コードクローン間の依存関係に基づくリファクタリング支援環境の実装

吉田　則裕[†]　　肥後　芳樹[†]　　神谷　年洋[‡]　　楠本　真二[†]　　井上　克郎[†]

大阪大学大学院情報科学研究科[†]　　科学技術振興機構　さきがけ[‡]

1. はじめに

コードクローンとは，ソースコード中に存在する同一，または類似したコード片のことであり，ソフトウェアの保守を困難にしている要因の一つとされている．そのため，コードクローンは集約（リファクタリング[3]）を検討するべきものであると指摘されている[1][2].

我々の研究グループでは，コードクローン検出ツール CCFinder[4]と，リファクタリング支援環境 Aries[5]の開発を行ってきた．Aries では，各コードクローンに対して個別にリファクタリング支援を行う機能が実現されている．

本稿では，より効率的にリファクタリングを行うために，ソフトウェア中に含まれるコードクローンをグループ分けし，グループ単位でのリファクタリング支援を行う手法を提案する．具体的には，メソッド呼び出し関係のあるコードクローンを一つのグループにまとめ，グループに対しメトリクスを用いて特徴づけを行うことにより，適用可能なリファクタリングパターンを提示する．

2. 準備

2.1. コードクローンの定義

あるトークン列中に存在する二つの部分トークン列（α，β） が等価であるとき，（α，β）は互いにクローンであるという．またペア（α，β）をクローンペアと呼ぶ．（α，β）それぞれを真に包含する如何なるトークン列も等価でないとき，（α，β）を極大クローンと呼ぶ．また，クローンの同値類をクローンセットと呼ぶ．ソースコード中でのクローンを特にコードクローンという．

2.2. CCFinder

CCFinder は，単一又は複数のソースコード中から極大クローンを検出する．この検出は数百万行規模のソフトウェアであっても実用時間で可能である．ここにおけるクローンとは，ソースコードを字句解析することにより得られたトークン列に対し，名前空間の正規化等の変換を行った結果，トークン単位で等価である部分をいう．

2.3. Aries

Aries は，クローンセットに対し，様々なメトリクスを用いた特徴付けを行うことにより，適用すべきリファクタリングパターン[3]の決定支援を行う．また，CCFinder が検出したクローンのうち，メソッドや繰り返し文といった，リファクタリングの対象となりうるプログラミング言語の構造単位でクローンとなっているもののみを扱う．

3. 提案手法

3.1. 概要

Aries では，各クローンセットに対して，個別にリファクタリングを行う必要がある．しかし，クローン間に何らかの依存関係がある場合には，個別に行うリファクタリングでは，作業が困難である，効率が良くないなどの問題がある．そこで，本稿ではメソッド呼び出し関係のあるクローンをまとめてリファクタリングを行う手法を提案する． まず，メソッド呼び出し関係のあるクローンを表す Chained Clone を定義し，その後，Chained Clone に対するリファクタリング支援を目的としたメトリクスを二つ定義する．

3.2. Chained Clone

メソッド間の呼び出し関係を，プログラム依存グラフと同じく有向グラフを用いて表現する．つまり，頂点（メソッド）の集合と辺（メソッド間の呼び出し関係）の集合を用いて表す．本稿では，この二つの集合の組で表される有向グラフを Chained Method と呼ぶ．

以下の条件が成立するとき，二つの Chained Method CM_A と CM_B は互いに Chained Clone であると定義する．

(1) CM_A と CM_B は同形グラフである．

(2) (1)において，対応する頂点（メソッド）は，互いにクローンである．

また，Chained Clone の同値類を Chained Clone Set と呼ぶ．

図１に例を示す．同形である三つのグラフ CM_1,CM_2,CM_3 はそれぞれ Chained Method を表している．ここで，対応する頂点（M_{A1} と M_{B1}，M_{B2} と M_{C2} など）は，互いにクローンになっているとす

Refactoring support tool based on code clone dependency

†Norihiro YOSHIDA, † Yoshiki HIGO, †Shinji KUSUMOTO, ‡Toshihiro KAMIYA, †Katsuro INOUE

†Graduate School of Information Science and Technology, Osaka University

‡PRESTO, Japan Science and Technology Agency

ると，各グラフは，他の二つのグラフとの間で前述の Chained Clone である条件を満たしている．よって，これら三つの Chained Method CM_1,CM_2,CM_3 は一つの Chained Clone Set を構成している．

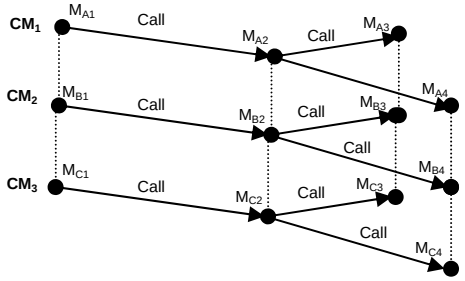

図 1 Chained Clone Set の例

3.3. Chained Clone Set に対するメトリクス

Chained Clone に対し“Pull Up Method”を適用する際に有用なメトリクスを提案する．コードクローンに対する“Pull Up Method”とは，クローンとなっているメソッドを共通の親クラスへ引き上げることによって，集約を行うリファクタリングパターンである．

Chained Clone に対し“Pull Up Method”を適用する際に，二つの事柄を考慮する必要がある．一つ目は，リファクタリングを行う単位である．つまり，Chained Clone 全体を一つの親クラスに引き上げるのか，それとも互いにクローンになっているメソッドをそれぞれが継承する共通の親クラスへ引き上げるかを決定する必要がある．二つ目は，既存の親クラスに引き上げるか，それとも新たに共通の親クラスを作るかを決定する必要がある．

これらの判断には，次の二つの情報が必要である．一つ目は，各 Chained Method に含まれるメソッドが同じクラスに存在するかどうかである．同じクラスに存在すれば，容易に一つのクラスに引き上げることができる． 二つ目は，図 1 の M_{A1} と M_{B1} のように Chained Clone 内で，互いにクローンになっているメソッドが共通の親クラスを持つかどうかある．共通の親クラスを持つ場合は容易に引き上げることができるが，持たない場合は新たな共通の親クラスの作成を検討する必要がある．

これらを踏まえ，一つ目の情報を表す DCHD(Dispersion of Class Hierarchy in Dependency fragments)メトリクスと二つ目の情報を表す DCHS(Dispersion of Class Hierarchy in Subset of clone class)メトリクスを定義した．Chained Clone Set S は，n 個の Chained Method CM_1,CM_2,…,CM_n を含んでいるとする．また，Chained Method CM_i は，m 個のメソッド M_{i1},M_{i2},…,M_{im} をそれぞれ含んでいる．また，メソッド M_{ij} は，CM_i に含まれる j 番目の頂点に相当するとする．更に，各 Chained Clone の j 番目のメソッドの集合 $MS_j=\{M_{0j},M_{1j},\cdots,M_{nj}\}$ とする．この集合に含まれるメソッドは，同一のクローンセットに含まれる．また，DCH は Aries で用いたメトリクスであり，各コード片共通の親クラスを求め，そのクラスと各コード片とのクラス階層における距離のうち最大のものを値とする．このとき，DCHD(S)と DCHS(S)は次の式で表される．

$$DCHD(S)=\max\{DCH(CM_1),DCH,(CM_2),...,DCH(CM_n)\}$$
$$DCHS(S)=\max\{DCH(MS_1),DCH,(MS_2),...,DCH(MS_m)\}$$

4. 適用実験

Java コードを対象として，Chained Clone の検出とメトリクスを計測するツールを実装した．更に，ANTLR 2.7.4 に対し，適用実験を行った．その結果，18 個の Chained Clone Set を検出した．その一つは CSharpCodeGenerator，JavaCodeGenerator の両クラスに含まれる呼び出し関係がある二つのメソッド mangleLiteral および getValueString で構成されていた．これらのメソッドは，同名のメソッド間でそれぞれクローンになっていた．更に，各メトリクスを計測すると，DCHS が 1，DCHD が 0 であった．この Chained Clone Set に対し，“Pull Up Method”の適用を試みたところ，アクセス制御子を変更するのみで適用することができた．

5. まとめと今後の課題

本稿では，Chained Clone を定義し，それに対するリファクタリング支援を目的とした二つのメトリクスを定義した．また，それらメトリクスの評価実験を行った．他の依存関係の利用が今後の課題である．

文献

[1] R. Komondoor and S. Horwitz, “Using Slicing to Identify Duplication in Source Code”, *Proc. SAS2001*, 2001, 40-56.

[2] J. Krinke, “Identifying similar code with program dependence graphs”, *Proc. WCRE2001,* 2001, 301-309.

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

[4] T. Kamiya et al. “CCFinder: A multilinguistic token-based code clone detection system for large scale source code”, *IEEE Trans. on Softw. Eng.*, 2002, 28(7):654-670.

[5] Y. Higo et al. “ARIES: Refactoring Support Environment Based on Code Clone Analysis”, *Proc. SEA2004,* 2004, 222-229.