NAIST-IS-MT0951038

修士論文

プログラム理解のための凝集度に基づくコード片の分類手法

木下 正喬

2011 年 2 月 3 日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

木下 正喬

審査委員：

飯田 元 教授 （主指導教員）
松本 健一 教授 （副指導教員）
安本 慶一 教授 （副指導教員）
吉田 則裕 助教

プログラム理解のための凝集度に基づくコード片の分類手法*

木下 正喬

内容梗概

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．ソースコードを理解するための手順の中でも，機能とそれを実現するソースコード領域の対応付けは高コストである．これまでに，この操作を支援可能な様々な手法が提案されてきた．しかし，これらの手法の多くはソースコードについての事前知識や手動の操作を必要とし，未知のソースコードに適用することは難しい．この問題を解決するため，本研究では “凝集度” を用いて機能と対応するソースコード領域を特定する手法を提案する．凝集度は，クラスやメソッドなどのモジュールを評価する指標の一つで，モジュール内のデータと処理がどの程度協調しているかを表す．例えば，モジュールが全体で一つの機能を実現していれば凝集度は高い値を，関連の無い複数の機能を実現してれば凝集度は低い値をとる．提案手法では，ソースコードから凝集度が高くなるような領域を自動探索により求める．凝集度の性質から，これらの領域は機能と一対一で対応している可能性が高いと考えられる．提案手法の適用実験を行ったところ，ソースコードの作成者が手作業で定義した機能とソースコード領域の対応関係と近いものを特定することができた．

キーワード

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT0951038, 2011 年 2月 3日.

プログラム理解，凝集度，ソフトウェア保守，ソースコード解析，ソフトウェア開発支援ツール

A Cohesion Metric-based Approach to Categorizing Code Portions for Program Comprehension*

Masataka Kinoshita

Abstract

Efficient methods are required to support source code comprehension because they consume much time and cost to understand source code during software maintenance and enhancement. One of expensive works of source code comprehension is assigning a functionary of software to code portions in source code. So far, several methods are proposed for supporting that work. However, since those methods require knowledge of subject software and perform manual initializations, it is difficult for a new member of a software development project to use those methods. In this thesis, we propose an automatic method based on cohesion metric to solve the difficulty for a developer who is a new member of a software development to assign a functionary of software to code portions in source code. Cohesion metric is an indicator of the modularity for a module (e.g., a class or a method) in object-oriented languages, and represents the degree of collaborations between processes and data within a modu le. For example, the metric increases when entire module implements functionality. On the other hand, it decreases when a module implements functionaries that are not closely related. Proposed method determines sets of code portions with high values of cohesion metric automatically. According to the nature of cohesion metric, there is a considerable possibility that each of those code portions corre-

*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0951038, February 3, 2011.

spond one-to-one with a functionality of software. Our experiment shows that proposed method is capable to approximately derive manual correspondence by developer of the subject source code.

Keywords:

Program Comprehension, Cohesion Metric, Software Maintenance, Source Code Analysis, Tool Support for Software Development

目次

1. はじめに **1**

2. 関連研究 **3**

2.1. 機能とソースコード領域の対応付け 3

2.2. 凝集度メトリクス 5

3. 提案手法 **8**

3.1. 概要 8

3.2. 任意のソースコード領域を対象とした凝集度メトリクス 8

3.3. ソースコード領域の機能別分類 11

3.3.1. 概要 11

3.3.2. ステップ１：構文解析 12

3.3.3. ステップ２：機能要素の抽出 15

3.3.4. ステップ３：機能候補の抽出 16

4. 制作ツール **17**

4.1. 機能 17

4.2. 実装 18

5. 適用実験 **21**

5.1. 概要 21

5.2. 手順 21

5.3. 結果と考察 22

6. おわりに **25**

謝辞 **27**

図目次

1 LCOM, LCOM* 算出例 7
2 COCP と $NCOCP_2$ の関係 10
3 $NCOCP_1$ と $NCOCP_2$ の関係 10
4 提案メトリクス算出例 11
5 手法の概要 12
6 構文木の作成例（1） 14
7 構文木の作成例（2） 15
8 機能要素の抽出例 16
9 制作ツール 17
10 LoadProtein メソッドの機能別分類 25

表目次

1 プログラム構文の例 19
2 アクセス変数の例 19
3 抽出結果 22
4 機能候補と正解集合の重複率 23

関連発表論文

研究会・シンポジウム

1. 木下 正喬, 吉田 則裕, 飯田 元, “ソースコード解析に基づいた機能群の推定とその分類手法の提案,” 平成 22 年度 情報処理学会関西支部 支部大会 講演論文集, 2010.
2. 木下 正喬, 吉田 則裕, 飯田 元, “凝集度によるコード片の機能別分類手法,” 電子情報通信学会技術報告, vol. 110, no. 336, pp. 79-84, 2010.

1. はじめに

ソフトウェア開発プロジェクトの多くは既存ソフトウェアの保守や拡張を目的としている [10]．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なため，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 [2][5]．

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ソースコード中の領域を特定する必要があるが，大規模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，機能とソースコード領域の対応づけを自動化，あるいは支援する手法が提案されてきた．Gold らは，ソースコード中の領域を抽象的な機能単位であるコンセプト別に分類する手法として HB-CA（Hypothesis-Based Concept Assignment）を考案した [2]．丸山はプログラム依存グラフと基本ブロックをもとに，特定変数の算出に必要なソースコード領域をメソッドとして抽出する手法を考案した [9]．兼光らはプログラム依存グラフのノード間の接続パターンをもとに，メソッド抽出に適したノードを近くに配置したプログラム依存グラフを表示する可視化手法を考案した [12]．

いずれの手法も，機能と対応するソースコード領域を抽出可能だが，分析のためにソースコードについての事前知識や手動の操作を必要とする．そのため，完全に未知のソースコードを対象にした分析は難しい．

本研究では，事前知識を用いず，自動的にソースコード領域を機能別に分類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と対応する領域を抽出の目標とする．

- その内部で強く協調しあい，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．一つ目の特徴は，ソースコード

を文よりも粒度の大きいコード片の単位で解析することである．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からである．そのため，文単位に比べ効率良く解析が可能である．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 [7][6][3][11].

提案手法では，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，それらのコード片を凝集度が高くなる組み合わせごとに分類する．こうして得られたコード片の組は，凝集度の性質から機能と一対一で対応している可能性が高いと考えられる．

提案手法の有効性を検証するため，2 種類のソフトウェアで適用実験を行った．実験では，対象ソフトウェアのソースコードを十分に理解している開発者がそのソースコード領域を手動で機能別に分類した結果と，提案手法によって自動で分類した結果を比較した．その結果，提案手法により手動での分類した領域と同様の領域を特定可能なことが確認された．

以降，2 章では機能とソースコード領域の対応づけに関する既存研究とその問題点，および本研究で利用する凝集度とそのメトリクスについて述べる．3 章では，本研究で提案するソースコード領域の機能別分類手法について説明する．4 章では，提案手法の実装例として作成した分析・可視化ツールについて説明する．5 章では，提案手法を実際のソフトウェアに適用し，手動分析との比較を行った結果と考察を述べる．最後に，6 章で本研究のまとめを述べる．

2. 関連研究

本章では，本研究に関連する既存の研究について述べる．2.1 節では，本研究と同じく機能と対応するソースコード領域の特定を行う研究について，2.2 節では本研究で利用する凝集度の概念と凝集度を測るメトリクスの研究について述べる．

2.1. 機能とソースコード領域の対応付け

機能とソースコード領域の対応付けを行う手法には，主にプログラム理解支援を目的とした手法と，メソッド抽出を目的とした手法がある．

プログラム理解支援を目的とした手法

Biggerstaf らは，ソースコードを理解するための重要なプロセスとして，コンセプト割り当てを定義した [4]．コンセプト割り当てでは，ソースコード中の領域とコンセプトの対応づけを行う．コンセプトとは，開発者にとって興味があり，機能的なまとまりのある抽象的な処理の事を指す．ソースコードを読む目的に応じて，“ファイル入出力”といった汎用的なものから，“特定数値の計算”など対象ソフトウェアに強く依存したものまで様々なものがコンセプトとなる．このプロセスにより，開発者は興味のあるソースコード領域を把握したり，ソースコード全体を俯瞰したりすることができるようになる．

Gold らは，コンセプト割り当てを自動的に実行する手法として，HB-CA（Hypothesis-Based Concept Assignment）を考案した [2]．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ワードを元に関連するコンセプトを辞書からリストアップする．次に，リストアップしたコンセプトの中から，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ものを決定する．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 らは，コンセプトにプログラムとして実行可能なソースコード領域を割り当てる手法として ECS（Executable Concept Slicing）を提案した [5]．ECS では，HB-CA によってコンセプトに割り当てた領域をもとに，そこに含まれる変数群の計算に必要な領域を依存性解析によって求める．

これらの手法は，先にどのようなコンセプトがあるかを辞書に定義しておくトップダウン分析である．そのため，あらかじめ辞書を用意しておけば，協調するソースコード領域を探し出すだけでなく，その領域がどのコンセプトに対応するのかを求めることができる．反面，辞書に定義されたコンセプトと対応した領域しか求められない．また，辞書作成のためにはソフトウェアに関する事前知識が必要で，作成にかかるコストも大きいという問題がある．そのため，これらの手法を未知のソースコードを理解するために用いることは難しい．

メソッド抽出を目的とした手法

一般に, オブジェクト指向プログラミングでは, メソッドやクラスなどのモジュールは一つの機能と対応していることが望ましい [7]．しかし，実際のソフトウェア開発では，大規模な一つのモジュールが関連の無い複数の機能を実現していたり，複数のモジュールに一つの機能に関する処理が分散しているようなソースコードがたびたび作成される．メソッド抽出は，このようなソースコードから一つの機能に対応するソースコード領域を，新たなメソッドとして取り出すリファクタリング操作の一種である [1].

これまでに，メソッド抽出を支援する手法が多数考案されている．これらの手法はプログラム理解支援手法とは目的が異なるが，機能と対応する領域を特定するという点で共通性がある．

丸山は，基本ブロックスライシングによるメソッド抽出手法を提案した [9]．この手法は，一つの機能からは一つの出力データが得られると仮定する．プログラム中ではデータは変数に格納されるため，着目した変数について算出に必要な文の集合をプログラム依存グラフを用いて求め，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．これにより，ユーザは着目すべき変数を指定するだけで，メソッド候補とそのメソッドを用いて書き換えたソースコードを得ることができる．

兼光らは，メソッド抽出に適したノードを近くに配置したプログラム依存グラフを表示する可視化手法を考案した [12]．この手法では，メソッド抽出に適したソースコード領域を発見するために，プログラム依存グラフのノード間の接続パターンに着目する．例えば，良く参照される変数は重要な値であり，それを参照する文の集合はつながりが強いと考えられる．このような 3 種類の接続パターンをもとに，つながりの強い文に対応するノードの組が近くなるようにノード間の距離を設定したプログラム依存性グラフを作成して表示する．可視化したグラフには，メソッド抽出に適したノードの組が集まって表示されるため，ユーザは容易にメソッド抽出の範囲を決定可能である．

しかし，丸山の手法は分析のために着目すべき変数をユーザが選択する必要がある．また，兼光らの手法は可視化した結果をヒントに，機能と対応する領域の決定はユーザが行う．このように，いずれの手法も手動の操作を必要とするため，HBCA などと同様に未知のソースコードを理解するために用いることは難しい．

2.2. 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，逆に関連の無い複数の機能を実現していれば低い値になる．一般に，凝集度の低いモジュールは再利用性が低く，ソースコードの理解を難くし，ソフトウェアの保守性を低下させる．そのため，凝集度はモジュールの品質を評価する尺度として用いられる [7]．

これまでに，凝集度を測るメトリクスが多数考案されてきた．Chidamber と Kemerer は，オブジェクト指向設計の複雑度に関する 6 つの CK メトリクスの一つとして，クラスの凝集性の欠如を表す LCOM（Lack Of Cohesion in Methods）を提案した [6]．クラスの凝集度が高い，つまりクラスに互いに関連の強いメンバ変数とメソッドが集まっていれば，そのクラスのそれぞれのメンバ変数にアクセスするメソッドの数は多くなると考えられる．そこで，LCOM はクラス中のメソッ

ドのペアについて，同じメンバ変数へのアクセスを行わないペアの数を P，行うペアの数を Q として次の式で定義される．

$$LCOM = \begin{cases} P - Q & (P > Q) \\ 0 & (P \leq Q) \end{cases}$$

P ：同じメンバ変数にアクセスしないメソッドのペアの数
Q ：一回以上，同じメンバ変数にアクセスするメソッドのペアの数

LCOM は凝集度の欠如を表すため，低い値を持つことが望ましい．しかし，LCOM はメンバ変数やメソッドの数によって影響を受けやすく，クラスの凝集度を正確に評価できないことがある．例えば，図 1 の (a) と (b) では明らかに (b) の方が凝集度が高いように見えるが，LCOM の値はどちらも 8 になる．

Henderson は，LCOM の問題点を解決するため，クラスのメンバ変数とメソッドの数を考慮して LCOM を改良した LCOM* を提案した [3]．LCOM* は次の式で定義される．

$$LCOM^* = \frac{\frac{1}{a}\sum_j^a \mu(A_j) - m}{1 - m} \qquad (0 \leq LCOM^* \leq 1)$$

A_j ：クラス中の j 番目のメンバ変数
a ：クラス中のメンバ変数の数
m ：クラス中のメソッド数
$\mu(A_j)$ ：A_j にアクセスするメソッドの数

LCOM と同様に，LCOM* も低い値を持つことが望ましい．図 1 の例では，LCOM* は LCOM と異なり (a) の凝集度がより低いことを示している．

三宅らは，メソッドの凝集度を表す COB(Cohesion Of Blocks) を考案した [11]．LCOM や LCOM* がクラス中の各メンバ変数がどの程度多くのメソッドからアクセスされるかをもとに定義されているのに対し，COB はメソッド中でアクセス可能な各変数がいくつのコードブロックからアクセスされるかを元に定義される．具体的な定義は次の式で表される．

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

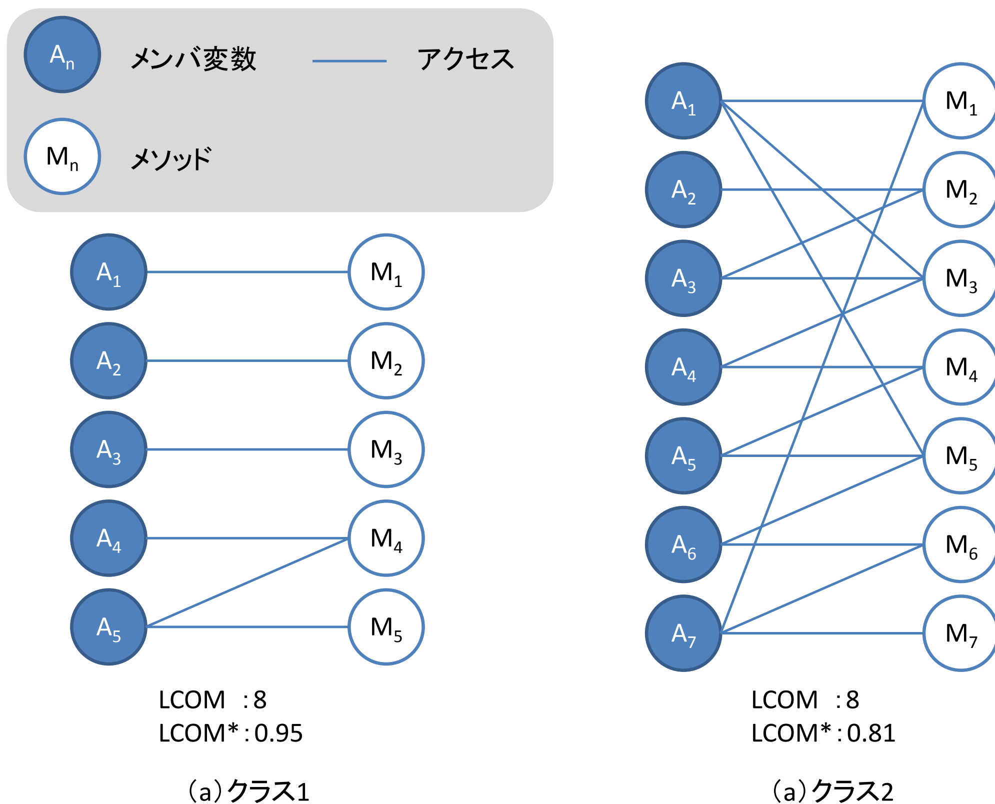


図 1 LCOM, LCOM* 算出例

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 : メソッド中のコードブロックの数
$\mu(V_j)$: V_j にアクセスするコードブロックの数

三宅らは，apache-ant-1.7.0 の約 20 万行のソースコード中で定義された 9,937 のメソッドについて COB を計算し，メソッド抽出等により分割すべきメソッドを COB の値が低いメソッドとして検知可能なことを確かめた．

3. 提案手法

3.1. 概要

提案手法は，機能と対応するソースコード中の領域を凝集度を用いて特定することを目的とする．本手法の基本的なアイデアは次の通りである．

1. ソースコードを構文に基づいてコード片に分割する
2. コード片の様々な組み合わせについて，その凝集度を評価する
3. 凝集度が高くなるコード片の組合せを機能候補として出力する

凝集度の性質から，こうして得られた機能候補はその内部で協調しており，単一の機能と対応すると考えられる．

以下では，本手法での利用するために新しく定義した凝集度メトリクスと，そのメトリクスを用いてソースコードから機能候補を抽出する手法について説明する．

3.2. 任意のソースコード領域を対象とした凝集度メトリクス

提案手法では，単一の機能に対応するソースコード領域を発見する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 などの既存メトリクスと異なり，任意のソースコード領域に対して凝集度を求める必要がある．提案手法では，コードブロックなどのプログラム構文により区切られたコード片を解析の単位としている．そのため，以降ではソースコード中の任意の領域をコード片の集合で表現する．

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，ソースコード中の任意のコード片集合を対象とした凝集度メトリクス COCP（Cohesion of Code Portions）を定義する．COB がメソッド内の各変数がいくつのコード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 にアクセスするコードブロックの数

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 はコード片の数に対するスケーラビリティが非常に悪い．

そこで, 異なる方法でコード片数について正規化を行った NCOCP_1 と NCOCP_2 を定義する．

NCOCP_1 は，コード片の数に関わらず値の範囲が 0.0 から 1.0 になるように，COCP を補正したメトリクスである．例えば，対象の領域のコード片数が 2 つの場合には COCP は 0.5 から 1 の値をとりうるので，その範囲を 0 から 1 に線形に拡大する．これは LCOM* がメンバ変数やメソッドの数によって行った正規化と同様である．NCOCP_1 は，COCP を用いて次の式で定義される．

$$NCOCP_1 = \frac{p \cdot COCP - 1}{p - 1} \qquad (0 \leq NCOCP_1 \leq 1)$$

p　：コード片の数

NCOCP_2 は，コード片の数ごとに COCP の分布を求め，対象コード片集合の COCP が分布の下位何 % に属するかをもとに定義される．図 2 に COCP の分布を示す．グラフの横軸は様々なコード片集合を COCP 順に並べた時に対象のコード片集合が下位何 % に属しているかを，縦軸は対象のコード片集合の COCP の値を示している．例えば，コード片集合のコード片の数が 2 の時，COCP の値が 0.5 であればそのコード片集合はコード片の数が 2 のコード片集合の中で最も COCP の値が低い．この時，NCOCP_2 の値は 0 になる．同様に，コード片の数が 4 のとき，COCP の値が 0.5 であればそのコード片集合はコード片の数が 4 のコー

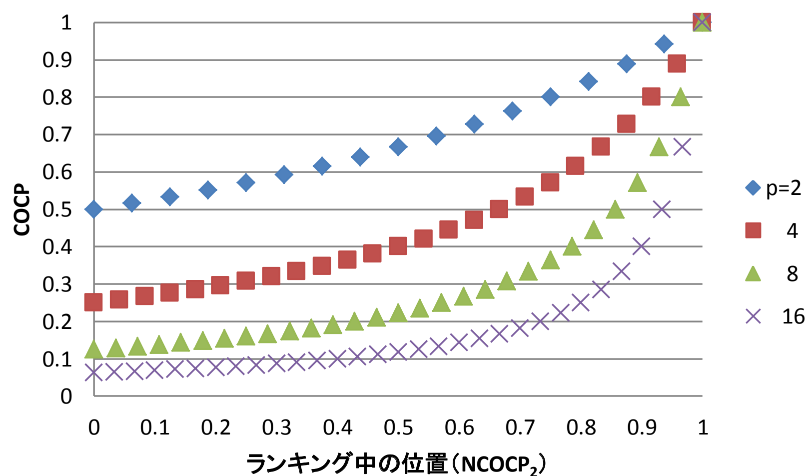


図 2 COCP と $NCOCP_2$ の関係

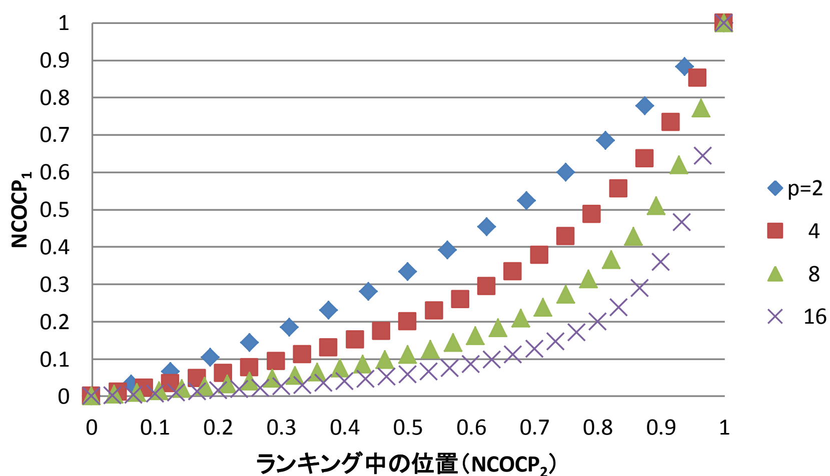


図 3 $NCOCP_1$ と $NCOCP_2$ の関係

ド片集合の中で下位 67% に属している．この時，$NCOCP_2$ の値は 0.67 になる．$NCOCP_2$ は COCP を用いて次の式で定義される．

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

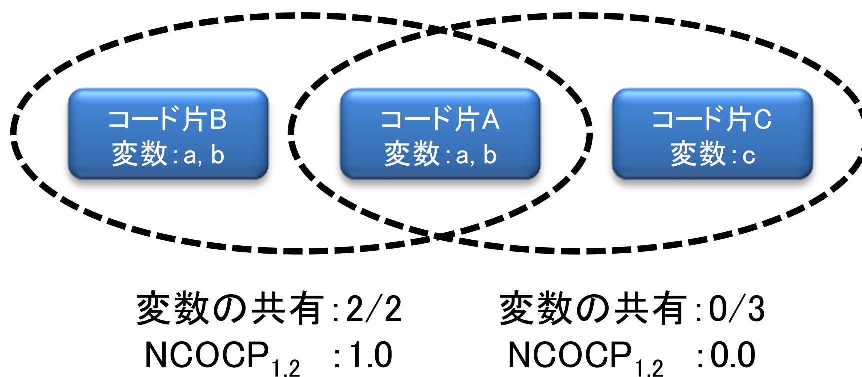


図 4 提案メトリクス算出例

p : コード片の数

図 3 に NCOCP_1 と NCOCP_2 の関係を示す．NCOCP_1 は値の範囲は 0 から 1 とるものの，NCOCP_2 と比較するとコード片の数が増えるほど値が低くなる傾向がある．このため，多くのコード片を含む，すなわち広い範囲で実現される機能を抽出するには，NCOCP_2 がより適していると考えられる．

図 4 は提案メトリクスの算出例である．コード片 A とコード片 B は共有する変数が多いため，これらの間でメトリクスは高い値になる．逆に，コード片 A とコード片 C の間ではメトリクスは低い値になる．提案手法では，単一の機能と対応するソースコード領域を特定するために，これらのメトリクスのうちいずれかを用いる．凝集度の性質から，単一の機能と対応する領域に対してこれらのメトリクスを計測すると高い値が得られると考えられる．

3.3. ソースコード領域の機能別分類

3.3.1. 概要

図 5 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，単一の機能に対応すると考えられるソースコード領域のことを言う．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．

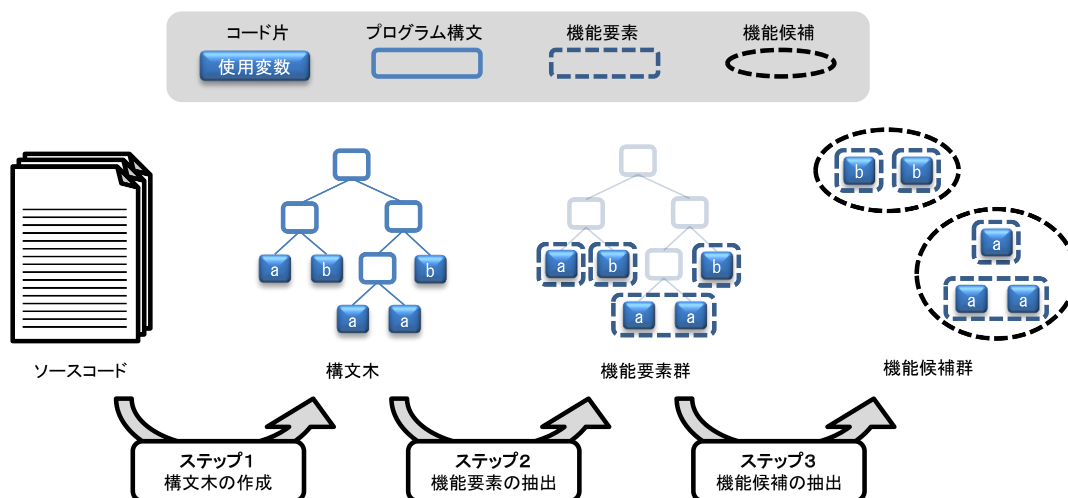


図 5　手法の概要

ステップ 1：構文解析　ソースコードの構文を解析し，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する．

以降，3.3.2 節から 3.3.4 節で各ステップの動作について述べる．

3.3.2.　ステップ 1：構文解析

本ステップで行う構文解析は，C 言語や C#言語などの一般的な手続き型言語とオブジェクト指向言語を対象としている．本ステップで構文を解析しておくことにより，ステップ 2 以降ではプログラミング言語に依存しない共通の処理を用いることができる．以下では，様々なプログラミング言語が持つ代表的な構文を例に，疑似コードを用いて解析の方法を説明する．

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) 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．

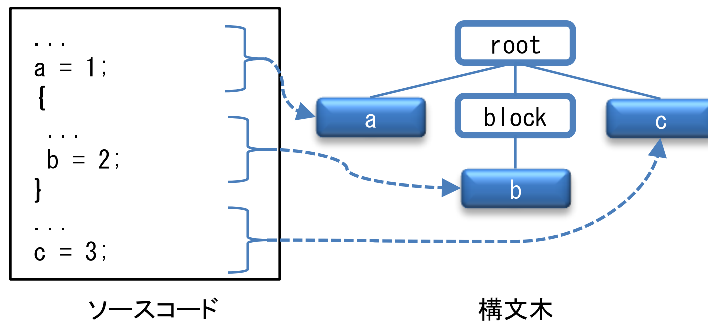
(2) プログラム構文の探索　現在の処理範囲を先頭から走査し，コードブロックなどのプログラム構文を探す．

(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．

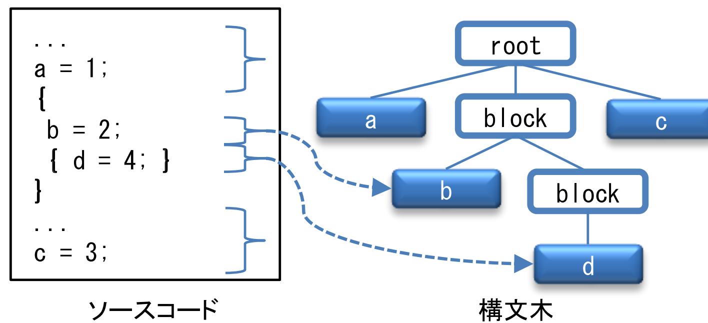
(3-b) コード片の取得（処理範囲の走査終了時）　(2) でプログラム構文が見つからずに処理範囲の走査を終了した場合，その範囲のコード片を構文木のカレントノードに葉として追加する．また，その範囲でアクセスされている変数の情報をコード片に付与する．

(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 6，図 7 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 6 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



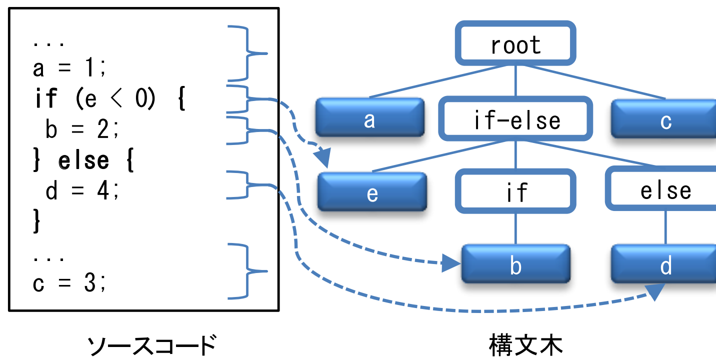
(a)基本的な分割



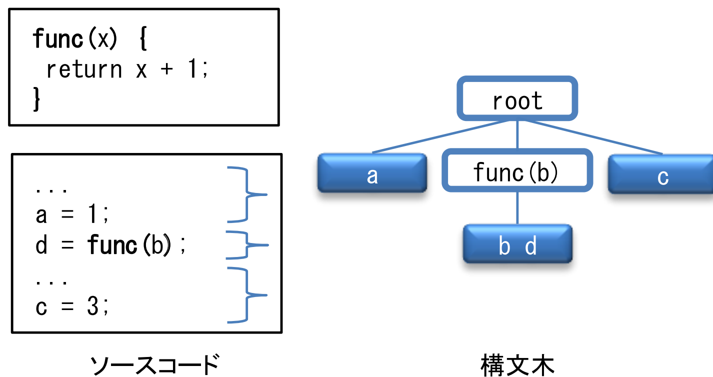
(b)ネスト構造の分割

図 6　構文木の作成例（1）

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 7 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．



(c)if-else文による分割



(d)関数呼び出しによる分割

図 7　構文木の作成例（2）

3.3.3.　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，3.2 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

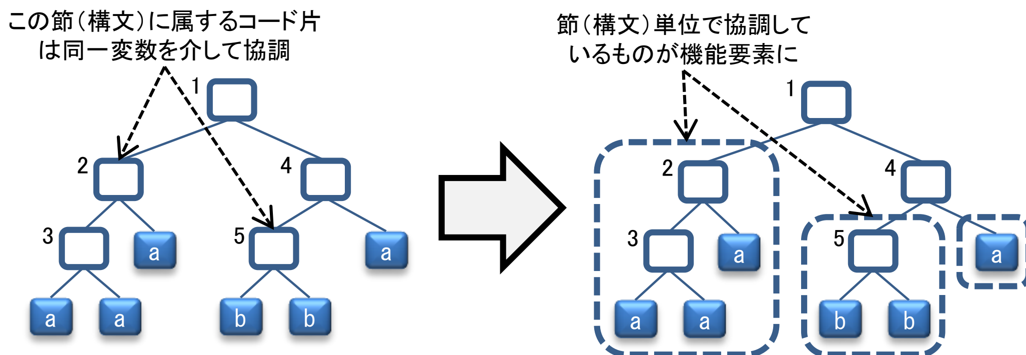


図 8　機能要素の抽出例

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，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 8 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．

3.3.4.　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，凝集度やコード行数によってランキングして開発者に提示する．

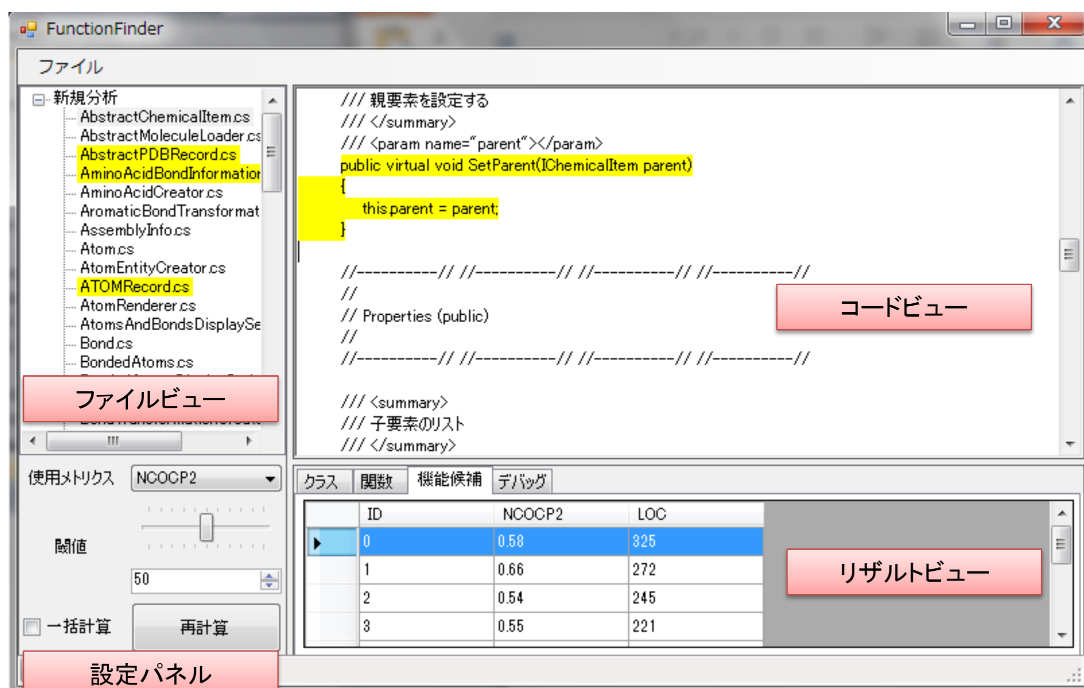


図 9　制作ツール

4.　制作ツール

4.1.　機能

図 9 に制作ツールの画面を示す．この画面は，以下のビューから構成される．

- **ファイルビュー**　解析対象のソースコードを一覧表示する
- **設定パネル**　提案手法が用いる凝集度メトリクスの種類と閾値を設定する
- **コードビュー**　ファイルビューで選択されたソースコードの内容を表示する
- **リザルトビュー**　提案手法で得られた機能候補の一覧や，ソースコードの構文解析結果を表示する

本ツールの使用は，次の手順で行う．

(1) ソースコードを開く　開いたソースコードはファイルビューに一覧表示される．ソースコードを開くと同時に構文解析が行われる．

(2) パラメータを設定する　設定パネルから，機能候補抽出に用いる凝集度メトリクスの種類と閾値を設定する．メトリクスには COCP, $NCOCP_1$, $NCOCP_2$ が選択できる．また，閾値は 0.01〜1.00 まで 100 段階で設定できる．

(3-a) 計算を開始する　設定パネルの再計算ボタンを押すと設定パラメータに基づいて解析を開始する．得られた機能候補一覧は，リザルトビューに表示される．

(3-b) 一括計算を行う　一括計算をオンにして計算ボタンを押すと，凝集度の全ての閾値について解析を行う．以降，閾値の値を再設定すると即座にその結果がリザルトビューに反映される．

(4) 結果を表示する　リザルトビューから機能候補を選択すると，その機能候補と対応するソースコード中の領域が強調表示される．

4.2.　実装

構文解析や機能候補の抽出を次のように実装した．

構文解析

提案手法のステップ 1 では，ソースコードを構文解析して構文木を作成する．制作ツールでは，解析対象のプログラミング言語を C#とした．表 1 に，構文木の節となるプログラム構文の例を示す．解析するコード例の... と表記した部分の内，実行可能な部位がその構文に対応する節の子要素となる．また，これらの構文によって区切られたコード片が，構文木の葉となる．

次に，凝集度計算のために構文木の葉に対応するコード片ごとにアクセスする変数やオブジェクトを取得する．表 2 にアクセスする変数やオブジェクトの取得例を示す．

制作ツールでは，解析が困難な次の構文について解析の制限がある．まず，オーバーライドされたメソッドの呼び出しでは，静的解析で呼び出し先を特定することが困難なため，呼び出し可能性のある全てのメソッドを構文木のメソッド呼び出し

節の子要素としている．同様に呼び出し先の特定が困難なデリゲートを用いた構文は，デリゲートの初期化時に実行文が与えられた場合以外には呼び出し先の解析を行っていない．また，ラムダ式やメソッドの再帰呼び出しなど複数の機能に分類すべきでないと考えられる構文は解析を行っていない．

機能候補の抽出

提案手法のステップ 3 では，機能候補を抽出するために凝集度が高くなるような機能要素の組合せを求める．制作ツールには，常に最適解を求められる全探索と，

表 1　プログラム構文の例

構文の種類	コード例	備考
クラス定義	class ExampleClass { ... }	
メソッド定義	int ExampleMethod() { ... }	
プロパティ定義	int ExampleProperty { get { ... } set { ... } }	・...部分が省略された場合はメンバ変数として扱う
デリゲート定義	ExampleDelegate ed = delegate { ... }	・初期化時に定義される場合のみ ・他はメソッドと同様
・メソッド ・プロパティ ・デリゲート の呼び出し	int a = ExampleMethod();	・対象ソースコード内でメソッドが定義されている場合のみ ・インライン展開する ・再帰呼び出しは展開しない ・オーバーライドされたメソッドは，呼び出し可能性のある全てのメソッドを子要素として展開する
if-else文	if (...) ... else ...	

swtich文	swtich (...) {
	case A: ... break;
	:
	}	

while文	while (...) ...	
for文	for (...; ...; ...) ...	
foreach文	foreach (var e in MyArray) ...	
コードブロック	{ ... }	

表 2　アクセス変数の例

アクセスの種類	コード例	アクセス変数
ローカル変数	{ b = a + 5; }	a b
メンバ変数	{ obj.m = b + 5; }	b obj obj.m
節として展開されないメソッド呼び出し	{ obj.methodB(c); }	c obj

良い結果が得られる可能性が高くソースコードサイズに対する処理時間のスケーラビリティが良い遺伝的アルゴリズムを用いた探索の 2 種類の探索法を実装した．全探索では，共通の変数を持つ組合せのみを計算の対象とすることで計算量を削減している．遺伝的アルゴリズムでは，各個体を機能要素の集合，個体の評価関数を $NCOCP_2$ などの凝集度メトリクス，世代交代のアルゴリズムを一点交叉法を用いた MGG（Minimal Generation Gap）法 [8] とした．

5．　適用実験

5.1．　概要

制作したツールを用いて，提案手法の有効性を評価するための適用実験を行った．実験は，タンパク質の構造情報をもとにその可視化と三次元形状モデルへの変換を行うソフトウェア Chem3D3 のソースコードを対象に，その開発者が手動で機能を分析した結果と，ツールを用いて自動で分析した結果を比較した．Chem3D3 のコード行数は 3641 行，クラス数は 70，メソッド数は 600 である．

5.2．　手順

実験の手順を以下に示す．

正解集合の作成

Chem3D3 のソースコードを手動で分析し，ソースコード中の領域を機能別に分類したものを正解集合とした．分析者は，当ソフトウェアの開発者であり，そのソースコードを十分に理解している．分析は次の二つの操作により行った．

- メソッド単位での分類
- 複数の機能を含むメソッドの分割

分析者はメソッドを機能ごとに分類し，もし複数の異なる機能に属する処理を持つメソッドがあれば，そのメソッドを分割してそれぞれの機能に割り当てる．このようにして，複数のメソッドにまたがる機能や適切にメソッドに分割されていない機能についても正解集合を作成する．

ツールによる分析

ツールによる分析は，凝集度メトリクスとして $NCOCP_2$ を用い，閾値を 0.25, 0.50, 0.75 の 3 種類の値を設定して行った．また，コード片の組合せ最適化には，全探索を用いた．

正解集合とツールによる分析結果の比較

ツールによる分析で得られた機能候補が，正解集合の機能とどれだけ類似しているかを，機能候補と正解集合の機能の領域がどれだけ重複しているかで評価する．重複している範囲が大きければ，その機能候補は正解集合の機能と類似していると判断する．

5.3. 結果と考察

得られた機能候補

表 3 に凝集度ごとに得られた機能要素，機能候補の数と大きさを示す．

$NCOCP_2$ の閾値が大きくなるほど機能要素，機能候補の数が大きくなっている．これは，閾値を大きくすると，コード片同士が結びつきにくくなるからである．メトリクスの閾値 0.25 では，多くのクラスが一つの機能要素になり，逆に閾値 0.75 ではクラスが一つの機能要素になるものは少なかった．今回は，3 種類の閾値のみで実験を行ったが，より細かく閾値を設定することで，任意の粒度で機能候補を抽出可能だと考えられる．

正解集合と機能候補の比較

表 4 に正解集合の機能群と得られた機能候補の領域の重複率を示す．重複率は，2 つの領域の重複した部分の文字数と，2 つの領域の平均文字数の比で表され，領域が完全に一致する場合には 1 に，1 文字も一致しない場合には 0 に，それ以外には 0 から 1 の間の値となる．

今回の実験では，閾値 0.5 の時に正解集合と最も良く一致し，正解集合の 30 個

表 3 抽出結果

$NCOCP_2$の閾値	機能要素の数	機能候補の数	機能候補の平均行数
0.25	94	24	377
0.5	328	51	177
0.75	860	215	42

の機能について平均で機能候補の 81% の範囲が重複した．また，いくつかの機能については 100% の範囲が一致した．これは，今回の正解集合の機能の粒度と，閾値 0.5 のときの機能候補の粒度が最も一致したからだと考えられる．

図 10 に，メソッド中のコード片が複数の機能に分類されたメソッドの例として，LoadProtein メソッドを示す．LoadProtein メソッドは，1 つのメソッドの中に 3

表 4 機能候補と正解集合の重複率

分類名	関連メソッド数	0.25	0.5	0.75
原子モデルの作成	11	0.52	0.76	0.22
球の作成	4	0.46	0.64	0.49
結合モデルの作成	50	0.71	1.00	0.05
円筒モデルの作成	4	0.31	0.76	0.49
厚みの無いリボン曲面の作成	26	0.66	0.70	0.13
リボン曲面の作成	53	0.76	1.00	0.06
曲線の分割	2	0.59	0.61	0.74
デッドコード1	8	0.61	0.68	0.27
デッドコード2	3	0.32	0.68	0.58
TubesEntityの作成	13	0.46	0.91	0.18
Richardson形式のリボン作成	19	0.66	0.86	0.13
円モデルの作成	2	0.53	0.79	0.72
通常のリボンの作成	1	0.41	0.74	0.89
TubeEntityの作成	35	0.54	0.92	0.11
TubeEntity2の作成	7	0.36	0.91	0.34
タンパク質の読み込み	129	0.84	0.92	0.05
ログの記述	9	0.63	0.89	0.28
分子の読み込み	23	0.66	0.70	0.15
描画オプションの設定	73	0.77	0.95	0.05
エンティティ情報の表示	42	0.52	0.84	0.07
初期状態に戻す	3	0.39	0.67	0.58
二次構造の計算	42	0.43	0.75	0.11
3Dモデルの書き出し	17	0.54	0.89	0.18
エントリポイント	3	0.55	0.65	0.57
原子と結合の表現	33	0.62	0.86	0.10
分子の表現	22	0.62	0.67	0.15
エンティティの選択	11	0.46	0.74	0.21
3Dモデルの書き出し2	2	0.43	0.84	0.72
エンティティの描画	1	0.59	0.82	0.90
タンパク質の表現	40	0.46	0.99	0.07
平均		0.55	0.81	0.32

つの機能に属するコード片を持つ．実験では，$NCOCP_2$ の閾値が 0.5 の時には，正解集合の機能の通りに正しく分割された．閾値が 0.25 の時にはメソッド全体が一つの機能候補として抽出され，逆に閾値が 0.75 時には，メソッドが 4 つに分割された．このように，凝集度メトリクスの閾値が高くなると，抽出する機能の粒度が細かくなると考えられる．

また，図中の “C. 二次構造の計算” に関する機能は 1 つのメソッドコールのみで構成されているが，呼び出されているメソッドをインライン展開して解析したことにより抽出ができた．

これらの結果から，本手法はソースコードを開発者が考えるのと同様に，機能別に分類可能なことが確認できた．しかし，今回の実験は，凝集度の閾値や分析対象のソースコード，正解集合の作成などの条件が非常に限定的なため，本手法の有効性を確認するためにはより多くの条件で実験を行う必要がある．また，本手法がプログラム理解支援に適しているかを確認するためには，本手法によりプログラム理解にかかる時間を短縮できるかなど，別の観点からの実験も必要である．

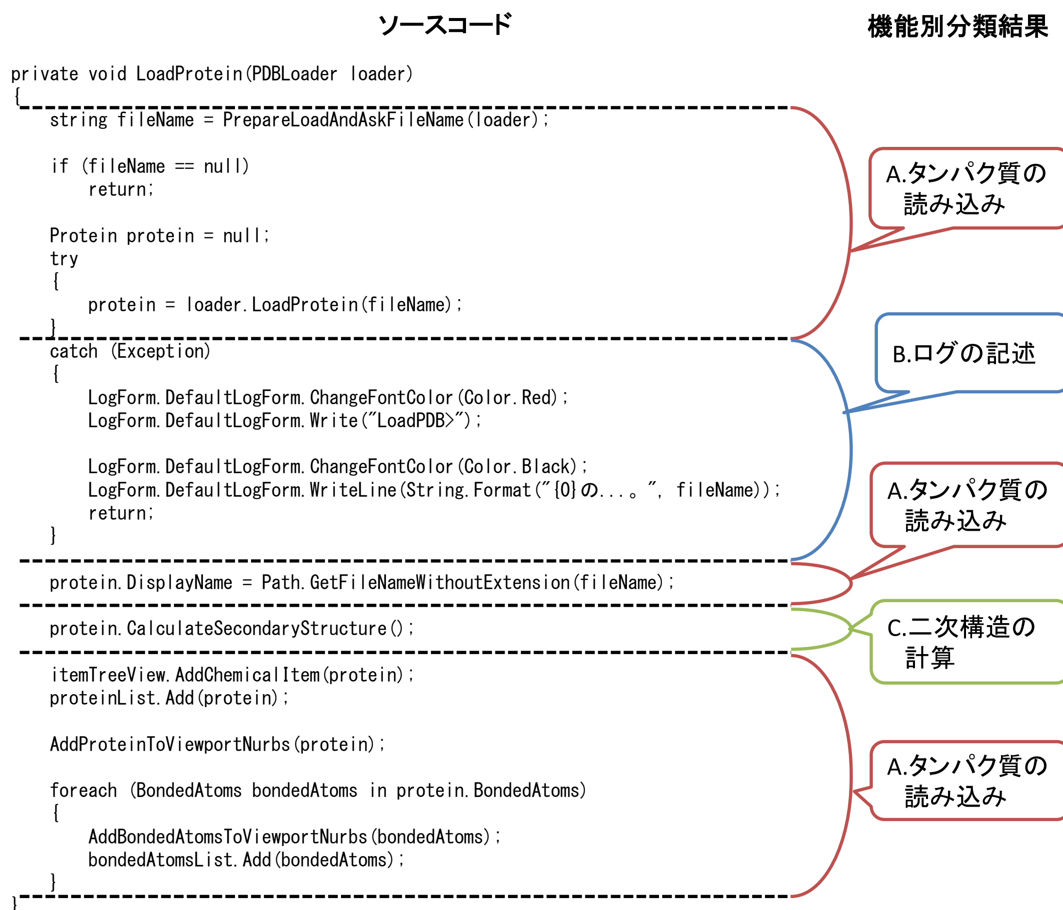


図 10 LoadProtein メソッドの機能別分類

6. おわりに

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手法を提案した．凝集度を用いて強調するコード片の組合せを探すことで，事前知識や手動の操作な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．また，本手法に向けた新しい凝集度メトリクスを提案した．提案手法のツールを実装し，適用実験を行った結果，実際にソースコードを十分に理解している分析者と同様の機能を発見できることを確認した．

実験では，凝集度メトリクスの閾値を 3 種類設定することで，抽出する機能候補の粒度を変化させることができた．より，細かく閾値を設定することで任意の粒度で機能候補を求めることができると考えられる．

今後は，凝集度メトリクスの閾値と分類結果の関係を確かめるため，凝集度メトリクスの閾値をより細かく設定しての実験を行う必要がある．また，今回は一つのソフトウェアを分析対象とし，正解集合は一人の開発者による分析の結果を元に作成した．本手法の有効性を確認するためには，より多くのソフトウェア・正解集合での実験が必要である．さらに，この実験は本手法の機能別分類の正確性を示すためのもので，プログラム理解支援手法としての有用性を確認するためのものではない．そのため，今後は本手法を用いてソースコード理解にかかる時間をどの程度短縮できるかなどの観点でも実験を行う必要がある．

謝辞

本研究を進めるにあたり，多くの方々に御指導，御協力，御支援を頂きました．ここに謝意を添えて御名前を記させて頂きます．

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飯田元教授には，本研究の全過程において熱心な御指導を賜りました．研究方針だけではなく，研究に対する姿勢，論文執筆，発表方法についても多くの御助言を頂きました．心より厚く御礼を申し上げます．

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工学講座松本健一教授には，様々な場面で本研究に対し貴重な御指導，御助言を賜りました．心より感謝申し上げます．

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基礎学講座安本慶一准教授には，様々な場面で本研究に対し貴重な御指導，御助言を賜りました．心より感謝申し上げます．

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は，本研究を進めるに当たり，広範囲かつ多大な御助力を頂きました．特に，学会発表や論文投稿時に貴重な御助言を頂戴いたしました．心より感謝申し上げます．

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伏田享平氏ならびに大蔵君治氏には，本研究を進めるに当たり，広範囲かつ多大な御助力を頂きました．心より感謝申し上げます．

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藤原賢二氏には，実験を進めるにあたり貴重な御助言・御協力を頂戴いたしました．心より感謝申し上げます．

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，ならびにソフトウェア工学講座の皆様には，日頃より多大な御協力と御助言を頂き，公私ともに支えていただきました．ありがとうございました．最後に，日頃より私を励まし，温かく見守ってくれた家族に心より深く感謝します．

参考文献

[1] Martin Fowler, 児玉公信 (訳), 友野晶夫 (訳), 平澤章 (訳), 梅澤真史 (訳). リファクタリング - プログラミングの体質改善テクニック. ピアソンエデュケーション, 2005.

[2] N. Gold and K. Bennet. Hypothesis-based concept assignment in software maintenance. In *IEE Proceedings - Software*, Vol. 149, pp. 103–111, 2002.

[3] Brian Henderson-Sellers. *Object-oriented metrics : measures of complexity*, pp. 142–147. Prentice-Hall, 1996.

[4] Ted J.Biggerstaff, Bharat G.Mitbander, and Dallas Webster. The concept assignment problem in program understanding. In *Proc. of 15th International Conference on Software Engineering(ICSE 1993)*, pp. 482–498, 1993.

[5] R. Hierons M. Harman, N. Gold 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.

[6] 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.

[7] Edward Yourdon, Larry L.Constantine, 原田実 (訳), 久保未沙 (訳). ソフトウェアの構造化設計法, pp. 123–165. 日本コンピュータ協会, 1986.

[8] 佐藤浩, 小野功, 小林重信. 遺伝的アルゴリズムにおける世代交代モデルの提案と評価. 人工知能学会誌, Vol. 12, No. 5, pp. 734–744, 1997-09-01.

[9] 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.

[10] 独立行政法人情報処理推進機構（IPA）, ソフトウェア・エンジニアリング・センター（SEC）. ソフトウェア開発データ白書 2009. 日経 BP 社, 2009.

[11] 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, 第 J92-D 巻, pp. 1071–1073,

2009.

[12] 兼光智子, 肥後芳樹, 楠本真二. プログラム依存グラフを用いたリファクタリング候補の特定と可視化. 電子情報通信学会技術報告, Vol. 110, No. 336, pp. 61–66, 2010.