特別研究報告

題目

プログラムスライスを用いた凝集度メトリクスに基づく類似メソッド集約候補の順位付け手法

指導教員
井上 克郎 教授

報告者
後藤 祥

平成 24 年 2 月 16 日

大阪大学 基礎工学部 情報科学科

平成 23 年度 特別研究報告

プログラムスライスを用いた凝集度メトリクスに基づく類似メソッド集約候補の順位付け手法

後藤 祥

内容梗概

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．もし，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，その全てのコードクローンにも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．

コードクローンを取り除くための方法の 1 つとしてコードクローンの集約が挙げられる．特に，コードクローンとなっている類似メソッド対を集約する方法として，Template Method の形成というリファクタリングパターンがある．Template Method の形成は，対象としている類似メソッド対が完全に一致しておらず差異が存在する場合にも共通部分のみを集約することができる．しかし，Template Method の形成は複数のリファクタリングパターンを利用するため，リファクタリングの熟練者でなければ困難である．

Template Method の形成を支援する手法を政井らが提案している．この手法では，類似メソッド間の差異を吸収する際に，メソッドとして抽出する部分の候補を利用者に提示することでリファクタリングの支援を行っている．しかし，政井らの手法には利用者に提示する候補数が膨大になることや，提示する候補の順序に意味がなく有用な候補を探すのが困難であるという問題がある．

これらの問題を解決する手法を井岡らが提案している．井岡らの手法は，類似メソッドの集約候補において，抽出メソッドに機能的なまとまりがあるものを良い候補と考え，そのような良い候補を先に利用者へと提示することを目的としている．この手法では，類似メソッドの集約候補に対して，凝集度メトリクス COB を用いて良い候補が上位にくるように順位付けを行い利用者へと提示している．しかし，COB ではブロック単位の凝集度しか測ることができないため，文単位での凝集度が高い候補が存在しても上位に提示できない場合がある．

本研究では，COB を用いた井岡らの手法の改善を目的として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順位付けを行う．プログラムスライスを用いた凝集度メトリクスは文単位でメソッドの凝集度を測ることができる．

適用実験では，提案手法と井岡らの手法の結果の比較を行い，提案手法の方が利用者にとって有用な候補を上位に提示できていることを確認できた．

主な用語

コードクローン
リファクタリング
Template Method の形成
プログラムスライス
凝集度メトリクス

目 次

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 [15]．もし，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，その全てのコードクローンにも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．

コードクローンを取り除くため方法としてリファクタリングが挙げられる．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 [2]．リファクタリングにはさまざまなパターンがあり，そのいくつかがコードクローンを取り除くために有効である．特に，共通の親クラスを持つクラス間に存在するコードクローンとなっている類似メソッドに対して有効なリファクタリングとして，Template Method の形成というリファクタリングパターンが存在する．

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 [7]．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 [3]．このパターンを用いることで類似メソッドの共通部分を親クラスに集約することができる．しかし，Template Method の形成は複数のリファクタリングパターンを利用するため，リファクタリングの熟練者でなければ困難である．

Template Method の形成を支援する手法を政井らが提案している [14]．この手法では，類似メソッド間の差異を吸収する際に，メソッドとして抽出する部分の候補を利用者に提示することでリファクタリングの支援を行っている．しかし，入力となる類似メソッドによっては候補数が 10 万以上になることや，提示する候補の順序に意味がなく有用な候補を探すのが困難であるという問題がある．

これらの問題を解決する手法を井岡らが提案している [5]．井岡らの手法では Template Method の形成において，抽出するメソッドが機能的なまとまりをもつものを良い候補だと考え，凝集度メトリクス COB [13] を用いて候補の順位付けを行っている．しかし，COB ではブロック単位の凝集度しか測ることができないため，文単位での凝集度が高い候補が存在しても上位に提示できない場合がある．

本研究では，COB を用いた井岡らの手法の改善を目的として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順位付けを行う．プログラムスライスを用いた凝集度メトリクスは，文間の依存関係を用いて凝集度を求めるためメソッドの凝集度を文単位で測ることができる．そのため，プログラムスライスを用いた凝集度メトリクスを使用することで，既存手法より良い候補を上位に順位付けすることができると考えられる．

適用実験では，オープンソースソフトウェア上の類似メソッドに提案手法を適用した．またその結果をもとに評価アンケートを行い，提案手法と井岡らの手法の比較を行った．比較の結果，提案手法の方が利用者にとって有用な候補を上位に提示できていることを確認できた．

以降，2 節では研究背景について説明する．3 節では提案手法について説明する．4 節では適用実験と手法の評価と結果の考察について述べる．5 節では関連研究について述べ，6 節ではまとめと今後の課題について述べる．

2　背景

本研究の背景として，コードクローン，デザインパターン，リファクタリング，プログラムスライス，凝集度メトリクス，既存研究とその問題点について説明する．

2.1　コードクローン

コードクローンとは，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指す．一般的にコードクローン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．例えば，あるコードクローンを持つコード片に欠陥が存在する場合に，その全てのコードクローンに対して修正を行わなければならない．コードクローンとなっているコード片が多く存在する場合，これら全てに対する修正は大きなコストとなる．コードクローンの主な発生要因を以下に示す [1].

コピーアンドペーストによる既存のコードの再利用

既存のコードをコピーアンドペーストして，そのままもしくは一部を修正して再利用することが多い．

コーディングスタイル

エラー出力やユーザインタフェースの表示などの単純なコードはクローンになりやすい．

定型処理

データ構造へのアクセスのような繰り返し書く処理は，プログラマが処理手順を覚えてしまい似たようなコードを書く傾向がある．

データ構造の違い

同じ処理を異なるデータ構造に対して行う場合，データへのアクセス部分のみが異なるコードを複数個所に書いてしまいクローンになることがある．

パフォーマンス改善

厳しい時間制約を持つシステムにおいて，コンパイラがインライン展開を提供していない場合に，最適化するために繰り返し処理を記述することがある．

2.2　デザインパターン

ソフトウェア開発におけるデザインパターンとは，過去に発生した典型的な問題の解決方法をカタログとしてまとめたものであり，有名なデザインパターンとして GoF デザインパ

ターンがある [3]．ここでは，本研究に関連したデザインパターンである Template Method パターンについて説明する．

Template Method パターン

Template Method パターン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デザインパターンである．親クラスで実装するメソッドには処理の順序を実装して，そのメソッド内で呼び出されるメソッドの一部を抽象メソッドとして親クラスで定義する．それらの抽象メソッドを各子クラスでオーバーライドして，その子クラス独自の処理を実装する．このようにすることで，共通の処理を各子クラスで実装する必要がなくなる．また，似た処理を行う新たな子クラスを作成する場合も，抽象メソッドをオーバーライドするだけで良い．

2.3 リファクタリング

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 [2]．Fowler はソフトウェア開発における設計の不備をリファクタリングによって解消できると述べている [2]．Fowler は文献 [2] において，様々なリファクタリングのパターンを説明しており，その中にコードクローン (文献 [2] 中では “Duplicated Code”) を解消するためのリファクタリングも含まれている．コードクローンを解消するための代表的なリファクタリングとしては，メソッド抽出やメソッド引き上げが挙げられる．それらのリファクタリングの中から，本研究で対象としている Template Method の形成について説明する．

Template Method の形成

Template Method の形成は，2.2 節で説明した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．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コードクローンとなっている類似メソッドである．対象となる類似メソッド間に不一致部分が存在する場合は，不一致部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．以下に Template Method の形成の手順を示す．また，図 1 に Template Method の形成の適用例を示す．

1. 類似メソッド間の不一致部分を検出する．
2. 各子クラス固有の処理として抽出するコード片を不一致部分を含むように決定する．

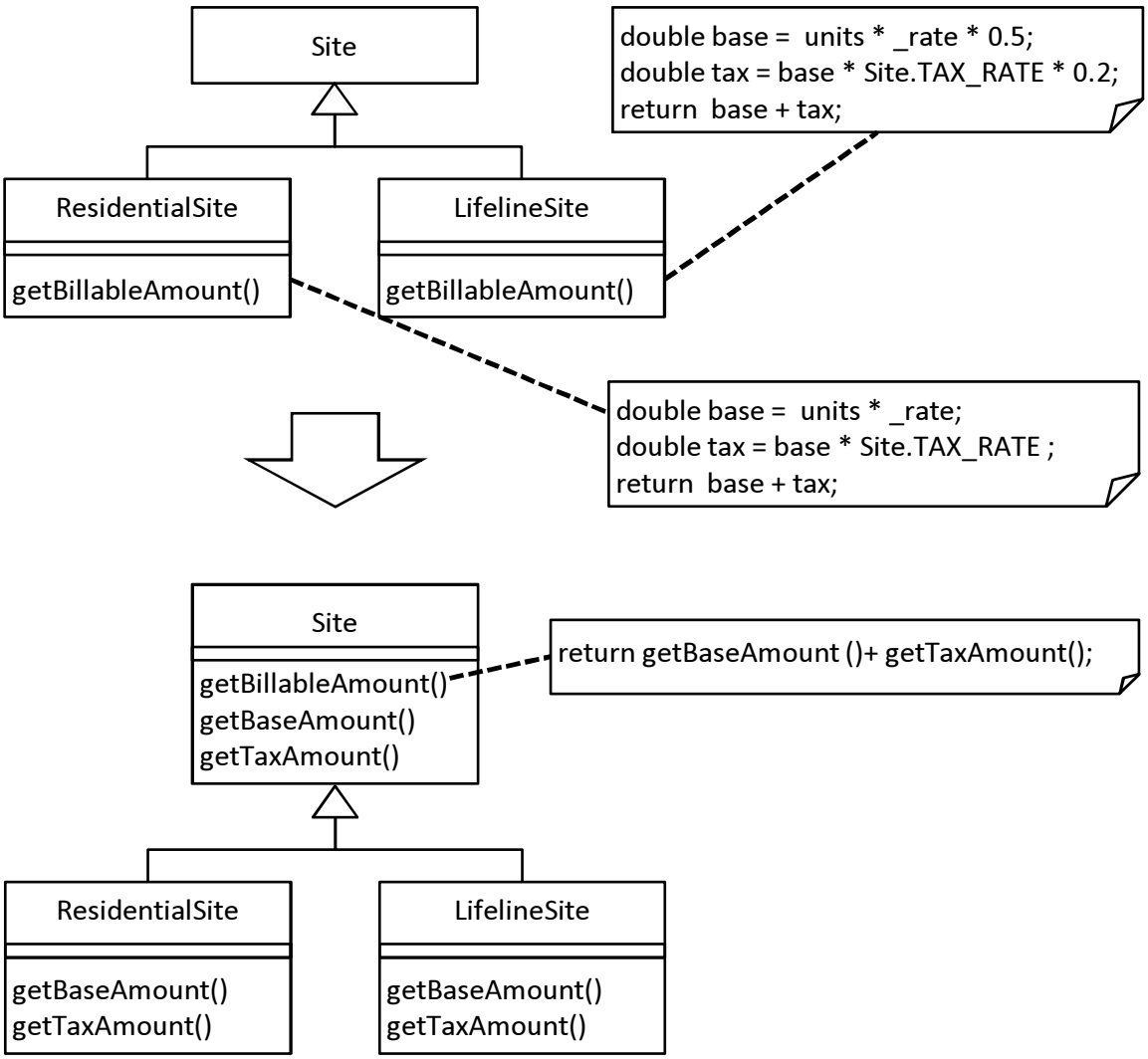


図 1: Template Method の形成の例 [2]

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．

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，ステップ 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．

2.4 プログラムスライス

プログラムスライスとは，プログラム中のある文と変数の組に対して，その組に関連のある文を集めた集合のことである [11]．以降では，プログラムスライスを求めるために必要となるプログラム依存グラフとプログラムスライシングについて説明する．

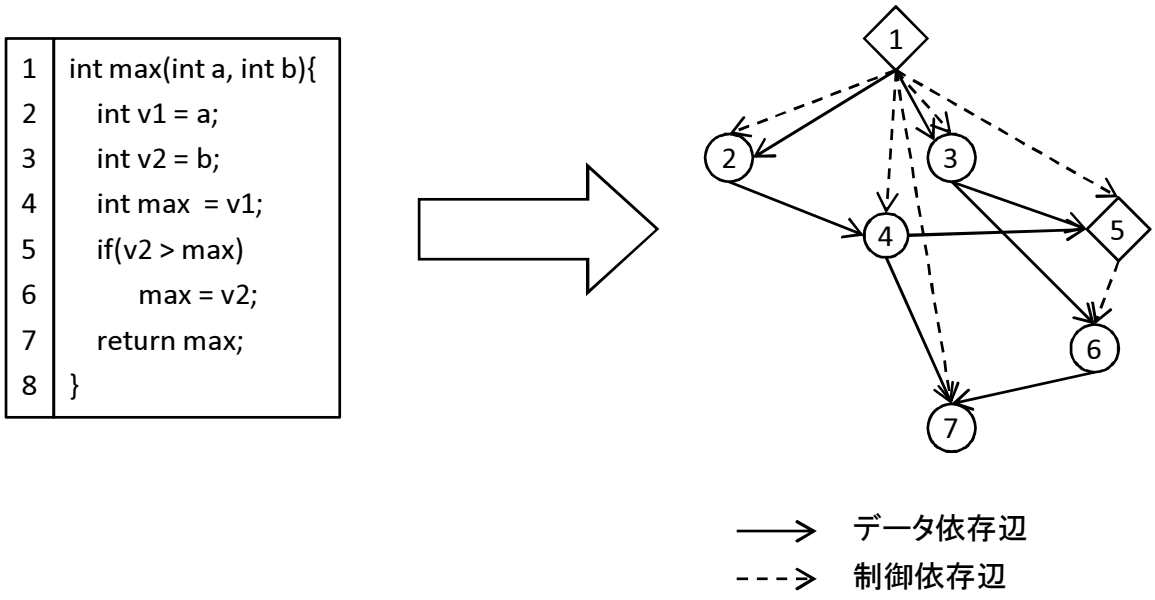


図 2: PDG の例

2.4.1 プログラム依存グラフ

プログラム依存グラフ (以下 PDG) とは，プログラムの各文を頂点，各文間の依存関係を有向辺で表したグラフである．依存関係にはデータ依存関係と制御依存関係があり，以下のように定義される．

データ依存関係

文 s1 において変数 v が定義され，その定義が変更されることなく変数 v を参照する文 s2 に到達する実行経路が 1 つ以上存在する場合，s1 から s2 へデータ依存関係があるという．

制御依存関係

文 s1 が繰り返し文または分岐文であり，s1 の結果によって文 s2 が実行されるかどうかが直接決まる場合，s1 から s2 に制御依存関係があるという．

PDG の例を図 2 に示す．図 2 の PDG では，通常の文を丸の頂点，制御文とメソッドの入口をひし形の頂点で表しており，中の数字はその頂点が表しているプログラム中の文の行番号である．一般的に PDG ではメソッドの入口を表す頂点が存在し，図中では 1 と書かれたひし形の頂点がそれに該当する．また，メソッドの入口を表す頂点からはメソッドの直下にある全ての頂点へ制御依存辺がひかれる．依存関係については，データ依存関係を実線の矢印，制御依存関係を破線の矢印でそれぞれ表している．

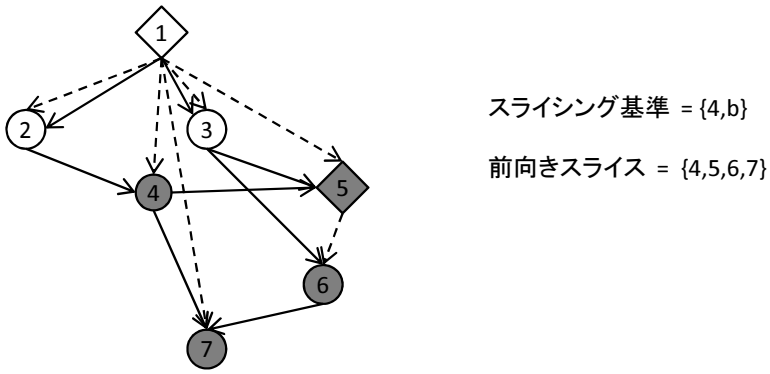


図 3: 前向きスライシング

2.4.2 プログラムスライシング

プログラムスライシングとは，PDG を用いてプログラムスライスを求める手法のことである [11]．プログラムスライシングでは，まず文と変数の組で表されるスライシング基準を定め，そこから PDG 上で依存関係をたどることによってスライシング基準と依存関係がある文の集合を抽出する．また，プログラムスライシングは PDG 上で基準点から依存関係をたどる方向によって前向きスライシングまたは後ろ向きスライシングと呼ばれる．

前向きスライシングの例を図 3 に，後ろ向きスライシングの例を図 4 に示す．図 3 では，塗りつぶされている頂点が {4,b} をスライシング基準とした前向きスライスに含まれる頂点である．前向きスライシングでは，スライシング基準となる頂点から依存関係がある頂点をスライスに加え，さらにその頂点から依存関係がある頂点を再帰的にスライスに加えていくことでプログラムスライスを求める．図 4 でも同様に，{6,max} をスライシング基準とした後ろ向きスライスに含まれる頂点を塗りつぶして表現している．後ろ向きスライシングでは，スライシング基準となる頂点への依存関係をもつ頂点をスライスに加え，さらにその頂点へと依存関係をもつ頂点を再帰的にスライスに加えていく．また，スライシングにはメソッド入口を表す頂点 (図 4 ではひし形の 1 の頂点) は含めない．

2.5 凝集度メトリクス

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したものである．一般に凝集度が高いほどモジュールの保守性や可読性が向上する．ここでは，凝集度を測定するメトリクスとして COB メトリクスとプログラムスライスを用いたメトリクスについて説明する．

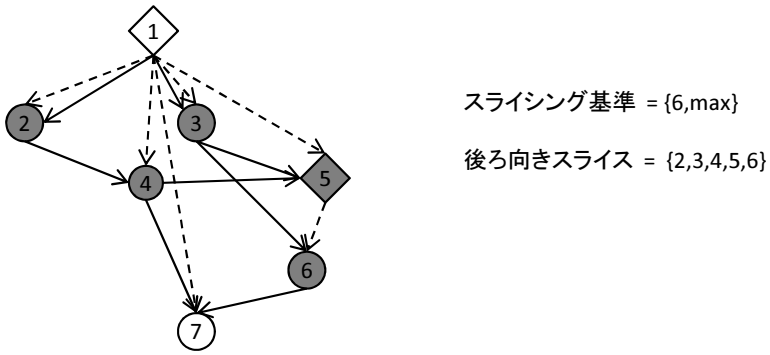


図 4: 後ろ向きスライシング

2.5.1 メソッド抽出支援を目的としたメトリクス COB

COB (Cohesion Of Blocks) は三宅らが提案した凝集度メトリクスである [13]．COB は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して凝集度を算出する．COB ではコードブロック間で多くの変数を共有しているほど凝集度が高いとしている．式 (1) に COB の定義を示す．式において 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表す．

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

図 5 のサンプルコードを用いて COB の計算例を説明する．図 5 の右図はサンプルコードにおけるブロック間での変数の参照関係を抽象的に表している．サンプルコードにおいてブロック数は 4 つ，使用されている変数は 4 つである．また 4 つの変数は全て 2 つのブロックで参照されている．これらの値を上記の数式に代入して計算すると COB の値は 0.5 となる．

2.5.2 プログラムスライスを用いたメトリクス

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 [11]．その後， Ott らが定式化と実験を行い 5 つメトリクスのうち Tightness, Coverage, Overlap の 3 つの有用性が高いことを示している [10] ．

Tightness, Coverage, Overlap の定義を式 (2)，(3)，(4) に示す．式において，メソッドを M，$length(M)$ をメソッド M の文の数，V_o を M における返り値などの出力変数の集合，SL_x を変数 x を起点にした後ろ向きスライス，SL_{int} を 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．

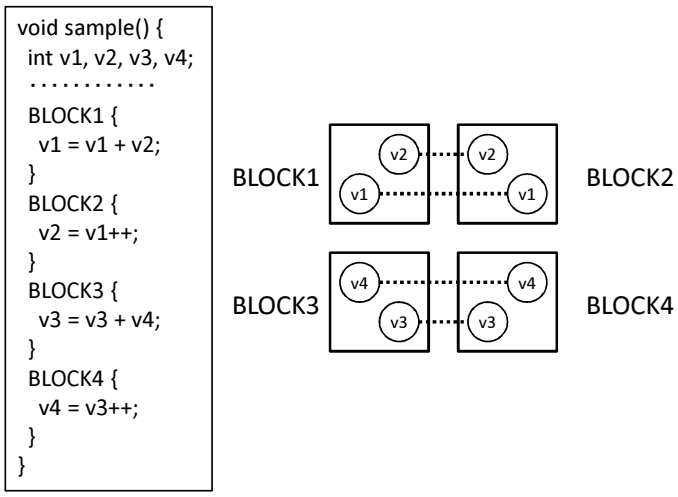


図 5: サンプルコード [13]

$$Tightness(M) = \frac{|SL_{int}|}{length(M)} \tag{2}$$

$$Coverage(M) = \frac{1}{|V_o|} \sum_{x \in V_o} \frac{|SL_x|}{length(M)} \tag{3}$$

$$Overlap(M) = \frac{1}{|V_o|} \sum_{x \in V_o} \frac{|SL_{int}|}{|SL_x|} \tag{4}$$

2.6 既存研究とその問題点

Template Method の形成におけるメソッド抽出部分の決定支援を目的とした既存研究について説明する．2.6.1 節では類似メソッドの集約候補を挙げる FTMPATool について述べ，2.6.2 節では凝集度メトリクス COB を用いて集約候補の順位付けを行う手法について述べる．

2.6.1 類似メソッドの集約候補を挙げる FTMPATool

Template Method の形成で類似メソッドの不一致部分をメソッドとして抽出する際に，その抽出候補を提示することでリファクタリング支援を行う手法を政井らが提案している．また，政井らはこの手法を FTMPATool というツールとして実装している．FTMPATool は統合開発環境 Eclipse[1] のプラグインとして実装されている [14]．以下，FTMPATool の動作手順を説明する．

[1]Eclipse, http://eclipse.org/

ステップ 1： 抽象構文木の生成

入力として与えられた類似メソッド対の抽象構文木を Eclipse の機能を用いて生成する．抽象構文木のノードは大きく以下の 2 つに分類される．

タイプ A(値や子ノードを持つノード)

“ユーザ定義名”や “return 文”などのノードが該当する．“ユーザ定義名” は値を持ち，“return 文” は子ノードを持つ．

タイプ B(子ノードの列を持つノード)

中括弧で囲まれた記述が該当する．セミコロンで区切られる文や，中括弧で区切られる記述がそれぞれ子ノードとなる．

ステップ 2： 差分となっている部分木の検出

ステップ 1 で生成された抽象構文木の比較を行い，差分となってい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て，子ノードへと再帰的に比較を繰り返すことで行う．ノードの比較では，まずノードの種類が同じかどうかを比較する．種類が異なっていれば差分として検出し，同じであればノードのタイプによって以下のように比較を行う．

タイプ A のノードの場合

ノードの持つ値や子ノードをそれぞれ比較する．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．比較には動的計画法を用いた類似文字列マッチングアルゴリズムを用いている．

差分と判断されたノードの子ノードは全て差分と判断されるので，差分となるノードは抽象構文木の部分木となる．

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．検出された部分木に対応するソースコードがメソッドとして抽出可能であるかどうかは，Eclipse の “メソッドの抽出” リファクタリングを行う事前条件判定を用いて判定する．この処理に関連する事前条件を以下に示す．

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，それらに対応する制御文が含まれていない．

条件 3 返り値を持たない return 文を含んでいる．

これらのいずれか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．メソッド抽出できないコード片が検出された場合は，抽出範囲を抽象構文木上で段階的に拡大していく．拡大と条件判定を繰り返すことで，これらの条件を満たさないメソッド抽出可能なコード片を探索する．また，抽出が可能と判定されたコード片に対しても範囲の拡大を繰り返し行い，全ての抽出範囲の組み合わせが検出されるまで処理を続ける．

ステップ 4：部分木列の分類

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．分類に使用する 3 つの条件と 6 つの分類を以下に示す．

分類条件 1 返り値の型，または値を返す変数が異なる．

分類条件 2 引数として渡す変数が異なる．

分類条件 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 分類条件 1，分類条件 2 を満たす抽出箇所が存在しない．

分類 2 分類条件 1 を満たす抽出箇所が 1 つ存在する．

分類 3 分類条件 2 を満たす抽出箇所が 1 つ存在する．

分類 4 分類条件 1，および分類条件 2 を満たす抽出箇所が 1 つ存在する．

分類 5 分類条件 3 を満たす抽出箇所が 1 つ存在する．

分類 6 分類条件 1〜3 を 1 つ以上満たす抽出箇所が複数存在する．

分類 1 はコード片をそのままメソッドとして抽出できるため，Template Method の形成を容易に行うことができる．その他の分類については，コード片を抽出した後で類似メソッドが完全に一致しないため，事前にソースコードの修正が必要となり Template Method の形成が容易であるとは言えない．

上記のように 6 つに分類された部分木列は，類似メソッドの集約候補として利用者へと提示される．利用者はその中から最適と思う候補を選び Template Method の形成を行う．

FTMPATool の問題

FTMPATool には 2 つの問題がある．1 つは入力によっては集約候補数が膨大になるという点である．FTMPATool では抽出範囲の全ての組み合わせが検出されるように探索し，検出された候補は全て利用者へ提示する．候補数が多い場合は 10 万を超えることもあり，利用者が全てを確認することは現実的ではない．2 つ目の問題は，提示される候補の順序が意味を持たないということである．FTMPATool では，候補を検出した順番で表示するだけで並び替えは行っていない．そのため候補の順序が意味を持たず，利用者が有用な候補を見つけ出すのが困難である．

2.6.2 COB を用いた類似メソッド集約候補の順位付け手法

井岡らは FTMPATool が出力する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 [5]．この手法は 2.6.1 節で説明した FTMPATool の 2 つの問題の解決を目的としている．井岡らの手法で行われている FTMPATool の問題を解決する方法について説明する．

集約候補のフィルタリング

FTMPATool の問題の 1 つとして，入力によっては候補数が膨大になるという点がある．井岡らはこの問題の解決策として集約候補のフィルタリングを提案している．フィルタリングの方法は，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 つでもある場合は候補に挙げないというものである．このフィルタリングを行う理由は，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．

凝集度メトリクス COB による集約候補の順位付け

FTMPATool の 2 つ目の問題は，提示される候補の順序が意味を持たないということである．この問題に対して，井岡らは凝集度メトリクス COB を用いた集約候補の順位付け手法を提案している．この手法では，Template Method の形成で抽出されるメソッドが機能的にまとまっているのものが利用者にとって良い候補だと考え，COB を用いて並び替えを行い，良い候補から利用者に提示することで問題の解決を図っている．

この手法では，抽出コード片を 1 つのメソッドと考えて COB を計算する．まず，抽出コード片に対して 2.5.1 節で説明した方法を用いて COB の値を算出する．類似メソッド中の全ての抽出コード片に対して計算を行い，それらの値の平均を計算して集約候補の凝集度とする．このようにして算出された凝集度を用いて集約候補の順位付けを行い，利用者へ結果を提示する．

井岡らの手法の問題

井岡らの手法は FTMPATool の問題を解決するために凝集度メトリクス COB を用いて順位付けを行っている．しかし，COB は計算時間は早いがブロック単位でしか凝集度を測ることができないという問題がある．COB では計算の過程でブロック間の変数の参照情報しか使用しないため，文単位では凝集度の高い候補であっても，ブロック間での変数の共有が少なければ COB の値は低くなる．このように，COB では凝集度の高い候補を上位に順位付けできていない場合が考えられる．

3　提案手法

既存研究の問題として説明した，COB ではブロック単位の凝集度しか測ることができないという点について，提案手法ではプログラムスライスを用いたメトリクスを使用することで解決を図る．プログラムスライスを用いたメトリクスは，文間の依存関係を用いて凝集度を算出することで文単位の凝集度を測ることができる．このことから，プログラムスライスを用いたメトリクスを使用することで，既存手法より利用者にとって有用な候補を上位に順位付けできると考えられる．

提案手法では，2.5.2 節で説明したプログラムスライスを用いたメトリクスを応用して使用する．既存のメトリクスでは，凝集度の計算に出力変数を基準としたスライスを使用しているが，提案手法では引数も基準として使用する．出力変数を基準とする場合は既存のメトリクスと同様に後ろ向きスライスを使用し，引数を基準とする場合は前向きスライスを使用する．提案手法では Tightness, Coverage, Overlap の 3 つのメトリクスを応用したものをそれぞれ FTightness, FCoverage, FOverlap とした．それぞれのメトリクスの定義を式 (5), (6)，(7) に示す．式において，メソッドを M，$length(M)$ をメソッド M の文の数，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．

$$FTightness(M) = \frac{|SL_{int}|}{length(M)} \tag{5}$$

$$FCoverage(M) = \frac{1}{|V_i|+|V_o|}\left(\sum_{x\in V_i}\frac{|FSL_x|}{length(M)} + \sum_{x\in V_o}\frac{|BSL_x|}{length(M)}\right) \tag{6}$$

$$FOverlap(M) = \frac{1}{|V_i|+|V_o|}\left(\sum_{x\in V_i}\frac{|SL_{int}|}{|FSL_x|} + \sum_{x\in V_o}\frac{|SL_{int}|}{|BSL_x|}\right) \tag{7}$$

提案手法は以下の手順で行う．図 6 に提案手法全体の流れを示す．

ステップ 1： FTMPATool が出力する集約候補の取得

ステップ 2： 集約候補のフィルタリング

ステップ 3： プログラム依存グラフの構築

ステップ 4： 凝集度メトリクスの計算

ステップ 5： 集約候補の順位付け

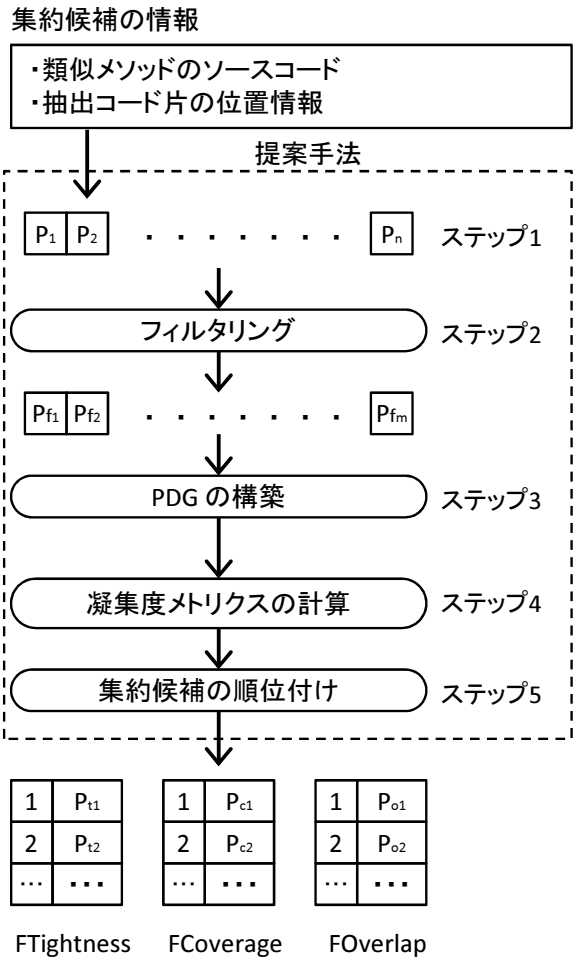


図 6: 提案手法の全体の流れ

3.1 ステップ 1 : FTMPATool が出力する集約候補の取得

FTMPATool を実行して，各候補の抽出コード片の情報を取得する．FTMPATool からはリファクタリング対象となる類似メソッド対のソースコードと，類似メソッド上での抽出コード片の位置情報を取得する．

3.2 ステップ 2 : 集約候補のフィルタリング

利用者に明らかに不適切と思われる候補を提示しないことと，凝集度の計算時間を削減することを目的として，集約候補のフィルタリングを行う．フィルタリングには，既存手法と同様に抽出元メソッドの大きさに基づく閾値を用いる [5]．

3.3 ステップ 3 : プログラム依存グラフの構築

対象としている類似メソッドの PDG をソースコード解析ツール MASU を用いて構築する [4].

3.4 ステップ 4 : 凝集度メトリクスの計算

集約候補に対する凝集度の計算においてはまず，各抽出コード片に対してプログラムスライスを用いた凝集度メトリクスの値を計算する．この処理では，抽出コード片をメソッドに抽出したとして凝集度を計算する．抽出コード片が抽出後に図 7 のようなメソッドになるとして，凝集度の計算手順について説明する．凝集度の計算においてはまず，スライシング基準を定める．スライシング基準は引数となる変数とそれらの変数を最初に参照している文の組，返り値になる変数と return 文の組とする．図 7 においては，引数となる変数とそれらの変数を最初に参照している文の組は {2,a} と {3,b} となる．図 7 の FSL_a と FSL_b はこれらの基準から計算した前向きスライスを表しており，縦棒が記述されている文がスライスに属する文である．また，返り値になる変数と return 文の組は {7,max} となり，図 7 の BSL_{max} はこの基準から計算した後ろ向きスライスを示している．これらのスライスの計算が終了したら，全てのスライスの積集合 SL_{int} を求める．このようにして求めたスライスを用いて凝集度を計算すると図 7 の下部に示した値になる．

抽出コード片に対する凝集度の計算が終了したら，類似メソッド中の全ての抽出コード片のメトリクスの平均値を図 8 のように計算して，その平均値を集約候補の凝集度とする．

3.5 ステップ 5 : 集約候補の順位付け

ステップ 4 で計算した凝集度を用いて集約候補の順位付けを行う．提案手法ではメトリクスごとに個別に順位付けを行う．順位付けの処理が終了したら，出力としてそれぞれのメトリクスによって順位付けされた候補を利用者へと提示する．

		FSLa	FSLb	BSLmax	SLint
1	int max(int a, int b){				
2	int v1 = a;	|		|	
3	int v2 = b;		|	|	
4	int max = v1;	|		|	
5	if(v2 > max)	|	|	|	|
6	max = v2;	|	|	|	|
7	return max;	|	|	|	|
8	}				

$$FTightnees = 0.5$$

$$FCoverage = 0.6167$$

$$FOverlap = 0.8333$$

図 7: 抽出コード片に対する凝集度の計算例

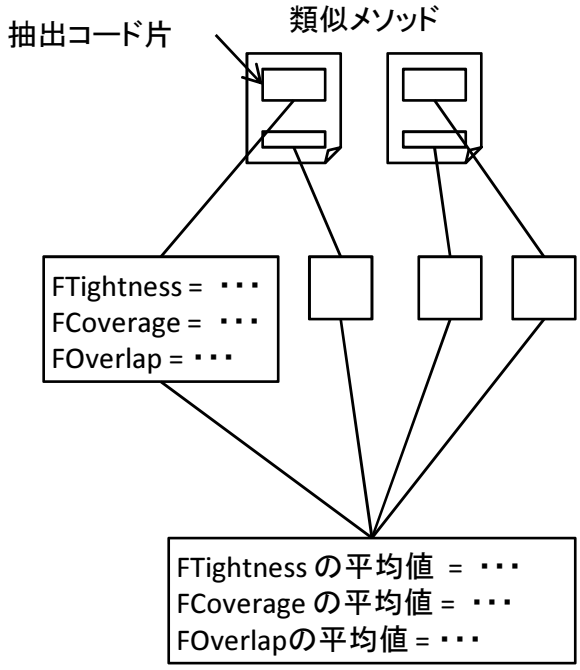


図 8: 集約候補に対する凝集度の計算

4 適用実験

適用実験として，オープンソースソフトウェア上の実際の類似メソッドに対して提案手法を適用した．また，適用実験の結果をもとにアンケートを行い評価を行った．評価の目的は以下の通りである．

- 被験者の考えに近い候補を提示できているか.
- リファクタリング作業において提案手法が有用であるか.
- 被験者にとって有用な候補を上位に順位付けできているか.

ここでは 4.1 節で実験の準備について説明する．4.2 節では実験方法として，アンケート概要と結果に対する評価の方法を述べる．4.3 節で評価結果とそれに対する考察を述べる．

4.1 実験準備

まず，適用実験の対象について説明する．実験対象のオープンソースソフトウェアには，Apache Ant[2], Antlr[3]，Azureus[4] の 3 つを用いた．実験対象としたオープンソースソフトウェア上の類似メソッドを表 1 に示す．本研究では，結果の比較を行うため既存研究の評価で使用されたものと同じ類似メソッドを対象として実験を行っている．

まず，これらの類似メソッドに FTMPATool を適用して集約候補を取得する．本実験では，Template Method の形成が容易である分類 1 の集約候補のみを対象とした．各プロジェクトにおける分類 1 の集約候補数は，Ant プロジェクトが 34, Antlr プロジェクトが 23, Azureus プロジェクトが 479 である．各類似メソッドのソースコードは付録として末尾に掲載する．

表 1: 実験対象の類似メソッド

プロジェクト	クラス	メソッド
Apache Ant	Arc	executeDrawOperation
	Ellipse	executeDrawOperation
Antlr	CppCodeGenerator	genErrorHandler
	JavaCodeGenerator	genErrorHandler
Azureus	MD5	digest
	SHA1	digest

[2]Apache Ant, http://ant.apache.org/
[3]Antlr, http://www.antlr.org/
[4]Azureus, http://azureus.sourceforge.net/

4.2 実験方法

4.2.1 アンケート概要

適用実験の結果をもとに，提案手法の評価のためにアンケートを行った．評価アンケートの被験者はコンピュータサイエンス専攻の研究室に所属する学生 15 人である．アンケート対象とした類似メソッドに対して，以下の三つの質問を行った．

1. 自分がリファクタリングを行うならどのようにコード片を抽出するか．
2. 提案手法によって上位に順位付けされた候補が良い候補であるかどうか．
3. 設問 2 で上位に順位付けされた候補を見て設問 1 から考えが変化したか．変化したならどのようにコード片を抽出するか．

設問 2 では 3 つのメトリクスによって順位付けされた候補からそれぞれ上位 10 候補ずつを選択して被験者に提示した．ただし，Antlr プロジェクトにおいてはフィルタリングによって全候補数が 6 つになったため，6 つの候補を用いて実験を行った．

4.2.2 類似性の評価

設問 1 と設問 3 については，被験者の回答を良い候補と考え提案手法によって上位に順位付けされた候補と比較して類似性の評価を行う．設問 1 と設問 3 に対する評価には以下の 2 つの目的がある．1 つ目の目的は，上位に順位付けされた候補と被験者の回答の類似度を求めることで，提案手法によって上位に順位付けされた候補が被験者が良いと考える候補に近いものかどうかを評価することである．2 つ目の目的は，提案手法によって上位に順位付けされた候補を見ることによって被験者がどのような影響を受けるかを調べることである．

候補の比較については，候補間の類似度を定義して使用した．まず，類似度を求める過程で使用する定義について説明する．

定義 1 2 つのコード片 $CF1$ と $CF2$ の類似度として，$sim(CF1, CF2)$ 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．

$$sim(CF1, CF2) = \frac{|lines(CF1) \cap lines(CF2)|}{|lines(CF1) \cup lines(CF2)|}$$

定義 2 メソッド M において，類似メソッド間の不一致部分のコード片 *diff* を含んでいる抽出コード片を *contained(M,diff)* とする．

定義 3 メソッド M において，*diffs(M)* を M に含まれる類似メソッド間の不一致部分のコード片の集合とする．

Algorithm 1 similarity(CA1,CA2)

$diffs \leftarrow diffs(CA1.M1)$
for all $diff$ such that $diff \in diffs$ **do**
  $CF1 \leftarrow contained(CA1.M1, diff)$
  $CF2 \leftarrow contained(CA2.M1, diff)$
  $left \leftarrow sim(CF1, CF2) + left$
end for
$left \leftarrow left/|diffs|$

$diffs \leftarrow diffs(CA1.M2)$
for all $diff$ such that $diff \in diffs$ **do**
  $CF1 \leftarrow contained(CA1.M2, diff)$
  $CF2 \leftarrow contained(CA2.M2, diff)$
  $right \leftarrow sim(CF1, CF2) + right$
end for
$right \leftarrow right/|diffs|$

return $(left + right)/2$

次に，2 つの候補 $CA1$ と $CA2$ を比較して類似度を求める $similarity(CA1, CA2)$ の疑似コードを Algorithm 1 に示す．疑似コード中では候補 CA の 2 つの類似メソッドをそれぞれ $CA.M1$，$CA.M2$ と表す．

4.2.3 順位付けの妥当性評価

設問 2 に対しては，3 つの評価尺度を用いて評価を行う．この評価の目的は，上位に順位付けされた候補のうちどの程度が被験者にとって有用な候補であったのかと，被験者にとって有用な候補を上位に順位付けできているかを調べることである．以下，順位付けの妥当性評価に用いる 3 つの評価尺度について説明する．

1 つ目の評価尺度として，被験者の選択率を用いる．この評価尺度では，提案手法によって上位に順位付けされた候補の中に，被験者にとって有用な候補が 1 つ以上存在したかどうかを調べることを目的としている．リファクタリングを行う際は 1 つの候補を選択してそれを適用するため，上位の候補の中に 1 つでも良い候補があれば被験者にとって提案手法は有益であると言える．以下に被験者の選択率の定義を示す．なお，SUB は被験者の集合，SUB_{sel} は候補を一つ以上選択した被験者の集合を表す．

$$被験者の選択率 = \frac{|SUB_{sel}|}{|SUB|}$$

2 つ目の評価尺度として，平均候補選択率を用いる．この評価尺度では，上位に順位付けされた候補のうちどの程度が被験者にとって有用な候補であったかを調べることを目的としている．上位に順位付けされた候補の多くが被験者にとって有用な候補であれば，提案手法による順位付けは妥当であるといえる．以下に平均候補選択率の定義を示す．ここで SR_i は i 番目の被験者が提示された全ての候補の中から選択した候補の割合を示す．

$$平均候補選択率 = \frac{1}{|SUB|}\sum_{i=1}^{|SUB|} SR_i$$

3 つ目の評価尺度として，平均適合率を用いる．平均適合率とは検索エンジンなどの順位の評価に使用される評価尺度であり，正解とされるものを上位に提示できているかを評価するものである [12]．本研究では，この平均適合率を用いて被験者にとって有用な候補がどれだけ上位に順位付け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 番目の候補が順位に現れた時点での適合率を表している．i 番目の候補が順位に現れた時点での適合率とは，提示されている i 個の候補のうちの正解候補の割合である．平均適合率を計算する際には，設問 2 において被験者が良いと回答した候補を正解集合として用いる．平均適合率の計算例を図 9 に示す．

$$平均適合率 = \frac{1}{|A|}\sum_{i=1}^{|A|} P(A_i)$$

4.3 実験結果と考察

ここでは，アンケート結果をもとに提案手法の評価を行った結果を示す．また，結果を既存手法と比較する．井岡らは既存手法の評価として，本研究と同様のアンケート評価を行っているためその結果を用いて比較を行う．ただし，本研究で行ったアンケートと井岡らが行ったアンケートは被験者と被験者数が異なっている．井岡らが行ったアンケートの被験者はコンピュータサイエンス専攻の研究室に所属する学生 9 人である．

4.3.1 類似性の評価

ここでは，4.2.2 節で述べた類似度の定義に従って，設問 1 と設問 3 における被験者の回答とそれぞれのメトリクスによって上位に順位付けされた候補とを比較した結果を示す．図 10 に Ant プロジェクト，図 11 に Antlr プロジェクト，図 12 に Azureus プロジェクトに対

正解集合 = {B,D,J}

順位	候補	適合率
1	A	0
2	B	0.5
3	C	0.333
4	D	0.5
5	E	0.4
6	F	0.333
7	G	0.286
8	H	0.25
9	I	0.222
10	J	0.3

$P(B) = 0.5$

$P(D) = 0.5$

$P(J) = 0.3$

$$\text{平均適合率} \ = \frac{1}{3}(P(B) + P(D) + P(J)) \ = \ 0.433$$

図 9: 平均適合率の計算手順

する結果をまとめたグラフそれぞれ示す．なお，Azureus においては，2 人の被験者が類似度を計算できない候補を回答したため，Azureus のみその回答を除いた結果を示している．

グラフでは，既存手法で用いられている COB と，提案手法で用いている FTightness, FCoverage，FOverlap の 4 つのメトリクスに対する結果を示している．それぞれのメトリクスの左側の棒グラフは設問 1 の回答と比較した結果であり，右側の棒グラフは設問 3 の回答と比較した結果である．グラフ中の値は，まず，被験者の回答と上位に順位付けされた候補の全ての組み合わせに対して類似度を計算した後に，それらの値を平均したものである．

まず提案手法の結果について考察する．結果をみると，類似度にはプロジェクトごとにばらつきがあることがわかる．その中でも Ant では設問 3 の類似度が 7 割を超えており，被

験者の良いと思う候補に近い候補を提案手法によって提示できていることがわかる．また，設問 1 と設問 3 に対する結果を比較すると全てのプロジェクトにおいて，設問 3 の結果が設問 1 よりも高くなっていることがわかる．これは，設問 2 で上位に順位付けされた候補の一覧を見た後で，提案手法によって上位に提示した候補により近い候補を設問 3 で回答したということである．このことから，提案手法によって被験者が思い付かなかった良い候補が提示できていることがわかる．

次に，提案手法と既存手法の結果を比較する．被験者が最終的に良いと考えた候補である設問 3 の回答に対する類似度を比較すると，Ant と Antlr では提案手法の方が類似度が高く，Azureus では既存手法の方が類似度が高いという結果になった．Azureus においても FOverlap によって上位に順位付けされた候補の類似度は COB によって順位付けされた候補の類似度とほぼ同じ値となっている．このことから提案手法が有効であり，特に FOverlap による順位付けが被験者の回答に近い候補を提示できていることがわかった．

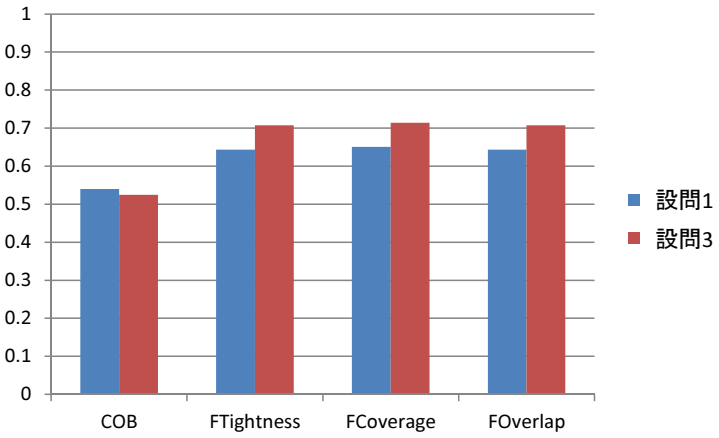


図 10: Ant プロジェクトにおける類似度

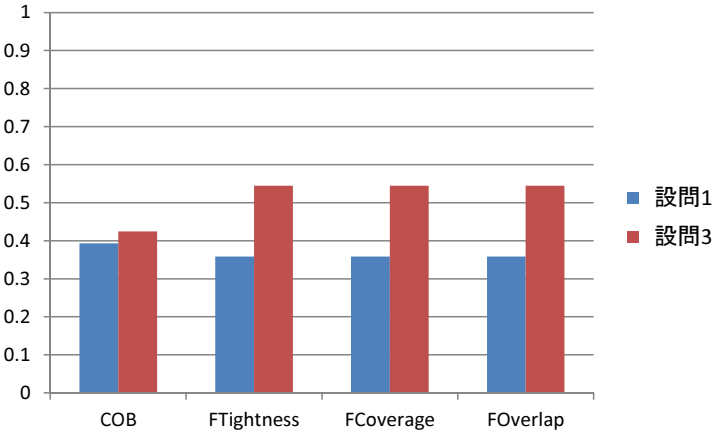


図 11: Antlr プロジェクトにおける類似度

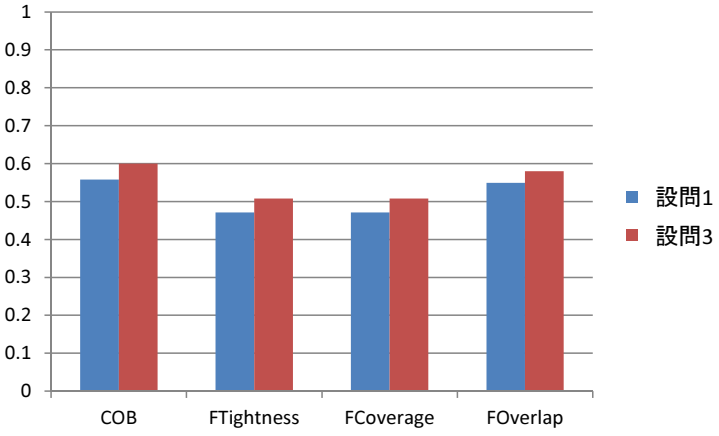


図 12: Azureus プロジェクトにおける類似度

4.3.2 順位付けの妥当性の評価

ここでは，設問 2 に対する被験者の回答をもとに，4.2.3 節で述べた 3 つの評価尺度を用いて提案手法の順位付けの妥当性を評価した結果を示す．図 13 に被験者の候補選択率，図 14 に平均候補選択率，図 15 に平均適合率の結果をまとめたグラフそれぞれ示す．グラフでは，結果をプロジェクトごとにまとめてあり，棒グラフは左から COB , FTightness，FCoverage，FOverlap に対する結果を表している．また，平均適合率は被験者ごとに算出したものを平均した値を示している．

まず，提案手法の結果について考察する．結果を見ると Ant と Antlr では 3 つのメトリクスで結果に大きな差はなかったが，Azureus のみ結果に差が見られた．まず，結果に差があった Azureus の FTightness，FCoverage 以外について考察する．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にとって，少なくとも 1 つは有用な候補を提示することができているのがわかる．平均候補選択率からは，提示した候補のうち 2 割以上が良い候補として判定されていることがわかる．また，平均適合率の値は Ant において 0.5 以上と高い結果になった．

Azureus の FTightness，FCoverage では結果が他のものに比べて悪かった．その理由として，Azureus の類似メソッド内で同名のメソッド呼び出しが連続している部分があったことが考えられる．これらの同名のメソッド呼び出しは機能的にまとまっていると考えられるものだが，他の文と依存関係がないためにいずれのスライスにも含まれない．そのため，これらの同名のメソッドをまとめた候補において FTightness と FCoverage の値が低くなり上位に提示されなかったために結果が悪くなったと考えられる．

次に既存手法との比較結果について考察する．被験者の候補選択率と平均候補選択率は，Azureus 以外では提案手法の方が良い結果となっている．Azureus においては，被験者の候補選択率は提案手法の結果が既存手法の結果を下回っており，平均候補選択率では FOverlap のみ既存手法より良い結果となっている．また，平均適合率では提案手法の方が全て良い結果となっている．以上の結果より Ant と Antlr における結果では提案手法の方が優れているといえる．Azureus においては，被験者の候補選択率が既存手法を下回ったことから，既存手法に比べて，提案手法では全ての被験者に対して有用な候補を提示することはできなかったことがわかる．しかし，FOverlap の結果は平均候補選択率と平均適合率が既存手法より高い値であるため，FOverlap による順位付けは，既存手法より優れた候補を上位に順位付けできているといえる．

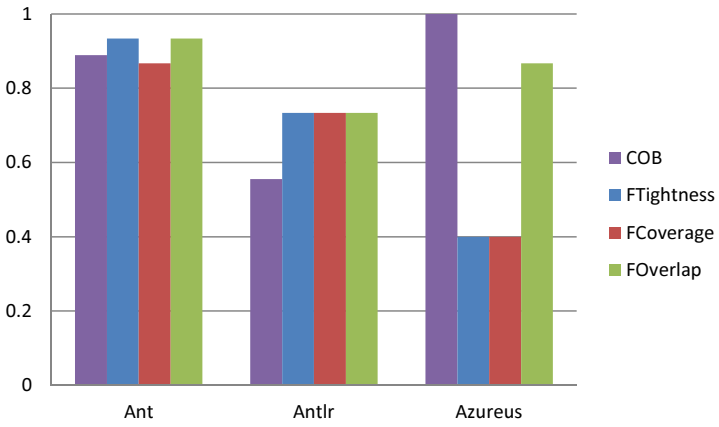


図 13: 被験者の候補選択率

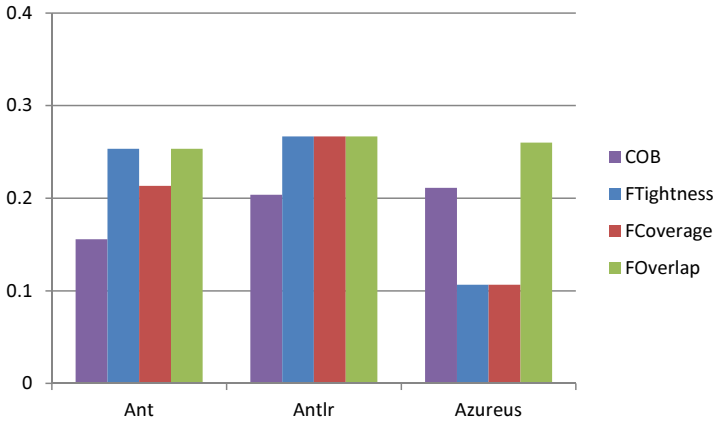


図 14: 平均候補選択率

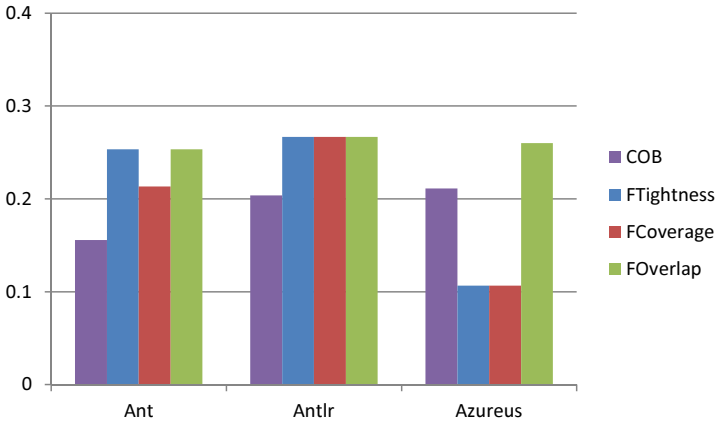


図 15: 平均適合率

5　関連研究

5.1　Template Method の形成の自動化手法

Juillerat らは Template Method の形成を自動的に行う手法を提案している [6]．この手法では，抽象構文木を用いて差分を検出してその差分を含む部分木を固有のメソッドとして抽出している．また，コード片の形式的な修正方法を定義することによって，メソッドとして抽出不可能なコード片が差分として検出されても，それらを自動的に修正してメソッドとして抽出可能にする処理を実現している．ただし，これらの自動的に決定されたコード片は必ずしも機能的にまとまっているとは言えない．

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，機能的にまとまった候補を先に提示することにより，利用者の支援とリファクタリング後のメソッドの品質向上を実現している．

5.2　凝集度メトリクスの評価

Meyers らはスライスベースのメトリクスの実験と評価を行っている [8]．この研究では，大規模ソフトウェアに対する適用結果をもとにしたメトリクスの相関関係の比較や，長期的な実験によるソフトウェアの進化とメトリクスの有用性の検証が行われている．実験の結果として，スライスベースのメトリクスで凝集度を測ることによって，低品質なメソッドの特定やその改善に有効であることが述べられている．また，Ott らはプログラムスライスを用いた様々な凝集度メトリクスについて調査している [9]．この研究では，ソフトウェアにおいて異なる粒度での観点を持つことによって，目的に合った凝集度を定義することができ，それらのメトリクスの定義にプログラムスライスが有効であると述べられている．

提案手法においては，高い品質のメソッドを抽出することを目的としたため，文単位の凝集度メトリクスを用いている．

6　まとめと今後の課題

本研究では，既存手法の改善を目的として，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 Template Method 形成時の集約候補の順位付けを行った．プログラムスライスを用いた凝集度メトリクスは，文単位でコード片の凝集度を測ることができるため，既存手法より利用者にとって有用な候補を上位に順位付けることが可能である．利用者にとって有用な候補を上位に順位付けすることで，全体の候補の中から有用な候補を探すという利用者の作業を支援することができる．

提案手法の評価として，オープンソースソフトウェア上の類似メソッドに提案手法を適用し，その結果をもとにアンケート評価を実施した．また，アンケート結果をもとに既存手法との比較を行い，提案手法の方が優れていることが確認できた．

本研究における今後の課題として，適切なフィルタリングの閾値の調査，提案手法の FTMPATool への実装，異なる被験者への評価アンケートの実施の 3 つが挙げられる．

提案手法では，不適切な候補を除外するためにフィルタリングを行っている．このフィルタリングに用いる閾値を適切に設定することで，不要な候補を少なくして利用者の選択を支援することや，全体の候補数を減らすことで凝集度を計算する時間を削減することができる．ただし，有用な候補まで除外してしまう場合も考えられるので適切な閾値を調査して決定する必要がある．

本研究では，FTMPATool の出力を取得して順位付けを行うツールを作成したが，FTMPATool への提案手法の実装は行っていない．FTMPATool へ提案手法を実装することで，Eclipse 上で提案手法を利用することができる．

また，提案手法を実装したツールをソフトウェア開発者に使用してもらい，アンケート評価を行うことも今後の課題として考えられる．本研究では評価アンケートの被験者を学生としたが，実際にソフトウェア開発を行っている開発者を被験者としてアンケートを行うことにより，異なる観点からの手法の評価と改善点の発見が期待できる．

謝辞

本研究において，常に適切な御指導及び御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします.

本研究において，随時適切な御指導及び御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授に心より深く感謝いたします.

本研究において，逐次適切な御指導及び御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 石尾 隆 助教に心より深く感謝いたします.

本研究において，終始適切な御指導及び御助言を頂きました奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学講座 吉田 則裕 助教に心より深く感謝いたします.

本研究において，適時適切な御指導及び御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 眞鍋 雄貴 特任助教に心より深く感謝いたします.

本研究において，様々な御指導及び御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井岡 正和 氏に深く感謝いたします.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝いたします.

参考文献

[1] I.D. Baxter, A. Yahin, L. Moura, M. S. Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proc of the ICSM*, pp. 368 –377, 1998.

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

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

[4] Y. Higo, A. Saitoh, G. Yamada, T. Miyake, S. Kusumoto, and K. Inoue. A pluggable tool for measuring software metrics from source code. In *Proc of IWSM-MENSURA*, pp. 3–12, 2011.

[5] M. Ioka, N. Yoshida, T. Masai, Y. Higo, and K. Inoue. A tool support to merge similar methods with a cohesion metric cob. In *Proc of IWESEP 2011*, pp. 23–24, 2011.

[6] N. Juillerat and B. Hirsbrunner. Toward an implementation of the "form template method" refactoring. In *Proc of SCAM 2007*, pp. 81–90, 2007.

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

[8] T. M. Meyers and D. Binkley. An empirical study of slice-based cohesion and coupling metrics. *ACM Trans. Softw. Eng. Methodol.*, Vol. 17, No. 2, pp. 1–27, 2007.

[9] L. M. Ott and J. M. Bieman. Program slices as an abstraction for cohesion measurement. *Information and Software Technology*, Vol. 40, No. 11-12, pp. 691–699, 1998.

[10] L. M. Ott and J. J. Thuss. Slice based metrics for estimating cohesion. In *Proc of METRICS 1993*, pp. 71–81, 1993.

[11] M. Weiser. Program slicing. In *Proc of ICSE 1981*, pp. 439–449, 1981.

[12] R. B. Yates and B. R. Neto. *Modern Information Retrieval.* Addison Wesley, second edition, 2011.

[13] 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.

[14] 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. 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.

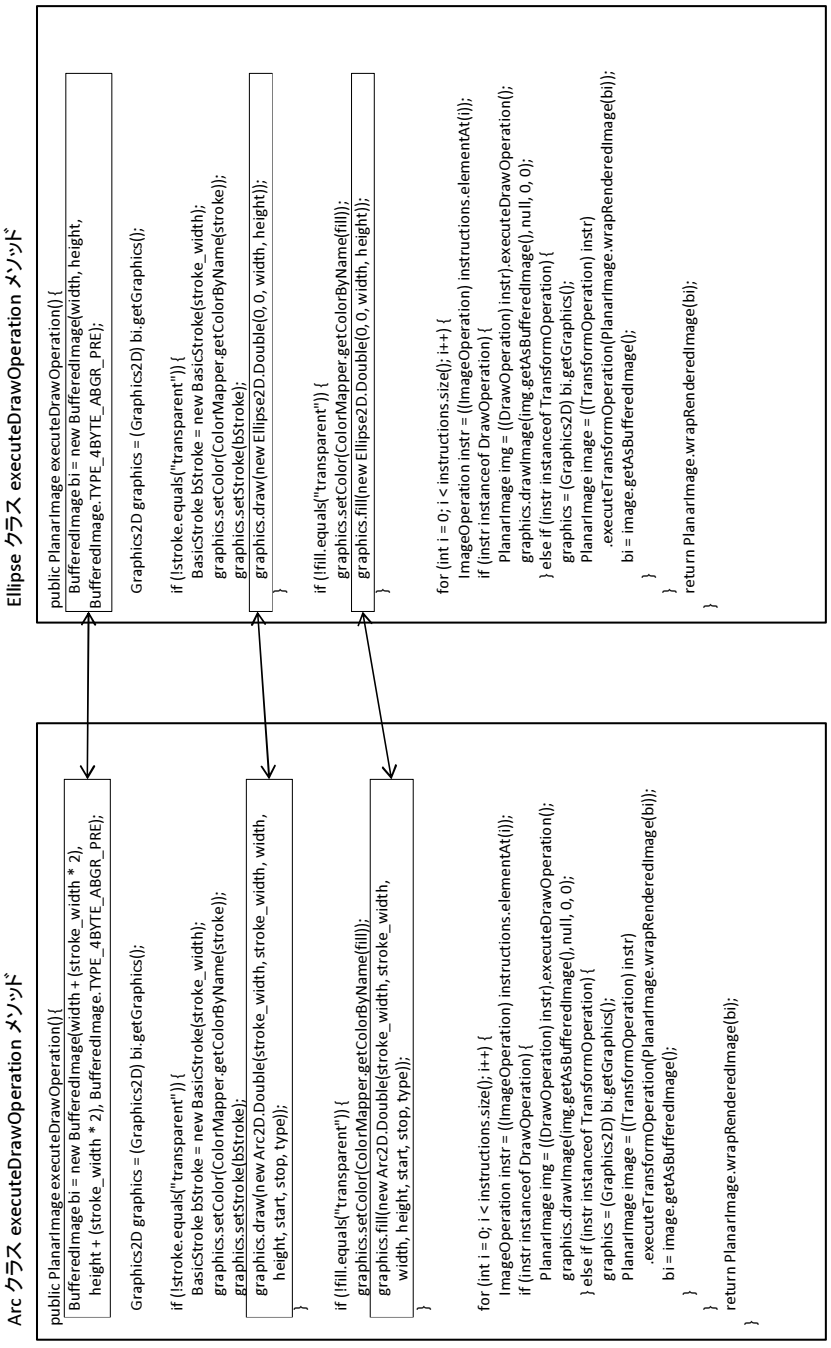
[15] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

付録

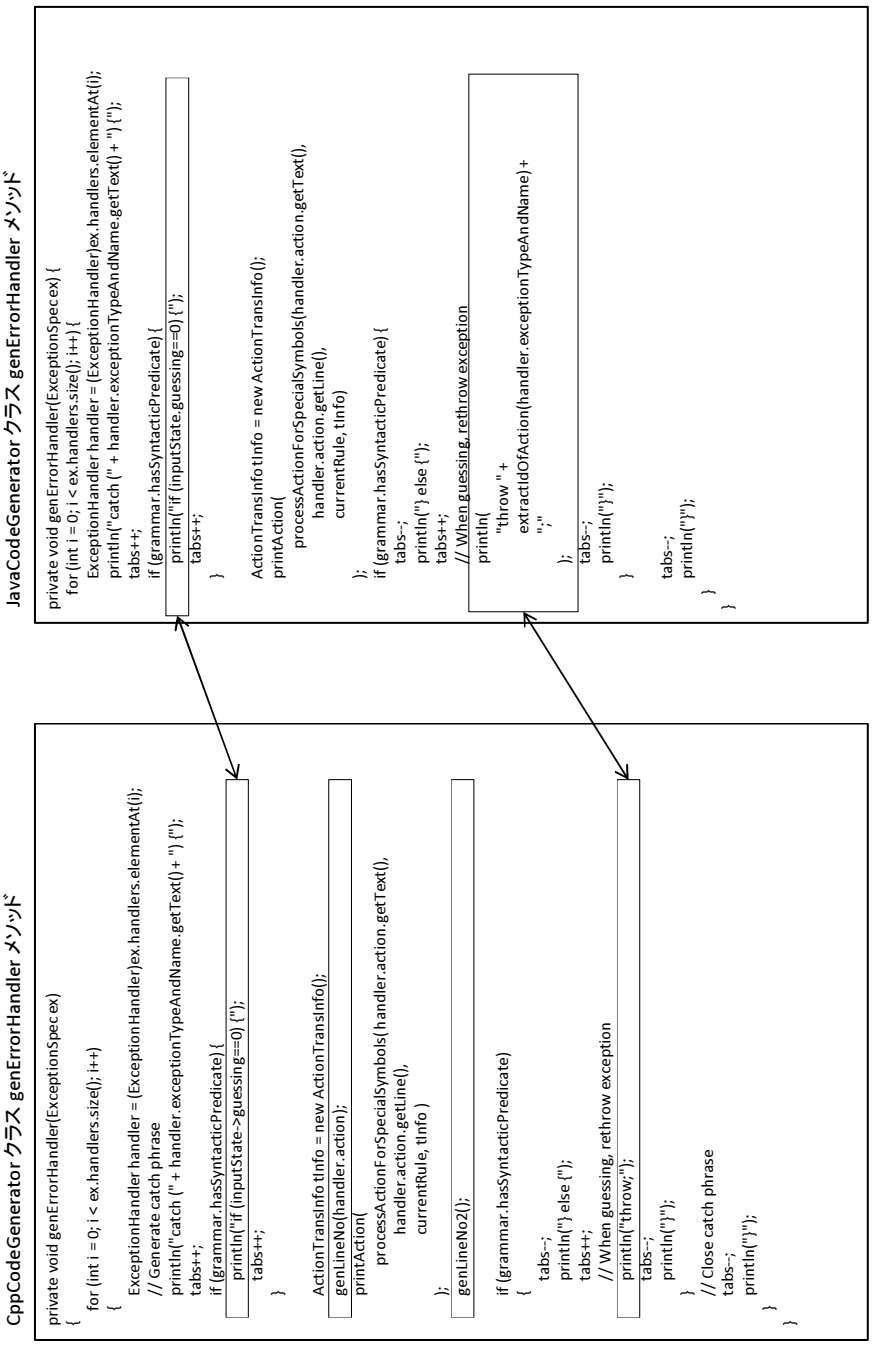
A 実験対象メソッド

実験で使用した類似メソッドを掲載する．メソッド中の四角で囲まれている箇所は類似メソッド間の差分である．

Ant プロジェクト



Antlr プロジェクト



Azureus プロジェクト