Paper Title

Author One name, Author Two Name
Department of Your Department
TCSE University
City, Country
Email: {email1, email2}@domain.name

***Abstract*—Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear. Here is the abstract. Keep it simple but clear.**

I. Introduction

In the current trend, design pattern has been widely used for improving software quality. In the academic research, design patterns is one of important research area and applied in different areas [5], [11], [12], many design pattern detection approaches are proposed [6], [13], [14], some work discuss pattern quality metrics [1], [7], [8], design pattern formalization and specification [**?**], [**?**], [**?**], and its benefits [9], [10], [15], [16].

However, using design patterns is not easy, ... Therefore, several approaches are proposed to check the violence of pattern application in a system [2], [17]. We think only static checking to pattern structure is not enough, dynamic testing to test the patterns' semantics is necessary.

In this paper, we propose a test model for design patterns. ....

For example, when we apply *Singleton* pattern, we should conduct the following tests: creating two objects and check if they have the same references. Another example is *Strategy*, when we applied strategy a with this paper, and replace it by another algorithm b, they should have the same behavior, that is, the same output. The following code shows the testing context:

```
1  class TestStrategyDesignPattern {
2      void testSameResult() {
3          Context c = new Context();
4          Strategy a = new StrategyA();
5          c.setStrategy(a);
6          int resultA = c.doIt();
7          Strategy b = new StrategyB();
8          c.setStrategy(b);
9          int restultB = c.doIt();
10         assertEquals(a, b); // should have same
               result
11     }
12 }
```

A. Paper outline

The remainder of this paper is structured as follows: In section 2, we describe the related work to this research.

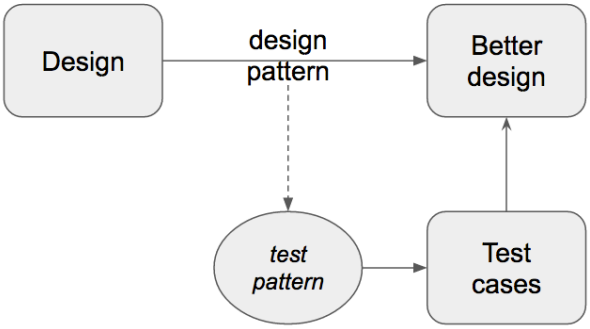

Figure 1. The concept of *test pattern*

Section 3 introduces the details of our approach by two design pattern examples. Section 4 summarizes our approach and future work.

II. Related work

以下針對此研究的相關研究做探討分析：

A. Pattern testing

Chu and Hsueh proposed test refactoring approach in a pattern driven developent [3]. Test-first strategy and code refactoring are the important practices of Extreme Programming for rapid development and quality support. ... However, when the developers perform a pattern-based refactoring to improve the quality, the effort of revising the test cases is much more than that in simple code refactoring (see Fig. 2).

To address this problem, ...

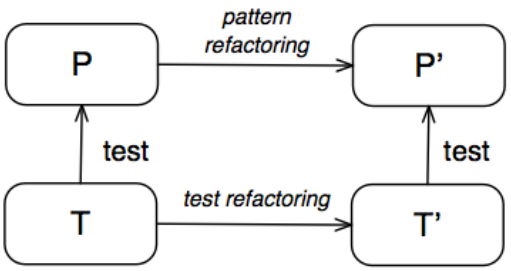

Figure 2. Pattern based test case refactoring

To understand the application context for a pattern in a system, Lin proposed a *Design Pattern Unit Test (DPUT)* approach, which utilizes java Annotation skill to record the pattern utilization and verifies with the expection in the DPUT. This research also design a software framework to help developers design the DPUP in a specification basis. The code is implemented as an Eclipsed plug-in which can automatically transform DPUT into class diagram for better understanding. For the system maintainers they can find out the errors in the earlier phases.

B. Pattern evaluation

設計樣式的效用一直是重要的研究議題。也因此許多學者專家開始投入這方面的研究。Huston [8] 設計了一個實驗以驗證設計樣式是否與軟體衡量相容，亦即，探討應用設計樣式所提昇的軟體品質是否真的可由軟體衡量顯示出來？Huston 的實驗將設計樣式的結構還原成非設計樣式的結構，並以相關的軟體衡量 (software metric) 分別衡量這兩個不同的架構，來比較有無採用設計架構在該軟體衡量上的差異。此實驗顯示設計樣式的品質效能可以透過軟體衡量顯現出來。

III. Our approach

To design the test model for pattern testing, we have to consider the following issues:

- Correctness and Precision. Correctness considers if all faults can be explored by the proposed test cases, and Precision considers if the test can explore the faults in an efficient way. Our test model should satisfy correctness and precision. ...
- Coverage. In general we have many coverage strategies for testing, for example statement coverage or branch coverage. When we test we should enlarge the coverage as we can. Therefore we should care about the normal cases, boundary cases and exception cases. When we utilize this concept to testing a pattern-applied system, what are the boundary cases and exception cases? ...
- Generalization. Gamma et al. classified patterns into creational patterns, structural patterns and behavioral patterns [4]. Patterns in the same group may have same testing issues and methods. We can modularize our testing programs by class generalization. ...

a) Potential Pattern Violation (PPV): To test the pattern-applied application more efficiently, ...

b) Test Pattern for Design Pattern (TP4DP): One of the benefits of design pattern is it is well structured and contains executable code for programmers. Our TP4DP follows the same concept. A test pattern has the following sections: ...

A. Testing "Singleton" pattern

a) Pattern name: Singleton: The intent of the singleton is to ensure a class has only one instance, and provide a global point of access to it.

b) Potential pattern violation: There are two PPV in the Singleton pattern:

- PPV1: When we create two objects from the same class, they refer to different objects.
- PPV2: The original object constructor is not set to be private.

c) Testing guideline: Create two objects of the same Singleton class, check if there are equal by using the operator "==".

d) Demo code: Here is the demo code for the class *Radio*

```
1 class TestRadio {
2     void testRadioSingletonPPE1() {
3         Radio r1 = Radio.instance();
4         Radio r2 = Radio.instance();
5         assertTrue(r1==r2): "Singleton is violated
                ";
6     }
7 }
```

To test the PPV2, we use static testing to check if the constructor is declared as **private**. The example below is the one that doesn't pass the PPV2 checking.

```
1 class Radio {
2     static Radio instance;
3     static Radio getInstance() {
4         if (instance == null)
5             instance = new Radio();
6         }
7         return instance;
8     }
9 }
```

IV. Conclusion

In this paper we propose an idea of pattern test for guiding the programs with patterns applied. We believe a program is error-prone when they are designed in delicate and complex structure. Design patterns are good for flexible design but not easy to understand and apply. For pattern beginners we need a guideline, framework or tool to help them. We demonstrate two examples of *Singleton* and *Observer* to illustrate our ideas. In the future we will explore more design patterns and develop a framework for the pattern test.

References

[1] Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Sofia Charalampidou, and Paris Avgeriou. The effect of gof design patterns on stability: a case study. *IEEE Transactions on Software Engineering*, 41(8):781–802, 2015.

[2] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 324–327. IEEE, 2001.

[3] Peng-Hua Chu, Nien-Lin Hsueh, Hong-Hsiang Chen, and Chien-Hung Liu. A test case refactoring approach for pattern-based software development. *Software Quality Journal*, 20(1):43–75, 2012.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[5] Alan R Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1):36–41, 2000.

[6] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103. IEEE, 2003.

[7] Nien-Lin Hsueh, Peng-Hua Chu, and William Chu. A quantitative approach for evaluating the quality of design patterns. *Journal of Systems and Software*, 81(8):1430–1439, 2008.

[8] Brian Huston. The effects of design pattern application on metric scores. *Journal of Systems and Software*, 58(3):261–269, 2001.

[9] Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013.

[10] TH Ng, SC Cheung, WK Chan, and Yuen-Tak Yu. Do maintainers utilize deployed design patterns effectively? In *Proceedings of the 29th international conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2007.

[11] Michael Oduor, Tuomas Alahäivälä, and Harri Oinas-Kukkonen. Persuasive software design patterns for social influence. *Personal and ubiquitous computing*, 18(7):1689–1704, 2014.

[12] Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Initial explorations on design pattern energy usage. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 55–61. IEEE, 2012.

[13] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on*, 32(11):896–909, 2006.

[14] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 103:102–117, 2015.

[15] Cheng Zhang and David Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

[16] Cheng Zhang and David Budgen. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*, 55(5):822–835, 2013.

[17] Chunying Zhao, Jun Kong, and Kang Zhang. Design pattern evolution and verification using graph transformation. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 290a–290a. IEEE, 2007.