日本ソフトウェア科学会第 28 回大会 (2011 年度) 講演論文集

プログラム理解のための凝集度に基づく機能候補抽出

吉田 則裕　木下 正喬　飯田 元

ソフトウェアの保守では，開発者がプログラム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，プログラム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．プログラム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コード片（プログラム中の一部分）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，プログラム中から機能候補（単一の機能を実現するために協調して動作しているコード片の集合）を凝集度に基づいて抽出する手法を提案する．適用実験において，凝集度の閾値を適切に設定すれば，本ツールが提示する機能候補の大部分は，手作業で検出した機能と一致することがわかった．

1　はじめに

ソフトウェアの保守作業では，ソースコードの理解に膨大な時間が費やされると指摘されている [2] [4]．特に，ソースコードの作成者と保守担当者が異なる場合は，その理解により大きな時間が必要となる．

ソースコードはシステムを正確に表しているが，開発者が理解するためには膨大かつ詳細すぎる．そのため，各ソースファイルの先頭から 1 行ずつ読んでいくことで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，一般の保守担当者は，最初から各関数の先頭から 1 行ずつ読んでいくのではなく，システムの理解から始まり，次いでモジュールや関数の理解といったように，トップダウンにソースコードを理解する．これは関数の理解においても同様であり，最初に設計文書やコメント，関数名などから概要を理解し，次いでその関数が持つ各機能が実装されている場所を特定し，最後に各機能が実装されているコード片（ソースコード上の連続した領域）を理解するといったように，トップダウンに関数を理解する．

しかし，モジュール化が適切に行われていない場合は，関数をトップダウンに理解することが困難となる [4]．また，設計書やソースコードのコメントが不足している場合にも，関数が持つ機能を関数中のコード片に対応付けることができず，同様にトップダウンに理解することが困難になる [3]．

本研究では，関数の理解支援を目的として，関数中の同一機能を実現するコード片の集合を凝集度メトリクスを用いて抽出する手法を提案する．まず，本研究では，コード片の集合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か計測するために，コード片の集合を対象とした凝集度メトリクスを定義する．そして，その凝集度メトリクスが閾値以上をとるコード片の集合を検出し，機能候補として提示する．本手法は，ソースコードのみから自動的に機能候補を抽出することができるため，テストケースが不十分であったり，実行環境がない場合でも使用することができる．

また，適用実験では，実際に使用されているソフトウェアのソースコードに提案手法を実装したツールを適用した．その結果，凝集度メトリクスの閾値を適切に設定すれば，本ツールが提示する機能候補の大部分は，手作業で機能を検出した結果と一致することがわ

A Cohesion Metric-based Approach to Extracting Functionalities for Program Comprehension.

Norihiro Yoshida, Masataka Kinoshita, Hajimu Iida, 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Nara Institute of Science and Technology.

かった.

2 凝集度メトリクス

一般に，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 [5]．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，逆に関連の無い複数の機能を実現していれば低い値になる．代表的な凝集度メトリクスとして，Chidamber らが提案したクラスの凝集性の欠如を表す LCOM（Lack Of Cohesion in Methods）が挙げられる [1].

三宅らは，メソッドの凝集度を表す COB(Cohesion Of Blocks) を考案した [6]．COB はメソッド中でアクセス可能な各変数がいくつのブロックからアクセスされるかを元に下式のとおり定義される．

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

V_j : メソッド中でアクセス可能な j 番目の変数
v : メソッド中でアクセス可能な変数の数
b : メソッド中のブロックの数
$\mu(V_j)$: V_jにアクセスするブロックの数

3 提案手法

本研究では，関数の理解支援を目的として，関数中から機能を実現するコード片の集合を抽出する手法を提案する．本研究では，ソースコードと機能を以下のように定義する．

- 1 つのソフトウェアのソースコード S は，コード片の集合 $CF = \{cf_0, cf_1, \ldots, cf_m\}$ に分割される．
- ソースコード S は機能の集合 $F = \{f_0, f_1, \ldots, f_n\}$ を実現している．
- F 中の各機能は，CF 中に含まれる 1 つ以上のコード片が協調することによって実現される．ただし，CF に含まれる各コード片は，F 中の複数の機能に属することができる．

これら定義を用いると，本研究の目的は，$f_i \in F$ を実現する $Z_i \in CF$（ただし，$0 \leq i \leq n$，$\bigcup_{i=0}^{n} Z_i = CF$）を求める手法を提案することである．

本研究では，コード片集合に対する凝集度メトリクスを定義し，これを用いることで 1 つの機能を実現するために協調して動作するコード片集合を特定し，機能候補として提示する．以降 3.1 節においてコード片集合に対する凝集度メトリクスを定義し，3.2 節において凝集度メトリクスを用いて機能候補を抽出する手法を提案する．

3.1 コード片集合を対象とした凝集度メトリクス

1 つの機能を協調して実現するコード片の集合を求めるため，コード片の集合を対象とした凝集度メトリクスを提案する．

まず，メソッドを対象とした凝集度メトリクスの COB を拡張し，コード片の集合を対象とした凝集度メトリクス COCP を定義する．COB がメソッド内の各変数がいくつのブロックからアクセスされるかを元に定義されたのと同様に，COCP はコード片集合内の各変数がいくつのコード片からアクセスされるかを元に定義される．COCP の具体的な定義は次の式で表わされる．

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

V_j : コード片集合内でアクセス可能な j 番目の変数
v : コード片集合内でアクセス可能な変数の数
p : コード片の数
$\mu(V_j)$: V_jにアクセスするブロックの数

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．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．このように，COCP はコード片数の影響を受けやすい．そこで，コード片数について正規化を行った NCOCP_2 を定義する．NCOCP_2 は $COCP$ とコード変数 p を用いて次の式で定義される．

$$NCOCP_2 = \frac{p - 1/COCP}{p - 1} \qquad (0 \leq NCOCP_2 \leq 1)$$

NCOCP_2 の値は，コード片の集合が特定の機能を実現するためにどの程度協調しているかを表す．図

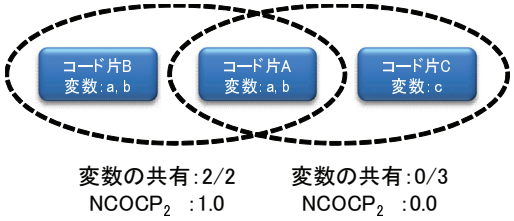


図 1　$NCOCP_2$ の算出例

1 は $NCOCP_2$ 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 $NCOCP_2$ は高い値になる．逆に，コード片 A とコード片 C の間では $NCOCP_2$ は低い値になる．

3.2　凝集度に基づく機能候補の抽出手法

3.2.1　ステップ 1：構文解析

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特殊な構文木を作成する．次に，具体的な処理手順を示す．

(1) 構文木の初期化　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) ブロックの探索　現在の処理範囲を先頭から走査し，ブロックを探す．

(3-a) ブロック発見時　(2) でブロックが見つかった場合，ソースコードを「処理範囲の先頭からブロックの先頭まで」,「ブロックの先頭からブロックの最後まで」,「ブロックの最後から処理範囲の最後まで」の 3 つに分割する．ブロック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，見つかったブロック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，ブロック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) 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．

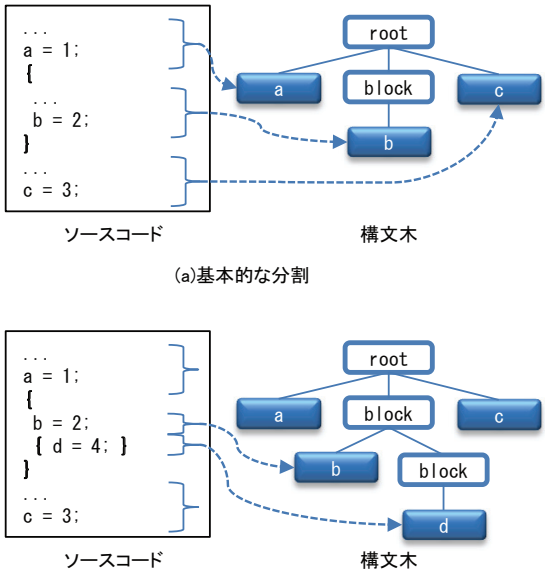
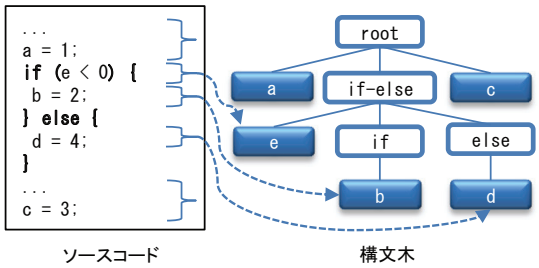
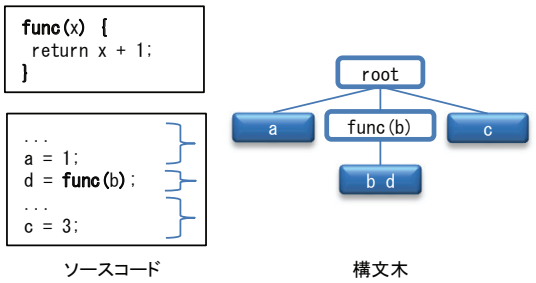


図 2　構文木の作成例 (1)

図 2，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 2 中の (a) は単純な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．図 3 中 (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．



(c)if-else文による分割



(d)関数呼び出しによる分割

図 3　構文木の作成例（2）

3.2.2　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，3.1 節で定義した凝集度メトリクス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凝集度メトリクスの値を計算する．ノードと，そのノードに属する全てのノードについてメトリクス値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，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ために行う．

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．

3.2.3　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領域がソースコード中の離れた位置に分散してい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を行う．

4　適用実験

提案手法が提示した機能候補が，開発者が手作業で検出した機能を一致するかどうか確認した．適用対象には，タンパク質の構造情報の解析を行うソフトウェア Chem3D3 を選んだ．Chem3D3 は C#で開発されており，3641 行，70 クラス，600 メソッドからなる．Chem3D3 の全ソースコードを担当した開発者に手作業で機能候補の正解集合を構築してもらったところ，全ソースコードを 80 機能に分類した．なお，この分類では，複数の機能に分類されるコード片が存在した．

提案手法が提示した機能候補との一致率は，提案手法が提示する機能候補，開発者が特定した機能候補に含まれる文字列から空白とタブ文字を除去したものを A，B とすると，以下のように表される．

$$ratio(A,B) = \frac{2 \times |A \cap B|}{|A| + |B|}$$

ただし，$A \cap B$ は，文字列 A，B の共通部分を表す．

提案手法を実装したツールを Chem3D3 に適用し，提示された機能候補と開発者が特定した機能の一致率を計算したところ，表 1 の結果が得られた．

$NCOCP_2$ の閾値を 0.50 に設定したときに，一致率の平均値，最大値，最小値の全てが最も高くなった．また，$NCOCP_2$ の閾値を 0.50 に設定したときに機能候補が 51 個抽出され，開発者が検出した 80

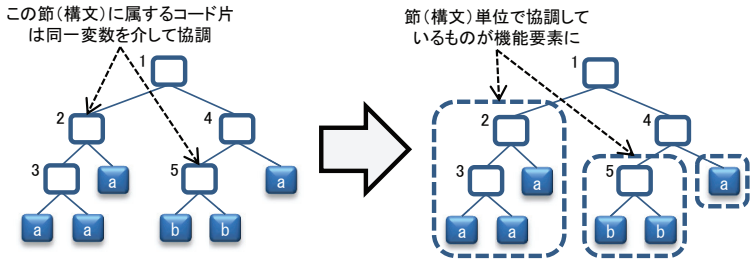


図 4　機能要素の抽出例

表 1　適用実験の結果

$NCOCP_2$ の閾値	機能候補数	一致率（平均）	一致率（最大）	一致率（最小）
0.25	24	0.55	0.84	0.32
0.50	51	0.81	1.00	0.61
0.75	215	0.32	0.90	0.05

機能に最も近かった．このことから，Chem3D3 を対象とする場合，閾値 0.50 に設定すれば機能候補を高い精度で提示することができ，有効なソースコード理解支援が行えることが考えられる．

図 5 に，$NCOCP_2$ の閾値を 0.50 に設定したときの機能候補の例を示す．この例には 3 種類の機能候補が含まれており，開発者の検出結果と完全に一致していた．タンパク質データの読込機能は，3 つのコード片に跨っているが，1 つの機能候補として抽出することができた．このように複数のコード片に跨っていても 1 つの機能候補として抽出できる理由は，単に 1 つのコード片を機能候補として提示するのではなく，3.3.4 節で説明したように単一のコード片からなる機能要素を組み合わせて機能候補として提示しているからである．

また，蛋白質構造の同定機能は 1 行のメソッド呼び出し文で 1 つの機能候補になっている．提案手法は，3.3.2 節で述べたようにインライン展開を行うため，この例のメソッド呼び出しはインライン展開されたあと，機能候補の検出が行われる．提案手法がインライン展開を行わなかったから，蛋白質構造の同定機能に含まれるメソッド呼び出し文は識別子 protein を他の文と共有しているため，他の機能候補に含まれた可能性がある．

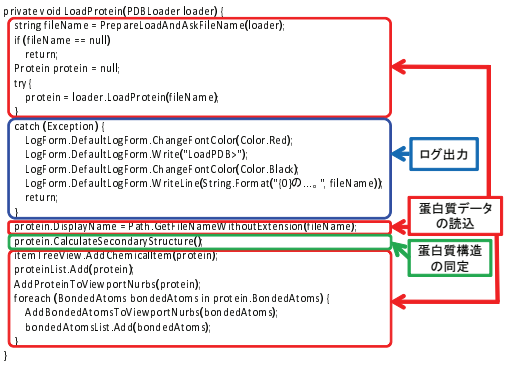


図 5　機能候補の例

5　まとめと今後の課題

本研究では，関数の理解支援を目的として，関数中の同一機能を実現するコード片の集合を凝集度メトリクスを用いて抽出する手法を提案した．適用実験の結果，$NCOCP_2$ の閾値を適切に設定すれば，本手法が提示する機能候補の大部分は，開発者が手作業で機能を検出した結果と一致することが確認できた．今後，他のソースコードに適用することにより，言語やドメインに共通する $NCOCP_2$ の決定方法を確率する必要がある．

謝辞 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.

参 考 文 献

[1] Chidamber, S. and Kemerer, C.: A Metric Suite for Object-Oriented Design, *IEEE Trans. Softw. Eng.*, Vol. 20, No. 6(1994), pp. 476–493.

[2] Deimel, Jr., L. E.: The uses of program reading, *SIGCSE Bull.*, Vol. 17(1985), pp. 5–14.

[3] Eisenbarth, T., Koschke, R., and Simon, D.: Locating Features in Source Code, *IEEE Trans. Softw. Eng.*, Vol. 29(2003), pp. 210–224.

[4] Raymond, D. R.: Reading source code, *CASCON '91*, 1991, pp. 3–16.

[5] Stevens, W. P., Myers, G. J., and Constantine, L. L.: Structured design, *IBM Systems Journal*, Vol. 13, No. 2(1974), pp. 115–139.

[6] 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(2009), pp. 1071–1073.