業務システム理解のための
外部システムとの入出力を用いたクラスタリング手法

秦野 智臣[1,a)]　石尾 隆[1,b)]　師 芳卓[2,c)]　坂田 祐司[2,d)]　井上 克郎[1,e)]

概要：数十年以上前に開発された業務用の情報システムは，現代でも重要な役割を果たしている．このような古いシステムは大規模かつ複雑であり，保守性と生産性が低下していることが多いことから，今後の保守作業に適した構造へと再設計することが重要となっている．しかし，古いシステムではソースコードがうまく整理されておらず，ソースコード中の識別子にその役割を表すような名前が付けられていないことから，その構造を理解することが困難である．そこで本研究では，識別子を利用せずに，ソースコード中の関数を動作の類似性によって分類するクラスタリング手法を提案する．手法によって形成されたクラスタに含まれるある 1 つの関数を理解すれば，同じクラスタに属する関数を理解する際に，その知識が利用できると考えられる．提案手法は，システムの操作画面やデータベースへのアクセスといった業務システムの基本的な命令を各関数から抽出し，その命令列の類似度によって動作が類似した関数を検出する．手法を評価するために，2 つの Java システムに対して適用実験を行った．その結果，提案手法は，人手によるクラスタリングに近いクラスタリングを行えることを示した．

Understanding Business System through Software Clustering Using I/O Instructions for External Systems

Abstract: Business systems written decades ago play a key role in modern society. The maintenance of these systems requires a huge effort because they are typically large-scale and complicated. System modernization is important to facilitate the maintenance task of a legacy system. However, understanding a system for its modernization is a difficult task; one of the main reasons is that identifier names in the legacy code do not represent their roles. This paper proposes a software clustering technique which groups functions in source code by similarity of their actions without using identifier names. The technique can provide a benefit: if a developer understands a function included in a cluster, the developer can easily understand other functions included in the same cluster. This technique extracts a sequence of input/output instructions for a database and a user interface from each function to detect similar functions. An experiment for two Java systems showed that the clustering results by the technique resemble those by a developer.

1. はじめに

現在，多くの組織が業務を遂行するための情報システム，すなわち業務システムを利用している．業務システムの中には，数十年以上前から保守開発が続けられているものも多く存在しており，継続的な機能追加や仕様変更により，システムの大規模化と複雑化が進み，保守費用の増加を招いている [1]．そのため，システムをユーザの要求の変化に合わせて迅速に変更していくことも困難となっている．このような問題を解決するために，現行のシステムの構造を分析し，今後の新機能の追加や保守作業に適した構造へと再設計することが重要となっている．

システムの構造を理解する際によく問題になるのが，システムの設計書が残されていないこと，もしあっても，最新の状態が記述されていない場合が多いことである [2]．対象システムの保守開発を行っている開発者がいる場合でも，最近の変更点しか理解しておらず，システム全体の構

1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
2 株式会社 NTT データ
NTT DATA Corporation
a) t-hatano@ist.osaka-u.ac.jp
b) ishio@ist.osaka-u.ac.jp
c) moroys@nttdata.co.jp
d) sakatayu@nttdata.co.jp
e) inoue@ist.osaka-u.ac.jp

© 2015 Information Processing Society of Japan

造や動作までは把握していないことがある．これは多くのプログラム理解の研究における共通の課題であるが，古い業務システムの構造を理解する作業においては，さらに次のような難しさが加わる．

(1) ファイルがディレクトリに適切に分割されていない．現代のシステム開発における主要言語である Java ではパッケージ構造を用いてソースコードを階層的に整理することが一般的であるが，開発当時にはそのような仕組みが利用されておらず，システムのすべてのファイルが 1 つのディレクトリに存在しており，システム全体の構造が分からなくなっている．
(2) 識別子として，設計時に機械的に割り当てた文字列（たとえば FUNC0001，FUNC0002，··· のような連番）など，機能と無関係な名前が使用されている [3]．現在では，ソースコード中の識別子にその役割を表す名前を付けることが主張されている [4] が，当時はそのような識別子は利用されておらず，名前からその役割や機能を推測することが難しい．

このうち，前者の問題に対しては，ソフトウェアクラスタリングが提案されている．ソフトウェアクラスタリングは，システムのソースコードの構成要素（ファイルや関数など）を何らかの観点に基づく類似性によって分類する技術であり，システムを分割して複雑度を下げることで大規模システムの理解を支援する [5]．これまで提案されている多くの手法は，ファイル間の依存関係や識別子の類似度に基づいて，システムを凝集度が高く結合度の低いサブシステムに分類することを目的としている [6]．Kobayashi らは，クラスタリングの結果に基づいてシステムの構造を可視化し，既存システムの構造分析を支援する手法を提案している [7]．

システムの構造の分析が活発に研究されている一方で，後者の問題への対応策は提供されていない．たとえば，Kobayashi らの可視化手法 [7] では，クラスタに割り当てるキーワードとして，パッケージ名やクラス名を使用しており，機能などを表現する識別子が付けられていない古い業務システムの分析にそのまま適用することが難しい．

本研究では，識別子に意味のある名前が与えられていない業務システムにおいて，関数（Java の場合はメソッド）をその動作の類似性に基づいて分類するクラスタリング手法を提案する．業務システムでは，社員情報や在庫情報といった様々な業務ごとのデータについて，登録，検索，更新，削除などの類似した動作が実装されていることが多い．そのため，扱うデータは異なるが類似した動作をする関数をまとめることで，集合に含まれる 1 つの関数を理解すれば，同じ集合に属する他の関数を理解する際に，その知識が利用できると考えられる．また，既存のクラスタリング手法で業務に対応するサブシステムが得られていれば，そのサブシステムに異なる動作を表現したクラスタがどれだけ含まれているかで，サブシステム内部の機能の数を推測することができる．

具体的な手法としては，プログラムの各関数を，システムの操作画面やデータベースへのアクセスといった業務システムに共通する命令の列で表現する．そして，プログラムの各関数を頂点，関数の命令列の類似度を重みとした辺を持つグラフを作成し，密接に関連した頂点のグループを検出するクラスタリングアルゴリズムを適用することで，類似度の高い関数をクラスタとしてまとめる．

提案手法を評価するために，Java で書かれたシステムに対する実装を行い，適用実験を行った．2 つのシステムを対象として，それらのメソッド名などを参照して手作業で作成したクラスタを正解とみなして，手法を適用した結果得られたクラスタを評価した．また，提案手法によるクラスタリングが，既存研究で行われている凝集度と結合度に基づくクラスタリングとは異なることを確認するために，関数の呼び出し関係によるクラスタリングとの比較を行った．

以降，2 章では既存のクラスタリング手法とその関連技術であるソースコードの類似性を認識する手法について述べる．3 章では提案手法を述べ，4 章では手法を適用した評価実験について述べる．5 章では手法の実用化に向けた議論を行い，最後に 6 章でまとめと今後の課題を述べる．

2. 背景

2.1 ソフトウェアクラスタリング

ソフトウェアクラスタリングとは，システムのソースコードの構成要素（ファイルや関数など）を，何らかの観点に基づく類似性によって分類する技術である．ソフトウェアクラスタリングでは，まず，ソースコードの構成要素からある特徴を抽出し，要素間の関連を数値化する．そして，機械学習において利用されているクラスタリングアルゴリズムを適用し，関連の強い要素を 1 つのクラスタとしてまとめる．クラスタリングの結果は，システムの構造を復元したり，構造の改善点を発見したりするために使われている [6], [7]．Mitchell らは，クラスタリングを用いて可視化したシステムの構造が，保守開発者にとって有用であることを示している [8]．これまでに，様々なクラスタリング手法が提案されてきたが，それぞれの目的に応じた特徴の抽出方法が用いられている．

Mancoridis らは，関数の呼び出し関係や変数の参照関係をファイル間の依存関係として抽出し，対象システムを凝集度が高く結合度の低いサブシステムに分類することを目的としている [9]．サブシステムに分類することによってシステムの複雑度を下げ，システムの構造理解を支援している．[10] では，よく使われるライブラリ関数の組合せ（イディオム）を検出するために，関数の呼び出し関係を依存関係として抽出している．しかし，依存関係によるクラ

© 2015 Information Processing Society of Japan

スタリングは，様々な関数から呼び出される関数が存在する場合，クラスタリング結果を開発者が手動で洗練する必要があるという問題があった [11]．そこで，そのような関数を除去する手法や，その影響を軽減する手法が提案されている [12]．Kobayashi らは，[12] の手法を用いてシステムの構造を可視化し，構造の理解を支援する手法を提案している [7]．

ソースコード中の識別子の類似性に基づくクラスタリング手法も提案されている．Anquetil らは，識別子に含まれる単語の類似度を用いたクラスタリングが有用であることを示しており [13]，Andritsos らは，単語の出現頻度による重みを考慮することが重要であることを示している [14]．また，Anquetil らは，ファイル名の命名規則を用いたクラスタリング手法を提案している [15]．これらの手法は識別子にその役割を表す名前が付けられている場合，有用であると考えられる．

Scanniello らは，オブジェクト指向言語で書かれたシステムを対象に，クラスの継承関係とインターフェースの実装関係を用いたクラスタリング手法を提案している [16]．この手法は，クライアントサーバシステムでよく用いられる階層化アーキテクチャの層構造を認識するために，オブジェクト指向言語の特徴を利用しており，認識された層ごとに識別子の類似性によるクラスタリングを適用している．

Tzerpos らは，大規模システムでよく見られるパターンに基づくクラスタリング手法を提案している [17]．この手法におけるパターンとは，たとえば，C 言語で書かれたシステムにおいて fileA.c と fileA.h を同じクラスタにまとめる，デバイスドライバを同じクラスタにまとめる，様々な関数から呼び出される関数を同じクラスタにまとめるなどといったクラスタリングの方針を指す．これらのパターンを対象システムの特徴に合わせて開発者に定義してもらうことで，開発者にとって理解しやすいクラスタリングを行うことを目的としている．

2.2 ソースコードの類似性を認識する技術

2.2.1 コードクローン検出

コードクローンとは，ソースコード中に存在する同一または類似した部分を持つコード片のことである．これまでの研究において，コードクローンを検出する様々な手法が提案されており，識別子の情報を利用せずに，大規模システムにおける類似したコード片や関数を検出する手法も提案されている [18]．[18] では，ソースコードをトークンの列に変換し，ある一定以上の長さで一致しているトークン列をコードクローンとして検出している．

しかし，これらのコードクローン検出手法は，本研究の目的に対しては適切ではない．コードクローン検出では，サブルーチンの呼び出しを考慮した類似処理の検出は行われていないが，本研究ではそれを考慮する必要がある．また，コードクローン検出における主な目的は，類似処理の集約やクローンとなっている複数のコード片に対する一貫した修正を行うという作業を支援し，プログラムの保守性を向上させることである．そのため，コードクローン検出ではある程度長いコード片の検出を行うが，本研究では，短い関数であっても動作が類似していれば同じクラスタとして認識する必要がある．さらに，コードクローン検出では，データベースを操作する前後でのエラー処理といった動作の種類に関係なく行われる処理をクローンとして検出してしまい，異なる機能内で同様の処理が行われている関数を類似していると判定してしまう可能性がある．これらの処理は，関数の動作の特徴を表すものではないため，本研究ではこれらの処理の類似性を無視する必要がある．

2.2.2 類似したアプリケーションの検出

McMillan らは，ソースコードの再利用を効率的に行うために，類似した Java アプリケーションを検出する手法を提案している [19]．この手法では，同じパッケージや同じクラスに属するメソッドを呼び出しているアプリケーションを類似したアプリケーションとして検出している．この検出方法は，同じ API を利用するものは似ているアプリケーションであるという考え方に基づいている．また，この手法は，パッケージやクラスといった概念で整理されたアプリケーションがオープンソースとして数多く公開されていることを利用している．

3. 提案手法

本研究では，システムのソースコードを与えると，動作が類似している関数を 1 つのクラスタにまとめるクラスタリング手法を提案する．動作が類似しているとは，関数が扱うデータの内容によらず，データの検索，更新，削除といった処理の内容が類似していることを言う．本手法は，ソースコード中の識別子を利用しないため，識別子に意味のない名前が使われているシステムに対して適用することができる．また，本論文では，実験対象としたシステムで用いられている Java を例として提案手法について説明するが，業務用のプログラミング言語として広く用いられている COBOL などの他言語への適用も可能となるように，一般的な手続き型言語に存在する概念のみを使用した手法となっている．

本手法は，システムの操作画面に対する入出力処理と SQL 文を実行する処理（これらをまとめて外部アクセスと呼ぶ）によってメソッドを特徴づける．外部アクセスは，システムが様々な機能を提供するための基本的な処理であるため，メソッドの動作内容をよく表していると考えられる．また，システムの機能を実装するためには，外部アクセスの実行を条件分岐や繰り返しなどによって制御する必要があるため，これらの制御文もメソッドの特徴として抽出する．

© 2015 Information Processing Society of Japan

対象システム中のどのメソッドが外部アクセスに対応するかは，開発者に定義してもらう．外部アクセスを行う方法は限定されているため，対応関係を定義してもらうことは現実的であると考えられる．たとえば，Java では JDBC の API を呼び出すことによってデータベースを操作する方法が一般的であり，その他のライブラリを利用する場合も，データベースを操作するためには決められた API を呼び出す必要がある．

本手法は，以下の 3 つのステップからなる．

Step 1．プログラムの変換　各メソッドを，外部アクセスと制御文で構成されるプログラムに変換する．外部アクセスと制御文は，それぞれの種類に対応した記号で表現され，メソッドは記号列で表現される．

Step 2．類似度の計算　変換した各記号列を N-gram による集合で表し，すべての集合間のジャッカード係数を計算する．この値をメソッド間の類似度とする．

Step 3．クラスタリングアルゴリズムの適用　各メソッドを表す頂点と，Step 2 で計算した類似度を重みとした辺を持つグラフを作成する．このグラフに対してクラスタリングアルゴリズムを適用し，クラスタリング結果を得る．

以降では，各ステップの詳細について説明する．

3.1 Step 1．プログラムの変換

クラスタリング対象の各メソッドを，外部アクセスと制御文で構成されるプログラムに変換する．外部アクセスと制御文は，それぞれの種類に対応した記号で表現され，メソッドは記号列で表現される．本手法で抽出する文と本論文における記号表現を**表 1** に示す．どのメソッド呼び出しが外部アクセスに対応するかは，開発者が定義する．また，外部アクセスに対応するメソッド呼び出し文は，その文が単一のレコードを扱うか，複数のレコードを扱うかを区別する．本論文ではそれぞれの記号に，単一レコードの場合は s を，複数レコードの場合は m を添えることによって表現する．単一レコードと複数レコードのどちらを扱うメソッドであるかは，メソッドのシグネチャによって判断する．SELECT 文を実行するメソッドと画面から値を読み込むメソッドの呼び出しについては，そのメソッドの返り値が配列かリスト（java.util.List）である場合は複数レコードとし，そうでない場合は単一レコードとする．これら以外のメソッド呼び出しについては，そのメソッドの引数に配列かリストが含まれる場合は複数レコードとし，そうでない場合は単一レコードとする．

各メソッドは，抽象構文木の探索によって，ソースコード上の順番で記号列に変換される．抽象構文木とは，プログラムの構文解析の結果を木構造で表現したものである．抽象構文木によって，条件分岐や繰り返しなどのプログラムの静的な構造を認識することができる．本手法では，抽象構文木の深さ優先探索を行い，その訪問順に対応する記号を並べる．外部アクセスと制御文の開始はその頂点を初めて訪問したときに，制御文の終了はその頂点以下の探索が終了したときに，対応する記号を取得する．外部アクセス以外のメソッド呼び出しに対応する頂点を訪問した場合は，その呼び出し先メソッドを再帰的に探索し，記号列を取得する．抽象構文木の探索が終了したら，取得した記号列を前から順に走査し，外部アクセスを間に含んでいないような制御文の開始と終了の記号の組を記号列から削除する．本手法の実装は，Cesare らの手法 [20] を参考にしており，Java development tools [*1] の抽象構文木を利用している．メソッド呼び出しは Class Hierarchy Analysis [21] によって解決したが，本論文の実験対象システムでは呼び出し先が動的に切り替わるようなコードは存在しなかった．

図 1 のコードを用いて変換例を示す．図 1 のコードは，あるシステムの顧客情報管理画面において，ユーザが選択したすべての顧客情報を顧客データベースから一括で削除する機能を実装しているメソッドである．このメソッドは，選択された各顧客情報に対して，それを削除して問題ないか確認してから削除を行う．確認処理は 7，8 行目で，注文テーブルの検索を行い，削除しようとしている顧客が何も注文していないことを確認している．注文がなければその顧客情報を削除し，そうでなければスキップする．このメソッドに対して，外部アクセスの対応付けを**表 2** のように定義したとすると，図 1 のメソッドの 4 行目から 11 行目までの内容が [Rm, L, Sm, If, Ds, If}, L}] という記号列で表現される．

表 1　抽出する文の一覧

カテゴリ	文	記号
SQL 文の実行	SELECT 文を実行する	Ss，Sm
	INSERT 文を実行する	Is，Im
	UPDATE 文を実行する	Us，Um
	DELETE 文を実行する	Ds，Dm
画面との入出力	画面から値を読み込む	Rs，Rm
	画面に値を書き込む	Ws，Wm
制御文	if 文の開始	If
	if 文の終了	If}
	else 節の開始	E
	for, while, do 文の開始	L
	for, while, do 文の終了	L}

表 2　図 1 のコードに対する外部アクセスの対応付け

外部アクセス	対応するメソッド
SELECT 文の実行	OrderDao.select
DELETE 文の実行	CustormerDao.delete
画面からの入力	Vo.getSelectedCustomers

*1　http://eclipse.org/jdt/

```
1  void batchDeleteCustomer(CustomerDao cstDao,
2                           OrderDao ordDao, Vo vo) {
3    // 選択された顧客のIDを取得する
4    String[] idArray = vo.getSelectedCustomers();
5    for (String id: idArray) {
6      // 削除対象の顧客が注文中でないか確認する
7      List<OrderDto> ordList = ordDao.select(id);
8      if (ordList.isEmpty()) {
9        cstDao.delete(id); // 顧客情報をテーブルから削除する
10     }
11   }
12 }
```

図 1　選択された顧客の情報を一括で削除するメソッド

3.2　Step 2. 類似度の計算

Step 1 で変換したすべての記号列間の類似度を計算する．記号列間の類似度は，各記号列を N-gram から構成される集合で表現し，集合間のジャッカード係数を計算することで求める．N-gram とジャッカード係数は，文字列間の類似度を計算する手法として広く使われており，同じく文字列間の類似度計算として使われている最長共通部分列と比較して計算コストが小さい．そのため，N-gram とジャッカード係数による類似度計算は，大規模システムの解析により適していると言える．

N-gram は，文字列を長さ N の文字列に分解して記述する手法である．N はパラメータであり，N = 3 とする tri-gram がよく用いられる．例として，文字列 “ABCDE” を tri-gram による集合で表現すると以下のようになる．

$$\{\$\$\mathrm{A}, \$\mathrm{AB}, \mathrm{ABC}, \mathrm{BCD}, \mathrm{CDE}, \mathrm{DE}\$, \mathrm{E}\$\$\} \tag{1}$$

’$’ は，文字列の先頭と末尾を表す特殊文字である．このように，N-gram を用いることで文字列の順序を考慮しつつ，それを集合で表現することができる．

集合間の類似度は，ジャッカード係数により計算する．ジャッカード係数は，2 つの集合 X, Y に共通する要素の割合を表すものであり，以下のように定義される．

$$\mathrm{Jaccard(X, Y)} = \frac{|\mathrm{X} \cap \mathrm{Y}|}{|\mathrm{X} \cup \mathrm{Y}|} \tag{2}$$

定義より，X と Y が集合として等しい場合，ジャッカード係数は 1 となり，X と Y に共通する要素がない場合は 0 となる．この定義にしたがって，各記号列を N-gram で表した集合間のジャッカード係数を計算し，その値をメソッド間の類似度とする．

3.3　Step 3. クラスタリングアルゴリズムの適用

Step 2 で計算した類似度を用いて，メソッドのクラスタリングを行う．これまでに様々なクラスタリングアルゴリズムが提案されてきたが，本研究では Newman らが提案したアルゴリズム [22] を用いる．このアルゴリズムは，ソーシャルネットワークにおけるコミュニティを検出する目的で提案されたが，ソフトウェアクラスタリングにおいても有用であることが示されている [12], [23]．また，アルゴリズムの計算複雑度は，クラスタリングの対象となる要素の数を |V| としたとき，$O(|\mathrm{V}|\log^2|\mathrm{V}|)$ であり [24]，大規模システムのクラスタリングに適している．

Newman らのアルゴリズムは，クラスタリングの対象となる要素をグラフの頂点で表し，要素間の関連を対応する頂点間の辺で表す．辺には，関連の強さを表す重みを付けることができる．このグラフにおいて，以下に定義される Q 値が最大となるようなクラスタリングを求める．

$$\mathrm{Q} = \frac{1}{W}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{W}\right]\delta(c_i, c_j) \tag{3}$$

ここで，A_{ij} は，頂点 i と頂点 j を接続している辺の重みを表す．W は，すべての頂点間の辺の重みの和であり，$W = \sum_{i,j} A_{ij}$ で表される．k_i は，頂点 i に接続されているすべての辺の重みの和であり，$k_i = \sum_j A_{ij}$ で表される．c_x は，頂点 x が属するクラスタであり，$\delta(c_i, c_j)$ は，$c_i = c_j$ のとき 1 をとり，そうでないときは 0 をとる関数である．Q 値は，クラスタ内の辺の重みの和が大きい状態であると高くなり，クラスタ間の辺の重みの和が大きい状態であると低くなる．したがって，Q 値が最大となるようなクラスタリングを求めることで，クラスタ内の関連が強い分類を求めることができる．しかし，Q 値の最大化は NP 困難であるため [25]，Newman らのアルゴリズムでは以下のような貪欲法で Q 値最大化の近似計算を行う．

(1) すべての頂点がそれぞれ 1 つのクラスタであるとする．
(2) Q 値が最も高くなるように 2 つのクラスタを併合する．
(3) すべてのクラスタが 1 つに併合されるまで 2 を繰り返す．
(4) 併合過程の中から最も Q 値が高い状態を選び，それをクラスタリングの結果とする．

提案手法では，プログラムの各メソッドをグラフの頂点とし，Step 2 で計算した類似度を頂点間の辺の重みとして Newman らのアルゴリズムを適用する．ただし，すべての頂点間に辺が存在していると，Newman らのアルゴリズムにおける併合が過剰に行われてしまい，類似度の低いメソッドが同じクラスタに分類されてしまうと考えられるため，類似度がある閾値未満であるメソッドについては，それらの間の辺の重みを 0 とする．したがって，2 つのメソッド m_i，m_j に対応する頂点間の辺の重み k_{ij} は，それぞれの N-gram による集合を M_i，M_j とすると，以下の式で表される．

$$k_{ij} = \begin{cases} \mathrm{Jaccard}(M_i, M_j) & \text{if } (\mathrm{Jaccard}(M_i, M_j) \geq th) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

閾値 th は手法のパラメータとし，類似度の分布によって変化させるものとする．

© 2015 Information Processing Society of Japan

4. 評価実験

提案手法を実装し，2 つの Java システムに対して適用した．本実験では，クラスタリングの良さを計測する 3 つの指標（4.1 節で述べる）を計測し，手法の評価を行う．また，手法のパラメータを変化させた場合に，クラスタリング結果がどのように変化するかを調査するために，手法の Step 2 における N-gram の N を 1，3，5 と変化させ，Step 3 における類似度の閾値 th を 0.0 から 1.0 まで 0.1 刻みで変化させて各指標の計測を行う．N=1 とすることは，記号列の順序を考慮せずに単純な集合として比較することを意味する．また，th=1.0 とすることは，N-gram による集合が完全に一致しているメソッドのみを同じクラスタにまとめることを意味する．一般に，N を小さくすると類似度は高くなりやすく，N を大きくすると類似度は低くなりやすいため，N が小さい場合は th を高く設定し，N が大きい場合は th を低く設定するべきであると予想される．

提案手法が，凝集度と結合度に着目したクラスタリング手法とは異なるクラスタリングを行うものであることを確認するために，メソッドの呼び出し関係を用いたクラスタリングを実装した．呼び出し関係を用いたクラスタリングは，各メソッドを頂点とし，呼び出される可能性のあるメソッド間を辺で接続したグラフに，提案手法と同様に Newman らのアルゴリズムを適用することによって行った．また，提案手法の適切な比較対象となる既存研究が存在しないため，メソッド名が同じであるものを 1 つのクラスタとするクラスタリングを行い，メソッド名を利用しない本手法を評価するための目安とした．

4.1 評価方法

クラスタリング手法の評価は，以下の 3 つの指標を計測することによって行う [26]．

信頼性　手法によるクラスタリングが正しいクラスタリングにどのくらい近いか

クラスタの分布　クラスタの大きさが極端でないか

安定性　システムのソースコードが少し変更されても，クラスタリング結果が大きく変わらないか

4.1.1 信頼性

信頼性は，手法の目的に合ったクラスタリングを手作業で行い，それを正解とみなして手法によるクラスタリングとの距離を計測することによって評価する．本実験では，企業におけるリエンジニアリングの専門家である第三著者が，理解において有用であると考えるクラスタリングを手作業で行った．手作業のクラスタリングは，対象システムのデータベースに対してどのような操作が行われるかという観点で分類されている．各クラスタは 1 レコードの登録，更新，検索，項目詳細表示，複数レコードの一括更新といったシステムの動作に対応している．クラスタリング作業は，主に対象システムのソースコードを読解することによって行い，システムの操作マニュアルやメソッド名なども参考にした．

2 つのクラスタリング間の距離を計測する指標として，MoJoFM [27] を用いた．MoJoFM は，あるクラスタリング A から別のクラスタリング B に変換するときに，以下の 2 つの操作が何回必要であるかを計測する指標である．

Move　あるクラスタの要素を別のクラスタに移動させる．移動させる要素を新たに 1 つのクラスタとすることも含む．

Join　2 つのクラスタを併合し，1 つのクラスタにする．

MoJoFM は，上記の操作回数が少ないほど A は B に近いとする指標であり，以下のように定義される．

$$\mathrm{MoJoFM}(A,B) = (1 - \frac{mno(A,B)}{max(mno(\forall A,B))}) \times 100\% \quad (5)$$

ここで，$mno(A,B)$ は，A から B に変換するのに必要な最小の操作回数である．$max(mno(\forall A,B))$ は，B に変換するのに必要な最小の操作回数が最大であるようなクラスタリングの操作回数である．MoJoFM の値が高ければ高いほど，A は B に近いことを意味し，MoJoFM が 100% であれば，A は B に等しいことを意味する．MoJoFM は距離を計測する指標として提案されているが，MoJoFM(A, B) と MoJoFM(B, A) は一般に等しくないため，A を手法によるクラスタリングとし，B を正解のクラスタリングとして評価を行う．

4.1.2 クラスタの分布

クラスタの大きさが極端であるかどうかは，Non-Extremity of cluster Distribution（NED）によって計測される．NED は，クラスタの要素数が決められた範囲に収まっているかを割合で表現したものであり，以下のように定義される．

$$\mathrm{NED}(C) = \frac{1}{M} \sum_{c \in C; 5 \le |c| \le \max(20, M/5)} |c| \quad (6)$$

ここで，c はクラスタリング C におけるクラスタであり，$|c|$ は c の要素数である．また，M はクラスタリング C におけるすべての要素数である．範囲の下限と上限は，すべての要素数 M を考慮して適切な値を決定する必要がある．ここでは，本実験と要素数が近い既存研究の定義 [12] を用いた．NED が 1 に近ければ，クラスタの大きさが極端でないことを意味する．

4.1.3 安定性

安定性は，同じシステムの連続した 2 つのバージョンに対してクラスタリング手法を適用し，得られた 2 つのクラスタリング間の距離を計測する．安定性における距離は，クラスリング A, B 間の両方向の距離が等しい MoJoSim [28] を用いる．MoJoSim は MoJoFM と同様に Move と

© 2015 Information Processing Society of Japan

表 3　対象システム

システム名	対象メソッド数	テーブル数	クラスタ数
MosP	253-334	56-75	7
販売管理システム	9	6	3

Join の回数を数えるが，以下の定義のように A から B と B から A のうち，操作回数が少ない方向を採用する．

$$\mathrm{MoJoSim}(A,B) = (1-\frac{OP}{N}) \times 100\% \quad (7)$$

$$OP = min(mno(A,B), mno(B,A)) \quad (8)$$

連続した 2 バージョンに対する MoJoSim が高ければ，そのクラスタリング手法は安定性が高いことを意味する．ただし，2 バージョン間でクラスタリングの要素が異なる場合は，両バージョンに共通している要素に対して MoJoSim を計算する．複数の連続したバージョン間に対して MoJoSim を計算し，それらの平均値を求めることで安定性の評価を行う．

4.2　実験対象

本実験では，勤怠管理システムである MosP 勤怠管理 V4 [*2] と，ある企業における開発練習用の販売管理システムを用いて手法の評価を行う．それぞれのシステムの規模を**表 3** に示す．表 3 における対象メソッド数とは，ユーザからのボタン操作などによって呼び出されるメソッドのうち，データベースへのアクセスを行うものの数であり，これらのメソッドをクラスタリングの対象とする．MosP では JDBC，販売管理システムでは iBATIS をそれぞれ利用してデータベースへのアクセスを行う．テーブル数とは，対象メソッドからアクセスされるテーブルの数である．MosP では，クラス名が Action で終わるクラスの action メソッドから呼び出されるメソッドを対象メソッドとした．テーブル数は，sql ファイルにおいて “CREATE TABLE” によって定義されているテーブルの名前を取得し，フィールド変数 TABLE にそれらの名前が定義されているクラス数によって計測した．表中の “-” は，MosP の 11 バージョン（4.0.0-4.4.2）における最小値と最大値を示しており，これらのバージョンを用いて安定性の評価を行う．信頼性とクラスタの分布は，最もメソッド数が多い最新バージョンを用いてそれぞれ評価を行う．表のクラスタ数は，手作業で作成したクラスタの数である．

提案手法を適用するには，外部アクセスに対応するメソッドを定義する必要があるため，第一著者がソースコードとコメントを読み，**表 4** のような対応付けを行った．MosP では，クラス名が “Dao” で終わるクラスがデータベースを操作する機能を提供しており，クラス名が “Vo” で終わるクラスが操作画面とシステムの間で値の受け渡しをしている．販売管理システムについても，ソースコードと仕様書を読み，メソッドの対応付けを行った．

[*2] http://sourceforge.jp/projects/mosp/releases/62164

表 4　MosP におけるメソッドの対応付け

外部アクセス	クラス名	メソッド名
SELECT 文	Dao で終わる	findFor で始まる
INSERT 文		insert
UPDATE 文		update
DELETE 文		delete，logicalDelete
画面からの入力	Vo で終わる	get で始まる
画面への出力		set で始まる

4.3　結果と考察

4.3.1　MosP

MosP に対して手法を適用した結果を**表 5** に示す．表における MoJoFM の列は，手法によるクラスタリングが人手によるクラスタリングにどのくらい近いかを MoJoFM で計測した値である．NED は，クラスタの分布が極端でないかを計測した値であり，MoJoSim は，連続したすべてのバージョンに対するクラスタリング間の MoJoSim の平均値であり，手法の安定性を表している．表の各列において最も高い値を太字で表記している．また，表の最下段には，メソッドの呼び出し関係によるクラスタリングとメソッド名によるクラスタリングにおける MoJoFM の値をそれぞれ示している．

表 5 から，MoJoSim はパラメータによらず高いことが分かる．これは，機能の仕様や実装が変更されても，外部アクセスが大きく変わることが少ないためであると考えられ，手法の安定性が高いことを示している．NED は N=5 かつ th=0.1 の場合に最も高い値（0.94）となっており，極端な大きさのクラスタが非常に少ないことを示している．MoJoFM は N の値によって異なるが 45 から 55%であり，メソッドの呼び出し関係によって得られるクラスタリングよりも高い値が得られたことから，動作が類似したメソッドに分類する目的においては，提案手法の方が正しいクラスタリングを行えるといえる．

MoJoFM と NED の値がバランスよく高い結果を分析するために，MoJoFM と NED のそれぞれの値によってパラメータの順位付けを行い，2 つの順位の和の上位 10 件を抽出した．その結果を**表 6** に示す．順位の列の括弧内の数字は，それぞれの指標の値を表す．表 6 から，N=1 かつ th=1.0 の場合 MoJoFM は最も高いが，NED は N=3 や N=5 の場合に比べて低い値になっている．N=1 かつ th=1.0 の場合は，抽出された文の種類が集合として等しいメソッドのみを同じクラスタに集めることを意味している．そのため，手法の分類が非常に細かく，人手によるクラスタリングで異なるクラスタに属するメソッドが同じクラスタに含まれてしまうことは少ないが，要素数 1 のクラスタが数多く形成されてしまう．実際に，88 個のクラス

表 5 MosP に対する指標

	N=1			N=3			N=5		
th	MoJoFM	NED	MoJoSim	MoJoFM	NED	MoJoSim	MoJoFM	NED	MoJoSim
0.0	36.70	0.0	98.26	38.23	0.19	91.11	39.14	0.21	92.87
0.1	36.70	0.0	98.33	32.72	0.17	88.67	**50.76**	**0.94**	92.29
0.2	36.70	0.0	97.88	45.57	0.70	94.67	48.62	0.74	95.73
0.3	35.78	0.0	96.81	**46.48**	**0.73**	95.01	46.18	0.51	97.05
0.4	36.70	0.0	98.15	44.95	0.54	97.23	42.51	0.43	96.78
0.5	42.81	0.11	96.81	44.04	0.42	97.18	35.47	0.34	96.69
0.6	37.61	0.0	94.60	36.70	0.36	96.88	29.97	0.27	97.13
0.7	45.26	0.18	95.98	31.19	0.26	97.41	27.22	0.24	97.14
0.8	49.85	0.23	**98.34**	25.69	0.20	97.55	20.49	0.16	97.91
0.9	50.46	0.21	97.82	18.35	0.10	98.27	16.51	0.10	98.72
1.0	**55.35**	**0.59**	96.94	15.60	0.08	**98.98**	14.98	0.08	**99.01**

呼び出し関係によるクラスタリングの MoJoFM : 26.91
メソッド名によるクラスタリングの MoJoFM : 77.37

表 6 パラメータの順位付け

N, *th*	MoJoFM 順位	NED 順位	順位の和	クラスタ数
5, 0.1	2 (50.76)	1 (0.94)	3	23
1, 1.0	1 (55.35)	5 (0.59)	6	88
5, 0.2	5 (48.62)	2 (0.74)	7	79
3, 0.3	6 (46.48)	3 (0.73)	9	84
3, 0.2	8 (45.57)	4 (0.70)	12	25
5, 0.3	7 (46.18)	7 (0.51)	14	135
3, 0.4	10 (44.95)	6 (0.54)	16	126
1, 0.8	4 (49.85)	15 (0.23)	19	25
1, 0.9	3 (50.46)	16.5 (0.21)	19.5	57
3, 0.5	11 (44.04)	9 (0.42)	20	152

タのうち，73 個のクラスタが要素数 5 個未満の非常に小さいクラスタであった．一方，N=3 あるいは N=5 のときに *th* を低く設定すると N=1 の場合に比べて NED は高いが，MoJoFM の値が低くなっている．これは，手法のクラスタリングにおいて，人手によるクラスタリングで異なるクラスタに属するメソッドが同じクラスタに含まれてしまったためである．実際に，N=5 かつ *th*=0.1 としたクラスタリングにおけるあるクラスタでは，27 個のすべて要素が search メソッドであったが，別のクラスタでは，13 個の batchUpdate メソッドと 8 個の delete メソッドが混在していた．このときの各クラスタの要素数は**図 2** のような分布となった．小さいクラスタはまだ存在しているが，N=1 の場合に比べて大きなクラスタが得られていることが分かる．

クラスタ数が図 2 のように比較的少ない場合，提案手法は，人手によるクラスタリングでは異なるクラスタに分類されるようなメソッドを，1 つのクラスタに含めてしまう傾向があった．そこで，表 6 においてクラスタ数が少ない 3 つのクラスタリングについて，各クラスタに対してクラスタリングを適用し，サブクラスタへの分類を行うという追加調査を行った．その結果を**表 7** に示す．表 7 から，サブクラスタへの分類によって MoJoFM が向上しており，特

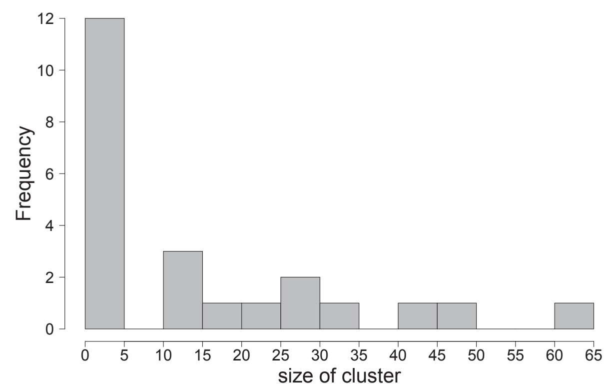

図 2 N=5，*th*=0.1 としたクラスタリングの要素数の分布を表すヒストグラム

に N を 3 あるいは 5 とした場合は，NED も高い値となっていることが分かる．メソッド名によるクラスタリングの MoJoFM は 77.37%であり，要素数 300 のランダムなクラスタリングの MoJoFM は平均して約 15%である [27] ことから，提案手法の値はメソッド名を利用しない手法としては高い値であると考えられる．

手法を実際のシステムに適用することを考えると，N と *th* の適切な値を開発者が判断する必要がある．MoJoFM の計測には人手によるクラスタリングが必要であるが，NED の計測には不要であるため，複数通りのクラスタリングを試行することができる場合は，NED の値によって判断することが考えられる．表 5 において MoJoFM と NED の相関係数は 0.62 であり，正の相関があることが確認されたため，NED の値が高いクラスタリングを選択すれば，それは信頼性の高いクラスタリングである可能性がある．ただし，NED はクラスタの分布の均一さを測る指標として用いられているが，分布が均一であることがシステム理解において常に有利であるとは限らない．そのため，複数の開発者で作業を分担することを考慮し，クラスタ数によ

© 2015 Information Processing Society of Japan

表 7 サブクラスタへの分類

N, th	MoJoFM	NED	クラスタ数
5, 0.1	59.94	0.88	48
3, 0.2	59.63	0.83	50
1, 0.8	56.27	0.66	32

る判断を行うことも考えられる．大規模システムでは複数通りのクラスタリングを試行することが困難であることも考えられるが，そのような場合は，表 7 の結果から，N=3 あるいは N=5 とし，th を 0.1 あるいは 0.2 などの低い値に設定し，段階的にクラスタリングを適用することで信頼性が高く，クラスタの大きさが適切なクラスタリングが得られることが期待できる．

4.3.2 販売管理システム

販売管理システムは規模が小さく，パラメータによる差異が小さいため，N=3，th=0.0 とした場合のクラスタリング結果のみを示す．図 **3** は，クラスタリングの結果を樹形図で示したものである．赤色の水平線は手法によるクラスタリング結果を表しており，その線で樹形図を切断すると，各ツリーが 1 つのクラスタに対応することを意味している．update1，search1 などの要素名は，人手によるクラスタリング結果を表しており，数字を除いて同じ名前のものが同じクラスタに分類されたことを意味する．図 3 から，検索を行うメソッドについては手法と人手による分類が一致しているが，その他のメソッドは両者の分類が異なっている．これは，発行する SQL 文が異なるメソッドでも，類似した動作を実現していることがあるためだと考えられる．たとえば，batch1 と batch2 はマスタテーブルの情報を一括で更新する処理であるが，batch1 はテーブルのレコードを全件削除した後，新たなレコードを INSERT 文によって登録することで更新処理を行っており，batch2 は UPDATE 文によって既存のレコードの情報を書き換えて更新処理を行っている．提案手法では，これらのメソッドが発行する SQL 文が異なるため，メソッド間の類似度は小さくなるが，人手によるクラスタリングでは，両メソッドとも複数のレコードを一括で更新する処理であることから似ているメソッドであると判断された．このように，人手によるクラスタリングでは，発行する SQL 文が異なっていても，それらの文が実現している処理の内容を考慮するため，提案手法によるクラスタリングと異なる結果になったと考えられる．また，販売管理システムでは，バッチ処理とオンライン処理を区別することが困難であった．オンライン処理においても INSERT 文を繰り返す処理が行われており，両者の制御構造が類似することがあった．これらを区別するためには，外部アクセスの種類だけでなく，データの入力元が画面であるかファイルであるかという情報を取得する必要があると考えられる．

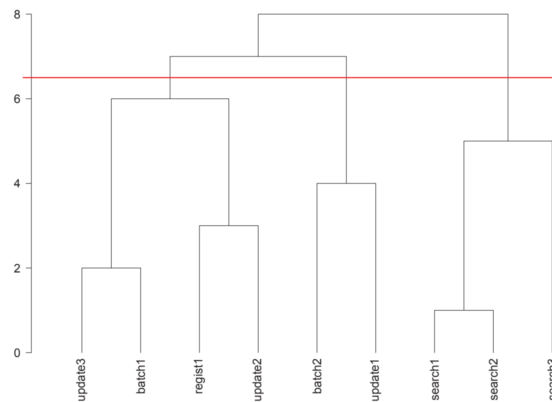

図 **3** 販売管理システムのクラスタリング

4.4 妥当性への脅威

本実験では，企業の専門家が作成したクラスタリングを用いて手法の評価を行った．専門家は，対象システムのソースコードを見て分類を行ったが，ソースコード中のメソッド名を参考にしたため，命名が不適切なものがあった場合は，目的と異なる分類を行っている可能性がある．また，異なる開発者が作成したクラスタリングを用いた場合は，本実験と結果が異なる可能性がある．

手法を対象システムに対して適用するために，第一著者が外部アクセスとメソッドの対応付けを行った．第一著者は，システムのソースコードを読解した上で，表 4 のようなクラス名とメソッド名によるルールで外部アクセスとメソッドの対応付けを行ったが，すべてのソースコードを目視で確認することは行っていない．そのため，実際には外部アクセスとして相応しくないメソッドを対応付けていたり，対応付けるべきメソッドを見逃している可能性がある．これらの対応付けの方法によって，実験結果が異なる可能性がある．

提案手法は，識別子に意味のある名前が与えられていないシステムに向けて設計されているが，本実験では，Java で書かれた業務システムを対象に手法の評価を行った．評価実験に用いることができるシステムが限られていることと，クラスタリングの信頼性を計測するために人手によるコード読解とクラスタリング作業が必要であることから，表 3 の Java システムを対象にしている．他のシステムや他言語で書かれたシステムに対しては，本実験と同じような有用性が得られない可能性がある．

5. 手法の実用化に向けた議論

提案手法は，識別子が有用でないシステムに対して適用することを想定している．そのため，実際のシステムで外部アクセスと関数の対応付けを行うことは，本実験用いたような Java システムにおける作業ほど容易ではない．しかし，一般的に，外部アクセスを行うためには決められた

© 2015 Information Processing Society of Japan

API や READ/WRITE などのシステムコールを呼び出す必要があるため，外部アクセスに対応する可能性のある関数や命令は限定される．また，対象システムの保守を行っている開発者であれば，部分的な理解しかできていない場合でも，どの関数によって外部アクセスが行われているかを判断できると考えられる．保守開発者が判断を行う作業は必要であるが，その結果を用いて提案手法によるクラスタリングを行うことで，他の開発者との知識の共有が可能になり，保守開発者へのヒアリングが集中することを避けることができる．また，1 つのクラスタに属する関数を類似処理としてまとめて理解することで，システム全体の理解が効率化されると考えられる．

本手法は，ファイル数が 10 万を超えるような大規模システムに対して適用することを目指しているため，計算コストの小さいアルゴリズムが要求される．本手法では，Step 1 における構文解析，Step 2 における N-gram によるジャッカード係数の計算，Step 3 における Newman らのクラスタリングアルゴリズムが主な処理となる．構文解析は，プログラミング言語によって異なるが，一般に高速に行うことができるように設計されていることが多い．N-gram によるジャッカード係数の計算では，すべての集合間で類似度を計算する必要があるが，並列計算による高速化が可能である．Newman らのクラスタリングアルゴリズムは，頂点数が 40 万を超えるグラフに対しても適用されている [24]．以上のことから，本手法は計算コストの観点において，大規模システムへの適用可能性が高いと考えられる．

6. まとめと今後の課題

本研究では，ソースコード中の各関数からシステムの操作画面やデータベースへのアクセスを行う命令を抽出し，動作が類似している関数の集合に分類するクラスタリング手法を提案した．評価実験では，2 つの Java システムに対して手法を適用し，クラスタリング結果を評価する指標を計測した．クラスタの分布の指標 NED を用いて適切なクラスタリング結果を選定できる可能性があり，また，大きなクラスタに対してはさらにクラスタリングを行うことによって段階的に分割することで，人手によるクラスタリングに近づくことが分かった．MoJoFM で計測した人手によるクラスタリングとの距離は，提案手法が約 60%，メソッド名による分類が約 77%であり，識別子に意味のある名前が与えられていない状況下でも動作が類似したメソッドを検出できることを示した．

本実験では，専門家がシステム理解において有用であると判断したクラスタリングと手法によるクラスタリングの距離を定量的に評価した．これらのクラスタリングは理解において有用であると考えているが，理解作業のコストへの影響は評価できていないため，作業コストを定量的に評価することが今後の課題である．さらに，企業で古くから開発が続いている大規模なシステムに対して手法を適用することも必要である．大規模システムでの適用結果の分析や，開発者からのフィードバックを得ることなどによって，手法の改善を行うことが考えられる．特に，非常に古いシステムでは，リレーショナルデータベースが使われていないこともあるため，ファイルの読み書きを行う命令を外部アクセスに対応付けてクラスタリングを行うことなども発展的な課題として挙げられる．また，提案手法は，外部アクセスに対応するメソッドを開発者が定義する必要があり，対象システムに関する知識がある程度要求されるため，対応付けを容易にすることが拡張的な課題として挙げられる．具体的には，メソッドの呼び出し関係を抽出し，様々なメソッドからの呼び出しが集中しているメソッドを外部アクセスの候補として提示することが考えられる．

謝辞　本研究は科研費 No.25220003, 26280021 の助成を得たものである．

参考文献

[1] 一般社団法人日本情報システム・ユーザー協会（JUAS）：ソフトウェアメトリックス調査 2012 (2012).

[2] Ducasse, S. and Pollet, D.: Software Architecture Reconstruction: A Process-Oriented Taxonomy, *IEEE Transactions on Software Engineering*, Vol. 35, No. 4, pp. 573–591 (2009).

[3] Sneed, H. M.: Object-oriented COBOL recycling, *Proceedings of the 3rd Working Conference on Reverse Engineering*, pp. 169–178 (1996).

[4] McConnel, S.: *Code Complete, Second Edition*, Microsoft Press (2004).

[5] Koschke, R. and Simon, D.: Hierarchical Reflexion Models, *Proceedings of the 10th Working Conference on Reverse Engineering*, pp. 36–45 (2003).

[6] Maqbool, O. and Babri, H.: Hierarchical Clustering for Software Architecture Recovery, *IEEE Transactions on Software Engineering*, Vol. 33, No. 11, pp. 759–780 (2007).

[7] Kobayashi, K., Kamimura, M., Yano, K., Kato, K. and Matsuo, A.: SArF map: Visualizing software architecture from feature and layer viewpoints, *Proceedings of the 21st International Conference on Program Comprehension*, pp. 43–52 (2013).

[8] Mitchell, B. S. and Mancoridis, S.: On the automatic modularization of software systems using the bunch tool, *IEEE Transactions on Software Engineering*, Vol. 32, No. 3, pp. 193–208 (2006).

[9] Mancoridis, S. and Mitchell, B.: Bunch: A clustering tool for the recovery and maintenance of software system structures, *Proceedings of the 7th International Conference on Software Maintenance*, pp. 50–59 (1999).

[10] 渥美紀寿：プログラム解析技術を用いたソースコード再利用支援に関する研究，博士論文，名古屋大学 (2007).

[11] Müller, H. A., Orgun, M. A., Tilley, S. R. and Uhl, J. S.: A Reverse Engineering Approach To Subsystem Structure Identification, *Journal of Software Maintenance: Research and Practice*, Vol. 5, No. 4, pp. 181–204 (1993).

[12] Kobayashi, K., Kamimura, M., Kato, K., Yano, K. and Matsuo, A.: Feature-gathering dependency-based soft-

© 2015 Information Processing Society of Japan

ware clustering using Dedication and Modularity, *Proceedings of the 28th International Conference on Software Maintenance*, pp. 462–471 (2012).

[13] Anquetil, N. and Lethbridge, T.: Comparative Study of Clustering Algorithms and Abstract Representations for Software Remodularization, *IEE Proceedings-Software*, Vol. 150, No. 3, pp. 185–201 (2003).

[14] Andritsos, P. and Tzerpos, V.: Information-theoretic software clustering, *IEEE Transactions on Software Engineering*, Vol. 31, No. 2, pp. 150–165 (2005).

[15] Anquetil, N. and Lethbridge, T.: Recovering software architecture from the names of source files, *Journal of Software Maintenance*, Vol. 11, No. 3, pp. 201–221 (1999).

[16] Scanniello, G., D'Amico, A., D'Amico, C. and D'Amico, T.: Using the Kleinberg Algorithm and Vector Space Model for Software System Clustering, *Proceedings of the 18th International Conference on Program Comprehension*, pp. 180–189 (2010).

[17] Tzerpos, V. and Holt, R.: ACDC: An algorithm for comprehension-driven clustering, *Proceedings of the 7th Working Conference on Reverse Engineering*, pp. 258–267 (2000).

[18] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder : A Multi-linguistic Token-Based Code Clone Detection System for Large Scale Source Code, *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670 (2002).

[19] McMillan, C., Grechanik, M. and Poshyvanyk, D.: Detecting similar software applications, *Proceedings of the 34th International Conference on Software Engineering*, pp. 364–374 (2012).

[20] Cesare, S. and Xiang, Y.: Malware Variant Detection Using Similarity Search over Sets of Control Flow Graphs, *Proceedings of the 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 181–189 (2011).

[21] Dean, J., Grove, D. and Chambers, C.: Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis, *Proceedings of the 9th European Conference on Object-Oriented Programming*, pp. 77–101 (1995).

[22] Newman, M. E. J.: Fast algorithm for detecting community structure in networks, *Physical Review E*, Vol. 69, No. 6, pp. 1–5 (2004).

[23] Ural, E., Umut, T. and Feza, B.: Object Oriented Software Clustering Based on Community Structure, *Proceedings of the 18th Asia-Pacific Software Engineering Conference*, pp. 315–321 (2011).

[24] Clauset, A., Newman, M. E. J. and Moore, C.: Finding community structure in very large networks, *Physical Review E*, Vol. 70, No. 6 (2004).

[25] Brandes, U., Delling, D., Gaertler, M., Robert, G., Hoefer, M., Nikoloski, Z. and Wagner, D.: On Modularity Clustering, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20, No. 2, pp. 172–188 (2008).

[26] Wu, J., Hassan, A. and Holt, R.: Comparison of clustering algorithms in the context of software evolution, *Proceedings of the 21st International Conference on Software Maintenance*, pp. 525–535 (2005).

[27] Wen, Z. and Tzerpos, V.: An effectiveness measure for software clustering algorithms, *Proceedings of 12th International Workshop on Program Comprehension*, pp. 194–203 (2004).

[28] Tzerpos, V. and Holt, R.: MoJo: a distance metric for software clusterings, *Proceedings of the 6th Working Conference on Reverse Engineering*, pp. 187–193 (1999).

© 2015 Information Processing Society of Japan