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

信学技報
TECHNICAL REPORT OF IEICE.

スライスに基づく凝集度を用いて自動分割を行うプログラム理解支援手法

平山　力地† 　吉田　則裕† 　飯田　元†

† 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 8916–5
E-mail: †{rikichi-h,yoshida}@is.naist.jp, ††iida@itc.naist.jp

あらまし　ソフトウェア開発では，ソフトウェアの保守に多くの時間とコストが費やされている．開発者は保守作業において，ソースコードを実現している機能ごとに区切りながら，各コード片が実現している機能を理解する．本稿では，プログラムスライスに基づく凝集度を用いて協調し合う複数の文を求めることで，ソースコードを実現している機能ごとに分割・提示する理解支援手法を提案する．提案手法をツールとして実装し，複数の機能を含むメソッドに対してケーススタディを行った．その結果，機能と考えられるコード片を検出できる場合があることを確認できた．
キーワード　プログラム理解，凝集度，プログラムスライシング，ソフトウェア保守

Automatic Program Segmentation Using Slicing-based Cohesion Metrics for Supporting Comprehension

Rikichi HIRAYAMA†, Norihiro YOSHIDA†, and Hajimu IIDA†

† Graduate School of Information Science, Nara Institute of Science and Technology
Takayama-cho 8916–5, Ikoma-shi, Nara, 630–0192 Japan
E-mail: †{rikichi-h,yoshida}@is.naist.jp, ††iida@itc.naist.jp

Abstract　During software development, much time and cost have been spent for software maintenance. Typically, software developers segment source code into code fragments, each of which implements a single feature in it, and then understand every feature implemented in each of the code fragments individually. In this paper, we propose an approach based on slicing-based cohesion metrics to dividing source code into cohesive fragments, each of which implements a single feature in it. As a case study, we applied a tool implementing proposed approach into examples of methods including multiple features, and then confirmed the cases that the tool suggested code fragments, each of which implements a single feature in a method.
Key words　Program comprehension, cohesion metric, program slicing, software maintenance

1. はじめに

近年のソフトウェア開発では，ソフトウェアの保守や拡張に多くの時間とコストが費やされている．この保守に費やされるコストは，ソフトウェア開発全体にかかるコストの 3 分の 2 となっており，中でもソースコードを読み理解する作業は多くの時間を要する [1] [2] [3]．そのため，ソースコードを読み理解する作業を支援し，時間とコストを削減するための手法が求められている．

ソースコードは開発者が理解するためには膨大であり，各ソースコードの先頭から 1 行ずつ読んでいき，ボトムアップに全体を理解することは困難である．そのため開発者はソースコードを読む際，ソースコードの先頭から 1 行ずつ読んでいくのではなく，初めにドキュメントなどを読み全体的なシステムの理解から始め，その次に各メソッドの役割などをトップダウンに理解していく．メソッドを理解する際も同様であり，まずコメントなどを読み，そのメソッドの役割を理解した上で，メソッド内のソースコードを単一の機能を実現するコード片に区切りながら，各コード片が実現している機能を理解し，メソッド全体を理解していく．この時，読もうとしているソースコードにドキュメントやコメントが書かれており，ソースコード自体にモジュール化が充分に成されていた場合，ソースコードの理解はスムーズに行われる．しかし実際はそのようなソースコードばかりでなく，ドキュメントやコメントが書かれておらず，モジュール化も不充分なソースコードが存在し，そのようなソースコードをトップダウンに理解することは困難であ

る [3] [4]．そのようなソースコードでは，単一のメソッド内に複数の機能を実現するコード片が存在していたり，また複数のメソッドに 1 つの機能を実現するコード片が跨って実装されていることがあり，開発者はソースコードの理解にさらに時間を費やすこととなる．

そこで本稿では，ソースコードを解析し単一の機能を実現するコード片単位で開発者に提示することで，開発者のソースコードの理解を支援する手法を提案する．またソースコードは機能ごとに完全に分割できるものではなく，複数の機能に含まれるステートメントが存在すると考えられる．そのためソースコードを単純に分割するのではなく，ステートメントの被りを考慮した機能候補の検出を行う．本研究では，依存関係の強いコード片の集合を 1 つの機能と仮定し，そのようなコード辺の集合を探すことでこれを実現する．ソースコード内の各要素間の依存関係からプログラム依存グラフを作成し，このプログラム依存グラフから依存関係の強いノード集合を探し出す．また依存関係を調べるために凝集度を利用するが，既存のプログラムスライスを利用した凝集度メトリクスの定義では本手法に適用できないため，本手法に適用するために新たにコード片を対象としたプログラムスライスを定義する．そして，プログラム依存グラフからプログラムスライスを利用した凝集度メトリクスを利用し依存関係の強いコード片集合を探し出すことで，開発者に単一の機能を実現するコード片の提示を行った．

本研究ではソースコードのメソッドから，機能候補を得る手法について述べ，実際にソースコードで実験を行う．複数の機能を含んでいるとされているメソッドに対して実験を行い，機能とされるコード片を検出できる場合を確認した．

2. プログラムスライスを用いた凝集度メトリクス

本節では，プログラムスライスを用いた凝集度メトリクスについて述べる．凝集度メトリクスはソースコードメトリクスの一種であり，ソースコードの品質を測るためのものである．本節で紹介する凝集度メトリクスはプログラムスライスを利用して，凝集度（対象メソッドの各文が協調し合って実装されているかの尺度）を調べる．一般に凝集度が高いメソッドの各ステートメントは互いに密な依存関係を持っており，同一の 1 つ機能を実現するためのメソッドであると考えられる．また，凝集度メトリクスに利用するプログラムスライスは，プログラム依存グラフから得ることが可能である．

図 1 は，このメソッドの変数 s と変数 f をスライス基点とした，プログラムスライスの例である．2〜5 行目が変数 s をスライス基点としたプログラムスライスであり，2 行目と 7〜10 行目が変数 f をスライス基点としたプログラムスライスである．

プログラムスライスを用いた凝集度メトリクスの代表的なものに Weiser の定義した凝集度メトリクスが存在する [5]．Weiser の定義した凝集度メトリクスは Tightness，Overlap，Coverage，Parallelism，Clustering の 5 種があるが，中でも Coverage，Overlap，Tightness が有用性が高いとされている．ここでは 4. 節で述べる適用実験で用いた Overlap を説明する．

s	f	
		01: public void compute(){
❙	❙	02: int a = 100;
❙		03: s = 0;
❙		04: for(int i = 0; i <= a; i++){
❙		05: s = s + i;
		06: }
	❙	07: f[0] = 1;
	❙	08: f[1] = 1;
	❙	09: for(int i = 2; i <= a; i++){
	❙	10: f[i] = f[i-1] + f[i-2];
		11: }
		12: }

図 1　プログラムスライスの例

凝集度メトリクスの計算を行うメソッドを M，メソッドに含まれるプログラムスライスの総数を V_o，x 番目のプログラムスライスの長さを SL_x，全てのスライスに共通する部分の長さを SL_{int} としたとき，

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

で表される．これは，対象メソッドに含まれる全てのスライスの共通部分の長さを各スライスで割ったものを，足し合わせてスライスの数で割ったものである．Overlap は各スライスに含まれる共通部分の割合を表している．Overlap が高い場合，そのメソッド内の各ステートメントは非常に高い依存関係を持っている可能性がある．図 1 の例では，全てのスライスに共通する部分の長さは 1 であり，各スライスの長さは 4 と 5 である．よって Overlap の値は 40 分の 9 となる．このように図 1 の例では，プログラムスライスを利用した凝集度メトリクスの値が小さくなることがわかる．これはメソッド内部が，明らかに変数 s を計算する箇所と変数 f を計算する箇所に分かれているためであり，モジュール化が正しく成されていないためであると考えられる．このように凝集度メトリクスの値は，そのメソッドに含まれる機能の協調度合いを表している．

3. 提 案 手 法

本節では，プログラム依存グラフとプログラムスライス，凝集度メトリクスを利用した機能候補を探し出す手法について述べる．本研究では，ソースコード内の機能候補箇所の特定のために，プログラムスライスを用いた凝集度メトリクスを使用する．2. 節で述べたように，凝集度は，そのメソッドに含まれる機能の協調度合いを表している．そこでメソッド内の任意のコード片を対象に凝集度メトリクスを計測し，そのコード片の有する機能の数を調べる．そのコード片の凝集度が高ければ，そのコード片は単一の機能を実現するためのコード片である可能性が高いためである．逆に，凝集度が低いコード片は，複数の機能の実現のためのコード片である可能性が高く，開発者のプログラム理解支援のために提示するコード片としては不向きである．

以下にメソッドから機能候補を抽出するための手順と，その

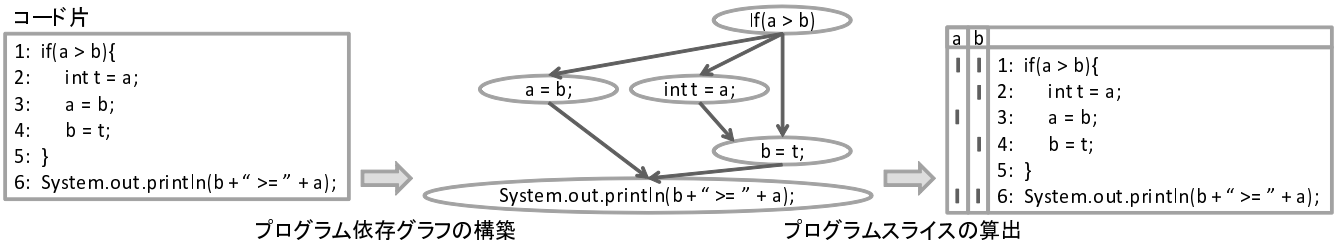

図 2　コード片を対象としたプログラムスライスの算出

具体的な手法を述べる．

3.1　手順 1: プログラム依存グラフの生成

ソースコードの対象メソッドから，プログラム依存グラフを作成する．また本研究では Java のソースコードを対象とし，プログラム依存グラフの作成には MASU [6] を用いた．

3.2　手順 2: ノード集合からのスライス基点の探索

作成したプログラム依存グラフから，解析対象となる任意の部分グラフ（ノード集合）を得る．この部分グラフは，ソースコードの任意のコード片と対応している．この部分グラフに凝集度メトリクスを適用し凝集度を得ることで，そのコード片が機能候補となりうるか調べる．しかし，既存のプログラムスライスを用いた凝集度メトリクスは単一のメソッド全体を対象としているため，単一のメソッド内の任意のコード片を対象とした凝集度を得ることはできない．そこでメソッドの一部の任意のコード片を対象とした，プログラムスライスを用いた凝集度メトリクスを定義する．

既存のプログラムスライスを用いた凝集度メトリクスでは，単一のメソッド内の任意のコード片を対象とした凝集度を得ることはできない．これは凝集度メトリクスの計算に用いるプログラムスライスのスライス基点がメソッドの出力変数と定義されているように，メソッド全体を解析することを前提に定義されているためである．そこで本研究では，単一メソッド内の任意のコード片のプログラムスライスを取得し，それを凝集度メトリクスの計算に利用することで，これに対応する．

既存のプログラムスライスでは，単一のメソッドの出力変数をスライス基点と定義しているが，これは，解析対象外へ出力する変数をスライス基点としている，と言い換えられる．これを単一メソッド内の任意のコード片を対象とできるように拡張すると，プログラム依存グラフ上の以下のノードをスライス基点とすることに相当する．

（1） プログラム依存グラフ内の解析対象範囲外のノードへ依存関係を持つ，解析対象範囲内のノード

（2） プログラム依存グラフ全体で他のどのノードにも依存関係を持たず，解析対象範囲に含まれるノード

(1) は元のプログラムスライスの定義を元に，新たに定義を拡張したものである．解析対象範囲を 1 つのメソッドと考えた場合，解析対象範囲外のノードに依存関係を持つノードは，その解析対象範囲の出力と考えられる働きを持つからである．

(2) は本研究で新たに定義したスライス基点である．プログラム依存グラフの末端ノードは，元のプログラムスライスの定義でもスライス基点に選ばれることはない．しかしプログラム依存グラフの末端ノードには，出力文などソフトウェアの機能に関わるノードがあるため，本研究ではそのようなノードも機能の一部として出力されるようにしたいと考えた．

また 1 つのスライス基点ノードから 1 つのプログラムスライスを出力するのではなく，複数のデータ被依存を持つスライス基点ノードは，各々のデータ被依存ごとに別のスライスを出力するようにしている (図 2)．これは，一般的にスライス基点はステートメントと変数であるが，プログラム依存グラフのノードのみをスライス基点とすると変数ごとのプログラムスライスを得ることができないためである．そのため図 2 のように，選ばれたスライス基点からデータ被依存ごとに複数のスライスを出力することで，変数ごとのプログラムスライスを得られるようにしている．

3.3　手順 3: 凝集度メトリクスの計算

得られたスライス基点から，単一メソッド内の任意のコード片のプログラムスライスを得る．この時，他のプログラムスライスに完全に包含されるプログラムスライスは計算に利用しない．その結果，残ったプログラムスライスを利用して凝集度メトリクスの計算を行う．本研究では，凝集度メトリクスに Overlap を利用している．

3.4　手順 4: 誤検出の除去

手順 2 と手順 3 を，解析対象メソッドのプログラム依存グラフから得られた，全てのノードの組み合わせに対して行い，誤検出を削減する．全てのノードの組み合わせに対して計算を行うのは，プログラムスライスを用いた凝集度メトリクスの値は実際に計算するまで予測しづらいためである．例えば，計算済みのノードの組み合わせに対して新たなノードを追加した時，その新たなノード集合の凝集度は，ノード追加の前後で上がる場合もあれば下がる場合もある．そのため，探索を打ち切る基準を持つことができない．そうして得られた候補から，以下の方法で誤検出を削減する．

3.4.1　凝集度の低い候補の除去

Overlap の値は 0〜1 の範囲をとるが，1 つの機能を実現するためのコード片集合である可能性のある候補は，Overlap の値が高い候補だけである．そのため，Overlap の値が低い候補は，解析結果として利用しない．

3.4.2　包含された候補の除去

前述した通り，全てのノードの組み合わせに対して解析を行い，その結果を保持するため，機能候補として得られたコード

片が，他の機能候補として得られたコード片に包含されていることがある．この両方の結果を機能候補として出力するのは冗長であるため，包含される側の候補を除去する．しかし，ソースコード上の範囲が小さいコード片の方が凝集度が高く，機能的にも重要な範囲を示している可能性がある．そこで，コード片を S_n，その範囲の凝集度を $M(S_n)$ とした時，

$$S_x \in S_y \tag{2}$$

$$M(S_x) \leqq M(S_y) \tag{3}$$

以上の両方の条件を満たした時，S_x を除去する．

3.5 手順 5: 候補の合成

手順 4 までで得られた候補は，機能候補となるコード片が小さい事が多く，開発者が認識する機能単位とは一致しないことがあると考えられる．そのため，似たようなコード片を機能候補として指す結果同士を合成することで，開発者の認識する機能単位に近付ける．本研究では，手順 4 までで得られた候補のコード片同士の差分を確認し，その差分が小さい場合のみ候補の合成を行い，合成元となった候補を除去する．

3.6 手順 6: 候補の順位付け

以上の手順を経て複数の機能候補を得られるが，開発者はその全ての候補に目を通すことは現実的ではない．そのため，機能候補を開発者に提示する順番は非常に重要である．本研究では，以下の条件を満たす結果を，高順位の候補として提示した．

（1） 合成を行っていない候補の場合，Overlap の値が高い候補

（2） 手順 5 の候補の合成の際，より多くの候補を合わせて出来上がった候補

（3） 合成前の Overlap が，より高い機能候補同士を合わせた候補

4. ケーススタディ

3. 節で述べた手法を，実際にソースコードで実験した結果を述べる．実験では，凝集度メトリクスとして Overlap を用いた．図 3 は 2. 節で凝集度メトリクスの例として紹介したメソッドである．また図 4 と図 5 のソースコードは，複数の機能を含みリファクタリングの必要のあるソースコードとして紹介されている例 [7] を用いた．

図 3 のソースコードに対する適用結果について説明する．図 3 は変数 s の値を計算する機能と，変数 f の値を計算する機能の 2 つの機能を含むメソッドとなっているが，得られた機能候補 A と B は，正しくその 2 つの機能を実現する箇所を指していることがわかる．また，どちらの機能も計算結果は変数 a の値に依存しており，このメソッドでは二つの機能が共通して使用するステートメントが存在しているが，両方の機能候補に変数 a の宣言文を含めることができた．これは凝集度メトリクスの計算にプログラムスライスを用いて，ステートメント間の依存関係を調べたため可能になったと考えられる．

図 4 のソースコードに対する適用結果について説明する．図 4 は，実験に用いたメソッドと，そこから検出された機能候補の例である．解析を行うメソッドは引数 machines の情報を書きだす機能と，引数 robot の情報を書きだす機能を含んでおり，複数の機能を持つメソッドとなっている．機能候補 C は引数 machines の情報を書きだす機能に相当し，この箇所だけで機能を実現できることがわかる．機能候補 D は引数 robot の情報を書きだす機能に相当し，これもこの箇所だけで機能を実現することができる．機能候補のどちらにも含まれていないステートメントが複数あるが，これらは全て値を決め打ちしている出力文となっており，先述した 2 つの機能に関係のある出力文とは言い難い．また，これらの出力文を機能に含めるのであれば，メソッド全体を 1 つの機能と見る方が適切であり，今回出力された 2 つの機能候補には含まれないと考えられる．機能候補 1 で出力されているステートメントの中にも，値を決め打ちしている出力文が存在しているが，これは machines の情報を書きだした場合は必要となる出力文であるため，機能に含まれている出力文であると考えられる．

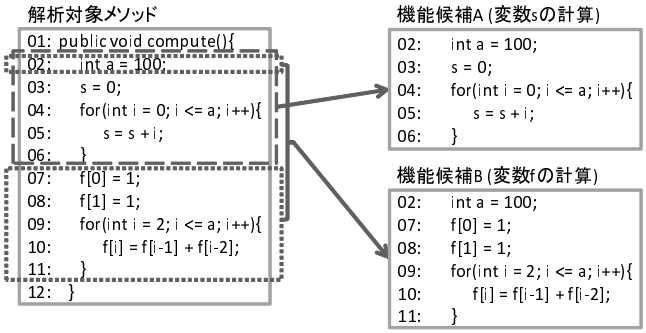

図 3 2 変数の計算を行うメソッドに対する適用結果

図 5 のソースコードに対する適用結果について説明する．図 5 は，本手法を用いて適切な機能候補が得られなかったソースコードである．このメソッドは変数 code を計算する機能と，変数 altcode を計算する機能に分けることができるが，本手法ではメソッドのほぼ全体が 1 つの機能として提示された．これは最終的にプリント出力する変数 templatePartTwo の計算を行うために変数 code の計算が必要なため，メソッド全体が templatePartTwo に依存関係を持ったためであると考えられる．しかし 3.2 節で述べたように，本手法では出力文を機能の 1 つと仮定している．そのため，図 5 は templatePartOne, altcode，templatePartTwo の 3 つの変数を計算しプリント出力するためのメソッドであると考えられる．出典である書籍 [7] で定義されていた機能領域とは異なったが，本手法で出力文を機能とした別の機能領域を見つけることができたと考えられる．

5. 関 連 研 究

以前，我々の研究グループでは各ステートメントの使用する変数に着目し，それが類似したコード片を 1 つの機能候補と見なし提示することで，ソースコードの理解支援を行った [8]．この手法では同一の機能を実現しているコード片が別メソッドにある場合でも，1 つの機能候補として同時に検出可能である．しかし我々の研究グループが提案した手法は，まとまったソースコードの領域単位で機能候補の検出を行っているため，正確

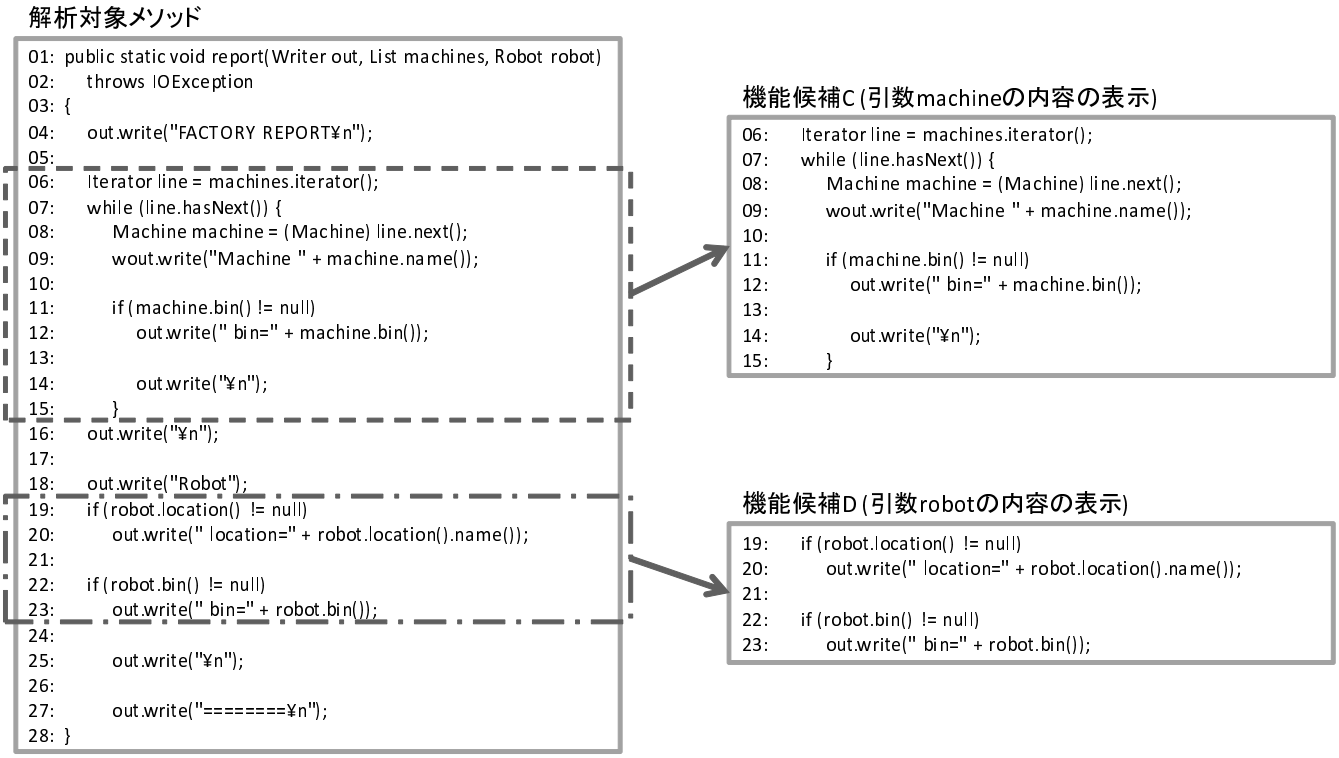

図 4　入力した 2 変数の内容を表示するメソッドに対する適用結果

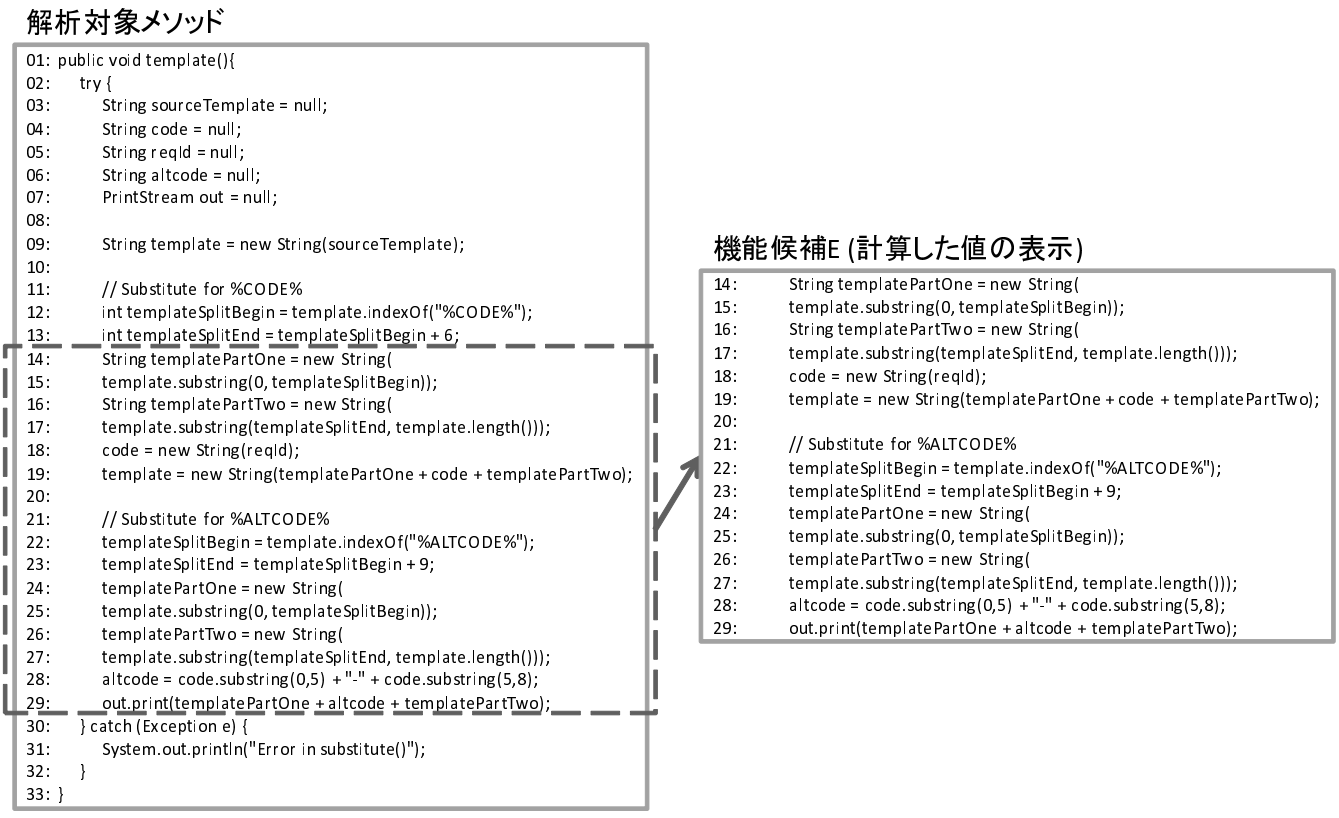

図 5　変数 code と変数 altcode の計算を行うメソッドに対する適用結果

に 1 つの機能を実現しているコード片を検出することができない．本手法では，プログラムスライスを用いてステートメント単位で解析を行っているため，より正確に検出することが可能であると考えられる．

Wang らは，ソースコードに適切に空行を入れることにより，ソースコードの可読性を改善した [9]．開発者が空行を入れる頻度の高い箇所，またガイドラインで空行を入れた方が良いと定められている箇所を分析し，自動で空行を入れるアプリを開発

した．適切な空行は可読性を上げるだけでなく，コメント挿入箇所の指標にもなるとされている．ソースコードの可読性を向上させることは，開発者のプログラム理解を支援することになるが，可読性が上がった後に開発者が行うべき作業は単一の機能を実現しているコード片ごとに区切り，理解する作業である．そのため，本手法はより直接的に開発者のプログラム理解を支援できていると考えられる．

Krinke は，ステートメントレベルの凝集度を計算することで，ソースコードの欠陥となりうる箇所の提示を行った [10]．ステートメントレベルの凝集度の計算には，Weizer の凝集度メトリクスをステートメントレベルに適用したものを利用している．それに基づいたリファクタリング手法を提案しており，ステートメントレベルの凝集度を測ることでリファクタリングの効果自体がわかりやすくなるとしている．この手法はソースコードの欠陥を事前に予測し修正するための指標となるものであり，そこを実際に修正するためにはプログラムの理解が必要であると考えられる．

Nikolaos らは，メソッドから機能を実現している箇所を特定し，その箇所を自動でメソッド抽出リファクタリングする手法を提案した [11]．機能を実現している箇所を特定する方法に，本手法と同様にプログラム依存グラフとプログラムスライスを利用している．また，機能を実現している箇所を特定するために多数のヒューリスティックに基づく手法が述べられており，それぞれの手法の有意差の検定を行っている．しかし Nikolaos らの手法も木下らの手法と同じく，Boundary Block と呼ばれる，まとまったソースコードの領域単位で機能候補の検出を行っているため，正確に 1 つの機能を実現しているコード片を検出することができない．これはメソッド抽出を前提とした機能候補の検出を行っているためだと考えられ，そのような機能候補の検出を行う場合は，機能候補の範囲外で参照・代入されている変数を含んだステートメントを機能候補に含むことができない．本手法ではメソッド抽出を前提とはしておらず，容易にメソッド抽出が可能な機能候補を検出することはできないが，より機能として正確なコード片を検出することができると考えられる．

6. おわりに

本稿では開発者のソースコード理解支援のために，メソッド内を 1 つの機能を実現しているコード片ごとに提示する手法を提案し手法の有用性の検証を行った．検証にはリファクタリングの必要があるとされているソースコードをいくつか用い，分割する必要があるとされている機能ごとに，もしくは本研究で仮定した機能ごとにコード片を提示することができた．

今後の課題としては，より多くのソースコードで評価実験をする必要がある．現在の実験は規模が小さく，実際に第三者の考える機能領域との比較を行えていない．そのため，機能候補を提示することにより，開発者のプログラム理解にどの程度影響があるか，実験する必要があると考えられる．またソースコードの解析時間の改善が必要である．3.4 節で述べたように，本手法は全てのプログラム依存グラフノードの組み合わせを計算している．そのため計算量が $O(2^n)$ となっており，大きなメソッドに対しての解析に非常に時間を要する．また，複数のメソッド間に跨る機能の提示を行いたいと考えている．これはメソッド間の依存関係を調べ，プログラム依存グラフ同士の結合を行うことで実現する予定である．

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 (C)（課題番号： 22500077）の助成を得た．

文　献

[1] 独立行政法人情報処理推進機構 ソフトウェア・エンジニアリング・センター，“ソフトウェア開発データ白書 2009,” p.27，日経 BP 社，2009.
[2] L.E. Deimel, Jr., “The uses of program reading,” SIGCSE Bull., vol.17, pp.5–14, June 1985.
[3] D.R. Raymond, “Reading source code,” In Proc. CASCON, pp.3–16, Oct. 1991.
[4] Eisenbarth, T., Koschke, R., Simon, and D.:, “Locating features in source code,” IEEE Trans. Softw. Eng., vol.29, pp.210–224, 2003.
[5] M. Weiser, “Program slicing,” In Proc. of ICSE, pp.439–449, 1981.
[6] 三宅達也，肥後芳樹，楠本真二，井上克郎，“多言語対応メトリクス計測プラグイン開発基盤 MASU の開発,” 電子情報通信学会論文誌 D，vol.92，no.9，pp.1518–1531，Sept. 2009.
[7] ウィリアム・C. ウェイク，リファクタリングワークブック，アスキー，2004.
[8] N. Yoshida, M. Kinoshita, and H. Iida, “A cohesion metric approach to dividing source code into functional segments to improve maintainability,” In Proc. of CSMR, pp.365–370, Mar. 2012.
[9] X. Wang, L. Pollock, and K. Vijay-Shanker, “Automatic segmentation of method code into meaningful blocks to improve readability,” In Proc. of WCRE, pp.35–44, Oct. 2011.
[10] J. Krinke, “Statement-level cohesion metrics and their visualization,” In Proc. of SCAM, pp.37–46, Oct. 2007.
[11] N. Tsantalis and A. Chatzigeorgiou, “Identification of extract method refactoring opportunities for the decomposition of methods,” Journal of Systems and Software, vol.84, no.10, pp.1757–1782, May 2011.