コードクローン解析に基づくリファクタリング支援

肥後　芳樹[†]　　神谷　年洋[††]　　楠本　真二[†]　　井上　克郎[†]

あらまし　近年，コードクローンがソフトウェア保守を困難にしている一要因といわれている．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る．例えば，あるコード片にバグが含まれていた場合，そのコード片のコードクローン全てについて修正の是非を考慮する必要がある．このような理由によりソフトウェアからコードクローンを取り除くことはソフトウェアの保守性や複雑度などの面からみて有効である．本稿では，大規模ソフトウェアに対しても適用可能なコードクローンの集約支援手法の提案を行う．

キーワード　リファクタリング，コードクローン，ソフトウェア保守

1. まえがき

本稿では，実用的な時間でソースコード中から集約に適したコードクローンを検出し，さらに，検出したコードクローンの特徴をメトリクスを用いて定量化する手法を提案する．そして，提案手法に基づき，リファクタリング支援環境 Aries の試作を行なう．

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 2 つの部分トークン列 α，β が等価であるとき，α と 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 [2] はプログラムのソースコード中に存在する極大クローンを検出し，その位置をクローンペアのリストとして出力する．検出されるコードクローンの最小トークン数はユーザが前もって設定できる．

3. 提 案 手 法

本稿では CCFinder の検出したコードクローンに対して集約支援を行う手法を提案する．まず第一ステップとして，CCFinder の検出したコードクローンから，集約に適した部分の抽出を行う．次に第二ステップとして，抽出したコードクローンをメトリクスを用いて定量的に特徴づけを行い，適用可能なリファクタリングパターンの決定支援を行う．

† 大阪大学大学院情報科学研究科，大阪府
Graduate School of Information Science and Technology, Osaka University, Toyonaka, 560-8531 Japan
†† 科学技術振興機構 さきがけ，埼玉県
PRESTO, Japan Science and Technology Agency, 4-1-8, Honmachi, Kawaguchi, Saitama, 332-8531 Japan

3.1 集約に適したコードクローンの抽出

CCFinder はその性質上，集約に適していないコードクローンも多く検出する．そのようなコードクローンを取り除くため，プログラミング言語における構造的なまとまりを持った部分のみを，集約に適したコードクローンとして抽出する．図 1 はその例を示している．図 1 では，A と B の 2 つのコード片が示されている．A と B それぞれの灰色の部分は，その部分が A と B の間の最大長のコードクローンであることを示している．本手法ではこのような場合，灰色で示されたコードクローンから構造的なまとまりを持った部分，つまり for 文の部分のみを抽出する．

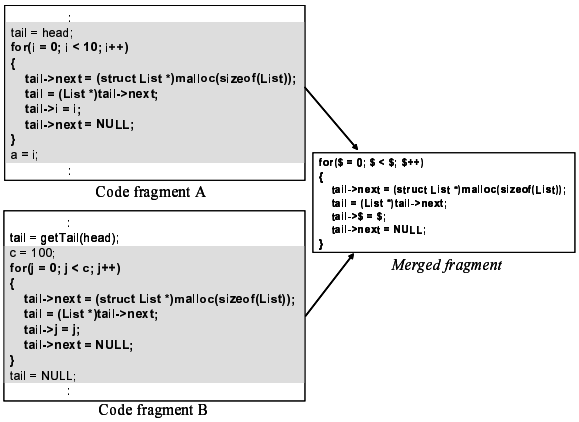

図 1　コードクローン集約の例

3.2 メトリクスを用いた特徴づけ

次に，3.1 節で抽出したコードクローンがどのように集約が可能であるかメトリクスを用いて判定する．これまでに様々なリファクタリングパターン [1] が提案されている．本手法では既存のパターンを用いたコードクローンの集約支援を行う．これらのパターンを用いた支援を “Extract Method” と “Pull Up Method” を例にとって考える．

“Extract Method” は，本来は長過ぎるメソッドや，複雑な処理の一部分に対して適用することによって，

コードの可読性，保守性を向上させることができる．しかし，コードクローンに対して適用することにより，重複したコード片を集約することも可能である．コード片を新たなメソッドとして再定義することになるため，抽出部分は周囲との結合度が低いことが望ましい．抽出部分とその周囲の結合度を計測するために **NRV(S)**(the Number of Referred Variables) と **NSV(S)**(the Number of Substituted Variables) の２つのメトリクスを定義した．ここでは，クローンセット S は n 個のコード片 $f_1,\ f_2,\ \cdots,\ f_n$ を含んでおり，コード片 f_i では s_i 個の外部定義の変数を参照しており，t_i 個の外部定義の変数に対して代入を行なっているとする．この時 $NRV(S)$ と $NSV(S)$ はそれぞれ次の式で表される．

$$NRV(S) = \frac{1}{n}\sum_{i=1}^{n} s_i, \qquad NSV(S) = \frac{1}{n}\sum_{i=1}^{n} t_i,$$

次に，“Pull Up Method” を例にとって考える．“Pull Up Method” とは，ある親子クラス関係が存在した場合に，子クラスに存在するメソッドを親クラスに引き上げることである．共通の親クラスを持つ複数の子クラスに定義された重複したメソッドを引き上げることによって集約を行なうことが可能である．つまり重複したメソッドを含むクラスは共通の親クラスを継承している必要がある．そのため，クローンセットのクラス階層内における位置関係を計測する．これについては，メトリクス **DCH(S)**(the Dispersion on Class Hierarchy) を定義する．すでに示したように，クローンセット S はコード片 $f_1,\ f_2,\ \cdots,\ f_n$ を含んでいるとする．また C_i はコード片 f_i を含んでいるクラスを表す．もしクラス $C_1,\ C_2,\ \cdots,\ C_n$ が共通の親クラスを持つ場合は，その共通の親クラスの中で，最もクラス階層的に下位に位置するクラスを C_p で表すとする．また $D(C_k,\ C_h)$ はクラス C_k と C_h のクラス階層における距離を表すとする．この時，

$$DCH(S) = max\ \{D(C_1,\ C_p),\ \cdots,\ D(C_n,\ C_p)\}$$

と表される．例えば，クローンセット S 中の全てのコード片が１つのクラス内に存在する場合は $DCH(S)$ の値は 0，あるクラスとその直接の子クラス内に存在する場合は $DCH(S)$ の値は 1 となる．例外的に，コードクローンが存在するクラスが共通の親クラスを持たない場合は $DCH(S)$ の値は ∞ とする．

3.3 リファクタリング支援環境: Aries

提案手法を，リファクタリング支援環境 **Aries** として実装した．現在のところ対象は Java 言語としている．Java 言語を対象としているため抽出する構造的なまとまりは以下の 12 種類である．

宣言	:	class { }, interface { }
メソッド	:	メソッド本体, コンストラクタ, スタティックイニシャライザ
文	:	if, for, while, do, switch, try, synchronized

3.4 メトリクスを用いた絞り込み

Aries を用いてのクローンの絞り込みの例を示す．

(1) “Pull Up Method”

例えば，以下のような条件が考えられる．

(PC1) 対象となる単位はメソッド本体,

(PC2) $DCH(S)$ の値が 1 以上.

“Pull Up Method” はメソッドが対象であるので，条件 (PC1) が必要である．また，重複したメソッドを含むクラスが共通の親クラスを継承している必要があることから条件 (PC2) が必要である．

(2)“Extract Method”

“Extract Method” を行なう際の条件としては例えば，以下のものが上げられる．

(EC1) 対象となる単位は文単位,

(EC2) $DCH(S)$ の値が 0,

(EC3) $NSV(S)$ の値が 1 以下,

“Extract Method” とはメソッド内のコード片に対して適用されるので，(EC1) が必要である．また，全てのコードクローンが同一のクラス内に存在する場合は容易に集約が可能であるので，条件 (EC2) を考慮している．コードクローンの内部において，外部定義変数に対して代入を行なっている場合は，その変数を引数として与え，返り値として返し，メソッドの呼び出し元に反映させなければならない．このような変数が複数あった場合は新たなデータクラスを定義し，そのオブジェクトを介して値を受け渡す必要がある．もしこのような変数が１つの場合は単に return 文を用いて返すだけで良く，容易に集約を行なうことができるので，条件 (EC3) を考慮している．

4. ま と め

本稿では，コードクローンを対象とした集約支援手法を提案した．また，提案手法をリファクタリング支援環境 Aries として実装した．しかし，現在の解析はリファクタリングの可能性について述べているが，積極的にすべきかの判断はしていない．今後は，ソフトウェアの品質の面からリファクタリングの是非を判断するように拡張を行なう予定である．

文　献

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

[2] T. Kamiya, S. Kusumoto, and K. Inoue, *CCFinder: A multi-linguistic token-based code clone detection system for large scale source code* IEEE Transactions on Software Engineering, vol.28, no.7, pp.654-670, Jul. 2002.