CHAPTER VII

DEFINING ASPECT-ORIENTED SOFTWARE MAINTAINABILITY METRICS

Chapter VII describes processes of defining aspect-oriented software maintainability metrics. Firstly, assumptions about the effects of design principles on a set of maintainability-related metrics are set. Secondly, those metrics are measured automatically from two sets of systems (systems violating design principles and systems obeying design principles) and are evaluated their changes in metric results. Finally, the qualified design principles with positive results are selected to construct aspect-oriented software maintainability metrics. The proposed metrics are specified in three levels that are the *unit level*, the *system level*, and the *system comparison* level.

7.1 Selecting Design Principles for Aspect-Oriented Software Maintainability

Table 7.1: The validation metric suit [21, 45]

Attributes	Metrics	Definitions
Separation of Concerns	Coupling on Advice Execution (CAE)	Counts a number of aspects containing advices possibly triggered by the execution of operations in a given module.
Coupling	Coupling on Field Access (CFA)	Counts a number of modules or interfaces declaring fields that are accessed by a given module.
	Coupling on Method Call (CMC)	Counts a number of modules or interfaces declaring methods that are possibly called by a given module.
	Depth of Inheritance Tree (DIT)	Counts a length of the longest path from a given module to the class/aspect hierarchy root.
	Number Of Children (NOC)	Counts a number of immediate subclasses or sub-aspects of a given module.
Cohesion	Lack of Cohesion in Operations (LCO)	Counts number of pairs of operations working on different class fields minus pairs of operations working on common fields (zero if negative). When all methods don't access to any field, then LCO = 0
Size	Lines of Class Code (LOCC)	Counts number of lines of class code.
	Weighted Operations in Module (WOM)	Count number of operations in a given module
	Number of Public Methods (NOPM)	Count the number of public methods

The design principles in Chapter IV are analyzed their association with maintainability using a hundred samples and a set of metrics. Nine selected metrics

representing 4 aspects of the software engineering attributes: *separation of concerns*, *coupling*, *cohesion*, and *size*, are described in Table 7.1. Each design principle is investigated its impact on systems following it and violating it from changes in metric values. Section 7.1.1 sets assumptions of the effects of design principles on metrics first. Section 7.1.2 measures two set of systems violating design principles and obeying design principles and selects the design principles which produce the positive results. Lastly, the qualified design principles are combined to construct aspect-oriented software maintainability in Section 7.2.

7.1.1 Assumptions of the Design Principle Effects on Metrics

Table 7.2 summarizes all assumptions in metric changes for systems which are corrected to follow the design principles. Details of assumptions are described as follows:

- *Guideline 1* may increase LOCC from calling *proceed()*.
- *Guideline 2* and *Guideline 3* may not change all metrics.
- *Guideline 4* may decrease CFA and increase CMC, LOCC, or WOM because of accessing field indirectly via accessor methods. NOPM may increase if method access controls are public.
- *Guideline 5* may decrease LOCC, WOM, or CAE because of demoting an unnecessary aspect.
- *Guideline 6* may decrease LOCC because the duplicate method is replaced by another method or an advice.
- *Guideline 7* may decrease CMC from removing dependency between two aspects, but it may increase LOCC or WOM due to making one aspect crosscutting another aspect instead.
- *Guideline 8* may decrease LOCC, WOM, CAE, or NOPM from merging or removing two aspects which crosscut the same concern.
- *Guideline 9* may decrease CMC because of removing dependencies between aspects and classes.

Table 7.2: Assumptions of design principle influences on metrics

Design Principles	Metrics to be decreased	Metrics to be increased	Design Principles	Metrics to be decreased	Metrics to be increased
Guideline 1	LOCC	-	Heuristic 4.5	LOCC/ CMC	-
Guideline 2	-	-	Heuristic 4.7	LOCC	LOCC/ WOM/ CMC/ NOPM
Guideline 3	-	-	Heuristic 4.13	LOCC	LOCC/ WOM/ CMC/ NOPM
Guideline 4	CFA	CMC/ LOCC/ WOM/ NOPM	Heuristic 4.14	-	-
Guideline 5	LOCC/ WOM/ CAE	-	Heuristic 5.2	-	WOM/ NOPM
Guideline 6	LOCC	-	Heuristic 5.5	DIT/ NOC	-
Guideline 7	CMC	LOCC/ WOM	Heuristic 5.6	LOCC/ WOM/ NOPM	-
Guideline 8	LOCC/ WOM/ CAE/ NOPM	-	Heuristic 5.9	LOCC/ CFA	LOCC/ WOM/ NOPM
Guideline 9	CMC	-	Heuristic 5.10	LOCC/ CFA/ WOM NOPM	LOCC/ WOM/ NOPM
Guideline 10	CMC	-	Heuristic 5.13	-	WOM/ NOPM
Guideline 11	DIT/ NOC	-	Heuristic 5.15	LOCC/ WOM/ DIT/ NOC/ CMC/ CAE/ NOPM	-
Guideline 12	LOCC	-	Heuristic 5.17	LOCC/ WOM/ NPBM	-
Guideline 13	LOCC	-	Heuristic 9.2	CFA	CMC/ LOCC/ WOM/ NOPM
Guideline 14	DIT/ NOC	-	Duplicate Code	LOCC	CMC
Guideline 15	WOM/ NOPM	-	Long Method	-	LOCC/ WOM/ NOPM
Heuristic 2.1	CFA	CMC/ LOCC/ WOM	Long Parameter List	CMC	-
Heuristic 2.3	NOPM	-	Feature Envy	CMC	-
Heuristic 2.5	NOPM	-	Freeloader	LOCC	-
Heuristic 2.6	NOPM	-	Speculative Generality	LOCC/ WOM/ NOPM	-
Heuristic 2.7	CFA	CMC/ LOCC/ WOM/ NOPM	Message Chains	-	LOCC
Heuristic 2.9	WOM/ LOCC		Anonymous Pointcut Definitions	-	LOCC
Heuristic 2.10	LCO	LOCC/ WOM	Lazy Aspects	LOCC/ WOM/ NOPM/ DIT/ NOC/ CFA/ CMC/ CAE/ LCO/ NOPM	-
Heuristic 3.2	-	LOCC/ WOM	Aspect Feature Envy	-	-
Heuristic 3.7	LOCC/ WOM/ NOPM/ DIT/ NOC CFA/ CMC/ CAE/ LCO/ NOPM	-	Abstract Method Introductions	LOCC/ WOM/ NOPM	LOCC

Design Principles	Metrics to be decreased	Metrics to be increased	Design Principles	Metrics to be decreased	Metrics to be increased
Heuristic 3.8	LOCC/ WOM/ NOPM/ DIT/ NOC CFA/ CMC/ CAE/ LCO/ NOPM	-	Large Aspects	-	LOCC/ CAE
Heuristic 3.10	LOCC/ WOM/ CMC/ NOPM	-			

- *Guideline 10* may decrease CMC because of letting aspects crosscutting base classes instead.
- *Guideline 11* may decrease DIT and NOC.
- *Guideline 12* and *Guideline 13* may decrease LOCC due to the unused methods which are override are disappeared.
- *Guideline 14* may decrease DIT and NOC from removing over-design hierarchy.
- *Guideline 15* may decrease WOM or NOPM from removing methods introduced to interfaces.
- *Heuristic 2.1* may decrease CFA and increase CMC, LOCC, or WOM because changing access control of a field from non-private to private may require adding accessor methods for other units accessing the field of the given unit.
- *Heuristic 2.3* and *Heuristic 2.5* may decrease NOPM.
- *Heuristic 2.7* may decrease CFA and increase CMC, LOCC, or WOM because of accessing field indirectly via accessor methods. Heuristic 2.7 may also increase NOPM if access control of a method is changed to public.
- *Heuristic 2.9* may decrease WOM or LOCC because the unused get methods is eliminated. NOPM may decrease if the cut off get methods are public.
- *Heuristic 2.10* may decrease LCO from introducing new cohesion units. However, this introduction may increase LOCC or CMC (from constructor calls to new units).

- *Heuristic 3.2* may increase LOCC or WOM because extracting the god behavior to sub-meaningful behaviors.
- *Heuristic 3.7 – Heuristic 3.8* may decrease LOCC, WOM, CFA, CMC, CAE, LCO from demoting a unit with only set methods, get methods, print-type methods, and constructors or from cutting off the out scope units. DIT or NOC may also decrease if the unit extends another unit. NOPM may decrease if methods of that unit are public.
- *Heuristic 3.10* may decrease LOCC, WOM, or CMC from removing the agent unit. NOPM may decrease if methods of that unit are public.
- *Heuristic 4.5* may decrease LOCC from removing unused contained objects. CMC may also decrease from calling to constructors of the contained units.
- *Heuristic 4.7* may increase LOCC, WOM, CMC, or NOPM if fields are grouped into new containing units. LOCC may decrease from removing unused fields.
- *Heuristic 4.13* may decrease LOCC or LCO from removing unnecessary contained objects. However, LOCC, WOM, CMC, or NOPM may increase from extracting a unit to manage the many-to-many relationship.
- *Heuristic 4.14* may not change all metrics.
- *Heuristic 5.2* may increase WOM from introducing methods for polymorphism. NOPM may increase if these methods are public.
- *Heuristic 5.5* may decrease DIT and NOC.
- *Heuristic 5.6* may decrease LOCC or WOM from eliminating unused abstract units. NOPM may increase if methods in those abstract units are public.
- *Heuristic 5.9* may decrease LOCC or CFA from merging fields to superclass or super aspect. Heuristic 5.9 may also increase LOCC,

WOM, or NOPM from introducing accessor methods to the fields that are merged into the super unit.

- *Heuristic 5.10* may decrease LOCC, CFA, WOM, or NOPM from merging fields and methods to superclass or super aspect. Heuristic 5.10 may also increase LOCC, WOM, or NOPM from introducing accessor methods to the fields that are merged into the super unit.
- *Heuristic 5.13* may increase WOM or NOPM from introducing methods to deal with different cases.
- *Heuristic 5.15* may decrease LOCC, WOM, DIT, NOC, CMC, CAE, or NOPM from demoting the sub unit.
- *Heuristic 5.17* may decrease LOCC or WOM from removing the NOP methods. NOPM may increase if these methods are public.
- *Heuristic 9.2* may decrease CFA and increase CMC, LOCC, or WOM because fields are set indirectly via set methods. NOPM may also increase if the set methods are public.
- *Duplicate Code* may decrease LOCC from removing the duplicate part, but may increase CMC due to replacing that part with call dependency.
- *Long Method* may increase LOCC, WOM, or NOPM due to the extracted methods.
- *Long Parameter List* may decrease CMC from removing the digging data agent.
- *Feature Envy* may decrease CMC from changing inter-couplings to intra-couplings.
- *Freeloader* may decrease LOCC from vanishing of the unit header.
- *Speculative Generality* may decrease LOCC, WOM, or NOPM from removing unused methods.
- *Message Chains* may increase LOCC from extracting the complex statements.

- *Anonymous Pointcut Declarations* may increase LOCC from clearly declaring pointcuts.
- *Lazy Aspects* may decrease LOCC, WOM, CFA, CMC, CAE, and LCO from demoting aspects. DIT or NOC may also decrease if aspects extend another unit. NOPM may decrease if methods of those aspects are public.
- *Aspect Feature Envy* remains all metrics.
- *Abstract Method Introductions* may increase LOCC if concrete statements are added to that method. It may decrease LOCC, WOM, or NOPM if a concrete behavior of the sub unit is moved to that abstract method.
- *Large Aspects* may increase LOCC or CAE from extracting large aspects.

7.1.2 Design Principles for Aspect-Oriented Software Maintainability

After specifying assumptions, two sets of aspect-oriented software samples consist of examples from the source in Chapter V and systems from [12, 32, 41, 42, 43] are investigated their changes in metric values from comparing between systems which conform and violate the design principles from Chapter IV. All metrics are automatically calculated using the AOP-metrics tool [46].

Table 7.3 shows systems which violate design principles and systems which are modified to conform design principles. The green highlights show metric results which are in line with the positive assumptions (metrics to be decreased). The red highlights indicate metric results which are in line with the negative assumptions (metrics to be increased). Underline italic numbers with no highlights are metric results which are increased after modifications, but they are out of assumptions. Italic numbers with no highlights are metric results which are decreased after modifications, but they are out of assumptions.

Table 7.3: The effects of violating or following each design principle on metrics

Design Principles	Metrics									Sample Systems
	LOCC	WOM	DIT	NOC	CFA	CMC	CAE	LCO	NOPM	
Guideline 1	97	9	1	1	0	4	1	0	6	Strategy
	97	9	1	1	0	4	1	0	6	(Modified)
	38	8	0	0	0	1	2	0	6	Source
	39	8	0	0	0	1	2	0	6	(Modified)
Guideline 2	58	9	0	0	0	5	1	0	9	Facade
	58	9	0	0	0	5	1	0	9	(Modified)
	31	5	0	0	0	3	2	0	4	Source
	31	5	0	0	0	3	2	0	4	(Modified)
Guideline 3	144	22	0	0	2	7	2	3	19	StateajDev
	144	22	0	0	2	7	2	3	19	(Modified)
	33	5	3	3	0	3	3	0	4	Source
	33	5	3	3	0	3	3	0	4	(Modified)
Guideline 4	75	14	3	3	0	5	4	0	10	VisitorAj
	75	14	3	3	0	7	4	0	10	(Modified)
	26	4	0	0	1	1	1	0	1	Source
	26	4	0	0	0	2	1	0	1	(Modified)
Guideline 5	28	5	0	0	0	1	2	0	3	DecoratorAj
	24	5	0	0	0	1	1	0	3	(Modified)
	19	4	0	0	0	1	1	0	3	Source
	13	3	0	0	0	1	0	0	3	(Modified)
Guideline 6	182	29	1	1	1	8	5	28	21	Composite
	181	29	1	1	*2*	8	*6*	28	21	(Modified)
	40	8	0	0	0	2	1	0	7	Source
	36	*7*	0	0	0	2	1	0	*6*	(Modified)
Guideline 7	285	52	5	5	6	18	9	33	43	Telecom
	285	52	5	5	6	17	9	33	43	(Modified)
	41	9	0	0	0	2	2	0	7	Source
	43	13	0	0	0	1	2	0	7	(Modified)
Guideline 8	28	5	0	0	0	1	2	0	3	DecoratorAj
	24	5	0	0	0	1	1	0	3	(Modified)
	47	8	0	0	0	1	2	1	6	Source
	38	6	0	0	0	1	1	1	5	(Modified)
Guideline 9	97	9	1	1	0	4	1	0	6	Strategy
	97	9	1	1	0	3	1	0	6	(Modified)
	40	8	0	0	0	3	1	0	7	Source
	39	8	0	0	0	2	1	0	7	(Modified)
Guideline 10	97	9	1	1	0	4	1	0	6	Strategy
	87	*7*	1	1	0	3	1	0	*4*	(Modified)
	32	7	0	0	0	2	1	0	6	Source
	32	7	0	0	0	1	1	0	6	(Modified)
Guideline 11	23	5	1	1	0	2	1	0	1	HomeSaving EnergyDev
	23	5	0	0	0	*1*	1	0	1	(Modified)
	23	5	1	1	0	2	1	0	1	Source
	23	5	0	0	0	*1*	1	0	1	(Modified)
Guideline 12	272	55	5	5	0	10	6	105	39	IteratorAj
	270	55	5	5	0	*9*	6	105	39	(Modified)
	25	4	2	2	0	2	3	0	3	Source
	22	*3*	2	2	0	2	*2*	0	*2*	(Modified)
Guideline 13	91	11	1	1	0	5	5	0	9	Prototype
	89	11	1	1	0	5	5	0	9	(Modified)
	34	7	1	1	0	1	1	0	4	Source
	31	*6*	1	1	0	1	1	0	4	(Modified)
Guideline 14	79	8	2	2	0	4	2	0	4	Singleton
	76	8	1	1	0	4	2	0	4	(Modified)
	105	16	1	1	0	4	3	19	13	Source
	101	*15*	0	0	0	4	3	19	*12*	(Modified)

Design Principles	Metrics									Sample Systems
	LOCC	WOM	DIT	NOC	CFA	CMC	CAE	LCO	NOPM	
Guideline 15	18	3	1	1	0	0	1	0	3	TestDev
	18	2	1	1	0	0	1	0	2	(Modified)
	18	3	1	1	0	0	1	0	3	Source
	18	2	1	1	0	0	1	0	2	(Modified)
Heuristic 2.1	87	14	4	1	1	4	1	4	10	Memento
	93	16	4	1	0	5	1	4	*12*	(Modified)
	25	5	0	0	1	1	0	6	5	Source
	28	6	0	0	0	1	0	*8*	*6*	(Modified)
Heuristic 2.3	272	55	5	5	0	10	6	105	39	IteratorAj
	272	55	5	5	0	10	6	105	36	(Modified)
	32	10	0	0	0	0	0	0	10	Source
	26	*8*	0	0	0	0	0	0	6	(Modified)
Heuristic 2.5	127	18	3	3	0	8	6	19	13	Observer
	127	18	3	3	0	8	6	19	11	(Modified)
	23	4	0	0	0	1	0	0	4	Source
	23	4	0	0	0	1	0	0	3	(Modified)
Heuristic 2.6	1779	225	24	9	6	75	33	761	44	Spacewar
	1779	225	24	9	6	75	33	761	43	(Modified)
	20	4	1	1	0	1	0	0	4	Source
	20	4	1	1	0	1	0	0	3	(Modified)
Heuristic 2.7	75	14	3	3	0	5	4	0	10	VisitorAj
	75	14	3	3	0	7	4	0	10	(Modified)
	19	4	0	0	1	1	0	0	4	Source
	19	4	0	0	0	1	0	0	4	(Modified)
Heuristic 2.9	1779	225	24	9	6	75	33	761	44	Spacewar
	1773	223	24	9	6	75	33	*736*	44	(Modified)
	40	7	0	0	0	1	0	10	7	Source
	25	2	0	0	0	1	0	*0*	2	(Modified)
Heuristic 2.10	100	18	3	3	1	11	6	10	10	MediatorAj
	95	*17*	*2*	*2*	1	11	6	*12*	10	(Modified)
	25	5	0	0	0	1	0	2	5	Source
	28	5	0	0	0	2	0	0	5	(Modified)
Heuristic 3.2	26	5	0	0	0	1	1	0	3	HomeDev
	33	7	0	0	0	1	*2*	0	3	(Modified)
	43	7	0	0	0	3	0	0	7	Source
	48	9	0	0	0	*2*	0	0	*9*	(Modified)
Heuristic 3.7	70	8	3	3	1	4	2	0	7	FactoryMAj
	66	6	1	1	1	2	1	0	5	(Modified)
	36	6	0	0	0	2	0	0	6	Source
	24	3	0	0	0	1	0	0	3	(Modified)
Heuristic 3.8	114	19	2	2	0	8	3	3	11	ProxyAj
	89	14	1	1	0	8	2	2	8	(Modified)
	36	7	0	0	0	1	0	0	7	Source
	22	3	0	0	0	1	0	0	3	(Modified)
Heuristic 3.10	58	9	0	0	0	5	1	0	9	Façade
	52	7	0	0	0	4	*0*	0	7	(Modified)
	33	5	0	0	0	5	0	0	5	Source
	22	3	0	0	0	3	0	0	3	(Modified)
Heuristic 4.5	127	18	3	3	0	8	6	19	13	Observer
	125	18	3	3	0	7	6	*25*	13	(Modified)
	28	5	0	0	0	3	0	0	5	Source
	24	5	0	0	0	1	0	0	5	(Modified)
Heuristic 4.7	1779	225	24	9	6	75	33	761	44	Spacewar
	1777	225	24	9	*7*	75	33	761	44	(Modified)
	26	2	0	0	0	1	0	0	2	Source
	46	6	0	0	0	3	0	0	6	(Modified)
Heuristic 4.13	285	52	5	5	6	18	9	33	43	Telecom
	306	57	5	5	6	19	9	19	45	(Modified)
	27	5	0	0	0	2	0	0	5	Source
	22	*4*	0	0	0	2	0	0	*4*	(Modified)
Heuristic	1779	225	24	9	6	75	33	761	44	Spacewar

Design Principles	Metrics									Sample Systems
	LOCC	WOM	DIT	NOC	CFA	CMC	CAE	LCO	NOPM	
4.14	*1777*	225	24	9	6	*74*	33	*768*	44	(Modified)
	37	6	0	0	0	7	0	0	6	Source
	37	6	0	0	0	7	0	0	6	(Modified)
Heuristic 5.2	77	9	3	3	1	4	1	0	17	FactoryMAjDev
	73	12	3	3	1	4	1	0	17	(Modified)
	29	2	2	2	0	3	0	0	2	Source
	25	4	2	2	0	3	0	0	4	(Modified)
Heuristic 5.5	54	9	28	7	0	1	1	0	8	AnimalHierarchyDev
	49	*8*	21	6	0	1	*0*	0	8	(Modified)
	49	8	28	7	0	1	0	0	7	Source
	25	*4*	6	3	0	1	0	0	*3*	(Modified)
Heuristic 5.6	1779	225	24	9	6	75	33	761	44	Spacewar
	1739	215	24	9	6	*74*	33	*635*	36	(Modified)
	16	3	0	0	0	0	0	0	3	Source
	5	1	0	0	0	0	0	0	1	(Modified)
Heuristic 5.9	75	14	3	3	0	5	4	0	11	VisitorAjDev
	74	14	3	3	0	5	4	0	11	(Modified)
	34	5	2	2	0	2	0	0	5	Source
	44	9	2	2	0	2	0	*2*	7	(Modified)
Heuristic 5.10	75	14	3	3	0	5	4	0	10	VisitorAj
	70	12	3	3	0	5	4	0	8	(Modified)
	42	7	2	2	0	2	0	2	7	Source
	49	10	2	2	0	2	0	*6*	8	(Modified)
Heuristic 5.13	70	8	3	3	1	4	2	0	7	FactoryMAj
	57	8	*5*	*5*	*0*	*6*	*0*	0	8	(Modified)
	44	5	3	3	0	4	0	0	5	Source
	42	8	3	3	0	4	0	0	8	(Modified)
Heuristic 5.15	70	8	3	3	1	4	2	0	7	FactoryMDevAj
	60	6	1	1	1	2	1	0	5	(Modified)
	28	4	3	3	0	3	0	0	5	Source
	27	3	0	0	0	1	0	0	3	(Modified)
Heuristic 5.17	17	3	1	1	0	2	0	0	3	DogDev
	14	2	1	1	0	2	0	0	2	(Modified)
	17	3	1	1	0	2	0	0	3	Source
	14	2	1	1	0	2	0	0	2	(Modified)
Heuristic 9.2	87	14	4	1	1	4	1	4	10	Memento
	93	16	4	1	0	5	1	4	12	(Modified)
	54	6	0	0	2	4	0	0	2	Source
	55	6	0	0	1	4	0	0	2	(Modified)
Duplicate Code	79	8	2	2	0	4	2	0	4	Singleton
	76	*9*	2	2	0	5	*3*	0	4	(Modified)
	30	5	0	0	0	2	0	0	5	Source
	25	*4*	0	0	0	3	0	0	*4*	(Modified)
Long Method	182	29	1	1	1	8	5	28	21	Composite
	185	30	1	1	1	*9*	5	28	21	(Modified)
	24	2	0	0	0	1	0	0	2	Source
	26	3	0	0	0	1	0	0	2	(Modified)
Long Parameter List	100	18	3	3	1	11	6	10	10	MediatorAj
	100	18	3	3	1	*12*	6	10	10	(Modified)
	18	3	0	0	0	2	0	0	3	Source
	18	3	0	0	0	*3*	0	0	3	(Modified)
Feature Envy	70	8	3	3	1	4	2	0	7	factoryMAj
	62	8	3	3	1	3	*1*	0	7	(Modified)
	15	3	0	0	0	2	0	0	3	Source
	15	3	0	0	0	1	0	0	3	(Modified)
Freeloader	79	8	2	2	0	4	2	0	4	Singleton
	74	8	2	2	0	4	*1*	0	4	(Modified)
	14	2	0	0	0	2	0	0	2	Source
	11	2	0	0	0	*1*	0	0	2	(Modified)
Speculative	182	29	1	1	1	8	5	28	21	Composite

Design Principles	Metrics									Sample Systems
	LOCC	WOM	DIT	NOC	CFA	CMC	CAE	LCO	NOPM	
Generality	179	28	1	1	1	8	5	28	20	(Modified)
	14	3	0	0	0	1	0	0	3	Source
	11	2	0	0	0	1	0	0	2	(Modified)
Message Chains	27	8	0	0	0	0	0	0	8	StringChainDev
	33	8	0	0	0	0	0	0	8	(Modified)
	36	7	0	0	0	11	0	0	7	Source
	41	7	0	0	0	11	0	0	7	(Modified)
Anonymous Pointcut Definitions	28	5	0	0	0	1	2	0	3	DecoratorAj
	29	5	0	0	0	1	2	0	3	(Modified)
	50	10	0	0	0	2	1	3	8	Source
	51	10	0	0	0	2	1	3	8	(Modified)
Lazy Aspects	97	9	1	1	0	4	1	0	6	Strategy
	95	9	0	0	0	1	1	0	6	(Modified)
	14	3	0	0	0	2	0	0	3	Source
	13	3	0	0	0	0	0	0	3	(Modified)
Aspect Feature Envy	1779	225	24	9	6	75	33	761	44	Spacewar
	1779	225	24	9	6	75	33	761	44	(Modified)
	1779	225	24	9	6	75	33	761	44	Source
	1779	225	24	9	6	75	33	761	44	(Modified)
Abstract Method Introductions	75	14	3	3	0	5	4	0	10	VisitorAj
	77	14	3	3	0	5	4	0	10	(Modified)
	285	52	5	5	6	18	9	33	43	Source
	284	51	5	5	6	18	8	33	42	(Modified)
Large Aspects	279	51	5	5	6	14	9	33	43	TelecomDev
	282	51	5	5	6	14	10	33	43	(Modified)
	40	8	0	0	0	2	3	0	6	Source
	42	8	0	0	0	2	4	0	6	(Modified)

Because almost all systems are close systems, the modified systems are tested by rerunning to check their behavioural changes. All systems give the same running results. Side-effects after adjusting design are also examined. Details of side-effects after correcting each principle violations are as follows:

- Accessing field indirectly from accessor methods and changing the field access control to private for Guideline 4 can remove some types of violation of Heuristic 2.7 (directly accessing other's fields).
- Replacing duplicate methods with methods of advices for Guideline 6 can remove the violation of Guideline 6 in another unit.
- Changing access control of a field from non-private to private and adding accessor methods for Heuristic 2.1 can remove the violation of Heuristic 2.7 for units which access that field.
- Removing unused public methods for Heuristic 2.3 can remove the violation of Speculative Generality.

- Changing access control of inner-used methods to non-public for Heuristic 2.3 can remove the violation of Heuristic 2.5.
- Changing access control of inner-used methods to non-public for Heuristic 2.5 can remove the violation of Heuristic 2.3.
- Accessing field indirectly from accessor methods and changing the field access control to private for Heuristic 2.7 can remove the violation of Heuristic 2.1 in another unit.
- Moving disjoint set of methods and fields to another unit for Heuristic 2.10 can accidentally cause the violation of Heuristic 2.10 in that unit.
- Moving main method that uses the contained object for Heuristic 4.7 can unexpectedly cause the violation of Heuristic 2.10 in the given unit.
- Extracting a unit to cope with many-to-many relationship for Heuristic 4.14 can remove the violation of Heuristic 2.10.
- Moving fields from subunits to their super unit and defining the fields as protected for Heuristic 5.9 or Heuristic 5.10 can cause the violation of Heuristic 2.1 in the super unit.
- Adding unused concrete behaviors to an abstract method for Abstract Method Introductions can cause the violation of Guideline 12.

Table 7.4: Design principles for aspect-oriented software maintainability

Design Principles for Aspect-Oriented Software Maintainability		
Proposed guidelines	**Design heuristics**	**Bad smells**
Guideline 2, Guideline 3, Guideline 5, Guideline 8, Guideline 9, Guideline 10, Guideline 11, Guideline 12, Guideline 13, Guideline 14, Guideline 15	Heuristic 2.3, Heuristic 2.5, Heuristic 2.6, 2.9, , Heuristic 3.7, Heuristic 3.8, Heuristic 3.10, Heuristic 5.5, Heuristic 5.6, Heuristic 5.15, Heuristic 5.17	Feature Envy, Freeloader, Speculative Generality, Lazy Aspects, Aspect Feature Envy

Finally, the design principles which affect the systems in positive ways following the assumption with no negative metric results or bad side-effects are selected to construct aspect-oriented software maintainability metrics. Principles which present no differences on all metrics are taken as well because more principles can help to reveal

more flaws which are detected in the aspect-oriented software maintainability assessment. Table 7.4 lists 27 design principles which are passed from the selection process to construct the maintainability metrics.

7.2 Metrics for Aspect-Oriented Software Maintainability

This section describes the metrics for aspect-oriented software maintainability. Following the idea of FS quality model, the metrics for aspect-oriented software maintainability are constructed using a set of qualified principles from Section 6.1.

Metrics for maintainability are defined as follows:

$$M(Ui) = \sum_{m=1}^{n} (Pm \times Wm) \quad (1)$$

where M(*Ui*) is the *degree of maintainability of a unit* (a class, an abstract class, an aspect, an abstract aspect, or an interface), *Ui* is the *i*th unit, *Pm* is equal to -1 if the unit violates the given *m*th principle; otherwise it is equal to 0, *Wm* is a weight value for each design principle (if any); otherwise it is equal to 1, *m* is equal to 1,...,*n* where *n* is the total number of all principles that this unit can be applied. The suggested weight values for all design principles are described in Section 7.3.

$$M(S) = \sum_{i=1}^{t} M\ (Ui\ ) \quad (2)$$

where M(*S*) is the *degree of maintainability of a system* that can be either an aspect-oriented system or an object-oriented system, *Ui* is *i*th unit, t is the number of all units in this system.

$$AM(S) = \frac{\sum_{i=1}^{t} M(Ui)}{t} \quad (3)$$

where AM(*S*) is the *average degree of maintainability per unit.*

For example, the system *S* contains 2 classes and 2 aspects. The class *C1* violates the *Feature Envy* bad smell. The aspect *A1* violates the *Principle 4*. The class *C2* and the aspect *A2* violate none. So, each metric is calculated as follows (assumes that all principle violations are weighted as 1):

M(*C1*) = (-1x1) = -1,

M(*C2*) = 0,

M(*A1*) = (-1x1) = -1,

M(*A2*) = 0,

M(*S*) = -1 x -1 = -2, and

AM(*S*) = (-2)/4 = -0.5

For the *unit level*, M(*C1*) and M(*A1*) = -1 mean the class *C1* and the aspect *A1* violate one rule each. The class *C2* and the aspect *A2* violate none of design principles. For the *system level*, M(*S*) = -2 means this system has totally two rule breaks. For the *system level comparison*, AM(*S*) = -0.5 means this system breaks average one rule per unit. The average AM of fifty systems in Section 7.1 is between -0.70 and -0.51, thus the maintainability of this system is evaluated to be good ($\geq$ -0.51 is good, between -0.70 and -0.51 is fair, and $\leq$ -0.70 is poor).

7.3 Suggested Weight Values for Aspect-Oriented Software Maintainability Metrics

This section defines suggested weight values for the aspect-oriented software maintainability metrics. FCM quality models along with the ISO 9126 definitions are used to estimate the weight values.

ISO 9126 defines software maintainability as "a set of attributes that are related to the effort needed to make specified modifications". It comprises 5 sub-factors which are: *analyzability*, attributes of software that relate to the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified; *changeability*, attributes of software that relate to the effort needed for modification, fault removal or for environmental change; *stability*, attributes of software that relate to the risk of unexpected effect of modifications; *testability*: attributes of software that relate to the effort needed for validating the modified software; and *compliance*, attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions [1].

Three major sub-characteristics; that is, *analyzability* (A), *changeability* (C), and *stability* (S), are selected and are sought for their connections to the design principles

by adapting an idea of structural complexity metrics for the object-oriented software maintainability proposed by Kiewkanya [47]. Kiewkanya combines 8 types of relationships: *generalization*, *aggregation*, *composition*, *common association*, *association class*, *dependency*, *realization*, and class *complexity* to define metrics for two sub-characteristics of the object-oriented design maintainability, that is, understandability and modifiability. For conciseness, only two types of relationships: *generalization* and *dependency*, which are enough to cover all principles, are quoted.

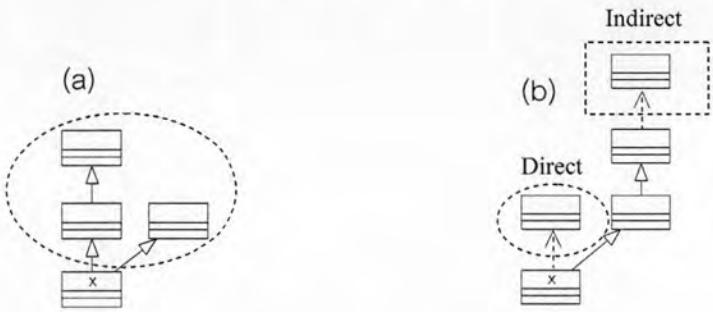

Figure 7.1: The effects of generalization (a) and dependency (b) on analyzability [47]

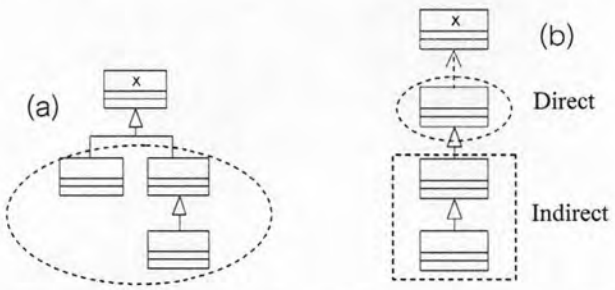

Figure 7.2: The effects of generalization (a) and dependency (b) on changeability [47]

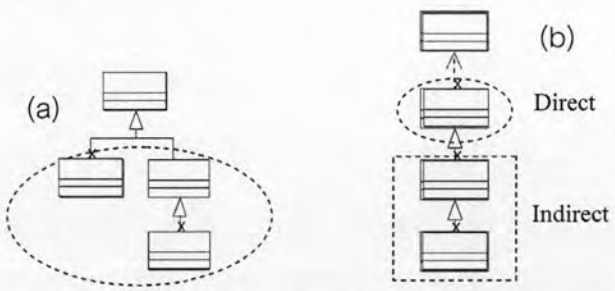

Figure 7.3: The effects of generalization (a) and dependency (b) on stability

This work also proposes the effects of relationships on *stability* and the effects of complexity on *analyzability* to complete three maintainability sub-characteristics. *Complexity* is classified as structural complexity and cognitive complexity. *Structural complexity* involves the number of modules in a program [48]. *Cognitive complexity* is

defined in terms of readability and understandability of the software by the human [49]. Understandability and modifiability are comparable to analyzability and changeability respectively.

Figure 7.1 to Figure 7.3 successively shows the effects of generalization and dependency on analyzability, changeability, and stability. For *analyzability*, analyzing class X is required the comprehension on all classes in dashed circles: class X's ancestors (generalization relationship) and the classes that class X depends on them (dependency). For *changeability*, changing class X may require the alteration to all classes in dashed circles: class X's descendants (generalization relationship) and the classes that depend on class X (dependency). For *stability*, changing a class that class Xs depends on may require alteration to all classes Xs in dashed circles: class Xs which are descendant classes (generalization relationship) and the class Xs that depend on other classes (dependency).

Table 7.5: The effects of design principles on maintainability sub-characteristics

Design Principles	Types of Relationships						Complexity		Maintainability Sub-characteristics		
	Gen (T)	Call (H)	Call (T)	Derive (H)	Usg (H)	Usg (T)	Structural	Cognitive	A	C	S
Guideline 11, Heuristic 5.5, 5.15	X								X		X
Heuristic 3.10		X	X						X	X	X
Heuristic 2.9				X						X	
Guideline 9, Aspect Feature Envy					X					X	
Guideline 8, 10, Heuristic 3.7, Feature Envy						X			X		X
Guideline 3, 5, 12, 13, 14, Heuristic 2.3, 2.5, 2.6, 3.8, 5.6, 5.17, Freeloader, Speculative Generality							X		X		
Guideline 2, 15, Lazy Aspects								X	X		

Design principles for maintainability are considered their influences on analyzability (A), changeability (C), and stability (S) from exploring whether they relate to generalization, dependency, structural complexity, or cognitive complexity. *Dependency*

relationships are divided into five types: *call dependency* (a method on one class calls an operation on another class), *derivation dependency* (one element can be computed from another element), and *usage dependency* (one element requires the presence of another element for its correct functioning which includes call, instantiation, parameter, send, but open to other kinds) [50]. Table 7.5 reveals the relationships between each design principle and the maintainability sub-characteristics. Design principles which are related to one, two, three maintainability sub-characteristics should be weighted (*Wm* values in Section 7.2) as 1, 2, and 3 respectively.