1ZK-4

コードクローン検出に基づくデザインパターン適用支援

吉田　昌友†　　吉田　則裕‡　　井上　克郎‡

大阪大学基礎工学部情報科学科†　　大阪大学大学院情報科学研究科‡

1. はじめに

デザインパターン[1]とは，オブジェクト指向プログラムの設計において発生する特定の問題に対して，過去の熟練プログラマが用いてきた典型的な解決策をいう．だが，実際のソフトウェア開発では，デザインパターンを利用すべき場合でも，利用していないことがある．

そのため，既存のソフトウェアの設計品質を高めることを目的に，デザインパターンを適用するリファクタリングが提唱されている[2]．**リファクタリング**とは，“外部的振る舞いを保ちつつ，理解や修正が容易になるように，ソフトウェアの内部構造を変化させること”[3]である．しかし，大規模ソフトウェアのソースコードからデザインパターンの適用が可能な部分を手作業で発見するには，大きなコストがかかる．

そこで，ソースコードからデザインパターンの適用が可能な部分を自動的に検出することで，既存のソフトウェアに対するデザインパターンの適用を支援する手法を提案する．具体的には，類似コードを除去するために Factory Method パターン[1]を適用するリファクタリングの支援を目的に，**コードクローン検出法**(類似コードを検出する手法)[4]を利用した手法の実現を行った．さらに，提案手法をオープンソースソフトウェアに適用する実験を行った．

2. デザインパターンを適用するリファクタリング

今回の実験では，“Factory Method によるポリモーフィックな生成の導入”[2]を支援した．

デザインパターン適用前は，兄弟クラス(共通の親クラスを持つクラス対(図 1 の**DOMBuilderTest**クラスと**XMLBuilderTest**クラス))に，オブジェクト生成文以外は内容が等しいメソッド(図 1(a)の2 つの**testAddAboveRoot**メソッド)が存在し，それらのメソッドが類似コードになっている．まず，それらのメソッド間で異なるオブジェクト生成文を，共通のシグニチャを持ち，オブジェクトを生成して返すメソッド(図 1(b)の**createBuilder**メソッド)に置換することで，類似コードになっているメソッドの内容を揃える．オブジェクトを生成して返すこのメソッドがFactory Method[1]である．次に，類似コードになっているメソッドを共通の親クラス(図 1 では新たに**AbstractBuilderTest**クラスを親クラスとして作成している)に引き上げる．最後に，Factory Methodを共通の親クラスに抽象メソッドとして宣言する．Factory Methodの実装を子クラスごとに変えていることになるので，Factory Methodパターンを適用した形になる．

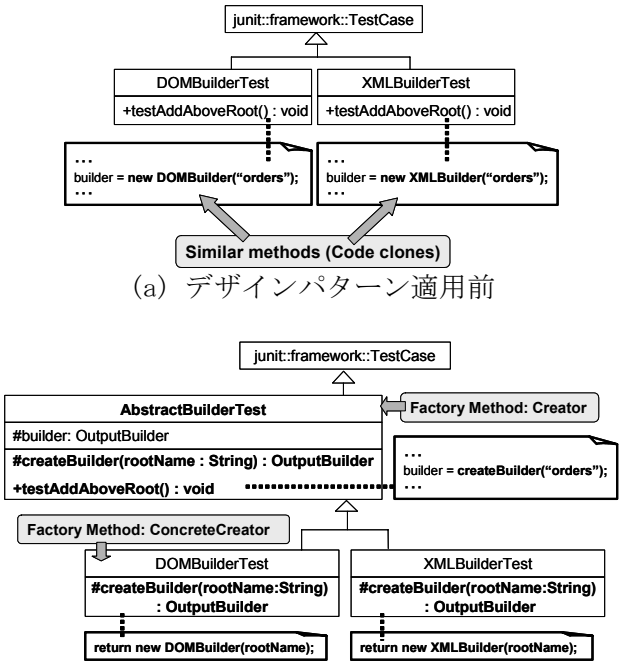

図 1　Factory Method によるポリモーフィックな生成の導入[2]

以上のように変更することで，兄弟クラスで重複していたメソッドを共通の親クラスにまとめることができ，類似コードを除去することができる(図 1(b))．また，**OutputBuilder** クラス(**DOMBuilder** クラスと **XMLBuilder** クラスの親クラス)に子クラスを追加して同様の処理を行う場合，デザインパターン適用前では重複するメソッドが増えてしまうが，デザインパターン適用後は Factory Method を新たに実装すればよい．

Design Pattern Application Support Based on Code Clone Detection
†Masatomo YOSHIDA ‡Norihiro YOSHIDA
‡Katsuro INOUE
†Department of Information and Computer Sciences, School of Engineering Science, Osaka University
‡Graduate School of Information Science and Technology, Osaka University

3. 提案手法

本稿では，ソースコードから図 1(a)の構造を持つ部分を自動的に検出することによって，“Factory Method によるポリモーフィックな生成の導入”を支援する手法を提案する．

まず，図 1(a)の構造を持つ部分を自動的に検出するために，自動的に判定できる条件を定める．前述の“Factory Method によるポリモーフィックな生成の導入”が掲載されている文献[2]を参考に，以下の条件を定めた．

条件 1. 類似コードになっているメソッドが兄弟クラス中に存在している．

条件 2. それらメソッドの内容で異なる部分は，オブジェクト生成文のみである．

条件 1.を定めた理由は，図 1 は“Template Methodの形成”[2]の特殊な事例だからである．兄弟クラス中の類似コードになっているメソッド(図 1(a)の 2 つの**testAddAboveRoot**メソッド)では，大部分の処理は共通しているが，一部分(オブジェクト生成文)が異なっている．その異なる部分を，共通のシグニチャを持つメソッド(図 1(b)の**createBuilder**メソッド)に置換すれば，類似コードになっているメソッドの内容が揃う．それらのメソッドを共通の親クラスに引き上げてまとめることで，類似コードを除去することができる．この引き上げたメソッド(図 1(b)の**AbstractBuilderTest**クラスにある**testAddAboveRoot**メソッド)がTemplate Method[1]である．条件 2.を定めた理由は，Factory Methodに置き換えることができるオブジェクト生成文を判定するためである．

次に，以上の条件を満たすコードを検出するため，以下の手順を定めた．

手順 1. コードクローン検出ツール CCFinder[4]を使い，類似コードになっているメソッドを検出する．CCFinder を使用する理由は，リファクタリングについての研究で利用されている実績があるためである[5]．

手順 2. 検出したメソッドが兄弟クラスに存在し，かつ，オブジェクト生成文を含むかを判定する．

4. 適用実験

Java コードを対象として，デザインパターンの適用が可能な部分を検出するツールを実装した．ツールは，コードクローン検出ツール CCFinder の出力ファイルとソースコードを入力とし，デザインパターンの適用が可能な部分を出力する．

更に，ANTLR2.7.4(オープンソースのパーサ生成ツール)に対し実験を行った．結果として，デザインパターンの適用が可能とツールが判定した部分は 10 個であった．出力結果と実験対象のソースコードとを参照したところ，デザインパターンの適用が可能と判断できた部分が 1 個あり，オブジェクト生成文が異なっていた．そこに含まれるコンストラクタのクラスに共通する親クラスかインタフェースが，デザインパターン適用前から存在していることが本来ならば望ましい．しかし，そうではなかったので，インタフェースを新たに作成すれば，リファクタリングを行うことができると考えられる．また，オブジェクト生成文が一致したため，デザインパターンの適用は可能だが，“メソッドの引き上げ”[3]を行うのが適切であると判断できた部分が 9 個あった．

5. まとめと今後の課題

本稿では，デザインパターンの適用を支援する手法の提案と，手法に基づき実装したツールの適用実験の結果を述べた．

今後の課題は，実際にデザインパターンをソースコードに適用し，提案手法の有効性を確認することである．また，今回支援した“Factory Method によるポリモーフィックな生成の導入”とは違うデザインパターン(例えば“Compositeによる単数・複数別の処理の置き換え”[2])の適用を支援することも課題である．

謝辞 本研究は一部，文部科学省リーディングプロジェクト「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた．

文献

[1] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addition Wesley, 1995.

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

[3] M. Fowler. *Refactoring: improving the Design of Existing Code*. Addition Wesley, 1999.

[4] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, 28(7):654-670, 2002.

[5] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue. Method and Implementation for Investigating Code Clones in a Software System. *Information and Software Technology*, 49(9-10):985-998, 2007.