CHAPTER VIII

APPLYING ASPECT-ORIENTED SOFTWARE MAINTAINABILITY METRICS

Chapter VIII shows applications of the metrics to evaluate fifty systems and to compare maintainability between systems written in AspectJ and written in Java. The results show that the metrics can be applied to evaluate and to compare maintainability of both systems implemented by an aspect-oriented approach and an object-oriented approach.

8.1 Measuring Aspect-Oriented Systems

Table 8.1: Measurement of fifty systems

Systems	*M(S)*	*AM(S)*	Systems	*M(S)*	*AM(S)*
Adapter	-2	-0.5	Composite2	-6	-0.6
Facade	-5	-0.83	Bridge2	-2	-0.25
Composite	-5	-0.42	Singleton2	-3	-1
Bridge	0	0	Observer2	-4	-0.44
Singleton	-2	-0.67	Mediator2	-5	-0.56
Observer	-7	-0.78	Proxy2	-6	-0.75
Mediator	-5	-0.56	Chain of Responsibility2	-2	-0.29
Proxy	-5	-0.63	Flyweight2	-2	-0.29
Chain of Responsibility	-7	-0.58	Builder2	-4	-0.36
Flyweight	-6	-0.86	Factory Method2	-5	-0.72
Builder	-1	-0.2	Abstract Factory2	-3	-0.25
Factory Method	-2	-0.25	Prototype2	-16	-2.67
Abstract Factory	-1	-0.09	Memento2	-3	-0.43
Prototype	-4	-0.67	Template Method2	-2	-0.67
Memento	-6	-0.75	State2	-5	-0.625
Template Method	-1	-0.2	Strategy2	-1	-0.33
State	-3	-0.43	Command2	-8	-0.89
Strategy	-9	-1.13	Interpreter2	-1	-0.13
Command	-8	-0.67	Decorator2	-5	-1.25
Interpreter	-2	-0.2	Iterator2	-10	-0.91
Decorator	-4	-1	Visitor2	-9	-1.29
Iterator	-2	-0.4	HindiManner	-2	-0.5
Visitor	-6	-0.5	Telecom	-8	-0.62
Adapter2	-2	-0.4	Spacewar	-31	-0.82
Facade2	-2	-0.33	AspectTetris	-14	-0.67

This section shows an application of the metrics to evaluate fifty AspectJ systems. Fifty systems compose of *two sets of twenty-three design patterns* implementation from [32, 41], *three tutorial software* from [12, 42], and *one academic software* from [43]. These systems have totally 426 units which are 253 classes, 16 abstract classes, 75 aspects, 22 abstract aspects, and 60 interfaces. The number of

units contained in each system is varied from 3 – 38 units, approximately 8 units per system.

Table 8.1 presents the *degree of maintainability of a system* (*M(S)*) and the *average degree of maintainability of a unit* (*AM(S)*) of fifty systems. The results show that 15 of 50 systems have poor maintainability (the systems which their *AM(S)* values are highlighted with the darken grey) because their average degree of maintainability per unit are less than -0.70. Bigger systems tend to contain more faults than smaller systems, so the maintainability should not always be judged by the software size. Considering average maintainability per unit (*AM(S)*) which reflects the fact of system maintainability, the smaller system such as *Prototype2* system which contains only 6 units can be harder to maintain than the bigger system such as *Spacewar* system which contains 38 units.

8.2 Comparing between Aspect-Oriented Systems and Object-Oriented Systems

Table 8.2: Measurement of systems in two versions

Software	AM of OO software	AM of AO software
Adapter	-0.5	-0.5
Facade	-0.6	-0.83
Composite	-1	-0.42
Bridge	-0.3	0
Singleton	-0.7	-0.67
Observer	-1.2	-0.78
Mediator	0	-0.56
Proxy	0	-0.63
Chain of Responsibility	0	-0.58
Flyweight	-0.8	-0.86
Builder	0	-0.2
Factory Method	0	-0.25
Abstract Factory	-0.1	-0.09
Prototype	0	-0.67
Memento	-0.3	-0.75
Template Method	-0.5	-0.2
State	-0.1	-0.43
Strategy	0	-1.25
Command	0	-0.67
Interpreter	-0.2	-0.2
Decorator	-0.3	-1
Iterator	0	-0.4
Visitor	-0.1	-0.5

In comparing between two versions, well-known collection of design patterns, which are developed under the aspect-oriented concept and the object-oriented

concept [32] which are used in many research works [35, 16, 17], are applied. The authors, owners of the design pattern implementations, claim that the aspect-oriented concept improves modularity in 17 or 23 systems. Table 7.2 presents the results of evaluating those systems following the proposed metrics. The results argue that the average maintainability degrees of 15 systems are decreased after they are applied the aspect-oriented concept, the average maintainability degrees of 2 systems are stable, and the average maintainability degrees of only 6 systems are increased.

After walking through the code, I presume that decreases in the average maintainability degrees more than expected after applying the aspect-oriented approach are happened because developers might implement systems in the Java version first then they adapted the aspect-oriented concept to transform the code to AspectJ version. These ways of implementations prevent them from writing the clean code since the beginning. The results also show that the metrics can be used to evaluate and to compare the maintainability of either systems written in AspectJ or Java.