A Semi-Automatic Approach for Test Case Traceability in a Test-Driver Development

Nien-Lin Hsueh, Peng-Hua Chu, Hong-Hsiang Chen
Dept. of Info. Engineering and Computer Science
Feng Chia University, Taichung, Taiwan
nlhsueh@fcu.edu.tw
cph@selab.iecs.fcu.edu.tw, detpiston@gmail.com

Chih-Hung Chang
Dept. of Info. Management
Hsuping Institute of Technology
Taichung County, Taiwan
chchang@mail.hit.edu.tw

Abstract

Test-first strategy and code refactoring are both important features in Extreme Programming methodology. In the test-first strategy, test cases are designed before system implementation. If we want to improve certain non-functional attributes of the programs which are revised by refactoring, the original test cases may be broken or inefficient for testing the new programs. In this paper, we propose an approach for test case traceability in a test-driven development and a plug-in of eclipse for performing it semi-automatically. Finally, we evaluate test coverage to show the practicability of our approach.

1. Introduction

Extreme Programming is a systematic approach with a set of values, rules and practices for rapidly developing high-quality software that provides the highest value for customers [2]. In principle, there are two important features of Extreme Programming: Test-first development and refactoring. Test-first strategy requires the developers to design the test cases before implementation and perform testing immediately as the implementation is complete. Refactoring refers to code revision. Developers revise code by making little changes (refactoring) to improve original design without changing its semantics [7, 1].

In recent years, applying design patterns [5] is considering as an useful way to improve software design. Since it provides “better” solutions to meet non-functional requirements as well as satisfy original functional requirements. As a result of such advantages, some software refactoring approaches perform code revision taking advantage of design patterns, that is, pattern-based refactoring [6, 4].

The following scenario describes a typical sequence of applying pattern-based refactoring in Extreme Programming: 1) A design *D* is proposed; 2) A test case *T* is designed for testing the design *D*; 3) A program *P* is implemented according the design *D*; 4) *P* is tested by *T*; 5) *P* is revised based on a specific design pattern into a new program, said *P’*; 6) *P’* is re-tested by *T* to confirm that the refactoring does not impair the functionality of *P*.

Although software refactoring does not impair the functionality, its interface may be changed and causes the original test cases *T* un-workable in *P’* anymore. Therefore, in the step 6, we have to revise the test *T* into *T’* to make it testable and efficient for testing *P’*. However, the revision from *T* to *T’* is time-consuming and tends to make mistakes. In this paper, we propose the idea of “test case refactoring” and investigate the possibility of automatic test case refactoring.

$$\begin{tikzcd}
D \arrow[r] \arrow[dr, no head] & P \arrow[r] & P' \\
 & T \arrow[u, dashed, "test"] \arrow[r] & T' \arrow[u, dashed, "test"]
\end{tikzcd}$$

Figure 1. Test case refactoring form T to T' for testing P'.

As presented in Figure 1, it illustrates the relationship between *P*, *P’*, *T* and *T’*. *T* is designed specifically for designing *D* to test program *P*. To realize the design *D*, we construct the program *P*. *P* is transformed to *P’* by a pattern-based refactoring process, and hence *P’* is a well-designed structure with high software quality. Although refactoring does not change the original functionality of *P*, but its interface may be modified. Moreover, even the test cases are workable, the test case *T* may be ineffective for testing *P’* since the new program is improved for certain quality attributes by modifying structure. That is, we have to modify *T* and add some test cases which are specific to the property of the design pattern.

2. Our Approach

We develop an Eclipse plug-in to realize this concept. As presented in Figure 2, we have to get the refactoring process (PR_{dp}) of pattern *dp* from *P* to *P'*. We take advantage of the internal API of Eclipse, Language ToolKit (LTK)[8] to record each step (PR_i) of PR_{dp} as the script *dp.xml* when *P* is refactoring to *P'*. Based on *dp.xml*, we can perform refactoring process TR_{dp} to get *T'* from the original test case *T*.

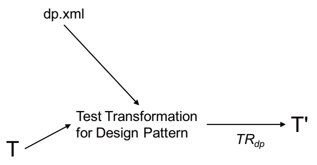

Figure 2. Overview of transformation of test case refactoring process TR_{dp}.

Finally, we utilize EclEmma [3] to compare the coverage on *T* and *T'*. As shown in Figure 3, $cv(P,T)$, the statement coverage of *P* on *T* is 82.5%. When *P* is changed to *P'*, the coverage $cv(P',T)$ is reduced to 3.5% since many test cases are broken or unworkable. After refactoring the test case *T* to *T'*, the coverage $cv(P',T')$ increases to 86.4% - it is 23 times to the coverage $cv(P',T)$.

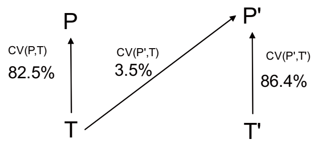

Figure 3. The relationship between refactoring and test case.

The obvious promotion on coverage indicates that our approach can generate test cases effectively and efficiently from *T* to *T'*. The other main objective of our approach is to validate whether the structure is consistent with the design pattern. In addition to above objectives of our approach, there is a benefit of this approach, that is high extensibility of test case. As presented in Figure 4, if we want to implement another enhance *P'* as P_2, we do not have to re-design a test case T_2 for it. We can extend *T'* to create the skeleton of T_2 and modify jot of details of it.

3. Conclusions

Design pattern, testing and refactoring are popular software quality improvement techniques. However, when we

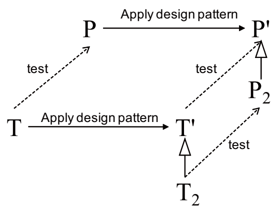

Figure 4. Test case has not only features of design pattern but also high maintainability.

apply them together in the extreme programming approach, certain issues should be considered. So far, there are many CASE tools supporting automatic or semi-automatic design pattern application, however, none of them propose the related test case refactoring. If the test cases cannot be generated in a convenient way, the test-first strategy and design pattern application is difficult to apply together.

In the future, we plan to discover, analyze and classify the refactoring rules for more design patterns, and induce their related rules for test cases refactoring. In the implementation aspect, a development environment for integrating the code refactoring and test refactoring is needed to realize our approach.

Acknowledgment

This work was supported by the National Science Council, Taiwan, under Contract Number NSC97-2218-E-035-004.

References

[1] S. W. Ambler. Test-driven development of relational databases. *Software, IEEE*, 24(3):37–43, May-June 2007.
[2] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.
[3] EclEmma. Eclipse community. *http://www.eclemma.org/index.html*, 2006.
[4] R. France, S. Chosh, E. Song, and D. Kim. A metamodeling approach to pattern-based model refactoring. *Software, IEEE*, 20(5):52–58, Sept.-Oct. 2003.
[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Software*. Addison-Wesley, 1994.
[6] S. U. Jeon, J. S. Lee, and D. H. Bae. An automated refactoring approach to design pattern-based program transformations in java programs. In *APSEC '02: Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, page 337, Washington, DC, USA, 2002. IEEE Computer Society.
[7] J. Kerievsky. *Refactoring to Patterns*. Addison-Wesley, 2004.
[8] T. Widmer. Unleashing the power of refactoring. *http://www.eclipse.org/articles/article.php?file=Article-Unleashing-the-Power-of-Refactoring/index.html*, 2007.