Analysis of Maintainability Metrics for Aspect Oriented Software & Object Oriented Software

Puneet Jai Kaur¹, Sarita Rani²
University Institute of Engineering and Technology,
Panjab University, Sector 25, Chandigarh
INDIA

Abstract: Software maintenance is a valuable part of software development. Maintainability is one of the main aspects of software quality in software products. Object Oriented Programming has played a crucial role in improving software maintainability, but crosscutting concerns affects the modularization of object oriented software. Aspect oriented programming is a new paradigm which is supposed to provide more modularization of crosscutting concerns which benefits the programmers to reuse the code. The study consists of analytic comparison between the OO version and AO version of five projects on the basis of chosen maintainability metric. The results show that an AOP version is more maintainable than OOP version.

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

I. Introduction

ISO 9126 standard characterized six main attributes of software quality: maintainability, efficiency, functionality, reliability, usability, and portability [1, 2]. The primary intention of software developer is to deliver maximum software quality. Maintainability is further categorized as: adoptive maintainability, corrective maintainability, preventive maintainability and perfective maintainability. Among these types an enormous amount of effort in terms of cost is expended on the enhancements of composing elements of component-based software systems [3].

In software development methodology, a vital role is carried out by maintainability. Software maintainability is an essential attribute of software quality. Software maintenance is different from software maintainability. Software maintenance reorganizes a software system or component after delivery to fix the defects for improved performance, whereas software maintainability allows software system or component to be altered, fix the defects, enhances performance, or other attributes [4]. Software maintenance is a necessary and costly activity that expends software development cost up to 50-70 percent [5]. Due to this reason, developers have designed techniques for development that can ease the analysis of the program, reduce the effects of change and aid the initial detections of fault and can be preferred.

A concern is defined as a feature which includes all the functional and non-functional requisites and the design constraints in the system, and can be mapped easily to different modules [6]. Crosscutting concerns affect the maintainability of object oriented software. Crosscutting concerns are difficult to factor out in functions and classes such as logging, exception handling and authorization checks. Crosscutting concerns are of two types: static and dynamic. Static crosscutting concerns allow changes into the classes and interface of the system, whereas dynamic crosscutting alters system behaviour by enhancing or substituting the core program execution flow in a way that cuts across modules. When these crosscutting concerns are implemented using OOP, then it leads to code tangling and code scattering problem. This limitation is overcome by AOP. AOP is a new emerging paradigm which is supposed to provide more modularization of crosscutting concerns which benefits the programmers to reuse the code.

The remaining paper is formulated as follows. Section 2 gives a concise outline of aspect-orientation and AspectJ explaining what aspects are. Section 3 discusses the chosen metrics, the selected projects, the computed statistics, the data collection procedure, and the evaluation criteria for the chosen metrics. Section 4 presents results and conclusions of the metrics. Section 5 provides conclusion of the study.

II. Aspect Orientation & Software Metrics

AO is about specifying, modularizing and encapsulating the crosscutting concerns in modules instead of having them tangled within the system’s components. For AO to achieve its objective of providing better software
designs, increasing software reuse software metrics are required. AO software metrics are also needed to
determine the appropriate design practices when designing AO systems. AO systems can also be poorly
designed like OO systems. An introduction of aspects in the object-oriented software might reduce its
understandability and increase the system complexity. These are the reasons due to which metrics are required
for measuring how understandable, efficient, and reusable an aspect-oriented design is.

A. AspectJ

AspectJ [7] is an extension to Java programming language called JCore, which yields support for modular
employment of crosscutting concerns. AspectJ is used for defining the core functionality of a program. Various
crosscutting concerns such as synchronization, protocol management, consistency checking and others, has been
successfully modularized by AspectJ. AspectJ supports the following definitions:

- **Aspect**: An aspect is a modularized employment of a crosscutting concern and its definition contain
  when, where and how to invoke a concern.
- **Join points**: A joint point represents well defined point in the code at which concern crosscut the
  application. There can be many join points. For example, access to class members, method call and the
  execution of exception handler blocks.
- **Pointcuts**: A pointcut chooses a set of join points on the basis of specified criteria.
- **Advice**: An advice sets behavior before, after, or around the chosen 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 carried out by a class, and also permits aspects to customize the static structure of a
  program.

III. Empirical Metric Data Collection & Evaluation Criteria

This section discusses the chosen metrics for the study, projects used to gather empirical data, computed statistics,
data collection properties and evaluation criteria.

A. Chosen Metrics

For empirical study, five metrics were selected for AO and OO software [8]: number of children (noc), depth of
inheritance (dit), lack of cohesion of methods (lcom), weighted methods per class (wmc), and lines of code (loc).
These metrics helps in comparison with the classical object oriented software, in terms of how the modules are
linked regarding inheritance and how wide the classes and aspects of a given software system.

B. Selected Projects

For this empirical study five projects from open source repositories were selected. Two versions of each project
were selected: AO version and OO version. Table 1 provides summary of projects.

<table>
<thead>
<tr>
<th>Projects Id</th>
<th>Aspect Version</th>
<th>Java Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>AJHotDraw</td>
<td>JhotDraw</td>
</tr>
<tr>
<td>P2</td>
<td>AspectJ Design Pattern</td>
<td>JavaDesign Pattern</td>
</tr>
<tr>
<td>P3</td>
<td>AspectJ Calculator</td>
<td>Java Calculator</td>
</tr>
<tr>
<td>P4</td>
<td>Aspectj Producer Consumer Problem</td>
<td>Java Producer Consumer Problem</td>
</tr>
<tr>
<td>P5</td>
<td>Aspectj Resource pool Management</td>
<td>Java Resource pool Management</td>
</tr>
</tbody>
</table>

C. Computed statistics

AOP metrics [9] was used to collect the metrics for all the aspect oriented and object oriented projects. For each
metric, mean was selected as a measure of central tendency and the standard deviation as a measure of dispersion.
D. Data Collection Properties

Fenton et al. [10] defined some properties which were used for the data collection process and are described as follows:

- Accuracy: the higher the difference between the actual data and measured data, the lower is the accuracy and vice-versa.
- Replicability: means that the analysis can be done at different times by different people by using the same setting.
- Correctness: According to the metrics definition data was collected.
- Precision: Data is expressed by number of decimal places. Less decimal places shows a lower accuracy.
- Consistency: It counts the differences with the metric values when collected using different tools by different people.

E. Evaluation Criteria

All the metrics values of AO and OO projects are compared on the basis of mean and standard deviation. Ratio between the mean value of metric for aspect version and the mean value for object oriented version is also used for evaluation. If value of this metric comes greater than 1, it indicates that mean value of the metric is higher in the aspects than in the class and vice versa.

IV. Results & Conclusion

A. Number of Children (NOC)

NOC computes the number of immediate sub-aspects or sub-classes of a given module. Following viewpoints from Chidamber et al. [11] were adapted for noc metric:

- The higher the values of noc, the higher the level of reuse.
- A class with a large number of children is a case of misuse of subclassing.
- The value of number of children provides an idea of the influence on the design by a class. A class with a large number of children requires more testing of the methods in that class.

### TABLE III SUMMARY STATISTICS FOR NOC VALUES

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Mean $x$ (AO)</th>
<th>Std. Dev. (AO)</th>
<th>Mean $x$ (OO)</th>
<th>Std. Dev. (OO)</th>
<th>Ratio $x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1.567</td>
<td>2.229</td>
<td>2.109</td>
<td>2.503</td>
<td>0.743</td>
</tr>
<tr>
<td>P2</td>
<td>0.5</td>
<td>0.764</td>
<td>0.5</td>
<td>0.866</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>P4</td>
<td>0.5</td>
<td>0.5</td>
<td>1.5</td>
<td>0.866</td>
<td>0.333</td>
</tr>
<tr>
<td>P5</td>
<td>0.8</td>
<td>0.4</td>
<td>2</td>
<td>0.0</td>
<td>0.4</td>
</tr>
<tr>
<td>Average</td>
<td>0.673</td>
<td>0.778</td>
<td>1.222</td>
<td>0.847</td>
<td>0.551</td>
</tr>
</tbody>
</table>

Table II shows summary statistics for noc metric and figure 1 shows the empirical data for noc metric both in AO and OO system. Value of the ratio metric is less than 1 i.e. 0.551, it indicates that mean value of the metric is higher in the OO version than in the aspect version.

B. Depth of Inheritance Tree (DIT)

DIT measures the maximum distance from a chosen module to the aspect/class hierarchy. Following viewpoints from Chidamber et al. [11] were adapted for dit metric:

- The deeper a class is in the hierarchy, more it is likely to inherit the number of methods, it becomes more complicated to predict its behavior.
- Due to more methods and classes are involved, deeper trees comprises of greater design complexity.
- The deeper a specific class is in the hierarchy, the potential reuse of inherited methods is more.
TABLE III Summary Statistics for Dit Values

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Mean $\bar{x}$ (AO)</th>
<th>Std. Dev. (AO)</th>
<th>Mean $\bar{x}$ (OO)</th>
<th>Std. Dev. (OO)</th>
<th>Ratio $\bar{x}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0.251</td>
<td>1.727</td>
<td>2.538</td>
<td>1.363</td>
<td>0.099</td>
</tr>
<tr>
<td>P2</td>
<td>1.5</td>
<td>0.5</td>
<td>1.5</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>0.0</td>
<td>1</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>0.0</td>
<td>1.5</td>
<td>0.5</td>
<td>0.667</td>
</tr>
<tr>
<td>P5</td>
<td>1.2</td>
<td>0.4</td>
<td>1</td>
<td>0.0</td>
<td>1.2</td>
</tr>
<tr>
<td>Average</td>
<td>0.99</td>
<td>0.525</td>
<td>1.507</td>
<td>0.473</td>
<td>0.657</td>
</tr>
</tbody>
</table>

Table III shows summary statistics for dit metric and figure 2 shows the empirical data for dit metric both in AO and OO system. Value of the ratio metric is less than 1 i.e. 0.657, it indicates that mean value of the metric is higher in the OO version than in the aspect version.

C. Lack of cohesion of Methods (LCOM)

LCOM measures the relationship among the methods in a given module or it computes the lack of cohesion of an aspect or a class. Following viewpoints from Chidamber et al. [11] were adapted for lcom metric:

- Cohesiveness of methods within a class is required, because it increases encapsulation.
- Lack of cohesion means classes must be split into two or more subclasses.
- Low cohesion leads to increase in complexity, thereby increasing the probability of errors during the development process.

TABLE IV Summary Statistics for LCOM Values

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Mean $\bar{x}$ (AO)</th>
<th>Std. Dev. (AO)</th>
<th>Mean $\bar{x}$ (OO)</th>
<th>Std. Dev. (OO)</th>
<th>Ratio $\bar{x}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0.238</td>
<td>0.343</td>
<td>0.256</td>
<td>0.327</td>
<td>0.93</td>
</tr>
<tr>
<td>P2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.25</td>
<td>0.433</td>
<td>0.0</td>
</tr>
<tr>
<td>P3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>P4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-</td>
</tr>
<tr>
<td>P5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.375</td>
<td>0.217</td>
<td>0.0</td>
</tr>
<tr>
<td>Average</td>
<td>0.048</td>
<td>0.069</td>
<td>0.176</td>
<td>0.195</td>
<td>0.246</td>
</tr>
</tbody>
</table>

Table IV shows summary statistics for lcom metric and figure 3 shows the empirical data for lcom metric both in AO and OO system. Value of the ratio metric is less than 1 i.e. 0.246, it indicates that mean value of the metric is higher in the OO version than in the aspect version.

D. Weighted Methods per Class (WMC)

WMC computes the number of advices or methods in a module. Following viewpoints from Chidamber et al. [11] were adapted for wmc metric:

- The complexity of methods and the number of methods participated is a predictor of how much effort and time is needed to develop and maintain the class.
- The higher the number of methods in a class the more the potential influence on children, because children will inherit all the methods defined in the class.
- A class which involves large numbers of methods is probably to be more application specific, restricting the possibility of reuse.
Table V shows summary statistics for wmc metric and figure 3 shows the empirical data for wmc metric both in AO and OO system. Value of the ratio metric is less than 1 i.e. 0.816, it indicates that mean value of the metric is higher in the OO version than in the aspect version.

E. Lines of Code (LOC)

LOC calculates the number of lines of code. Following viewpoints can be made regarding loc metric:

- The value of loc can be used as an indicator of complexity [10].
- Value of loc can be also used as a weak indicator of how much effort and time is required to develop and maintain an aspect [10].
- The value of loc metric is also used to show the rates related to quality attributes, such as defects per loc [12].
- The loc is also used to computes the quantity of documentation of a module (comments per loc) and the method size (mean loc per method) [10].
- Methods with low values of loc are easily reusable.

Table VI shows summary statistics for loc metric and figure 5 shows the empirical data for loc metric both in AO and OO system. Value of the ratio metric is less than 1 i.e. 0.667, it indicates that mean value of the metric is higher in the OO version than in the aspect version.

V. Conclusion

This paper presented an empirical study for the assessment of the maintainability of AO projects by comparing them by their OO version. This paper argues that the impact of AOP on the efficiency of software development is significant. Five metrics were chosen for the assessment of maintainability. Every metric shows better results for AO projects than the OO projects. LCOM metric is found to be good indicators of maintainability of the system, since it has a value of ratio x is 0.246 and WMC metric is found to be weak indicators of maintainability of the system since it has a value of ratio x is 0.816. This paper concludes that AOP systems are conveniently maintainable than OOP systems.
VI. References


[3] cimitle@unisannio.it University of Sannio, Faculty of Engineering at Benevento Italy 29 November, 2000.


