CHAPTER VI

APPLYING DESIGN PRINCIPLE VIOLATION CHECK DEFINITIONS

Chapter VI applies the design principle violation check definitions to a set of aspect-oriented systems. Chi-Square test for estimating the percentage of repeated design principle violations is conducted as well. The test reveals that design flaws occur repeatedly in each system. This chapter also describes points of concern for violative-system adjustments that are *variations in metric values*, *side-effects of alterations*, and *changes in the system behavior*.

6.1 Applications of Design Principle Violation Check Definitions

Design principle violation check definitions are applied to evaluate fifty systems written in AspectJ. These 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]. A frequency distribution between *repeated* design principle violation units and *non-repeated* design principle violation units are examined using the Pearson's chi-square test [44] to suggest for the utilization of design principle violation check definitions. It is used to test a null hypothesis that whether the relative frequencies of occurrence of observed events follow a specified frequency distribution or not. Pearson's chi-square is calculated by finding the difference between each observed and theoretical frequency for each possible outcome, squaring them, dividing each by the theoretical frequency, and taking the sum of the results. The number of degrees of freedom is equal to the number of possible outcomes, minus 1. The hypothesis is tested with 1 degree of freedom, so the *adjusted chi-square test* is applied:

$$X^2 = \sum_{i=1}^{n} \frac{(|Oi - Ei| - 0.5)^2}{Ei}$$

where Oi = an observed frequency; Ei = an expected (theoretical) frequency, asserted by the null hypothesis; n = the number of possible outcomes of each event.

The null hypothesis (H0) is that *the number of repeated design principle violation units in the population is approximately 60 percents*. The alternative hypothesis (H1) is

that *the number of repeated design principle violation units in the population is not approximately 60 percents*. From fifty systems, there are 284 units which violate the same design principles and 208 units which violate the different design principles, so

$$X^2 = \frac{(|284-(284+208)\times 0.6|-0.5)^2}{(284+208)\times 0.6} + \frac{(|208-(284+208)\times 0.4|-0.5)^2}{(284+208)\times 0.4} =$$

$$\frac{(|284-295.2|-0.5)^2}{295.2} + \frac{(|208-196.8|-0.5)^2}{196.8} = 0.39 + 0.58 = 0.97$$

Consultation of the chi-square distribution for 1 degree of freedom shows that the probability of observing this difference is approximately 2.71. This probability (0.97) is lower than conventional criteria for statistical significance, so the null hypothesis is accepted. This result suggests that system developers frequently make the same mistakes, so the quality assurance people should keep their focuses on the design flaws which have been detected.

6.2 Major Considerations in Adjusting Design

Although violations of design principles are detected by design principle violation check definitions, final decision for correcting designs should be done by developers together with regression testing these points: *variations in metric values*, *side-effects of alterations*, and *changes in the system behavior*.

6.2.1 Variations in Metric Values

A quality factor is generally evaluated by a set of metrics. In the same way, adjusting designs for the quality improvement should be done together with considering changes in those metric values. For example, Figure 6.1 shows an implementation of software with Builder design pattern. The software contains 3 classes (class *Main*, class *TextCreator*, and class *XMLCreator*), 1 abstract class (abstract class *Creator*), and 1 aspect (aspect *CreatorImplementation*). Class *TextCreator*, class *XMLCreator*, and aspect *CreatorImplementation* separate representations from the construction of an input object. So, the same input (original context built in class *Main*) can be created with different representations (normal text or XML).

```
public class XMLCreator extends Creator {
 protected String type = null; // point1 of Heuristic 2.1
 protected String attribute = null;
 public void processType(String newType) {
  representation = "<"+newType+">\n"; // Heuristic 2.7
  type = newType;
 }
 public void processAttribute(String newAttribute) {
  checkAttribute();
  representation += ("\t<" + newAttribute + ">"); // H2.7
  this.attribute = newAttribute;
 }
 public void processValue(String newValue) {
  representation += newValue; // Heuristic 2.7
 }
 protected void checkType() {
  if (type != null) {
   representation += ("</" + type + ">\n"); // H 2.7
   type = null;  }
 }
 protected void checkAttribute() {
  if (attribute != null) {
   representation += ("</" + attribute + ">\n"); // H 2.7
   attribute = null;  }
 }
 public String getRepresentation() {
  checkAttribute();
  checkType();
  return representation; // Heuristic 2.7
 }
}
public abstract class Creator {
        ...
}
```

```
public class TextCreator extends Creator {
 public void processType(String newType) {
  representation = "This is a new "+newType+":\n"; // H 2.7
 }
 public void processAttribute(String newAttribute) {
  representation += ("Its " + newAttribute + " is "); // H 2.7
 }
 public void processValue(String newValue) {
  representation += (newValue + ".\n");// Heuristic 2.7
 }
}
public class Main {
 protected static void build(Creator builder) {
        ...
 }
 public static void main(String[] args) throws Exception {
  Creator builder1 = new TextCreator();
  Creator builder2 = new XMLCreator();
  build(builder1);
  build(builder2);
  System.out.println(builder1.getRepresentation());
  System.out.println(builder2.getRepresentation());
 }
}
public aspect CreatorImplementation {
 public String Creator.representation; // point2 of H 2.1
 public String Creator.getRepresentation() {
  return representation;  }
 public pointcut outsideCreator(): (set(public String
Creator+.representation) || get(public String
Creator+.representation)) && ! (within(Creator+) ||
within(CreatorImplementation)); // Speculative Generality
 declare error: outsideCreator(): "variable result is aspect
protected. Use getResult() to access it";
}
```

Figure 6.1: An implementation of software with Builder design pattern.

Table 6.1: Metric variations after changing Builder system

System Version	Metrics								
	LOCC	WOM	DIT	NOC	CFA	CMC	CAE	LCO	NOPM
Builder	79	16	2	2	0	3	1	11	14
Builder (corrected point 1 of Heuristic 2.1)	79	16	2	2	0	3	1	11	14
Builder (corrected point 2 of Heuristic 2.1)	82	17	2	2	0	4	1	11	15
Builder (corrected Speculative Generality)	80	17	2	2	0	4	1	11	15
Builder	79	16	2	2	0	3	1	11	14
Builder (corrected point 1 of Heuristic 2.1)	79	16	2	2	0	3	1	11	14
Builder (corrected Speculative Generality)	77	16	2	2	0	3	1	11	14
Builder (corrected point 2 of Heuristic 2.1)	80	17	2	2	0	4	1	11	15
Builder	79	16	2	2	0	3	1	11	14
Builder (corrected point 2 of Heuristic 2.1)	82	17	2	2	0	4	1	11	15
Builder (corrected point 1 of Heuristic 2.1)	82	17	2	2	0	4	1	11	15
Builder (corrected Speculative Generality)	80	17	2	2	0	4	1	11	15
Builder	79	16	2	2	0	3	1	11	14
Builder (corrected point 2 of Heuristic 2.1)	82	17	2	2	0	4	1	11	15
Builder (corrected Speculative Generality)	80	17	2	2	0	4	1	11	15
Builder (corrected point 1 of Heuristic 2.1)	80	17	2	2	0	4	1	11	15
Builder	79	16	2	2	0	3	1	11	14
Builder (corrected Speculative Generality)	77	16	2	2	0	3	1	11	14
Builder (corrected point 1 of Heuristic 2.1)	77	16	2	2	0	3	1	11	14
Builder (corrected point 2 of Heuristic 2.1)	80	17	2	2	0	4	1	11	15
Builder	79	16	2	2	0	3	1	11	14
Builder (corrected Speculative Generality)	77	16	2	2	0	3	1	11	14
Builder (corrected point 2 of Heuristic 2.1)	80	17	2	2	0	4	1	11	15
Builder (corrected point 1 of Heuristic 2.1)	80	17	2	2	0	4	1	11	15

Builder implementation violates 2 points of Heuristic 2.1 (two non-private fields in class *XMLCreator* and one inter-type field in aspect *CreatorImplementation*), 2 points of Heuristic 2.7 (directly accessing *representation* field in class *TextCreator* and class *XMLCreator*) and 1 point of Speculative Generality (unused pointcut *outsideCreator* in aspect *CreatorImplementation*). Consider the metric LOCC, WOM, DIT, NOC, CFA, CMC, CAE, LCO, and NOPM (the definitions of these metrics are in Section 7.2), Table 6.1 shows the metric variations (two violation points of Heuristic 2.7 is disappeared when

point 2 of Heuristic 2.1 is corrected) for each system after all types of its flaws are corrected in different sequences. The final metric values of all versions are the same because the modified part of each principle is independent from others. However, the metric variations suggest that developers should correct point 1 of Heuristic 2.1 and a point of Speculative Generality which produces the positive or neutral results and should skip modifying point 2 of Heuristic 2.1 because it may produce the negative metric results.

6.2.2 Side-effects of Alterations

Either positive or negative side-effects can happen while altering systems. Positive side-effects are removing other design principle violations after modifications. Negative side-effects are producing other design principle violations after modifications. Their occurrences mainly result from tangling of the affected part and refactorings which are applied. Happening or canceling violation points of other principles are risen depending on two main reasons above. For example, correcting point 2 of Heuristic 2.1 in Figure 6.1 suddenly removes the violation of Heuristic 2.7 in class *TextCreator* and class *XMLCreator* because methods in class *TextCreator* and class *XMLCreator* use an inter-type field *representation* of aspect *CreatorImplementation* (the effect of tangling). So, in case the positive side-effect is important than the negative metric results (modifying the software to conform Heuristic 2.7 generates negative metric results), developers should alter point 2 of Heuristic 2.1.

6.2.3 Changes in the System Behavior

Another consideration for refactoring a system is the preservation of the system behavior. Every time that developer makes changes, regression testing should be conducted. Table 6.2 describes black box test cases for Builder implementations which test every method that may be affected by modifications. Table 6.3 shows the testing results. The results indicate that the behavior of software after removing Heuristic 2.1 design flaws is preserved and the behavior of software after removing Speculative Generality design flaws is changed. So, a violation point of Heuristic 2.1 should be removed and a violation point of Speculative Generality should be ignored since an alteration changes the behavior *checking outsider call* of aspect CreatorImplementation.

Table 6.2: Black box test cases for Builder

Test ID	Input	Expected Results
1) testTextProcessType	- Create TextCreator object. - Call method processType with argument "Person".	"This is a new Person:\n"
2) testTextProcessAttribute	- Create TextCreator object. - Call method processType with argument "Person". - Call method processAttribute with argument "Name".	"This is a new Person:\n" + "Its Name is "
3) testTextProcessValue	- Create TextCreator object, - Call method processType with argument "Person". - Call method processAttribute with argument "Name". - Call method processValue with argument "James Brick".	"This is a new Person:\n" + "Its Name is James Brick.\n"
4) testXMLProcessType	- Create XMLCreator object. - Call method processType with argument "Person".	"<Person>\n" + "</Person>\n"
5) testXMLProcessAttribute	- Create XMLCreator object. - Call method processType with argument "Person". - Call method processAttribute with argument "Name".	"<Person>\n" + "\t<Name>"+ "</Name>\n" + "</Person>\n"
6) testXMLProcessValue	- Create XMLCreator object, - Call method processType with argument "Person". - Call method processAttribute with argument "Name". - Call method processValue with argument "James Brick".	"<Person>\n" + "\t<Name>James Brick" + "</Name>\n" + "</Person>\n"
7) testTextGetRepresentation	- Create TextCreator object. - Set representation field value with "ABC".	"ABC"
8) testXMLGetRepresentation	- Create TextCreator object. - Call method processType with argument "Person". - Call method processAttribute with argument "Name". - Set representation field with "ABC".	"ABC"+ "</Name>\n" + "</Person>\n"
9) testPointcutOutsideCreator	- Set representation field from Main class with "XYZ".	Compilation Error: "variable result is aspect protected. Use getResult() to access it"

Table 6.3: Black box testing results for the affected units

Test ID	Heuristic2.1(1)	Heuristic2.1(1) Heuristic2.1(2)	Heuristic2.1(1) Heuristic2.1(2) SpeculativeGen	Heuristic2.1(1) Speculative Gen	Heuristic2.1(1) SpeculativeGen Heuristic2.1(2)
1	Passed	Passed	Passed	Passed	Passed
2	Passed	Passed	Passed	Passed	Passed
3	Passed	Passed	Passed	Passed	Passed
4	Passed	Passed	Passed	Passed	Passed
5	Passed	Passed	Passed	Passed	Passed
6	Passed	Passed	Passed	Passed	Passed
7	Passed	Passed	Passed	Passed	Passed
8	Passed	Passed	Passed	Passed	Passed

Test ID	Heuristic2.1(1)	Heuristic2.1(1) Heuristic2.1(2)	Heuristic2.1(1) Heuristic2.1(2) SpeculativeGen	Heuristic2.1(1) Speculative Gen	Heuristic2.1(1) SpeculativeGen Heuristic2.1(2)
9	Passed	Passed	Failed!	Failed!	Failed!
Test ID	**Heuristic2.1(2)**	**Heuristic2.1(2) Heuristic2.1(1)**	**Heuristic2.1(2) Heuristic2.1(1) Speculative Gen**	**Heuristic2.1(2) SpeculativeGen**	**Heuristic2.1(2) SpeculativeGen Heuristic2.1(1)**
1	Passed	Passed	Passed	Passed	Passed
2	Passed	Passed	Passed	Passed	Passed
3	Passed	Passed	Passed	Passed	Passed
4	Passed	Passed	Passed	Passed	Passed
5	Passed	Passed	Passed	Passed	Passed
6	Passed	Passed	Passed	Passed	Passed
7	Passed	Passed	Passed	Passed	Passed
8	Passed	Passed	Passed	Passed	Passed
9	Passed	Passed	Failed!	Failed!	Failed!
Test ID	**SpeculativeGen**	**SpeculativeGen Heuristic2.1(1)**	**SpeculativeGen Heuristic2.1(1) Heuristic2.1(2)**	**SpeculativeGen Heuristic2.1(2)**	**SpeculativeGen Heuristic2.1(2) Heuristic2.1(1)**
1	Passed	Passed	Passed	Passed	Passed
2	Passed	Passed	Passed	Passed	Passed
3	Passed	Passed	Passed	Passed	Passed
4	Passed	Passed	Passed	Passed	Passed
5	Passed	Passed	Passed	Passed	Passed
6	Passed	Passed	Passed	Passed	Passed
7	Passed	Passed	Passed	Passed	Passed
8	Passed	Passed	Passed	Passed	Passed
9	Failed!	Failed!	Failed!	Failed!	Failed!