CHAPTER IV

DEFINING DESIGN PRINCIPLE VIOLATION CHECK DEFINITIONS

Chapter IV defines design principle violation check definitions and their validations. These concrete definitions can be applied to construct an automatic tool for detecting each design flaw separately or can be combined to construct aspect-oriented software maintainability metrics in Chapter VII.

Design guidelines in Section 3.2 with design heuristics and bad smells which are not sorted out in Section 3.3 are combined and are extracted their concrete violation check definitions in the form of *Boolean expressions*. The violation check definitions can be applied to detect design flaws in both aspect-oriented part and object-oriented part of software (specified in each design principle) that are *class units* (C), *abstract class units* (AC), *aspect units* (A), *abstract aspect units* (AA), and *interface units* (I).

For example, design principle violation check definition for *Guideline 2: Do suitably use wildcard operators for each captured join point to increase the pointcut readability and to decrease the unintended join point capture* is defined as:

Consider a unit U. Let *P* be the set of pointcuts of U. Let *CJ* be the set of captured join points of a pointcut *p* where $p \in P$. Let $ADWCJ_{pcj}$ be the set of asterisk and dotted line wildcard operators of a captured join point *cj* of *p* where $cj \in CJ$. The principle is violated if $\exists\, p\ \exists\, cj\ [\ |ADWCJ_{pcj}|>2]$. The number two is taken from the average number of the wildcard operators used in each capture join point of fifty sample systems in Section 8.1.

It means a unit (a given aspect unit or a given abstract aspect unit) is judged to be a violation unit if some captured join points in that unit is defined using more than two asterisks and dotted lines.

4.1 Design Principle Violation Check Definitions for Design Guidelines

Fifteen Boolean violation check definitions are described below. *Additional conditions* are additional details of measurement for each principle. Some design patterns [37] are considered as well to prevent the measurement from type II errors; such as in the *Heuristics 5.15: Do not turn objects of a class into derived classes of the class. Be very suspicious of any derived class for which there is only one instance,* Singleton classes which always have one instance should be skipped. *Design patterns* are general repeatable solutions to common problems in software design. They are not ready-made designs that can be transformed directly into code, but they are applied as templates to solve the similar problems in different situations [20]. *Type II errors* are the errors of accepting null hypotheses when the alternative hypotheses are true state of nature [38].

Guideline 1: Don't make the core concern unused by replacing the entire of its context with the crosscutting concern.

Violation check expressions:

Consider a unit U.

Let *AAVNCP* be the set of around advices of U which did not call *proceed()*.

The principle is violated if

$$AAVNCP \neq \emptyset$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 2: Do suitably use wildcard operators for each captured join point to increase the pointcut readability and to decrease the unintended join point capture.

Violation check expressions:

Consider a unit U.

Let *P* be the set of pointcuts of U.

Let *CJ* be the set of captured join points of a pointcut *p* where $p \in P$.

Let $ADWCJ_{pcj}$ be the set of asterisk and dotted line wildcard operators of a captured join point *cj* of *p* where $cj \in CJ$.

The principle is violated if

$$\exists p \; \exists cj \; [\; |ADWCJ_{pcj}| > 2]$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 3: Do define a concise pointcut for capturing sibling join points to decrease the accidental join point miss when a new class is added to the hierarchy.

Violation check expressions:

Consider a unit U.

Let P be the set of pointcuts of U.

Let ASU_p be the set of all sibling units which their similar methods are captured by a pointcut p where $p \in P$.

The principle is violated if

$$\exists p \; [ASU_p \neq \emptyset]$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 4: Don't use an aspect privilege to leak the secret of other types.

Violation check expressions:

Consider a privilege unit PU.

Let FT be the set of functions (methods/ intertype-methods/ advices) of PU.

Let $PVMOU_{ft}$ be the set of private members (fields/ inter-type fields/ methods/ inter-type methods) of other units except PU which are used by ft where $tf \in FT$.

The principle is violated if

$$\exists ft [\; PVMOU_{ft} \neq \emptyset]$$

Note: Heuristic 2.7 = T, Guideline 4 = F.

±23862506

Units can be applied: Aspects/ Abstract Aspects

Guideline 5: Do create an advantageous aspect which crosscuts at least two points in the core concern.

Violation check expressions:

Consider a unit U.

Let P be the set of pointcuts of U.

Let J_p be the set of corresponding join points of a pointcut p where $p \in P$.

Let AV be the set of advices of U. Let $WESD$ be the set of warnings, errors, and soft declarations of U.

Let AP be the set of abstract pointcuts of U.

Let PD be the set of parent declarations of U.

Let PDD be the set of precedence declarations of U.

Let F be the set of fields of U.

Let IF be the set of inter-type fields of U.

Let IM be the set of inter-type methods of U.

Let M be the set of methods of U.

Let $max(N)$ be the function to determine the max number of N where N is the set of counting numbers.

The principle is violated if

$$(F \cup IF \cup M \cup IM \cup P \cup AV \neq \emptyset) \wedge max\,(|J_p|, |AV|, |WESD|) + |AP| + |PD| + |PDD| + |IF| + |IM| < 2$$

Note: Lazy Aspects = T, Guideline 5 = F.

Units can be applied: Aspects/ Abstract Aspects

Guideline 6: Do localize all crosscutting contexts within a set of corresponding aspects.

Violation check expressions:

Consider a unit U.

Let *M* be the set of methods of U.

Let *IM* be the set of inter-type methods of U.

Let *OFT* be the set of other functions (methods, inter-type methods, and advices) of all units except the given method or the given inter-type method *mim* where $mim \in (M \cup IM)$.

Let ASF_{mim} be the set of adjacent statements of *mim*.

Let ASF_{oft} be the set of adjacent statements of *oft* where $oft \in OFT$.

The principle is violated if

$$\exists\, mim \exists\, oft\ (ASF_{mim} \cap ASF_{oft} \neq \emptyset)$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Guideline 7: Do try to separate one set of aspects to be independent of another.

Violation check expressions:

Consider a unit U.

Let *M* be the set of methods of U.

Let *IM* be the set of inter-type methods of U.

Let *AV* be the set of advices of U.

Let *MIMO* be the set of methods and inter-type methods of other aspect units except U.

Let $FCMIM_{mimo}$ be the set of functions (methods/ inter-type methods/ advices) of other units calling to a method or an inter-type methods *mimo* where $mimo \in MIMO$.

The principle is violated if

$$\exists\, mimo[\ (M \cup IM \cup AV) \cap FCMIM_{mimo} \neq \emptyset]$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 8: Do minimize a number of aspects which crosscut to the same concern.

Violation check expressions:

Consider a unit U.

Let P be the set of pointcuts of U.

Let J_p be the set of corresponding join points of a pointcut p where $p \in P$.

Let PO be the set of pointcuts of other aspect units except U.

Let J_{po} be the set of corresponding join points of a pointcut po where $po \in PO$.

This principle is violated if

$$\exists p \exists po \, (J_p \cap J_{po} \neq \emptyset)$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 9: Do keep scattering behaviors that represent the same concern in one set of corresponding aspects.

Violation check expressions:

Consider a unit U.

Let M be the set of methods of U.

Let UCM_m be the set of units calling to a method m where $m \in M$.

Let AC be the set of all classes in the system.

Let AAC be the set of all abstract classes in the system.

Let PR be the set of parent units of U.

Let CD be the set of children units of U.

This principle is violated if

$$\exists m \, (|UCM_m - AC - AAC - PR - CD - \{U\}| = 1)$$

Additional Condition: Enclosing aspect calling to its enclosed class are not included in $UCMT_m$.

Units can be applied: Classes/ Aspects

Guideline 10: Don't let a base class be dependence on its crosscutting aspects.

Violation check expressions:

Consider a unit U.

Let *CU* be the set of crosscutting units which crosscuts U.

Let *M* be the set of methods of *CU.*

Let *IM* be the set of inter-type methods of *CU.*

Let UCM_m be the set of units calling to a method *m* where $m \in M$.

Let $UCIM_{im}$ be the set of units calling to an inter-type method *im* where $im \in IM$.

This principle is violated if

$$\exists m\ (U \in (UCM_m)) \vee \exists im\ (U \in (UCIM_{im}))$$

Units can be applied: Classes/ Abstract Classes

Guideline 11: Don't use an inheritance relationship to model a crosscutting relationship because an aspect is not a special type of the class that it crosscuts.

Violation check expressions:

Consider a unit U.

Let *PR* be the set of parent units of U.

Let *AC* be the set of all classes in the system.

Let *AAC* be the set of all abstract classes in the system.

This principle is violated if

$$PR \subseteq (AC \cup AAC)$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 12: Don't introduce methods to all subclasses to override a concrete method of their abstract superclass.

Violation check expressions:

Consider a unit U.

Let *SIM* be the set of the similar signature inter-type methods of U.

Let *SU* be the set of sibling units of the same abstract superclass which all sibling units are crosscut by *sim* where $sim \in SIM$.

Let IHM_{su} be the set of concrete methods which *su* are inherited from its abstract superclass where $su \in SU$.

This principle is violated if

$$\exists\, sim\, \forall\, su\ (sim \in IHM_{su})$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 13: Don't define methods of all subaspects to override a concrete method of their abstract aspect.

Violation check expressions:

Consider a unit U.

Let *IHM* be the set of concrete methods of U which are inherited from its abstract aspect. Let *M* be the set of methods of U.

Let *SU* be the set of all sibling units of the same abstract aspect of U.

Let M_{su} be the set of methods of *su* where $su \in SU$.

This principle is violated if

$$\forall\, su\ ((IHM \cap M) \cap M_{su} \neq \varnothing)$$

Units can be applied: Aspects/ Abstract Aspects

Guideline 14: Do extract an abstract aspect if the same crosscutting concern has two or more variations.

Violation check expressions:

Consider a unit U.

Let *CD* be the set of children units of U.

This principle is violated if

$$|CD| \leq 1$$

Units can be applied: Abstract Aspects

Guideline 15: Don't introduce a method which has the same signature as an inherited method to the realized interface.

Violation check expressions:

Consider a unit U.

Let *CIM* be the set of concrete inter-type methods of U.

Let *I* be the set of interfaces which are crosscut by *cim* where $cim \in CIM$.

Let *IU* be the set of units implementing interface *i* where $i \in I$.

Let IHM_{iu} be the set of concrete methods which *iu* are inherited from its super-units where $iu \in IU$.

This principle is violated if

$$\exists iu \, (CIM \cap IHM_{iu} \neq \emptyset)$$

Units can be applied: Aspects/ Abstract Aspects

4.2 Design Principle Violation Check Definitions for Existing Design Heuristics and Bad Smells

Thirty-six violation check expressions for the rest design heuristics and bad smells are shown below. From all definitions, four definitions which are *Anonymous*

Pointcut Definitions, *Lazy Aspects*, *Aspect Feature Envy*, and *Abstract Method Introductions*, are excerpted from [17] to fulfill the design principles. These definitions are selected to define aspect-oriented software maintainability metrics in Chapter VI.

Heuristic 2.1: All data should be hidden within its class.

Violation check expressions:

Consider a unit U.

Let *F* be the set of fields of U.

Let *IF* be the set of inter-type fields of U.

Let *PVM* be the set of private members of U.

This principle is violated if

$$(F \cup IF) - PVM \neq \emptyset$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Heuristic 2.3: Minimize the number of messages in the protocol of a class.

Violation check expressions:

Consider a unit U.

Let *M* be the set of methods of U.

Let *IM* be the set of inter-type methods of U.

Let *DPU* be the set of methods and inter-type methods of U which are used by non-subclass units in different packages.

Let *PBM* be the set of public members of U.

This principle is violated if

$$|((M \cup IM) - DPU) \cap PBM| > 7$$

Note: The number seven is taken from the number of average NOM (Number of methods of the measured class) [39].

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Heuristic 2.5: Do not put implementation details such as common-code private functions into the public interface of the class.

Violation check expressions:

Consider a unit U.

Let M be the set of concrete methods of U.

Let IM be the set of concrete inter-type methods of U.

Let $FCMIM_{mim}$ be the set of functions (methods/ inter-type methods/ advices) in other units calling to a method or an inter-type methods mim (except the nested classes of U and descendant classes of U) where $mim \in (M \cup IM)$.

Let PBM be the set of public members of U.

The principle is violated if

$$\exists\, mim\ [\ (FCMIM_{mim} = \emptyset) \wedge (mim \in PBM)]$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Heuristic 2.6: Do not clutter the public interface of a class with items that users of that class are not able to use or are not interested in using it.

Violation check expressions:

Consider a unit U.

Let C be the set of constructors of U.

Let PBM be the set of public members of U.

This principle is violated if

$$C \cap PBM \neq \emptyset$$

Units can be applied: Abstract Classes/ Abstract Aspects

Heuristic 2.7: Classes should only exhibit nil or export coupling with other classes, that is, a class should only use operations in the public interface of another class or have nothing to do with that class.

Violation check expressions:

Consider a unit U.

Let *FT* be the set of functions (methods/ intertype-methods/ advices) of U.

Let $NPBMBOU_{ft}$ be the set of members (fields/ inter-type fields/ non-public methods/ non-public inter-type methods/ non-public pointcuts) of other units except U which are used by *ft* where $ft \in FT$.

The principle is violated if

$$\exists ft[\, NPBMBOU_{ft} \neq \emptyset]$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Heuristic 2.9: Keep related data and behavior in one place.

Violation check expressions:

Consider a unit U.

Let *GM* be the set of get methods of U.

Let *GIM* be the set of get inter-type methods of U.

This principle is violated if

$$|GM \cup GIM| > 4$$

Note: The number four is taken from half of average NOM [39]. Get methods are methods which their names begin with 'get' and are followed by field names.

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects/ Interfaces

Heuristic 2.10: Spin off non-related information into another class (i.e., non-communicating behavior).

Violation check expressions:

Consider a unit U.

Let *NSF* be the set of non-static fields of U.

Let *MC* be the set of methods or constructors of U.

Let $MCUF_{nsfx}$ be the set of methods or constructors using nsf_x where $nsf_x \in NSF$.

Let $MCUF_{nsfy}$ be the set of methods or constructors using nsf_y where $nsf_y \in (NSF-\{nsf_x\})$.

Let $MCUMC_{\mathrm{mcufnsfx}}$ be the set of methods or constructors using *mcufnsfx* where $mcufnsfx \in MCUMC_{mcufnsfx}$.

Let $MCUMC_{\mathrm{mcufnsfy}}$ be the set of methods or constructors using *mcufnsfy* where $mcufnsfy \in MCUMC_{mcufnsfy}$.

This principle is violated if

$$((|NSF|> 1) \wedge (|MC|> 1)) \wedge (\exists nsfx \exists nsfy ((MCUF_{nsfx} \neq \emptyset) \wedge (MCUF_{nsfy} \neq \emptyset) \wedge (MCUF_{nsfx} \cap MCUF_{nsfy} = \emptyset)) \wedge (\forall mcufnsfx \forall mcufnsfy ((mcufnsfy \notin MCUMC_{\mathrm{mcufnsfx}}) \wedge (mcufnsfx \notin MCUMC_{\mathrm{mcufnsfy}}) \wedge (MCUMC_{\mathrm{mcufnsfx}} \cap MCUMC_{\mathrm{mcufnsfy}} = \emptyset))$$

Units can be applied: Classes/ Abstract Classes

Heuristic 3.2: Do not create god classes/ objects in your system. Be very suspicious of a class whose name contains Driver, Manager, System, or Subsystem.

Violation check expressions:

Consider a unit U.

Let *UDMSSS* be the set of units of the system that contain Driver, Manager, System, or SubSystem at the end of their names.

This principle is violated if

$$U \in UDMSSS$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects/ Interfaces

Heuristic 3.7: Eliminate irrelevant classes from your design.

Violation check expressions:

Consider a unit U.

Let *FT* be the set of functions (methods/constructors) of U.

Let C be the set of constructors of U.

Let *GM* be the set of get methods of U which each get method has only one statement returning a field in get method's name.

Let *SM* be the set of set methods of U which each set method has only one statement setting a field in set method's name.

Let *PTM* be the set of print type methods of U which each print type method has only one statement printing a field in print type method's name.

Let *AEX* be the set of exception classes in the system.

This principle is violated if

$$(FT \neq \emptyset) \wedge (FT - (C \cup GM \cup SM \cup PTM) = \emptyset) \wedge (U \notin AEX)$$

Units can be applied: Classes/ Abstract Classes

Heuristic 3.8: Eliminate classes that are outside the system.

Violation check expressions:

Consider a unit U.

Let *FT* be the set of members (methods/ inter-type methods/ constructors) of U.

Let UCM_{ft} be the set of units calling to a function *ft* where $ft \in FT$.

Let *P* be the set of pointcuts of U.

Let J_p be the set of corresponding join points of a pointcut *p* where $p \in P$.

This principle is violated if

$$(FT \neq \emptyset) \wedge \forall ft\,(UCM_{ft} - \{U\} = \emptyset) \wedge \forall p\,(J_p = \emptyset)$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Heuristic 3.10: Agent classes are often placed in the analysis model of an application. During design time, many agents are found to be irrelevant and should be removed.

Violation check expressions:

Consider a unit U.

Let M be the set of methods of U.

Let ST_m be the set of statements of a method m where $m \in M$.

Let MSG_{stm} be the set of messages contained in a statement stm of a method m where $stm \in ST_m$.

Let ADC be the set of adapter classes in the system.

Let PX be the set of proxy classes in the system.

This principle is violated if

$$\forall m((|ST_m| = 1) \wedge \forall st\,(|MSG_{stm}| = 1)) \wedge (U \notin ADC) \wedge (U \notin PX)$$

Units can be applied: Classes

Heuristic 4.5: If a class contains objects of another class, then the containing class should be sending messages to the contained objects, that is, the containment relationship should always imply a use relationship.

Violation check expressions:

Consider a unit U.

Let CT be the set of class type of field or inter-type field types which are contained by U.

Let M be the set of methods of CT.

Let IM be the set of inter-type methods of CT.

Let *F* be the set of fields of *CT*.

Let *IF* be the set of fields of *CT*.

Let $UCMIM_{mim}$ be the set of units calling to a method or an inter-type method *mim* where $mim \in (M \cup IM)$.

Let $UUFIF_{fif}$ be the set of units using a field or an inter-type field *fif* where $fif \in (F \cup IF)$.

This principle is violated if

$$\forall\, mim\ (U \notin UCMIM_{mim}) \wedge \forall\, fif\ (U \notin UUFIF_{fif})$$

Units can be applied: Classes / Aspects

Heuristic 4.7: Classes should not contain more objects than a developer can fit in his or her short-term memory. A favorite value for this number is six.

Violation check expressions:

Consider a unit U.

Let *F* be the set of fields of U.

Let *IF* be the set of fields of U.

This principle is violated if

$$|F \cup IF| > 6$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects/ Interfaces

Heuristics 4.13: A class must know what it contains, but it should never know who contains it.

Violation check expressions:

Consider a unit U.

Let *UC* be the set of units which contain U type fields or U type inter-type fields.

Let *M* be the set of methods of *UC*. Let *IM* be the set of inter-type methods of *UC*.

Let *F* be the set of fields of *UC*. Let *IF* be the set of fields of *UC*.

Let $UCMIM_{mim}$ be the set of units calling to a method or an inter-type method *mim* where $mim \in (M \cup IM)$.

Let $UUFIF_{fif}$ be the set of units using a field or an inter-type field *fif* where $fif \in (F \cup IF)$.

This principle is violated if

$$\exists\, mim\ (U \in UCMIM_{mim}) \vee \exists\, fif\ (U \in UUFIF_{fif})$$

Units can be applied: Classes

Heuristics 4.14: Objects that share lexical scope-those contained in the same containing class should not have uses relationships between them.

Violation check expressions:

Consider a unit U.

Let *UC* be the set of units which contain U type fields or U type inter-type fields.

Let *CT* be the set of class type of field or inter-type field types which are also contained by *UC*.

Let *M* be the set of methods of *CT*.

Let *IM* be the set of inter-type methods of *CT*.

Let *F* be the set of fields of *CT*.

Let *IF* be the set of fields of *CT*.

Let $UCMIM_{mim}$ be the set of units calling to a method or an inter-type method *mim* where $mim \in (M \cup IM)$.

Let $UUFIF_{fif}$ be the set of units using a field or an inter-type field *fif* where $fif \in (F \cup IF)$.

This principle is violated if

$$\exists\, mim\ (U \in UCMIM_{mim}) \vee \exists\, fif\ (U \in UUFIF_{fif})$$

Units can be applied: Classes

Heuristic 5.2: Derived classes must have knowledge of their base class by definition, but base classes should not know anything about their derived classes.

Violation check expressions:

Consider a unit U.

Let *FT* be the set of functions (methods/ inter-type methods/ advices) of U.

Let $TPISF_{ft}$ be the set of types defined in predicates of if statement of a function *ft* where $ft \in FT$.

Let *CD* be the set of children units of U.

This principle is violated if

$$\exists ft\ (CD \cap TPISF_{ft} \neq \emptyset) \vee (U \in TPISF_{ft})$$

Units can be applied: Classes/ Abstract Classes/ Abstract Aspects

Heuristic 5.5: In practice, inheritance hierarchies should be no deeper than an average person can keep in his or her short-term memory. A popular value for this depth is six.

Violation check expressions:

Consider a unit U.

Let *dit*(U) be the function to determine depth of inheritance of U.

This principle is violated if

$$dit(U) > 6$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects/ Interfaces

Heuristic 5.6: All abstract classes must be base classes.

Violation check expressions:

Consider a unit U.

Let *RU* be the set of realization units of U.

Let *CD* be the set of children units of U.

This principle is violated if

$$RU \cup CD = \emptyset$$

Units can be applied: Abstract Classes/ Abstract Aspects/ Interfaces

Heuristics 5.9: If two or more classes share only common data (no common behavior), then that common data should be placed in a class that will be contained by each sharing class.

Violation check expressions:

Consider a unit U.

Let *SU* be the set of sibling units of U.

Let *F* be the set of fields of U.

Let F_{su} be the set of fields of *su* where $su \in SU$.

This principle is violated if

$$\forall su\ (F \cap F_{su} \neq \emptyset)$$

Note: Heuristic 5.10 = T, Heuristic 5.9 = F.

Units can be applied: Classes/ Aspects

Heuristics 5.10: If two or more classes have common data and behavior, then those classes should each inherit from a common base class that captures those data and methods.

Violation check expressions:

Consider a unit U.

Let *SU* be the set of sibling units of U.

Let F be the set of fields of U.

Let F_{su} be the set of fields of su where $su \in SU$.

Let M be the set of methods of U.

Let M_{su} be the set of methods of su where $su \in SU$.

Let ST_m be the set of statements in a method m where $m \in (M \cap M_{su})$.

Let ST_{msu} be the set of statements in a method msu where $msu \in (M_{su} \cap M)$.

This principle is violated if

$$(\forall su\ (F \cap F_{su} \neq \emptyset) \wedge (M \cap M_{su} \neq \emptyset) \wedge (\forall m\ \forall msu\ (ST_m \subseteq ST_{msu} \wedge ST_{msu} \subseteq ST_m))$$

Units can be applied: Classes/ Aspects

Heuristics 5.13: Explicit case analysis on the value of an attribute is often an error. The class should be decomposed into an inheritance hierarchy, where each value of the attribute is transformed into a derived class.

Violation check expressions:

Consider a unit U.

Let FT be the set of functions (methods/ inter-type methods/ advices) of U.

Let $SPPTF_{ft}$ be the set of switch statements having primitive type fields in their predicates of a function ft where $ft \in FT$.

This principle is violated if

$$\exists ft\ (SPPTF_{ft} \neq \emptyset)$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Heuristics 5.15: Do not turn objects of a class into derived classes of the class. Be very suspicious of any derived class for which there is only one instance.

Violation check expressions:

Consider a unit U.

Let *NAPR* be the set of non-abstract parent units of U except class libraries.

Let *IS* be the set of instance of U.

Let *STC* be the set of singleton classes in the system.

This principle is violated if

$$(NAPR \neq \emptyset) \wedge (|IS| \leq 1) \wedge (U \notin STC)$$

Units can be applied: Classes

Heuristics 5.17: It should be illegal for a derived class to override a base class method with a NOP method, that is, a method that does nothing.

Violation check expressions:

Consider a unit U.

Let *OVM* be the set of methods of U which override inherited methods.

Let $ihm \in IHM$.

Let ST_{ovm} be the set of statements in a method *ovm* where $ovm \in OVM$.

This principle is violated if

$$\exists\, ovm\ (ST_{ovm} = \emptyset)$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Heuristics 9.2: Do not change the state of an object without going through its public interface.

Violation check expressions:

Consider a unit U.

Let *FO* be the set of fields of other units except U.

Let *IFO* be the set of inter-type fields of other units except U.

Let UWF_{fo} be the set of units writing to a field *fo* where $fo \in FO$.

Let $UWIF_{ifo}$ be the set of units writing to an inter-type field *ifo* where $ifo \in IFO$.

This principle is violated if

$$\exists fo\ (U \in UWF_{fo}) \vee \exists ifo\ (U \in UWIF_{ifo})$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Duplicate Code: The same code structure in two or more places is a good sign that the code needs to be refactored: if you need to make a change in one place, you'll probably need to change the other one as well, but you might miss it.

Violation check expressions:

Consider a unit U.

Let *M* be the set of methods of U.

Let *OM* be the set of other methods of all units except the given method *m* where $m \in M$.

Let ASM_m be the set of adjacent statements of *m*.

Let ASM_{om} be the set of adjacent statements of *om* where $om \in OM$.

The principle is violated if

$$\exists m \exists om\ (ASM_m \cap ASM_{om} \neq \emptyset)$$

Units can be applied: Classes/ Abstract Classes

Long Method: Systems that have a majority of small methods tend to be easier to extend and maintain because they're easier to understand and contain less duplication.

Violation check expressions:

Consider a unit U.

Let *FT* be the set of functions (methods/ inter-type methods/ advices) of U.

Let ST_{ft} be the set of statements of a function *ft* where $ft \in FT$.

This unit is violated if

$$\exists ft\ (|ST_{ft}| > 13)$$

Note: The number thirteen is taken from very high LOC/Class divided by very high NOM [39].

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Long Parameter List: Long parameter lists are hard to understand. You don't need to pass in everything a method needs, just enough so it can find all it needs.

Violation check expressions:

Consider a unit U.

Let *M* be the set of methods of U.

Let *IM* be the set of inter-type methods of U.

Let $UGDFP_{mim}$ be the set of units getting other's data and sending them (without any pre-processing) to *mim* where $mim \in (M \cup IM)$.

This principle is violated if

$$\exists mim\ UGDFP_{mim} \neq \emptyset$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Feature Envy: Data and behavior that acts on that data belong together. When a method makes too many calls to other classes to obtain data or functionality, Feature Envy is in the air.

Violation check expressions:

Consider a unit U.

Let *F* be the set of fields of U.

Let *IF* be the set of fields of U.

Let *M* be the set of methods of U.

Let *IM* be the set of inter-type methods of U.

Let *STM* be the set of static methods of U.

Let $FSMTU_{stm}$ be the set of fields that the static method *stm* uses where $stm \in STM$.

Let $IFSMTU_{stm}$ be the set of inter-type fields that the static method *stm* uses where $stm \in STM$.

Let $MSMTU_{stm}$ be the set of methods that the static method *stm* uses where $stm \in STM$.

Let $IMSMTU_{stm}$ be the set of inter-type methods that the static method *stm* uses where $stm \in STM$.

Let *UIU* be the set of user interface units in the software.

Let *ADU* be the set of adapter units in the software.

Let *FCU* be the set of façade units in the software.

This principle is violated if

$$\exists stm\ ((|FSMTU_{stm} - F| + |IFSMTU_{stm} - IF| + |MSMTU_{stm} - M| + |IMSMTU_{stm} - IM|) > (|FSMTU_{stm} \cap F| + |IFSMTU_{stm} \cap IF| + |MSMTU_{stm} \cap M| + |IMSMTU_{stm} \cap IM|)) \wedge (U \notin (UIU \cup ADU \cup FCU))$$

Note: This principle is checked from a pair of collaboration aspects, a pair of collaboration classes, and a pair of collaboration aspect and class which does not include a pair of class or aspect and its ancestors and a pair of class or aspect calling to class libraries.

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Freeloader (a.k.a. Lazy Class): A class that isn't doing enough to pay for itself should be eliminated.

Violation check expressions:

Consider a unit U.

Let *F* be the set of fields of U.

Let *IF* be the set of fields of U.

Let *M* be the set of methods of U.

Let *IM* be the set of inter-type methods of U.

Let *P* be the set of pointcuts of U.

Let *AV* be the set of advices of U.

This principle is violated if

$$F \cup IF \cup M \cup IM \cup P \cup AV = \varnothing$$

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects/ Interfaces

Speculative Generality: Often methods or classes are designed to do things that in fact are not required. The dead-wood should probably be removed.

Violation check expressions:

Consider a unit U.

Let *FT* be the set of members (methods/ inter-type methods/ constructors) of U.

Let UCM_{ft} be the set of units calling to a function *ft* where $ft \in FT$.

Let *P* be the set of pointcuts of U.

Let J_p be the set of corresponding join points of a pointcut *p* where $p \in P$.

This principle is violated if

$$(FT \neq \varnothing) \wedge \exists ft\ (UCM_{ft} = \varnothing) \wedge \exists p\ (J_p = \varnothing)$$

Note: Heuristic 3.8 = T, Speculative Generality = F.

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

Message Chains: Occur when you see a long sequence of method calls or temporary variables to get some data. This chain makes the code dependent on the relationships between many potentially unrelated objects.

Violation check expressions:

Consider a unit U.

Let *FT* be the set of functions (methods/ inter-type methods/ advices) of U.

Let ST_{ft} be the set of statements of a function *ft* where $ft \in FT$.

Let MSG_{stft} be the set of messages contained in a statement *st* of a function *ft* where $st \in ST$.

This principle is violated if

$$\exists st \exists ft(|MSG_{stft}| > 6)$$

Note: The number six is taken from the average number of items that a person can keep in his or her short-term memory [36].

Units can be applied: Classes/ Abstract Classes/ Aspects/ Abstract Aspects

* Anonymous Pointcut Definitions: The use of the pointcut definition predicate directly in an advice may reduce legibility and hide the predicate's intention.

Violation check expressions:

Let *A* = {*call*, *execution*, *get*, *set*, *initialization*, *preini-tialization*, *static-initialization*, *handler*, *adviceexecution*, *within*, *withincode*, *cflow*, *cflowbelow*, *if*} be the set representing all the primitive pointcuts in AspectJ that are not related to context exposure.

Let *B* be the set of the tokens in a given pointcut definition. The pointcut definition is an anonymous pointcut definition if and only if the predicate $\forall b \in B \neg \exists a \in A | b = a$ is true. [17]

Units can be applied: Aspects/ Abstract Aspects

*Lazy Aspects: if an aspect has few responsibilities, and its elimination could result in benefits at the maintenance phase.

Violation check expressions:

Consider an aspect α .

The crosscutting members of α are the collection of all advice, pointcuts, declare constructions and inter type declarations directly defined in α .

Consider η as the number of crosscutting members of α .

An aspect is considered a lazy one whenever the predicate $\eta == 0$ holds.

The function could be defined as: $f(\alpha) = \eta == 0$. [17]

Units can be applied: Aspects/ Abstract Aspects

*Aspect Feature Envy: In AspectJ, pointcuts could be defined in aspects and also in classes. If a single aspect uses a class-defined pointcut, it is interesting to move the pointcut from the class to the aspect that uses it.

Violation check expressions:

If the number of pointcuts in a class X is given by η, the class suffers from the feature envy bad smell if the predicate $\eta > 0$ holds.

The function could be defined as: $f(X) = \eta > 0$. [17]

Units can be applied: Classes/ Abstract Classes

*Abstract Method Introductions: Aspects could be used to add state and behavior into existing classes. This is made through the inter-type declaration mechanism. However, the use of this functionality may cause problems when abstract methods are inserted in application classes.

Violation check expressions:

If the set of modifiers of a inter type method declaration ι is given by $m(\iota)$ and the abstract modifier is given by α , the function that describes if an inter type method declaration is an abstract one could be defined as $f(\iota) = \alpha \in m(\iota)$.

If the result of the function evaluation is true, then the inter type declaration is an abstract one. [17]

Units can be applied: Aspects/ Abstract Aspects

Large Aspects: Whenever an aspect tries to deal with more than one concern, it could be divided in as many aspects as there are concerns.

Violation check expressions:

Consider a unit U.

Let *NSF* be the set of non-static fields of U.

Let *NSIF* be the set of non-static inter-type fields of U.

Let *FT* be the set of functions (methods/ inter-type methods/ advices/ constructors) of U.

Let $FTUFIF_{nsfx}$ be the set of functions using nsf_x where $nsf_x \in (NSF \cup NSIF)$.

Let $FTUFIF_{nsfy}$ be the set of functions using nsf_y where $nsf_y \in (NSF \cup NSIF)- \{nsf_x\})$.

Let $FTUFT_{\mathrm{ftufifnsfx}}$ be the set of functions using *ftufifnsfx* where $ftufifnsfx \in FTUFIF_{nsfx}$.

Let $FTUFT_{\mathrm{ftufifnsfy}}$ be the set of functions using *ftufifnsfy* where $ftufifnsfy \in FTUFIF_{nsfy}$.

This principle is violated if

$$(((|NSF \cup NSIF\ | > 1) \wedge (|FT\ |> 1)) \wedge (\exists\ nsfx\ \exists\ nsfy\ ((FTUFIF_{nsfx} \neq \emptyset) \wedge (FTUFIF_{nsfy} \neq \emptyset) \wedge (FTUFIF_{nsfx} \cap FTUFIF_{nsfy} = \emptyset)) \wedge \forall\ ftufifnsfx\ \forall\ ftufifnsfy\ ((ftufifnsfy \notin FTUFT_{\mathrm{ftufifnsfx}}) \wedge (ftufifnsfx \notin FTUFT_{\mathrm{ftufiffnsfy}}) \wedge (FTUFT_{\mathrm{ftufifnsfx}} \cap FTUFT_{\mathrm{ftufiffnsfy}} = \emptyset)))$$

Units can be applied: Aspects/ Abstract Aspects