社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2011-22,KBSE2011-19(2011-7)

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

井岡　正和†　吉田　則裕††　政井　智雄†　井上　克郎†

† 大阪大学大学院情報科学研究科
〒 560-0871 大阪府吹田市山田丘 1 番 5 号
†† 奈良先端科学技術大学院大学 情報科学研究科
〒 630-0192 奈良県生駒市高山町 8916 番 5 号
E-mail: †{m-ioka,t-masai,inoue}@ist.osaka-u.ac.jp, ††yoshida@is.naist.jp

あらまし　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．この問題を解決する方法として，コードクローンの集約が挙げられるが，リファクタリングの熟練者でない限り難しい．これを支援する手法を政井らが提案している．この手法は，不一致部分を含み，メソッドとして抽出することが可能なコード片の候補を提示すが，候補の順序に意味を持たないため，本研究では，凝集度メトリクスを用いて候補の順位付け手法を提案する．実験では，メソッド間クローン率の高いオープンソースソフトウェアに対して本手法を適用し，利用者にとって有用な候補が上位に現れていることを確認できた．

キーワード　コードクローン，リファクタリング，Template Method パターン，凝集度メトリクス

Ranking Candidates for Applying Template Method Pattern with a Cohesion Metric COB

Masakazu IOKA†, Norihiro YOSHIDA††, Tomoo MASAI†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka, 565-0871 Japan
†† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0192 Japan
E-mail: †{m-ioka,t-masai,inoue}@ist.osaka-u.ac.jp, ††yoshida@is.naist.jp

Abstract　Code clone is a factor that makes software maintenance complicate. Merging code clones is one of way that reduces cost to maintain source code. However, merging code clones is difficult unless refactoring expert. Masai et al. proposed an approach for merging code clones by showing candidates of code fragments which include mismatch parts and can be extracted as a method. This approach has the problem that candidates are out of order. Therefore, the study proposes to rank candidates based on a metric cohesion. In the experiment, the proposed approach is applied to high clone ratio of Java methods in open source software. The result shows high order of candidates is suitable for users.

Key words　Code Clone, Refactoring, Template Method Pattern, Cohesion Metrics

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類似または一致したコード片のことである [3]．もし，あるコードクローンを持つコード片に欠陥が含まれている場合，そのコードクローンすべてに対して修正を検討する必要がある．つまり，コードクローンを効率的に集約し，取り除くことができれば，保守コストを下げることができる [7], [8], [12]．コードクローンのうち，不一致部分を含むコードクローンは，不一致部分の修正と除去を考慮する必要があるため，完全に一致するコードクローンの集約に比べて，困難である．

コードクローンを取り除く手法に，リファクタリングがある．リファクタリングとは，“外部から見たときの振る舞いを保ちつ

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 [1]．このリファクタリングの手法として，Kerievsky はデザインパターンを使用する手法を提案している [5]．ソフトウェア開発におけるデザインパターンとは，過去の設計者が編み出した典型的な問題に対する解決策をカタログ化したものである．Kerievsky が文献 [5] の中で提案しているリファクタリングパターンの中に，コードクローンに対して効果的なリファクタリングを行う “Template Method の形成” がある．

“Template Method の形成” とは，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パターンである [2]．Template Method パターンとは，親クラスでおおまかな処理のアルゴリズムを決めておき，具体的な内容を子クラスに任せるものである．そのため，子クラスでは共通の処理を実装する必要がなく，クラス間で類似したコード片の出現を避けることができる．

類似したメソッド対の集約を支援する手法を政井らが提案している [9]．この手法は，不一致部分を含み，メソッドとして抽出することが可能なコード片の候補を提示することで，Template Method パターンの適用を支援している．しかし，対象とするメソッド対によっては提示される候補数が 10 万を超えてしまうという問題がある．利用者が非常に多くの候補から利用者にとって有用な候補を見つけ出すの現実的ではない．

そこで，本研究では，候補の順位付けを行い，利用者にとって有用な候補を上位に提示することを支援する．Template Method パターンでは，不一致部分を包括するようにメソッドとして抽出するので，このメソッドとして抽出する範囲に機能的なまとまりがあれば良いと考えた．そこで，メトリクスとして，値が大きいほどソースコードブロック間で機能的なまとまりを持つとされる凝集度を使用し，降順で順位付けを行った．

凝集度として，計算量の少ない三宅らが提案した COB(Cohesion Of Blocks) を用いる [10]．COB は，特定のデータ要素と協調する機能要素の割合の平均に着目しており，メソッドの構成要素の協調度合を示す．

メソッド間クローン率 [9] の高いオープンソースソフトウェアに対して本手法を適用し，評価実験を行った．利用者にとって有用な候補が上位に現れていることを確認できた．

以降，2 節では本研究に関連する用語を説明する．3 節では，候補の順位付けの手法について説明し，4 節では適用実験について述べる．そして，5 節で関連研究を述べ，6 節で本研究のまとめと今後の課題について述べる．

2. 背　　景

2.1 Template Method の形成

Template Method の形成とは，親クラスが共通で，類似メソッドを持つ子クラス間に，Template Method パターンを適用するリファクタリングパターンである [1], [5].

Template Method の形成は，以下の手順で行う．

（1） 類似メソッド間の不一致部分を見つける．

（2） ステップ 1 で見つけた不一致部分を含むように，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．

（3） ステップ 2 で決定したコード片を子クラスにメソッドとして抽出し，元のコード片をそのメソッドの呼び出し文に置き換える．

（4） 記述を揃えたメソッドを親クラスに引き上げる．また，親クラスにステップ 3 で抽出したメソッドを抽象メソッドとして定義する．

2.2 類似メソッド集約候補を挙げる手法

FTMPATool は政井らが提案した手法を実装したツールで，Template Method の形成を用いた類似メソッド集約の支援を行う [9]．このツールは，統合開発環境 Eclipse(注1) の Java 開発キットのプラグインとして実装されているので，Eclipse の既存の機能を利用でき，また，Eclipse を用いたコーディングの過程で利用できる．なお，入力として類似メソッド対が与えられるとする．

2.2.1 [ステップ 1] 抽象構文木の生成

Eclipse の抽象構文木生成機能を用いて，与えられた類似メソッド対の抽象構文木をそれぞれ生成する．この抽象構文木のノードは，大きく以下の 2 つに分類される．

- “ユーザ定義名”，“return 文”，“代入文” 等の値や子ノードを持つノード (タイプ A).
- 中括弧で括られた範囲に対応する子ノードの列を持つノード (タイプ B).

2.2.2 [ステップ 2] 差分となる部分木の検出

ステップ 1 で生成された 2 つの抽象構文木を比較し，抽象構文木間の差分となる部分木を検出する．抽象構文木の比較は，メソッド宣言に対応するノードから開始し，子ノードへと再起的に比較を繰り返すことで行う．ノードの比較は，種類が同じか比較し，異なっていれば差分として検出する．ノードの種類が同じであればタイプによって以下の比較を行う．

- タイプ A のノードの場合，ノードの持つ値や子ノードを比較する．異なっていれば差分として検出する．
- タイプ B のノードの場合，子ノードの列を比較する．比較には，類似文字列マッチングアルゴリズム [11] を用いている．

差分はソースコード上においては，ステートメント単位である．また，差分と判断されたノードの子ノードはすべて差分と判断される．

2.2.3 [ステップ 3] 抽出が容易な部分木の検出

差分となる部分木を含む，メソッドとして抽出可能な部分木を検出する．

メソッドとして抽出が可能か否かは，Eclipse のリファクタリング機能である “メソッド抽出” を行うための事前条件判定機能を用いて判定している．この事前条件が抽出不可と判定するもので，この手法に関係するものは以下の 3 つである．

条件 1 複数の変数の初期化を含む宣言文，または，複数の変数への代入文が含まれており，かつ，それらの変数が後のコードにおいて参照されている．

条件 2 break 文，continue 文が含まれているが，これらに対

(注1)：Eclipse. http://eclipse.org/.

応する制御文が含まれていない．

条件 3　戻り値を持たない return 文を含んでいる．

これらの条件を満たすコード片は，そのままメソッドとして抽出することができない，または，抽出した場合に動作の保証ができない．よって，このようなコード片はメソッドとして抽出不可とし，段階的に範囲を拡大して条件判定を繰り返すことで抽出可能であるコード片を表す部分木列の検出を行う．

また，抽出範囲すべての組み合わせが検出されるように，差分となる部分木や抽出可能として検出された部分木からも範囲の拡大を行う．

2.2.4　[ステップ 4] 部分木列の分類

メソッド抽出可能であると検出された部分木列が表すコード片を，実際にメソッドとして抽出した後のメソッド呼び出し文の差異に基づいて分類する．以下の条件を用いて，6 つに分類する．

条件 1　戻り値の型，または値を戻す変数が異なる．

条件 2　引数として渡す変数が異なる．

条件 3　類似メソッド対の片方にのみ，対応する位置にコード片がない．

分類 1　条件 1，条件 2 を満たす抽出箇所が存在しない．

分類 2　条件 1 を満たす抽出箇所が 1 つ存在する．

分類 3　条件 2 を満たす抽出箇所が 1 つ存在する．

分類 4　条件 1，および条件 2 を満たす抽出箇所が 1 つ存在する．

分類 5　条件 3 を満たす抽出箇所が 1 つ存在する．

分類 6　条件 1~3 を 1 つ以上満たす抽出箇所が複数存在する．

分類 1 はそのままメソッドの抽出を行うことができるため，Template Method の形成が容易である．その他の分類は抽出を行った後の記述が揃わないため，事前にソースコードの修正が必要であり，Template Method の形成が容易とはいえない．

2.3　メソッド抽出を支援する凝集度メトリクス COB

COB(Cohesion Of Blocks) は三宅らが提案したメトリクスで，メソッド抽出の必要性を評価する凝集度メトリクスである [10]．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を表す．

メソッドの構成要素の協調度合いを表すために，メソッド内で使用されている変数をデータ要素，コードブロックを機能要素とみなすと，メトリクス COB は式 (1) で定義される．b はメソッド内のコードブロック数，v はメソッド内で使用されている変数の数，V_j はメソッド内で使用されている j 番目の変数，$\mu(V_j)$ は変数 V_j を使用しているコードブロック数を示す．

$$COB = \frac{1}{b}\frac{1}{v}\sum_{j}^{v}\mu(V_j) \quad (0 \leqq COB \leqq 1) \qquad (1)$$

図 1(a) の *sample* メソッドのコードを用いた例を挙げる．図中の BLOCK1~4 はコードブロックを表す．図 1(b) は，*sample* メソッドのコードブロックと変数の協調関係の抽象図である．四角はコードブロック，四角の中の丸はコードブロック中で使用されている変数，点線はコードブロック間で変数が協調していることを表す．図 1(b) より，*sample* メソッドは BLOCK1

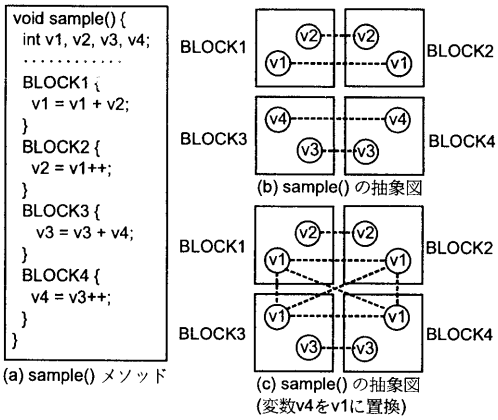

図 1　サンプルコード [10]

と BLOCK2 は変数 v1，v2 を介して協調しており，同様に，BLOCK3 と BLOCK4 は変数 v3，v4 を介して協調している．一方，BLOCK1，2 と BLOCK3，4 は協調していない．このときの COB の値は 0.5 となる．また，図 1(c) は，(b) での変数 v4 を v1 に置き換えたものである．このようにしたとき，すべてのコードブロックが変数 v1 を介して協調するため，COB の値は大きくなり，値は 0.66 になる．

このことから，メソッド内のコードブロック間で変数が互いに協調しているときに COB の値が大きくなることが分かる．つまり，メソッド内のコードブロック間で変数が互いに協調していない，すなわち，COB の値が小さいときはメソッドを分割すべきであるといえる．

2.4　既存研究の問題点

政井らの提案した手法は主に 2 つの問題がある．

（1）　候補を提示する際に，利用者にとって有用な候補から順に表示されない．

（2）　入力として与えるメソッド対によっては，候補が膨大な数になる．

以下で，各問題の原因について説明する．

(1) 提示される候補の順序に意味を持たない　候補は各差分となるコード片から範囲を段階的に拡大させて検出する．その後，この候補群を抽出時のメソッド呼び出し文の差異に基づいて分類しているが，候補自体のソーティングは行っていない．そのため，提示される候補の順序に意味を持たないので，利用者にとって有用な候補を見つけ出すのは難しい．

(2) すべての組み合わせの候補を提示する　FTMPATool では，抽出範囲すべての組み合わせが検出されるように候補を挙げる．そのため，類似メソッド対両方の行数が大きいと候補数が膨れ上がってしまう．候補数が 10 万を超えることもあり，すべてを確認することは現実的ではない．

3.　提 案 手 法

本研究では，メトリクス COB を用いて Template Method パターン適用時の抽出メソッドに機能的なまとまりのあるものから利用者に提示することで，前節 2.4 で述べた問題点の解決をはかる．機能的なまとまりを持つものが利用者にとって有用

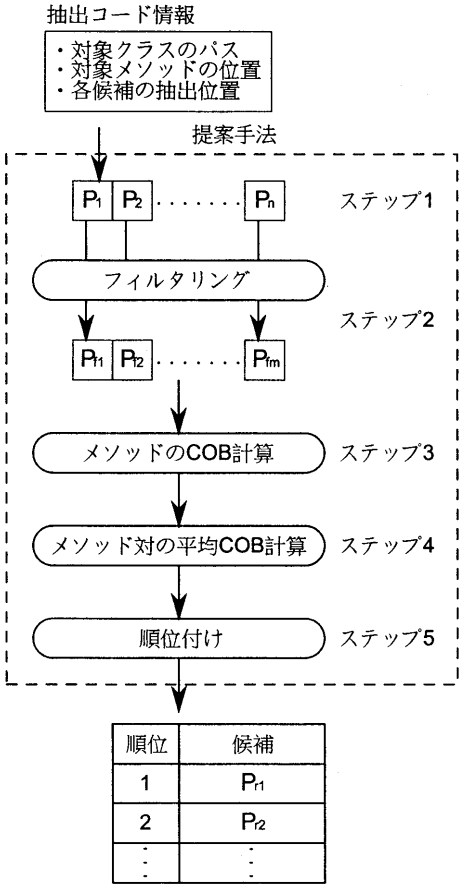

図 2　提案手法全体の流れ

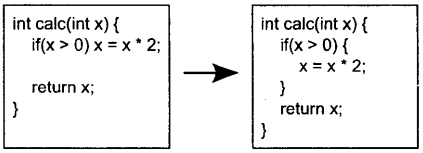

図 3　中括弧の補完

であると考えたので，利用者に機能的なまとまりのあるものから提示することで問題点の 1 つ目を解決できる．また，機能的なまとまりのないものを候補に挙げないことで問題点の 2 つ目を解決できる．入力として類似メソッド対が与えられるものとする．図 2 は提案手法全体の流れを示す．

3.1　[ステップ 1] FTMPATool が出力する集約候補の取得

FTMPATool 実行し，各候補の抽出コード片情報を取得する．FTMPATool からテキストファイルとして，メソッド対のファイルパス，メソッドの位置，各候補ごとの抽出コード片の位置が出力される．

この出力ファイルより 2 つのメソッドを取得し，COB の計算に不必要なものを削除するための前処理を行う．前処理では，コメントの削除，文字列の削除，型パラメータの削除，空白の削除，if 文等で中括弧の補完 (図 3)，代入のない宣言の削除，を行う．

3.2　[ステップ 2] メソッド抽出候補の大きさに基づく集約候補のフィルタリング

抽出元メソッドのコードの大きさに対して閾値を設定し，抽出するコード片の大きさがその閾値を超えるコード片が 1 つでもある場合は，フィルタリングを行い，候補に挙げない．

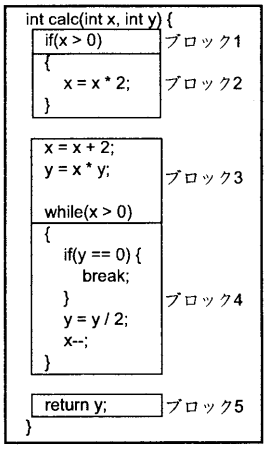

図 4　ブロック分けの例

このフィルタリングを行うのは，大きい範囲をメソッドとして抽出した場合に，抽出したメソッド対が再びコードクローンとなるためである．

3.3　[ステップ 3] メソッド抽出候補に対する凝集度の算出

各抽出コード片を 1 つのメソッドと考えて COB を計算する．図 4 のように，中括弧で括られている箇所を 1 つのブロックとする．なお，中括弧が入れ子になっている場合は，中括弧で括られている一番外側を 1 つのブロックとし，内側はブロックとみなさない．また，ブロックとブロックの間のコード片を 1 つのブロックとみなす．

3.4　[ステップ 4] 集約候補に対する凝集度の計算

ステップ 3 で求めた各抽出コードへの COB の値を用いて，類似メソッド対に含まれるすべての抽出コード片の COB の値の平均を計算する．

3.5　[ステップ 5] 凝集度に基づく集約候補の順位付け

FTMPATool における 6 つの分類ごとに，類似メソッド対の平均 COB 値の大きさ順に順位付けを行う．このとき，類似メソッド対内のすべての抽出コード片がブロック 1 つのみを抽出する場合は，順位を低くする．これは，ブロック 1 つのみを抽出する場合は，必ず COB の値が 1 となるが，機能的なまとまりであるとはいえないことが多いためである．

4.　適 用 実 験

3. 節で述べた提案手法を，実際のソースコードを用いて適用実験を行った．提案手法では，FTMPATool の問題点を改善することを目的としている．そのため，適用実験では，FTMPATool の出力と本手法を用いた場合の結果を比較する．

4.1　準　　備

適用実験の対象には，ANTLR(注2)，Apache Ant(注3)，Azureus(注4) の 3 つのオープンソースソフトウェアを用いた．また，提案手法を適用する類似メソッド対を見つけるために，

(注2)：ANTLR parser generator. http://www.antlr.org/.
(注3)：Apache Ant. http://ant.apache.org/.
(注4)：Azureus. http://azureus.sourceforge.net/.

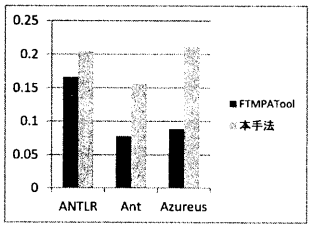

図 5　優れた候補の割合

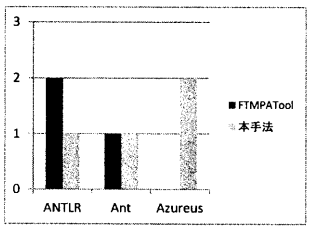

図 6　半数以上が判断した数

コードクローン検出ツールとして Scorpio(注5)を用いた．Scorpio は，プログラム依存グラフを用いたコードクローン検出ツールであり，不一致部分を含むコードクローンを検出できる．そして，Scorpio で検出した結果を用いてメソッド間クローン率 [9] を計算し，その値が大きいメソッド対を実験の対象とした．なお，Template Method の形成が容易である分類 1 の候補のみを対象に実験を行った．各実験対象の FTMPATool の出力における分類 1 の候補数は，ANTLR プロジェクトは 34，Ant プロジェクトは 23，Azureus プロジェクトは 479 である．

利用者に優れた候補を提示できていることを確認するために，実験として，FTMPATool の出力の表示順の上位 10 件と本手法の順位付けの結果の上位 10 件の中で，どの候補が優れた候補であるかを判断してもらった．被験者は，コンピュータサイエンスを専攻している学部 4 年から博士前期課程 2 年の学生 9 人を対象にした．

4.2 実　　験

ANTLR プロジェクトの CppCodeGenerator クラスと JavaCodeGenerator クラスの *genErrorHandler* メソッド，Ant プロジェクトの Arc クラスと Ellipse クラスの *executeDrawOperation* メソッド，Azureus プロジェクトの MD5 クラスと SHA1 クラスの *digest* メソッドの 3 つの類似メソッド対に対して評価実験を行ったところ，表 1，2，3 の結果が得られた．ただし，Ant プロジェクトの対象については，フィルタリングの結果候補数が 6 件となった．

4.3 考　　察

全候補数に対する優れた候補に選ばれた割合を図 5 に，半数以上の被験者が優れた候補とした候補数の推移を図 6 に示す．

図 6 より，どのプロジェクトについても，本手法を用いた場合に少なくとも 1 つは半数以上の被験者が優れた候補と判断した候補が存在する．また，図 5 より，どのプロジェクトについても，FTMPATool の結果に比べて本手法のほうが優れた候補が多いという結果となった．このことより，本手法を用いることによって，利用者にとって有用な候補を上位に提示することができている．つまり，凝集度による順位付けは妥当であるといえる．また，候補数が多い場合にも，候補の上位のみを提示すれば優れた候補を提示できるので，候補数が 10 万を超えるという問題を解消することができる．

表 1 より，ANTLR プロジェクトについては，抽出範囲によるフィルタリングによって優れた候補を除外しているので，フィルタリングの閾値を対象によって柔軟に設定する必要があるといえる．フィルタリングの閾値を決定するために，閾値ごとの本手法の結果を用いた追加実験が必要である．

(注5)：Scorpio.　http://sdl.ist.osaka-u.ac.jp/~higo/cgi-bin/moin.cgi/scorpio/.

5. 関 連 研 究

5.1 Juillerat らの手法

Juillerat らは，Template Method の形成の自動化手法を提案している [4]．この手法は，政井らの手法と同様に抽象構文木から差分を検出する．また，この手法では，差分を検出するために，2 つの抽象構文木を深さ優先探索の帰りがけ順に探索することで作成したノード列を用いて比較を行うため，高速に比較を行うことができるが，抽象構文木の構造的な情報を失う．また，抽出範囲において，片方に足りない代入文がある場合は，対応する位置に同じ変数に値が変化しない代入文を追加し，抽出範囲を変化させない．これらによって，抽出を自動化することは可能であるが機能的なまとまりを抽出しているとはいえない．一方，本手法は，自動的に抽出することはできないが，凝集度を用いて機能的なまとまりを持ったものから利用者に提示しているため，利用者が適切であるものを選ぶことができる．

5.2 堀田らの手法

堀田らは，プログラム依存グラフを用いた Template Method パターンの適用を支援している [13]．この手法は，プログラム依存グラフを作成するため，実行コストが非常に大きいが，ユーザ定義名や表現の違いを吸収することが可能である．また，この手法は，複数のメトリクスを利用者が自由に設定することで，Template Method の適用対象を選ぶことができる．メトリクスの設定という手間が増えるが，細かく条件を指定することができるので，利用者が必要とする適用対象を選ぶことが可能である．一方，本手法では，COB を用いて意味的なまとまりを持つものから利用者に提示するので，利用者が必要とする Template Method の適用候補を選択しやすい．

5.3 Krinke の手法

Krinke は，スライスベースでステートメント単位の凝集度メトリクスを提案している [6]．この凝集度メトリクスは式 2 で定義される．なお，SL_x は出力変数 x に対するスライスである．

$$Cohesiveness_{SL}(s) = \frac{\sum_{x|s\in SL_x} |SL_x|}{\sum_{x\in V_O} |SL_x|} \qquad (2)$$

このメトリクスは，スライスを計算するためにプログラム依存グラフを作成する必要があり処理に少し時間がかかるが，ステートメント単位で凝集度を求めることができるので，本手法のブロック単位の凝集度に比べて精度が高い．

6. まとめと今後の課題

本研究では，ソースコードブロック間の凝集度を用いることによって，政井らの手法の改善を行った．具体的には，政井らの手法によって提示される候補の抽出コード片を凝集度の大きさによって順位付けを行うことで，利用者にとって有用な候補を見つけやすくした．また，非常に多くの候補が現れる場合も，上位の結果だけを表示することで Eclipse 上のウィザード表示

表 1　ANTLR 実験結果
(a) FTMPATool の結果

	候補 1	候補 2	候補 3	候補 4	候補 5	候補 6	候補 7	候補 8	候補 9	候補 10	候補選択率
被験者 A		✓			✓			✓	✓		0.40
被験者 B									✓		0.10
被験者 C					✓				✓		0.20
被験者 D											0.00
被験者 E											0.00
被験者 F					✓						0.10
被験者 G					✓				✓		0.20
被験者 H				✓					✓		0.20
被験者 I		✓			✓				✓		0.30
選択率	0.00	0.22	0.00	0.11	0.56	0.00	0.00	0.11	0.67	0.00	

(b) 本手法の結果

	候補 1	候補 2	候補 3	候補 4	候補 5	候補 6	候補選択率
被験者 A							0.00
被験者 B							0.00
被験者 C						✓	0.17
被験者 D							0.00
被験者 E	✓	✓			✓	✓	0.67
被験者 F		✓				✓	0.33
被験者 G	✓		✓			✓	0.50
被験者 H							0.00
被験者 I						✓	0.17
選択率	0.22	0.22	0.11	0.00	0.11	0.56	

表 2　Apache Ant 実験結果
(a) FTMPATool の結果

	候補 1	候補 2	候補 3	候補 4	候補 5	候補 6	候補 7	候補 8	候補 9	候補 10	候補選択率
被験者 A											0.00
被験者 B		✓									0.10
被験者 C		✓									0.10
被験者 D		✓	✓								0.20
被験者 E		✓									0.10
被験者 F											0.00
被験者 G							✓				0.10
被験者 H											0.00
被験者 I		✓									0.10
選択率	0.00	0.56	0.11	0.00	0.00	0.00	0.11	0.00	0.00	0.00	

(b) 本手法の結果

	候補 1	候補 2	候補 3	候補 4	候補 5	候補 6	候補 7	候補 8	候補 9	候補 10	候補選択率
被験者 A					✓				✓		0.20
被験者 B						✓					0.10
被験者 C						✓	✓				0.20
被験者 D						✓	✓				0.20
被験者 E					✓	✓					0.20
被験者 F						✓					0.10
被験者 G	✓					✓	✓				0.30
被験者 H											0.00
被験者 I							✓				0.10
選択率	0.11	0.00	0.00	0.00	0.22	0.67	0.44	0.00	0.11	0.00	

表 3　Azureus 実験結果
(a) FTMPATool の結果

	候補 1	候補 2	候補 3	候補 4	候補 5	候補 6	候補 7	候補 8	候補 9	候補 10	候補選択率
被験者 A	✓		✓								0.20
被験者 B			✓								0.10
被験者 C			✓	✓							0.20
被験者 D			✓								0.10
被験者 E	✓										0.10
被験者 F											0.00
被験者 G											0.00
被験者 H				✓							0.10
被験者 I											0.00
選択率	0.22	0.00	0.44	0.22	0.00	0.00	0.00	0.00	0.00	0.00	

(b) 本手法の結果

	候補 1	候補 2	候補 3	候補 4	候補 5	候補 6	候補 7	候補 8	候補 9	候補 10	候補選択率
被験者 A								✓		✓	0.20
被験者 B										✓	0.10
被験者 C							✓			✓	0.20
被験者 D							✓	✓	✓	✓	0.40
被験者 E							✓		✓	✓	0.30
被験者 F							✓				0.10
被験者 G			✓						✓		0.20
被験者 H							✓		✓		0.20
被験者 I								✓		✓	0.20
選択率	0.00	0.00	0.11	0.00	0.00	0.00	0.56	0.33	0.44	0.67	

時にかかる時間の削減，リソースの節約が可能である．

Java で記述されたオープンソースソフトウェアのソースコードを対象に，被験者を用いて適用実験を行った結果，提案手法の有効性を確認できた．

今後の課題は，COB 以外のメトリクスを使用して候補の順位付けを行うことである．提案手法では，計算量が少ない COB を用いたが，COB はソースコードブロック間での協調関係のみに関与しているので，ステートメント単位の協調関係は分からない．そこで，プログラムスライスベースの凝集度や，プログラム依存グラフのデータ依存辺，制御依存辺を用いることによって，ステートメント単位での協調関係を利用することが考えられる．ステートメント単位での協調関係を利用すれば，ブロック間での協調関係を見ただけでは検出できなかった優れた候補を検出可能である．

謝辞　本研究で用いた Scorpio の提供をはじめ，様々なご協力をいただきました大阪大学肥後芳樹助教に感謝いたします．

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），および研究活動スタート支援（課題番号:22800040）の助成を得た．

文　献

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

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

[3] 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 D, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

[4] N. Juillerat and B. Hirsbrunner. Toward an Implementation of the "Form Template Method" Refactoring. In *Proc. of SCAM 2007*, pp. 81–90, Paris, France, 2007.

[5] J. Kerievsky. *Refactoring to Patterns.* Addison Wesley, 2004.

[6] J. Krinke. Statement-Level Cohesion Metrics and their Visualization. In *Proc. of SCAM '07*, pp. 37–48, Paris, France, 2007.

[7] B. Laguë, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudepohl. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. In *Proc. of ICSM' 97*, pp. 314–321, Bari, Italy, 1997.

[8] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. *IEEE Trans. Softw. Eng.*, Vol. 32, No. 3, pp. 176–192, 2006.

[9] 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集約支援. 日本ソフトウェア科学会 FOSE2010 ソフトウェア工学の基礎 XVII, pp. 125–130, 2010.

[10] 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 D, Vol. J92-D, No. 7, pp. 1071–1073, 2009.

[11] R. B. Yates and B. R. Neto. *Modern Information Retrieval.* Addison Wesley, 1999.

[12] A. Zeller. *Why Programs Fail.* Morgan Kaufmann Pub., 2005.

[13] 堀田圭佑, 肥後芳樹, 楠本真二. プログラム依存グラフを用いた Template Method パターン適用によるコードクローン集約支援. 情報処理学会研究報告, Vol. 2011-SE-171, No. 14, pp. 1–8, 2011.