# 差分を含む類似メソッドの集約支援ツール

後藤 祥[1]　吉田 則裕[2]　井岡 正和[1]　井上 克郎[1]

**概要**：ソースコード中で互いに一致または類似したメソッドを類似メソッドという．可読性や保守性の向上を目的として，類似メソッドを 1 つに集約する作業がよく行われるが，類似メソッド間に差分が存在する場合，その集約は難しい．そこで，本研究では，差分を含む類似メソッドの集約作業を，集約候補を提示することで支援する手法を提案する．また，提案手法を統合開発環境 Eclipse のプラグインとして実装した．実験では，オープンソースソフトウェア上の類似メソッドに対して提案手法を適用した．そして，適用結果をもとにアンケートを行い，結果に有用な集約候補が含まれていることを確認した．

# A Tool Support to Merge Similer Methods with Differences

Akira Goto[1]　Norihiro Yoshida[2]　Masakazu Ioka[1]　Katsuro Inoue[1]

***Abstract:*** A method that has identical or similar methods is called similar method. Merging similar methods into a single method is often performed to improve the maintainability of a program. However, it is difficult for a developer to merge similar methods with differences. In this paper, we propose an approach that supports to merge similar methods with differences by showing candidates of merging similar methods. The proposed approach has been implemented as a plugin of Eclipse. In the case study, we applied the proposed approach to actual similar methods in open source projects. As a result, we confirmed useful candidates in the output of the proposed approach by the questionnaire.

## 1. まえがき

ソフトウェアの保守を困難にしている要因としてコードクローンが挙げられる．コードクローンとは，ソースコード中で互いに一致または類似した部分をもつコード片のことである [11]．特に，メソッド単位のコードクローンを類似メソッドと呼び，類似メソッドとなっている 2 つのメソッドの組を類似メソッド対と呼ぶ．

ソースコードの保守性や可読性の向上を目的として，リファクタリングによる類似メソッドの集約がよく行われる [2]．類似メソッドの集約に有効なリファクタリングとして，Template Method の形成が挙げられる．Template Method の形成は，共通の親クラスを持つクラスに存在する類似メソッドを集約するためのリファクタリングであり，類似メソッド中の差分を含む処理は各子クラスで実装し，共通部分を親クラスに集約する [2]．Template Method の形成では，まず子クラスにメソッドとして抽出するコード片を決定するが，このコード片はメソッドとして抽出可能であることや，抽出後に類似メソッド対が完全に一致することなどの 4 つの条件を満たす必要がある (2 節定義 2 参照)．しかし，リファクタリングの経験が十分でない開発者にとって，この条件を満たすコード片を探すのは困難である．

そこで本研究では，リファクタリング作業者が選択した類似メソッド対に対して，前述の 4 条件を満たすコード片の集合 (集約候補) の一覧を提示することで，リファクタリング作業を支援する手法を提案する (**図 1**)．提案手法では，作業者が選択した類似メソッド対を入力として (図 1 中の 1)，類似メソッド対の抽象構文木 (AST) を比較することで，集約候補を検出する (図 1 中の 2，3)．また，開発者にとって有用な集約候補から順に提示するために，検出された集約候補の並び替えを行い (図 1 中の 4)，集約候補一覧を作業者へと提示する (図 1 中の 5)．提案手法では，リファクタリングによって子クラスに作成されるメソッドの凝集度が高いものを，開発者にとって有用な集約候補

[1] 大阪大学
Osaka University
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

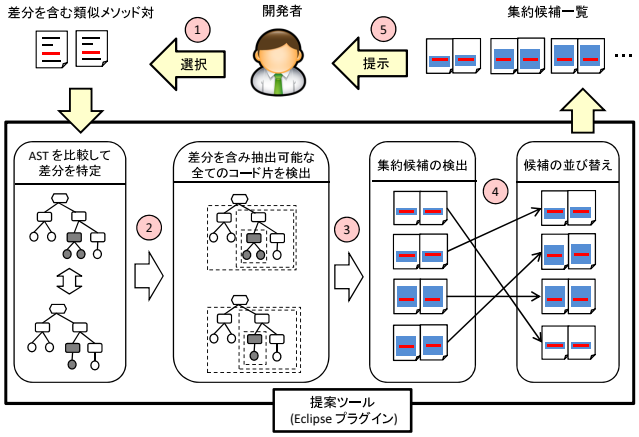


**図 1** 提案手法の概要
**Fig. 1** Overview of the Proposed Approach

と考えて並び替えを行う．凝集度の計測には，プログラムスライスを用いた凝集度メトリクスを使用した．提案手法は，統合開発環境 Eclipse のプラグインとして実装した．

実験として，オープンソースソフトウェア上の類似メソッド対に提案手法を適用した．また，適用結果をもとにアンケートを行い，手法の評価を行った．評価の結果，提案手法によって上位に提示された集約候補に，開発者にとって有用な集約候補が含まれていることがわかった．

## 2. 集約候補の検出

提案手法では，まず与えられた類似メソッド対 $M_A$ と $M_B$ に対して，それらの集約候補を検出する．集約候補の検出は，AST を用いた類似メソッド間の差分検出と，差分を含み，メソッド抽出可能なコード片の検出の 2 つのステップで行われる．以下に，提案手法におけるコード片の定義を示す．

**定義 1（コード片）** 本稿では，コード片を (ファイルを一意に識別できる番号，開始行番号，開始桁数，終了行番号，終了桁数) という 5 項組とする．

### 2.1 抽象構文木の構築

入力として与えられた類似メソッド対 $M_A$ と $M_B$ の AST を構築する．提案手法では Eclipse JDT を使用して類似メソッド対の AST を構築している．構築された AST の各ノードはラベルを持っており，ラベルはタイプ (Assginment や Expression など) または値 (変数や定数など) を表している．

提案手法では，AST とソースコード中の文の対応をとるために，AST における特殊ノードを定義する．特殊ノードとは，ソースコード中の 1 つの文を表しているノードであり，式文 (ExpressionStatement) やリターン文 (ReturnStatement) などのタイプを持つノードが該当する．そして，以降の処理のために，特殊ノードに対して，根ノードからの幅優先探索で到達した順番に番号を付ける．**図 2** に

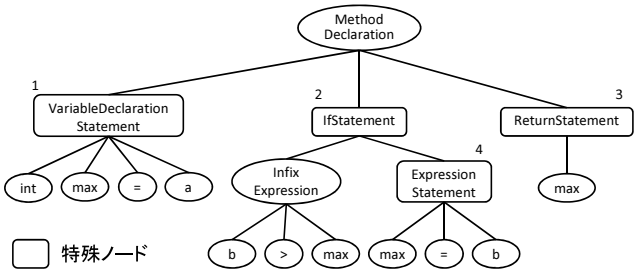


**図 2** AST 中の特殊ノードへの番号付けの例
**Fig. 2** An Example of Indexing for Particular Nodes in AST

AST と特殊ノードへの番号付けの例を示す．図 2 の AST は，説明のためにいくつかのノードを省略して簡単化したものである．

### 2.2 類似メソッド間の差分検出

差分の検出では，まず AST の根ノードから深さ優先探索の順にノードの比較を行い，AST 上で異なっているノードを特定する．2 つのノード $N_A$ と $N_B$ の比較は以下の手順で行われる．

**(1) ラベルの比較** $N_A$，$N_B$ のラベルを比較し，異なる場合は，$N_A$，$N_B$ を AST 上の差分とする．等しい場合は，$N_A$，$N_B$ に子ノードが存在しなければ比較を終了し，存在すれば子ノードに対して再帰的に比較を行う．子ノードの比較方法は $N_A$，$N_B$ のタイプによって異なる．$N_A$，$N_B$ のタイプが Block タイプでない場合は各子ノードに対して再帰的に手順 (1) から比較を行う．Block タイプの場合は手順 (2) の比較を行う．

**(2) Block タイプの場合の子ノードの比較** Block タイプのノードはプログラム中の中括弧で囲まれた文の列に対応するノードであり，中括弧内の文の数によって子ノードの数が変化する．そのため，Block タイプのノードの子ノードに対しては，動的計画法を用いた類似文字列マッチング [1] を用いて，子ノード中の一致している部分と差分を検出する．類似文字列マッチングを使用することで，単純に子ノード列を比較するより差分を少なくすることができる．

以上の比較の操作を行うことによって，AST 上で差分となっているノードを特定する．次に，差分となっているノードから，親ノードを辿っていき，最初に到達する特殊ノードを探索する．この操作を行うことで，差分となっているノードが，ソースコード中のどの文に含まれているかを特定できる．

**図 3** はここまでの操作で検出される差分の例である．AST のノードの比較と，親ノードを辿って最初に到達する特殊ノードを求めることで，ソースコード中で差分となっている文に対応した部分木 (図 3 中の破線で囲まれた部分) を特定する．ここまでの操作で得られる差分となっている部分木は，全て特殊ノードを根とする部分木である．

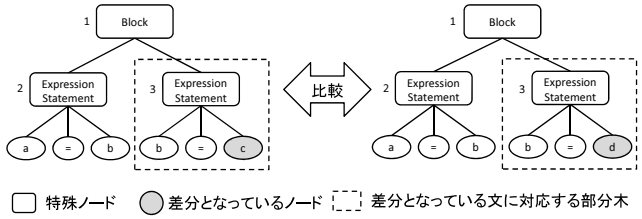


**図 3** 類似メソッド間で差分となっている文の検出
**Fig. 3** Detecting Statement-level Differences between Similar Methods

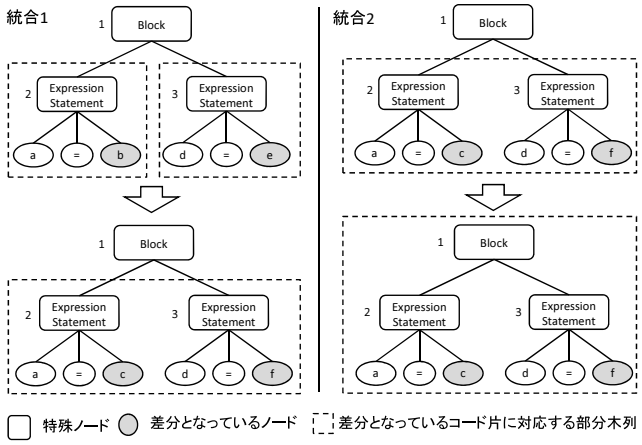


**図 4** 差分の統合操作
**Fig. 4** Merging Differences

次に，特定の条件を満たす差分を統合する．最終的に検出される差分となっているコード片の数を，統合によって少なくすることで，メソッド抽出可能なコード片の検出 (2.3 節参照) の処理を容易にすることができる．差分を統合する操作は以下の 2 つであり，これらを適用できる箇所がなくなるまで行う．**図 4** は各操作の例を示している．

**統合 1 (隣接した差分の統合)** 差分となっている番号 $i$ の特殊ノードに対して，その兄弟ノードの中で $(i+1)$ または $(i-1)$ 番の特殊ノードが存在して，その特殊ノードも差分となっている場合は 2 つの差分を統合して 1 つの差分とする．

**統合 2 (Block 内の差分の統合)** Block タイプのノードについて，その全ての子が差分となっている場合に，Block タイプのノードも差分に含める．

以上の操作によって類似メソッド対 $M_A$ と $M_B$ の差分となっているコード片の集合を特定する．以後，これらの差分となっているコード片の集合を $(\Delta_A, \Delta_B)$ とする．$\delta_{Ai} \in \Delta_A$ はメソッド $M_A$ 中の差分となっているコード片で，メソッド $M_B$ 中の差分となっているコード片 $\delta_{Bi} \in \Delta_B$ と対応関係にある．

### 2.3 メソッド抽出可能なコード片の検出

前の処理で特定した，差分となっているコード片の集合 $(\Delta_A, \Delta_B)$ をもとに，差分を含み，メソッド抽出可能なコード片の集合を検出する．そして，その結果をもとにして，類似メソッド対の集約候補を検出する．類似メソッド対の集約候補の定義を以下に示す．

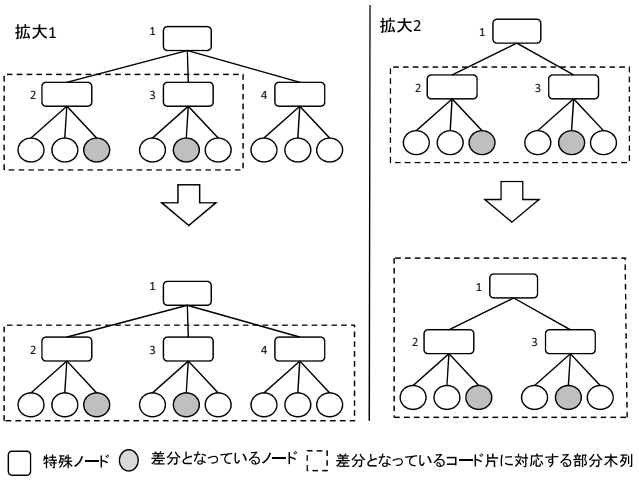


**図 5** コード片に対する拡大操作
**Fig. 5** Expansion for Code Fragments

**定義 2（類似メソッドの集約候補）** 類似メソッド対 $M_A$ と $M_B$ の差分となっているコード片の集合 $\Delta_A, \Delta_B (|\Delta_A| = |\Delta_B| = n)$ に対して，以下の条件を満たす類似メソッド中のコード片の集合 $E_A, E_B (|E_A| = |E_B| = m, 1 \leq m \leq n)$ を，類似メソッド対の集約候補という．

**条件 A** $E_A, E_B$ 中の全てのコード片はメソッド抽出可能である[*1]．

**条件 B** $E_A, E_B$ 中の全てのコード片をメソッドとして抽出した後，類似メソッド対のトークン列が完全に一致する．

**条件 C** 任意のコード片 $e_{Ai}, e_{Aj} \in E_A (i \neq j)$ のトークン列が互いに重複していない ($E_B$ についても同様)．

**条件 D** 差分となっている任意のコード片 $\delta_{Ai} \in \Delta_A$ に対して，それを含むコード片 $e_{Aj} \in E_A$ がただ 1 つ存在する ($\Delta_B$ と $E_B$ についても同様)．

まず，差分となっている各コード片 $\delta$ に対して，$\delta$ を含み，メソッド抽出可能な全てのコード片の集合 $include(\delta)$ を求める．$include(\delta)$ は，コード片 $\delta$ に対応する AST の部分木列に対して，AST 上での拡大とメソッド抽出可能かどうかの判定を，繰り返し行うことにより求める．拡大の操作は，コード片 $cf$ の初期値を $\delta$ として以下の手順で行われる．**図 5** は拡大の操作の例を示している．

**拡大 1 (兄弟ノード間での拡大)** コード片 $cf$ に対応する AST の部分木列の中で，最も大きい部分木の根ノード (特殊ノード) の番号を $i_{max}$，最も小さい番号を $i_{min}$ とする．$(i_{max}+1)$ または $(i_{min}-1)$ 番の特殊ノード

[*1] 本研究では，コード片がメソッド抽出可能であるかを，文献 [5] の Figure.2 に示された条件を用いて判定している

が兄弟ノードならば，そのノードを根とする部分木に対応したコード片を $cf$ に追加する．拡大後のコード片 $cf$ が抽出可能であれば， $include(\delta)$ に $cf$ を追加する．拡大の後，兄弟ノードに $cf$ に含まれていない特殊ノードが存在すれば，兄弟ノード間での拡大を続ける．存在しなければ，拡大 2 の操作を行う．

**拡大 2 (親ノードへの拡大)** コード片 $cf$ に対応する AST の部分木列中の根ノードから，親ノードを辿っていき，最も近い特殊ノードまで部分木を拡大する．拡大後の部分木に対応するコード片を新たな $cf$ とする．拡大後のコード片 $cf$ が抽出可能であれば $include(\delta)$ に $cf$ を追加する．もし，メソッド宣言を表す “Method Declaration” タイプのノードに到達したら，拡大の処理を終了する．そうでなければ，拡大 1 に戻り兄弟ノード間の拡大を行う．

上記の拡大の操作を $\Delta_A, \Delta_B$ 中の全ての差分に対して行い $\{include(\delta_{A1}), include(\delta_{A2}), \cdots, include(\delta_{An})\}$ と $\{include(\delta_{B1}), include(\delta_{B2}), \cdots, include(\delta_{Bn})\}$ を求める．そして，各 $include(\delta)$ から要素を抽出して，コード片の集合を作る．これらのコード片の集合のうち，上記の集約候補の定義を満たすものを全て列挙することで，類似メソッド対の全ての集約候補を検出する．

さらに，検出された集約候補に対してフィルタリングを行う．フィルタリングには，対象としている類似メソッド対の文の数と，集約候補中の各コード片の文の数の比を用いる．フィルタリングの式を以下に示す．

$$\frac{len(e_{Ai})}{len(M_A)} > threshold, \frac{len(e_{Bi})}{len(M_B)} > threshold$$

式中の $len(x)$ は $x$ の文の数を表している．もし，集約候補中に上記の式を満たす $e_{Ai} \in E_A$ または，$e_{Bi} \in E_B$ が 1 つでも存在すれば，その集約候補は出力する候補から除外する．これは，広い範囲を子クラスに抽出すると，その抽出したメソッドが再び類似メソッドになるためである．

## 3. 集約候補の並び替え

検出された候補をそのまま提示すると，候補数が膨大になった場合に，開発者の候補選択作業が困難となる．そこで，提案手法では集約候補の並び変えを行い，開発者にとって有用と思われる集約候補から順に提示する．Template Method の形成を行うと，集約候補中の各コード片は子クラスにメソッドとして抽出される．提案手法では，これらの新たに作成されるメソッドの凝集度が高いものが良い集約候補であると考え，集約候補中のコード片の凝集度が高いものから順に提示するように並び替えを行う．凝集度とはモジュール内の構成要素が特定の機能を実現するため協調している度合であり，一般的に凝集度が高いメソッドは可読性や保守性に優れている [7].

凝集度の計測には，プログラムスライスを用いた凝集度メトリクスを使用した [6], [8]．既存のメトリクスでは，返り値を起点とした後ろ向きスライスを用いているが，提案手法では，引数を起点とした前向きスライスも使用する．提案手法では，返り値がないメソッドに対しても凝集度の計算を行う必要があるため，このように定義を修正した．

提案手法で使用しているメトリクスの定義を以下に示す．式において，$M$ をメソッド，$len(M)$ を $M$ の文の数，$V$ を $M$ における引数と返り値の集合，$V_i$ を $M$ における引数の集合，$V_o$ を $M$ における返り値の集合，$FSL_x$ を変数 $x$ を起点にした前向きスライス，$BSL_x$ を変数 $x$ を起点にした後ろ向きスライス，$SL_{int}$ を $V_i$ 中の全変数に対する前向きスライスと $V_o$ 中の全変数に対する後ろ向きスライスの積集合とする．

$$FTightness(M) = \frac{|SL_{int}|}{len(M)}$$

$$FCoverage(M) = \frac{1}{|V|}\left(\sum_{x \in V_i} \frac{|FSL_x|}{len(M)} + \sum_{x \in V_o} \frac{|BSL_x|}{len(M)}\right)$$

$$FOverlap(M) = \frac{1}{|V|}\left(\sum_{x \in V_i} \frac{|SL_{int}|}{|FSL_x|} + \sum_{x \in V_o} \frac{|SL_{int}|}{|BSL_x|}\right)$$

集約候補の並び替えの処理では，まず集約候補中の各コード片をメソッド抽出したとして，凝集度を計算する．そして，集約候補中の全てのコード片の凝集度を計算し，それらの平均値を集約候補の凝集度とする．そして，凝集度の高い順に集約候補の並び替えを行い，開発者へと提示する．提案手法では 3 つのメトリクスをそれぞれ独立に使用して，3 つのランキングを生成する．

## 4. 実装

提案手法を統合開発環境 Eclipse 上で利用できるようにプラグインとして実装した．**図 6** は実装したツールのスクリーンショットである．図 6 において，差分を含む類似メソッド対がそれぞれ左右に表示されている．画面上部の 1 つのタブが 1 つの集約候補を表しており，タブを切り替えることで他の集約候補を見ることができる．集約候補中の各コード片は，背景色がついている部分であり，左右のメソッドで背景色が同じ部分が対応関係にある．また，画面下部のボタンで，並び替えに使用するメトリクスを変更することができる．

## 5. 適用実験

オープンソースソフトウェア上の類似メソッド対に対して，提案手法を適用した．類似メソッド対は Ant と ANTLR の 2 つのプロジェクトから，1 つずつ選択した[*2].

[*2] Ant : executeDrawOperation メソッド (Arc,Ellipse クラス)
ANTLR : genErrorHandler メソッド (CppCodeGenerator,JavaCodeGenerator クラス)

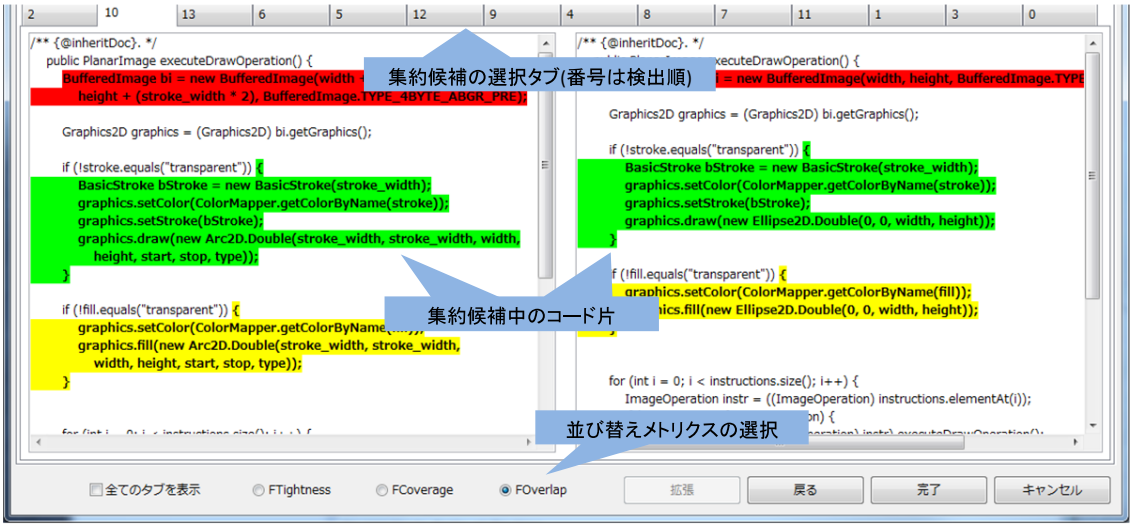


**図 6** 実装したツールのスクリーンショット
**Fig. 6** A Screenshot of the Tool

そして，提案手法によって提示された集約候補の中に，開発者が良い集約候補と考えるものがどの程度含まれているか調査するために，アンケートによる評価を行った．アンケートでは，提案手法によって生成された 3 つの集約候補のランキングから，それぞれの上位 10 件を抽出して被験者に提示して，良いと思う集約候補を全て選択してもらった．アンケートの被験者は，ソフトウェア工学に関連した研究室に所属している 15 名の学生である．

### 5.1 評価尺度

アンケート結果をもとにして，2 つの尺度を用いて評価を行った．1 つ目の尺度である平均候補選択率 (ARSC) は，アンケートで提示した全ての候補のうち，被験者に選択された候補の割合を表している．この尺度では，ツールが上位に提示した集約候補のうちどれだけの候補が選択されていたかを調べる．2 つ目の尺度である平均適合率 (AP) は，検索エンジンなどのランキングで正解とされるものを上位に提示することができているかを評価する尺度である [1]．ここでは，被験者に選択された集約候補を正解として，選択された集約候補が上位に提示されていたかを調べる．平均適合率は以下の数式で表される．数式において，$A$ は正解集合，$P(A_i)$ は $A$ 中の $i$ 番目の要素が順位に現れた時点での適合率をである．

$$AP = \frac{1}{|A|}\sum_{i=1}^{|A|} P(A_i) \qquad (1)$$

### 5.2 結果と考察

提案手法を適用した結果，全体の集約候補の数は，Ant では 23 個，ANTLR では 34 個であった．さらにフィルタリングを行った結果，最終的な集約候補数は，Ant では 14 個，ANTLR では 6 個となった．そのため，ANTLR については，6 件の集約候補を使用してアンケートを行った．なお，フィルタリングの閾値は実験的に 0.5 に設定した．

**表 1** に上記の評価尺度を用いて評価を行った結果を示す．表中の値は，それぞれの被験者について個別に算出したものを平均した値である．

結果を見ると，平均候補選択率の値は 0.213 から 0.267 となっており，各被験者が全体の 2 割程度の集約候補を選択していることがわかる．平均適合率の結果を見ると，Ant においては値が 0.5 以上と高く，被験者に選択された集約候補が上位に順位付けされていたことがわかる．Ant と ANTLR の両方のアンケート結果の詳細を確認すると，1 つ以上の集約候補を選択した被験者は，上位 3 つの集約候補の中から少なくとも 1 つは集約候補を選択していた．以上の結果から，提案手法によって，被験者にとって有用な集約候補を提示できていることがわかった．

今回の実験結果では，3 つのメトリクスによる結果に，大きな差異は見られなかった．この結果については，さらなる実験を行ったうえで，メトリクスによって結果がどのように変化するかを確認する必要がある．

次に，実験結果の妥当性について考察する．今回の適用実験では，提案手法で順位付けされた上位の候補に対してアンケートによる評価を行ったが，ツールに対する評価は行っていない．そのため，ツールにさらなる機能の追加を

**表 1** アンケートに対する評価結果
**Table 1** Result of the Questionnaire

| プロジェクト | メトリクス | ARSC | AP |
|---|---|---|---|
| Ant | FTightness | 0.253 | 0.533 |
| | FCoverage | 0.213 | 0.560 |
| | FOverlap | 0.253 | 0.535 |
| ANTLR | FTightness | 0.267 | 0.438 |
| | FCoverage | 0.267 | 0.346 |
| | FOverlap | 0.267 | 0.438 |

行ったうえで，リファクタリング作業の支援に，ツールがどの程度有効であるかの評価を行う必要がある．現在のツールの入力は，差分を含む 2 つの類似メソッドであるが，実際のリファクタリング作業を考えると，完全一致した類似メソッドや 3 つ以上の類似メソッドについても適用できる必要がある．また，順位付けされた集約候補を上位から順に提示しているが，コード片を指定することで集約候補を検索する機能を追加することで，利用者が集約候補を探す作業を軽減できると考えられる．

## 6. 関連研究

我々の研究グループではこれまでに，類似メソッドの集約候補を検出する手法と，それらの集約候補を凝集度メトリクス COB を用いて順位付けする手法を提案している [9], [10]．本研究では新たに，プログラムスライスを用いた凝集度メトリクスを使用して順位付けを行った．異なる凝集度メトリクスを使用している文献 [9] と本研究の結果を比較すると，本研究で提示した集約候補の方が，より多くの有用な集約候補を含んでいることがわかった．

Juillerat らは Template Method の形成を自動的に行う手法を提案している [4]．この手法では，類似メソッドの差分に対して，メソッドとして抽出可能となるように形式的な修正を行い，Template Method の形成を適用している．また，Hotta らは，コードクローン検出ツールを用いて，集約対象となる類似メソッドを特定する手法を提案している [3]．Hotta らの手法では，Template Method の形成が適用可能な類似メソッドを自動で特定して，特定した類似メソッドの差分と共通部分を開発者へと提示している．これらの手法と比べて，提案手法では，1 つの類似メソッド対に対して，複数の集約候補を検出している．提案手法では集約対象の特定や，集約候補の選択という開発者の作業が必要になるが，複数の集約候補の検出と，凝集度メトリクスを用いた並び替えを行うことで，集約作業によって生成されるメソッドの凝集度が高くなるように，開発者の支援を行っている．

## 7. まとめ

本研究では，集約候補を提示することで差分を含む類似メソッド対の集約を支援することを目的とした手法を提案した．また，プログラムスライスを用いた凝集度メトリクスを使用して集約候補の並び替え行うことにより，開発者の候補を選択する作業の支援を行った．

今後の課題として，ツールの機能拡張とツールに対する評価を行うことが挙げられる．ツールの機能拡張としては，完全一致した類似メソッドや 3 つ以上の類似メソッドを入力可能とすることや，集約候補の検索機能の追加などが必要となる．完全に一致した類似メソッドについては，提案手法で行っている差分の特定や集約候補の提示を行うことなく，集約を実現することができる．3 つ以上の類似メソッドについては，差分の特定処理において，3 つ以上の AST を同時に比較して，1 つでも異なっているノードがあれば，その部分を差分として検出することによって，対応可能であると考えられる．これらの機能拡張を行った後に，ツールを使用した実験を行い，ツールの有効性を調査することが今後の課題である．

**謝辞** 本研究は，日本学術振興会科学研究費補助金基盤研究（A）（課題番号:21240002），基盤研究（C）（課題番号:22500026）の助成を得た．

### 参考文献

[1] Baeza-Yates, R. and Ribeiro-Neto, B.: *Modern Information Retrieval*, Addison Wesley, second edition (2011).

[2] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).

[3] Hotta, K., Higo, Y. and Kusumoto, S.: Identifying, Tailoring, and Suggesting Form Template Method Refactoring Opportunities with Program Dependence Graph, *Proc. of CSMR2012*, pp. 53–62 (2012).

[4] Juillerat, N. and Hirsbrunner, B.: Toward an Implementation of the “Form Template Method” Refactoring, *Proc. of SCAM2007*, pp. 81–90 (2007).

[5] Murphy-hill, E. and Black, A. P.: Breaking the barriers to successful refactoring: observations and tools for extract method, *Proc. of ICSE2008*, pp. 421–430 (2008).

[6] Ott, L. M. and Thuss, J. J.: Slice Based Metrics for Estimating Cohesion, *Proc. of METRICS1993*, pp. 71–81 (1993).

[7] Stevens, W. P., Myers, G. J. and Constatine, L. L.: Structured design, *IBM Syst.J*, Vol. 13, No. 2, pp. 115–139 (1974).

[8] Weiser, M.: Program slicing, *Proc. of ICSE1981*, pp. 439–449 (1981).

[9] 井岡正和，吉田則裕，政井智雄，井上克郎：凝集度メトリクス COB を用いた Template Method パターン適用候補の順位付け手法，信学技報， Vol. 111, No. 169, pp. 57–62 (2011).

[10] 政井智雄，吉田則裕，松下　誠，井上克郎：テンプレートメソッドの形成に基づく類似メソッドの集約支援，ソフトウェア工学の基礎 XVII，pp. 125–130 (2010).

[11] 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術, 信学論 (D), Vol. J91-D, No. 6, pp. 1465–1481 (2008).