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.

凝集度によるコード片の機能別分類手法

木下　正喬† 　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: †{masataka-k,yoshida}@is.naist.jp, ††iida@itc.naist.jp

あらまし　ソフトウェアの保守や拡張では，開発者がソースコードを理解するために多くの時間とコストが費やされる．そのため，ソースコードを効率的に理解するための理解支援手法が求められている．ソースコード理解の重要なプロセスに，開発者の考える機能とそれを実現するソースコード中の記述範囲の対応付けがある．本稿ではこのプロセスを支援するため，ソースコード中のコード片（ソースコードの断片）を凝集度に基づいて機能別に分類する手法を提案する．試作ツールによる適用実験の結果，手作業による定義と同様の機能範囲を特定可能なことが確認された．
キーワード　ソースコード理解，ソースコード解析，凝集度

Finding Code Portions According to Functionality using Cohesion

Masataka KINOSHITA†, Norihiro YOSHIDA†, and Hajimu IIDA†

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

Abstract　In software maintenance, source code comprehension is an expensive task. Our research goal is to reduce maintenance time and cost. Finding code portions according to functionality is an important process in source code comprehension. We proposed an automatic method of this process using cohesion of code portions. We use a case study with our method to compare with what we detected manually.
Key words　Source code comprehension, Source code analysis, Cohesion

1. はじめに

ソフトウェア開発プロジェクトの多くが既存ソフトウェアの保守や拡張を目的としている．それらのプロジェクトでは，開発者はソースコードの機能・構造・振る舞いなどを理解する必要がある．しかし，ソースコードの理解は高コストであるため，ソースコードを効率的に理解するための理解支援手法が求められている．

ソースコードの理解が難しい原因の一つは，ソースコードの記述単位である文が詳細すぎることである．一般に一つの機能はソースコード中の複数の文が密接に協調しあうことで実現される．そのため，開発者はソースコードを理解するために協調しあう文の集合を探し出す必要があるが，膨大なソースコードの中からそれらを探し出すのは容易では無い．

これまでに，機能と文の対応づけを自動化，あるいは支援する手法が考案されてきた．Gold らは，開発者にとって興味のあるソフトウェア中のコンセプトとそれに関連するキーワードを登録した辞書を用いて，ソースコード中の文をコンセプト別に分類する手法 HB-CA（Hypothesis-Based Concept Assignment）を考案した [1]．また，丸山らはプログラム依存グラフを用いて，特定の変数の算出に必要な文の集合をメソッドとして抽出する手法を考案した [2]．この手法はプログラム理解ではなくリファクタリングを目的としているが，メソッドとして協調しあう文の範囲を特定可能である．

いずれの手法も，機能やそれに準ずる単位で関連する文の集合を抽出可能だが，辞書の作成や基準変数の選択のために対象ソフトウェアに関する事前知識が必要である．

本研究では，事前知識を用いずに機能に対応する文集合を抽出する手法を提案する．また，ソースコードの理解支援を目的とするため，次のような特徴を持つ機能を抽出の目標とする．

- 属する文集合は強く協調しあい，モジュール性が高い
- 開発者にとって理解するのに適した大きさである

我々の提案する手法は二つの大きな特徴を持つ．まず一つ目の特徴は，文よりも粒度の大きいコード片を解析の単位とすることである．コード片はソースコード中の連続する領域を指し，（ファイル ID，開始行番号，終了行番号，開始桁番号，終了桁

番号）で表される．コード片を解析の単位とするのは，ソースコード中の連続する領域は同一の機能に属する可能性が高いと考えられるからである．二つ目の特徴は，協調するコード片の組み合わせを発見するために凝集度を用いることである．一般に，凝集度の高いコード片の集合は互いに協調しあっておりモジュール性が高い．

提案手法では，まずソースコードをコードブロックなどのプログラム構文で区切り，コード片に分割する．次に，コード片の中から凝集度が高くなる組み合わせを見つけ出し，機能候補として抽出する．凝集度の性質から，機能候補に属するコード片の集合は機能と一対一で対応していると考えられる．最終的に，すべてのコード片を機能候補ごとに分類する．

今後の開発・実験計画の参考とするため，提案手法を評価する簡単な予備実験を行った．その結果，手作業で定義した機能と類似する機能候補の抽出に成功した．

次節以降では，2. 節で凝集度の定義と提案手法における利用方法について，3. 節でソースコードを解析してコード片を機能別に分類する手法について，4. 節で試作ツールによる適用実験の結果と考察を述べる．また，5. 節で本研究に関連する研究を解説し，最後に 6. 節でまとめと今後の課題を述べる．

2. 提案する凝集度メトリクス

凝集度とは，モジュール内の各構成要素が特定の機能を実現するためにどの程度協調して動作しているのかを表す度合いである．凝集度は，モジュールが単一の機能を実現していれば高い値に，逆に複数の機能を実現していれば低い値になる．一般には，凝集度の高いモジュールはより高品質であるといわれている．

LCOM（Lack Of Cohesion in Methods）はオブジェクト指向設計の複雑度に関する 6 つの CK メトリクスの一つで，クラスの凝集度の欠如を表す [3]．LCOM は，特定のデータ要素をいくつの機能要素が共有するかに着目し，一つのデータ要素が多くの機能要素で共有されていれば低い値となる（つまり，凝集度が高いことを表す）．LCOM は，データ要素をメンバ変数，機能要素をメソッドとして計算する．COB（Cohesion Of Blocks）はメソッドの凝集度を表す，LCOM と同様の着眼点から定義されたメトリクスである [4]．COB は，データ要素をメソッド内で使用されている変数，機能要素をコードブロックとして計算する．COB は，b をメソッド内のコードブロック数，v をメソッド内で使用されている変数の数，V_j をメソッド内で使用されている j 番目の変数，$\mu(V_j)$ を変数 V_j を使用しているコードブロック数として以下のように定義される．

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

提案手法では，協調するコード片の組み合わせを探すために凝集度を用いる．そのため，特定のモジュールを評価する LCOM や COB とは異なり任意のコード片集合に対して凝集度を求める必要がある．そこで，そのようなメトリクスとして COCP（Cohesion of Code Portion）を考える．COCP は任意のコード片集合について，データ要素をコード片集合に含まれる変数，機能要素をコード片として COB と同様に計算する．COCP は p をコード片集合に含まれるコード片の数，他のパラメータを COB と同様にして以下の式で定義される．

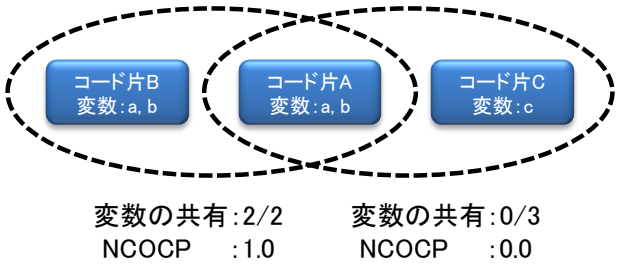

図 1　NCOCP 算出例

$$COCP = \frac{1}{p}\frac{1}{v}\sum_{j}^{v}\mu(V_j) \qquad (0 \leqq COCP \leqq 1)$$

COCP の定義域は 0 から 1 の間だが，コード片の数が少数の場合には小さな値をとらなくなるという問題がある．例えばコード片の数が 2 つのとき，全く協調していないコード片の組み合わせでも COCP の値は 0.5 となる．この性質は協調するコード片の組を探すうえで悪影響を及ぼすため，COCP をコード片の数よらず 0 から 1 の間の値をとるように正規化した NCOCP（正規化 COCP）を考える．NCOCP は COCP とコード片の数 p を用いて以下のように定義される．

$$NCOCP = \frac{COCP - 1/p}{1 - 1/p} \qquad (0 \leqq NCOCP \leqq 1)$$

図 1 は NCOCP の算出例である．コード片 A とコード片 B は共有する変数が多いため，これらの間で NCOCP は高い値になる．逆に，コード片 A とコード片 C の間では NCOCP は低い値になる．提案手法では，ソースコードの中から高い協調性を持つコードの組み合わせを探すため，コード片集合の凝集度を NCOCP を用いて計測する．

3. コード片の機能別分類

本節では，ソースコード中のコード片を機能別に分類する手法について説明する．図 2 に提案手法の概要を示す．本手法はソースコードを入力し，最終的に機能候補群を出力する．機能候補は，それぞれ単一の機能に対応すると考えられるコード片の集合のことを言う．本手法は，大まかに以下の 3 つのステップにより実現される．

ステップ 1：構文木の作成　ソースコードを解析し，関数やコードブロックなどのプログラム構文を節，それらに区切られたコード片を葉とする構文木を作成する．

ステップ 2：機能要素の抽出　ステップ 1 で得られた構文木をもとに，構文的に近い位置関係にあるコード片の組み合わせについて協調しているものを機能要素として抽出する．

ステップ 3：機能候補の抽出　ステップ 2 で得られた機能要素群から協調性の高い組み合わせを探し，機能候補として抽出

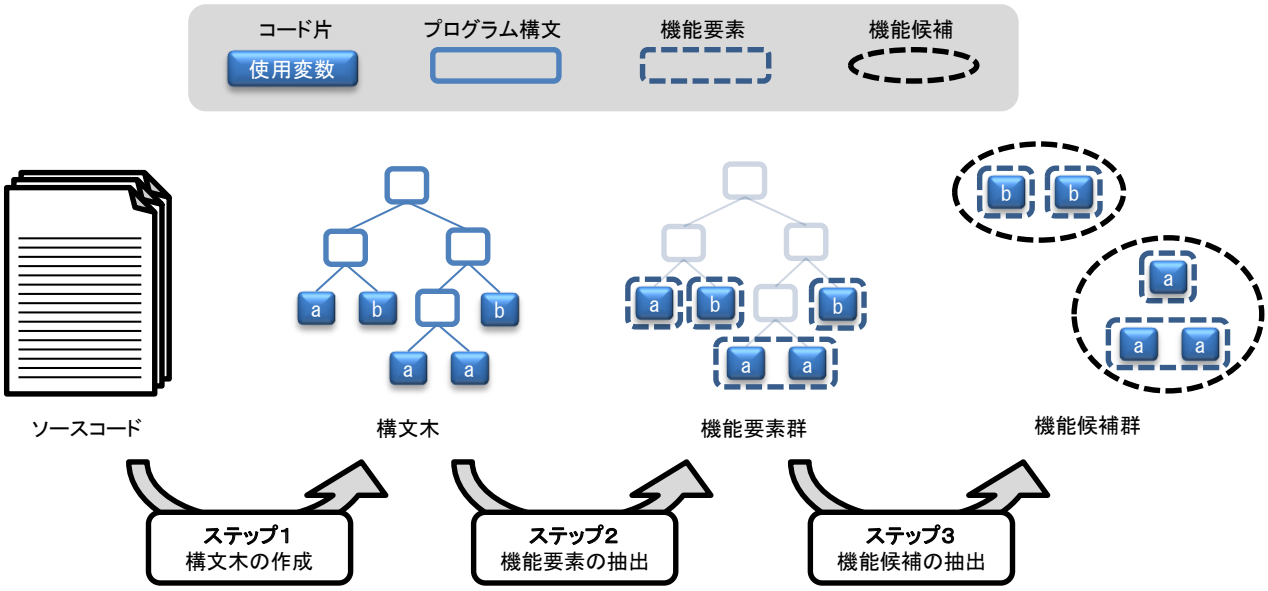

図 2　手法の概要

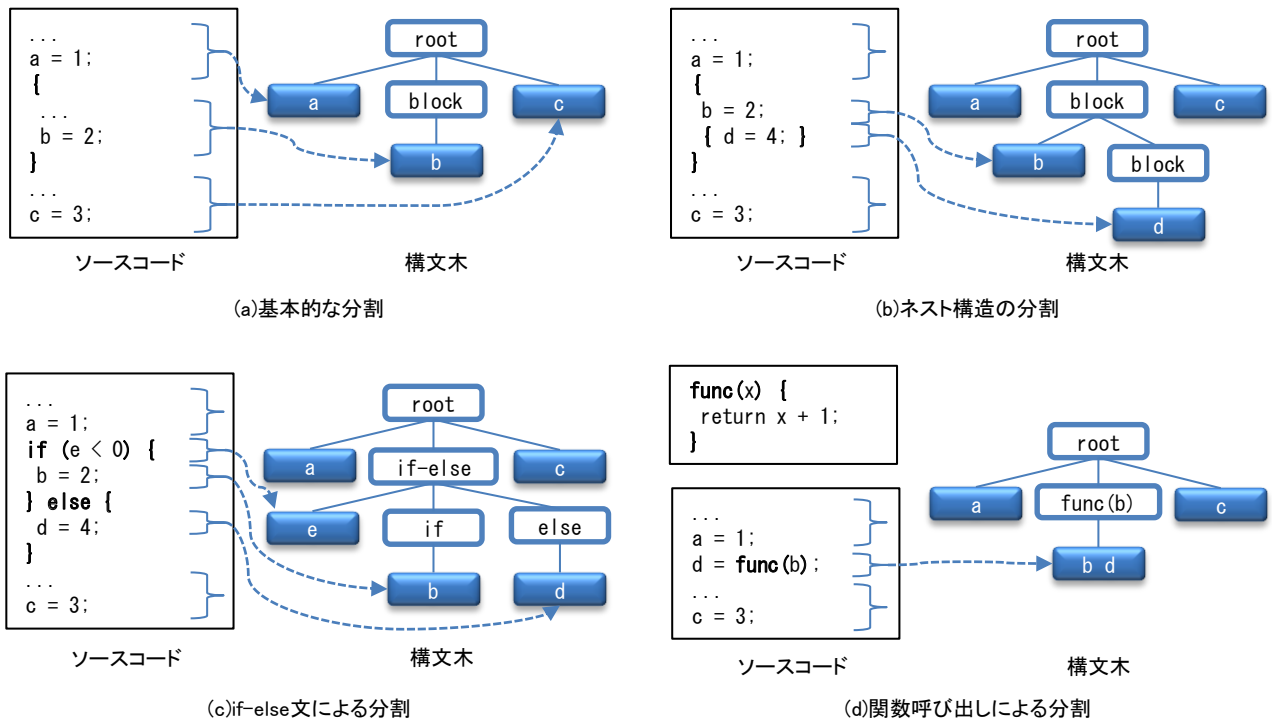

図 3　構文木の作成例

する．

以降，3.1 節から 3.3 節で各ステップの動作について述べる．

3.1　ステップ 1：構文木の作成

ソースコード中の連続する領域は，同一の機能に属している可能性が高いと考えられる．そこで，本手法では関数・コードブロック・制御構文などのプログラム構文によって区切られたコード片を解析の最小単位とする．コード片の単位で解析を行うことで，過度に詳細な範囲が機能として抽出されるのを防ぐ．また，解析の粒度が文よりも粗くなるため，文単位での解析よりも実装コストや計算コストの点で有利である．

本ステップでは，ソースコードをプログラム構文をもとにコード片に分割し，プログラム構文を節，コード片を葉とする構文木を作成する．次に，具体的な処理手順を示す．

（1）状態の初期化　ソースコード全体を最初の処理範囲とする．構文木にソースコード全体を示すルートノードを追加し，カレントノードとする．

（2）プログラム構文の探索　現在の処理範囲を先頭から走査し，コードブロックなどのプログラム構文を探す．

（3-a）ソースコードの分割（プログラム構文発見時）　（2）でコードブロックなどのプログラム構文が見つかった場合，ソー

スコードを「処理範囲の先頭から構文の先頭まで」,「構文の先頭から構文の最後まで」,「構文の最後から処理範囲の最後まで」の 3 つに分割する．構文の種類ごとの分割方法の違いについては後述する．構文前後のそれぞれの範囲について,（2）以降を再帰的に適用する．次に，見つかった構文を構文木のカレントノードに節ノードとして追加して次のカレントノードとした後,構文内部の範囲についても（2）以降を適用する．再帰は処理が（3-b）に到達するまで続けられる．

（3-b）コード片の取得（処理範囲の走査終了時）　（2）でプログラム構文が見つからずに処理範囲の走査を終了した場合,その範囲のコード片を構文木のカレントノードに葉として追加する．また，その範囲でアクセスされている変数の情報をコード片に付与する．

（4）構文木の出力　（2）と（3）でソースコードの全範囲を分割して構文木に追加し終えたら，構文木を出力する．

図 3 に，疑似プログラミング言語のソースコードから構文木を作成する例を示す．図 3 中の（a）は単純なコードブロックによってソースコードを分割する例である．構文がネストしている場合には,（b）のように再帰的に分割する.（c）は，if-else 文よる分割例である．ソースコードを制御構文によって分割する場合には，ソースコードは構文の前後及び，条件式，実行式,更新式などに分割される.（d）は外部 API などや再帰関数などを除く，展開可能な関数呼び出しによる分割例である．このような関数やメソッドは，インライン展開を行ったうえでコードブロックと同様に処理する．そのため，複数箇所で呼び出される関数等は，複数の機能に属する可能性がある．

例では疑似プログラミング言語のソースコードを対象としたが，プログラム構文ごとに適切に分割方法を定義することにより，C，C#，Java など様々な言語のソースコードを解析することができる．以降のステップはここで作成した構文木と各コード片の使用変数のみを参照するため，プログラミング言語に依存しない．

3.2　ステップ 2：機能要素の抽出

ステップ 2 とステップ 3 ではステップ 1 で得られたコード片群の中から協調していると考えられる組み合わせ求めることを目的とする．ステップ 2 では構文木を考慮し，構文的に近い位置関係にあるコード片についてステップ 3 よりも優先的に組み合わせる．コード片の組み合わせが協調しているかの判断は,2. 節で定義した凝集度メトリクス NCOCP の値が閾値を超えているかどうかで行う．

具体的な手順は，まずステップ 1 で得られた構文木の各ノードについて，属するコード片集合の NCOCP を計算する．ノードと，そのノードに属する全てのノードについて NCOCP が閾値以上であるような，最も上位のノードを探し出して機能要素とする．この処理は，コードブロックや関数などプログラム構文の単位で協調しているものを探し，できるだけ広い範囲で協調しているものを機能要素として抽出することに相当する．

図 4 に機能候補抽出の例を示す．節の左上にあるのは節番号である．例では，コード片の変数アクセスの傾向から節 2・節 3・節 5 に属するコード片の集合は凝集度が高く，節 1・節 4 に属するコード片の集合は凝集度が低い．節 2・節 3・節 5 の中で，節 3 は節 2 の下位ノードなので，残りの節 2・節 5 が機能要素となる．また，節 2・節 5 のいずれにも属さないコード片は単体で機能要素となる．

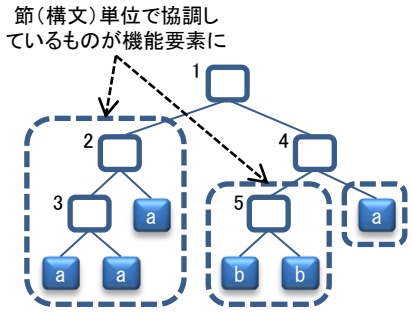

図 4　機能要素の抽出例

3.3　ステップ 3：機能候補の抽出

ステップ 3 では凝集度の高くなる機能要素の組み合わせを求め，機能候補として抽出する．ステップ 2 と異なり，組み合わせる機能要素の位置関係を考慮しないため，単一の機能に属するがソースコード中の離れた位置にある場合でも一つの機能候補として抽出可能である．機能要素の組み合わせには，凝集度を評価値として全探索や遺伝的アルゴリズムなどの最適化手法を用いる．

こうして得られた機能候補を，凝集度や大きさによってランキングして開発者に提示する．

4.　適 用 実 験

提案手法のツールを試作し，実ソースコードへの適用実験を行った．本実験は，今後の開発・実験計画を検討するための予備実験である．そのため，実験自体は非常に小規模で，一部の評価を手作業で行っている．以降では，実験の概要とその結果,考察を述べる．

4.1　試作ツール

C#言語を用いて，提案手法により C 言語と Java 言語のソースコードを解析対象とするツールを試作した．提案手法にはまだ厳密に仕様が決まっていない部分があるため，試作したツールの仕様は暫定的なものである．

機能要素の抽出では，抽出する NCOCP の閾値を 0.2 とした．機能候補の抽出では，機能要素の組み合わせ最適化手法として全探索を用い，共通変数を持たない機能要素の組み合わせなどを除く全ての組み合わせを NCOCP により評価した．また，コード片数が多い機能候補ほど NCOCP が小さくなる傾向が見られたため，コード片の数に応じて異なる NCOCP の閾値を設定した．具体的には，コード片が 4 つ以下の機能候補は NCOCP に関わらず除外，8 つ以下の機能候補は 0.1，それ以上の機能候補は 0.05 とし，残ったものを凝集度でランキングした．

4.2　実 験 方 法

対象のソースコードから 2 種類の正解集合を手作業で作成し，

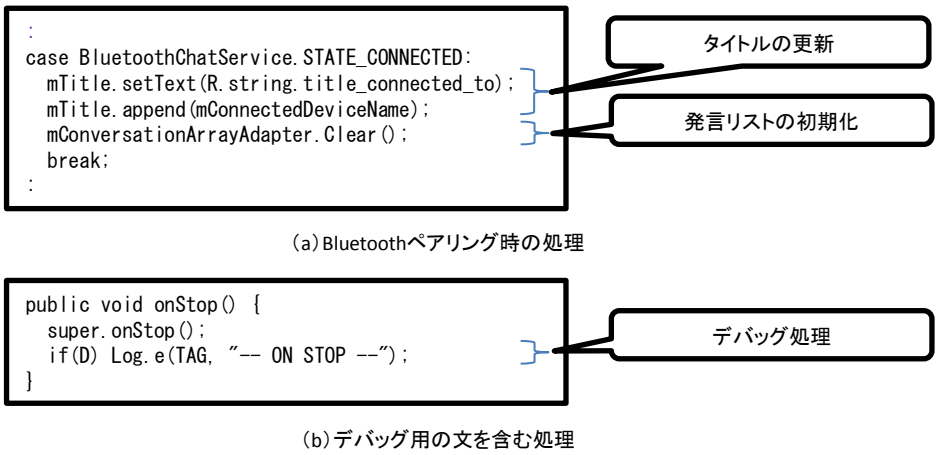

図 5　ソースコードの解析例

抽出した機能候補と比較した．

正解集合 A　主要変数の計算に必要な文の集合を一つの機能として定義する．まず，プログラム中から主要な変数を選択する．主要な変数の判断は人の手で行う．次に，その変数の更新に関わる文を依存性解析にもとづいて求める．正解集合 A は丸山らのメソッド抽出手法 [2] をベースにしている．

正解集合 B　正解集合 A は単一のメソッドであらわされるような機能しか抽出することができない．正解集合 B では，より粒度の大きな機能を定義する．まず，プログラム中から，一つの機能に属すると考えられる主要変数の組を選択する．次に，正解集合 A と同様に，その変数の組の更新に関わる文を依存性解析にもとづいて求める．

正解集合の機能と抽出した機能候補の比較は，重複部分のサイズをそれぞれのサイズの平均で割り，75%以上一致をする機能候補を正解とした．

4.3　実 験 結 果

実験は，Android SDK に含まれるサンプルアプリケーションの一つ，BluetoothChat(注1)を対象に行った．BluetoothChat は，Java 言語で作成された 1000 行程度のアプリケーションである．正解集合には，ソースコード中の 8 つの主要変数から正解集合 A として 8 つ，正解集合 B として 3 つを作成した．

試作ツールによって BluetoothChat のソースコードを解析した結果，合計 30 の機能候補が得られた．また，それらは正解集合 A の 8 つの機能のうち 1 つと，正解集合 B の 3 つの機能のうち全てと一致した．

図 5 は，ソースコードの解析例である．図中（a）は Bluetooth のペアリングに成功し，画面を更新する際の処理である．（a）では，タイトルの更新と発言リストの初期化の 2 つの処理を行っている．正解集合 A ではこれらの 2 つの処理は別々の機能に属し，正解集合 B では同一の機能に属している．本手法では，正解集合 B と同様にこれら二つの処理は単一の機能候補として抽出された．これは，本手法がコード片単位で解析を行っているため，それより詳細な粒度の機能を抽出できない反面，同一のコード片に属する変数を結びつける働きがあることを示している．(b) は，ソースコード中の一般的なメソッドの一つを示している．入力としたソースコードには多くの部分にデバッグ用の処理が埋め込まれているが，ブロックが分かれているためにこれらをうまく抽出することができた．

4.4　考　　察

4.3 節で述べた結果から，本手法では正解集合 B のような機能候補を抽出可能であると考えられる．しかし，今回の実験は非常に小規模なので，これだけでは十分な評価が行えたとは言い難い．本手法が有効であることを示すためには，様々なソースコードへの適用実験や，プログラム理解への貢献度など別の観点からの評価を行う必要がある．

また，今回の実験を進める中で解決すべき課題も明らかになった．まず，今回定義した凝集度メトリクス NCOCP はスケーラビリティが非常に低く，求める対象のコード片数が多くなるほど小さな値に偏ることが分かった．そのため，今回の実験では小さな機能候補ばかり抽出されることを防ぐために，試作ツールには暫定的に小さな機能候補を絞りこむ機構を実装して対処した．また，当初の予想に比べ NCOCP の閾値を非常に小さな値設定しなければならなかった．この問題を解決するためには，よりスケーラビリティの高い凝集度メトリクスや，抽出したい機能の特徴を考慮した機能候補の絞り込み方法などが必要である．

5.　関 連 研 究

本節では，ソースコードからの機能抽出に関連する既存研究について述べる．

Gold らは，ソースコード中のそれぞれの文がソフトウェアの持つどのコンセプトを実現しているかを知るための手法として HB-CA（Hypothesis-Based Concept Assignment）を考案した [1]．HB-CA は，事前に作成した辞書を用いる知識ベースの手法である．辞書には，ソフトウェアの持つ様々なコンセプトとキーワード，そしてそれらの関係を記述しておく．まず，ソースコード中のそれぞれの文について，そこに含まれるキー

(注1)：http://developer.android.com/ (2010/05/06)

ワードを元に関連のあるコンセプトを辞書からリストアップする．次に，連続する文ができる限り同じコンセプトに属するように割り当てるコンセプトを決定する．割り当てのための最適化手法には様々な方法が考えられるが，ここでは教師無し学習を行うニューラルネットワークの一種である自己組織化マップ（SOM：Self-Organizing Map）が用いられている．Gold らは，HB-CA を COBOLII で記述された 100 から 1500 行の 22 の商用ソフトウェアに適用し，手作業と同様のコンセプト割り当てを高い精度で行える事を示した．

また，HB-CA をさらに発展させた手法についても研究が行われている．例えば Harman，Gold らは，コンセプトにプログラムとして実行可能な文の集合を割り当てる手法として ECS（Executable Concept Slicing）を考案した [5]．ECS では，HB-CA によってコンセプトに割り当てた文の範囲をもとに，そこに含まれる変数群の計算に必要な文の範囲をプログラムスライシングで求める．

これらの手法は協調する文を探すだけでなく，その文がどのコンセプトと対応するのかを答えることができる．しかし，解析対象のソフトウェアに関して事前知識を元に辞書を作成する必要性があり，辞書作成のコストや辞書の内容によって結果が変わることが問題がある．

丸山らは，基本ブロックスライシングによるメソッド抽出手法を考案した [2]．基本ブロックスライシングは，ソースコード中の変数についてその計算に必要な文の範囲をプログラムスライシングを用いて求め，基本ブロックをもとにメソッドとして抽出するのに適した範囲の絞り込みを行う．本手法は，特定変数の計算に必要な文の集合を様々な大きさでメソッドとして抽出可能である．しかし，メソッド抽出を対象とした手法のため，単一のメソッドであらわされるような機能しか抽出することができない．また，本手法の利用には手作業を伴い，対象ソフトウェアに関する事前知識が必要である．

6. まとめと今後の課題

本研究では，ソースコード中のコード片を機能別に分類する方法を提案した．凝集度を用いて協調するコード片の組み合わせを探すことで，事前知識無しにソースコードから機能に対応するコード範囲を特定できると考えられる．提案手法の試作ツールを用いた適用実験により，実際にソースコードから機能を発見できることを確認した．

しかし，凝集度メトリクス NCOCP のスケーラビリティに問題があり，機能候補の抽出に悪影響を与えていることが分かった．この問題を解決するため，今後はスケーラビリティに優れた凝集度メトリクスや求める機能に合致しない機能候補のフィルタリング方法について検討を進めていく．また，今回の実験は非常に小規模であったため十分な評価が行えていない．今後の実験計画として，新たな正解集合との比較や，プログラム理解への貢献度など別の観点からの評価を検討している．また，これらの他にも本手法の実装の容易さを生かした Java 以外のプログラミング言語への適用や，解析結果の可視化などを行いたいと考えている．

謝辞

本研究は一部，文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．また本研究は，日本学術振興会科学研究費補助金 研究活動スタート支援（課題番号:22800040）の助成を得た．

文　　献

[1] N. Gold and K. Bennet. Hypothesis-based concept assignment in software maintenance. IEE Proceedings - Software., Vol. 149, No. 4, pp. 103-111, 2002.

[2] 丸山 勝久. 基本ブロックスライシングを用いたメソッド抽出リファクタリング. 情報処理学会論文誌, Vol. 43, No. 3, pp. 1625-1637, 2002.

[3] S.Chidamber and C.Kemerer. A Metric Suite for Object-Oriented Design. IEEE Transactions on Software Engineering., Vol. 20, No. 6, pp. 476-493, 1994.

[4] 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.

[5] M. Harman, N. Gold, R. Hierons and D. Binkley. Code extraction algorithms which unify slicing and concept assignment. In Proc. of 9th Working Conference on Reverse Engineering(WCRE 2002), pp. 11-22, 2002.