情報処理学会論文誌 Vol.54 No.2 922–932 (Feb. 2013)

差分を含む類似メソッドの集約支援ツール

後藤 祥[1,a)] 吉田 則裕[2] 井岡 正和[1] 井上 克郎[1]

受付日 2012年5月14日, 採録日 2012年11月2日

概要：ソースコード中で互いに一致または類似した部分を持つコード片のことをコードクローンと呼び，特にメソッド単位のコードクローンを類似メソッドという．もし，類似メソッドを持つメソッドに欠陥が含まれている場合，類似メソッドとなっている他のメソッドにも同様の欠陥が含まれている可能性が高く，それらを修正するのは大きなコストとなる．この問題の解決策として，類似メソッドの集約が有効であるが，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，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，その集約候補を提示することにより，作業を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対に対して本手法を適用した．また，ツールによって提示された集約候補が開発者の考えに近いものであるかアンケート評価を行い，提案手法が有効であることを確認した．

キーワード：ソフトウェア保守，リファクタリング，凝集度メトリクス，プログラムスライシング

A Tool Support to Merge Similar Methods with Differences

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

Received: May 14, 2012, Accepted: November 2, 2012

Abstract: A code fragment that has identical or similar code fragments is called code clone. Especially, method-base code clone is called similar method. If a similar method contains a defect, a developer needs to check all of similar methods for the same defect and it needs very high cost. Merging similar methods by refactoring is one of solutions to this problem. 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 providing candidates of merging similar methods given by a developer. 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. We conducted questionnaire and then investigated the similarity between candidates derived by our tool and the intention of the developers. As a result of the investigation, we confirmed the effectiveness of the proposed approach.

Keywords: software maintenance, refactoring, cohesion metrics, program slicing

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あげられる．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持つコード片のことである [1]．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．もし，類似メソッドを持つメソッドに欠陥が含まれている場合，類似メソッドとなっている他のメソッドにも同様の欠陥が含まれている可能性が高く，それらを修正するのは大きなコストとなる [2], [3]．

このような問題の解決策として，リファクタリングによる類似メソッドの集約があげられる．リファクタリングとは，“外部から見たときの振舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理すること” である [4]．リファクタリングにはさまざまなパター

[1] 大阪大学
Osaka University, Suita, Osaka 565–0871, Japan
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[a)] a-gotoh@ist.osaka-u.ac.jp

© 2013 Information Processing Society of Japan

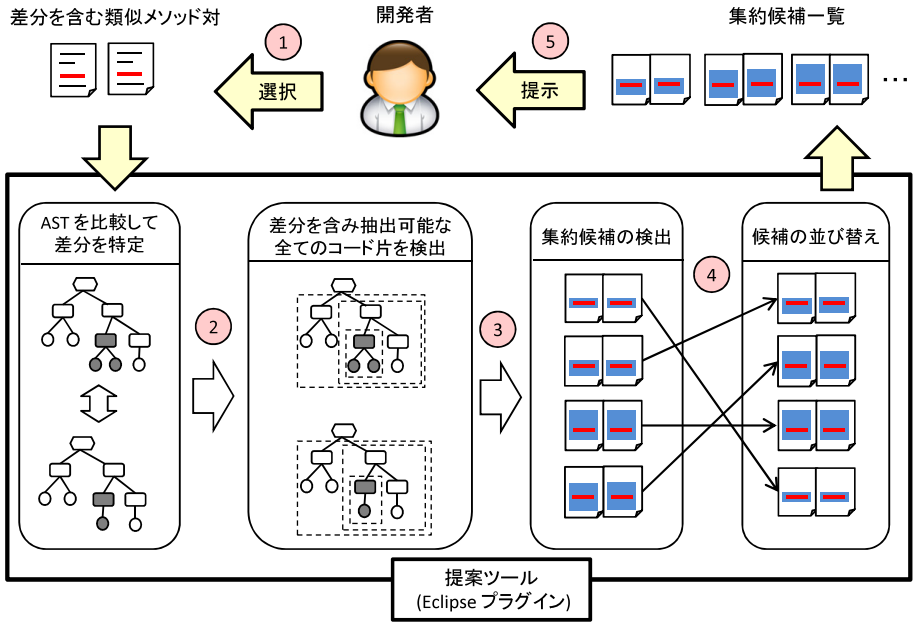

図 1 提案手法の概要
Fig. 1 Overview of the proposed approach.

ンがあり，そのいくつかが類似メソッドを取り除くために有効である．特に，差分を含む類似メソッド対が共通の親クラスを持つクラス間に存在する場合，Template Method の形成 [4] というリファクタリングパターンが集約に有効である．

Template Method の形成は GoF デザインパターンの 1 つである Template Method パターンに基づくリファクタリングである．Template Method パターンではアルゴリズムの骨格を親クラスで実装して，具体的な実装を子クラスで行う [5]．このリファクタリングを適用することで，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，その他の共通部分を親クラスに集約することができる．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（3.3 節定義 3 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，条件を満たすコード片を探すのは困難な作業となる．

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**）．また，手法を統合開発環境 Eclipse のプラグインとして実装した．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と考えて並べ替えを行う．凝集度には，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．プログラムスライスを用いた凝集度メトリクスは，Meyers らによって行われた 2 つのオープンソースソフトウェアに対する，長期間にわたる品質の変化を扱ったケーススタディによって，ソースコードの品質の定量的な評価に有用であることが示されている [6]．

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，適用結果を基にアンケートを行い，手法の評価を行った．評価の結果，提案手法によって提示された集約候補が，開発者の考えに近い集約候補であることが分かった．この結果から，提案手法がリファクタリング作業者の支援に有効であることが確認できた．

以降，2 章では研究背景について説明する．3 章と 4 章では提案手法について説明し，5 章では手法を実装したツールについて説明する．また，6 章では適用実験に基づく手法の評価とその結果について述べる．そして，7 章では関連研究について述べ，8 章ではまとめと今後の課題について述べる．

2. 背景

2.1 類似メソッド

ソースコード中に存在する互いに一致または類似したコード片をコードクローンという [1]．特にメソッド単位でのコードクローンを類似メソッド，類似メソッドとなっ

© 2013 Information Processing Society of Japan

ている 2 つのメソッドの組を類似メソッド対と呼ぶ．

一般的に，類似メソッドの存在はソフトウェアの保守を困難にするといわれている [2], [3]．もし，類似メソッドを持つメソッドに欠陥が含まれている場合，類似メソッドとなっている他のメソッドにも同様の欠陥が含まれている可能性が高く，その修正は大きなコストとなる．この問題の解決策として，リファクタリングによる類似メソッドの集約が有効である．しかし，文の編集や挿入，削除などが行われたために，類似メソッド間に差分が存在する場合は，集約作業が困難となる．

2.2 Template Method の形成

Template Method の形成は，デザインパターンの 1 つである Template Method パターンに基づいたリファクタリングである [4], [7]．Template Method の形成の対象となるのは，共通の親クラスを持つクラス間に存在する類似メソッドである．対象となる類似メソッド間に差分が存在する場合は，差分は子クラスごとの固有の処理として抽出し，共通部分は Template Method として親クラスに引き上げる．**図 2** は Template Method の形成の適用例である．この例では，類似メソッドとなっている *getBillableAmout* メソッドの差分（図 2 中の赤字の部分）を，それぞれのクラスに *getBaseAmount* メソッド，*getTaxAmount* メソッドとして抽出し，その後，親クラス *Site* へ類似メソッド対を集約している．

以下に Template Method の形成の手順を示す．

(1) 類似メソッド間の差分を検出する．
(2) 各子クラス固有の処理として抽出するコード片を差分を含むように決定する．
(3) 決定したコード片を各子クラスにメソッドとして抽出する．元のコード片は抽出したメソッドの呼び出し文に置き換える．

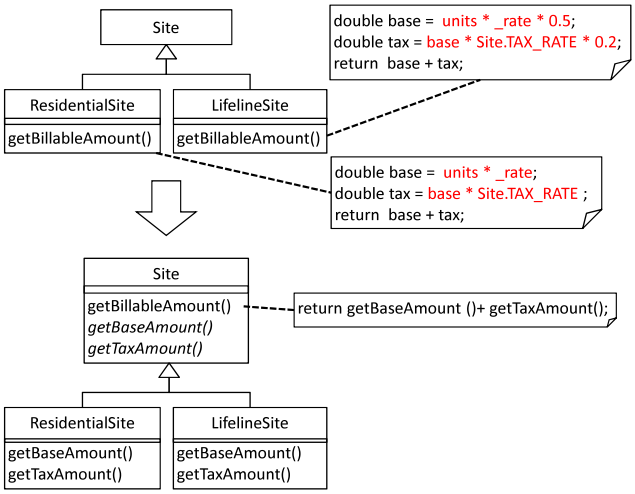

図 2 Template Method の形成の適用例 [4]
Fig. 2 An example of form Template Method refactoring.

(4) 記述が一致した類似メソッドを親クラスに引き上げる．また，(3) で抽出したメソッドを抽象メソッドとして親クラスで定義する（図 2 中の斜体のメソッド）．

3. 集約候補の検出

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） 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．

3.1 抽象構文木の構築

まず，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 [*1] を使用して類似メソッド対の AST を生成している．生成された 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，Eclipse JDT における，org.eclipse.jdt.core.dom パッケージの Statement クラス [*2] を継承するクラスが表すタイプ（ExpressionStatement や ReturnStatement など）を持つノードが該当する．提案手法では，AST の根ノードから幅優先探索を行い，到達した順番に特殊ノードに番号付けを行う．特殊ノードに付けられた番号は後述する差分の統合処理や，コード片の拡大処理において使用する．特殊ノードにのみ番号付けを行うのは，提案手法におけるコード片の最小単位を文と考えているためである．提案手法では，メソッド抽出はソースコード中の文を最小単位として行われると考え，提案手法の処理の過程で検出される，差分となっているコード片や，メソッドとして抽出するコード片について，その最小単位を文としている．以上の理由から，AST 上での操作をソースコード中の文に対応した単位で行う必要があるため，特殊ノードにのみ番号付けを行っている．**図 3** に AST と特殊ノードへの番号付けの例を示す．図 3 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．

*1 http://www.eclipse.org/jdt/
*2 http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fdom%2FStatement.html

© 2013 Information Processing Society of Japan

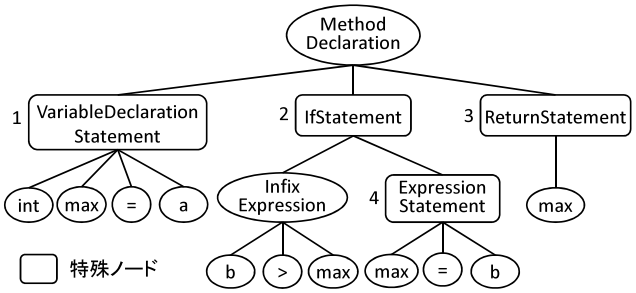

図 3 AST 中の特殊ノードへの番号付の例
Fig. 3 An example of indexing for particular nodes in AST.

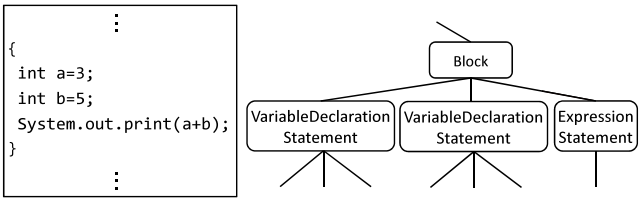

図 4 Block タイプのノードの例
Fig. 4 An example of a block type node.

3.2 類似メソッド間の差分検出

まず，処理手順の説明に必要となる Block タイプのノードについて説明する．Block タイプのノードはプログラム中の中括弧で囲まれた 1 つのブロックに対応するノードであり，ブロック内の各文を表すノードを子ノードとして持っている．**図 4** は Block タイプのノードの例である．図 4 の例では，ソースコード中の中括弧内に 3 つ文があり，AST では，それぞれの文に対応したノードが Block ノードの子ノードになっている．

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 が Block タイプであるかどうかによって異なる．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 タイプの子ノードは，対応する中括弧内中の文を表すノードであり，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 [8] 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．

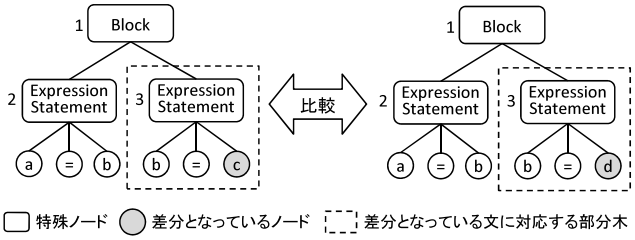

図 5 類似メソッド間で差分となっている文の検出
Fig. 5 Detecting statement-level differences between similar methods.

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が，ソースコード中のどの文に含まれているかを求める．まず，差分となっているノードから親ノードをたどっていき，最初に到達する特殊ノードを探索する．もし，差分となっているノードが特殊ノードであれば，自分自身とする．この最初に到達する特殊ノードを根とする部分木が，類似メソッド間で差分となっている文に対応している．

図 5 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，親ノードをたどって最初に到達する特殊ノードを求めることで，コード中で差分となっている文に対応した部分木（図 5 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，すべて特殊ノードを根とする部分木である．

次に，特定の条件を満たす差分を統合する．ここまでの操作で検出される差分は，すべて 1 つの文となっており，これらの差分となっている文を統合して，1 つのコード片にする．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，後述するメソッド抽出可能なコード片の検出の処理時間を削減することができる．差分を統合する操作は隣接した差分の統合と，ブロック内の差分の統合の 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 6** は，それぞれの統合操作について，AST 上でどのように統合が行われるかと，その操作がソースコード上でどのような意味を持つかを示している．以下に，それぞれの統合操作の手順を示す．

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（共通の親ノードを持つノード）の中で $i-1$ 番または $i+1$ 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．この統合操作は，ソースコード上で連続した文が差分となっている場合に，それらを統合して 1 つのコード片とする操作である（図 6 上段）．

統合 2（ブロック内の差分の統合） Block タイプのノードについて，そのすべての子が差分となっている場合

© 2013 Information Processing Society of Japan

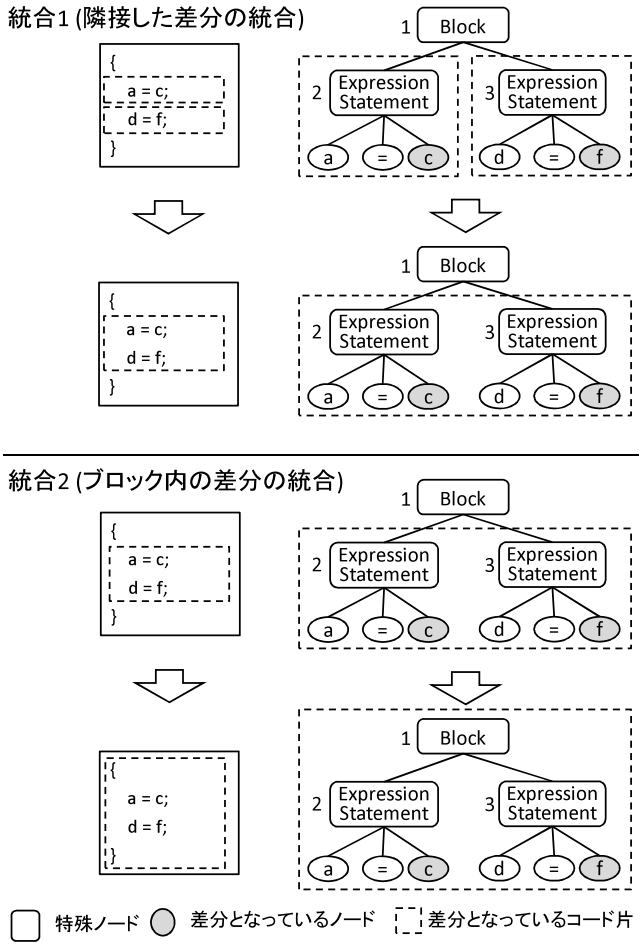

図 6 差分の統合操作
Fig. 6 Merging differences.

に，Block タイプのノードも差分に含める．この統合操作は，ソースコード中において，ブロック内のすべての文が差分となっている場合に，そのブロックを囲んでいる中括弧を含めて 1 つの差分とする操作である（図 6 下段）．

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 (Δ_A, Δ_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あり，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．

3.3 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 (Δ_A, Δ_B) を基に，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，その結果を基にして，類似メソッド対の集約候補を検出する．以下に，メソッド抽出可能なコード片と，類似メソッド対の集約候補の定義を示す．

定義 2（メソッド抽出可能なコード片） Murphy-hill らは，コード片がメソッド抽出可能であるための 3 つの条件を定義している [9]．提案手法では，Murphy-hill らの条件を用いて，コード片がメソッド抽出可能か判定する．条件は以下の 3 つであり，これらの条件をすべて満たすコード片をメソッド抽出可能なコード片とする．

条件 1：抽出するコード片中で宣言または代入が行われている変数のうち，抽出するコード片の後で参照される変数はたかだか 1 つである．

条件 2：抽出するコード片は，return 文を含まない，またはすべての実行経路において return 文で終了する．

条件 3：抽出するコード片は，対応する制御文（for, while など）がない break 文，continue 文を含んでいない．

定義 3（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 Δ_A, Δ_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 中のすべてのコード片はメソッド抽出可能である（条件 1，2，3 をすべて満たす）．

条件 B E_A，E_B 中のすべ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（Δ_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 δ に対して，δ を含み，メソッド抽出可能なすべてのコード片の集合 $include(\delta)$ を求める．$include(\delta)$ は，コード片 δ 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 δ として以下の手順で行われる．**図 7** 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 の部分木列に対して，根ノードである特殊ノードの中で，最も大きい番号を 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 に新たに追加する．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．拡大後，兄弟ノードにまだ 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 の親ノー

© 2013 Information Processing Society of Japan

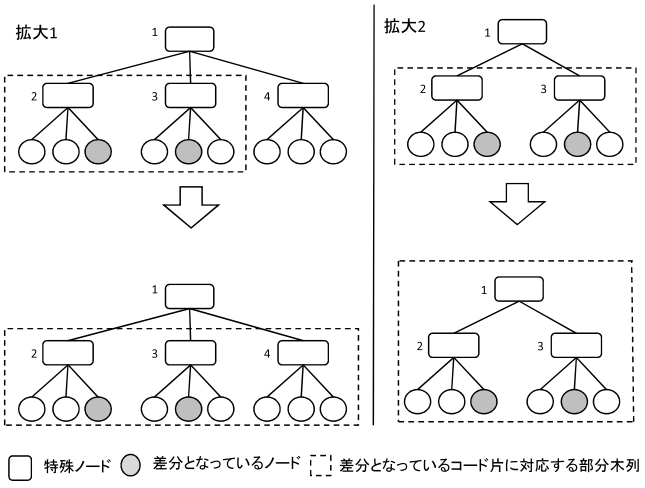

図 7 コード片に対する拡大操作
Fig. 7 Expansion for code fragments.

ドへの拡大操作を行う．

拡大 2（親ノードへの拡大） コード片 cf に対応する AST の部分木列に対して，親ノードをたどっていき，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 Δ_A, Δ_B 中のすべ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すべて列挙することで，類似メソッド対のすべ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．フィルタリングは，対象メソッドの文の数と，集約候補中の各コード片の文の数の比を基にして行う．集約候補 (E_A, E_B) に対するフィルタリングの式を以下に示す．式中の $len(x)$ は x の文の数を表しており，e_{Ai} と e_{Bi} はそれぞれ E_A と E_B の要素であるコード片を表している．また，$threshold$ はフィルタリングの閾値であり，0 以上 1 以下の値を任意に設定する．

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

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．

4. 集約候補の並べ替え

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べ替えを行い，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．差分を含む類似メソッド対の集約を行うと，集約候補中の各コード片は，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べ替えを行う．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に協調している度合いであり，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 [10], [11]．

Weiser はメソッドの凝集度を測るためのプログラムスライスを用いたメトリクスを 5 つ提案している [12]．プログラムスライスとは，ある文と依存関係を持つ文の集合のことで，ソースコード中の各文の依存関係を表したグラフである，プログラム依存グラフ（PDG）を使用して求められる．Meyers らは，Weiser が提案したメトリクスを用いて，2 つのオープンソースソフトウェアに対する，長期間にわたる品質の変化を扱ったケーススタディを行い，その結果から，Tightness，Coverage，Overlap の 3 つメトリクスが，ソースコードの品質の定量的な評価において，有用であることを示している [6]．提案手法における凝集度の計算には，これらの 3 つのメトリクスを使用する．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，提案手法ではそれに加えて，引数を起点とした前向きスライスも用いる．提案手法では，返り値が存在し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．V_o が空集合（返り値が存在しない）の場合は，V_i 中の全変数に対する前向きスライスの積集合を SL_{int} とする．同様に，V_i が空集合（引数が存在しない）の場合は，V_o 中の全変数に対する後ろ向きスライスの積集合を SL_{int} とする．

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

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

© 2013 Information Processing Society of Japan

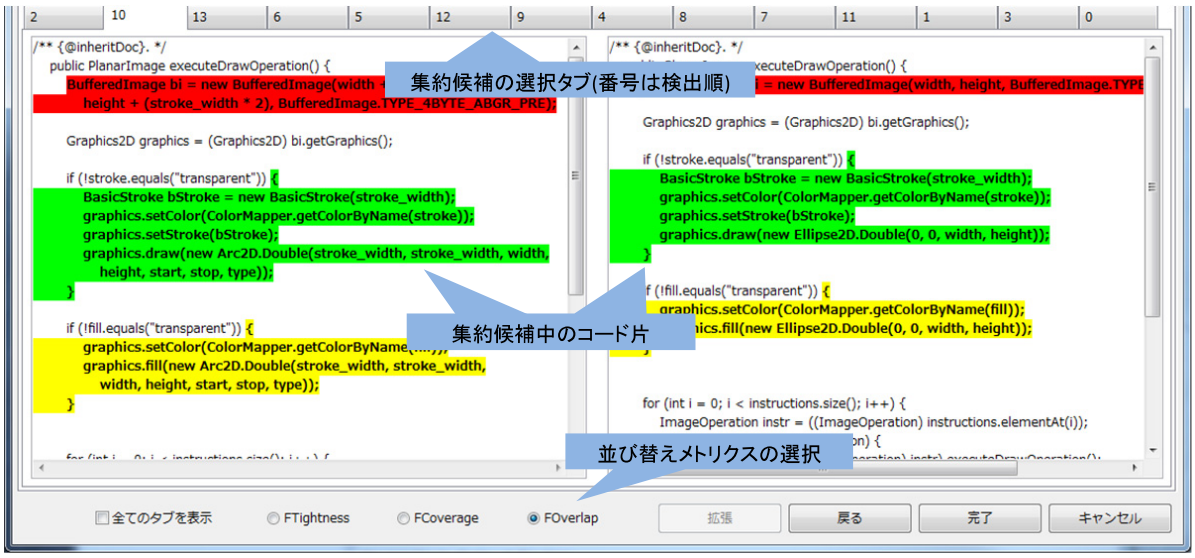

図 8 実装したツールのスクリーンショット
Fig. 8 A screenshot of the tool.

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

上記のメトリクスは，メソッド中に引数や返り値を表す変数に関連した処理を行っている文が，どの程度存在するかによって凝集度を測るものである．これらのメトリクスは，メソッドの引数と返り値を表す変数のうち，それらの多数に関連している文が，メソッド中に多く存在しているとき，高い値となる．このことから，あるメソッドおいてメトリクスの値が高いことは，引数と返り値を表す変数がある処理について協調していることを意味している．

集約候補の並べ替えの処理では，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，集約候補中のすべ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べ替えを行い，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．

5. 実装

提案手法は統合開発環境 Eclipse 上で利用できるようにプラグインとして実装した．AST の生成や，コード片がメソッド抽出可能かどうかの判定には Eclipse の機能を使用している．凝集度を計算するための PDG は，ソースコード解析ツール MASU を使用して構築している [13]．図 8 は実装したツールのスクリーンショットである．図 8 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．コード中の背景色がついている部分が集約候補中の各コード片であり，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，画面下部でボタンでメトリクスを選択することで，並べ替えに使用するメトリクスを変更することができる．

図 8 で対象としている類似メソッド対は，図形の描画を行うメソッドである．各コード片について，背景色が赤の部分は描画領域の確保，緑の部分は図形の輪郭の描画，黄の部分は図形内部の描画の処理を行っている．このように，図 8 の集約候補は各色のコード片が 1 つの処理を行っており，凝集度の高い集約候補である．

6. 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．実験対象は Ant と ANTLR の 2 つのプロジェクトから，2 つの類似メソッド対を選択した[*3]．これらの実験対象の類似メソッド対は，複数個所の差分を含むものを選択した．また，対象とした類似メソッド対は，それぞれ図形の描画処理とエラーコードの生成処理を行っており，それぞれの類似メソッド対で処理内容が類似している．このような類似メソッド対を選択した理由は，提案手法が，差分を含む類似メソッド対の集約支援を目的としていることと，処理内容が近い類似メソッド対が集約作業の対象になりやすいと考えたためである．

また，適用結果をもとに提案手法の評価を行った．評価の目的は以下の 2 点を調べることである．

目的 1 開発者が考える集約候補に近いものを提示することができているか．

目的 2 ツールが提示する集約候補を閲覧することで，開発者が考える集約方法が変化するか．

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

© 2013 Information Processing Society of Japan

以上の点を評価するために，ツールによって提示された上位 10 個の集約候補を用いて，アンケート評価を行った．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．アンケートでは以下の 3 つの質問を行った．設問 1 と設問 3 では，類似メソッド対とその差分を被験者へ提示して，抽出するコード片の範囲を囲んでもらった．

設問 1 対象類似メソッド対を集約する場合，どのようにコード片を抽出するか．

設問 2 対象類似メソッド対を集約する場合，ツールが上位に提示した集約候補のうち，どの候補を選択するか（複数選択可）．

設問 3 ツールの提示した集約候補を見た後で，設問 1 から考えが変化したか．変化したならば，どのようにコード片を抽出するか．

6.1 評価方法

6.1.1 設問 1 と設問 3 に対する評価

設問 1 と設問 3 について，被験者の回答を良い集約候補と考え，ツールが提示した上位の集約候補との比較を行う．この評価では，集約候補間の類似度を定義して，目的 1 に関して，各集約候補が被験者の考える候補に近いものか数値的に評価する．さらに，設問 1 と設問 3 の結果を比較することで，目的 2 に関する評価を行う．

評価に用いる集約候補間の類似度について説明する．以下に，類似度の定義のための定義 4，5 と，2 つの候補間の類似度の定義 6 を示す．

定義 4（コード片の類似度） 2 つのコード片 CF_1 と CF_2 の類似度として，$sim(CF_1, CF_2)$ を定義する．ここで，$lines(CF)$ はコード片 CF 中の文の集合である．

$$sim(CF_1, CF_2) = \frac{|lines(CF_1) \cap lines(CF_2)|}{|lines(CF_1) \cup lines(CF_2)|}$$

定義 5（差分を含むコード片） 集約候補 C において，差分となっているコード片 δ を含む C 中のコード片を $contain(C, \delta)$ と定義する．

定義 6（集約候補の類似度） 類似メソッド対 M_A，M_B に対する，2 つの集約候補 C_1，C_2 の類似度 $similarity(C_1, C_2)$ を以下のように定義する．ただし，M_A 中の差分となっているコード片の集合を Δ_A，M_B 中の差分となっているコード片の集合を Δ_B とし，$n = |\Delta_A| = |\Delta_B|$ とする．また，δ_{Ai}，δ_{Bi} はそれぞれ，Δ_A，Δ_B の i 番目の要素とする．

$$\begin{aligned} &similarity(C_1, C_2) \\ &= \frac{1}{2n}\sum_{i=1}^{n}(sim(contain(C_1, \delta_{Ai}), contain(C_2, \delta_{Ai})) \\ &\quad + sim(contain(C_1, \delta_{Bi}), contain(C_2, \delta_{Bi}))) \end{aligned}$$

6.1.2 設問 2 に対する評価

設問 2 に対しては，被験者に選択された集約候補が被験者の考えに近い候補として，選択された集約候補の割合や，選択された集約候補が上位に提示されているかを評価する．設問 2 の評価は，目的 1 に関する評価であり，評価のために以下の 3 つの尺度を使用する．

1 つ目の尺度である被験者の選択率（RSSC）は，全体の被験者のうち，1 つ以上の候補を選択した被験者の割合である．この尺度では，ツールが上位に提示した集約候補に被験者の考えに近いものが存在したかどうかを調べる．リファクタリングを行う際は 1 つの集約候補を選択してそれを適用するため，上位に提示された集約候補の中に，1 つでも選択された集約候補があれば提案手法が有効であるといえる．

2 つ目の尺度である平均候補選択率（ARSC）は，設問 2 で提示したすべての候補のうち，被験者に選択された候補の割合である．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たかを調べる．

3 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 [8]．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表している．

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

6.2 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．

設問 1 と設問 3 の回答に対して，類似度を計算した結果を**表 1** に示す．表 1 では，まず上位に提示された各集約候補と各被験者の回答を比較して類似度を計算した後，それらの結果を各集約候補ごとに平均した値の最大，最小，平均をそれぞれ示している．ANTLR では全体の集約候補数が 6 つであり，それぞれのメトリクスで上位に提示された集約候補が同じであるため，類似度の計算結果も 3 つのメトリクスで同じになっている．設問 1 と設問 3 の結果を見ると，Ant では設問 3 の類似度の平均が 7 割を超えており，被験者の考えに近い集約候補を提示できていることが分かる．また，すべての結果において設問 3 の方が設問 1 より類似度が高くなっている．このことから，被験者が想

© 2013 Information Processing Society of Japan

表 1 設問 1 と設問 3 に対する結果
Table 1 Result of question 1 and question 3.

プロジェクト	メトリクス	設問	最大	最小	平均
Ant	FTightness	1	0.697	0.513	0.643
		3	0.787	0.611	0.708
	FCoverage	1	0.697	0.604	0.651
		3	0.787	0.651	0.714
	FOverlap	1	0.697	0.513	0.643
		3	0.787	0.611	0.708
ANTLR	FTightness	1	0.441	0.278	0.358
		3	0.612	0.459	0.545
	FCoverage	1	0.441	0.278	0.358
		3	0.612	0.459	0.545
	FOverlap	1	0.441	0.278	0.358
		3	0.612	0.459	0.545

表 2 設問 2 に対する結果
Table 2 Result of question 2.

プロジェクト	メトリクス	RSSC	ARSC	AP
Apache Ant	FTightness	0.933	0.253	0.533
	FCoverage	0.867	0.213	0.560
	FOverlap	0.933	0.253	0.535
ANTLR	FTightness	0.733	0.267	0.438
	FCoverage	0.733	0.267	0.346
	FOverlap	0.733	0.267	0.438

定していなかった集約候補をツールが提示して，それを見ることによって被験者が考えを変えたと考えられる．

次に設問 2 の結果を**表 2** に示す．表 2 中の平均適合率は，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．被験者の候補選択率を見ると，7 割以上の被験者がいずれかの候補を選択している．このことからほとんどの被験者が，少なくとも 1 つは候補を選択していることが分かる．平均候補選択率からは，提示した候補のうち 2 割以上が選択されていることが分かる．また，平均候補選択率と平均適合率の結果をあわせて考えると，少なくとも上位 3 つのうち 1 つが選択されているということになる．以上のことから，提案手法によって提示された集約候補の中に，被験者の考えに近い候補が存在することが分かった．また，被験者の考えに近い候補が上位に提示されていることから，提案手法で行っている凝集度による並べ替えが有効であることが分かった．

7. 関連研究

Juillerat らは Template Method の形成を自動的に行う手法を提案している [14]．この手法では，AST の比較によって検出した差分に対して，形式的な修正を行うことでメソッドとして抽出可能にした後に，Template Method の形成を行っている．それに対して，提案手法では，候補選択という利用者の作業が必要になるが，複数の集約候補の検出と，凝集度による並べ替えを行うことで，開発者の考えに近い集約候補を提示する．

Hotta らは，集約対象となる類似メソッド対を，自動で特定する手法を提案している [15]．Hotta らの手法の特徴として以下の点があげられる．

- コードクローン検出ツールを用いて対象の類似メソッド対を自動で検出するため，開発者が集約対象の類似メソッド対を選択する必要がない．
- PDG を用いて類似メソッド対の差分を特定するため，ユーザ定義名（変数名や定数名など）の違いを吸収することができる．

たとえば，類似メソッド間で変数名のみが異なっている部分がある場合，変数名を変更すれば集約が可能であるため，共通部分として提示することが望ましい．PDG を用いている Hotta らの手法では，変数名の違いを吸収して，このような部分を共通部分として検出できるが，AST を用いている提案手法では，このような部分はすべて差分と判断され，集約することができない．

Hotta らの手法に対する，提案手法の特徴としては，1 つの類似メソッド対に対して複数の集約候補の検出を行っている点と，凝集度による集約候補の並べ替えを行っている点があげられる．提案手法では，開発者の考え方や開発組織の規約によって集約方法が異なると考え，開発者が集約候補を選択できるように，1 つの類似メソッド対に対して複数の集約候補の検出を行っている．それに対して，Hotta らの手法は，類似メソッド対の差分と共通部分を特定して，差分のみをメソッド抽出する集約候補の 1 つだけを提示している．提案手法では，メソッド抽出するコード片が機能的にまとまったものになるように，差分だけではなく，共通部分もあわせてメソッド抽出するような集約候補を検出して提示するが，このような集約候補は Hotta らの手法では検出することができない．本研究の適用実験において，フィルタリング前の集約候補数は，Ant に対して 23 個，ANTLR に対して 34 個であったが，これらのうち 1 つは差分のみを抽出する集約候補であり，それ以外の集約候補は Hotta らの手法では検出できない候補である．アンケートでは，上位の集約候補のうち 2 割程度が被験者に選択されており，このことから，1 つの類似メソッド対に対して複数の集約候補を提示することは，集約作業の支援において有効であると考えられる．さらに，提案手法では，検出した集約候補に対して，凝集度を計算して，凝集度が高い集約候補から順に開発者へと提示している．凝集度の高いメソッドは保守性や可読性に優れており，集約作業においては，凝集度が高くなるようにメソッドを抽出することが望ましいと考えられる．Hotta らの手法では，PDG の比較により類似メソッドの差分と共通部分を特定し，開発者へと提示しているが，凝集度の計算は行っていない．

このように，Hotta らの手法は，大規模なソフトウェア

© 2013 Information Processing Society of Japan

の中から，多くの集約対象の類似メソッドを特定することを目的としており，提案手法は，与えられた類似メソッド対に対して，開発者の考えに合った集約方法を提示することや，集約によって生成されるメソッドの保守性の向上を目的としている．提案手法と Hotta らの手法を組み合わせて，集約対象の類似メソッド対の検出と，差分の特定までを Hotta らの手法で行い，次に，その結果を入力として，提案手法で行っている複数の集約候補の検出と並べ替えを行うことで，集約対象の類似メソッド対の検出から実際に集約を行うまでの一連の作業の支援に，有効な手法が提案できると考えられる．

Wang らはメソッド中のコード片を意味的なまとまりに自動的に分割する手法を提案している [16]．この手法では，構文情報やデータフローの情報をもとに，コード片の意味的なまとまりを定義して，それに基づいたコード片の分割を行っている．提案手法では，プログラムスライスを用いた凝集度メトリクスを用いて，コード片の意味的なまとまりを求めた．これは，PDG におけるデータ依存関係と制御依存関係をもとに算出したものであり，構文情報は考慮していない．構文情報を用いることによって，たとえば，同じメソッドの呼び出しが連続している部分を意味的なまとまりと判定することができる．プログラムスライスを用いた凝集度メトリクスに加えて，構文情報を用いて意味的なまとまりを判定することで，より作業者の考えに近い集約候補を上位に提示することができると考えられる．

リファクタリングは保守性や可読性の向上を目的として行われるが，リファクタリングによる効果を品質メトリクスを使って評価する方法がある．松本らは，CK メトリクス [17] を使用したリファクタリング効果の予測手法を提案しツールとして実装している [18]．松本らのツールは，リファクタリング前後で CK メトリクスの値がどのように変化するか提示することによって，リファクタリング効果の予測支援を行っている．提案手法においても，集約候補と，その候補を用いてリファクタリングを行った場合のメトリクス値の変化を提示することで，作業者のリファクタリングによる効果の予測を支援することができると考えられる．

8. 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べ替え行うことにより，開発者の候補選択作業の支援を行った．

今後の課題として，3 つ以上の類似メソッドに対して適用可能なようにツールの機能を拡張することがあげられる．現在のツールは類似メソッド対を対象としているが，Template Method の形成は 3 つ以上の類似メソッドに対しても適用可能である．3 つ以上の類似メソッドに適用する場合，差分の特定処理が困難になることや，集約候補数が増大するなどの問題点が考えられる．特に集約候補数に増大に対しては，適切なフィルタリング閾値の決定や，並べ替え処理の改善が必要となる．

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）の助成を得た．

参考文献

[1] 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，電子情報通信学会論文誌，Vol.J91-D, No.6, pp.1465–1481 (2008).
[2] Laguë, B., Proulx, D., Mayrand, J., Merlo, E.M. and Hudepohl, J.: Assessing the Benefits of Incorporating Function Clone Detection in a Development Process, *Proc. ICSM1997*, pp.314–321 (1997).
[3] Balazinska, M., Merlo, E., Dagenais, M., Laguë, B. and Kontogiannis, K.: Measuring Clone Based Reengineering Opportunities, *Proc. METRICS1999*, pp.292–303 (1999).
[4] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).
[5] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley (1995).
[6] Meyers, T.M. and Binkley, D.: An empirical study of slice-based cohesion and coupling metrics, *ACM Trans. Softw. Eng. Methodol.*, Vol.17, No.2, pp.1–27 (2007).
[7] Kerievsky, J.: *Refactoring to Patterns*, Addison Wesley (2004).
[8] Baeza-Yates, R. and Ribeiro-Neto, B.: *Modern Information Retrieval, 2nd edition*, Addison Wesley (2011).
[9] Murphy-hill, E. and Black, A.P.: Breaking the barriers to successful refactoring: observations and tools for extract method, *Proc. ICSE2008*, pp.421–430 (2008).
[10] Stevens, W.P., Myers, G.J. and Constatine, L.L.: Structured design, *IBM Systems Journal*, Vol.13, No.2, pp.115–139 (1974).
[11] 三宅達也，肥後芳樹，井上克郎：メソッド抽出の必要性を評価するソフトウェアメトリックスの提案，電子情報通信学会論文誌，Vol.J92-D, No.7, pp.1071–1073 (2009).
[12] Weiser, M.: Program slicing, *Proc. ICSE1981*, pp.439–449 (1981).
[13] Higo, Y., Saitoh, A., Yamada, G., Miyake, T., Kusumoto, S. and Inoue, K.: A Pluggable Tool for Measuring Software Metrics from Source Code, *Proc. IWSM-MENSURA2011*, pp.3–12 (2011).
[14] Juillerat, N. and Hirsbrunner, B.: Toward an Implementation of the “Form Template Method” Refactoring, *Proc. SCAM2007*, pp.81–90 (2007).
[15] Hotta, K., Higo, Y. and Kusumoto, S.: Identifying, Tailoring, and Suggesting Form Template Method Refactoring Opportunities with Program Dependence Graph, *Proc. CSMR2012*, pp.53–62 (2012).
[16] Wang, X., Pollock, L. and Shanker, K.V.: Automatic segmentation of method code into meaningful blocks to improve readability, *Proc. WCRE2011*, pp.35–44 (2011).
[17] Chidamber, S.R. and Kemerer, C.F.: A Metrics Suite for Object Oriented Design, *IEEE Trans. Softw. Eng.*, Vol.20, No.6, pp.476–493 (1994).
[18] 松本義弘，肥後芳樹，楠本真二，井上克郎：CK メトリク

© 2013 Information Processing Society of Japan

スに基づくリファクタリングの効果予測手法の提案と実装，電子情報通信学会技術研究報告 SS2006-84，Vol.106, No.523, pp.31–36 (2007).

後藤 祥

平成 24 年大阪大学基礎工学部情報科学科卒業．現在，大阪大学大学院情報科学研究科博士前期課程 1 年．リファクタリング支援の研究に従事．

吉田 則裕 （正会員）

平成 16 年九州工業大学情報工学部知能情報工学科卒業．平成 21 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員（PD）．平成 22 年奈良先端科学技術大学院大学情報科学研究科助教．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

井岡 正和

平成 23 年大阪大学基礎工学部情報科学科卒業．現在，大阪大学大学院情報科学研究科博士前期課程 2 年．リファクタリング支援および実行履歴分析の研究に従事．

井上 克郎 （フェロー）

昭和 59 年大阪大学大学院基礎工学研究科博士後期課程修了（工学博士）．同年大阪大学基礎工学部情報工学科助手．昭和 59〜61 年ハワイ大学マノア校コンピュータサイエンス学科助教授．平成 3 年大阪大学基礎工学部助教授．平成 7 年同学部教授．平成 14 年大阪大学大学院情報科学研究科教授．平成 23 年 8 月より大阪大学大学院情報科学研究科研究科長．ソフトウェア工学，特にコードクローンやコード検索等のプログラム分析や再利用技術の研究に従事．

© 2013 Information Processing Society of Japan