Towards an Assessment of the Quality of Refactoring Patterns

Norihiro Yoshida, Masatomo Yoshida, Katsuro Inoue
Graduate School of Information Science and Technorogy, Osaka University
1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan
Email: {n-yosida, mstm-ysd, inoue}@ist.osaka-u.ac.jp

***Abstract*—Refactoring is a well-known process that is thought to improve the maintainability of object-oriented software. Although a lot of refactoring patterns are introduced in several pieces of literature, the quality of refactoring patterns is not always discussed. Therefore, it is difficult for developers to determine which refactoring patterns should be given priority. In this paper, we propose two quality characteristics of refactoring pattern, and then describe an open source case study on assessing those quality characteristics.**

***Keywords*-refactoring; quality of software pattern; object-oriented programing; software maintenance;**

I. Introduction

Refactoring [1] is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. That is to say, refactoring is a process to improve the maintainability of software systems.

Several practitioners introduce a lot of **Refactoring Patterns (RP)** [1][2]. Each RP includes both a description of a **refactoring opportunity (RO)** (i.e., a set of code fragments that should be refactored) and the corresponding procedure to perform refactoring (i.e., how to perform refactoring). However, the quality of each refactoring pattern is mostly never assessed. Therefore, it is difficult for developers to determine which refactoring patterns should be given priority.

In this paper, we propose two quality characteristics of RPs, and then describe a case study on assessing those quality characteristics.

II. Proposed quality characteristics of refactoring patterns

We introduce the following two quality characteristics of RP.

- **Number of ROs:** Because a lot of refactoring patterns exist and developers have only a limited time, it is desirable to choose RPs that have a lot of ROs.
- **Ease of Refactoring:** It means that ease of applying each RP to ROs in source code. When the ease of refactoring of a RP is high, it means that software systems involve a lot of ROs that can be easily performed refactoring. A RP that is difficult to apply often leads to time-consuming refactoring. Because the aim of refactoring is to reduce maintenance cost, time-consuming refactoring is not desirable. There are two kinds of refactoring pattern. The first one requires developers to apply only steps described in its description. On the other hand, another sometimes requires developers to apply not only steps described in its description but also additional steps.

III. Case Study

In this section, we assess the quality characteristics of RP which is named **Introduce Polymorphic Creation with Factory Method (IPCFM)** [2].

We introduce IPCFM and an automated method to identify ROs in software systems for IPCFM. Then, we discuss the ROs in several software systems from proposed quality characteristics of RP.

A. Introduce polymorphic creation with factory method

IPCFM is a kind of **Pull up Method** [1] pattern that is aimed at merging similar methods from different classes into a common superclass. Figure 1 shows an example of IPCFM. The aim of IPCFM is to merge similar methods except for object creation statements by introducing factory methods. An RO for IPCFM is defined as *"Classes in a hierarchy implement a method similarly except for an object creation step"* [2].

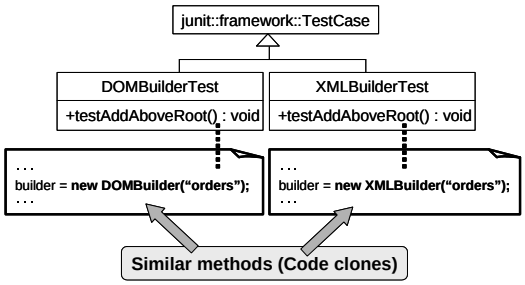
As shown in Figure 1(a), the targets of the refactoring are the test classes DOMBuilderTest and XMLBuilderTest for testing DOMBuilder and XMLBuilder, respectively. Because the target classes have similar methods except for an object creation step, they indicate an RO for applying IPCFM. This refactoring is comprised of following two steps.

Step1 As shown in Figure 1(b), a common superclass (AbstractBuilderTest) for the target classes is introduced, and similar methods in the target classes are merged into new method in the common superclass.

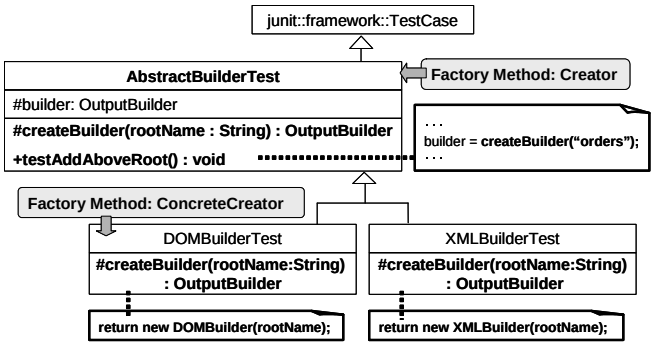
Step2 A factory method is introduced in each of the common superclass (AbstractBuilderTest) and the subclasses (DOMBuilderTest and XMLBuilderTest).

B. Assessment Method

For our case study, we have developed the tool that identifies ROs for the target RP by the steps below.



(a) Before refactoring



(b) After refactoring

Figure 1. Introduce Polymorphic Creation with Factory Method

Step1 Detect similar methods using a code clone detection tool CCFinder [3][1].

Step2 Evaluate whether detected methods belong to classes that have common superclasses in target source code and whether they include object creation statements.

We apply the target RP to the ROs in Ant and ANTLR. To assess the ease of refactoring, we confirm the steps that are not described in the description of the target RP.

C. Results

Table I includes the result of identifying ROs for IPCFM in several software systems. For comparison, in Table I, we show the number of ROs for Pull up Method (PM). We identify ROs for PM by detecting code clones belonging to classes that have common superclasses. We should note that because IPCFM is kind of PM, an RO for IPCFM is counted towards the number of ROs for PM. According to Table I, 17.9% of the ROs for PM are the ROs for IPCFM. We can say that ROs for PM includes more than few ROs for IPCFM. This indicates that when developers found RO for PM, they should inspect whether those RO are also for IPCFM.

When we apply IPCFM to all ROs in Ant and ANTLR, we did not have to apply additional steps that are not described in the description of IPCFM. This result indicates that the ease of refactoring of IPCFM is high.

[1]In our case study, we set 30 tokens as the minimum length of code clone.

IV. Related Works

Hsueh, et al.[4] and Huston[5] focus on the quality of design patterns. We focus on the quality of RPs, and propose the two novel quality characteristics of RPs.

V. Summary and Future work

In this paper, we proposed two quality characteristics of RP, and then described a case study on assessing those quality characteristics of IPCFM. To compare the quality characteristics of RP, we are planning to assess other RPs. We should discuss not only proposed quality characteristics but also change in maintainability because the aim of refactoring is to reduce maintenance cost.

Acknowledgment

We thank the anonymous SPAQu'09 reviewers for useful feedback on earlier versions of this paper. This reseach was supported by JSPS, Grant-in-Aid for Scientific Research (A) (No.21240002) and Grant-in-Aid for JSPS Fellows (No.20-1964).

References

[1] M. Fowler, *Refactoring: improving the design of existing code*. Addison Wesley, 1999.

[2] J. Kerievsky, *Refactoring to Patterns*. Addison Wesley, 2004.

[3] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Trans. Sofw. Eng.*, vol. 28, no. 7, pp. 654–670, 2002.

[4] N.-L. Hsueh, P.-H. Chu, and W. Chu, "A quantitative approach for evaluating the quality of design patterns," *Journal of Systems and Software*, vol. 81, no. 8, pp. 1430–1439, 2008.

[5] B. Huston, "The effects of design pattern application on metric scores," *Journal of Systems and Software*, vol. 58, no. 3, pp. 261–269, 2001.

Table I
Number of ROs for IPCFM

name	LOC	#classes	#opportunities	
			IPCFM	PM
Ant 1.7.0	198K	994	2	23
ANTLR 2.7.4	32K	167	1	33
Azureus 3.0.3.4	538K	2226	20	42
JEdit 4.3	168K	992	0	1
JHotDraw 7.0.9	90K	487	1	26
SableCC 3.2	35K	237	0	1
Soot 2.2.4	352K	2298	5	53
WALA 1.1	210K	1565	7	22