintainability. This paper also discusses several surveys that were accompanied to figure out the resourcefulness of maintainability metrics. WOM cannot be adopted as a stable pointer of changeability because the correlation factor between the change impact and WOM are found to be weak. LCOO metric are found to be uncertain for measuring maintainability. The metrics suggested by Sant' Anna et al. [14] are good indicators of changeability. This paper concludes that AO systems are more adaptable to changes in comparison to OO systems and therefore AOP systems are conveniently maintainable than OOP systems.

V. References

- [1] J. Boegh, S. Depanfilis, B. Kitchenham, and A. Pasquini, "A method for software quality planning, control, and evaluation" in *IEEE Software*, vol 16, pp. 69-77, April 1999.
- [2] H. Jung, S. Kim, and C. Chung, "Measuring software product quality: A survey of ISO/IEC 9126" in *IEEE Software*, vol 21, no.5, pp. 88-92, September 2004.
- [3] cimitile@unisannio.it University of Sannio, Faculty of Engineering at Benevento Italy 29 November, 2000.
- [4] P. Jalote, *An Integrated Approach to Software Engineering*, Springer, Berlin, Germany, 3rd edition, 2005.
- [5] I. Jacobson and P. -W. Ng, *Aspect-Oriented Software Development with Use Cases*, Addison-Wesley Object Technology Series, Addison-Wesley Professional, Reading, Mass, USA, 2004.
- [6] Aspectj project <http://eclipse.org/aspectj/>.
- [7] IEEE, "*IEEE Standard Glossary of Software Engineering Terminology*," IEEE Std 610.12-1990, 1990.
- [8] K. Mguni, and Y. Ayalew, "An Assessment of Maintainability of an Aspect-Oriented System" in *ISRN Software Engineering*, vol. 2013, 2013.
- [9] V.K.K Mohanta, "Maintainability of software using aspect oriented programming for process quality analysis" *International Journal of Engineering and Computer Science*, vol 1, no. 3, pp. 137-142, December 2012.
- [10] H. Shen, S. Zhang, and J. Zhao, "An empirical study of maintainability in aspect oriented system evolution using coupling metrics" in *Proceedings of the 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE'08)*, pp. 233-236. June 2008.

- [11] A. Kumar, R. Kumar, and P.S. Grover, "An evaluation of maintainability of aspect-oriented system: a practical approach" *International Journal of Engineering and Computer Science*, vol 1, no. 2, pp. 1-9, 2007.
- [12] U. Kulesza et al., "Quantifying the effects of aspect-oriented programming: a maintenance study" in *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pp. 223–232, September 2006.
- [13] M. Ceccato and P. Tonella, "Measuring the effects of software aspectization," in *Proceedings of the 1st Workshop on Aspect Reverse Engineering (WARE '04)*, 2004.
- [14] C. Sant'Anna, A. Garcia, C. Chavez, C. Lucena, and A. V.von Staa, "On the reuse and maintenance of aspect-oriented software: an assessment framework," in *Proceedings of the 17th Brazilian Symposium on Software Engineering*, pp. 19–34, 2003.