Selection of Metrics for Predicting the Appropriate Application of design patterns

Jonatan Hernandez Atsuto Kubo Hironori Washizaki
Fukazawa Yoshiaki
Dept. of Computer Science and Engineering,
Waseda University,
Tokyo, Japan
{jonatan,washizaki,fukazawa}@waseda.ac.jp
kubo@nii.ac.jp

March 2010

Abstract

Design patterns are known for their usefulness to solve recurrent problems. They are also a way of transmitting knowledge and experience by using proven, high quality solutions. A problem that emerges when using design patterns is that it is not clear how to measure the impact that has its application on the source code. The relationships between metrics and design patterns is not clear. We propose an experiment for measuring the usefulness of metrics and their success in predicting correct usage of design patterns. With this experiment we will explore which metrics capture best the relationship of design patterns quality with the source code.

1 Introduction

Design patterns are a established solution for recurring problems. The GOF Patterns[4] are particularly well known. However its many advantages and wide spread use of design patterns, is not clear how to measure the impact that has the application of design patterns. Although it is well known that designs pattern add value to the source code by increasing its maintainability, scalability, extensibility and other quality attributes it is not clear how to measure its influence in a quantitative way. This makes hard to judge whether a design pattern is applied inappropriately or not.

2 Problem

There have been researches to measure the effects of design patterns in the source code.[7, 5, 9] However no consensus has been reached in regards to which metrics are good for evaluating the effects of a design patterns in the source code. The objective of this research is towards the understanding of the relationship among design patterns, metrics and appropriate usage of design patterns.

3 Examples

The following four examples of appropriate and inappropriate application of design patterns are provided. The examples are divided in two examples taken from the book Refactoring to Patterns[6] and from the source code of JUnit.

3.1 Appropriate Application

The intent of the Strategy Pattern is: "define a family of algorithms, encapsulate each one and make them interchangeable"[11]

3.1.1 From Refactoring to Patterns

The first example of a good use of design pattern is from the chapter replace conditional logic with strategy[6].

In the example below there are several algorithms for making a calculation. The appropriate algorithm is selected using an if clause. Figure 1 shows the structure of the code before applying the design pattern.

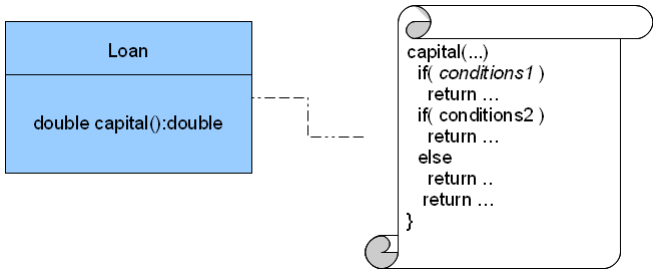

Figure 1: Structure before applying the Strategy Pattern

Figure 2 shows the code after applying the Strategy Pattern. This application of the Strategy Pattern has the benefits of clarifying the logic, and making it simpler[6].

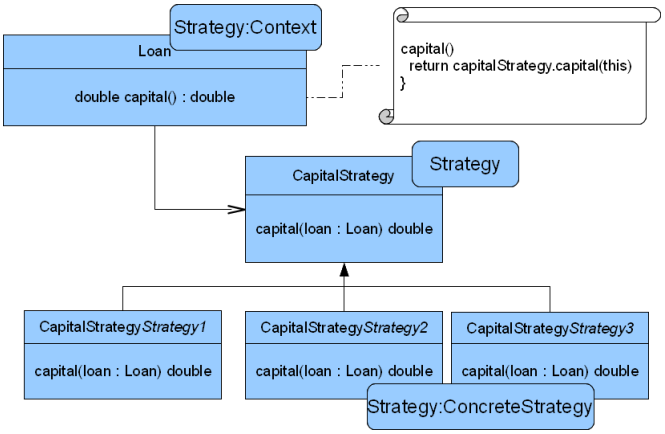

Figure 2: Structure after applying the Strategy Pattern

3.1.2 From JUnit

The example in the Figure 3 is from JUnit. This example of the Strategy Pattern was in the source code from the start and it still remains in recent versions of JUnit, with very little modifications. It is considered a success because of this permanence.

3.2 Inappropriate Application

In the following sections examples of the Singleton Pattern. The intent of the Singleton Patterns is "Ensure a class has only one instance, and provide a global point of access to it". [10]

3.2.1 From Refactoring to Patterns

The example is called "Inline Singleton Pattern". This is the example of the singleton design pattern being a failure[6]. First we have the code as shown in Figure 4. Then the code is refactored as shown in Figure 5.

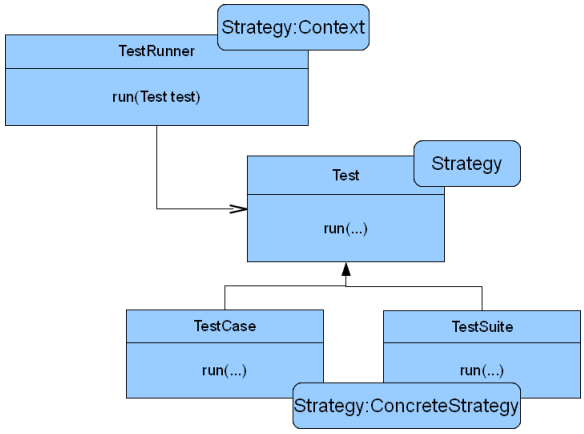

Figure 3: Structure of the Strategy Pattern in JUnit

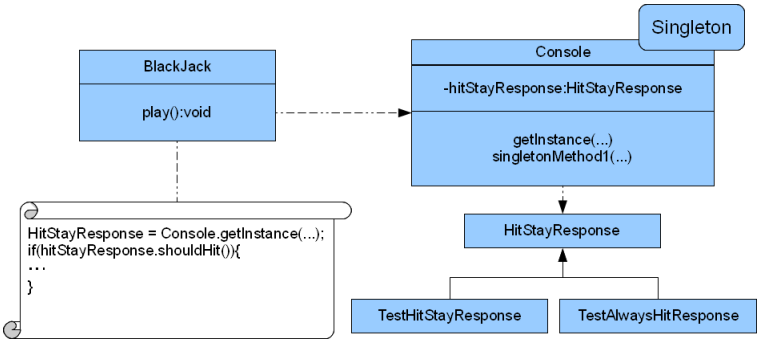

Figure 4: Structure when the Singleton Pattern is present

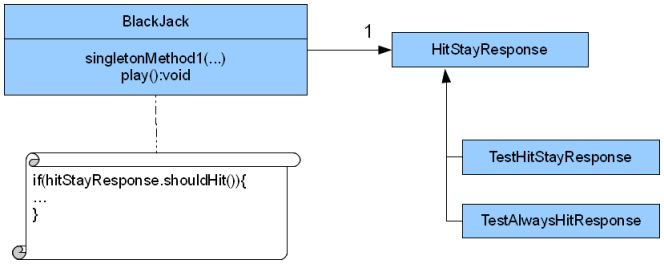

Figure 5: Structure after removing the Singleton Pattern

In this case the application of the design pattern was a failure because "the code needs access to an object buy doesn't need a global point of access to it."[6] and give us the following advice: "It is better to decide the right access level for an object than to make it global"

3.2.2 From JUnit

This is an example from the class org.junit.internal.runners.CompositeRunner of JUnit(Program 1):

This code was removed, and although this sorting functionality is preserved in subsequent versions, it is implemented in a different way that does not make use of the Strategy Pattern.

Program 1 Strategy Pattern in JUnit

```
public void sort(final Sorter sorter) {
  Collections.sort(fRunners, new Comparator<Runner>() {
    public int compare(Runner o1, Runner o2) {
      return sorter.compare(o1.getDescription(), o2.getDescription());
    }
  });
  for (Runner each : fRunners)
  sorter.apply(each);
}
```

4 Solution

We propose a process for observing how the metrics change as the design patterns evolve in the source code repository. We chose to use metrics because we make the supposition that there is a relationship among design patterns, metrics and quality.

There are results of studies concluded that high scores of CK metrics are not obtained when using design patterns[9] and some others concluded that CK metrics and design patterns are "mainly congruent"[5]. This results leave room for our premise that the metrics values could have a relationship with the application of design patterns.

We stablish the rule that the longer a design pattern stays in the code, the higher the possibility that it is a code of high quality. The metrics will change accordingly.

In this approach we also stablish the following rules: (Figure 6):

1. If the design pattern remains, then it was applied correctly
2. If the design pattern is removed, then it was not applied correctly
3. The change in the metrics is always the same, an it is related to the design pattern

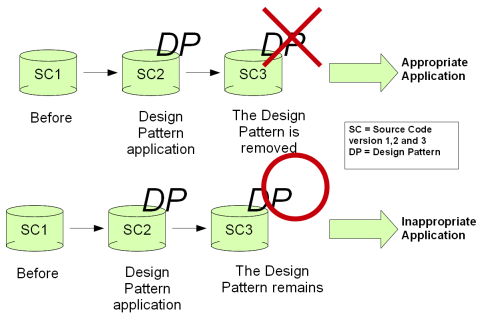

Figure 6: Examples of design pattern application: appropriate and inappropriate application

4.1 Using metrics to identify an inappropriate application

In this example we see an inadequate application of the design pattern. How the metrics changed and how this can be used to identify this application is also shown.

The following metrics are calculated using the eclipse plug-in metrics[1]

- **VG** McCabe Cyclomatic Complexity[8]
- **PAR** Number of Parameters[1]
- **LCOM** Lack of Cohesion of Methods[3]

Using the previous examples of applications of design patterns as a start, we will show how the metrics can be used for the identification task. The example that will be presented is the inappropriate use in JUnit from the previous section.

In this pattern example there are two classes involved. The class *Runner*, which has the role of Strategy, and the class *CompositeRunner*, which has the role of Context. For this example there are two versions of the source code. One with the design pattern and one without the design pattern. The class that will be used for the metrics calculation is *CompositeRunner* because the metrics in the class *Runner* did not change. This example of an inappropriate application has two sets of values: 1) the values for the metrics in the version with the design pattern are shown in Table 1. 2) The metrics without the design pattern are shown in Table 2.

Metric	Total	Average	Standard Deviation	Max
VG	-	1	0	1
PAR	-	0.875	0.781	2
LCOM	0.667	-	-	-

Table 1: Inappropriate application example part 1: Metrics with the design pattern

Metric	Total	Average	Standard Deviation	Max
VG	-	1.556	0.685	3
PAR	-	0.667	0.471	1
LCOM	0	-	-	-

Table 2: Inappropriate application example part 2: Metrics without the design pattern

This changes of the metrics (shown in Table 3) can be associated to the inappropriate application of the design pattern, therefore using them as a predictor for this inappropriate usage of the Strategy design pattern. From the tables the metrics values seem to be higher when the design pattern is present. The metrics PAR and LCOM both have a higher value when the pattern is present. Only the VG metric reduced its value in all three measurements.

Metric	Total	Average	Standard Deviation	Max
VG	-	0.556	0.685	2
PAR	-	-0.208	-0.309	1
LCOM	-0.667	-	-	-

Table 3: Inappropriate application example part 3: Changes in the metrics values

While this is no evidence for the metrics as a good predictor for the appropriate application of design patterns, it can be used as a starting point for the following experiment.

4.2 Process

The process has two main parts, learning and classifying. Each part has its own set of data. For this the collected dataset will be divided in two, one for each part.

- **Learning:** Samples from previous applications of design pattern → Evaluation of metrics → Machine Learning → Classification model
- **Classifying:** Test data for applications of design pattern → Machine Classification

4.2.1 Steps

The steps of the process detailed are as follows:

- *Learning:*
  1. **Collect design patterns** in the target source code repository

2. **Decide** which of those changes are failure cases and which are success cases
3. **Measure** the values the selected metrics for the source code related to the pattern
4. **Calculate** the difference of the metrics values
5. **Learn** using these values as input for the machine learning

- *Classifying:*

1. **Test** using the model generated in the previous part of the process with test data.

4.3 Example

4.3.1 Metrics

The following metrics are used in this example:

Name	Acronym	Scope	Source
NSC	Number of Children	Total	CK[2]
NBD	Nested Block Depth	Average	Eclipse[1]
VG	McCabe Cyclomatic Complexity	Average	McCabe[8]
VG	McCabe Cyclomatic Complexity	Standard Deviation	McCabe[8]
NSM	Number of Static Methods	Total	Eclipse[1]
WMC	Weighted Methods per Class	Total	CK[2]

4.3.2 Data

For this classification problem, as training and test date we have chosen the open source project JUnit. We find 8 cases were the Strategy design pattern is used. Using this 8 data points we take 4 as input for the learning machine, and the remaining 4 as the test data. The following table is an example of input from the class *Junit4TestAdapter*:

When	NSC-Total	NBD-Avg.	VG-Avg.	VG-Std	NSM-Total	WMC-Total
Before Application	0	1.08	1.25	0.83	0	15
After Application	0	1	0.5	0.71	0	8
DELTA(After - Before)	0	-0.08	-0.75	-0.12	0	-7

4.4 Results

Results of using the previous metrics as a predictor of good use of the design pattern are 50%.

Total Cases	Correct Cases	Precision
4	2	50%

The result means that the selected metrics did not provide a good prediction. Then we have to repeat this experiment with different metrics, until metrics that are a good predictor are found.

5 Conclusions and Future Work

From this preliminary result we can see that the chosen metrics are not a good predictor of the future success or failure of the design patterns. In the example presented there is still very few data points and more metrics need to be analyzed. Future work will help in finding metrics that can be used as more accurate predictors.

6 Acknowledgements

We would like to thank Norihiro Yoshida for his insightful questions during the shepherding process. The quality of this paper improved a because of this. Also we would like to thank Yann-Gaël Guéhéneuc for his advice and pointers. Thank you.

References

[1] G. Boissier. Metrics 1.3.8, December 2010. URL `http://metrics2.sourceforge.net/`.

[2] SR Chidamber and CF Kemerer. A Metrics Suite for Object-Oriented Design. *IEEE Trans. Software Eng*, 20(6):476–493, 1994.

[3] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 2002. ISSN 0098-5589.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: elements of reusable object-oriented software. 1994. *Reading, Massachusets, US: Addison-Wesley*.

[5] B. Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001. ISSN 0164-1212.

[6] J. Kerievsky. *Refactoring to patterns*. Pearson Education, 2005. ISBN 3827322626.

[7] F. Khomh, Y.G. Guéhéneuc, and P. Team. An Empirical Study of Design Patterns and Software Quality. 2008.

[8] T.J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, pages 308–320, 1976. ISSN 0098-5589.

[9] G. M. Norihiro, N. Sakamoto, and K. Ushijima. Evaluation and analysis of applying design patterns. 1999.

[10] Portland Pattern Repository. Singleton pattern, December 2010. URL `http://c2.com/cgi/wiki?SingletonPattern/`.

[11] Portland Pattern Repository. Strategy pattern, December 2010. URL `http://c2.com/cgi/wiki?StrategyPattern/`.